**Proceedings of the**
**1999 ASME Design Engineering Technical Conferences**
**September 12–15, 1999, Las Vegas, Nevada**

# DETC99/CIE-9077

## AN INTERNET-BASED DISTRIBUTED SERVICE ARCHITECTURE

Charles S. Han
Stanford University

John C. Kunz
Stanford University

Kincho H. Law
Stanford University

## ABSTRACT

This paper describes a distributed architecture that enables the delivery of design services over the Internet. The architecture of an individual service is three-tiered. The first tier is a common communication protocol interface. The middle tier is the common product model interface. The third tier is the core of the design service. Though fundamentally decoupled, methods in the communication layer and the product model layer have been formalized to enable the aggregation of services and the support of problem decomposition. In addition, with the standardization of the first two tiers, it is possible to rapidly deploy various design services, both new and legacy applications, that can be easily made accessible via the Internet. As examples of design services, the prototype implements a project manager service with a companion CAD package, services that incorporate two legacy applications (a building code analysis service and a service that generates and displays an accessible path for a given floor plan design using motion planning and animation techniques), and a disabled access service that takes advantage of the decomposable infrastructure.

## INTRODUCTION

Traditional CAD systems are monolithic in that all functions or "services" are bundled in a software package. With the maturation of information and communication technologies, the concept that distributed CAD services are delivered over the Internet, Internet-based Computer-Aided Design (I-CAD), is becoming a reality. Technologies are now readily available to make the network-enabled CAD environment possible. Specifically, the technologies include a standard product model and a distributed object environment that allows for the development and transfer of a design based on the standard product model. In order to fully-leverage the power of the Internet, engineering and design services should be able to interact in a formal yet flexible manner. Services should be able to combine existing services to provide added functionality. This paper describes the developed service infrastructure and the prototype implemented to illustrate a framework that provides design services over the Internet.

The framework provides a means to distribute design services in a modular and systematic way. The protocol between services that has been developed specifies the manner in which services can aggregate, and this aggregation supports problem decomposition. In addition, users have the ability to select appropriate design services as opposed to having to use a large monolithic design tool. Furthermore, users can easily replace a service if a superior service becomes available without having to recompile the existing services being used. The framework provides a means to seamlessly integrate a legacy application as one of the modular design services in the infrastructure. With the standardization of the communication protocol and the exchange of product model data, integration of legacy applications as well as deployment of new design packages becomes a straightforward task.

In the prototype implementation, a project manager service with a companion CAD package, two design applications with companion viewers that incorporate legacy applications, and a design application specifically tailored to take advantage of the decompositional features of the infrastructure have been developed. The project manager service acts as a design model repository as well as a portal to direct a client application to other services provided in the infrastructure. A companion CAD package, a client service of the project manager service, has access to the design repository of the project manager service and queries the project manager service for the location of other appropriate services available over the Internet. The first two applications are examples of services that incorporate legacy applications. The first design application is the integration of a service that performs compliance checking of a design against governmental regulations, in this case for disabled accessibility (ADAAG 1997). The second design

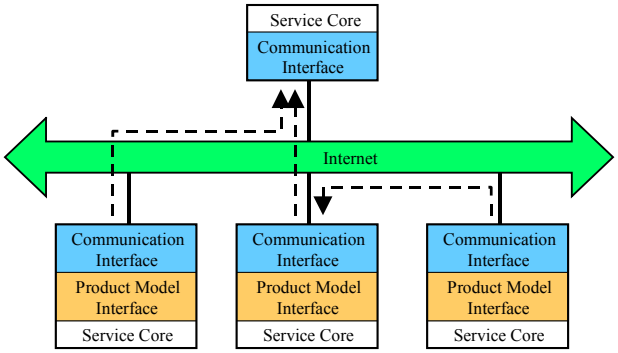


**Figure 1: A conceptual diagram of the distributed service architecture.**

```
module DistributedServiceArchitecture
{
   interface Service;

   typedef sequence<boolean>  BooleanArray;
   typedef sequence<Service>  ServiceArray;
   typedef sequence<string>   StringArray;

   interface Service
   {
      void             registerService();
      Service          getRegisteredService();
      ServiceArray     getRegisteredServices();

      string           getServiceId();
      string           getServiceType();
      void             putServiceType();
      void             putServiceId();

      void             execute ();
      boolean          getStatus();
      void             notifyService();

      void             clear();
      boolean          containsKey();
      BooleanArray     containsKeys();
      string           get();
      StringArray      gets();
      void             put();
      void             puts();
      void             remove();
      void             removes();

   };
};
```

