Service-Based P2P Overlay Network for Collaborative Problem Solving

Sanjay Goel[1], Michael Sobolewski[2], Shashishekara Talya[3]

Abstract

This paper describes a service-based P2P overlay network architecture to support a collaborative environment for solving complex business processes over the network. In the proposed architecture, autonomic service providers corresponding to various activities that occur in the processes reside on the overlay network and are discovered dynamically during the execution of the process. To consummate a specific process, a set of services that map into the business process are federated together and executed in a choreographed sequence. All services have standardized interfaces and this allows any service to be seamlessly replaced with another service without affecting the performance of the federation. The paper presents two cases of application of this architecture, namely, business-to-business collaboration in an engineering environment (General Electric) and multiparty financial transactions (mortgage).

1. Introduction

Design of complex engineering systems, such as aircraft engines, requires simulation tools that are computationally expensive. Such systems are composed of hundreds of mutually interacting components that should ideally be designed concurrently. Concurrent design of all components, however, is computationally and logistically infeasible. So, the design is usually decomposed into smaller tasks such that components and subsystems are designed independently. The interactions between the components are modeled by inserting boundary conditions of the interacting components into each other. The design of the system is done at multiple levels of granularity, starting with the

[1] Sanjay Goel, Business Administration 310b, State University of New York, Albany 12222
[2] Texas Tech. University, Lubolk, Texas
[3] General Electric Global Research Center, One Research Circle, Niskayuna, NY 12301

system design in which all the components are analyzed together using simple analysis tools that use empirical models, and ends with detailed design of each component, which requires precise analysis codes that are computationally expensive. At each level of granularity, the boundary conditions for the next level of granularity are computed. As the design progressively gets decomposed into increasingly smaller tasks, the process map transforms into a complex network. This network of tasks represents the design process that needs to be executed in order to complete the design. The design process is dynamic and evolves over time as new advances are made in technology and analysis techniques.

In this paper, we consider the engineering analysis of an aircraft engine that requires a large number of analysis codes from different disciplines such as Structural Analysis, Heat Transfer, and Fluid Mechanics to analyze different components of the engine. In the past, the design process was executed manually wherein designers would design a component and exchange the coupling information via data files with other designers working on different components and subsystems. Such a manual process is not only an inhibitor to increased productivity, but is also very prone to errors. In addition, the accuracy of the results is limited due to the fact that the process is executed manually and is highly dependent on each individual engineer. Furthermore, since the design was independently optimized for a single discipline with constraints imposed from other disciplines, a sub-optimal design was obtained. For instance, if the design was optimized for turbine fuel efficiency (fluid mechanics) it may not be optimum for weight (structural analysis) or life (heat transfer). Several attempts have been made in the past to create a collaborative design environment through the automation of tasks in the design process, [20] [33] [49] and consequently reduce the design cycle time and improve performance. The design process automation requires automation and integration of each individual task into the process map that incorporates the loops, forks, and transitions of the process. The design of each component usually does not have a closed form solution; an individual task involves repeated execution of an analysis code in which alternate configurations of the design are evaluated. Automation of these tasks requires incorporation of optimization models that drive the design to maximize (or minimize) a set of objectives subject to the constraints of the problem. The entire design process is then simulated by executing the codes in sequences provided in the process map and cascading outputs from one analysis code to the next. These automation efforts, though robust at the individual task level where the process is fairly standardized [21], are very brittle at the process level. Brittleness here refers to inflexibility and inability to adapt to changes. The reason for brittleness at the process level is that analysis codes, as well as the process, change and render the couplings between tasks ineffective thereby breaking the process map. The maintenance burden of fixing broken links is onerous and makes the automated processes inefficient in terms of the maintenance cost versus productivity gains.

In this paper, we describe a network architecture that supports an adaptive collaborative environment for engineering design through use of modular services with standardized interfaces. Sanchez and Mahoney [42] have investigated the benefits of modularity in product and organizational design. They examine the benefits of standardized interfaces between components in a product design on coordination of product development processes. They assert that using modularity enables effective coordination of processes without the tight coupling of organizational structures. Shani et al. [44] state that technology is not enough to drive change in the organization. Rather, a *sociotechnical* approach is required. In their investigation, they use case-based reasoning to investigate the reasons U.S. companies lag behind their counterparts in Japan and Europe in the implementation of computer integrated manufacturing and robotics. The proposed architecture supports the use of modular process components (services) to support automation of flexible business processes such as engineering analysis. The services have fixed interfaces and are discovered in real time to create an impromptu federation of services that map to the current business process. This provides the users a flexibility to rapidly change business processes and also provides resilience to failures of networks and systems. This architecture also fits well with the sociotechnical environment by aligning well with the designer habits and management mandates.

The architecture is constituted of well-known autonomic services that represent specific engineering tasks on the service grid. Such architectures fall under the general category of service-oriented architectures. In such architectures the service grid is usually an overlay network of service providers above the underlying network of computing devices. The services have standardized types (interfaces) allowing them to be located by searching for complementary attributes associated with types. The standardization of interfaces also ensures that one service can be seamlessly replaced by another service without requiring reconfiguration of the network. The services have standardized input and output interfaces; strong coupling by cascading data from one analysis to the next is unnecessary. When a new engine configuration needs to be analyzed, a process map is generated, services that map to the process are discovered, and a federation of trusted services is created. Once the process is complete, the services disperse and join other federations to perform other activities. Changes to any individual service on the grid are usually transparent to the process map. In this architecture, services can enter and leave the grid at will. Resilience in the service grid is achieved due to the redundancy in the overlay network whereby several services can exist for the same task. The standardized interfaces allow seamless substitution of one service with another. Such standardization is provided by defining interface classes in *Jini* and by using XML-based data exchange in a heterogeneous InterGrid. Using an InterGrid allows partner access and sharing across disparate organizations and allowing them to share heterogeneous databases and collaborate on design initiatives.

The paper presents a new architecture for managing business processes and hypothesizes that such modular software components (services) on the network can facilitate modeling of complex business processes and provide greater flexibility in managing process changes and improved resilience to system failures. This architecture is demonstrated using two case studies. The first case study demonstrates the application of the architecture at General Electric for engineering design and the second case study demonstrates its application to the banking problem of mortgage transaction. Even though the domains are completely different, they are both business processes, they share common attributes, and some common conclusions can be drawn.

Case-based research has been established as a proven tool for MIS research. Sarkar and Lee [43]have used case-based research to investigate business process re-engineering. They claim that a balance between socio-centric and techno-centric approaches is required for effective business process reengineering. Lee [36] argues that the propose methodology satisfies the stands of the natural science model of scientific research. He defines four different problems that are encountered in holding case-based research in MIS to standards of natural science. That is, *making controlled observations, making controlled deductions,* and *“allowing for replicability, allowing for generalizability*. He states that for rigorous case analysis in MIS research, the theory must be: 1) falsifiable, 2) logical consistent, 3) more predictive than other theories and 4) not falsified by the tests it experiences. Eisenhardt [14] describes the process of inducting theory using case studies. She presents a roadmap for building theories from case-study research and positions theory building from case studies into the context of social science research. The current work has two contributions: 1) development of a novel architecture for supporting business processes on the network, and 2) investigation of the suitability of the architecture in business process and identifying factors that makes such architecture suitable via use of case studies.

The rest of the paper is organized as follows: section 2 discusses the relevant literature, section 3 describes the architecture of the system, section 4 discusses some applications based on the architecture, and section 5 discusses observations from the case studies. Section 6 provides concluding remarks.

2. Literature Review

Design of self-configuring overlay networks has been investigated under the purview of P2P systems, notably file sharing applications, distributed data storage and mobile ad hoc networks. At the same time, the problem of distributing large computations on a network has been investigated under the purview of distributed (grid) computing systems [15][16]. The current work straddles the fields of grid computing as well as P2P architectures. The extant literature relevant for the current work is thus organized into two separate streams: P2P architectures and grid computing systems.

P2P architectures have received a lot of visibility in context of computer network communication. P2P communication is an abstract concept that refers to direct interaction between peers while performing specific tasks rather than using an intermediary to facilitate all communication. In context of computer networks, P2P networks facilitate direct interaction between nodes at the edges of the network and the resource and computation burden are shared by all the peers rather than being controlled by a single to a few nodes that act as conduits for all communication and controls access to resources.

Most of the new P2P architectures that have emerged are overlay networks on top of the physical network. Overlay networks are virtual networks that do not correspond with the physical network topology. This implies that a physical network node may contain multiple virtual peers and the network traffic within a virtual peer group may transcend multiple sub-networks. P2P networks can operate at multiple levels of the network stack, that is, network level or application level. The Internet itself is a P2P architecture that operates at the network level as the network nodes on the Internet communicate directly with each other rather than through use of an intermediary. File sharing and distributed data storage systems such as Napster, Gnutella, and Freenet, OceanStore and Farsite are based on the application layer. These protocols communicate directly with the transport layer protocols (TCP/UDP). They usually provide APIs and toolkits that allow development of P2P applications on top of these protocols. Several toolkits such as JXTA, *Jini*, and Web Services either use existing application layer protocols or define their protocols for facilitating communication among peers.

P2P architectures became mainstream due to the popularity of the P2P file sharing applications such as Napster, Gnutella, Morpheus, Freenet, and Kazaa. These applications allow peers on the network to exchange music and movie files with other peers on the network. All of them support dynamic discovery of resources. However they differ in the techniques used for searching and locating resources. There are two distinct steps in the process of resource sharing: location of a resource and transfer of content. In all of these systems, the exchange of content occurs directly between peers. However, resource location can be centralized, distributed, or hierarchical in nature.

Napster [35] uses a centralized directory to store and locate resources (e.g. files) on the network. In this model, the peers of the community connect to a central database where they publish information about the content that they wish to share with the other peers. A user queries the central database to obtain the IP-address of the best node where the resource is located and then communicates directly with the node to obtain the resource.

