8th INTERNATIONAL CONFERENCE ON DURABILITY OF BUILDING MATERIALS AND COMPONENTS
May 30 - June 3, 1999 Vancouver, Canada

AN INTERNET-BASED DISTRIBUTED BUILDING DESIGN SERVICE FRAMEWORK

Internet-Based Building Design Services

C.S. HAN, J.C. KUNZ, and K.H. LAW
Center for Integrated Facility Engineering, Stanford University, Stanford, CA 94305-4020

Abstract

This paper describes a distributed service architecture that enables the delivery of building design services over the Internet. With this architecture, it is possible to rapidly deploy various services, both new and legacy applications, that can be easily accessed via the Internet. As examples of services, the prototype implements a top-level brokering service, a project manager service with a companion CAD package service, and a set of disabled building code analysis services.

Keywords: automation, distributed object environment, World-Wide Web (WWW), Computer-Aided Design (CAD), Industry Foundation Classes (IFC)

1 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, is becoming a reality. Regli (1997) outlined the technologies that 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.

This paper describes a prototype implemented to illustrate a framework that provides building design services over the Internet. The framework provides a means to distribute design services in a modular and systematic way. With this infrastructure, users have the ability to select appropriate design services (as opposed to having to use a large monolithic design tool) and can easily replace a service if a superior service becomes available without having to recompile the existing services

being used. 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. As examples of services, the prototype implements a top-level brokering service, project manager service with a companion CAD package service, and a set of disabled building code analysis services.

2 Infrastructure of the Framework

Fig. 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 the application services a common means 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.

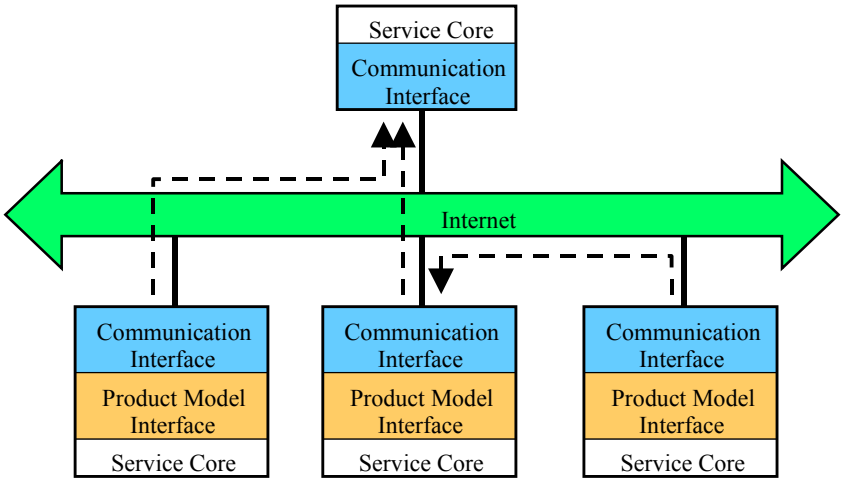

Fig. 1: A conceptual diagram of the distributed service architecture.

As shown in Fig. 1, a non-design 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 Fig. 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 act as a brokering service. The registration and query service is based on a predetermined constraint language. In the following, the design of the multi-tiered architecture is examined in detail.

2.1 The Mandatory Communication Protocol Interface

Methods that define the communication protocol interface that are made publicly available by a service is illustrated in Fig. 2. 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.

```
module DistributedServiceArchitecture
{
  interface Service;

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

  interface Service
  {
    void         registerService(in Service service);
    Service      getRegisteredService(in string serviceId, in string serviceType);
    ServiceArray getRegisteredServices();

    string       getServiceId();
    string       getServiceType();
    void         putServiceType(in string serviceType);
    void         putServiceId(in string serviceId);

    void         execute (in Service callingService, in string sessionId, in string command);
    boolean      getStatus(in Service callingService, in string sessionId);
    void         notifyService(in Service calledService, in string sessionId, in StringArray data);

    void         clear(in Service callingService, in string sessionId);
    boolean      containsKey(in Service callingService, in string sessionId, in string key);
    BooleanArray containsKeys(in Service callingService, in string sessionId, in StringArray keys);
    string       get(in Service callingService, in string sessionId, in string key);
    StringArray  gets(in Service callingService, in string sessionId, in StringArray keys);
    void         put(in Service callingService, in string sessionId, in string key, in string data);
    void         puts(in Service callingService, in string sessionId, in StringArray keys, in StringArray data);
    void         remove(in Service callingService, in string sessionId, in string key);
    void         removes(in Service callingService, in string sessionId, in StringArray keys);

  };
};
```

Fig. 2: The communication protocol interface.