**Figure 2: The communication protocol interface.**

application is a service that generates and displays a wheelchair accessible route for a given floor plan design. The final design application is a disabled access service that takes full advantage of the decomposition features of the distributed infrastructure.

The distributed object environment layer of the services is implemented with the Common Object Request Broker Architecture (CORBA) (Vogel and Duddy 1997), and the Industry Foundation Class (IFC) product model proposed by the International Alliance for Interoperability (IAI) is used for the common product model layer (Industry 1997). For the CAD service implementation as well as views of the data generated by the design services, standard World-Wide Web (WWW) browser technologies are employed although the use of these technologies is not mandatory for the functionality of a service or a view of a service within the infrastructure. However, using a standard browser interface leverages the most widely available Internet environment as well as being a convenient means of quick prototyping.

## INFRASTRUCTURE FOR THE DISTRIBUTED SERVICE ARCHITECTURE

Figure 1 shows the conceptual network-enabled framework for a distributed service. In this framework, each individual service adheres to a two- or three-tiered architecture. The first mandatory tier, a communication protocol interface, gives applications a common means to register its services and to send and receive data over the Internet. The optional middle tier, the common product model interface, is a standard protocol that describes the design data for a design service. The mandatory third tier is the core of the service—if the service is a design service, it extracts the appropriate information of the building design through the common product model interface and either modifies the design data or generates a report based on the analysis of the data.

As shown in Figure 1, a non-design (brokering) service does not need the product model interface that is present in the design services. An application package can register its services (the dashed line in Figure 1) with a brokering service and advertise its services in the infrastructure. Another service will query the brokering service for the existence of services in the distributed service architecture. Note that a design service can also register with another design service thus implying an aggregation of services that supports problem decomposition. The registration and query service is based on a pre-defined constraint language. In the following, the design of the multi-tiered architecture is examined in detail.

### The Communication Protocol Interface

Methods that define the communication protocol interface are illustrated in Figure 2. The methods are made publicly available by a service. The communication protocol interface is mandatory for the distributed service architecture. Following the object-oriented paradigm, the "exposed" methods are the points of entry into a service, but the actual implementation of these methods is dependent on the individual broker or service.

The brokering portion of the interface defines two methods, `registerService()` and `getRegisteredService()`. In addition, one service can execute a command in another service using the `execute()` method, and the called service can then notify the calling service when the execution is complete using the `notifyService()` method.

When an application registers its services with the broker or top-level service, it provides two arguments, the service that is being registered and a string that describes the service. In addition, to support problem decomposition, when an

application registers its services with a broker, it specifies when it will be executed. For example, if the service is a child service, it is executed along with the other registered child services. When these child services notify the parent service that their tasks are complete, non-child services that have registered with the broker or top-level service will process the data extracted from the child services.

When the broker is queried for a service, the broker returns a registered service that matches the description of the query argument. In a real implementation of this infrastructure, a more sophisticated broker protocol would be necessary. For example the client would need to provide client registration information and, for building code analysis services, geographic location and a more specific description of the type of building code (structural, mechanical, electrical, etc). An even more sophisticated protocol would involve a more extensive communication sequence between a client and the broker. For example, the client could query the broker for a set of existing services that match a specific set of constraints, and then the client could choose among the services returned by the broker. For further description on a broker protocol, see the discussion on the CORBA Trading Service by Vogel and Duddy (1997).

Design data must be exchanged between services, and `puts()` and `gets()` are the most important methods. The `puts()` method sends the product model to the service with two arguments, a string identifying the name of the model and a stream of data that defines the model. The `gets()` method returns a model with the name identified by the single string argument. The `gets()` method retrieves data from an invoked service once that invoked service has notified the invoking application that it has completed its task. If several child services have been invoked, the `gets()` method will be invoked upon notification by all child services. The retrieved data must be resolved by aggregation methods developed for the product model.

