Federated P2P Services in CE Environments

Michael Sobolewski
GE Global Research Center, Niskayuna, New York, USA

ABSTRACT: The goal of the Federated Intelligent Product Environment (FIPER) environment is to form a federation of distributed services that provide engineering data, applications and tools on a network. A highly flexible software architecture has been developed, in which engineering tools like computer-aided design (CAD), computer-aided engineering (CAE), product data management (PDM), optimization, cost modeling, etc., act as distributed service providers and service requestors. Service providers can enter the federation by registering with a service registry and publish the services through a process of discovery and join. The individual services communicate via so-called context models, which are abstractions of the master model of a particular product. Many instances of mechanical analysis have been implemented. FIPER supports three centricities and deploys three neutralities. FIPER's three centricities are network centricity, service centricity, and web centricity. The three neutralities FIPER deploys are location neutrality, protocol neutrality, and implementation neutrality. In a federated peer-to-peer (P2P) environment, object-oriented concepts are applied to the network.

1 INTRODUCTION

GE has teamed with Engineous Software, BFGoodrich, Parker Hannifin, Ohio Aerospace Institute, Ohio University, and Stanford University in a four-year effort to develop the Federated Intelligent Product Environment (FIPER) under the sponsorship of the National Institute for Standards and Technology (NIST). FIPER strives to drastically reduce design cycle time, and time-to-market by intelligently automating elements of the design process in a linked, associative environment, thereby providing true concurrency between design and manufacturing. This will enable distributed design of robust and optimized products within an advanced integrated web-based environment.

The systematic integration of humans with the tools, resources, and information assets of an organization is fundamental to concurrent engineering (CE). In an integrated environment, all entities must first be connected, and they then must work cooperatively. Services that support concurrency through communication, team coordination, information sharing, and integration in an interactive and formerly serial product development process provide the foundation for a CE environment. Product developers need a CE programming environment in which they can build programs from other developed programs, built-in tools, and knowledge bases describing how to perform a complex design process. In every system, some part handles details unique to each user. In client/server systems, this is typically a client program. In monolithic systems, some program modules maintain status information for each user and provide an interface to other system modules. In a peer-to-peer (P2P) model (Oram 2001, Sobolewski 1996) system components dedicated to individual users are shared peers acting on behalf of those users. P2P environments distribute intelligence throughout the entire system, allowing individual components to remain simple. Interaction between components is minimized through the use of peers, which adapt automatically to changes in components. These are features that are necessary in large flexible design and manufacturing systems.

P2P is not about file sharing, music, or Napster, nor is it a collection of tools and applications. It is just a computing concept. It defines all decentralized distributed components in the system to be equal. These components might be devices, computers or objects on the network. In the FIPER environment, peers are distributed objects of the same type. They are equal since they define a common top-level interface, but the interface might be implemented differently by different groups of network objects. Rather than the currently prevalent client/server model, in which all communication passes through and is controlled by a central server (e.g., web, FTP, mail, and application servers), in P2P, the communication goes directly from one client's object to another client's object. Because accessing these decentralized object nodes means operating in an environment of unstable connectivity and unpredictable IP addresses, P2P nodes must operate outside the domain name system (DNS) and have significant or total autonomy from central servers (Edwards 1999, Li 2000).

The FIPER environment applies object-oriented techniques directly to the network. This software architecture houses a pool of peer objects called service providers. Providers provide two types of services: data services (called *context services*) and remote operations (called *method services*). Context services create dependencies by observing data they depend on and adapt dynamically to changes in the data. Since service providers serve each other in a uniform way, This type of model is called federated service-to-service (S2S) computing. The architecture is simultaneously web-centric, service-centric and network-centric. The web-centricity enables transparent web-based access to the globally distributed data and the pool of services. The individual services can act within this framework, both in the role of providing services (server mode) and requesting services (client mode). When requesting aggregated services, the FIPER infrastructure also brokers the requests, delegating them to the appropriate registered component's services.

The three centricities of the FIPER system are clearly identified in Figure 1. The client accesses the system portal through HTTP requests, indicating the web-centric approach. The network-centric concept that the service is the network is represented by FIPERnet, which contains various service providers. Since all the providers form a network-centric environment, the system portal or stand-alone requestor finds directly or indirectly a particular provider's proxy and then communicates with the provider. The direct access uses discovery and lookup protocols (Edwards 1999, Li 2000) to find a proper provider on the network. Indirectly, a provider can access a particular proxy in its dynamic catalog of currently available services via a provider called *cataloger*. Also, a provider can drop a task into an available shared object space (Freeman et al 1999) via a provider called *dropper*. Within FIPERnet, all service providers are connected to multiple object spaces and execute relevant tasks by taking a task and returning results to the object space. The persistence provider (*persister*) uses the file store to handle files or JDBC (Zhao et al. 2001, Reese 2000) to talk to the underlying database. As far as the client is concerned from the service-centric perspective, getting a service from the desired provider is the ultimate goal. This entry service becomes the network for its requestor.

