C. Project Description (including results from prior NSF support)

The name OCEAN is an acronym for "Open Computation Exchange and Arbitration (or Auctioning) Network." The purpose of OCEAN is to provide a world-wide automated commodities market for the on-demand purchase and remote usage of computing resources (such as CPU cycles, memory, disk space, network bandwidth, access to particular remote resources, etc.) by distributed applications and/or mobile agents. The strategy of OCEAN is geared towards rapid growth: The design is intended to make it as easy as possible for users to deploy OCEAN server nodes and to develop and run OCEAN applications at will, with the application users paying the server providers incrementally for resources utilized. OCEAN is intended to exhibit the following key design characteristics, not necessarily in order of importance:

1. **Open.** Every aspect of OCEAN operation is based on public, open, freely reimplementable standards, APIs, formats, and protocols. All OCEAN software comes with free source code, as a reference implementation of the standard. Anyone is free to modify or reimplement the OCEAN standards as long as they conform to its specification. Also, anyone can extend the standards, with approval from the OCEAN governing body. Non-approved modifications of the standard would constitute a violation of copyright. (This restriction is to prevent certain corporations with monopolistic tendencies from subverting and taking over the OCEAN standard, like they tried to do to Java.)

2. **Profitable.** Any entities (whether companies or individuals) who are involved in helping build the OCEAN infrastructure and providing servers, services, and applications for it should be able to profit financially (and not just via altruistic satisfaction) from doing so. This kind of true economic incentive system seems to be a prerequisite for rapid growth of the technology, its democratization, and the emergence of new industries based upon it.

3. **Portable.** OCEAN server software should run on a wide variety of host platforms so that it can be quickly deployed on the largest possible numbers of existing machines.

4. **Secure.** The OCEAN infrastructure should provide as much security for its providers and users and as much accountability for transactions as is feasible technically achievable.

5. **Efficient.** The OCEAN infrastructure should put as little overhead as possible in the way of the basic task - getting distributed computations done - and meanwhile it should allocate resources in a way that maximizes the overall economic efficiency of the resource allocation as much as possible.

6. **Scalable.** OCEAN servers should be able to be deployed on arbitrarily many hosts, and support arbitrarily high levels of usage, while degrading in performance as little as possible as the scale of the network increases.

7. **Easily deployable.** It should be trivially easy for anyone with a computer to deploy an OCEAN node. The intended standard of triviality is that any high school or college student with a PC and a credit card should be able to deploy an OCEAN node as easily as today they deploy Napster clients. With a modest amount of network administration work, it should also be relatively easy to deploy OCEAN nodes even behind firewalls and within private IP address spaces.

8. **Configurable.** With only a little extra effort on the provider's part, OCEAN server nodes should be able to be flexibly configured in a variety of ways, for example to be available only at certain times of day, or to set a certain pricing policy, or to provide access to special types of computational resources.

9. **Unobtrusive.** OCEAN server software should coexist unobtrusively with other programs running on the user's machines. So, for example, it should run at lower priority (or with higher "niceness" on Unix) than interactive user processes. It should (if possible) never hog all of the memory or other resources of a machine.

10. **Easily programmable.** It should be very easy to write applications for OCEAN, so that, for example, any programmer fluent in Java can download the API libraries, read a reasonably short and simple API document, and be able right away to start writing distributed OCEAN applications that use a variety of nodes to perform a computation.

11. **Monitorable.** It should ideally be easy for human users and providers to monitor the state and interesting statistics of both their own nodes, and the OCEAN market as a whole.

12. **Automatic.** The fundamental operation of an application deployer launching an OCEAN application, the application purchasing the resources it needs on behalf of its deployer, spawning child tasks on remote server nodes, performing its computation, and the resulting transfer of funds from the buyer to the seller (along with collection of any transaction fees by the market operator), should not require any human intervention.

13. **Dynamic.** Of course the set of available resources, the set of outstanding resource requests, and the set of jobs running on the OCEAN will all be constantly changing, and the system should adapt.

14. **Robust.** Insofar as possible, the system should be able to tolerate large fractions of its nodes going down or becoming disconnected from each other due to network failures.

Here is a list of overall system features of the OCEAN infrastructure. Key to feature status: "Required" - A core requirement of OCEAN; must be present in anything worthy of even being called a prototype implementation of OCEAN. “Planned” - A feature that is planned for inclusion in the initial beta release. "Future" - A feature that is desired but relegated to future development.
Overall Market Features

- Distributed computing resources should be bought and sold using real money - planned
- Anyone with a credit card should be able to open an OCEAN account and automatically incur monthly credits/debits as per the transactions they conduct - planned
- All account information should be maintained in a secure database - planned
- Resources and requests from all over the world should be tied together in a single commodities market (the OCEAN proper). - planned
- Despite its worldwide reach, the performance of market operations should nevertheless be scalable with the number of participating entities - planned
- Requests for resources should be matched with the lowest-priced, relatively nearby, available server that meets the request's requirements. - planned
- All participants in the OCEAN market should be able to monitor the market price history of various categories of resources - planned
- Participants or their agents should be able to be alerted when certain market conditions occur - future
- Available servers should be matched up with the highest-paying, relatively nearby, requests that meet the server's requirements. - planned
- There should be support for reselling and sub-letting of purchased resources. - future
- There should be support for commodity futures contracts (reserving resources to be provided at a certain time). - future
- There should be support for options and other derivative instruments - future

Features for the Market Operator

- Market operator should be able to verify the authenticity of user account information. - planned
- Market operator should be able to be compensated via a transaction fee for transactions that take place on the OCEAN. - planned
- Market operator should be able to prove (if true) that a transaction was contracted. - planned
- The process of matching buyers with sellers (constrained brokering, a research area that co-PI Helal has worked in) should be done in a distributed, peer-to-peer fashion so that the market operator does not need to provide all the resources needed to do it centrally. - planned

Features for the Server Provider (Resource Seller)

- The server software should be very easy to set up and deploy on a wide variety of host platforms – required
- Total operation of a server should be able to be programmed via an API or a scripting language, not requiring any human intervention for sophisticated control (e.g., dynamic adjustment of prices). - planned
- The provider should have full control over the operation of the server software - be able to adjust its priority, pause it, kill it - planned
- Multiple resources should be able to be sold in bundles together - planned
- Multiple resource bundles should be able to be sold under a single seller account. - planned
- The seller should be able to set minimum prices for use of resources and adjust these dynamically - planned
- The seller should be able to learn the market prices of similar packages of resources, as a guide to setting prices - future
- The provider should be able to deploy OCEAN nodes and sell resources even behind firewalls and inside private IP address spaces. - planned
- The seller should be able to set up a schedule for resource availability - future
- A single server host should allow multiple users to each run multiple OCEAN nodes (each perhaps providing access to a different subset of the machine's resources). - planned
- A single OCEAN server node should be able to run multiple simultaneous jobs for different clients - planned
- A node should be able to limit the resources used by client tasks - future
- The security of a provider's host, data files, private keys, and network should not be compromised by the running of OCEAN client tasks on its servers. - planned
- Seller should be able to prove (if true) that a buyer contracted to purchase resources. - planned
- Seller should be able to prove (if true) that contracted resources were actually provided to a buyer. - future
- The overhead of operation of the OCEAN infrastructure should not be overly demanding on the provider's computation resources - planned