Finally, services themselves have the ability to process a query concerning the existence of other services through a project manager service module. The motivation for providing this ability is that in an enterprise environment, the user of a CAD package may not be given the responsibility of knowing what services should be used. The project manager service has the responsibility of filtering the information given by the broker. We also allow services to register and query other services to provide the most flexibility and better distributed functionality. Following the distributed service paradigm, services may be distributed as processes of one CPU, processes of a multi-processor system, or among workstations within a cluster. When services register with a higher-level service, this higher-level service can execute commands of the registered services as threads. Finally, when a service is initiated, it can be queried for its status with the `status()` method since a service may not necessarily process the information or complete the work immediately. With this small set of methods for the broker and a service, the communication protocol layer is fully functional and can be utilized by an application that operates within the distributed network-based environment.

## The Product Model Interface for Design Services

In order to create an infrastructure in which design services can be incrementally added and since each service will have its own representation of the design data, the availability of multiple services suggests a mapping of each service's design data representation to a common model. Without this lingua franca, a process that needs to use a specific service would need to have a mapping of its representation of the design data to the service's design data representation, and the need to use a new service would require additional mapping.

With the three-tiered service architecture illustrated in Figure 1, the product model interface has been deliberately decoupled from the communication protocol interface. Using a distributed object paradigm, it would have been possible to expose the objects or the design components of a common product model in the communication protocol interface, thus combining the communication protocol and product model layers. However, if the objects (and their attributes and relationships) are made public at the communication protocol level, as the product model evolves, the communication protocol must also evolve to take into account the product model evolution. By decoupling the two layers, it is up to the individual service whether the product model interface needs to be modified to accommodate the product model evolution. Finally, since some services do not require the product model layer (thus optional for those services), separating it from the communication layer allows for a smaller communication protocol.

As previously noted, the `puts()` and `gets()` methods of the communications protocol interface take as input and output a stream of design data. In decoupling the two layers and keeping the infrastructure general, the trade-off comes in the efficiency of sending the design data. The integrated approach of defining the objects at the communication protocol layer would be much more efficient but requires the continuing evolution of the communication protocol as the product model evolves. Although there are no restrictions on how the composition of the data stream is imposed by the infrastructure, in the prototype, a string array describes the model in EXPRESS format that conforms with the standard IFC product model (Industry 1997).

The product model interface stores the design data according to the individual service's needs. In the implementation of the prototype, all application services (as well as the client CAD package) store the design data in a hash table keyed by the component identification string. The critical constraint is that each individual service is able to understand the common product model that has been agreed upon a priori. However, by making the product model storage consistent across the infrastructure, it would be easier to reuse methods to extract and to send the critical data from the product model to the core of the design service for the appropriate analysis.

Finally, even though the decoupling aspect of the distributed service architecture has been emphasized, the product model supports problem decomposition (and recomposition) as described in the previous section. Specifically, the model supports views that allow decomposition of the problem according to the type of analysis being performed. It also supports the recomposition and resolution of the modified decomposed parts.

### The Design Services

There is a distinction between the functionality of a non-design and a design service core. For a non-design service, the core layer will simply process queries and the registration of services. Since there is no product model associated with a non-design service, no transformation or modification of a product model is required.

Any design service needs to extract a view or a diagram from the product model (Clancy 1985). The transformation from the product model to a view or a diagram is unique to the application. The core of the design service needs to map the common product model data to its own design data representation. If part of the task of the design service is to modify the product model that has been deposited into its repository, the service must then perform a reverse mapping from its own design data representation back to the common product model to update the model. Otherwise, the service simply generates a report.

In the top-level view of the implemented services, the first two design applications simply generate a report after mapping the common product model to their own design data representation and performing some analyses. Therefore, each of these services represents a one-way exchange of design data. To generate their respective reports, these services translate their own internal design data representations into human-readable formats.