The FIPERnet capabilities can be obtained and configured at a moment's notice. New business applications can be assembled as needed on the fly by integrating new capabilities into existing workflows, systems, devices and applications. FIPERnet reduces the costs of solving business problems as well as of establishing and maintaining online business relationships. Services are provided by shared low-cost peers and are easily integrated into the core business of an enterprise.

Each computational module, independent of how large it is or how it is distributed, has three basic components: data, operations and control strategy. All of these are described herein with an explanation of how they constitute a megaprogram and are used in a federated S2S environment. A megaprogram is a program whose execution requires milions of objects to be used.

2 S2S DATA: CONTEXTS AND CONTEXT MODELS

The implementation of *natural language* knowledge definition and editing critically depends on the intricacy of translation between natural language constructs and internal knowledge representation struc-

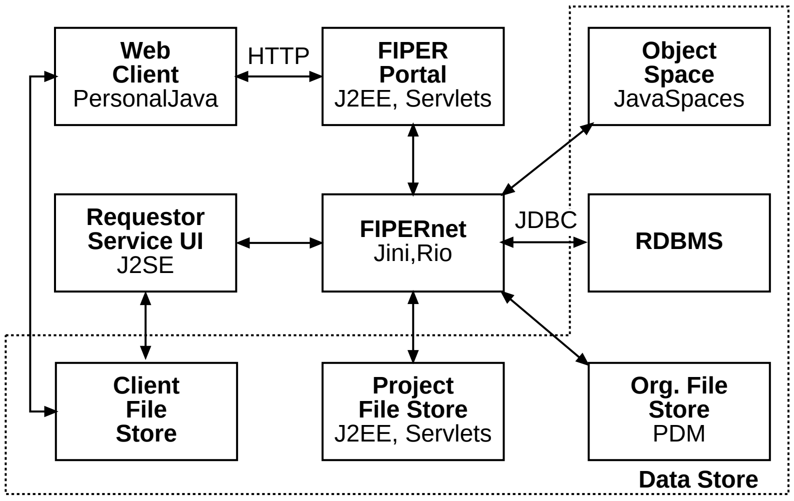

Figure 1. FIPER organizational architecture.

tures. This is a function of the chosen knowledge representation method. In humans, knowledge and thinking are believed to be based in large part on perceptual data processing (Arnheim 1969). This fact is, in turn, reflected in the structure and semantics of natural languages. The *percept-calculus* knowledge representation scheme used in the DICEtalk system (Kulpa et al. 1991, Sobolewski & Kulpa 1984, Sobolewski 1989a, b, Sobolewski 1990, Sobolewski 1991a, b) is based on this assumption.

In the percept formalism, an entity of the world is treated as the image given by perception, and that image is called a *percept*. A *percept conceptualization* is the semantic counterpart of the syntactic level of the knowledge description theory called *percept calculus*. In the percept conceptualization, attributes and their values are used as atomic conceptual primitives, and complements are used as molecular ones. A *complement* is an attribute sequence (*path*) with a value at the last position. An elementary *percept property* consists of a percept subject and a set of percept complements, and usually corresponds to a simple sentence of natural language. For example, a bit of knowledge about some dogs is carried by the following English sentences:

```
Fido is a dog.
White and black are colors.
Size may be long.
Fido has a long tail.
Fido’s fore legs are black and hind legs
  are white.
```

may be written in the “simplified English” language of the system as follows:

```
Tail of the dog Fido: size is long.
Legs of the dog Fido: fore color is black,
  hind color is white.
```

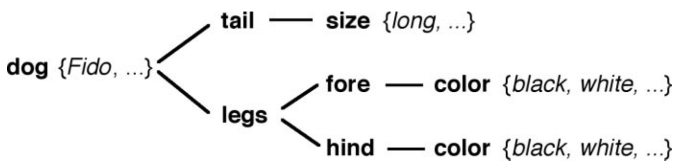

Figure 2. Attribute tree structure.

and produces the following percept calculus representation:

```
(dog, Fido: tail, size, long)
(legs, dog, Fido: fore, color, black;
  hind, color, white)
```

and the attribute tree structure shown in Figure 2, with attributes dog, tail, size, legs, fore, hind, color, and values Fido, long, black, white.

A FIPER context model is the basic element of FIPER data structures and is based on the percept calculus knowledge representation scheme. It forms the essential structure of the data being processed. A context model in the FIPER system is represented as a tree-like structure of context nodes. A data node is where the actual data resides. 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.

