Advanced Materials Research Vols. 44-46 (2008) pp 717-724
online at http://www.scientific.net

A Service-Oriented Collaborative Design Platform for Concurrent Engineering

W.S. Xu[1, a], J.Z. Cha[1,b] and M. Sobolewski[2,c]

[1] School of Mechanical, Electronic and Control Engineering, Beijing Jiaotong University, Beijing, China

[2] Computer Science, Texas Tech University, Lubbock, Texas, U.S.A

[a]wshxu@bjtu.edu.cn, [b]jzcha@bjtu.edu.cn, [c]sobol@cs.ttu.edu

Keywords: Collaborative Design, Concurrent Engineering, Service Oriented Architecture, Service Oriented Programming

Abstract. An important requirement for a collaborative design platform in Concurrent Engineering (CE) is the integration of various engineering software tools and utilities in product design and development. Some CE platforms based on a client/server architecture or static Service Oriented Architecture (SOA) are available in the marketplace, but they lack flexibility and reliability in the constantly changing Internet environment due to the dynamic nature of the network. Based on the current development of SOA, this paper presents a Service-oriented Collaborative Design platform (SCoD) based on SORCER—a dynamic SOA infrastructure that allows federated integration of engineering software components in CE environments. The architecture of SCoD is proposed, the wrapping methodology used to integrate engineering software tools in SCoD is presented, and the federated method invocation for services in SCoD is described. With the support of SCoD, collaborative design in CE environments can be deployed, and scalability, reliability, and flexibility can be achieved in the changing Internet environment.

Introduction

Concurrent Engineering (CE) is a systematic approach to the integrated, concurrent design of products and their related processes, including manufacture and support [1]. Various tasks are involved in the collaborative design and they may be carried out by different developers with various software tools. However, these tasks may interact with each other, and may be performed iteratively according to other tasks' feedback and flowdown. Also, developers and software tools involved in the collaborative design may even be distributed in different physical locations. Thus to implement a distributed collaborative design, a CE platform is needed to facilitate the communication, coordination and cooperation of the various tasks, software tools, and team members in the CE product development process. Actually, many research and development efforts have been done in this field, and there are a few commercial platforms available in the marketplace that support CE over the network. But since the network is inherently a dynamic environment, without fully considering the inherent changing nature of the network, the CE platforms will lack scalability, flexibility, and reliability. On the other hand, in recent years, research work on Service-Oriented Architecture (SOA) has progressed steadily from static SOA to dynamic SOA, and new results have been accomplished in this area recently. SOA can provide a flexible infrastructure for grid computing on the top of the dynamic network, for example the SORCER (Service ORiented Computing EnviRonment) system defines a dynamic SOA meta-computing infrastructure [2]. By combining the current development in SOA and concurrent engineering, we have developed a flexible CE platform—a Service-oriented Collaborative Design platform (SCoD) based on SORCER. With this platform, various software tools and utilities can be easily integrated as loosely coupled services that form a dynamic service federation for a specific design process when requested. The SCoD platform provides reusability, scalability, reliability, and efficiency that can be achieved by the service-oriented programming and relevant dynamic SOA infrastructure.

Overview of a CE Platform and Dynamic SOA

A Distributed CE Platform. In a CE multifunctional team, there are usually multiple tasks which are interrelated closely. For example, for mechanical product development, a team may be responsible for the design of a mechanical part of the product. The design job could comprise multiple tasks: geometric model design, static stress analysis and dynamic stress analysis. And the static stress analysis task can be further decomposed into 3 tasks—mesh generation, Finite Element Analysis (FEA) computation, and human expert decision making. In the team, after a mechanical modeling engineer finishes the geometric modeling of the part, a FEA expert will generate a computation mesh for it and then perform FEA computation on the mesh for static stress analysis. Then the FEA expert will access and analyze the FEA computation results, and make a decision whether the static strength of the part is acceptable and whether some parameters of the part should be modified. If a design modification is needed, the mechanical modeling engineer will modify the geometric model of the part accordingly, then the model will undergo mesh generation, FEA computation, and FEA expert decision making once again, as shown in Fig. 1. This iteration will continue to repeat until the static strength of the part is satisfying and the FEA expert approves the design. But this is not the end of story—since the part design should further undergo dynamic strength analysis, the parameters of the part geometric model might be further modified until dynamic strength analysis is satisfying. If the part geometric model is changed according to its dynamic strength analysis results, then it should undergo the static strength analysis once again to make sure the static strength requirement is not violated. In reality an optimization task with the support of some optimization software package may exist in the design process and provide recommendations for model changes. If more types of engineering analysis are needed, such as thermodynamic analysis, etc., then the whole design process can be very complex. As we can imagine, without a relevant support environment for CE, the concurrent product development can be a very time-consuming process. Therefore a CE platform is needed to integrate various engineering software tools with the iteratively executed tasks predominantly with short frequent loops, eliminating longer loops as much as possible. The platform should provide a flexible and user friendly environment for teamwork on the network as well.

