



**CIFE** CENTER FOR INTEGRATED FACILITY ENGINEERING

**A Model for Software Interoperation  
in  
Engineering Enterprise Integration**

By

Cameron T. Howie, Kincho H. Law and John C. Kunz

**CIFE Working Paper #61  
August, 2000**

**STANFORD UNIVERSITY**

**COPYRIGHT © 2000 BY**  
**Center for Integrated Facility Engineering**

If you would like to contact the authors, please write to:

*c/o CIFE, Civil and Environmental Engineering Dept.,  
Stanford University  
Terman Engineering Center  
Mail Code: 4020  
Stanford, CA 94305-4020*

# A MODEL FOR SOFTWARE INTEROPERATION IN ENGINEERING ENTERPRISE INTEGRATION

Cameron T. Howie, Kincho H. Law and John C. Kunz

Department of Civil and Environmental Engineering  
Center for Integrated Facility Engineering  
Stanford University  
Stanford, California 94305-4020

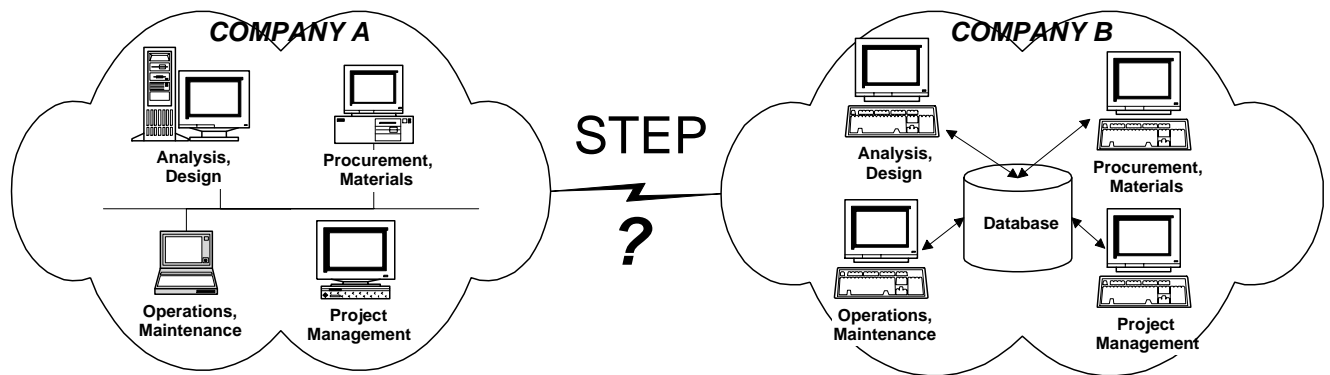
## **Abstract**

Engineering companies are seeking ways to achieve software interoperability between their internal applications and those of external vendors and customers. The emergence of open data exchange standards such as STEP, as well as the wide variety of possible implementation technologies, has led to our development of a model capable of supporting such software interoperation. Our model is independent of any specific implementation approach and protects proprietary knowledge of the software tools and data standards used within the participating organizations. It provides great flexibility in the configuration of software services and data exchange standards used. We have completed a formal description of the model and are now preparing to test it both locally at Stanford and with support from industrial corporations.

## **1 INTRODUCTION**

### **1.1 Motivation**

Leading engineering companies are seeking ways to integrate their internal computing systems ('intranets') with those of potential vendors and customers. The major problems confronting such integration stem from the challenges of exchanging and then possibly sharing data across the company boundaries. Present data exchange standards are inadequate for the industrial needs identified by leading companies, which has led to the emergence of the ISO standard 10303 [ISO 10303], commonly referred to as STEP – the STandard for the Exchange of Product data. This standard is being used in preliminary industrial projects aimed at building data exchange facilities between interested companies. Once such basic exchange facilities have been established, the participants will attempt to develop data sharing environments to allow greater



*Engineering companies are seeking ways to interconnect their proprietary information systems based on a neutral file exchange standard so that data sharing across company boundaries becomes possible.*

Figure 1: **Data Sharing between Companies**

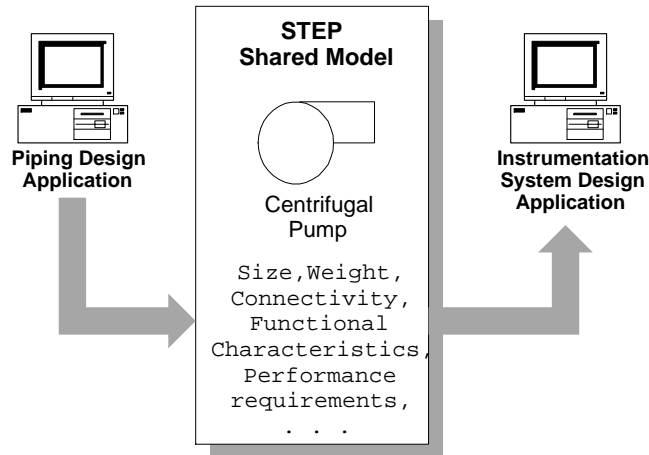
automation of information flow between them, an outcome that they have identified as being key to their business strategies in the years ahead. Figure 1 shows how such integration technology must link significantly different proprietary systems within the various companies.