Features for the Application Developer

- The developer should be able to write OCEAN applications in Java. - planned
• The application itself should be able to automatically (programmatically) purchase resources from the OCEAN on the
  launcher's behalf. - required
• An application should be able to specify in detail the type and quality of computational resources required at the time that it
  requests the resources. - planned
• A single OCEAN server node should be able to run multiple simultaneous threads or tasks within a single job. - planned
• A task should be able to securely migrate its state to another host in case the host it is running on announces that it will soon
  become unavailable - planned
• A task should be able to securely spawn child tasks to execute simultaneously on the same or other nodes - planned
• Fielded tasks should be able to safely sign digital documents on behalf of their deployer - planned
• A task should be able to purchase additional resources as needed. - planned
• Barring network failures, tasks should be able to communicate securely with any other tasks that are part of the same job (or
  even other jobs) - planned
• The developer should be able to write OCEAN applications in arbitrary languages - future
• Applications should have a way to ensure the quality or reputation of service providers - future
• Applications should be secure from other applications running on the same OCEAN node - planned
• Applications should be secure from theft of their code by the provider - future

Features for the Application Deployer/Launcher (Resource Buyer)
• Anyone with a credit card number and a software license to run an OCEAN application should be able to set up an OCEAN
  account charging to that card and run that application under that account. - planned
• Users of OCEAN applications should be able to set a cap on the amount they will pay (either in toto or per unit of resources)
  either to run a single job, or for all jobs they contract in a given time period. - planned
• Buyers should be able to monitor the resources consumed by a given job, or all jobs - planned
• Buyer should be able to prove (if true) that a provider contracted to provide resources. - planned
• Buyer should be able to prove (if true) that a provider failed to provide contracted resources. - future
• Buyers should be able to prove (if true) that they did not contract to purchase resources. - planned
• Buyers should be secure from theft of their data by providers - future
• Buyers should be secure from theft of their private keys by providers - planned

OCEAN System Architecture
High-level documentation of the structure of how OCEAN is broken into components, and their interactions. (Draft
of 6/7/01 by Michael Frank.) We begin with some system-level block diagrams. The following diagram is to
illustrate the different kinds of geographically and administratively dispersed units (nodes) of the OCEAN network
and give some idea of how they are interconnected.
We see several types of nodes here:

- **Central Accounting Server**: There are only a small number of these nodes, and they are operated by the administrators of the entire OCEAN network. The purpose of these nodes is to maintain OCEAN account information (e.g., balances, transaction records) in a secure location that is physically controlled by a trusted party (the OCEAN administrators). It also provides a connection point between the OCEAN network and real-world financial networks. It may also publish consolidated information about market activity and prices. In exchange for operating the server, which provides a useful service, the administrator of the OCEAN (a company started for this purpose) may collect a small fee on each transaction that is conducted.

- **OCEAN Server**: These are computation server nodes that are set up by service providers (from individuals to large organizations) to sell dynamic distributed computing resources to OCEAN users. There may be millions of them simultaneously present on the OCEAN at any given time. When discussing server nodes, we should distinguish the machines that advertise services and perform contract negotiation from the machines that actually run the computation - these functions might be managed separately.

- **Application Launchpad**: These nodes initiate distributed jobs or mobile agents which run on the OCEAN. They are operated by buyers of OCEAN resources (which again may be individuals or organizations). There may be millions of them present on the OCEAN at a time. Again, we should distinguish between the node that requests resources and negotiates a contract for them, and the node that actually sends the task to be remotely executed - these could potentially be distinct in some cases.

- **Firewall/gateway OCEAN node**: These nodes would be operated by the administrators of firewall machines or gateway machines sitting between the public Internet and a firewall-protected intranet or a private IP address space. (Often the resulting intranets further contain private sub-nets.) The role of these firewall OCEAN nodes is simply to provide a means for routing communications to nodes that may reside within private spaces. Essentially they act as proxies. These nodes provide an alternative to other approaches for getting through firewalls based on tunneling (as with ssh) or network address translation. The advantage of using an OCEAN node instead is that the firewall administrator can enable users within his organization to set up new OCEAN nodes at will without having to bother the administrator to reconfigure the address translation services on his firewall. The OCEAN communication system takes care of routing messages to nodes behind firewalls.

- **Auction service**: Actually, in the current design, all OCEAN nodes, by default, host public auctions which share information in a peer-to-peer fashion. However, some administrators may deploy nodes specifically for the purpose of auctioning. The auctioning protocols should be designed so that nodes which perform a public auctioning service successfully can be fairly compensated.

We now describe how application tasks are embedded within various layers of software on OCEAN hosts. The current architecture is based on Java, for ease of portability. At each level there may be more than one instance of the lower-level entity contained within the higher-level entity, but often there may just be a one-to-one mapping between the entities at different levels.

- At the top level is an IP-accessible computer, a networking node with one or more processors, network interfaces, memory, disk storage, and perhaps some specialized peripherals or access to local databases. Under either user or system accounts on this machine will run one or more instances of a Java Virtual Machine which can access local resources of interest.

- Within a JavaVM, there may be one or more **OCEAN nodes** (instances of a Java class OCEANNode). Usually there will be only one. The OCEAN node object itself is the primary fundamental unit of the OCEAN network. All entities on the OCEAN network are OCEANNodes of one type or another, though some nodes may be specialized to perform only certain kinds of duties. Each OCEAN node is addressed via an association with a specific IP port number on its host.

- Within an OCEANNode that is deployed as a computation server, there may be one or more **JTRONs**. The word “JTRON” is an acronym for “Job’s Tasks Running on a Node.” A **job** is a set of tasks to support the running of a specific instance of an OCEAN application. All of the tasks making up a JTRON are agents of the same responsible buyer, and their resource usage falls under the same contract. A JTRON is therefore the smallest unit associated with resource consumption. A JTRON is associated with a Java ThreadGroup object that contains all the threads that are running as part of that JTRON, and that runs under a SecurityManager that implements the security restrictions appropriate to a given resource utilization contract.

- Within a JTRON may be one or more **OCEAN tasks**. A task is a unit of code migration - the equivalent of a mobile agent in mobile agent systems. Tasks can spawn other tasks on the same node or on different nodes. (If the new task is on the same node and running under the same contract as its parent, it belongs to the same JTRON.) Tasks may also pack up their state and migrate completely off of a node and onto a different node. Or they may just die without spawning anything. Tasks are represented by OCEANTask objects and are associated with a set of threads.
An OCEAN task may contain one or more Java threads to carry out its work. When a multithreaded task needs to migrate its state to another machine, it should carefully terminate all its component threads and pack up their relevant state information for communication to the new location.