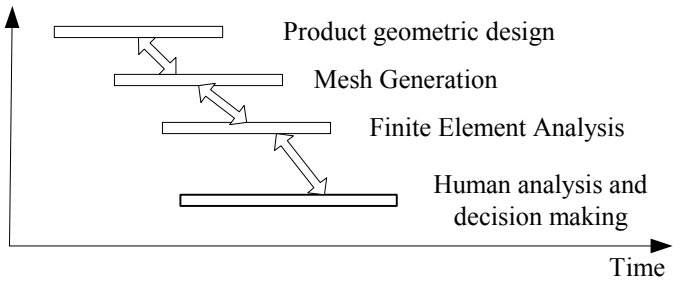

Fig. 1 Static stress analysis in a CE product development

Since the CE platform is built on top of extranet or intranet as a distributed application, the inherent dynamic nature of the network needs to be considered for reliability and flexibility of the platform. System architecture determines how computing software resources are connected in the CE platform, for example by what means they can be invoked by each other. A Remote Procedural Call (RPC) is one of the basic invocation means for constructing distributed applications. Currently there are different types of RPC, including Microsoft DCOM, CORBA [3], Java RMI, and Web/OGSA services [4, 5], etc. Based on these RPC techniques, various CE support platforms are available in the marketplace, and even more prototype systems are available in the academic field. These systems are either based on a client/server infrastructure, or based on static SOA infrastructure, which lack reliability and flexibility in the real dynamic Internet environment.

Since the inherent weaknesses of the client/server infrastructure and static SOA infrastructure in the dynamic Internet environment, dynamic SOA evolves from static SOA, where requestors do not need to know locations of providers and their communication protocols as well. A CE platform based

on dynamic SOA infrastructure can greatly improve its scalability, reliability, and flexibility in the dynamic network environment.

Dynamic SOA. In SOA, individual software components (including engineering data, tools, applications, and utilities) are packaged as services and can be distributed over a network. Low dependency between loosely coupled services can improve the scalability and reliability of complex, distributed, and collaborative engineering applications. Three generic roles can be distinguished in a SOA environment: service providers, service requestors, and service registries.

There are basically two types of SOA according to the requirements on service location and communication protocols. If requestors know the location of required services beforehand, and the communication protocol service providers implement, then this type of the SOA model is called static. For example, Web and OGSA Services [4,5] can be identified as static SOA services. If requestors do not need to know the location of the services and communication protocol beforehand, then the underlying SOA model is called dynamic. For example, Jini service-oriented architecture [6] can be identified as implementing dynamic SOA. Dynamic SOA is protocol neutral, and the location of the service also does not need to be known by the requestor beforehand. Even the location of the registries does not need to be known to requestors and providers beforehand, and providers and requestors can dynamically discover and join registries on the network.

The SORCER environment developed at the SORCER Lab, Texas Tech University is a federated service-to-service (S2S) metacomputing environment [2], in which not only the location of the service providers is dynamic, also the communication protocol is neutral, and the location of the registries is dynamic as well. In SORCER services can dynamically form federations for service-oriented programs submitted by requestors. The SORCER infrastructure is based on Jini programming model with explicit leases, distributed events, transactions, and discovery/join protocols. While Jini focuses on service management in a networked environment, SORCER is focused on service-oriented programming and the execution environment for service-oriented programs (in fact metaprograms). The dynamic architectural features and exertion-oriented programming [2] in SORCER make it a relevant infrastructure for constructing CE platforms in the Internet environment, where scalability, reliability and flexibility are critical to distributed applications.