As such companies are beginning to deal with the short-term problem of data exchange, there does not exist an interoperability model for interoperation of software applications among engineering organizations. The technological heterogeneity and business characteristics of these companies create the demand for an integration model that will both preserve proprietary information and yet achieve acceptable software integration. This paper describes the development of an interoperability model that provides the basis for such integrated environments and hence guides the engineering industries towards software interoperation via the exporting of software services across company boundaries. Our interoperability model will address more than data sharing in that software interoperability can be achieved without the need for the participants involved to actually share a data warehouse and data storage protocols.

Our initial focus of application will be the plant industries as this is one of the areas in which the development of an engineering exchange standard (introduced below) is more mature. While concentrating on this domain for development and testing purposes, we hope and expect that the model could eventually be applied broadly across a range of engineering disciplines.

## 1.2 The STEP Standard

As STEP is being developed for data exchange/archiving and sharing through a database approach, it does not, in the parlance of OMT (the Object Modeling Technique [Rumbaugh, 1991]), provide ‘dynamic’ and ‘functional’ models necessary for software interoperability in a computer network environment. STEP therefore provides an ‘object’ model only. An object model provides a framework for dynamic and functional models, but it only identifies the objects in its



*The open ‘STEP’ standard for engineering data exchange supports the sharing of ‘form’ and ‘functional’ data among software applications. Such a standard allows for complex data exchange content among interoperable software services across engineering company boundaries. However, as STEP is a data exchange standard, it lacks the dynamic and functional models necessary for such software interoperability.*

Figure 2: **STEP Shared Model**

domain and their inter-relationships, attributes and operations (also known as ‘methods’). By contrast, a dynamic model describes time and sequencing information and so defines control behavior. A functional model specifies how data flows through a system and how it may be transformed. However, OMT dynamic and functional models are defined with respect to an object model, so STEP’s object model cannot be used for interoperability – it is simply good as a data exchange standard.

Without an interoperability model, STEP-based exchange cannot penetrate the proprietary information systems within inter-connected engineering companies. Our interoperability model therefore assumes that the object models of open standards like STEP provide the content of information exchanged between companies over networks, but it will provide a complete object, dynamic, and functional model to enable software interoperability using such standards. Figure 2 shows how STEP supports sharing of ‘form and functional’ data among engineering applications, so providing extensive capabilities for exchanging engineering content among interoperable software services.

### 1.3 Related Research

We have reviewed a broad range of developments in the emerging field of software interoperability [Howie, 1996]. All the models, methodologies and technologies investigated are not well suited to inter-company software interoperation as they assume open availability of software components, data standards and databases without regard for the key proprietary investments

companies make in their software resources. Popular and emerging ‘middleware’ tools such as Remote Procedure Call (RPC), Object Linking and Embedding (OLE) and Common Object Request Broker Architecture (CORBA) aid developers in integrating remote object-oriented resources into applications, which presumes the owners of such resources are willing to allow them to be incorporated in new applications, an unlikely scenario for companies wishing to protect proprietary codes. Khedro et al [Khedro, 1993] developed an interoperability architecture based on the Knowledge Interchange Format [Neches, 1991] that demonstrated engineering applications sharing and subscribing to data exchanges defined in KIF partly because emerging standards like STEP were not readily available. Moreover, their open architecture was not based on interconnection of software agents across company boundaries and so lacked features for flexible network configuration and authentication, the provision of remote services, and project-specific software interoperation, features also lacking in the approaches described in [Fischer 1993], [Kunz, 1995], [Karagiannis, 1995], [Wileden, 1991], [Unisys, 1993], [Purtilo, 1994], [Pan, 1991], [Cutkosky, 1993], [Hardwick, 1996].

## 2 THE SOFTWARE INTEROPERABILITY MODEL

This section describes the principles and architecture of our interoperability model that make it suitable for connecting software services across engineering company boundaries.

