A Distributed Object Component-based Approach to Large-scale Engineering Systems and an Example Component Using Motion Planning Techniques for Disabled Access Usability Analysis

Charles S. Han[1], John C. Kunz[2], Kincho H. Law[3]

Abstract

This paper introduces a large-scale engineering systems distributed object framework that serves as the connecting infrastructure between individual engineering tools and analysis components. The framework can incorporate software services as well as hardware solutions. Implemented software services include motion planning simulations and animation visualizations. Hardware solutions include the ability to incorporate devices to manipulate either remote software or hardware services.

The success on such a framework relies on two levels of knowledge: understanding the underlying concepts that facilitate the component-to-component communication and the domain-specific knowledge to implement a particular service. As an example of a domain-specific service, the paper describes a motion-planning-based disabled access usability service that attempts to find accessible routes in a facility.

Introduction

Currently, Architecture/Engineering/Construction (AEC) information technology (IT) firms are developing specialized information-exchange solutions for domain-specific problems, and standardization enables users ubiquitous access to these tools. Indeed, these solutions use at least one level of Internet-based standard protocols: all firms are leveraging World Wide Web technology, and some firms are beginning to

[1] Graduate Student, Department of Civil and Environmental Engineering, Stanford University, Stanford, CA 94305, csh@galerkin.stanford.edu
[2] Senior Research Associate, Center for Integrated Facility Engineering, Stanford University, Stanford, CA 94305, kunz@cive.stanford.edu
[3] Professor, Department of Civil and Environmental Engineering, Stanford University, Stanford, CA 94305, law@cive.stanford.edu

leverage standard distributed object paradigms. However, the implementation of large-scale and interoperable engineering systems will require the development and adherence to a third layer of standard protocols built on top of the distributed object technology.

A distributed object environment provides the underlying application-to-application communication protocol allowing an application to access the services of other applications as if these services were part of the original application, a feature known as object location transparency. This paradigm provides the facility for development of computing environments across heterogeneous platforms. Benefits include the optimization of services on specific computing platforms and taking advantage of unused or under-utilized computing resources. Most importantly, distributed object applications now act as individual but inter-communicating components that can be aggregated to form the multiple and coordinated layers of large-scale systems. This protocol must be general enough to accommodate and anticipate future engineering-specific needs to build large-scale systems on a component-by-component basis but robust enough to be truly useful for engineering-specific services.

This paper introduces a distributed object framework that serves as the connecting infrastructure between individual engineering tools and analysis components. The framework can incorporate software services as well as hardware solutions. Implemented software services include motion planning simulations and animation visualizations. Hardware solutions include the ability to incorporate devices to manipulate either remote software or hardware services. The distributed object paradigm does not distinguish between local and remote access of components that can either be software applications or hardware devices.

As an example component service that integrates into the large-scale framework, the paper describes a disabled access usability analysis service. This development of this component represents a domain-specific problem that is dependent on the developer's knowledge of the domain. Specifically, the paper describes the application of motion-planning techniques used to determine accessible routes in a facility.

The Engineering Analysis Component-based Distributed Object Framework

This research uses the concepts developed in (Han99) and reifies notion of a service. 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.

The distributed object environment provides object transparency—an application accesses a `Service` object using the same protocol regardless of the object's location, either local or remote, and independent of the computer system platform assuming the platform supports the `Service` object interface.

Figure 1 shows the network-enabled framework with four `Service` objects. In this environment, each individual service adheres to a three-tiered architecture. The first tier, a communication protocol interface, gives the application services a common means to send and receive design data over the Internet. The middle tier, the optional common product model interface, is a standard protocol that describes the design data. The third tier is the core of the design service—the design service extracts the appropriate information such as 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.