The Architecture of the Service-oriented Collaborative Environment

The Architecture of the SCoD Platform. A CE platform should provide a distributed environment for collaborative product development spanning across several different departments in a company or even several collaborating companies. This platform should provide an integration framework to link various CAD (Computer Aided Design) and CAE (Computer Aided Engineering) tools and provide a means to invoke these software resources remotely at runtime. Furthermore, this environment should be able to dynamically organize the software resources and provide a federated environment for different projects, so the CAD and CAE software resources can dynamically participate in different project jobs. Since the CE platform may span across the Internet, the eight fallacies of the network should be taken into account and dealt with properly [7]. According to these requirements imposed on the CE environment, the architecture of the CE platform proposed here, which is called SCoD (Service-oriented Collaborative Design platform), is based on the SORCER infrastructure and is a service-oriented CE environment, as shown in Fig. 2. The architecture of the SCoD platform can be divided into five layers.

a. The network layer provides the basic network connectivity and processors for all software systems to run above. Please note that the network layer may be dynamic and suffer from temporary failures, and latency for remote invocations may exist in this layer.

b. The SORCER infrastructure layer provides the basic mechanism for implementing a dynamic SOA. The SORCER infrastructure components in this layer include Jini lookup services, which are service registries, and ServiceProviders and ServiceTaskers, etc., which are responsible for

implementing service providers in dynamic SOA. With these components, software tools can be integrated into the environment as service providers.

c. The domain-specific service layer provides all sorts of CAD and CAE services for CE projects based on the SORCER infrastructure layer. Actually, other CE enabling tools, such as CAPP (Computer Aided Process Planning), DFA (Design for Assembly), DFM (Design for Manufacturing) services, system simulation, etc., can also be integrated as services into this layer. Various SORCER wrappers might be developed to integrate various software tools respectively with this environment. Each service provider in this layer is a basic element in a CE application and can be dynamically incorporated into a CE application.

d. The resource management layer actually consists of two services: an aggregation service and a federation service. For the aggregation service, several services in the domain-specific service layer may simply be aggregated and integrated to form a new facade service—a single entry point into the system, so service requestors can invoke this facade service instead of managing themselves multiple service invocations one by one. For the federation service, based on the service jobs and tasks defined in applications in the CE application layer, multiple services in the domain-specific service layer can be dynamically selected and chosen according to the service signatures in the service jobs and tasks to form a required service federation. The services in the federation can perform specific service jobs and then can leave the federation after the completion of the jobs and then join new service federations. The SORCER Jobber service may be used here to dynamically integrate multiple services. The resource management layer is a key connection between the domain-specific service layer and the application layer discussed below.

e. The application layer is composed of end requestor applications for specific CE project tasks and jobs. These CE requestor applications are required to perform certain CE project tasks and jobs so they are created through exertion-oriented programming. Exertions [2] of ServiceJob and ServiceTask types are the key components in the service-oriented applications and they will subsequently invoke relevant services in the domain-specific service layer through the SCoD resource management layer.