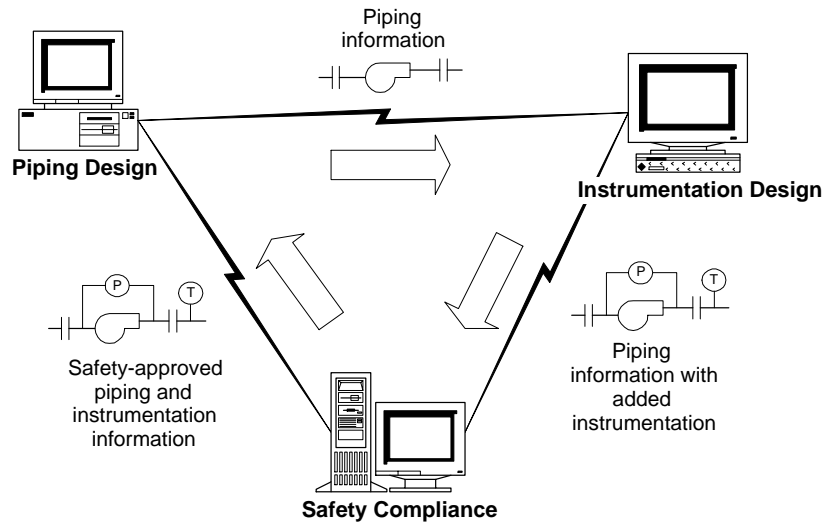
### 2.1 Underlying Principles

In contrast with other research efforts aimed at (or limited to) intra-company software interoperability, we are developing a new model for interoperation of computer programs among multiple companies, especially where the users and applications have distinct behaviors such as those in engineering industries. We assume that applications do not exchange behavioral models as these are considered to have no relevance beyond the applications themselves. Instead, only ‘form and functional’ data is exchanged, which is in keeping with traditional knowledge sharing in engineering industries (e.g., a piping system designer is not required to share his technical knowledge with an instrumentation vendor). Figure 3 shows a simple example of how these specific applications apply their behavioral knowledge to data exchanged among them.

The principle of this approach to software interoperability is shown in IDEF notation in Figure 4. The example data shows the adoption of the STEP standard as the interoperability framework and how a ‘piping’ design application has added form information (here it has determined a type of pump and piping diameter) to an input functional requirement (the need to pump material at 20 gallons per minute).

### 2.2 Major Obstacles to Software Interoperation

We have identified several major issues facing software interoperability in the engineering industries that may be unique to the problem at hand or may have received little attention in



*We assume that applications with specific behaviors, such as those in engineering industries, do not exchange behavioral knowledge with each other. Instead, they apply such behavior to data exchanged based on ‘form’ and ‘functional’ models.*

Figure 3: **Information Exchange Model**

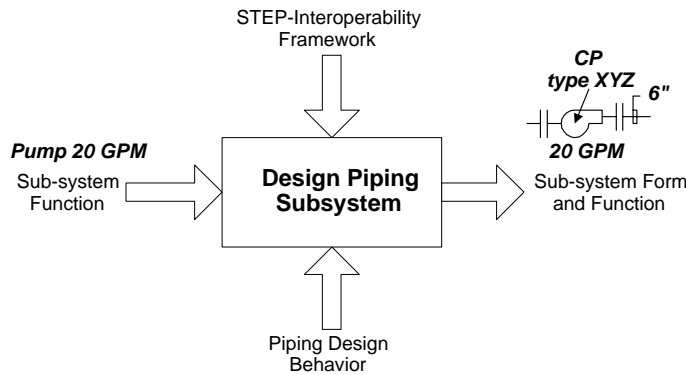
‘interoperability’ research. All of these issues arise from the proprietary and customized nature of engineering information systems. We have considered these in the development of our interoperability model.

### 2.2.1 Different ‘middleware’ technologies

Middleware is the term given to a range of products and services that shield applications and their developers from a multitude of network communication protocols and database programming interfaces. There are already many different middleware products emerging (more notable examples are Microsoft’s COM, the OMG’s CORBA, and the OSF’s DCE). As engineering enterprises adopt these middleware solutions, the heterogeneity of the resulting intranets will complicate attempts to achieve substantial and broad-ranging interoperability. Our model therefore is implementation independent in order to greatly simplify its acceptance in the industry by supporting existing software technologies without forcing any company to adopt a specific approach.