In the simple prototype implementation, the brokering portion of the interface defines two methods, `registerService()` and `getRegisteredService()`. For the purposes of the prototype, this simple interface is sufficient to illustrate the minimum specification of a broker. 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.

The interface for communication of design data is also very simple with `puts()` and `gets()` being 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.

In a previous prototype, the brokering methods were contained in a separate brokering object than the service object (Han 1999). However, allowing services to register and query other services provides 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. With this small set of methods, the communication protocol layer is fully functional and can be utilized by an application that operates within the distributed network-based environment.

2.2 The Optional Product Model Interface for Design Services

The availability of multiple design 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 multi-tiered service architecture illustrated in Fig. 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 building 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, 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 as strings. In decoupling the two layers and keeping the infrastructure general, the trade-off comes in the efficiency of sending the design data. Although there are no restrictions on how the string array is imposed by the infrastructure itself, in the prototype, the string array is a description of the model in EXPRESS format.

2.3 The Mandatory Service Core

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.

3 The Distributed Service Architecture Prototype

This section describes the technologies used to implement the distributed service architecture for building design services as illustrated in Fig. 3. The service communication protocol layer is implemented using CORBA. CORBA provides a high-level distributed object paradigm that is well-suited to implement a network-based distributed service architecture. A combination of a simple, flexible generic product model and the geometric properties of the International Alliance of Interoperability (IAI) Industry Foundation Classes (IFC) (*Industry* 1997) is employed as the common product model for the design services.

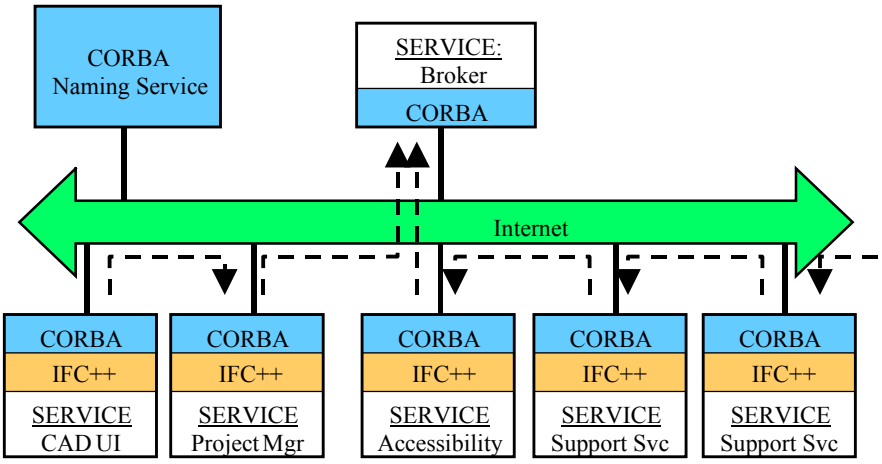

Fig. 3: A distributed service architecture implementation

Java is employed as the programming language to implement most parts of the distributed service architecture. 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 CAD service employs a Java applet to create browser-based interfaces.

3.1 The Common Distributed Object Environment

The communication protocol interface shown in Fig. 2 is the Interface Definition Language (IDL) file used to generate the CORBA-related Java source code. 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 top-level broker service 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 service is a client to the project manager service). One 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.

3.2 The Common Building Product Model for Design Services

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

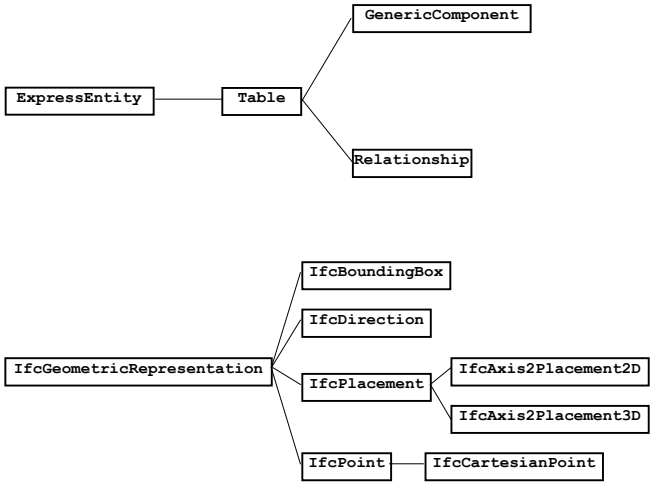

Fig. 4: The common product model hierarchy.