A partial context model structured for a turbine airfoil mechanical analysis is depicted in Figure 3. Ovals in the figure represent context nodes, which in turn form the paths of this model. The rectangle shape indicates the data node, which is located at the leaf. Since the FIPER environment is a federation of distributed services, the ability to share and access context models is itself one of the FIPER services. The first step for clients sharing a context model in a persistent and concurrent manner is finding the service provider, as they would for any other services.

Persistence services provide access to static context models in a data store, for example a PDM system. However, there are many context providers who also serve dynamic design data that represent parts in the product control structure. These context providers are linked via the observer/observable pattern (Grand 1999) to form a virtual network-centric product, such as an aircraft engine.The details for locating a service provider are presented in Section 5 below.

Data persistence control is a critical element of the FIPER environment, in which data from its context models and distributed business logic are available globally. In this model, the distributed clients might

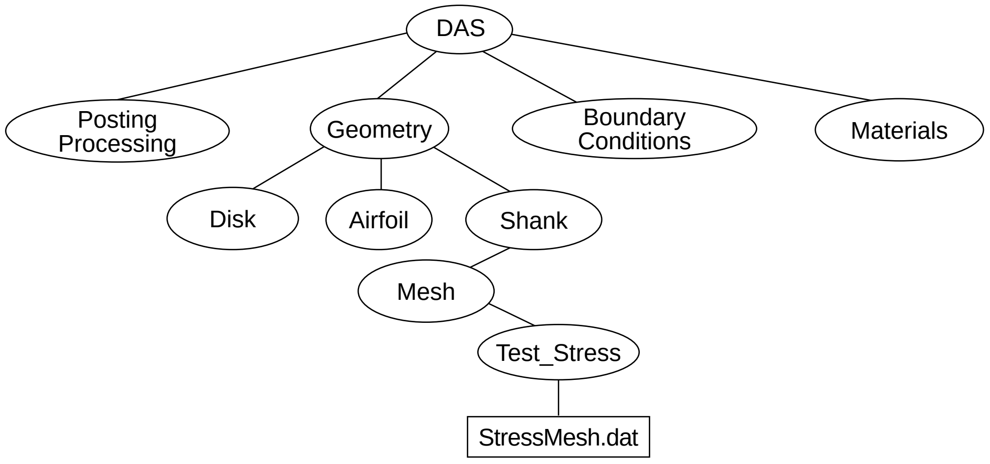

Figure 3. Example context model for turbine analysis showing the tree-like node structure.

request services concurrently for the same data. The data persistence provider can also provide database independence, so that multiple database vendors can be adopted (Zhao et al. 2001, Reese 2000). While persistence providers maintain data in static databases, context providers maintain dynamic distributed data as a product control structure. The latter is a network-centric, object-oriented form of distributed product data management.

3 S2S OPERATIONS: METHODS AND PROVIDERS

Communities of peers as network objects provide the domain business logic in the FIPER environment. The peers are network objects that are registered with object registries (Fig. 4). Each peer may implement multiple interfaces that are published when the peer joins the environment. All methods of these interfaces have the same format:

```
public FiperContext operationName(Fiper-
  Context)
```

Both arguments and return values of these methods are instances of type FiperContext that represent context models described in Section 2 above. By its interface (type) and optional attributes (e.g., provider name), the network object can be dynamically found on the network without a host name and port required. These interfaces and their implementations might change, as they are specific to particular service providers. Thus peers should not expose their specific interface explicitly at the S2S infrastructure level.

The top-level peer interface called *Servicer* is defined as follows:

```
public interface Servicer {
// Put into action the specified exertion
public Exertion service(Exertion exer-
  tion)
throws RemoteException, ExertionExcep-
  tion;
// Monitoring methods
…
}
```

So, all peers implement this interface and their equality is defined as being service providers or *servicers*. A *service* is an act of requesting a `service(Exertion)`operation from a service provider. The exertion is a distributed activity defined by the Exertion interface as follows:

```
public interface Exertion {
// Apply this exertion method to the spec-
  ified context
public Exertion exert()
throws RemoteException, ExertionExcep-
  tion;
…
}
```

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

A *method* is primarily defined by an interface and a selector (operation name) of an operation 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 task method allows the FIPER system to bind the task to the network object and process the task's context by one of its peer's operations, which is defined by its published interface. This type of service provider is called a *method provider.*

There are four types of methods: *preprocess*, *process*, *postprocess* and *append*. Only one method of process type can be associated with a task, but multiple methods of other types can be attached to the same task. The process method associated with a task defines the method provider to which the task is bound in runtime. All preprocess, process, and postprocess methods are executed by method providers.