If the services were to report their findings directly to the originating process, a two-way exchange of design data would be required. The third design service is actually composed of a hierarchy of services. Two-way communication between child and parent services in the hierarchy is executed within the communication layer, and the modified data is merged and resolved within the product model layer. The project manager is an example of a top-level service that performs both input and output of a design model, a two-way exchange of design data. Even though the project manager is only a repository and does not modify the design data, it still must perform a mapping and a reverse mapping corresponding to the input and the output of a design model.

## THE DISTRIBUTED SERVICE ARCHITECTURE PROTOTYPE

This section describes the technologies used to implement the distributed service architecture for a set of design services as illustrated in Figure 3. CORBA provides a high-level distributed object paradigm that is well-suited to implement a network-based distributed service architecture. Specific desirable features include interface implementation independence, object access independent of the implemented programming language, object access independent of location, and access to standard distributed object services and facilities (Vogel and Duddy 1997).

A combination of a simple, flexible generic product model and the geometric properties of the IAI's IFC model (Industry 1997) is employed as the common product model for the design services. Presently, The IAI IFC product model effort is the only product model that has the support of major CAD vendors and manufacturers associated with the building industry. A seamless interface between the communication layer and standard product models would be critical for the proliferation of design applications over the Internet.

Java is employed as the programming language to implement most parts of the distributed service architecture, both the services as well as the service clients. Java's well-designed object-oriented structure and platform independence were the influencing factors in using it as the implementing programming language.

The wide proliferation of the Internet can be attributed to the ease of World-Wide Web access, so taking advantage of this environment was appropriate. When interaction was needed within the Web browser environment, Java applets could be seamlessly integrated into the Java-written distributed architecture. The client CAD package and the accessible path viewer employ Java applets to create browser-based interfaces.

The disadvantage of Java's implementation of platform independence is execution speed. Thus, the motion planner and the animator are application service programs written in C++ for optimal computational performance. Since Java provides simple methods for incorporating external processes, the integration of the C++ applications within the Java-written infrastructure was straightforward.

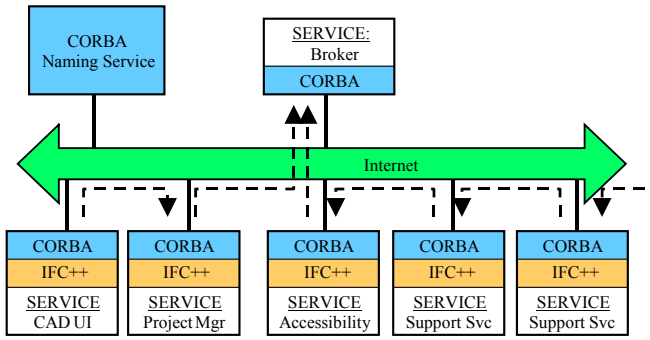


**Figure 3: A distributed service architecture implementation**

## The Common Distributed Object Environment

The communication protocol interface shown in Figure 2 is a simplified Interface Definition Language (IDL) file used to generate the CORBA-related Java source code—the arguments of the methods are not shown. Though applications access objects and their methods defined in the IDL file in a distributed object environment across the Internet, from the application's point of view, these objects and methods are treated as local entities. This concept underlines the power of the CORBA paradigm.

One feature of CORBA is the Naming Service which allows distributed objects or applications to register and locate other distributed objects or applications at a common location by name (Vogel and Duddy 1997). The order in which the various distributed applications are launched is critical: The Naming Service must be launched first. For the prototype, a broker called "SimpleTrader" is the first process that registers with the Naming Service since the design services will register with the broker. The design services must be launched before their respective clients (the CAD package is a client to the project manager service, and the path-viewing is a client to the path planning service). It is the responsibility of the project manager service to query the broker for the availability of design services. If a design service has not been launched (and therefore has not registered with the broker), when the broker is queried by the project manager service for that design service, the broker will simply inform the project manager service that the requested design service is not available.

When a design service is initialized, it queries the Naming Service for the broker object. The Naming Service returns the broker object to the design service, which can now register with the broker. Similarly, when a client of a design service is initialized, it queries the Naming Service for a specific design service object. For example, the client CAD package queries the Naming Service for the project manager service by its name "ProjectManager". Once the Naming Service returns the service object to the client, the client can interact with the service using the exposed methods that are described in the IDL file.