### 2.2.2 Customized software tools

It is standard practice within engineering companies, especially large organizations that can provide the necessary internal development resources, to optimize and customize off-the-shelf software products in order to achieve competitive advantages. Features particular to a company’s line of business will be encoded into the applications and the way in which they communicate, if



*Our approach to software interoperability is shown here in ‘IDEF’ notation. It illustrates an example where a piping application has added ‘form’ information to an input ‘functional’ requirement, based on the STEP data exchange framework.*

Figure 4: **Information Exchange with Form and Functional Descriptions**

any (since interoperability is only now starting to be addressed – in even the largest engineering companies). Software interoperability is certainly more difficult if companies display a penchant for modifying commercial products to their own tastes and requirements. For this reason we have proposed a ‘gateway’ approach to interoperability that considers the interconnected enterprises as significantly different (proprietary) computing environments, and so protects the intellectual property within their applications.

### 2.2.3 Customized inter-application data communication

In addition to the complication of having to support enterprise customization of software products and tools, inter- enterprise communication protocols must reflect the fact that many links between certain applications in an intranet (e.g., between a CAD system for plant design and a P&ID application) are customized solely for the two applications concerned. This arises from the simplicity for the developers working to link such programs, and because, until recently, most applications were not designed for wide-scale sharing of data. Furthermore, engineering applications typically share substantially less than 100% of their internal data structures. What data is actually published is often decided between the developers of the relevant applications. This sort of behavior complicates the sharing of information with external applications unaware of these proprietary arrangements. Therefore we have provided for both proprietary and open exchange standards when agents interoperate through the enterprise gateways. Existing arrangements can be maintained without forcing companies to adopt a single approach to interoperability.

### 2.2.4 Dynamic objects

Some companies are developing data exchange models based on object-oriented technology that supports dynamic definition of object attributes and relationships. As work on a project pro-



gresses, so an application creates new objects that are thus dynamic relative to the other applications sharing its data. While some middleware technologies (e.g., CORBA) support dynamic object definition, others do not. We have catered for this by providing a means for agents to share information by prior arrangement, thus allowing them to send each other dynamic objects that they themselves can interpret.

### **2.2.5 Project management and job control**

Engineering companies manage their design and analysis data exchanges within a particular job control and project environment. Coordination and sharing of engineering data among separate companies will have to map data exchanges to potentially different project management models. The issue of who owns and manages data in a shared environment is thus a legitimate issue for engineering organizations and one that has managers concerned about the complexities surrounding software interoperability among organizations.

## **2.3 Basic Model Architecture**

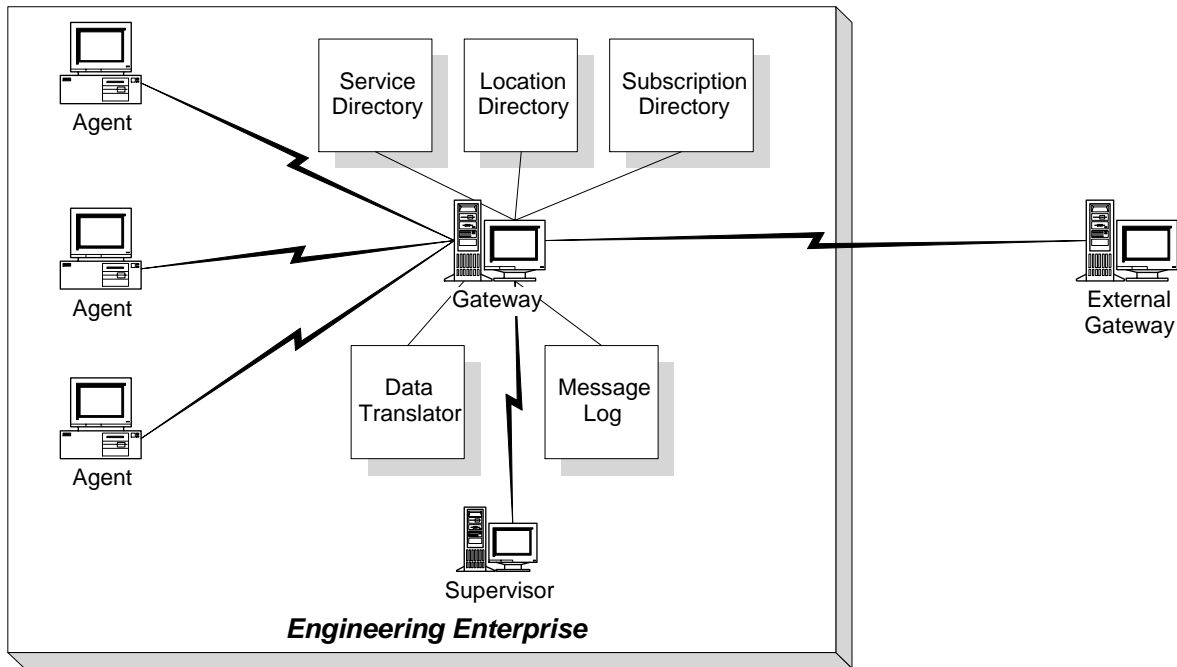
To date we have completed a prototype of the interoperability model and are currently preparing for testing. While testing may well lead to some architectural modifications, Figure 5 shows a network schematic of the current model whereafter the various components of the architecture are described in more detail.