**Node Software Architecture**

The below block diagram shows the different components of a typical OCEAN node’s software, and their interrelationships. It is roughly a layered view, with higher-level components tending to be towards the top of the diagram, and lower-level ones towards the bottom. Keep in mind that this is only an approximate rendering. To reduce clutter in the diagram, not all interconnections are shown.

Here are brief descriptions of the components, from bottom up:

- The **Naming** system is really just a specification for how the URIs that name OCEAN entities (nodes, JTRONs, tasks) should be mapped to the named physical entities.
- The **Communication** system uses the Naming system to pass messages and establish continuous communications channels between OCEAN entities (nodes and/or tasks). It may also provide support for concurrency abstractions such as distributed semaphores, virtual shared memory, etc.
- The **Security** system keeps track of keys, certificates, SecurityManagers, signed documents, etc. and provides security services to the communications layer and all higher layers.
- The **Peer List Update Manager (PLUM)** maintains a list of peer nodes that the current node knows about. This list is used by the Auction system to decide which nodes to pass trade proposals to, and in what order. The PLUM keeps track of information about the nodes’ quality, availability, and ability to deliver good trade proposals. It periodically updates its list automatically, by exchanging peer list information with its peers and measuring availability, latency, and bandwidth by itself using test messages.
- The distributed **Auctioning system** permits a node to announce the availability of its resources, and permits a task to request resources. Resource providers may characterize their resources and set a minimum "ask" price and contract conditions. Resource users (application launchers, or individual tasks) may request new or additional resources by submitting a bid describing the resources needed at the price offered. Both bids and asks are called *trade proposals*. Trade proposals are distributed to other auction nodes using a certain algorithm. When a proposed match is generated, it is sent back to the originating nodes’ Negotiation systems, which consider the offer and talk to each other to finalize the trade.
- The **Negotiation system** handles negotiation of resource usage agreements. The distributed Auction nodes communicate proposed matches (between resource requests and resource advertisements) to the nodes or tasks that originated the trade proposal. The originators consider the received proposals for some time period and then choose one with whom to negotiate. The negotiation systems pass a few messages back and forth and end up with a signed contract, which is archived by both sides, and by the central accounting system, and then the Job Migration system is requested to initiate use of the resources. (Though actually we plan to allow for negotiation of contracts where the resource usage begins at a future date as opposed to immediately.)
- The **Accounting system** at a local node (or within a mobile agent) keeps track of that entity’s own idea of its financial records... It also communicates critical information (transaction records, signed contracts, etc.) to the central accounting system (maintained by the global OCEAN operators), to allow real-world payments to actually be processed and archived.
However, micropayments for individual OCEAN contracts would be consolidated into larger amounts by the central accounting system before submitting them to the real financial networks which have a relatively high overhead.

- The **Node Configuration/Operation Interface** provides an API for use by the node deployer for configuring all of the important parameters of the OCEAN node component software. For example, the naming system needs to know the name of the current node, telling how it is reachable from the external internet. The security system needs to know (or generate) the node's keys and a certificate for them from a Certificate Authority. The PLUM needs to know some initial peers, how many nodes to maintain in its peer list, and what its criteria should be for expunging a node from its list. The Auctioning system needs to have caps on the hop counts and expiration dates on trade proposals it will forward, plus limits on how many old trade proposals to keep around. The Accounting system for a job-launcher's node needs to know a cap on the maximum dollar amount of transactions it will authorize (per month, say) and communicate this cap to the central system, to limit the possible fraud that may take place. An OCEAN node started to spawn a task needs to know what task to spawn.

- The **OCEAN Application Programmer's Interface** (the "real" API) provides a simple, easy-to-use set of packages, classes and methods for use by OCEAN Application Programmers, to write Tasks to run on the OCEAN. It provides a view of all of the parts of the OCEAN node software that tasks are supposed to be able to use.

- **Node Operator GUI**: This provides a way for node deployers who have simple needs to configure, operate, and monitor their nodes, whether server nodes, gateway nodes, application launcher nodes, etc. It would also allow users to monitor activity and prices on the overall OCEAN market. This uses the node configuration/operation interface.

- **Custom Node Operation & Maintenance Code**: Scripts or Java code written by node operators (or third-party providers) which uses the node configuration & operation interface to run a node in a customized way. For example, a server node may implement a particular strategy for when and how to adjust its prices.

- **Application-Specific Mobile Job/Task Code**: This is the application itself, performing a distributed and/or mobile-agent type of computation on various OCEAN nodes (server nodes and probably also the application launcher's node).

### Potential Key Underlying Technologies

This section lists some of the major existing, emerging, and proposed standards that OCEAN will be, or may be, based on. Beside the name of each standard we list the likelihood of inclusion in the early prototypes.

- **Java (definite)** - Early versions of OCEAN will rely on Java [Java] for their portability and for their support of networking APIs, code migration, database connectivity and various security features. A possibility for later development is to add support for other, platform-independent (or even platform-dependent) languages.

- **XML (definite)** - XML (eXtensible Markup Language, [XML]) and its DTDs (Document Type Definitions) are emerging as the standard framework and meta-language for defining extensible languages for communicating meaningful, structured packages of information. We will use it as the basis for most communication in OCEAN, particularly in OCEAN's languages for communicating descriptions of such complex entities as resources, usage terms, job requirements, schedules, and contracts. The extensibility of XML permits continual enhancement of the expressiveness of the languages for describing such entities, without fear of losing compatibility with application logic developed around earlier versions of the languages.

- **J2EE (possible)** - Sun's enterprise programming environment (Java 2, Enterprise Edition, [J2EE]) provides a number of features - such as support for XML processing, XML messaging, transactions, naming and directory services, and other standards such as UDDI and SOAP - which may be very useful for the implementation of the OCEAN infrastructure. However, it is currently a single-vendor solution, and therefore presents some risks to its use as a basis for development. At this point in time, we are also still considering custom implementation of the capabilities needed for OCEAN.

- **SOAP (likely)** - The Simple Object Access Protocol [SOAP] is a W3C standard for XML-based communication for message passing, object transfer, and remote procedure calls. It provides a certain degree of transparency to firewalls by communicating through the existing web server infrastructure. It also supports a wide variety of object types. We are likely to use it in our communications and code migration systems.

- **UDDI (possible)** - The Universal Description, Discovery and Integration [UDDI] initiative is a developing industry initiative spearheaded by IBM and Microsoft (among others) for internet-based service registration and service discovery. It could potentially offer a means for OCEAN nodes to discover each other. However, we already have in mind an alternative, totally distributed, peer-to-peer means for discovery that does not depend on any centralized authority, and at the moment we consider such a design to be preferable. However, we may consider integrating some degree of interoperability with UDDI into OCEAN at some later date.