Gnutella [10][18] uses a distributed search protocol that allows peers to conduct filename searches on the Gnutella overlay network without the need of any intermediate index server. It uses flooding for distributing the query among the nodes of the network. The searches are made over the Gnutella network and the files are downloaded offline. Gnutella is a resilient network where any client can enter or leave the network without significantly affecting performance. FastTrack [40]is a P2P library that drives some of the currently most successful P2P search engines like KaZaA, Morpheus, Grokster. They use a proprietary, encrypted protocol in which the nodes contain data, hierarchical super nodes provide search capabilities by forwarding search requests to other super peers, and a central controller manages all the super peers. The architecture of the registries has a strong implication in the scalability resilience and efficiency of search among the peers. In addition, there has been some work in content-based routing of data where storage and search has greater efficiency than simply flooding the network. Several systems based on content-based routing such as CAN, Chord, Tapestry and Pastry have emerged in the literature. Some work has been also done in the area of ad hoc networks where network configuration is constantly evolving as peers enter and leave the network. Most of this work, however, has focused on routing [19][53][32] and security [54][30][25][12][6] of such networks.

For the application suite considered for the current architecture, the execution time for the analysis codes far exceeds the time required to search for services, thus search time for services is less critical. However, issues such as coordination of services and process mapping and security are important to facilitate the smooth execution of the process. This architecture also requires richer development environments that will allow development of infrastructure for task coordination and monitoring of services in addition to search and discovery. Several toolkits such as JXTA, *Jini,* and Web Services can be considered for developing this architecture.

JXTA [23] is a modular network-programming platform for building distributed services and applications. It is a 3-layer architecture containing 6 XML-based protocols (discovery, membership, routing, etc.) and abstractions for several components (peer groups, pipes, advertisements, etc.). Since JXTA provides protocols instead of APIs, it can be implemented in any language on any operating system. JXTA defines low-level protocols that are more suitable for facilitating lightweight communication among devices rather than for creating a service-based network with a large footprint and very large network traffic.

Jini [34][39] is a protocol independent distributed computing architecture that can interact with distributed objects. It is object-oriented in nature as evidenced by its calling on remote objects identified by their interfaces. The protocol used to carry out a remote-call is protocol-independent. It defines the discovery and Join protocol to allow for spontaneous networking of services, registering of remote objects, and dynamic finding of these on the network through their interface and associate properties. Within the *Jini* protocol, any protocol, such as protocols that can carry out remote calls: JRMP (RMIJava Remote Method, JERI (Jini Extensible Remote Invocation), ILOPCORBA, SSL, HHTP, HTTPS, DCOM or other proprietary protocols can be used. *Jini* uses Java language that makes it platform independent. It also provides a lookup service through which the services advertise their availability to other peers on the network and the requestors can locate the services. The implementation of *Jini* that is currently available uses the JRMP protocol. *Jini* allows direct object-to-object communication through use of proxies.

Web Services is another P2P application for definition and interaction of services on the network. Web Services provide standardization by defining XML-based standards that allow objects on the network to exchange messages and to locate other services on the network. Three primary standards are defined: Simple Object Access Protocol (SOAP) [5], Universal Description and Integration (UDDI) [4], and Web Services Description Language (WSDL) [9]. SOAP defines standards for communication among objects, independent of the language of implementation of the objects. UDDI describes a registry that allows services to be advertised and discovered. UDDI has a capability of dynamic discovery of web services but in practice the web services are located off line and then incorporated into the business process. WSDL allows for a standard description of the web services allowing programs to automatically extract information about the service.

Web services provide a standard method of enabling communication between applications' middle tiers over the network. When packaged as a web service or as a set of web services, the application provides a reusable interface that can communicate with other properly configured applications. This means that applications can share processes rather than only content without the need for customized one-to-one solutions. A web service encapsulates a task. When an application passes data or instructions to it, the service processes that information and if required, returns something to the application. In addition, since they are written in XML they allow communication between disparate applications written in different languages. Web services are self-describing, meaning that they are accompanied by information explaining what they do and how other applications can access and use them. These descriptions are typically written in WSDL. Web Services are discoverable i.e. they can be located via use of registries using the UDDI protocol. Web Services are suitable for computationally light processes on very large networks such as the Internet, where real-time discovery by broadcasts on the entire network across multiple local area networks would not be feasible. The object-oriented approach allowing locate remote objects by interfaces and associated properties and call on live ready to use object (proxy) is more suitable for computationally intense problems on local area networks where the burden of wrapping data in XML is unnecessary.

Several XML based standards are emerging to support the process modeling capabilities in Web Services. Web Services Choreography Interface (WSCI) provides a way to describe the behavior of a Web Service as an interface or an API and allows the interfaces to interoperate. Primary focus of WSCI is tight coupling between APIs supporting different web services. BPML provides a means of modeling business processes and includes specifications for transactions and compensating transactions, dataflow, messages and scheduled events, business rules, security roles, and exceptions. It supports secure transactions (synchronous and asynchronous) and provides project management capabilities. Business Process Execution language for Web Services (BPEL4WS) allows creation of complex processes by creating and wiring together different activities that can, for example, perform Web services invocations, manipulate data, throw faults, or terminate a process. These activities may be nested within structured activities that define how they may be run, such as in sequence, in parallel, or depending on certain conditions. BPEL4WS seems to have more momentum than the other XML-based specifications such as the BPML and WSCI. These specifications are still emerging and when available as mature APIs would be very useful for deploying applications over the InterGrid.

P2P communication needs to be clearly distinguished from the concept of distributed computing that involves breaking down a computationally large task into several subtasks that are distributed over several nodes of the network. The problem of coordinating distributing computation across multiple network nodes has also been investigated under the rubric of parallel computing[4] and meta-computing (grid) systems. Grid computing organizes computational and data resources distributed across a network to make computationally intensive problems feasible to solve. Most of the distributed computing systems are based on centralized coordination, however, some distributed computing systems based on P2P architecture are beginning to emerge. One notable application is Seti@home, [2] which shares processing load across distributed nodes. The application divides processing into small chunks and distributes them to other nodes. The application reassembles the results, which contribute to the overall solution. The application builds resilience by giving multiple nodes the same processing task and eliminating specious or incongruous results. I-Way [11] is another project that demonstrates the feasibility of

[4] Traditionally, researchers and practitioners have called distributed resource sharing parallel computing; however, since the mid-nineties, grid computing is more commonly used, especially as it relates to high performance distributed computing.

sharing distributed resources. In the I-Way project, multiple super computers process multiple applications and communicate over high-bandwidth ATM networks reducing execution time for complex analysis.

Grid computing poses a large number of challenges at the network level, including scheduling, coordination of activities, access control, load balancing, fault tolerance, and integration of heterogeneous systems [29]. Researchers are just beginning to explore these issues as grid computing becomes more prominent. Interestingly, grid-computing applications employ CS architectures. Generally, a central scheduler manages the distribution of processing to network resources and aggregates the processing results. These applications assume a tightly coupled network topology, ignoring the changing topology of a network. Load sharing and job scheduling schemes have been studied extensively with formal performance evaluation models [26][1]. Powerful grid computing toolkits, such as Globus [15], Legion [38], Simgrid [7] and Globe [52], which provide basic capabilities and interfaces for communication, resource location, scheduling, authentication, and security and primarily use a CS architecture. Simpler systems, such as GradSolve [51], Ninf [49], and NetSolve [8], which are based on remote procedure calls (RPCs), also use C-S architecture.

The current architecture uses concepts from both the grid computing and P2P architectures wherein a service grid as a collection of service providers can be dynamically partitioned into federations that can be dynamically mapped to executing processes [45][46]. In the architecture, each service provider is autonomous and the services interact with each other on a peer-to-peer basis and federate together to map into business processes. Each service can itself be composed of other services and forks with loops can be generated between services to represent iterations of analysis codes.

3. Architecture

The P2P service-oriented framework being developed in this work targets complex business and engineering transactions. A transaction is composed of a sequence of activities with specific precedence relationships. The grid contains service providers that offer one or more services to other peers on the overlay network. Service providers do not have mutual associations prior to the transaction; they come together (federate) dynamically for a specific transaction. Each provider in the federation performs its services in a choreographed sequence. Once the transaction is complete, the federation dissolves and the providers disperse and seek other transactions to join. The architecture is service centric in which a service is defined as an independent self-sustaining entity performing a specific network activity. Each service is defined by a well-known public interface. A service provider that plans to offer a service implements its interface or multiple interfaces (services) to be eligible for participating in federations.

The same provider can provide multiple services in the same federation and different providers can provide the same service in different federations. The service grid is dynamic in which new services can enter the overlay network and existing services can leave the network at any instance. The service-based architecture is resilient, self-healing, and self-managing. The key to the resilience is the transparency of search and seamless substitution of one service with another. The architecture allows services to share data by using specialized data services or a shared data repository (distributed file store). The architecture also allows asynchronous execution of activities such that an activity can wait for a service to be available.

The architecture uses *Jini* network technology [27][28][13] and *JavaSpaces* technology [17][24] for implementing the service-based overlay network described above. However, the proposed service-oriented architecture is abstract and can be implemented using any distributed network technology that provides support for dynamic discovery of resources and a rich software development environment. The discovery and lookup protocols that *Jini* supports, allow services to be dynamically located and federated during process execution. The *Jini* infrastructure can support several lookup services on the network simultaneously. Jini also supports a concept of leasing by virtue of which each resource is granted a temporary membership of the overlay network. Whenever, a service joins the grid its availability is announced on the network and a lookup service picks up the announcement, registers the service, stores a proxy for the service, and provides a fixed-time lease to the service provider to advertise the service. If the lease is not renewed at its expiry period, the proxy is ejected from the lookup service. The self-healing ability occurs by virtue of the leasing concept wherein disabled services are unable to extend their lease and are automatically eliminated from the overlay network when the lease ends since they are unable to renew their lease. When a service requestor is searching for a service it first finds lookup services (service registries), what is called *discovery*. Then the requestor based on its requirements searches for a relevant service and obtains the service proxy for the service from the registry. The service requestor may find several service proxies for the same requirements from different registries and it selects the best service proxy based on its own criteria. In the case where there is no service available to perform a particular task, the request can be aborted and sent back to the requestor or the request may be held for a specified period of time in the registrar after which it is aborted.