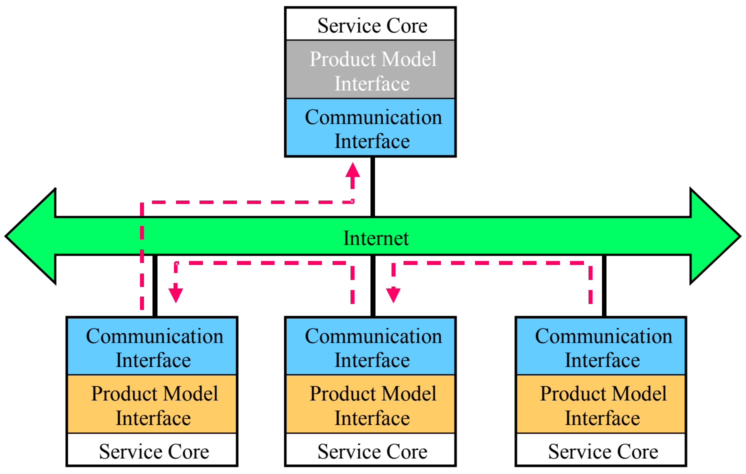

Figure 1: The three-tiered distributed object framework

One service can register with another service in the infrastructure. The registration and query or a `Service` object is based on a predetermined constraint language. When the core of a parent design service executes its analysis, it may send parts of the analysis to child services that have registered with the parent service.

The communication protocol interface makes certain methods of the `Service` object public as shown in Figure 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 service. The `Service` object consists of two registration methods, `registerService()` and `registerDecompositionService()`, that allow a service to register with another service. A service registers using the `registerService()` method with a broker that will advertise the service. Any child service that a parent service will execute will register with the `registerDecompositionService()` method. The `Service` object also provides both polling and callback mechanisms (`getStatus()` and `notification()`) to communicate with child services.

```
module DistributedServiceArchitecture
{
    interface Service;

    typedef sequence<string> StringArray;

    interface Service
    {
        void            registerService(in Service service, in boolean notify);
        void            registerSequentialService(in Service service);
        void            registerParallelService(in Service service);
        Service         getRegisteredService(in string id, in string type);
        Service         getRegisteredSequentialService(in string id, in string type);
        Service         getRegisteredParallelService(in string id, in string type);

        string          getServiceId();
        string          getServiceType();
        void            putServiceId(in string id);
        void            putServiceType(in string type);

        void            preProcess();
        void            sequentialDecomposition(in Service service, in string session);
        void            parallelDecomposition(in Service service, in string session);
        void            postProcess();

        boolean         getStatus(in Service service, in string session);
        void            notification(in Service service, in string session);

        StringArray     gets(in string session);
        void            puts(in string session, in StringArray strings);
    };
};
```

Figure 2: The communication protocol.

The product model interface of a service component allows each component to process the design data. This framework assumes a single product model interface, but realistically, services that translate product model information from one domain-specific format to another would need to reside in the infrastructure. The implemented framework uses the International Alliance for Interoperability (IAI) Industry Foundation Class (IFC) product model as the point-of-departure (IAI97).

Figure 3 shows the implemented framework. The *CAD Service* enables a CAD package to download a design to a browser-based central user interface (UI) component of the framework. The *CAD Client Service Object* registers with the *CAD Server Service Object*, and the CAD package uploads the design data to the UI component using this client service as the intermediary between the CAD package computer system and the UI component. The *CAD Client Service* resides on the same system as the CAD package. Leveraging the distributed object paradigm, the CAD package probably resides on a different system than the UI component, though the UI component can be seen on the CAD package system through the web browsing environment.