## The Common Product Model for Design Services

The communication protocol interface specifies methods for sending and retrieving a product model from a service without the interface having any knowledge about the semantics of the design model. However, understanding the semantics of the design model is the responsibility of the product model interface. In the prototype, a generic and flexible product model is used. A generic component, `GenericComponent`, has only an identification string and three fields defining its form, function, and behavior. The form is described using the IAI's IFC geometric representation scheme. Figure 4 illustrates the product model hierarchy.

The product model interface constructs an internal representation of the design data from the data stream. A Java class structure that mirrors the IFC EXPRESS schema's class hierarchy, attributes, and relationships has been constructed for the internal representation. The communication protocol interface or the common product model interface does not specify how the design data is stored, but in the prototype implementation, the data is stored in a hash table keyed by the identification string of the component. The only restriction mandated by the common product model interface is that the core of the design service accesses the components in the repository by the name of the model and the name of the component.

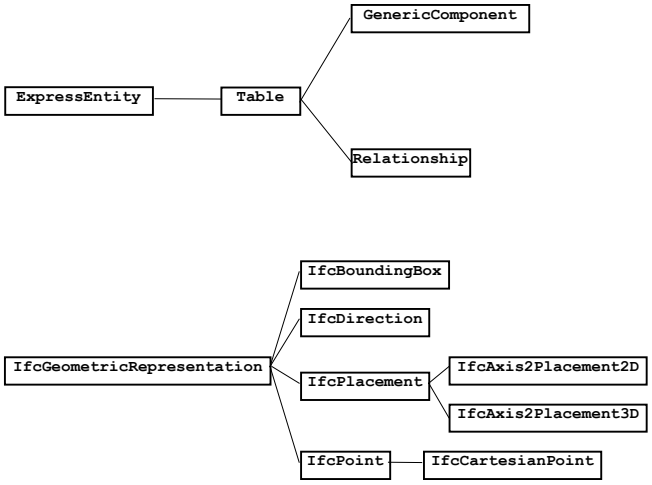


**Figure 4: The common product model hierarchy.**

Components can be aggregated using the `Relationship` object. The way in which the product model is partitioned with `Relationship` objects effects the manner in which the model is decomposed among child services. In addition, the product model supports recomposition and resolution of data modified by child services with a `merge()` method.

## The Design Services

A variety of the design services implemented using the prototype infrastructure demonstrate the flexibility of the distributed service architecture. The combination of the project manager service and a client CAD package illustrates a new application developed specifically for the distributed service architecture, while the other two design services are legacy applications to various degrees. As shown in Figure 3, the client applications of the design services vary in their use of the communication protocol interface and the product model interface.

### Top-Level Broker: a Non-Design Service

As previously noted, a non-design service does not require the common product model interface, and the top-level broker is an example of a non-design service (see Figure 3). The top-level broker registers with the CORBA Naming Service and

waits for other services (in this case, the design services) to

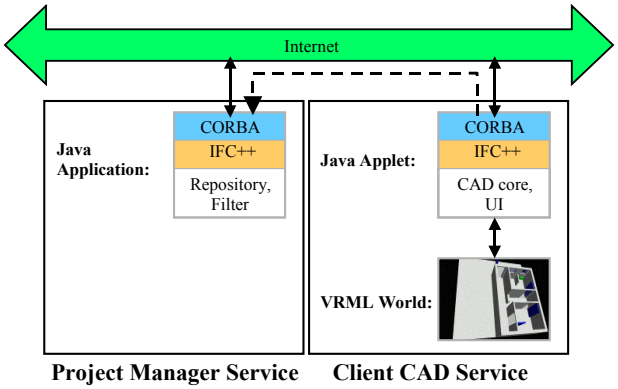


**Figure 5: The program manager service and the client CAD service.**

register with the top-level broker. When an application queries the top-level broker about the existence of a particular registered service, the top-level broker will reply by returning the applicable service object to the querying application. Once the querying application receives the applicable service object, the querying application is able to interact with that applicable service object by sending the appropriate design data.