The overlay network consists of a set of service providers where each service provider is defined as an independent self-sustaining entity performing a specific network activity called *exertion*. Two types of exertions are considered: an elementary exertion – analogues to network operation – is called *task,* and a compound one – analogous to network program - is called *job*. A job is a recursive structure expressed in terms of exertions that defines a transaction map. Each service is defined by a public interface. The service grid (overlay network of service providers) is dynamic in which new services can enter the grid and existing services can leave the network at any instance. Services advertise themselves and can be found and selected based on the type (interface) and other attributes that they exhibit. By its type and optional attributes (e.g., provider name), the network object can be dynamically found on the grid without requiring a host name and a port. The architecture defines all decentralized distributed components in the system to be equal [46]. In the overlay network, all peers are network objects of the same type. All peers implement the common *Servicer* interface and their equality is defined as being service providers or *servicers* (see Figure 1). A *service* is an act of requesting a *service(Exertion)* operation from a service provider. As the result of requested service a processed exertion is returned to a requestor. Other custom interfaces other than *Servicer* that the services implement stay static, however the implementations might change, as they are specific to particular service providers. What custom interface and its particular method is used to process an exertion, is defined within the exertion itself. Also, data to be processed by the service provider is encapsulated in the exertion along with the interface and its method applied to the exertion's data.

A UML-diagram showing the framework of the system developed is illustrated in Figure 1. The core of the architecture consists of service providers and service brokers interacting with lookup registries, a catalog of services, and exertion shared space. In general, a service provider executes a task (elementary exertion), and a service broker executes a job (compound exertion or a nested transaction). While executing a job, the service broker coordinates exertion execution within the nested transaction. It interprets the transaction map supplied by the service requestor and completes the nested exertions accordingly as presented in the transaction map.

At the start of the transaction, the service broker reads all the exertions in the transaction and executes those exertions, which have no precedence relationships. At each step, it executes the services for which all the precedence relationships have been satisfied (services complete). Whenever it gets a notification of a service being completed, it evaluates the remaining unfinished activities and invokes one or more exertions based on their precedence relationships.

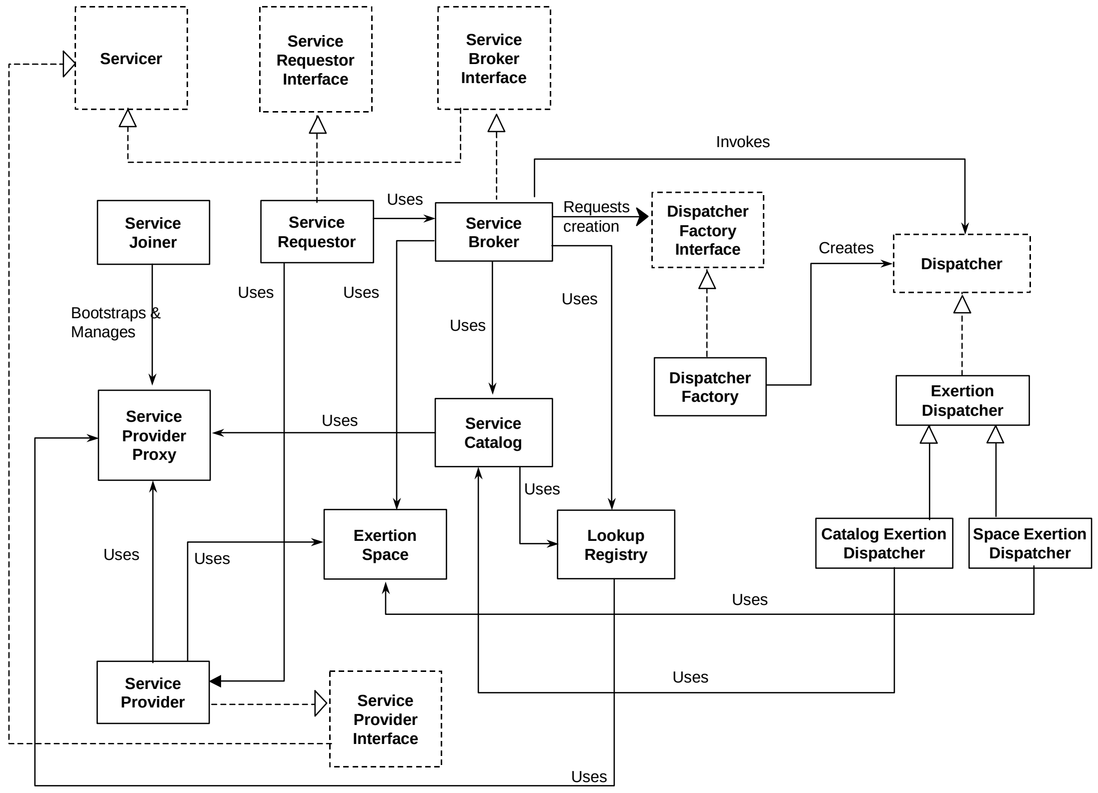

Figure 1: The simplified UML-Diagram for the service-based framework to support nested transactions.

The service broker, by using an appropriate exertion dispatcher, can directly access the service provider through a service catalog and select a provider or drop the exertion into Exertion Space for the first available provider to process the request. While the service broker is servicing a job, a nested job within the job being currently serviced can be executed locally, or can be dropped into the Exertion Space, or passed on directly to another service broker. Another available service broker can then federate and collaborate in the job execution by executing the nested job, and so on. Thus not only can the service providers federate to execute a job for a particular service broker, but the service brokers can also federate along with the service providers. The federated brokers with the originating broker execute the nested jobs while the regular service providers execute all the tasks within all jobs including the originating one. A service broker uses a factory design pattern to request a relevant exertion dispatcher that matches the control structure of the executed job.

Two main types of exertion dispatchers are used: a catalog exertion dispatcher, and a space exertion dispatcher. The catalog exertion dispatcher finds needed service providers using the service catalog. The space exertion dispatcher drops exertion into the exertion space to be picked up by matching available service providers. When the exertion is picked up from the space and is executed by a matching provider, the provider returns it into the space and the space exertion dispatcher gets it back from the space for the service broker. The service grid also defines a common service provider interface (*Provider* that extends the top level interface *Servicer*) and a set of utilities to create and register providers with the grid as service peers. A Service Joiner is used for bootstrapping service providers and for maintaining leases on registered proxies. The grid service brokers can also use dynamic provisioning based on Rio technology (Rio Project) for deploying and maintaining required federations.

Figure 2 shows the different ways in which a provider (the service broker or service provider) can submit requests to the providers. For a direct connection to the service provider, the provider can either use discovery to find a lookup service or use a Service Catalog provider for selecting a service. The lookup service caches all the proxies for services that have registered with it for a particular group(s) of services. The Catalog provider is a service-grid cache that periodically polls all relevant lookup services and maintains a cache of all the proxies that are registered with the lookup services for a particular group or groups of services. Thus, multiple service catalogs may be used for different logical overlay sub-networks. The provider has to discover lookup services each time it needs to use them where as it finds one of required catalogs only once when it (provider) is instantiated and then the Catalog continues service discovery for the provider. In case the provider finds an available service using a lookup registry or the Catalog, a proxy for the service is downloaded on to the provider who invokes the service by calling *service (Exertion)*. Alternately, the provider submits the service request to an Exertion Space that holds the request and waits for a matching service provider to accept the exertion. This is essential so that the transaction does not have to abort due to non-availability of a service. This also helps in better load balancing of the services since available providers will act at their own pace to process the exertions in the space. A notification management framework, [35] based on a notification provider, allows federated providers to notify the service requestor on their collaborative actions. Additionally, the File Store provider [48] allows federated providers to share exertion input as well as output data is a uniform service-oriented way.

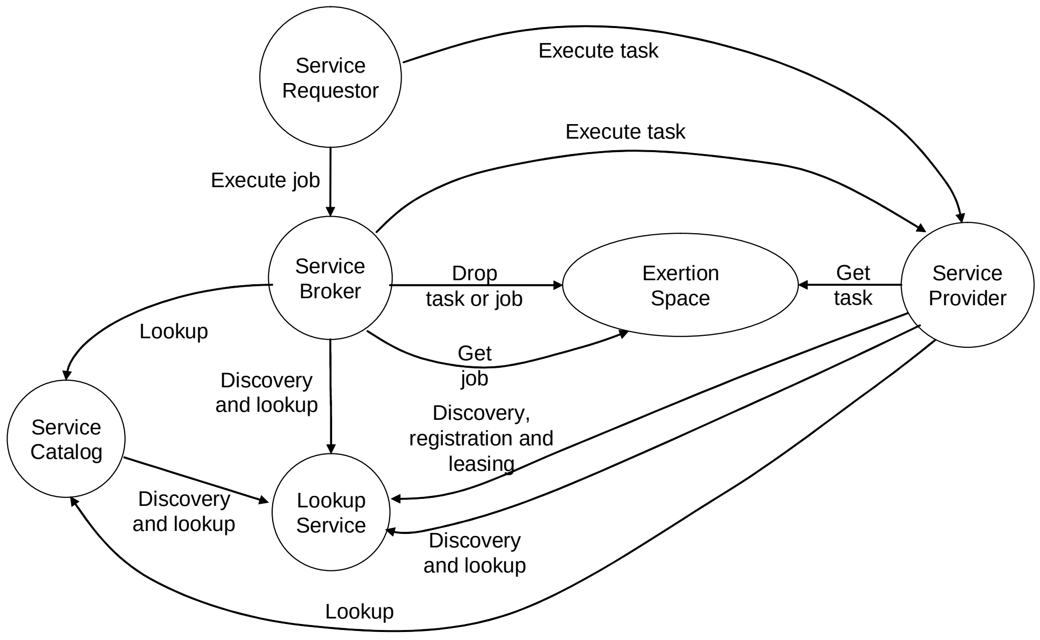

Figure 2: Diagram showing operation of the Service Broker and Service Provider

Transaction Maps

A *context model* is the exertion's data structure and is based on the percept calculus knowledge representation scheme [47]. It forms the essential structure of the data being processed as specified by the exertion's interface and operation. A context model of the exertion is represented as a tree-like structure of context nodes [54]. It is represented by the *ServiceContext* interface or alternatively can be represented in XML when used across heterogeneous (non Java-based) programming environments. The actual data resides on a *data node*. The *context* denotes an application domain namespace, and a context model is its context with data nodes as leaf nodes appended to its context paths. A context path is a name for a data in its leaf node. The leaf node might contain any object and in particular an object that represents a file, for example a URL. A special container object called *ServiceNode* acts as a wrapper that holds a reference to a remote document object available for example from the File Store provider [48].