Append methods bind to context providers, peers that maintain dynamic data (context models) on the network. For example, a turbine blade and disk context model might be maintained by context providers for a mechanical airfoil analysis. A method provider processing a task associated with an append method binds to the context provider and submits a context template associated with the append method. The returned context model from the context provider is appended to the task's context model and used later by remaining methods in the task. This dynamic context specified by the context template complements the static context model explicitly provided in the task.

The dynamic context models are observed by dependent context peers as well as method providers who use the dynamic data. This observer/observable dependency (Grand 1999) forces context providers to

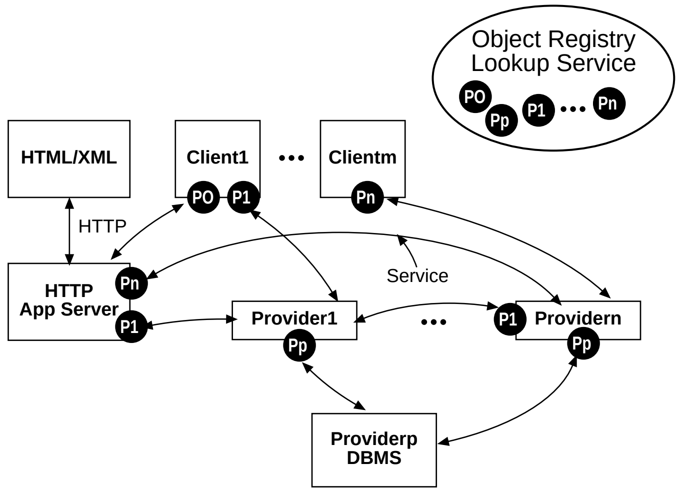

Figure 4. Service providers in a S2S environment.

adapt dynamically to changes while they occur and update their state according to their constraints. If these changes are allowed then all observers are notified about the changes, otherwise changes are refused and the requesting servicer is notified about the observable constraints violation.

The *job* is a compound exertion that is comprised of tasks and other jobs. A job is a recursive structure expressed in terms of exertions. While a classic computer program is an organized list of statements in a programming language, a job's exertions can be treated as distributed statements that tell the FIPER environment what to do with task context models. Since the execution of a job might require millions of objects to be used by all involved service providers a job is called a *megaprogram* and the process of job creation is called *megaprogramming*. The concept of service binding and a job as a distributed megaapplication is depicted in Figure 5.

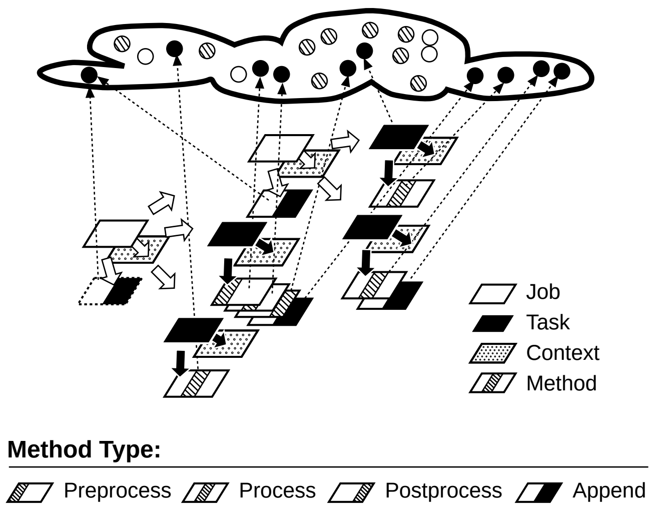

Figure 5. Service binding in a job as a megaapplication.

4 S2S CONTROL STRATEGY

In Section 3 above, basic S2S operations were described. This section will present how they are used, at the task level and job level, to execute a distributed megaprogram. As defined before, a task is associated with a collection of methods. There is only one process method in this collection and multiple instances of append, preprocess, and postprocess methods. The process method is responsible for binding to the service provider that executes the task. The task submitted by a service requestor can be submitted directly or indirectly. In the direct approach the service requestor finds the relevant provider using a task process method and submits the task to this provider. Alternatively, a service requestor can use a system object space (Freeman et al. 1999) and simply drops the task into the space. Each service provider looks continuously into the space for tasks that match a provider's interfaces and attributes. Each service provider that picks up a matched task from the object space returns the task being executed back into the space, then the requestor picks up the executed task from the space. The object space provides a kind of automatic load balancing – the fastest available service provider gets a task from the space. When a service provider gets a task then the task methods are executed in the following order:

1 All append methods are executed first and in the result the task's context model is complemented with dynamic data delivered from context providers specified by these methods.
2 All preprocess methods are executed in the order specified in the task; in the result the task context model is ready for applying a process method.
3 The process method is executed and results are captured in the task context model.
4 All postprocess methods are executed in the order specified in the task; in the result the task context model is ready to be returned to the requestor.