## Project Manager Service/Client CAD Service

The project manager is a very simple service to illustrate the functionality of the communication protocol and product model interface. Since the main function of the project manager service is to act as a repository of design models, the service core layer does not perform any design data extraction and analysis from the design model that resides in the product model interface.

The only other function the project manager service has is to process queries from the registered client CAD service by first passing the queries to the broker and then transferring the service object that the broker has returned back to the CAD package. In an enterprise environment, instead of simply acting as a pass-through mechanism, the project manager service would authenticate a CAD service when the CAD service registers with the project manager. The project manager would also act as a security agent that would filter the CAD package's service request depending on the constraints of the specific project.

The client CAD service, a combination of a Java Applet and a Virtual Reality Modeling Language (VRML) ("Information" 1997) interface, is launched when its Web page is accessed. Figure 5 depicts the details of the CAD package architecture. The applet first retrieves the project manager service object from the CORBA Naming Service and registers with the project manager. It can then send requests to the project manager using the `gets()` and `puts()` exposed methods. The third tier or service core of the client CAD package includes the UI.

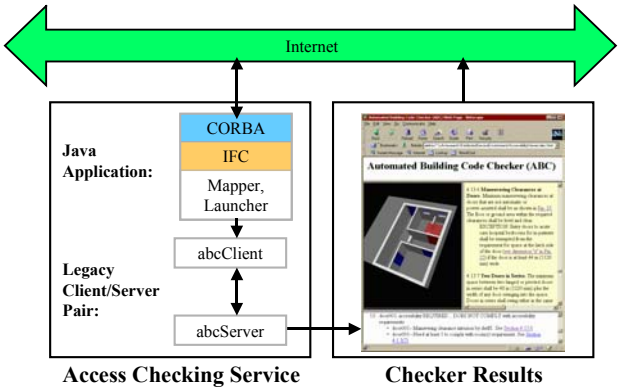


**Figure 6: The accessibility analysis service and the generated report**

The Java applet handles the textual UI, and the applet communicates with the VRML graphical UI using the VRML External Authoring Interface (EAI) (Marrin 1997). The various aforementioned Web browser-associated technologies allow rapid prototyping of the client CAD package. Any commercial IFC-compliant CAD package that could incorporate the CORBA communication protocol could also be used.

The CAD package can retrieve building models either from the project manager service repository or from a static local model file, which is in EXPRESS file format. The retrieved model can be viewed, modified, and saved back to the project manager repository. In addition, the CAD package queries the project manager for the available design applications that have been registered, and it can send the model that is currently being developed to the design applications.

## A Legacy Disabled Access Code Checking Service

The disabled access code analysis service is an example of a legacy client/server application (Han et. al. 1998) that is wrapped by the service. Once wrapped, the legacy application can be integrated into the distributed service architecture. Therefore, the core of the Java's application service is to map the common product model to the legacy application's product model. Once the design data is mapped, the service launches the legacy application which then sends the stream of design data to the server where the code compliance checker resides.

The legacy server application receives the design data, runs a code compliance analysis on the floor plan of a building, and generates a Web page with a VRML model of the design with redlines (see Figure 6). The redlines have hyperlinks to code comments, and the code comments have hyperlinks to the actual building code document, in this case, the Americans with Disabilities Act Accessibility Guidelines (1997).