As mentioned earlier, any service is identified by the interface it implements. All services implement the common interface called *Servicer*, other than service-specific interfaces. Hence, all providers are peers that can communicate with one another via the *Servicer* interface, requesting a *Exertion service(Exertion)* operation from any service provider. If a service provider cannot provide a service directly, (i.e. it does not implement a requested interface) then it is responsible to find an appropriate provider and forward the request to it. In that way, relaying providers also participate implicitly in federated collaboration.

In the service grid environment, two types of basic exertions are defined: *tasks* and *jobs*. A task is the *atomic exertion* that is defined by its *context model* (data), and by its *method* (operation). An exertion method defines a service provider (grid object) to be bound to in runtime. This network object provides the business logic to be applied to the exertion context model as specified in the exertion's method.

A method is primarily defined by a *provider type* (interface) and *selector* (operation name) in the provider's interface. Optionally, additional attributes might be associated with the method, for example a provider's name or provider's identifier. The information included in the exertion method allows the service-oriented program to bind the exertion to the grid provider and process the exertion's context by one of its operations, which is defined by its published domain specific interface. This type of service provider is called a *method provider*. Another type of service provider is a *context provider* that provides shared data to the grid via the distributed observer-observable paradigm. Thus, both context and method providers represent grid data and operations, respectively, to be used in the service-oriented programs (job exertions). Currently, only method providers are supported.

Conventional programs are still based on writing hundreds of lines of codes. In service-oriented computing, once providers are deployed, jobs can be built using interactive tools or programmatically in terms of reused exertions. Thus, programs can be created in which each task is like a network operation and a nested job is like a network program. Both task and jobs in fact are actually remote references to activities in the grid. A web-based user agent for editing, executing, and monitoring jobs by point-and-click was developed in Java. In addition, a web-based user agent for service-oriented file sharing was developed to allow the users to view and maintain files and folders in an interactive way in remote locations. These two agents integrated together, add a new dimension to the interactive service-oriented programming approach.

A service broker is a specialized service provider that executes a job – a compound exertion in terms of tasks and other jobs. In contrast, a method provider executes a task by taking a task context model as argument of service request and applies a task method to its context data nodes. Then, the provider returns a task with its output context model that contains the result of provided service. The data that the method provider generates can be in the form of any value object or file that is stored in the grid File Store. Thus, a service broker executes a job by binding in runtime all nested jobs to service brokers (including itself) and all nested tasks to available method providers.

Jobs by defining *is-part-of* relationships between exertions define transaction maps or workflows. In Figure 3, jobs are indicated by dashed outlines, in particular the job *J0 = (T1, J1, J2, T7)* consists of task *T1*, job *J1*, job *J2* and task *T7*. In turn the job *J1 = (T2, T3)* consists of task *T2* and *T3*, and the job *J2 = (J4, J5, J6)* consists of three tasks: *T4, T5*, and T6. Here we have a modular structure of seven tasks and in terms of these tasks three jobs: *J0, J1*, and *J2*. On the other hand, arrowed lines indicate a workflow that defines sequential relationship, unidirectional aggregation, explicit connections, and inherent control strategy.

In contrast, tasks are created and stored independently of each other in the grid Data Store using a data persisting provider [55]. Then they are used to create in an object-oriented way, (is-part-of relationships) three jobs and are potentially being reused in other jobs. The same applies to our three jobs *J0, J1*, and *J2* that can be persisted and reused in other jobs. Jobs implicitly define workflow as illustrated in Figure 3, and also support reusability. In addition, control strategy in jobs is not inherent as in workflows. Each job has its own control strategy along with all component exertions defined by its local context model called *control context*. In summary, object-oriented jobs provide a much more flexible and modular structure than workflows with explicit sequential connections.

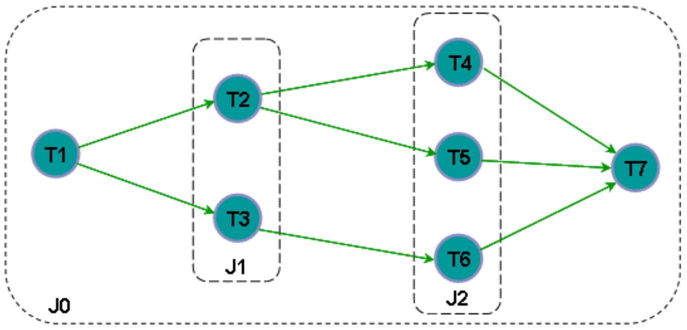

Figure 3: A job vs. workflow

4. Applications

A self-organizing network of autonomous services is very versatile and can be used for computing in several applications, namely, supply chain, engineering analysis, business transactions, etc. They provide a set of features (coordination of services, dynamic federations, self-healing, and self-managing) that change the paradigm and allow new applications beyond the traditional applications that can be developed using n-tier enterprise systems and peer-to-peer file sharing systems. Conceptually, the service-based network is applicable to any large network, including the Internet. The current implementation based on *Jini* has several advantages such as: efficient communication between the peers, and support of a rich programming environment. It uses multicast (or unicast) for the purpose of registration of services and discovery of services. The multicast is usually allowed only on local area networks and may not be supported across the firewalls.

This paper describes three applications using P2P systems that have been created using the proposed architecture presented in the paper. The first application is an engineering application of aerodynamic design of Turbomachinery components that was the initial motivation of this work. The second application is a B2B application for design of combustor nozzle across corporate firewalls using a secure pipe between two different local area networks application and the third application is a prototype financial application of mortgage that was developed using the proposed service-based architecture.

Turbine Aerodynamic Design

The turbine is the component of the engine that generates the power that uses hot compressed gases from the combustor to generate thrust on a shaft that either drives a fan in an aircraft engine or a generator in a power plant. It consists of several alternating rows of blades and vanes where the gases expand and drive the turbine. The aerodynamic design of a turbine involves determining the configuration of each stage so as to maximize the efficiency of the turbine. The design of a turbine is done in several steps as shown in Figure 4. It starts (Step 0) with the thermodynamic cycle analysis of the aircraft engine to determine the flow conditions at the interfaces between the different components (compressor, combustor, turbine etc.). Step 0 indicates the 0-D (0-Dimensional) fidelity of the analysis. The complete engine is modeled as a thermodynamic cycle and the performance of the engine is evaluated at different points in the flight regime of the aircraft engine. The different flight points, also called as cycle points can be ground idle, take-off, cruise, landing, etc. The turbine is usually designed for maximum or best performance at a particular cycle point, called the design cycle point and is required to meet the minimum performance criteria at all other cycle points, also called as off-design. Step 1 is the preliminary design in which a one-dimensional (1-D) analysis code based on empirical data is used to analyze all the turbine stages simultaneously in order to predict the pitchline performance. The next step involves a circumferentially averaged flow analysis in which Navier-Stokes equations are used for analyzing each blade row independently. There are several intermediate analysis codes that map the boundary conditions and flow parameters from the one-dimensional analysis code to each individual row of blades and vanes. At this step, a first cut at the airfoil definition is obtained. The subsequent step involves performing detailed 2-Dimensional (2-D) and 3-Dimensional (3-D) analysis of the different stages in the turbine using computational fluid dynamic (CFD) analysis codes in order to design the details of the geometric shape of the turbine blade, which include the blade cross-section and the stacking sequence. The goal at this step is to maximize the turbine efficiency while meeting the performance requirements laid out at Step 0 in the process. At the end of this step, the aerodynamic design of the turbine is complete and the following step indicates the mechanical design process. In addition to this, each step in the process involves optimizing the turbine configuration or geometry parameters to maximize the efficiency. In this paper, the details of the first 3 steps in the Turbine Aerodynamic process are described and the various implementation aspects are discussed.

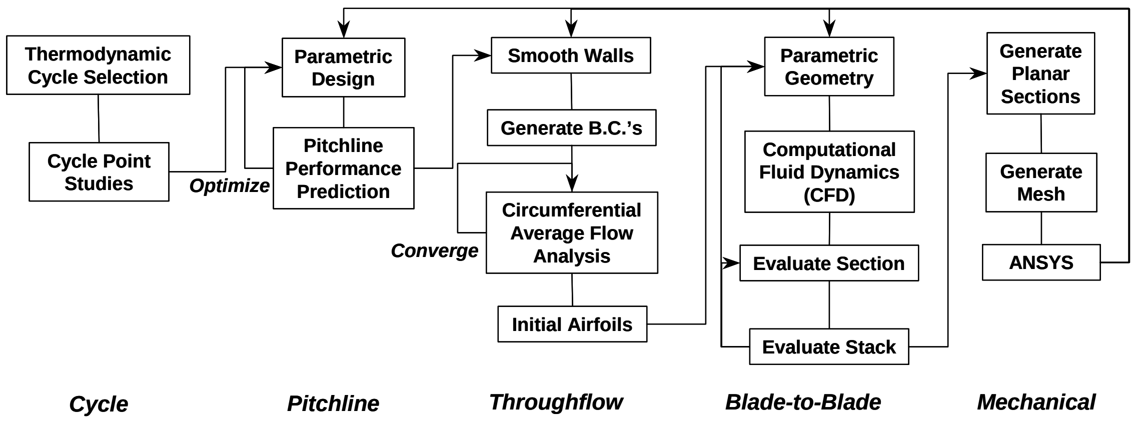

Figure 4: The aerodynamic design process

The analysis in each step is performed using in-house codes developed at General Electric (GE), written in the Fortran language in which the inputs and outputs for each analysis are data files in a pre-specified format. Each of the applications in the Turbine Aerodynamic design process is wrapped as a service and whenever it gets published to the network, it is available for service requestors to be able to execute the specific tasks in a particular job. In this specific example three service providers are defined, namely, Cycle Database Query Service, 1-Dimensional (1-D) Analysis Service and 2-Dimensional (2-D) axi-symmetric Analysis and Update service. The Cycle Database Query Service retrieves the cycle conditions for a specific cycle point from the engine database that stores the thermodynamic cycle information for different cycle conditions. The 1-D analysis service exposes several services a) evaluation of 1-D or pitchline turbine performance at a baseline cycle point by executing a 1-D analysis code, b) evaluation of 1-D performance given a baseline input and a new cycle point c) cycle performance evaluation for different cycle points. The 2-D axi-symmetric Analysis and

Update Service provider exposes two services: a) 2-D axi-symmetric analysis for a particular design point, b) Flow parameter specific iterative procedures to map the boundary conditions (inlet pressure, inlet temperature, mass flow rate, fuel flow etc.) and flow parameters (total pressures at different locations on the rotor/stator, cooling flow rates, etc.) from the results of the 1-D analysis code to flow parameters for individual rotor and stator in each stage.