### **2.3.1 Agents**

We define an agent to be a program that provides services for other agents, or a program that uses the services of other agents. The model uniquely identifies every agent by the instance triple: `<agent name, user name, project>` (abbreviated to `<A,U,P>`). This allows the same application and/or user to play different roles in the system on different projects. Agents connect to gateway computers through ‘proxies’ which are low-level stubs of code that handle the application programming interface for the network communication, thus greatly easing the process of implementing a gateway and agents within an organization. All proxies are configured to use the enterprise gateway i.e., they know its network address and the corresponding network communications protocol.

### **2.3.2 Gateways**

A gateway is a server that communicates with agent proxies within an organization and with gateways established at other participating companies, each company implementing and maintaining a single gateway. Gateways thus enable the interconnection of agents across company boundaries. A gateway can manage multiple projects (each configured differently) through the use of various system tables discussed below. Figure 6 illustrates a formal description (using the OMT) of the ‘gateway’ component.



*Agents (i.e., software applications) connect with other agents through gateways. Gateways are used to manage intercommunication and to protect proprietary information about the agents when they export services outside the company to agents in other companies.*

Figure 5: **Architectural Diagram of Software Interoperability Model**

### 2.3.3 Communication Messages

Communication channels are set up between agents and their company gateway and between the gateways of the integrated organizations. Messages between agents and their local gateway are called ‘internal’ messages and are supported distinctly from external ones (that are used in gateway-gateway communication). Internal messaging enables the use of an internal/proprietary network packet structure for legacy or strategic purposes. By comparison, external messages must use a packet structure comprising a basic data model (e.g., a STEP Part 21 file) and an ‘appendix’. The data model will comply with the standard associated with the input/output parameters of the invoked services (see Section 2.3.6), while the ‘appendix’ is provided for customized usage between agents interconnected across the enterprise boundary. Messages are further divided into ‘control’ and ‘data’ messages depending on the nature of the communication.

### 2.3.4 The Location Directory

The location directory for an organization maintains one or more location tables, each location table corresponding to a single project. A location table holds network location information for all agents and gateways. It declares nodes on the network to be agents or gateways, and what