## A Legacy Accessible Path Generating Service

The accessible path generating service illustrated in Figure

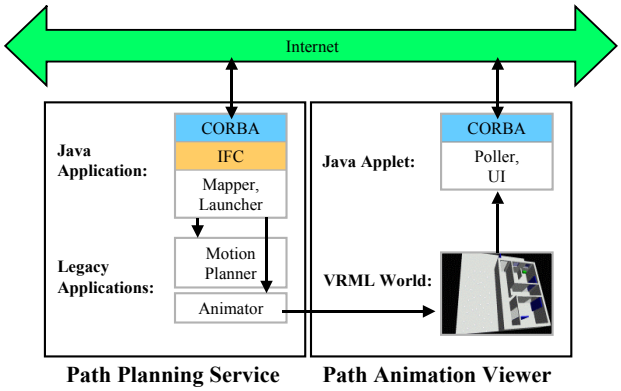


**Figure 7: The wheelchair path planning service and the animation viewer**

7 incorporates two legacy applications, a motion planner and an animation generator using techniques described by Latombe (1991) and Han et. al. (1999), both of which are written in C++. As illustrated in Figure 7, the Java application portion of the service launches the motion planner and the animator processes, a straightforward process using Java. The core of the application service extracts the necessary design data from the product model interface and sends it to the first legacy application, the motion planner. The animator then reads the path file generated by the motion planner, and it generates the VRML animation. The client viewer's communication interface polls the path-planning service for the service's status. The viewer is notified when the service has completed its analysis, and, if desired, then loads the animator-generated VRML file.

Before calling the motion planner, the design core interface first extracts the design data from the product model interface and creates a file composed of the positions of the obstacles, an initial point, and a goal point. The walls and windows of the building model are mapped to obstacles, the center of the entry space is mapped to the initial point, and the space in front of a designated fixture (the clearance space) is mapped to the goal point.

The motion planner reads in the obstacles, initial point, and goal point, and generates a predetermined number of configuration-space potential-field maps based on the granularity of the angular increment of the wheelchair robot. The motion planner creates a file with the sequence of positions of the wheelchair from the initial point to the goal point. The animator reads in this two-dimensional information to generate the motion of a wheelchair in VRML by calculating the major and minor wheel movements and the arm motion of the wheelchair occupant.

## The Disabled Access Code Analysis Service

The disabled access code analysis application is an example of a set of hierarchically structured design services. By structuring an application as a set of service modules, the

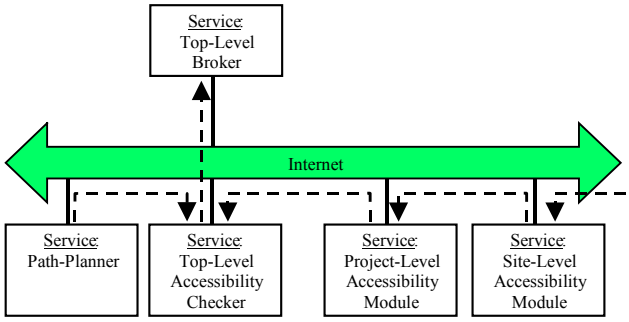


**Figure 8: The disabled access analysis service.**

application can be distributed across a network of processors or a cluster of workstations. If the service modules are initialized on the same workstation, it is the responsibility of the workstation's operating system to distribute the modules as separate processes in a single-processor machine or among the processors of a multi-processor machine. Currently, to distribute service modules of a particular application within a cluster environment, the service modules have to be manually initialized on separate workstations. Following the distributed object paradigm of location transparency, a service that registers other services does not know whether the registering service resides on the same workstation or on a different workstation in the cluster.

As shown in Figure 8, the top-level service of the disabled access code analysis application registers with the top-level broker service described above. Two service modules of the disabled access application in turn register with the top-level disabled access service. One of these modules is a path-planning service that analyzes a design for the existence of an accessible path using motion planning techniques described by Latombe (1991). The other module initiates a chain of modules that the disabled accessibility application uses to decompose the design in a hierarchical manner. The chain consists of modules that decompose the design on the project level, site level, building level, story level, space level, and finally on a cell or sub-space level. This hierarchy mirrors the implemented product model hierarchy.

The top-level design service receives the design data from another application (such as the previously described CAD service). The top-level design service acts as a broker for the path-planning service since it does not actually send design data to it. The top-level design service does, however, send design data through the chain of service modules. Each of these modules uses the path-planning service by first querying the top-level design service for the path-planning service object. Once the particular service module in the chain

receives the pertinent path-planning data, it further decomposes the design model by sending parts of the design to the next module in the chain.

## DISCUSSION

This paper has described a distributed service architecture that allows design services to be incorporated into a modular network-enabled infrastructure. The infrastructure supports aggregation of services and problem decomposition. Three such services, a project manager and three design applications, were implemented. By providing a modular infrastructure, services can be added or updated without re-compilation or re-initialization of existing services. While this work has focused on automated disabled access building design analysis, other building-design-related services can be implemented and linked to the network infrastructure. For example, if a data warehouse inventory service is implemented, a designer could query such a service for the availability and pricing of a specific component such as a fixture, an appliance, etc.