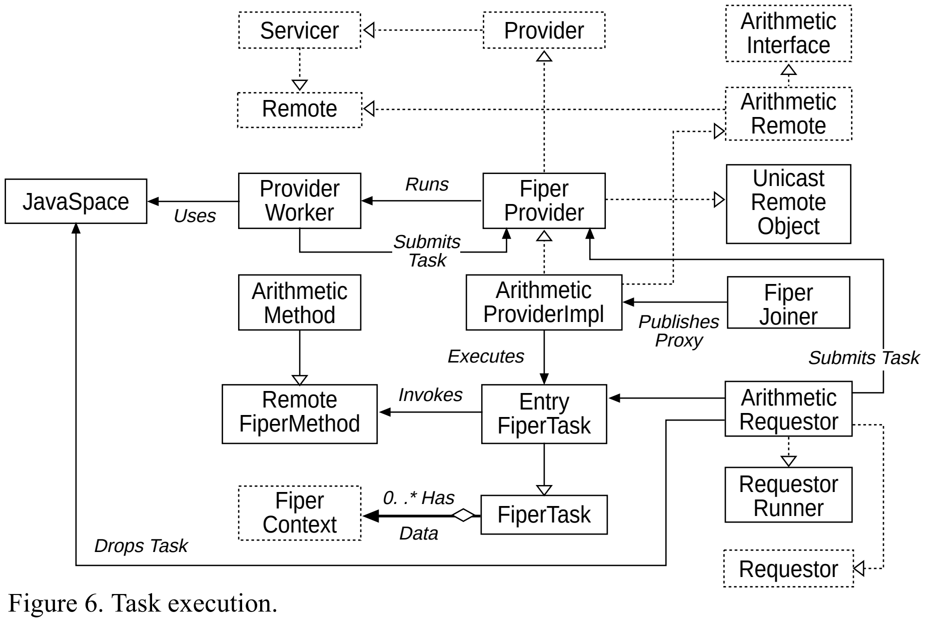

Figure 6. Task execution.

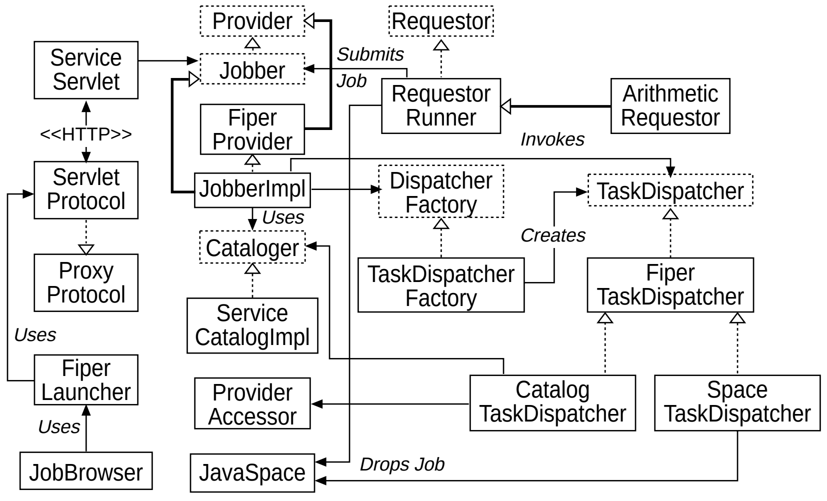

Figure 7. Job execution.

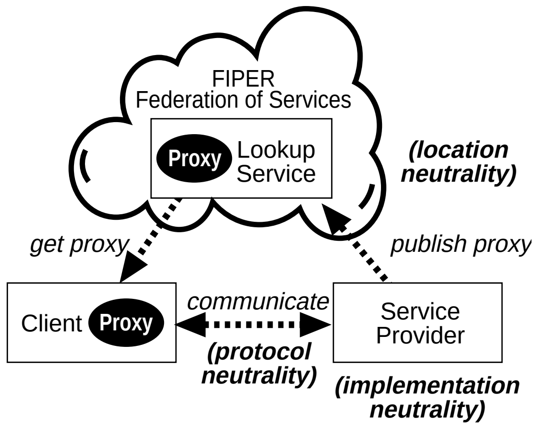

Figure 8. FIPER's three neutralities, providing a simplified, highly flexible software environment.

The UML diagram illustrating the task execution is presented in Figure 6.