All the above service providers are exposed on the network and get registered via one of the lookup services on the network. Each service provider is identified by a unique interface and a unique provider name and the service requestor can select individual service providers based on these attributes. In addition to this, each provider can perform more than one service and depending on the type and number of inputs specified, the provider can intelligently determine the specific service that is requested. Depending on the level of fidelity required in the analysis results, different jobs can now be defined to perform the different steps in the Turbine Aerodynamic design process. For example, a pitchline performance evaluation at a new cycle point would involve defining a job that consists of two tasks: a Cycle Database Query task that extracts the cycle conditions a the new cycle point using the corresponding service provider, and a 1-D Performance Evaluation task that evaluates the performance at the new cycle point using the new cycle conditions from the previous task and a baseline input. In addition, a dynamic data mapping exists between the Database Query task and the 1-D Performance Evaluation task to allow the passing of the output of the first task as the input to the second task. The two task definitions include the name and the interface of the service provider that can execute the particular task. The service providers for the above two tasks, namely the Cycle Database Query service and the 1-D Analysis service, are first registered on the network with a unique name and interface for a specific duration or lease. The job consisting of the above two tasks is submitted for execution. The tasks perform a lookup in the registry to search for specific service providers that can potentially execute the task. If the task finds the service that satisfies its requirements, the execution of the task is transferred to the specific service provider. In this case, the service Cycle Database Query receives the task to perform a database query to extract cycle points for a specific input. The results of the query are then passed back to the Database Query task. The dynamic mapping between the two tasks passes the results of the query to the second task. The second task (1-D Analysis) then looks for a service provider to perform its task. If the 1-D Performance Evaluation service provider is found in the lookup registry, then the second task is executed and the results of the execution are returned back to the task. In the specific implementation, a failure to find a service provider results in a failure of the task with the appropriate error message for the designer to realize that the service provider is not present or its lease has expired in the lookup registry.

The above job is extended further to be able to evaluate the pitchline performance at multiple cycle points. In this particular case, the same service provider is accessed multiple times via the lookup registry to perform the 1-D Performance Analysis. The input to the process is a data file containing the performance parameters at multiple cycle points. The cycle performance parameters are obtained by performing a cycle database query and the results of the query are used as input to this process. This multiple cycle point performance evaluation is a critical aspect of the turbine design as it provides the

Engineer with performance numbers at different off-design points (for e.g., at take-off, landing, etc.) in the aircraft flight cycle. Using the results of this analysis, the Engineer can create a Turbine Performance Map to study the turbine performance at different cycle points. A typical performance map for different cycle points is shown in Figure 5.

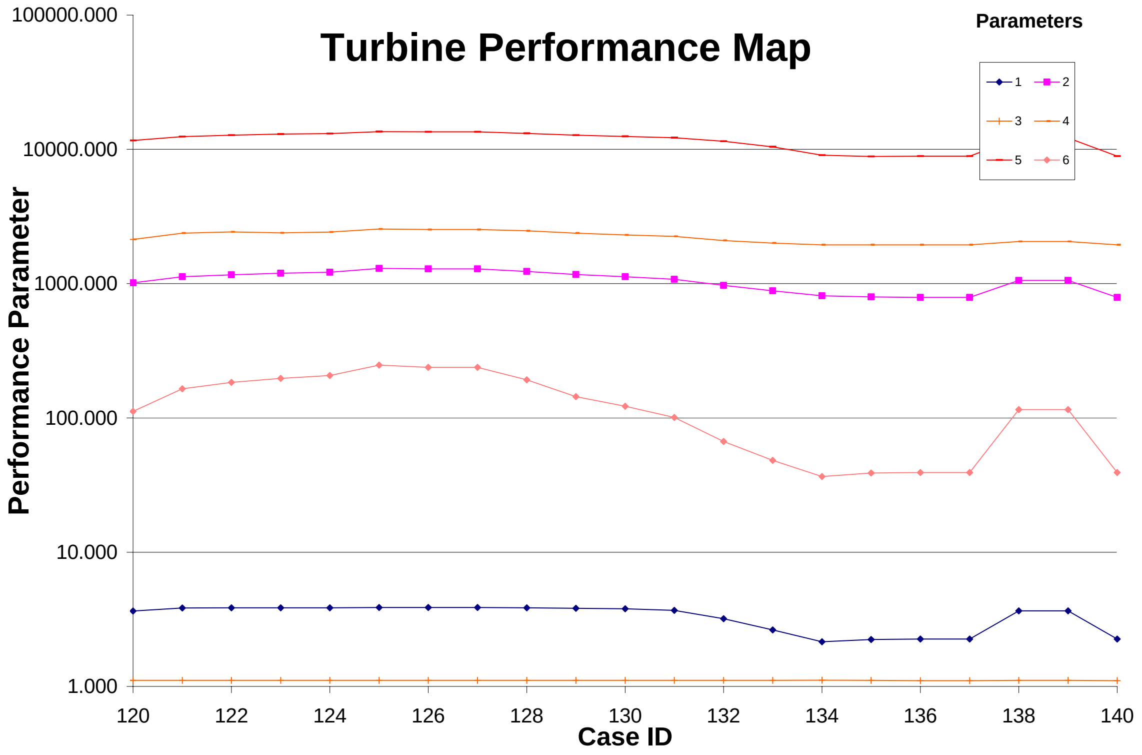

Figure 5: Turbine Performance Maps

Each of the several performance parameters is plotted against the cycle case ID. Actual names of performance parameters have been substituted by numbers to protect the proprietary information at General Electric. Compared to the traditional manual process of executing the 1-D Analysis code multiple times, this automated process has been demonstrated to result in a productivity savings of more than 50%.

At the next step in the design process, 2-D axi-symmetric analysis is performed to improve the fidelity of the above process. The axi-symmetric analysis is a bridge between the 1-D analysis and 2-D analysis and is usually a very cumbersome task requiring several iterations of the analysis with different parameter configurations. Introducing the axi-symmetric analysis in the job requires defining an additional task in the job defined above to perform the 2-D axi-symmetric analysis update task. The process flow is shown in Figure 6.

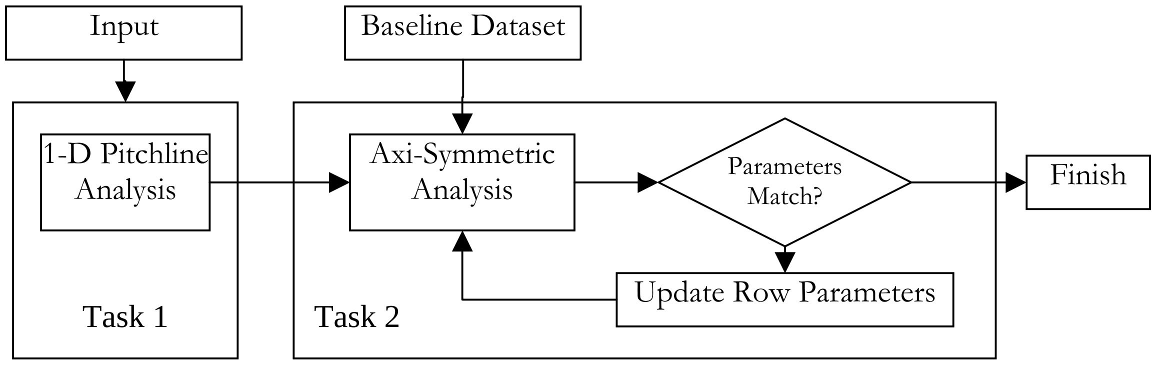

Figure 6: 2-D Axi-symmetric Parameter Update Process

To consummate this process the turbine designer needs to specify convergence limits for matching the flow parameters and boundary conditions of the 2-D analysis to the 1-dimensional analysis. Using the capability of the dynamic data mapping between different tasks, the data transfer from one task to the next is performed. In addition, each task uses the lookup registry to find the specific service provider to execute the task. Figure 7 shows the typical convergence history for one of the flow parameters. The traditional approach of performing the 2-D axi-symmetric update would require the designer to manually update each flow parameter incrementally till the desired target value is achieved. This manual process had to be done for each flow parameter and for a typical 2-stage conventional turbine there can be over 40 parameters. This would typically take 2-3 days to complete. By developing this integrated system to do the same analysis, the process time can be brought down to 3-4 hours. In addition to this, the automated process allows the designer to specify a tighter convergence criterion thus achieving a closer match between the 2-D and 1-D boundary conditions. This demonstrates the remarkable ease with which the design process can be incrementally built using the Lego-block approach wherein each additional “block” improves the fidelity of the process. Any changes in the process can be accommodated by changing the job description unlike the earlier systems, which required reconfiguration of the entire system by expert software developers. For example, the job described above can be easily extended to perform the 2-D axi-symmetric update for multiple cycle points.