The infrastructure is contingent on two common interfaces, the communication protocol interface and the common product model interface. As demonstrated, standardizing a communication protocol and decoupling it from the product model interface has its advantages and drawbacks. The major advantage is that the communication protocol interface does not depend on changes (updates and evolution) of the product model since the stream of design data is object-independent; it is the responsibility of the product model protocol to parse the design data stream. The major disadvantage is the inefficiency of the transmission of the design data from service to service. Standardization of the product model is more of a challenge. While the services demonstrated have simple mappings from the common product model to their own design representations without loss of critical information, other services may need more information than is provided in the standard product model, thus bringing up issues such as object overloading. There are at least two issues that need to be addressed. First, the product model must have a formal mechanism to extend the model with new attributes and relationships. Second, in order to address the object overloading issue, there must be a formal mechanism for a service to query the process that calls the service to extract only the necessary attributes and relationships of a building design model.

Finally, the distributed service architecture prototype described in this paper does not address issues of security raised by Regli (1997) and others (for example, see Wiederhold et. al. (1997) and version control (Krishnamurthy 1996)). Companies and IT vendors are actively investigating the technical requirements that must be met in order to support network-based collaborative design that can span multiple organizations (InfoTEST 1997). Overly protected rigid environment, such as “firewalls”, often make collaboration difficult (Gong 1996). We believe that some of the security issues could be addressed in the communication protocol interface and the project manager service while version control could be incorporated in the common product model interface.

## ACKNOWLEDGMENTS

This research is partially sponsored by the Center for Integrated Facility Engineering at Stanford University.

## REFERENCES

Americans with Disabilities Act Accessibility Guidelines. (1997). Access Board, U.S. Architectural and Transportation Barriers Compliance Board, Washington, D.C.

Clancy, W. J. (1985). Heuristic classification in artificial intelligence, Elsevier Publishers B.V., North-Holland.

Gong, L. (1996), “Enclave: enabling secure collaboration over the internet,” Proc. Of the Sixth USENIX Security Symp., pp. 149-159.

Han, C.S., Kunz, J.C., and Law, K.H. (1998). “A client/server framework for on-line building code checking,” J. Comp. in Civ. Engrg., ASCE, 12(4), 181-194.

Han, Charles S., Kunz, J.C., Law, K.H. (1999), "Building Design Services in a Distributed Architecture," J. of Comp. in Civ. Engrg., ASCE, Vol. 13(1), pp. 12-22.

Industry foundation classes release 1.5, specifications volumes 1-4. (1997). International Alliance for Interoperability, Washington D.C.

“Information technology—computer graphics and image processing—the virtual reality modeling language (VRML)—Part 1: Functional specification and UTF-8 encoding.” (1997). ISO/IEC 14772-1:1997, International Standards Organization, Geneva, Switzerland.

InfoTEST International Enhanced Product Realization Testbed (1997), a report by D.H. Brown Associates, Inc., Port Chester, NY.

Krishnamurthy, K. (1996). “A data management model for change control in collaborative design environments,” PhD Thesis, Dept. of Civ. Engrg., Stanford University, Stanford, Calif.

Latombe, J.C. (1991). Robot motion planning, Kluwer Academic Publishers, Norwell, Mass.

Marrin, C. (1997). Proposal for a VRML 2.0 Informative Annex: External Authoring Interface Reference, an unpublished draft of a proposal to the ISO.

Regli, W.C. (1997). “Internet-Enabled Computer-Aided Design,” IEEE Internet Computing, IEEE 1(1), 39-50.

Vogel, A., and Duddy, K. (1997). Java programming with CORBA, John Wiley and Sons, Inc., New York, N.Y.

Wiederhold, G., Bilello, M., Sarathy, V., and Qian, X.L. (1996). “Protecting Collaboration,” Proc., NISSC'96 National Information Systems Security Conference, Baltimore Md., 561-569

 