Jobbers are specialized service providers that execute jobs. A jobber coordinates execution of exertions in a job using the job context model called a *control context*. Similarly as for a task, a job's process method defines a runtime binding to a specific jobber. A control context defines a job's execution strategy. A strategy implements a master/slave computing model (Freeman et al. 1999) with sequential or parallel execution of slave exertions with the master exertion executed as the last one. In general, full algorithmic logic operations: concatenation, conditional, iterative and break are supported for exertion execution. The strategy also specifies the way a jobber accesses service providers: directly using a utility called *ProviderAccessor*, using a catalog of currently available service or indirectly via an object space (Freeman et al. 1999). The catalog is just another service provider called a *cataloger*. In the FIPER environment there are eight types of dispatchers that implement different types of control strategies. A relevant dispatcher is assigned to a jobber by a dispatcher factory based on a job control context. The UML diagram illustrating the job execution is presented in Figure 7.

5 IMPLEMENTATION OF THE FIPER S2S ENVIRONMENT

FIPER supports three centricities and deploys three neutralities (Zhao et al. 2001, Röhl et al. 2000). FIPER's three centricities are network centricity, service centricity, and web centricity. A FIPER federation is composed of various service providers (FIPERnet in Figure 1); any of these can come and go, and the system can respond to changes in its environment in a reliable way (network centricity). Services in FIPERnet can discover lookup services and join the federation or lookup for relevant services in order to cooperate in a distributed environment (service centricity). Users can request to use multiple services and check the status of their submissions in different locations through an HTTP portal with thin web clients (web centricity).

The three neutralities FIPER deploys are location neutrality, protocol neutrality, and implementation neutrality (Fig. 8).

With location neutrality, services need not be colocated; lookup services are discovered and used to find a particular service, which simplifies management of the entire network environment With protocol neutrality, the way in which clients communicate with a service provider is not important. Clients are not aware of what protocols are used or where the implementations reside. With implementation neutrality, the clients who use the FIPER services do not need to know what languages are used or how a service is implemented. In all, FIPER provides accessibility through web-centric architecture; self-manageability using federated services, scalability via network centricity, and adaptability with the power of mobile code inserted for execution through service providers.

To be able to achieve the network-centricity and service-centricity requirements, The Jini™ network technology from Sun Microsystems (Edwards 1999, Li 2000) is being used. Jini is a set of specifications allowing federations of services on a network and providing a framework that allows those services to participate in certain types of operations. As opposed to server-centricity, where all the data and services are located in a specific or predefined server, network-centricity can allow many services at unknown locations to be found and then executed. Service providers are found and resolved through a lookup service provider or *registrar*. New service providers are added to the registrar by discovery and join. To publish a service, its service provider first uses a discovery protocol to locate an appropriate lookup service and then joins, or *registers*, with the lookup service. Services can communicate with each other in the entire federation, creating communities of services.

Each service in the network is an autonomous business logic unit that can serve other units and also be served by them. Data persistence functionality is provided by one of the services that exist in the network. This persistence service handles client requests and communicates with the product data repository. It provides concurrent sharing and data access.

The lookup service is similar in principle to the naming and directory server used in server-centric distributed network environments. For each service the lookup service holds the service attributes along with its proxy. An application that wants to use a Jini service finds the desired service by matching the service's attributes within the lookup service. A Jini service provider must register itself with a Jini lookup service and maintain an active registration in order for applications to find its service.

To locate a Jini lookup service, which itself is also a Jini service, Jini provides three discovery protocols. When a FIPER service provider starts, it discovers all relevant lookup services on the network through discovery. Then it registers with a discovered lookup service by using Jini's join protocol. In addition, the service provider may listen for new lookup services that start on the network and join them if desired. After the service is registered, its provider has to actively manage its relationship with the lookup service. By managing the relationship, service providers have to renew the lease they received when they registered with the lookup service. Otherwise, the lookup service will assume this service provider has gone away and free the resource allocated. Figure 9 illustrates how a service provider seeks a lookup service and then registers its proxy.

For a client to be able to use services registered with a Jini lookup service, the client needs to discover the Jini lookup service as well. After the lookup service is located, the client will perform lookup for the prospective service according to its required attributes. The lookup service will pass back a copy of the service proxy to the client, and then the client will communicate with the service provider via the proxy, as illustrated in Figure 10.