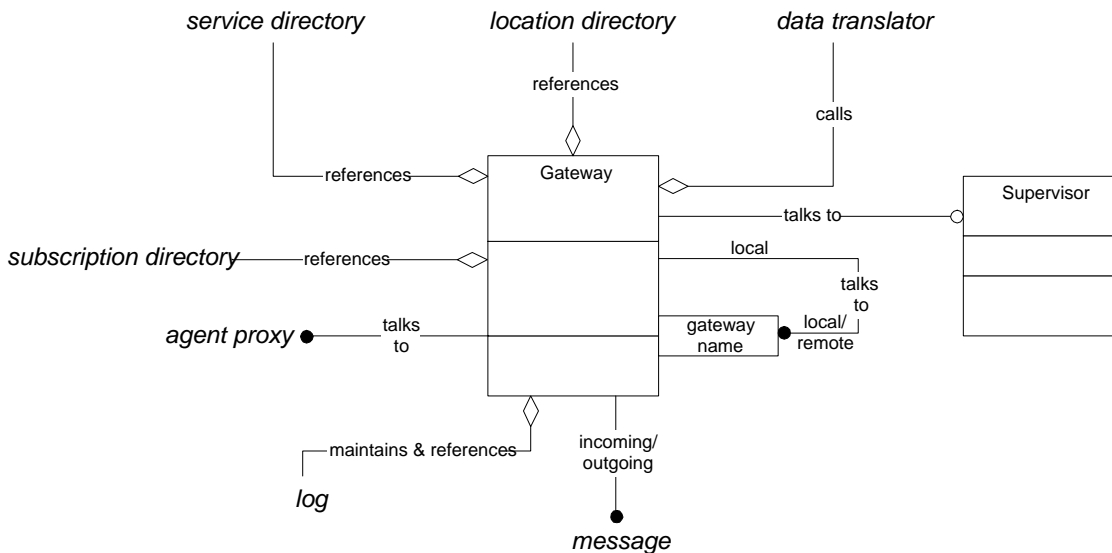


Figure 6: **The Gateway-Based Model in The OMT Format**

communications protocols and network addresses are used to reach them. This allows agents to connect to the system using any supported network protocol from anywhere i.e., network addresses are resolved dynamically when agents are launched and register with their company gateway. Agents are identified by global names allocated by the company, a name being assigned to each  $\langle A, U, P \rangle$  triple. This allows a one-to-many workload mapping, e.g., the name ‘Manager’ can be bound to the triple  $\langle \text{agent} = \text{manage.exe}, \text{user} = \text{Bob}, \text{project} = \text{accounts} \rangle$  and also to  $\langle \text{agent} = \text{fin.exe}, \text{user} = \text{John}, \text{project} = \text{finance} \rangle$ , so whenever either of these agents is up and running, the enterprise gateway will deem there to be an instance of the ‘Manager’ agent present. To protect proprietary information, external gateways are simply given a name, network address, and network protocol. e.g., name = ABC\_CORPORATION\_GATE, address = 12.34.56.78, protocol = TCP/IP. Any agents running behind external gateways are reached through their global names – no outsider can determine what network technology the remote company is using, what the user names are, or what applications they are using.

### 2.3.5 Data Translator

A data translator is a company-implemented service that translates actual objects between internal and (optionally) open standard formats by referring to a data translation directory. For ease of implementation, the translator would normally be coded into the gateway service as opposed to running as a separate system process. Translation directories manage one or more data translation tables that map input data types to those of the output standard. A different table is used for each project, thus enabling project-specific data types. A translation map specifies the name and data types of the objects required for data translation. For example, if it takes an object of type A defined by standard S to produce the equivalent two objects B and C

in standard T, then a map  $S/A \rightarrow T/B,C$  would be specified for translating instances of object A into instances of B and C. The translator, when given an instance of A can then output the equivalent B and C instances. Input data types may belong to a proprietary standard while the output data would normally be defined using an open standard. As the map specifies the names of the standards used, it allows multiple standards to be used on a single project. Companies implement their own translators as this protects any proprietary data standards they wish to use. Using an open standard for output data is optional as companies may prefer for strategic or legacy reasons to use prior data exchange standards in establishing interoperation with certain client companies.

### **2.3.6 Service Directory**

The service directory describes the list of software services that agents make available to one another on projects. A service is a function performed by an identified agent. The agent is identified using its global name defined in the location directory. Each service is listed with a brief description and the input and output parameters, if any, it requires. All parameters specified must be listed in the data translation table for the project so that gateways can ensure correct data translation when managing service invocations among agents.

A service table is constructed for each project and defines services offered by internal and/or external agents. Defining one table per project allows agents to respond differently depending on the project name. For example, within an organization an agent can provide a ‘full capability’ service based on an internal project name, but for external uses a company could offer a ‘limited capability’ service through that same agent under a different project name, thus keeping a competitive advantage with its internal software but at the same time being able to offer services to external clients.

### **2.3.7 Subscription Directory**

A subscription directory comprises one or more tables that define subscription interests for agents and supervisors on a per-project basis. Each entry in a table identifies a list of internal and/or external agents that are subscribing to a given project service. Each subscriber is identified by its  $\langle A,U,P \rangle$  triple.

### **2.3.8 Administration Features**

A gateway logs all communication messages for administration purposes, e.g., an accounting facility or work flow analysis. Logging might also be used to implement stable transaction management in the network environment. In addition, a ‘supervisor’ service is offered to which the enterprise gateway can connect if additional management of intercommunicating agents is required. For example, a company might want to ‘hook in’ a connection to a document tracking system so that certain data transmitted among agents can be copied into that system’s environment for additional processing. A supervisor is optional for any intranet configuration and, while only one supervisor ought to be sufficient, several could be supported. A gateway will

only update its supervisor based on the output of invoked services. If the supervisor requires additional information e.g., a list of agents currently registered, then it must query the gateway explicitly. As companies will typically implement their own gateways, such arrangements can be tailored to the needs of an organization.