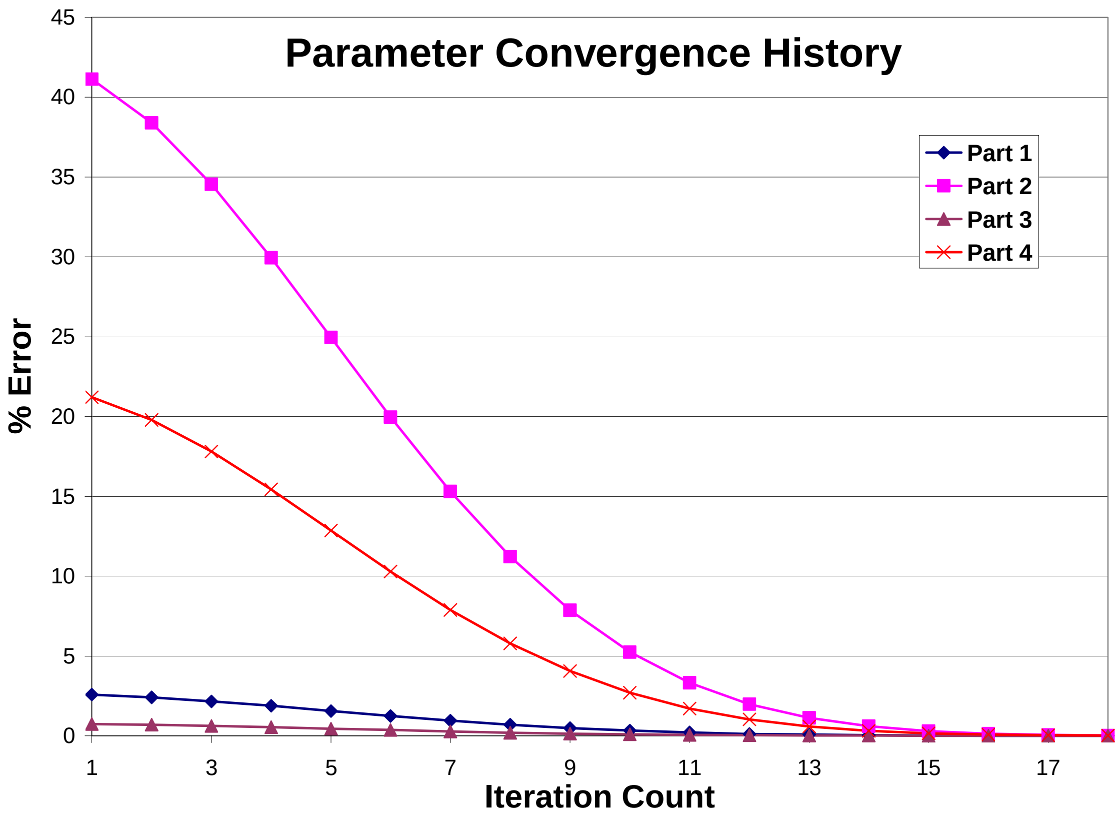

Figure 7: Flow Parameter Convergence History for different rotors and stators

Similarly, by providing additional services to the service grid, the job can be extended to Quasi- 3-D analysis and eventually 3-D analysis. Thus, a dynamic environment where designers with expertise in different specialty areas of design can maintain their own services on the network and users with design requirements, can access them dynamically. This approach provides the designers with the latest versions of the code and removes the potential problem of several obsolete versions of the analysis codes in different parts of the company. In addition to this, a super-user can define standard jobs or processes based on the latest design practices and make available to the designers for execution. This ensures that all designers follow the same process for a particular analysis or design thus allowing for standardization of processes. Such an approach also provides strong ownership to the developers and maintainers of the services. An additional advantage of this approach is improved load balancing whereby several users usually operating at a different point in the design process access the same service at different times.

Combustor Fuel Nozzle Design

The design of a Combustor Fuel Nozzle is complex multidisciplinary design problem involving disciplines such as aerodynamics, heat transfer, structural analysis, vibration analysis etc. In addition to this, it poses several organizational challenges. General Electric Aircraft Engines (GE) is responsible for the overall design of the Gas Turbine, including the Combustor, where as Parker Hannifin (Parker) is responsible for the design and manufacture of the Fuel Nozzle. For the design to be successful the fuel nozzle should fit into the combustor and its performance should match the rest of the combustor design. In most designs, the specs are constantly changing as the design is being optimized or some constraints in the design are violated. Manually executing this process requires a lot of patience and causes inordinate amounts of delay in the overall design cycle. Automating the process of the design across GE and Parker can provide a tremendous boost to both productivity as well as performance. In order to automate the design of the Combustor Fuel Nozzle, the service grid was extended to span GE and Parker such that the service grid performs seamlessly across business firewalls.

Figure 8 illustrates the complete B2B framework including the different P2P network framework. A P2P framework is installed at both GE and Parker and a secure communication between the two organizations is ensured using multiple levels of security. Communication between the two organizations is established with HTTPS (HTTP over SSL, a secure transport layer) to provide identity verification, confidentiality, and data integrity. The request goes to a server in a corporate DMZ, which is a buffer zone between the protected internal network and an external network. A DMZ server running an HTTPS server determines a perspective client's identity and authority to see if the individual has permission to execute services in the partner's network. If the access is granted, the job is launched and results are returned to clients. Clients can then use the returned reference to retrieve results via HTTPS. This HTTPS implementation, coupled with corporate firewall configurations, provides added security for business-to-business collaboration. All requests between the two organizations are routed via the DMZ servers that exist at both organizations. The DMZ servers are configured to listen to requests from the respective business proxy server. More detailed information on the different aspects of security in a B2B framework is discussed in the section on B2B Security below.

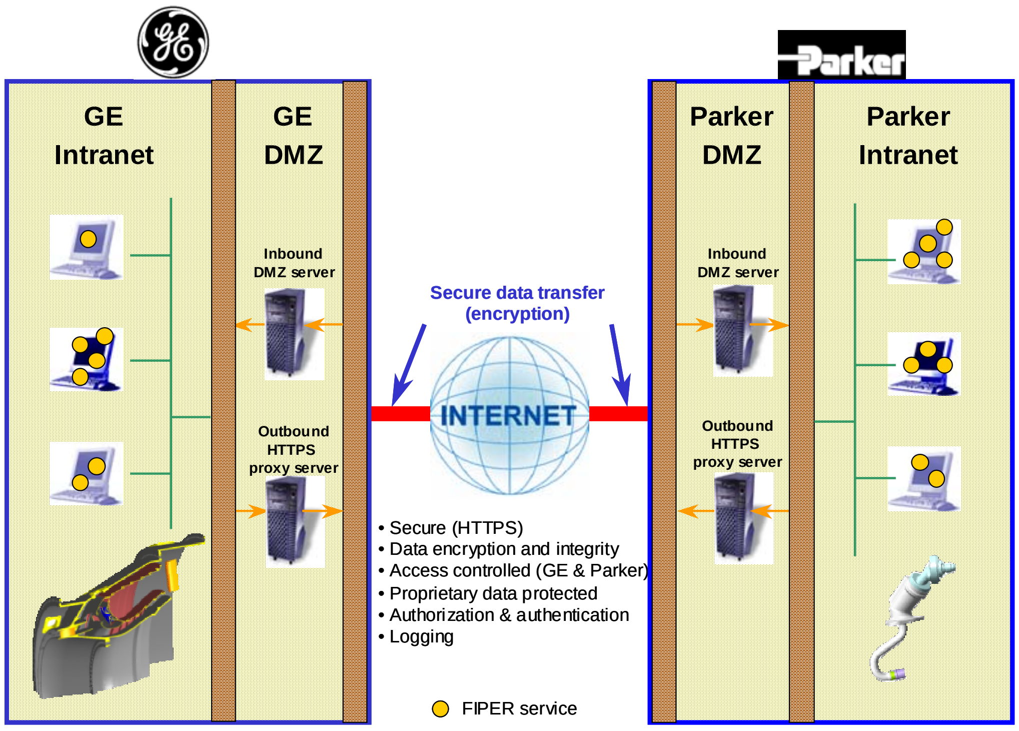
GE Intranet
GE DMZ
Inbound DMZ server
Outbound HTTPS proxy server
Secure data transfer (encryption)
INTERNET
Parker DMZ
Parker Intranet
Inbound DMZ server
Outbound HTTPS proxy server
• Secure (HTTPS)
• Data encryption and integrity
• Access controlled (GE & Parker)
• Proprietary data protected
• Authorization & authentication
• Logging
● FIPER service

Figure 8. B2B Framework

The business specific engineering service providers required for the fuel nozzle design are published to the network at the respective organizations. The following services are installed at GE:

1. *CAD Model Export Service*: Exports a model from a native CAD format after suppressing features and filtering geometric entities based on user-defined tags. ITI TranscenData (ITI) provides this functionality.
2. *CAD Model Import Service*: Imports a model into a user specified CAD format while maintaining user-defined tags on geometric entities. ITI TranscenData provides this functionality.

3. *OMG CAD Services*: Supports B-Rep model queries and parametric modification. This service implements OMG CAD Standards interface.
4. *ANSYS Service*: Executes ANSYS CAE application in batch mode, including meshing, solution, and post processing. This service is wrapped on top of the commercial ANSYS software [3]

The following services are installed at Parker:

1. *KBE Fuel Nozzle Service*: Creates and updates fuel nozzle design according to changes in design constraints and requirements. This service triggers a series of jobs that predict the design performance and update parameters based on built-in design rules. The KBE service is built on top of the Intent Knowledge Station by Engineering Intent (EI).
2. *CAD Model Export Service*: Same as above.
3. *CAD Model Import Service*: Same as above.
4. *OMG CAD Services*: Same as above.
5. *ANSYS Service*: Same as above, but installed on the Windows platform.
6. *Nozzle Insertion Check Service*: Checks if a nozzle so designed can be assembled into a combustor assembly. This is an in-house tool requiring a certain degree of user interaction.

<u>User Case: Fuel Nozzle Design Update</u>

The specific case of fuel nozzle design update presented in this paper is shown in Figure 9. The design process is initiated from GE when an Engineer at GE makes a design change that results in a change in the layout of the Combustor. The Intelligent Master Modeling concept is used to represent Combustor CAD (Computer Aided Design) model [31] in the Unigraphics CAD System. The design change results in a change in the interfaces that define the layout of the fuel nozzle (Step 1 in Figure 9). This design change in turn triggers a fuel nozzle design request that is sent to Parker across the business firewall. Along with the request, only the necessary Combustor interface information is transferred in a neutral CAD format using the ITI Transcendata's CAD Export Service. On the Parker side, with the updated interface information the fuel nozzle CAD model definition is created in Pro/E CAD System using the KBE service (Step 2 in Figure 9).