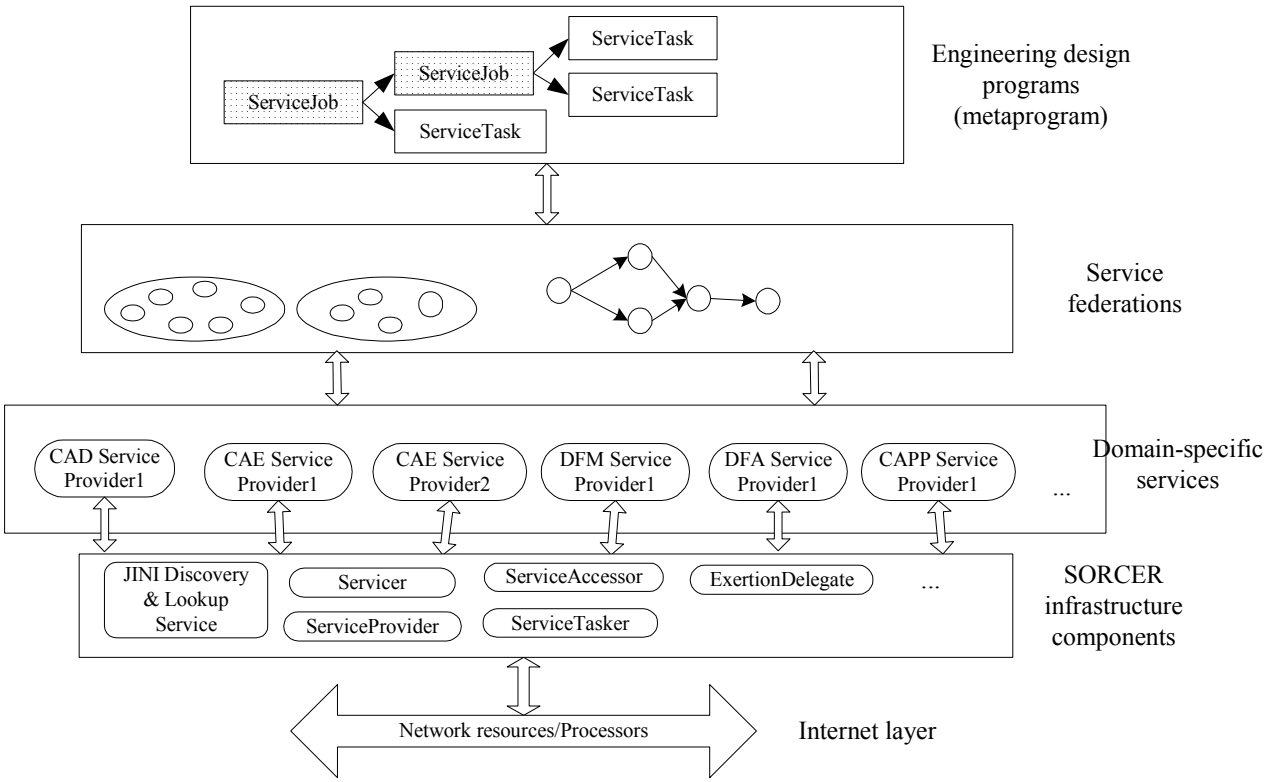

Fig. 2 The architecture of the service-oriented collaborative design platform

Wrapping CAD and CAE Software Tools as Services in SCoD. As discussed above, in the domain-specific service layer in SCoD, various CAD and CAE software tools are integrated in the platform as services. Currently most commercial CAD and CAE software tools are not service-oriented "ready" when they were developed and released, so we need to provide service wrappers for CAD and CAE software tools to be integrated into SCoD as service providers and then expose them to requestors on the network. Using ServiceTasker—a key service class in SORCER, we can wrap various legacy software applications, tools, and utilities as services.

As shown in Fig. 1, four example services could be identified in the static stress analysis job in the product development process: a Pro/E service for modeling a mechanical part, a HyperMesh service for meshing the geometric model, an Ansys service for FEA computation, and a human expert service for human decision making based on the FEA results and some relevant national and international standards. To support this job in SCoD, based on the class ServiceTasker in SORCER, four service providers have been developed in SCoD.

There could be two types of service providers in SCoD environment: one is autonomic service provider, which can run in the server mode or batch mode and needs no human involvement; the other one is human-involved service, which needs human support.