- **MPI (possible)** - The standard Message Passing Interface [MPI] for parallel & distributed programming in Fortran and C (among other languages) could be a useful feature to provide to OCEAN application developers who are accustomed to using MPI's API for their application programming. Several Java wrappers for MPI already exist. However, even if we provide MPI for application developers, we are currently leaning towards basing the OCEAN infrastructure, itself, on some XML-based framework for object exchange (such as SOAP) rather than on MPI. However, this is not yet completely decided.
CORBA (possible) - Somewhat similarly to MPI, the Common Object Request Broker Architecture [CORB] provides a framework for exchanging objects cross-platform in distributed environments. Also similarly to MPI, we may encourage application developers to use CORBA if they wish, but we will probably not base the OCEAN infrastructure itself on it. CORBA is most useful when one needs to communicate cross-platform, and cross-language to an existing legacy system, but the OCEAN infrastructure itself is not a legacy system, and uses Java for its cross-platform capabilities.

JKQML (possible) - A framework [JKQML] for Java-based communication using KQML, the Knowledge Query and Manipulation Language. KQML could be useful for expressing contracts and doing negotiation, since it supports expressing semantic content more so than XML does. However, at present we are planning to use XML for these purposes instead. There is an XML-based language, FLBC (Formal Language for Business Communication) which is designed to take the place of KQML, so we may use it instead, or develop our own language. We are still looking at these issues, but at the moment we are also designing our own simple languages for prototyping purposes.

ATP (possible) - The Agent Transfer Protocol [ATP]. This is a language for mobile agent transport that was defined as part of the IBM Aglets project. However, ATP is based on KQML, and we may prefer to use FLBC or our own XML-based language rather than KQML. We will likely end up designing something similar to ATP, however, and so we would do well to borrow ideas from it.

FLBC (possible) - The Formal Language for Business Communication [FLBC] is an XML-based agent communication language, having more semantic depth than XML itself. We are considering it as the basis for the OCEAN languages for resource advertising, requesting, bidding, and contracting.

WSDL (possible) - The Web Services Description Language [WSDL] is an XML-based language for describing internet-based services to a UDDI registry. If we use UDDI we will likely use WSDL, but if we don’t use UDDI, we may have to decide between WSDL, something based on FLBC, or some other custom language for describing OCEAN services.

WSFL (possible) - The Web Services Flow Language [Smi01] is IBM’s proposal for a standard language for defining web-based workflow. This could be important for OCEAN applications to use to manage the flow of application-specific data between widely-distributed nodes in the OCEAN network. It may also be useful as a basis for coordinating distributed operations of the OCEAN infrastructure itself, such as service discovery, auctioning, and ratings systems. This needs to be looked at in more detail.

PVM (unlikely) - Parallel Virtual Machine [PVM] is an old communication and remote process-spawning standard for distributed systems, that has largely been superseded by MPI. Anyway it is not very applicable since it was designed for a heterogeneous environment, rather than the more homogeneous environment provided by Java.

C.I. Statement of Work to be Undertaken

C.I.a. Objectives and Significance

Here are the major objectives of the work to be covered by this proposal, by the planned year and quarter of completion of each objective. At the end of each year, the significance of that year’s work is noted.

Year 0 (Jul.-Dec. 2001) (This is unfunded work to be done before start of proposed work period):
3rd quarter (Jul.-Sep.):
- Complete present work on detailed specification of OCEAN system architecture and all protocols & interfaces.
- Continue recruitment of team of initial collaborators on OCEAN application development.
- Research e-commerce platforms and acquire functioning merchant accounts for testing of financial components.

4th quarter (Oct.-Dec.):
- Complete present development of first alpha release of OCEAN infrastructure software.
- Release, publish, and advertise initial version of OCEAN protocols for public & industry review.

Significance of Year 0 work: Even before the start of the proposed funded work period, the OCEAN design will be fully fleshed out and published for review by the distributed computing & internet infrastructure community.

Year 1 (Jan.-Dec. 2002):
1st quarter (Jan.-Mar.):
- Implement several major applications for major OCEAN alpha-testing, in work involving all senior personnel as well as many informal collaborators.
- Deploy alpha OCEAN infrastructure widely around UF, and exercise it with small test programs. Fix minor bugs.

2nd quarter (Apr.-Jun.):
- Alpha testing of major OCEAN applications, using play money.
- Detailed experimental characterization of OCEAN performance characteristics on major test applications.

3rd quarter (Jul.-Sep.):
- Any necessary redesign of OCEAN architecture protocols or architecture, to incorporate lessons from alpha testing and/or important new standards.
• Refinement of applications code.

4th quarter (Oct.-Dec.):
• Any necessary reimplementation, bug fixes, new features.
• First public beta release of OCEAN infrastructure software and example applications.
• Deployment of publicly available OCEAN servers on campus.
• Publication of results of initial experiments on infrastructure & applications.

Significance of Year 1 work: At the end of year 1, a complete working beta version of the OCEAN system will be publicly available. At this point, due to OCEAN's scalable, peer-to-peer, profit-generating design, we anticipate that OCEAN should start to become more and more widespread, along an exponential growth curve.

Year 2 (Jan.-Dec. 2003):
1st quarter (Jan.-Mar.):
• Incorporate feedback from beta testing into OCEAN design & implementation.
2nd quarter (Apr.-Jun.):
• Complete "final initial" version of OCEAN reference implementation.
• Submit OCEAN standards for consideration by a recognized standards body (e.g. W3C).
3rd quarter (Jul.-Sep.):
• Re-test "final initial" versions of OCEAN & major test applications.
• Present final results of OCEAN work in academic publications.
4th quarter (Oct.-Dec.):
• Further promote the widespread adoption of OCEAN standards, and perhaps foster the formation of a few startup companies to operate OCEAN services on a commercial basis. Also, get a head start on year 3 work.

Significance of year 2 work: By the end of year 2, the reference implementation of OCEAN should be a stable, reliable, high-performance platform for market-based distributed computing, and should be highly popular and undergoing widespread adoption. We begin to see a large impact (e.g., significant new projects based on OCEAN springing up).

Year 3 (Jan.-Dec. 2004):
1st quarter (Jan.-Mar.):
• Design "2nd-generation" version of OCEAN: Add new significant but difficult features, such as support for programming languages other than Java (e.g. C, C++, C#, etc.), additional standard distributed computing interfaces (CORBA, shared memory models, etc.), or support for enhanced security measures such as computation certificates [Mic00].
2nd quarter (Apr.-Jun.):
• Implement & release 2nd-generation version of OCEAN.
3rd quarter (Jul.-Sep.):
• Further testing & enhancement of 2nd-gen release.
4th quarter (Oct.-Dec.):
• Publish results of 2nd-gen experiments.