The FIPER functional architecture overview is presented in Figure 11. All layers depend on the job/task/context/method framework presented above in terms of S2S data, operations and control strategy. Web clients and stand-alone requestors submit jobs to be executed by FIPERnet. Any peer to one of the available jobbers can submit jobs. In the case of web clients, jobs are submitted to a FIPER interportal or extraportal. A FIPER extraportal is an application proxy that forwards incoming requests to a FIPER interportal. The extraportal is used for business-to-business secure communication over security firewalls. The FIPER interportal is implemented by three servlets: controller, dispatcher and file upload servlet (Hunter & Crawford 1998, Callaway 1999). The controller implements a command execution engine. The controller executes commands mostly related to database processing. For example, it authenticates users, manages access control list for FIPER business objects, or manages notifications for the FIPER notification manger (Lapinski & Sobolewski 2001) called *notifier*. The dispatcher servlet dispatches requestes into FIPERnet.

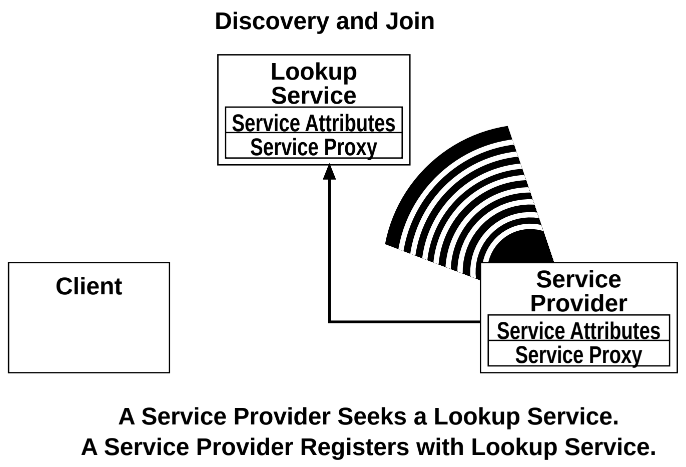

Figure 9. The discovery lookup and communicate protocols used by clients of Jini services.

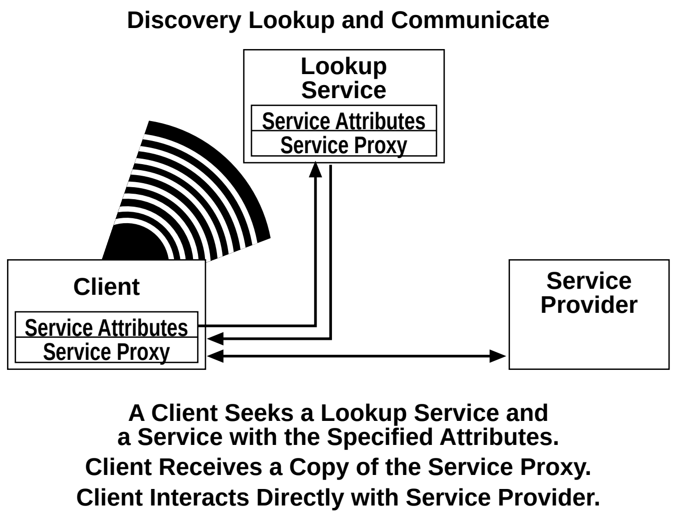
Discovery Lookup and Communicate
Lookup Service
Service Attributes
Service Proxy
Client
Service Attributes
Service Proxy
Service Provider
A Client Seeks a Lookup Service and a Service with the Specified Attributes.
Client Receives a Copy of the Service Proxy.
Client Interacts Directly with Service Provider.

Figure 10. The discovery and join protocols used for registering a Jini service.

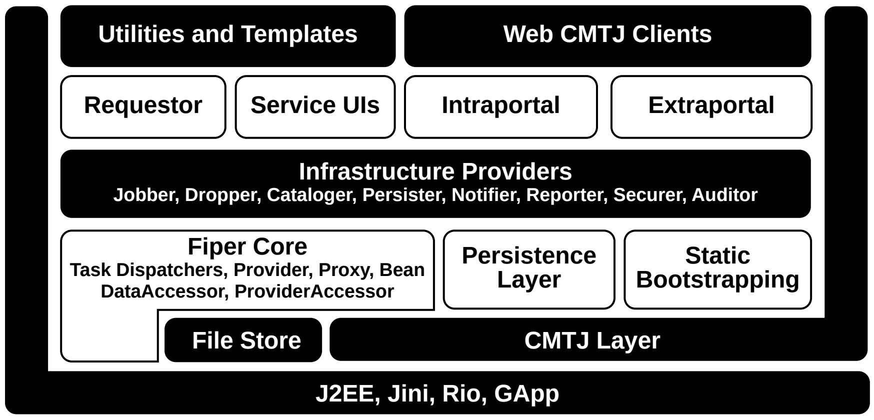

Figure 11. FIPER functional architecture overview.