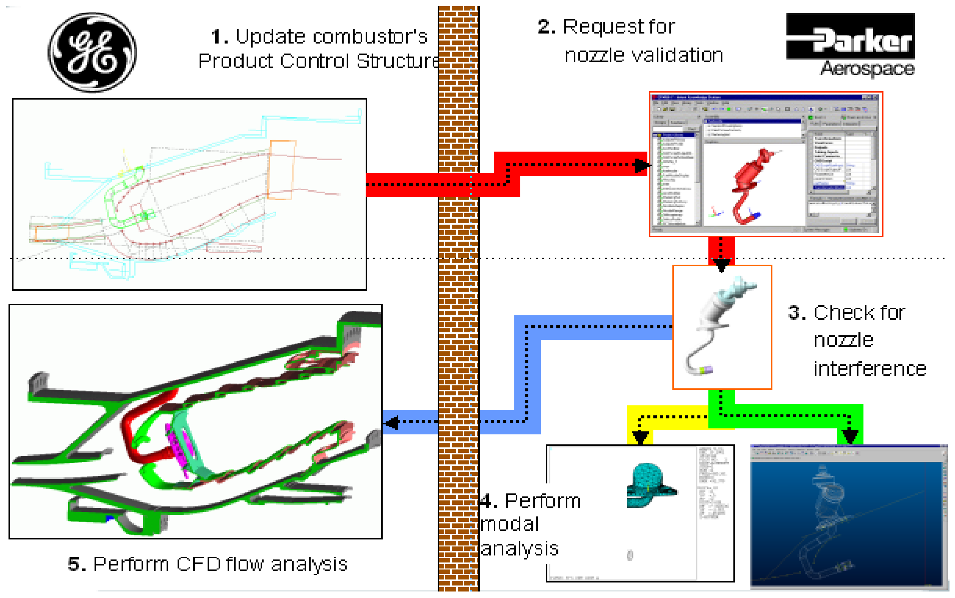
1. Update combustor's Product Control Structure
2. Request for nozzle validation
Parker Aerospace
3. Check for nozzle interference
4. Perform modal analysis
5. Perform CFD flow analysis

Figure 9: Fuel Nozzle Design Process

The new fuel nozzle design then needs to be validated by performing detailed downstream analysis. First, a fuel nozzle interference check is performed in the Pro/E CAD System (Step 3 in Figure 9) to ensure that the fuel nozzle assembly fits within the Combustor Assembly without any interference. The next step involves performing a Vibration Analysis (Step 4) of the fuel nozzle assembly to make sure that the natural frequencies of the system is within the specified limits. Once Parker finalizes the fuel nozzle design, the geometry information is then sent back to GE in a CAD neutral format. On the GE side, the neutral format from Parker is translated back into Unigraphics so that it can be referenced from within the Combustor CAD Model. With this updated Combustor design, a detailed CFD analysis of the flow over the fuel nozzle is performed to quantify the drag (Step 5) due to the fuel nozzle.

The above use case illustrates a real-time business-to-business collaboration and has been shown to result in considerable productivity and cost savings. Traditionally, the above process would require GE and Parker exchange data using a CD via FedEx which is very time consuming. In addition, this would require both GE and Parker to maintain an independent version of the geometry definition at each site in the native CAD system resulting in synchronization issues between the two sites. With the above implementation, a seamless integrated system has been developed ensuring accurate and secure transfer of data between the two organizations.

Business Application

To investigate the feasibility of developing business applications using a distributed environment a home mortgage application was implemented. The mortgage process involves several parties that interact with each other sharing data and resources in order to consummate the loan. The various parties in a mortgage process are the bank, credit agency, appraisal, insurance and buyer. All these entities need to federate together and execute a sequence of bipartite transactions to complete the mortgage transaction. Figure 10 shows a schematic of the interactions between the different parties involved in the transaction. In the transaction, the buyer first selects a bank service provider that provides loans for home purchases. The bank service interacts with the buyer to get pertinent information for the loan application. The bank then locates a credit service that provides a credit rating for the buyer. Once (if) the credit is approved, the bank uses an appraisal service that electronically provides the bank with the appraised value of the property. The bank approves the loan and communicates this to the buyer. The buyer then locates an insurance provider that provides home insurance for the property the in turn communicates with the bank with insurance information for the buyer. The documents could be directly exchanged with the entities or preserved in a common file store.

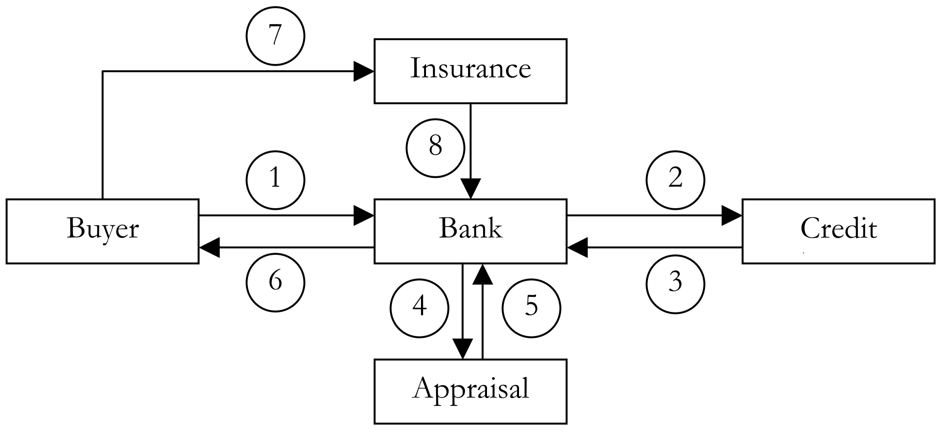

Figure 10: Mortgage Approval Process

In the prototype developed, these entities were defined as service providers or requestors that provide one or more services. Several tasks were defined for each service provider. For instance, the bank provides a loan approval service and the credit agency provides a credit rating service. The service broker and the bank are integrated together since the bank is the primary driver of the transaction. The goal of this specific investigation of a mortgage process is to demonstrate a feasibility study of executing complex business transactions using this architecture. The architecture was able to successfully execute the transactions in a well-choreographed fashion.

The payoff in terms of efficiency by using electronic services is tremendous, however, there are several issues that need to be addressed when financial applications are involved. Financial operations require a strong trust among the federated entities that is usually built using reputation and brand name recognition for traditional electronic as well as brick and mortar businesses. In an architecture that incorporates nameless and faceless entities, some alternate mechanism of trust is required among the entities. If the membership of the peer network is limited to prescreened vendors and the transactions are monitored by an enforcing authority, the trust issues are somewhat mitigated. In case the system is operating in an un-trusted environment, external mechanisms of trust can be employed.

Two alternate models of trust were developed a centralized model in which a single entity rates the different services and another in which chains of trust using mutual intermediaries is built. The two schemes are presented in the next section on security considerations.

5.Observations

The case studies show applications of a software architecture which are constituted of autonomous discoverable services on a network grid to engineering design and financial processing applications. This concept of using discoverable services that can create impromptu federations to consummate complex business transactions over the network provides greater flexibility and robustness making it attractive to several domains. The architecture removes the distinction between software and hardware services and can thus be used in home networking, where all appliances can be networked together enabling remote diagnostics as well as control via the network. There are several fundamental technology issues such as: security and trust implementation, and domain specific issues that need to be addressed as this approach is applied. However, none of the issues can be foreseen as insurmountable impediments to its widespread acceptance over time. The key attributes that make process automation successful are: flexibility, resilience, and robustness that are intrinsic to this architecture. The success of the applications also depends on the specific business case as well as the sociotechnical environment in which the technology is implemented.

Investigation of the two case studies for application of the proposed architecture not done with the express purpose of fitting the discussion into scientific framework for case studies in MIS described by Lee [36]. A discussion on how the investigation measures up to the criteria defined by Lee would be useful for the readers: 1) Can the following hypothesis be proven false? *"The service-based architecture is a suitable tool for managing complex business processes"*. 2) Are the predictions consistent with one another? 3) Does the case study confirm the hypothesis through empirical testing? 4) Does the case study rule out rival theories? Our hypothesis can be proven false if we were unable to demonstrate the applications. Improved robustness resilience and ability to manage process change are intuitively clear and were also observed. Tests of the architecture for several applications have proven its applicability. Multiple architectures have been used before with limited success. However, there is no guarantee that a new architecture that is even better than the one proposed will not emerge in the future. It would be an incorrect to state that the proposed architecture is suitable for all applications. It is an attractive architecture for problems involving complex processes, however, selection of the architecture depends on several other factors such as: application domain, expertise, usage, existing infrastructure, and cost.

The limitations of such systems include lack of standardized software and complexity of managing these systems. An additional concern is security that is expensive to maintain in the distributed environment where nameless and faceless entities need to trust each other and work together. Implementing security is expensive, thus security solutions need to be customized to the intended application such that applications are not encumbered with burden of managing security features that are not essential. The paper mentions that security is a key consideration. However, a detailed analysis of security in such architectures is beyond the scope of the current paper and will be discussed in a subsequent article.

6. Conclusions

The paper describes a versatile architecture to support collaborative problem solving involving multiple parties across multiple domains. This architecture uses ubiquitous autonomous services that reside on the network and federate dynamically to map into a specific process. The services have standardized interfaces allowing one service to be seamlessly replaced by another service in a transaction. The service-based computing model requires registries where services can register themselves and allow them to be discovered in real time for a federation. The conceptual model is generic and can be implemented on any platform that supports discovery and join protocols. This architecture reduces the brittleness in existing systems that break when the processes and tasks in the processes evolve over time. Several applications have been created in the domain of engineering and finance to demonstrate the concepts of this architecture. The architecture has reduced the design cycle time for preliminary aerodynamic analysis at GE by a factor of 10. It has also reduced the nozzle combustor development time from a few days to a few minutes. It has shown the feasibility of executing complex multiparty financial transactions on the Internet.

As Shani et al. [44] have rightfully concluded that technology is not enough to drive change in the organization. Rather, a *sociotechnical* approach is required. The technology selected should meld well with the organizational environment and work with the culture. There were several sociotechnical reasons for the success of this architecture. Firstly, the new architecture matched very closely with the designer behavior making it easy for them to adapt to the new environment. Designers are constantly evolving processes and want the tools to adapt to the changed processes rather than being saddled with obsolete rigid processes. Secondly, the new architecture provided much higher resilience and reliability: the lack of which had been a major factor in non-acceptance of earlier systems because it caused the designers frequent unanticipated delays. Thirdly, it supported the mandate of the management to promote interdisciplinary design for better performance and improved design cycle time.

The architecture removes the distinction between software and hardware services. The same concept can thus be used in home networking where all the appliances can be networked together enabling remote diagnostics as well as appliance control via the Internet. The primary limitations of such systems are lack of standardized software and complexity of managing these systems. An additional concern is security that is expensive to maintain in the distributed environment where nameless and faceless entities need to trust each other and work together. The security solutions that have been developed as a part of this work are customized to the intended application such that applications are not encumbered with the burden of managing security features that are not essential. Future work will involve working on models of trust among peers, investigating security issues in the implementation, and developing prototypes for networking hardware using these concepts.