Significance of year 3 work: Due to the important new features that are added, the range of applicability of the OCEAN technology is further increased, and thereby starts to thoroughly penetrate more and more of the existing markets for computation (and also to create more and more new markets).

C.I.b. Relation to Longer-Term Goals

The proposed work directly addresses the long-term goals of the OCEAN project. The proposed schedule is therefore rather ambitious, but we believe this is necessary due to the wide variety of similar projects that are beginning to emerge (see next section). We would like to see that this technology (of market-based distributed computation) soon becomes widely available in a form that is as flexible as possible, so that users are not faced with a possibly burdensome migration from a less flexible platform to a better one as the technology evolves. In the OCEAN project, we wish to short-cut the whole process, and provide the best solution we can envision, with as little delay as possible.

C.I.c. Relation to State of Field and to Work in Progress Elsewhere

State of the Field

There are currently millions of human users whose computing resources are connected via the Internet. It has been observed by numerous sources [Dist.He98,0cea.Pat99.Popc.Popu,Proc.Reg98] that these resources are frequently idle. This observation has led to the emergence of various projects that harness these resources for some benefit.
Some projects that have received much attention are the SETI@home project, and the various initiatives led by Distributed.net. Both of these projects are maintained on a strictly voluntary basis. In the case of SETI@home, users donate idle CPU cycles for the analysis of data collected by the Arecibo radio telescope in the search for intelligent extra-terrestrial life [SETI]. Distributed.net has used the donated CPU time of many users to solve computationally intense problems, such as cracking encryption codes [Dist].

However, there are some recently formed commercial ventures and research projects whose goal is to harness idle CPU cycles for the economic benefit of their human users. Such projects are referred to as computational markets.

A computational market is a marketplace in which a CPU cycle is the traded commodity. The major components of a computational market are the users, the computational resources associated with them, and the underlying market mechanisms that facilitate trade. Note that the users do not need to be human; in many instances, a user is a programmed agent acting on behalf of a human user [Dre88,Mil88]. In general, human users in computational markets are buyers or sellers. The buyers have a computation that needs to be performed, and the sellers have access to the idle resources that can execute the computation in question. The buyer and seller(s) will agree on some sort of payment mechanism, the program for the computation will be transported to the seller(s), and the computation will be performed. The computational market’s role in all of this is to provide an environment in which these interactions may take place.

**History of Computational Markets**

Possibly the first computational market was described by Sutherland [Sut68]. He demonstrated how auction methods were used to allocate computer time to users of the PDP-1 at the Aiken Computation Laboratory at Harvard. The users were allowed to use the computer for some time period. The hours of the day would be divided into regular time slots. Users were assigned different amounts of currency based on their project’s importance. Then, the users would submit bids for time slots. For a given time slot, the user who submitted the highest bid would have use of the computer. By using the auction methods described, a monetary value was associated with computation time, which is a basic feature of computational markets. This paper is important because it may be the first application of economic principles to the problem of computer resource allocation.

Another significant research project regarding computational markets was implemented by Shoch and Hupp [Sho82]. Their paper describes a “worm”, characterized by a couple of traits. First, it was parallelizable—it could be broken into many discrete segments, each running on a different host. Also, the worm was able to sniff out idle machines on a network. Although this project was not concerned with the economics of computation, it still represented a significant step toward fully functional computational markets. This observation lies in the fact that the worm’s traits are important features of programs used in computational markets. For one, idle machines in a network must be located. Also, users of programs that can be parallelized stand to benefit most from computational markets, because they achieve the highest utilization of available resources.

The next major research effort was the Agoric Papers [Dre88,Mil88]. Although there is no implementation associated with these publications, they established important groundwork for modeling distributed computation in economic terms. Although Sutherland’s work [Sut68] was important for using economic principles to solve resource allocation problems, it did not address the problem in distributed systems. Drexler and Miller approached this problem by describing auction mechanisms for allocating distributed resources. In one of their solutions, a seller auctions off its resources to multiple competing buyers. Escalator algorithms are used such that buyers only submit an initial bid. Each bid increases at a constant rate until the highest bid is determined. The highest bidder obtains the seller’s resources [Dre88]. Drexler and Miller continued, describing initial market strategies, and how they achieve stable pricing mechanisms. Later work [Wal92] has proven Drexler and Miller’s algorithms to be effective for achieving this goal. The Agoric Papers made two important contributions to computational markets. First, they provided market-based mechanisms to allocate distributed resources. And second, they addressed how stable pricing mechanisms can be achieved in computational markets. These are essential for a successful computational economy.

The Spawn project [Wal92] claimed to be the first implementation of a market-based computational system. Indeed, its implementation incorporated the most fundamental aspects of a computational market. Spawn’s principal features involved an approach to computational resource management based on economic principles, and the exploitation of idle time. This included assigning priorities based on monetary funding units, and the use of price information to control adaptive expansion. The resource allocation methods, specifically processor scheduling, were largely based on Drexler and Miller’s work. Spawn’s methods to harness idle processors are similar to those employed by the Condor system [Cond]. Perhaps the most important use of Spawn was in examining the price dynamics of a computational economy. This is relevant because the market must exhibit stable
pricing mechanisms. And in fact, the experiments carried out with the system confirm that computational economies that exhibit stable pricing and growth properties can be designed.

With the advent of the Java programming language, some of the most difficult aspects of computational markets were eliminated. Java makes the problem of migrating executable programs between platforms trivial. Of course, this does not benefit the programs written in other non-portable languages.

The Internet and World Wide Web make it possible for potentially millions of human users to participate in computational markets. With this dramatic increase in the number of users, the problems of scalability and stable pricing mechanisms are magnified.

The POPCORN [POPC,Reg98] project was the first computational market to focus on the benefits of using Java programs. The most interesting feature of POPCORN was its notion of the traded commodity. Regev and Nisan [Reg98] noted that the idea of trading in CPU time is not exact since it is processor dependent. Due to the fact that they were concerned with Java programs, they established the basic tradable good as a “JOP”, or Java Operation. Each program takes some number of JOPs to execute, and the price of executing a program is proportional to its number of JOPs. This convention made price estimation simpler, but only applied to Java programs.

Current Computational Market Implementations

Examples of commercial ventures of note are the (now-defunct) Popular Power system [Popu] and the ProcessTree Network [Proc]. Both of these endeavors promise some form of economic benefit to their human users. Although it is not yet a commercial venture, OCEAN, currently under development in the Computer & Information Science & Engineering Department of the University of Florida, is another example of such a system.

Both Popular Power and ProcessTree provide a facility for users to download specialized client software. The client software allows the user to become a seller of resources. When the user is connected to the Internet, the client software will download a buyer’s computation, executing it when the seller’s resources become idle [Popu,Proc].