### 3 INTEROPERABILITY SCENARIOS

We have identified six scenarios for interoperation among software agents that this proposed model will support:

- *Scenario 1: create a new data instance.* Here inter-application communication results in the generation of a new data within an organization. An example of this scenario is ‘procurement’: an engineer sitting at a CAD station downloads a design component from an external vendor providing a component selection service.
- *Scenario 2: load a data instance.* This interoperability scenario occurs when an agent loads in a data model from a database. This database might be managed by another company. For example, an engineer might load an HVAC model into a CAD application.
- *Scenario 3: change a data instance.* This interoperability scenario results in an alteration to a possible remote database. For example, an agent saves a design file to a project database.
- *Scenario 4: transfer a data instance.* This occurs when one agent exports a data model to another. The communication between the safety and piping agents in Figure 3 is an example of this.
- *Scenario 5: make concurrent changes to data instances.* This scenario is defined by two or more agents concurrently writing exported data into a shared database. An example is a concurrent design environment where an HVAC agent is executing alongside a structural steel agent in the concurrent design of a process facility and both are writing data into a shared database.
- *Scenario 6: make concurrent changes and load data.* This scenario arises when agents are concurrently importing data from and changing a shared data model. This can arise, for example, when one or more agents are accessing a model, say for quantity take-offs and construction planning, as it is being altered by another agent (perhaps the designer is making some changes).

The prototype is being developed using the TCP/IP networking protocol on Unix and Windows platforms. To test the model, we shall initially focus on the first four interoperability scenarios. We are in the process of testing the behavior of the different scenarios for real world engineering applications such as those used by industrial companies and those available to us at Stanford. Due to implementation complexities, any data exchanged among the applications will

not involve actual translation into and out of standards such as STEP. Instead all data models used for testing will be kept simple as the focus of the research is interoperability and not the development of a neutral data exchange specification.

To illustrate the software interoperation made possible by the model, Figure 7 shows an example of a test scenario where an engineer has used an external network service to select a valve component for insertion into a plant design model (i.e. interoperability scenario 1). The component request between the engineer's application and the remote component server is handled by the intermediate gateways in communication with the application and server agent proxies. The information for the component is passed back to the engineering application by using the STEP data standard in the open network environment. Other applications sharing valve component information can also be updated once the component transfer arrives at the initiating gateway. This example shows how software interoperability can be achieved without the need to export any proprietary information, and without regard to the internal protocols and topologies of the participating companies.

## 4 SUMMARY

We are developing an implementation independent software interoperability model that enables software interoperation across engineering companies through the provision of inter-company software services. It supports proprietary and open standards for data exchange in enterprise integration and is based on the principles of a software interoperability in which behavioral knowledge is contained within the interconnected applications. It protects the network configuration and identification of software tools and users within participating organizations, and allows for additional supervision through links to other company systems. Presently, we have permission from an industry company to link software between Stanford and their organization. This experiment will allow us to evaluate the strengths and weaknesses of the proposed model for practical application in engineering enterprise integration. The model will be assessed on its ability to support several interoperability scenarios.

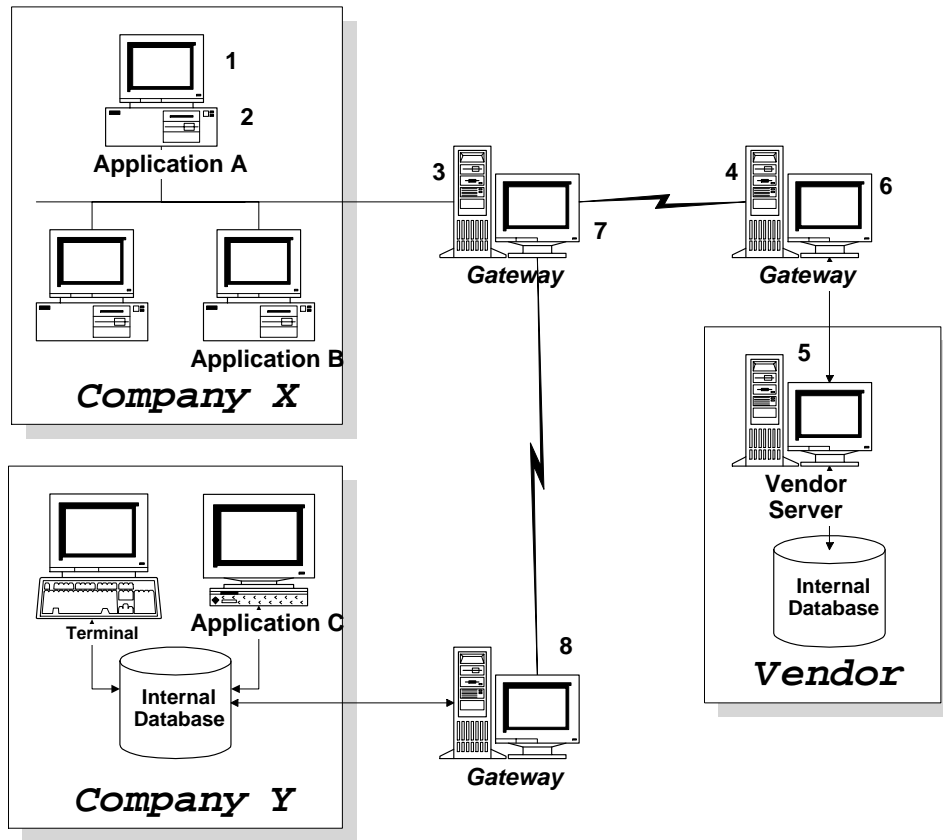
## 5 ACKNOWLEDGEMENTS

This research is sponsored by the Center for Integrated Facility Engineering at Stanford University. Discussions with engineers at Bechtel Corporation and other industrial affiliates have greatly facilitated the development of the software interoperability model described herein and the practical requirements related to the implementation of such a model.