References

[1] K. Aida, A. Takefusa, H. Nakada,S. Matsuoka, S. Sekiguchi and U. Nagashima, Performance evaluation model for scheduling in a global computing system, The International Journal of High Performance Computing Applications, 14, No. 3 (2000).

[2] David P. Anderson, Jeff Cobb, Eric Korpela, Matt Lebofsky, Dan Werthimer, SETI@home: An Experiment in Public-Resource Computing, (Space Sciences Laboratory, U.C. Berkeley, 2002).

[3] ANSYS Inc., http://www.ansys.com/.

[4] Ariba, IBM, and Microsoft, UDDI Technical White Paper, (2000). http://www.uddi.org.

[5] D. Box, D. Ehnebuske, G. Kakivaya, A. Layman, N. Mendelsohn, H.F. Nielsen, S. Thatte and D. Winer, SOAP: Simple Object Access Protocol, MSDN Library (2001).

[6] S. Capkun, L. Buttyan and J.P. Hubaux, Self-Organized Public-Key Management for Mobile Ad Hoc Networks, IEEE Transactions on Mobile Computing, 2, No. 1 (2003) 52-64.

[7] H. Casanova, Simgrid: A Toolkit for the Simulation of Application Scheduling, Proceedings of the First IEEE/ACM International Symposium on Cluster Computing and the Grid, (Brisbane, Australia, 2001).

[8] H. Casanova and J. Dongarra, NetSolve: A network-enabled server for solving computational science problems, The International Journal of Supercomputer Applications and High Performance Computing, 11(3) (1997) 212-223.

[9] E. Christensen, F. Curbera, G. Meredith and S. Weerawarana, Web Service Description Language (WSDL) 1.1., W3C Note, (2001). http://www.w3c.org/TR/wsdl.

[10] Clip2, The Gnutella Protocol Specification v0.4 (Document Revision 1.2), (June 15, 2001). http://www.clip2.com/GnutellaProtocol04.pdf.

[11] T. DeFanti, I. Foster, M. Papka, R. Stevens and T. Kuhfuss, Overview of the I-Way: Wide area visual supercomputing, International Journal of Supercomputing Applications and High Performance Computing, 10 (2) (1996) 123–131.

[12] H. Deng, W. Li, and D.P. Agrawal, Routing Security in Wireless Ad Hoc Networks, IEEE Communications Magazine, (2002) 70-75.

[13] W.K. Edwards, Core Jini, 2nd ed., Prentice Hall, (2000).

[14] Eisenhardt, K.M, building theories from case study research, Academy of Management Review, 14, No. 4, (1989) 532-550.

[15] I. Foster and C. Kesselman, Globus: A Metacomputing Infrastructure Toolkit, The International Journal of Supercomputer Applications and High Performance Computing, 11 (2) (1997) 115–128.

[16] I. Foster, C. Kesselman, J.M. Nick and S. Tuecke, The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration, Globus Project, (2002). http://www.globus.org/research/papers/ogsa.pdf.

[17] E. Freeman, S. Hopfer and K. Arnold (1999), JavaSpaces™ Principles, Patterns, and Practice, Addison-Wesley, (1999).

[18] Gnutella homepage, (2001). http://gnutella.wego.com/.

[19] S. Goel, M. Chai, D. Xu and B. Li, Efficient Peer-to-Peer Data Dissemination in Mobile Ad-hoc Networks, Proc. of International Workshop on Ad Hoc Networking, (2002).

[20] S. Goel, D. Cherry and B. Gregory, Knowledge-Based System for Preliminary Aerodynamic Design of Aircraft Engine Turbines, Applications of Artificial Intelligence XI: Knowledge-Based Systems in Aerospace and Industry, (Orlando, Florida, April 1993).

[21] S. Goel, J.I. Cofer and H. Singh, Turbine Airfoil Design Optimization, International Gas Turbine and Aeroengine Congress and Exposition, (Birmingham, UK, June 10-13, 1996).

[22] S. Goel, M. Sobolewski, Trust and Security in Enterprise Grid Computing Environment, Proceedings of the IASTED Intl., Conference on Communication, Network, and Information Security, (New York, NY, Dec 10-12, 2003).
[23] L. Gong, JXTA: A Network Programming Environment, IEEE Internet Computing, 5 (3) (2001) 88-95.
[24] S.L. Halter, Java Spaces: Example by Example, Prentice Hall PTR, (2002).
[25] Y.C. Hu, A. Perrig, and D.B. Johnson, Ariadne: A Secure On-Demand Routing Protocol for Ad Hoc Networks, MobiCom'02, (2002).
[26] D. Ingram, Soft Real Time Scheduling for General Purpose Client-Server Systems, Proc. of the 7th Workshop on Hot Topics in Operating Systems, (1999).
[27] Jini.org Web site, http://www.jini.org/
[28] Jini Architecture Specification, (2000). http://www.sun.com/jini/specs/jini1_1.pdf.
[29] W. Johnston, D. Gannon, and B. Nitzberg, Grids as production computing environments: The engineering aspects of NASA's information power grid, Proc. Eighth IEEE International Symposium on High Performance Distributed Computing, (1999).
[30] M. Just, E. Kranakis and T. Wan, Resisting Malicious Packet Dropping in Wireless Ad-Hoc Networks Using Distributed Probing, In: Proceedings of 2nd Annual Conference on Adhoc Networks and Wireless (ADHOCNOW'03), (Montreal, Canada, 2003).\
[31] K.J. Kao, C.E. Seeley, S. Yin, R.M. Kolonay, T. Rus and M. Paradis, (2003) Business-to-Business Virtual Collaboration of Aircraft Engine Combustor Design, Proceedings of ASME 2003 Design Engineering Technical Conference, (Chicago, IL, 2003).
[32] A.K. Kapur, N. Gautam, R.R. Brooks, and S. Rai, Design, Performance and Dependability of a Peer-to-peer Network Supporting QoS for Mobile Code Applications, Proc. of 10th Intl. Conf. on Telecom. Sys., Modeling and Analysis, (2002).
[33] M.A. Kolb and M.W. Bailey, FRODO: Constraint-Based Object-Modeling for Preliminary Design, Advances in Design Automation, (1993) 307-318.\
[34] I. Kumaran, JINI Technology: An Overview, Prentice Hall; 1st edition, (2001).
[35] M. Lapinski, M. Sobolewski, International Journal of Concurrent Engineering: Research & Applications, Dec 2002.
[36] Lee, A.S., A scientific methodology for MIS Case Studies, MIS Quarterly, (March 1989) 33-50.
[37] Napster homepage, 2001. http://www.napster.com/
[38] M.A. Natrajan, Humphrey and A.S. Grimshaw, Grids: Harnessing geographically-separated resources in a multi-organisational context, 15th Annual International Symposium on High Performance Computing Systems and Applications (2001).
[39] J. Newmarch, A Programmer's Guide to Jini Technology, Apress; 1st ed, (2000).
[40] Lily Shue, Kamal Khan and Rob Harmer, Peer-to-Peer Networking Security and Control, IT Governance Institute, (2003).
[41] Frank Sommers, Jini Starter Kit 2.0 tightens Jini's security framework, JavaWorld, (2003). http://www.javaworld.com/javaworld/jw-05-2003/jw-0509-jiniology_p.html.
[42] Sanchez, R., Mahoney, J.T, Modularity, flexibility, and knowledge management in product and organizational design, Strategic Management Journal, 17, (1996) 63-76.
[43] Sarker, S., Lee, A.S., Using a positivist research methodology to test three competing theories-in-use of business process redesign.
[44] Shani, A.B., Grant, R.M., Krishnan, R., Thompson, E., Advanced Manufacturing systems and organizational choice: sociotechnical system approach, California Management Review, Summer 1992, 91 -111
[45] Sobolewski, Federated P2P Services in CE Environments, Advances in Concurrent Engineering, A.A. Balkema Publishers, keynote paper, (2002) 13-22.

[46] Sobolewski, FIPER: The Federated S2S Environment, JavaOne, Sun's 2002 Worldwide Java Developer Conference, (San Francisco, 2002). http://servlet.java.sun.com/javaone/sf2002/conf/sessions/display-2420.en.jsp.

[47] M. Sobolewski, Knowledge-Based System Integration in a Concurrent Engineering Environment. J. Komorowski and Z.W. Ras (Eds.) Methodologies for Intelligent Systems, Lecture Notes in AI, No 689, Berlin: Springer-Verlag, pp. 601-611, 1993.

[48] M. Sobolewski, S. Soorianarayanan, R.K. Malladi-Venkata, Service-Oriented File Sharing, Proceedings of the IASTED Intl., Conference on Communications, Internet, and Information technology, (Scottsdale, AZ, Nov 17-19, 2003) 633-639.

[49] A. Takefusa, S. Matsuoka, H. Ogawa, H. Nakada, H. Takagi, M. Sato, S. Sekiguchi and U. Nagashima, Multi-client Performance Analysis of High-Performance Global Computing, Proceedings of the 1997 ACM/IEEE Supercomputing Conference (1997).

[50] S.S. Tong, D.J. Powell, and S. Goel, Integration of Artificial Intelligence and Numerical Optimization Techniques for the Design of Complex Aerospace Systems, 1992 Aerospace Design Conference, (Irvine, CA, February 1992), AIAA-92-1189.

[51] S. Vadhiyar and J. Dongarra, GrADSolve - A Grid-based RPC system for Remote Invocation of Parallel Software, Journal of Parallel and Distributed Computing, (2003).

[52] M. Van Steen, P. Homburg and A. Tanenbaum, Globe: A wide-area distributed system, IEEE Concurrency, 7(1) (1999) 70–78. http://www.cs.vu.nl/˜steen/globe/.

[53] K. Wang and B. Li, Efficient and guaranteed service coverage in partitionable mobile ad-hoc networks, IEEE INFOCOM, (2002).

[54] M. Zhou and Z.J. Haas, Securing Ad Hoc Networks, IEEE Network Magazine, 13 No.6 (1999). http://static.userland.com/gems/magistris/introducingGroove.pdf.

[55] B. Zhao, J. Kubiatowicz, and A. Joseph, Tapestry: An Infrastructure for Fault-Tolerant Wide-area Location and Routing, Berkeley Technical Report No. UCB/CSD-01-1141, (Computer Science Division, University of California, 2001).