Each of the commercial ventures employs similar mechanisms to disburse payments to sellers. They have noted [Popu,Proc] that one work unit does not have a significant cash value associated with it. Therefore payment will accrue over time in an account. When the account reaches a minimal value, payment can be disbursed in the form of a check, or as discounts to various online services. We propose a similar payment mechanism with the OCEAN.

However, we believe that users prefer to receive cash rather than discounts. Also, disseminating many small checks to users is impractical. Therefore we propose that payments be made through an online account with Paypal [Payp] or a similar service, or through credits to credit card based merchant accounts.

Summary of Work in Progress Elsewhere:

Here is a sampling of selected projects that are roughly similar to OCEAN (shareable infrastructures for distributed computing, particularly market-based ones), with a comparison between their capabilities, and OCEAN’s. We divide these roughly into non-market-based systems, and market-based ones. We see the market-based aspect of OCEAN (in particular, the use of real money for compensation) as the critical prerequisite for any distributed computing infrastructure ever becoming extremely widespread and popular for general use, and therefore we view only the market-based efforts as presenting serious potential competition. However, even among the market-based systems, none of the competing systems we have examined so far offer anything close to the flexibility and openness that we envision for the OCEAN market. Therefore in developing OCEAN we are providing a significant public service.

Not Market-Based:

GriPhyN - The Grid Physics Network [GriP], based in UF’s Physics Department, is a distributed computing infrastructure developed for physicists - its emphasis is not so much on publicly-accessible servers for general applications. It also seems to be a somewhat more structured, less organic architecture. Therefore, in our view it may lack some of the growth potential of OCEAN - although it may well outperform OCEAN on the sorts of applications and user markets for which it was designed. Nevertheless, we see physics applications as an interesting potential application area for OCEAN. We are in discussions with GriPhyN participants about collaborations to (for example) port GryPhyN applications to OCEAN, and/or use OCEAN’s auctioning system as a resource costing & allocation facility for GriPhyN, and/or interchange other design ideas between the two projects.

Globus - Globus [Glob] is a grid computing initiative associated with the Distributed.net project.

Condor - [Cond] An old volunteer-based distributed computing system.

Legion - [Legi] Another old volunteer-based distributed computing system.

Beowulf - [Beow] Software for turning clusters into virtual supercomputers.

OceanStore - [OcSt] Oriented towards distributed document storage, not computation.
SETI@Home - This Berkeley-run project [SETI] differs from OCEAN in that it is completely application-specific (aimed at analysis of extraterrestrial radio signals) and volunteer-based. However, it has in common with OCEAN the feature that its computation server can be freely and easily downloaded and installed by anyone. SETI@Home grew enormously rapidly; we believe that by offering users a financial reward for the use of their computational resources, OCEAN can grow even more rapidly, and provide a pre-existing platform which future developers of similar applications can utilize.
POPCORN - [POPC] Discussed in text above.
IBM Aglets - [Agle] A mobile agent framework
Sun Grid Engine - [SunG] For sharing computational resources within an enterprise.
XtremWeb - [Xtre] A non-market based system with good information about their technology.

Market-Based:

- PopularPower - This venture [Popu] was co-founded by Nelson Minar, who was one of the original participants in the OCEAN project as an MIT Media Lab student back in 1997. Unfortunately, PopularPower went bankrupt due to squabbling between its VCs. Their technical approach was mostly sound, but it lacked OCEAN's feature of open application development and deployment, which severely limited its growth potential.
- Economy Grid - [EcoG] The academic foundation for Compute Power Market (below).
- ProcessTree - [Proc] Based on a somewhat restrictive tree-structured architecture for the organization of distributed computations.
- Agorics - [Agor] Discussed in above text. A small company that apparently sells market-based computing products - but it seems the markets are within the enterprise, not public.
- United Devices - [UniD] With its heritage in Distributed.Net, SETI@Home, and the RSA challenge, this high-powered group of developers has provided good security in a C/C++ based (rather than Java) platform. However, their resource allocation model is based on volunteerism, sponsorships, and partnerships, rather than being an open market infrastructure that anyone can buy and sell resources on. (Instead, United Devices owns the market.)
- Porivo - [Pori] Not an open architecture; does not provide much public details. They do use a distributed peer-to-peer network, but their incentive system is based on a lottery for prizes rather than direct payment. They consolidate resources for resale to large customers. This is a much less democratic system than OCEAN.
- Entropia - [Entr] Features a charity-based incentive system for providers (although occasional sponsor clients pay Entropia for use of its volunteer network).
- Compute Power Market - [CPM] This project is more similar in spirit than most to OCEAN, and we are talking with them about collaboration on the development of pricing mechanisms.
- Ubero - [Uber] - This is not an open platform that anyone can develop for. It buys resources and then resells them to big customers who have contracted as customers of Ubero. It also does not allow communication between distributed client tasks.
- Parabon - [Para] This Java-based system is somewhat less flexible than OCEAN in that it has a centralized architecture, does not allow IPC (inter-process communication) between the tasks making up a job, returns results through a central server, and does not support operation through firewalls. However, it presents a nice, easy-to-use GUI.
- BaseOne - [Base] Software for resource sharing within the enterprise.

C.II. General Plan of Work

C.II.a. Activities to be Undertaken

Infrastructure Work

Michael Frank (PI) and Abdelsalam Helal (co-PI) will have primary responsibility for supervising design and implementation of the infrastructure. Dr. Frank, who already supervises a large group of students involved in this project, will be the lead designer on OCEAN itself, while Dr. Helal will apply his expertise in Internet Computing (in both mobile agent systems and e-services workflow management systems) to help refine the OCEAN infrastructure design in a variety of ways, for example to be compatible with the mobile-agent and workflow management infrastructures that his own group of students is already designing. Furthermore, another member of our department, Dr. Meera Sitharam (informal collaborator) has done some work [Par97,Par98,Par00] on noncooperative game theory as a market model for network resource allocation, and on the effects of pricing, and is willing to serve as a local source of expertise in that area, as we strive to improve our market & pricing mechanisms.
In addition, the infrastructure design process will continually incorporate input from the other senior personnel as to their needs and requirements as they proceed to try to implement significant applications that also further their own research goals on top of the OCEAN infrastructure.

Furthermore, we have begun informal collaborations with other researchers working on related infrastructure projects; continuing interactions between the projects will likely influence the design of our OCEAN, as it incorporates ideas from other projects or is designed to be compatible with them. For example, we are in discussions with Paul Avery of the UF Physics department to explore a collaboration between OCEAN and the GryPhyN (Grid Physics Network) project [GriP], in which OCEAN’s auctioning system could, for example, potentially be used as part of a costing and resource allocation system for GryPhyN. Similarly, Rajkumar Buyya of Monash University, Australia has expressed interest in collaboration between OCEAN his own Compute Power Market project [CPM], for example to use OCEAN’s auctioning system as a pricing mechanism for his system. Interactions with both of these projects will likely heavily influence the eventual design of the OCEAN infrastructure.