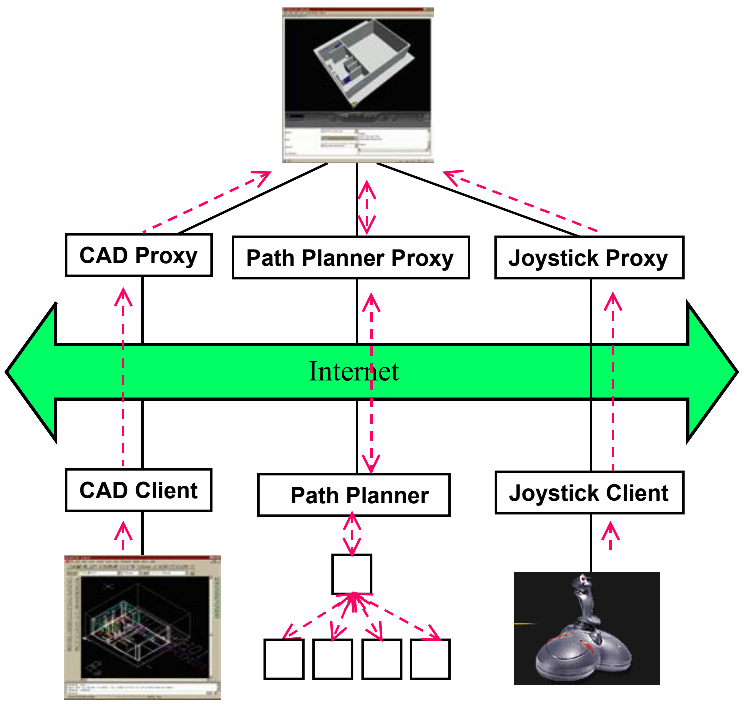

Figure 3: The implemented distributed object framework

Figure 3 also shows the *Joystick Service* which allows the manipulation of a wheelchair through the facility. Incorporation of the *Joystick Service* into the framework underlines the generality of the framework. *The Joystick Service* does not use the optional product model layer of the three-tiered architecture in the same manner as the aforementioned *CAD Service*. It simply sends a stream of joystick data to the UI component. The *Joystick Service* architecture is identical to the *CAD Service* architecture with a single *Client Service* object registering with the *Server Service* object. The UI component then uses the joystick data to manipulate the wheelchair in the facility. The *Joystick Client Service* object resides on the same machine as the joystick device. Again, following the distributed object paradigm, the joystick can manipulate the wheelchair in the UI from any system on the network but, in this case, would probably be most useful residing on the system.

Finally, Figure 3 includes the diagram for the collection of components aggregated to make up the path-planning service that generates the accessible route, a key component in disabled access analysis of a facility. The following section describes the motion-planning techniques developed to realize this particular service.

The Motion-planning-based Disabled Access Analysis Component

Given the description of the facility, the accessible route analysis attempts to generate an accessible route between two building components (for example, an entrance and a water closet). Motion-planning techniques naturally lend themselves to this type of analysis. Motion planning attempts to answer the question: How can a robot decide which motion to perform in order to achieve goal arrangements of physical objects (Latombe81)?

This analysis component uses motion-planning simulations that capture the geometric requirements of accessible route disabled access code provisions. Robot motion planning is a family of geometric search techniques in which an initial and a goal configuration of a robot is specified. The robot searches a configuration space generated from the robot's geometric and motion parameters and obstacles in the space to move from the initial to the goal configuration. The wheelchair and the wheelchair occupant are the robot that attempts to move through the design of a facility.

The signature of this movement mirrors "comfortable" wheelchair movement. The robot is restricted to move in three directions: right, straight ahead, and left, and there are two possible turning radii for right and left turn movements. The larger radius describes general wheelchair motion, and the motion planner uses the smaller radius for motion when the wheelchair comes within a certain distance of the goal point. The description of acceptable accessible route geometries in the *Americans with Disabilities Act Accessibility Guidelines* (ADAAG) document determine the value of the two turning radii (ADAAG97). Figure 4 shows the generated plan through an ADAAG-specified configuration. From this configuration, the larger turning radius was determined to be 24 inches.

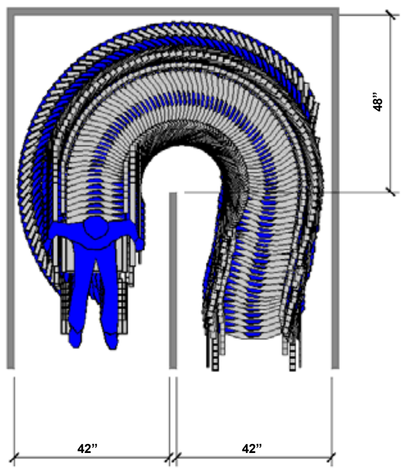

Figure 4: The generated accessible route using the larger turning radius

Figure 5 shows the generated plan using a 9-inch turning radius that allows the ADAAG-specified 60-inch turning circle. Combining these two radii creates an accessible route with the desired wheelchair behavior.