The product model interface constructs an internal representation of the design data from the data stream. The only restriction mandated by the common product model interface is that the core of the design service accesses the building

components in the repository by the name of the model and the name of the building component.

4 The Services

4.1 The 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 Fig. 3). The top-level broker registers with the CORBA Naming Service and waits for other services (in this case, the design services) to 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.

4.2 The Project Manager Service with a 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 building models, the service core layer does not perform any design data extraction and analysis from the building 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. Fig. 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. 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.

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.

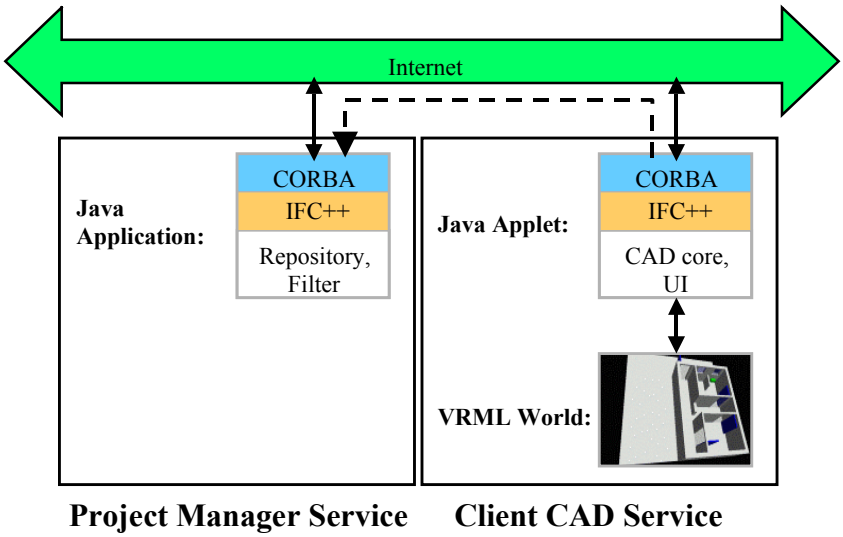

Fig. 5: The program manager service and the client CAD service.

4.3 The Disabled Access Code Analysis Application Services

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 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 Fig. 6, the top-level service of the disabled access code analysis application registers with the top-level broker service described in Section 4.1. 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) and Han (1999). 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.

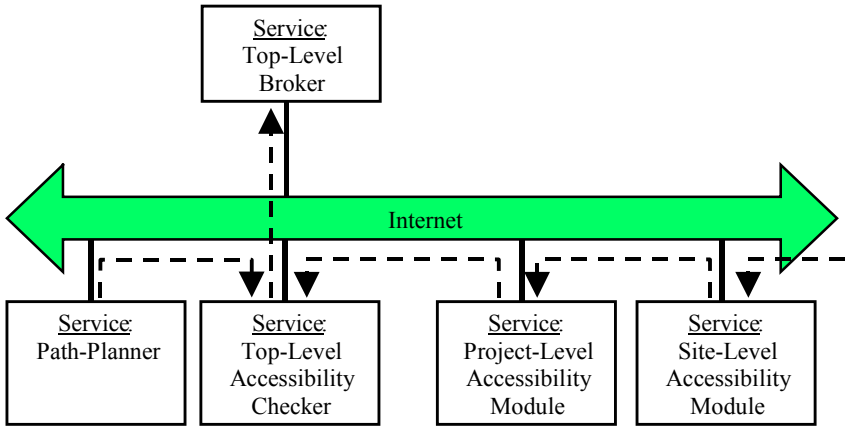

Fig. 6: The accessibility analysis service and the generated report.

5 Discussion

This paper has described a distributed service architecture that allows design services to be incorporated into a modular network-enabled infrastructure. By providing a modular infrastructure, services can be added or updated without re-compilation or re-initialization of existing services. In addition, by allowing portions of a service to be further modularized and to register with a parent service, separate tasks of an application can be distributed over the network. Other building-design-related services than the ones described can be implemented and linked to the network infrastructure. For example, if a building component 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 the interfaces for the communication protocol and the common product model that is required for design services. 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.

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)). 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.

6 Acknowledgments

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

7 References

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

Han, C.S., Kunz, J.C., and Law, K.H. (1999). "Building Design Services in a Distributed Architecture," Journal of Computing in Civil Engineering, ASCE, Vol. 13, No. 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.

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). "A fast planner for a car-like indoor mobile robot," *Proc., Ninth National Conference on Artificial Intelligence*, Anaheim, Calif., 659-665.

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.