## References

[Cutkosky, 1993] Cutkosky, M.R., et al (1993), *PACT: An Experiment in Integrating Concurrent Engineering Systems*, Computer Vol 26(1) pp 28-37.

- [Fischer 1993] Fischer, M., Kunz, J.C. (1993), *Circle Integration*, CIFE Working Paper #20, April, CIFE, Stanford University.
- [Hardwick, 1996] Hardwick, M., Spooner, D.L., Rando, T., Morris, K.C. (1996), *Sharing Manufacturing Information in Virtual Enterprises*, Communications of the ACM, 39(2), pp 46-54.
- [Howie, 1996] Howie, C., Kunz, J., Law, K. (1996), *Software Interoperability*, Kaman Sciences Corporation.
- [ISO 10303] International Organization for Standardization, *ISO 10303 Industrial Automation Systems and Integration – Product data representation and exchange – Part 1 Overview and fundamental principles*, ISO TC184/SC4/PMAG.
- [Karagiannis, 1995] Karagiannis, D., Marinos, L. (1995), *Integrating Engineering Applications via Loosely Coupled Techniques: A Knowledge-Based Approach*, Expert Systems With Applications, Vol 8, No 2, pp 303-319.
- [Khedro, 1993] Khedro, T., Genesereth, M.R., Teicholz, P.M. (1993), *Agent-Based Framework for Integrated Facility Engineering*, Engineering With Computers, Vol 9 pp 94-107.
- [Kunz, 1995] Kunz, J.C., Jin, Y., Levitt, R.E., Lin, S-D, Teicholz, P.M. (1995), *The Intelligent Real-Time Maintenance Management (IRTMM) System: Support for Integrated Value-Based Maintenance Planning*, CIFE Technical Report #100, January, CIFE, Stanford University.
- [Neches, 1991] Neches, R., Fikes, R., et al. (1991), *Enabling Technology for Knowledge Sharing*, AI Magazine 12(3), pp 36-56.
- [Pan, 1991] Pan, J.Y., Tenenbaum, J.M. (1991), *An Intelligent Agent Framework for Enterprise Information*, IEEE Transactions on Systems, Man, and Cybernetics, Vol 21 No 6, pp 1391-1408.
- [Purtilo, 1994] Purtilo, J.M. (1994), *The POLYLITH Software Bus*, ACM Transactions of Programming Languages and Systems, Vol 16 No 1 pp 151-174, January.
- [Rumbaugh, 1991] Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., Lorenson, W. (1991), *Object-oriented Modeling and Design*, Prentice-Hall, Inc..
- [Unisys, 1993] Software Technology for Adaptable, Reliable Systems (STARS), *STARS Conceptual Framework for Reuse Processes (CFRP)* Volumes I,II (1993), Unisys STARS Technical Report STARS-VC-A018/001/00, Advanced Research Projects Agency (ARPA) STARS Technology Center, 801 N. Randolph St. Suite 400, Arlington VA 22203, October.
- [Wileden, 1991] Wileden, J.C., Wolf, A.L., Rosenblatt, W.R., Tarr, P.L. (1991), *Specification-Level Interoperability*, Communications of the ACM, Vol 34 No 5 pp 72-87.



(1) An engineer at company 'X' using application 'A' locates a valve supplier, perhaps using internet technology. (2) A request for a valve component is made. (3) The gateway acts as the proxy for the request and connects with the gateway at the vendor's site. (4) The second gateway translates the request into one compatible with the vendor's internal system. (5) The vendor's server locates the necessary component data and returns it to the gateway which in turn (6) maps it into STEP format and returns it to the first gateway. (7) The first gateway sends the incoming valve data to application 'A'. However, since application 'B' is sharing valve data with 'A', it is also updated. the gateway also contacts its peer at company 'Y' to perform a similar update (8) as application 'C' is also sharing valve data with applications 'A' and 'B'.

Figure 7: Example of Software Interoperability Scenario 1