Applications Work

This applications work is important to ensure that OCEAN is truly a strategically beneficial technology, an infrastructure that facilitates useful research and development of all manner of applications. Here, we break down the applications work by the senior personnel and/or informal collaborators who will be involved in it.

Joachim Hammer (co-PI)

Dr. Hammer is interested in developing distributed database systems, and possibly mobile-agent systems that would use them, such as mobile web crawlers, as applications on top of the OCEAN system.

Jorg Peters (co-PI)

**Computational Geometry Application:**

**Database of vertex neighborhoods in a 3D space partition.**

A vertex in a 3D space partition can have four neighbors or five neighbors in two different configurations or six neighbors in twelve combinatorially unisomorphic configurations. Enumerating all such connectivity-distinct configurations and pruning out all isomorphic cases is compute-intensive. The result, even though exponentially growing with the number of neighbors, can be stored in compressed form and made accessible as a data base entry. At present, determining the entries for n=15 neighbors requires overnight usage of the best computational facilities in the department.

The characterization (enumeration) of all possible combinatorial neighborhoods of a point with up to, say 30 neighbors, in a volumetric partition of space is of interest when computing fields, say smoothly varying 3D fields, from isolated and scattered observations. Realistically, the partition of space is irregular and so is the layout of sources and sinks and distribution of cells, say in the context of the finite element method.

We propose to use OCEAN to investigate both variants of the Joy, McCracken 98 approach (generalizing the Catmull-Clark subdivision scheme) and a more efficient dual subdivision algorithm. We will leverage existing graph isomorphism algorithms and combine this with eigenanalysis packages. Finally some 3D fields will be visualized, say to simulate particle distribution in a closed environment.

**Computer Graphics Application: Web-based rendering farm for realistic scenes.**

Currently high quality animation is achieved by dedicated rendering farms that take the graphics scene through various filters (include ray-tracing, shadows, etc.) Public tools are available but computational resources are still expensive. The OCEAN network would replace this dedicated computational environment to allow realistic rendering of a small movie on a low budget.

Mark Schmalz (co-PI) — **Image Processing & Image Algebra Applications**

Dr. Mark Schmalz (Co-PI) specializes in image and data compression, image and signal processing (ISP), and computer vision (CV). In the proposed OCEAN project, Dr. Schmalz would apply his expertise in adaptive systems design, network computing, ISP/CV, and performance analysis to (a) develop ISP/CV applications for the OCEAN system, and (b) measure and analyze performance a wide variety of image-related tools and applications on OCEAN. Such applications are of key importance to rapidly expanding areas of technology including Internet video, telemedicine, and distance learning. Efficient porting of these applications to OCEAN would be facilitated by an image algebra application programming interface. The proposed effort leverages a decade of experience at UF in porting image algebra to a wide variety of sequential and parallel workstations, massively parallel processors, digital signal processors, and adaptive or reconfigurable computing systems [Rit01,Sch00]. After porting image algebra system to the proposed OCEAN platform, Dr. Schmalz further proposes to implement, test, analyze, and evaluate a wide variety of proven ISP/CV applications written in image algebra, such as object recognition, image and signal compression, automated error analysis and profiling routines, and medical image processing algorithms [Sch01]. This diverse suite of applications is both I/O- and compute-intensive, and provides a flexible yet highly specifiable suite of performance measurement and analysis benchmarks for the OCEAN platform. The proposed benchmarks would extend more common benchmarking program suites such as SPEC or iCOMP benchmarks to include multi-level data abstraction with I/O- and compute-challenged behavior designed specifically for the OCEAN paradigm. Performance would be analyzed in terms of computational error, as well as space and time complexity [Sch97,Sch00].

Paul Fishwick (informal collaborator) — **Interactive ray-traced VRML rendering**
The Virtual Reality Modeling Language (VRML) is the ISO standard for 3D graphics on the web, and while it is widely used, the graphics display suffers because of client-side rendering, which depends on which architecture is used. VRML, and its successor X3D, employ browser plugin software, which renders scenes using Gouraud and Phong shading algorithms. Even with fast client graphics acceleration, it is not currently possible to render scenes, with shadows and true reflection, in real-time. With the advent of the Internet2, bandwidth speed is sufficient to accommodate interactive video. Combined with fast, ray-traced rendering capabilities, it is possible to render VRML frames using ray-tracing. The two key challenges are 1) to ensure sufficient bandwidth and video compression to support real-time frame rates, and 2) to ensure ray-traced renderings at reasonable frame rates on the client machine, anywhere from 15 to 60 frames per second. We propose developing a distributed OCEAN-based rendering system to help achieve these goals.

**Christopher Carothers** (informal collaborator, Rensselaer Polytechnic Institute) - **Distributed simulators**

Dr. Carothers develops distributed architectures for simulators for complex systems such as parallel computers, networks, and mobile control environments. He is interested in using OCEAN as a test platform for deploying his distributed simulations. Dr. Carothers and Dr. Frank also have a collaboration on using reversible computing techniques for synchronization of speculative executions in parallel simulators.

**Wei Shyy** (AeMES dept., informal collaborator) and

**Marc Garbey** (U. Houston and Université Claude Bernard Lyon 1, France, informal collaborator)

**Distributed computational fluid dynamics modeling**

Drs. Shyy and Garbey work on distributed algorithms for performing finite-element simulations for computational fluid dynamics. They have expressed interest in collaborating to port their algorithms to the OCEAN platform.

### C.II.b. Experimental Methods and Procedures

The design of the OCEAN standards and architecture and the development of its prototype will involve standard software engineering procedures, with modular design, testing of components, and integration testing of the complete system. OCEAN nodes will be deployed widely on computing resources throughout the department and university, and the audit mechanisms tested to ensure their stable operation. However, the infrastructure cannot be thoroughly tested and benchmarked without a wide variety of user applications; that is the reason for the application work described above. Standard methods of performance characterization will be applied to determine how efficiently the OCEAN network runs these applications, and furthermore how well the performance scales with increasing numbers of processors. The results will be compared to the performance of running comparable applications on pre-existing platforms. The market mechanisms will be tested initially using accounts of “play money” assigned to the various application development groups; multiple applications will be run simultaneously on the OCEAN, and the market mechanisms will be examined for features of price stability, fairness, etc. Results of all OCEAN experiments will be recorded methodically for use in research reports.

### C.II.c. Plans for Preservation, Documentation, and Sharing of Research Products

All standards, design documents, source code, application code, experimental data, and results of the OCEAN project will be made permanently publicly available via the web by the UF CISE department. In addition, the major elements of the design and the experimental results will be archived via journal publications. However, the goal of OCEAN is for it to become a universal standard; if this goal is achieved, some standards organization such as W3C, ANSI, ISO or other consortium eventually will take over the primary responsibility for maintaining and distributing the standard.