6 CONCLUSIONS

In the S2S environment, object-oriented concepts are applied to the network. A job (a collection of exertions) is a distributed megaapplication executed in a federated S2S environment across multiple enterprises. Jobs are created using friendly, interactive web-based graphical interfaces. Jini connection technology from Sun Microsystems enables federated S2S, platform independent, real-world megaprogramming environments. It allows us to realize a whole aircraft engine as a virtual object-oriented product control structure that can be manipulated by multidisciplinary teams as a network-centric, active, evolving product in a distributed observer/observable mode. The FIPERnet capabilities can be obtained and configured at a moment's notice. New concurrent engineering applications can be assembled as needed on the fly by integrating new capabilities into existing workflows, systems, devices and applications. FIPERnet reduces the costs of solving business problems as well as of establishing and maintaining online business relationships. Services are provided by shared low-cost, easy-to-develop service providers and are easily integrated into the core business of an enterprise.

7 REFERENCES

Arnheim, R. 1969. *Visual Thinking*. University of California Press.

Callaway, Dustin R. 1999. *Inside Servlets*, Addison-Wesley.

Edwards, W. Keith 1999. *Core JINI.* Prentice Hall.

Freeman, Eric, Hupfer, Suzanne, Arnold, Keen. 1999. *JavaSpaces Principles, Patterns, and Practice*. Addison-Wesley.

Grand, Mark 1999. *Patterns in Java, Volume 1.* Wiley.

Hunter, Jason & Crawford, William 1998, *Java Servlet Programming.* O'Reilly & Asssociates.

Kulpa, Z., Sobolewski M. & Dwivedi, S.N. 1991. Graphical User Interface with Object-Oriented Knowledge-Based Engineering Environment; CAD/CAM, Robotics and Factories of the Future '90, Vol. 1. In Dwivedi, S.N., Verma, A.K. & Sneckenberger, J.E. (eds), *Concurrent Engineering*: 154–159. Berlin: Springer-Verlag.

Lapinski, Michael, & Sobolewski, Michael 2001. Notification Manager in the FIPER Environment, *Proc. of the 8th Int. Conference on Concurrent Engineering: Research and Applications*, Anaheim, CA.

Li, Sing 2000. *Professional Jini.* Wrox Press. ISBN 0-13-014469-X.

Oram, Andy (ed.) 2001. Peer-to-peer, Harnessing the Benefits of a Disruptive Technology. O'Reilly & Associates.

Reese, George 2000. *Database Programming with JDBC and Java, 2nd ed.* O'Reilly & Asssociates, ISBN: 1-56592-616-1.

Röhl, Peter J., Kolonay, Raymond M., Irani Rohinton K., Sobolewski, Michael, Kao, Kevin 2000. A Federated Intelligent Product Environment. AIAA-2000-4902, *8th AIAA/USAF/NASA/ISSMO Symposium on Multidisciplinary Analysis and Optimization, Long Beach, CA, September 6-8, 2000.*

Sobolewski M. 1989a. Percept Knowledge Description and Representation. *ICS PAS Reports.* No. 663. Warsaw, Poland.

Sobolewski, M. & Kulpa, Z. 1984. From Sentences to Attribute Networks. In Plander, I (ed.), *Artificial Intelligence and Information-Control Systems of Robots*:345-348. North-Holland.

Sobolewski, M. 1989b. EXPERTALK: An Object-oriented Knowledge-based System. In Plander, I. (ed.), *Artificial Intelligence and Information-Control Systems of Robots*. North-Holland.

Sobolewski, M. 1990. DICEtalk: An Object-oriented Knowledge-based Engineering Environment; CAD/CAM, Robotics and Factories of the Future '91, Vol 1. In Dwivedi, S.N., Verma, A.K. & Sneckenberger, J.E. (eds). *Concurrent Engineering*:117-122. Berlin: Springer-Verlag.

Sobolewski, M. 1991a. Object-oriented Knowledge Bases in Engineering Applications. *Proc. of the 6th Int. Conference on CAD/CAM, Robotics and Factories of the Future.* London, UK.

Sobolewski, M. 1991b. Percept Conceptualizations and Their Knowledge Representation Schemes. *Proc. of the 6th International Symposium on Methodologies for Intelligent Systems*. Charlotte, NC. Lecture Notes in Ras, Z.W. & Zemankova, M. (eds), *AI 542*: 236–245, Springer-Verlag.

Sobolewski, M. 1996. *Multi-Agent Knowledge-Based Environment for Concurrent Engineering Applications, Concurrent Engineering: Research and Applications*. Technomic.

Zhao, Shuo, & Sobolewski, Michael 2001. Context Model Sharing in the FIPER Environment, *Proc. of the 8th Int. Conference on Concurrent Engineering: Research and Applications*, Anaheim, CA.