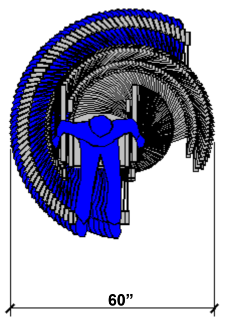

Figure 5: The generated accessible route using the smaller turning radius

Case Example

To validate the overall framework as well as the accessible route generating component, the research used as a case example an actual building on the Stanford University campus. The facility, modeled using a modified commercial CAD package successfully transferred the design data to the central UI component. From this UI component, the user is able to interactively manipulate a wheelchair through the virtual design using a joystick and determine the existence of an accessible route from one building component to another in the facility design. Figure 6 illustrates the case example and the generated accessible route between the entrance and the toilet in the Men’s bathroom.

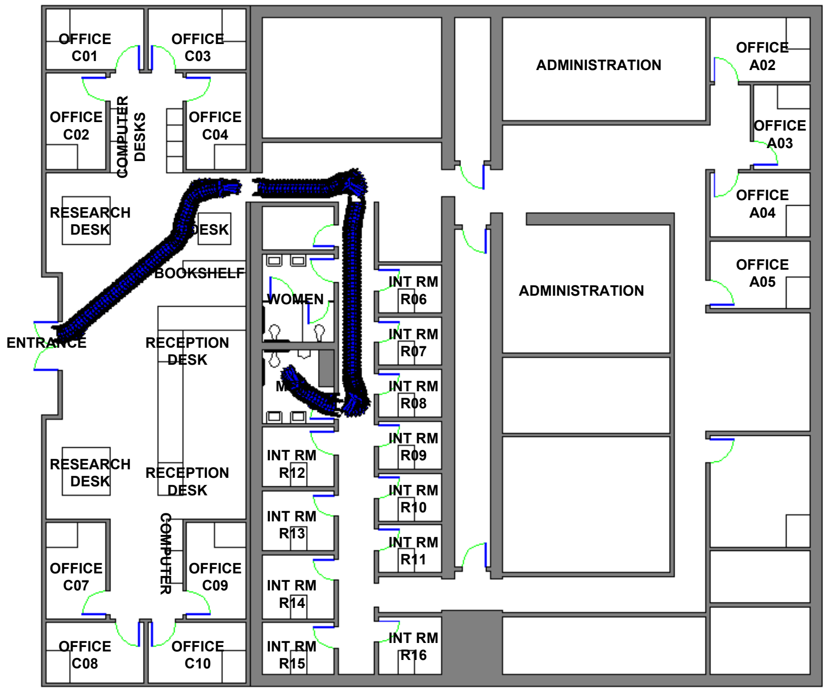

Figure 6: A generated accessible route through the case example

Discussion

This paper has presented a component-based approach to building large-scale engineering systems leveraging distributed object technology to enable general engineering analysis techniques. Specifically, the framework employs a three-tiered service architecture in which services can register with other services to aggregate and create larger components. The three-tiered architecture specifies a communication protocol among `Service` objects that supports the decomposition of an engineering problem, a product model protocol, and a space for the domain-specific analysis. As an example of a domain-specific analysis, the paper described the developed motion planning techniques used in a disabled access accessible route generating component of the framework. The techniques used attempt to directly capture actual wheelchair movement.

The paper illustrates the need for both general engineering problem-solving knowledge to develop the overall infrastructure and domain-specific knowledge to develop the individual components of the system. With the incorporation of other domain-specific analysis components, the power of this component-based approach can be realized.

Acknowledgements

This research is partially sponsored by the Center for Integrated Facility Engineering (CIFE) at Stanford University and the Intel Corporation.

References

(ADAAG97) Access Board (U.S. Architectural and Transportation Barriers Compliance Board) (1997). *Americans with Disabilities Act Accessibility Guide*, Washington, DC.

(Han99) Han, Charles S., J.C. Kunz, and K.H. Law (1999). "Building Design Services in a Distributed Architecture," *Journal of Computing in Civil Engineering*, ASCE, 13(1), 12-22.

(IAI97) *International Alliance for Interoperability (1997). Industry Foundation Classes Release 1.0, Specifications Volumes 1-4, Washington DC.*

(Latombe91) Latombe, Jean-Claude (1991). *Robot Motion Planning*, Kluwer Academic Publishers, Norwell, MA.