### C.III. Broader Impacts of Proposed Activity

#### C.III.a. Integration of Research and Education

OCEAN has already been the focus of a large number of student-semesters’ worth of student individual study projects, undergraduate senior projects, and graduate thesis research, and therefore has played a significant educational role already. (Note that none of these students have been paid to work on the project.) We anticipate that the enthusiasm displayed by students for this project will only increase once actual funding becomes available to support it. The student work on this project plays an important role in helping the students to learn about important new and emerging internet technologies such as those we are considering using in OCEAN (listed earlier). Finally, the other collaborators working in various aspects of OCEAN will have their own groups of students who will also become trained in how to do distributed programming and how to use OCEAN-like infrastructures. We believe that programming for such platforms will inevitably become an important software engineering skill sooner or later, whether or not OCEAN itself becomes the eventual standard.
C.III.b. Participation by Underrepresented Groups

Dr. Frank is active in encouraging participation by women, minorities, and other underrepresented groups in his projects. He serves in UF’s UMMP (University Minority Mentoring Program) [UMMP] and he works to help ensure diversity in the student body through his role on the CISE department’s graduate committee. This committee strives to ensure, for example, that women and students from underrepresented foreign countries are especially encouraged to attend the UF CISE graduate program. Dr. Frank has supervised a wide diversity of students on his own projects and will continue to do so in the future. The other senior personnel and collaborators on OCEAN themselves hearken from a wide variety of nations and backgrounds, and will certainly help to ensure diversity in the project as a whole.

C.III.c. Enhancements to Infrastructure for Research and Education

The explicit goal of OCEAN is to provide an extensive, scalable, easily usable infrastructure for distributed computing in all its forms. As such, it will directly facilitate an enormous variety of new, large-scale research projects that would not otherwise have been possible, projects that harness large numbers of distributed servers, or widely-wandering mobile agents. It also facilitates education in the design and use of computationally-intensive tools and technologies, since it provides an environment in which students can cheaply harness vast amounts of computational power for their computational projects for short periods of time. Many schools might not be able to afford to purchase large clusters of computing servers for students to run their big projects on – but might well be able to afford to rent the needed power for short periods. The educational applications are especially appropriate since their needs are so bursty - student projects often are run in large waves late in the semester, rather than continually over time – so school-owned resources for student projects would be underutilized.

C.III.d. Dissemination of Results

PC World magazine interviewed Dr. Frank about OCEAN recently for an upcoming article on distributed computing, so the project is already beginning to get some press. As the project develops, it will be further publicized through conference publications, announcements on relevant Usenet newsgroups, link sharing with related projects, and so forth. Third-party application developers will be contacted and encouraged to write applications for OCEAN by showing them the example applications that our project will develop.

Once the servers are available, and anyone can install them and begin to earn money by running applications for end-users, we expect that word-of-mouth will contribute significantly to further dissemination and growth of the OCEAN concepts and standards. As attention is drawn to the project, our collection of research on market-based distributed computing will be more widely seen and understood.

C.III.e. Potential Benefits to Society at Large

In transforming computational resources into a fundamentally very liquid commodity, OCEAN will have a number of extraordinary benefits for society at large: (1) It will make computation cheaper for everybody, as a result of eliminating the waste of unused cycles caused by bursty patterns of individual usage. (2) At the same time, it will aid the computer industry by making computers themselves more intrinsically valuable – if a user can recoup the cost of a computer in a short period of time by selling its unused cycles on the OCEAN, rather than wasting them, he will be willing to pay a higher price for the machine to begin with. (3) It will enable new kinds of computing applications that would not otherwise have been feasible, most especially, applications that require large amounts of parallel power for relatively short periods of time, and applications that require mobile agents, for example, to visit a variety of sites for high-bandwidth interactions with their local resources in order to quickly distill and collect information to support, say, a large data-mining operation. Due to the OCEAN’s open architecture, applications that harness the power of OCEAN can be developed by third-party vendors and then used by anybody – so that in the future the financial analysis package on one’s desktop, for example, may silently harness the power of thousands of distributed servers so as to return an answer more quickly. Or, for example, an airline-reservation web site might harness a vast, distributed database running on top of OCEAN to more quickly process a seat-reservation transaction.

C.IV. Results from Prior NSF Support

This section details the results from prior NSF support of all of the co-PIs on the project.
Dr. Frank and Dr. Schmalz have not had NSF support within the last 5 years.

Dr. Peters' prior NSF support:


(1) The theory of Geometric Continuity was brought to maturity with the characterization of the vertex-enclosure constraint. New algorithms for efficiently modeling free-form surface surfaces were discovered. [Pet95b,GP99,Pet98a]

(2) The research has made important contributions to a number of areas related to parametric surface representations:

(a) modeling implicitly defined surfaces based on box splines [PW97b]
(b) derivation of new and analysis of generalized subdivision [PR98b]
(c) a full characterization of quadratic parametric maps [PR98a]
(d) the first sharp estimates for the distance of a control polygon to its curve segment [NPL99]

The education short video *The topological house* was produced. The grant supported 5 graduate students and 7 undergraduate research assistants. The grant resulted in 20 journal publications and 10 refereed conference articles, and in educational outreach.

**NSF grant 1999-2001:**

Work with Georg Umlauf resulted in the first explicit published derivation of (Gauss and mean) curvature of subdivision surfaces in their extraordinary points [PU00]. This allowed us to give an alternative, more direct account of the criteria necessary and sufficient for achieving curvature continuity than earlier approaches that locally parametrize the surface by eigenfunctions.

Two papers [Pet00,Pet01] pointed out the availability of closed form smooth surface completions to refined quadrilateral or triangular mesh. To either type of refined mesh, in particular to meshes generated by subdivision, a smooth, low degree polynomial surface can be fitted provided extraordinary mesh nodes are separated by sufficiently many ordinary, 4-, respectively 6-valent mesh nodes.

**Dr. Helal & Dr. Hammer:** Abdelsalam (Sumi) Helal and Joachim Hammer have recently received a CISE CCR award, for Proposal number 0100770, titled "Adaptive Synchronization Framework Supporting Device-independent Mobile Computing." While the award period will start August 2001, Drs. Helal and Hammer have been pursuing this research and have published one paper and submitted another. Both papers are accessible from the following URL: http://www.harris.cise.ufl.edu/projects/3tier.htm, which is the web site for this project.

**Dr. Hammer** has been awarded an exploratory NSF grant titled "Scalable Enterprise Systems: Theory and Methodologies to Support the Operation of Flexible Production Networks." The goal of this research is to develop the next-generation wrapper, data engineering, and analysis technologies to support extended enterprise collaboration. So far, we have developed the initial concept and demonstrated an early prototype that can extract resource and cost information from legacy project resource planning software. Initial results from our prototype implementation demonstrate feasibility. This has allowed us to begin work with industrial partners including Pratt&Whitney Space Propulsion Operations. Preliminary results have been described in [Obr01].