a. Ansys service. By defining an AnsysRemote interface and implementing it, and by extending the ServiceTasker class, an Ansys service provider can be deployed in SCoD. Ansys is widely used in interactive mode, but here it is wrapped as a typical autonomic service provider and run in batch mode. When running in batch mode, the input for Ansys is an APDL (Ansys Parametric Design Language) file which is a command flow, and Ansys processes it on its command flow and then produces several output files including an Ansys result file, an Ansys database file, a log file, etc. The Ansys service wrapper is responsible for defining the URL links to the input file and the output files as context nodes in the service context [8, 9] being passed to and from the service provider. The Ansys service provider can be invoked as a standalone service, or as a service participating in a job federation. In the latter case, all it does is to execute the exertions, which match the Ansys service signature (at least an interface type and operation). More specifically, when an exertion matches the Ansys service signature, the Ansys service provider will first receive the service context from its invoker and extract the APDL file link from the service context, then will download the APDL file to a local location, and invoke Ansys to process this APDL file. After Ansys finishes the APDL file processing and produces several output files, the Ansys service provider will move these output files to the web server location accessible to the requestor, and then write the URL links to these web files into the context nodes in the service context and return the service context to the invoker. The invoker can be for example a service browser, or a standalone requestor. Or, in the case of a federating service job, the invoker is a SORCER Jobber, which will further forward the service context to the next service provider in the job federation. In this way, information can be passed and shared among different service providers under Jobber's dynamic management.

b. Human expert service. As the name implies, the human expert service is an expert human-involved service, as human know-how plays important roles in the FEA decision making. The human expert service provider is developed by extending the ServiceTasker class, and its main functions are to receive the service context, produce a use agent which can interact with a human expert and display all relevant information contained in the service context, and then allow the expert to enter and submit his or her decisions using the friendly graphic user interface and then return the service context back to its invoker.

c. HyperMesh service. By extending the ServiceTasker class, a HyperMesh service provider has been developed in SCoD. Here the service is deployed as a human-involved service, where the geometric model of a product is often too complex and human modifications to the automatically generated mesh are necessary. The input to the HyperMesh service is the geometric model generated by a geometric modeling service, for example, the Pro/E service, and its output is a mesh file of this geometric model which can be further processed by an FEA service, for example, the Ansys service.

d. Pro/E service. There are two types of Pro/E services. The first is autonomic Pro/E service; the

second one is human-involved Pro/E service. Both Pro/E service providers are based on the ServiceTasker class and can process the service context passed from the invoker. The input to the Pro/E service includes the parameters of a geometric model, which are wrapped in the context nodes of the service context, and the output is the modified or generated geometric model, and the URL link to this geometric model is provided as a context node in the service context and passed back to the invoker. The autonomic Pro/E service provider can receive the parameters of the geometric model and modify or generate the geometric model of a product automatically, without human involvement. The human-involved Pro/E service provider can also receive the parameters of the geometric model, but the expert will then start up a Pro/E interactive application to work on, and based on all these parameters will generate the new geometric model for the product. When the engineer submits his task, the Pro/E service provider will wrap the URL link to this geometric model file as a context node in the service context and pass the service context back to the invoker. From the requestor's point of view, basically there is no essential difference between the autonomic and human-involved Pro/E services, only except that the autonomic Pro/E service can help construct a fully automatic design environment for some design tasks.

Invocations of the Services in SCoD. As illustrated in Fig. 1, the static stress analysis job is composed of four basic steps, which involves Pro/E geometric modeling service, HyperMesh meshing service, Ansys FEA service, and a human expert decision-making service. In a procedural programming approach, we should invoke the four services one by one as needed, and we should know the specific locations of each service. But in the SCoD platform, there could be replicated services for the same task in the environment, for example, there could be several Pro/E services and several Ansys services in the environment, but we do not need to know the locations of the services in the network. In the application layer, an application requesting services in SCoD is based on federated method invocation [2] used by exertion-oriented programming.

In the SORCER environment, a remote request, which is called a service exertion, can be sent to a service provider, and the service provider can then perform an activity (collaboration) according to the exertion type [2, 8, 10]. Exertions are the basic means of communicating with service providers and passing control to service providers. An exertion encapsulates a triplet: service signatures, a service context, and control strategy. Service signatures specify the operations to be performed on the service context that contains a tree-like data structure which stores multiple data items in the leaf nodes. The service data is processed by the operations specified in the service signatures according to the provided exertion's control strategy. Upon creating an exertion and executing it in a client application, the exertion will federate and invoke all needed service providers dynamically as specified by the signature in the exertion. There are two types of basic exertions in SORCER: ServiceTask and ServiceJob and control exertions as well. A ServiceTask is an elementary exertion, which can bind to a single service provider or a small federation processing the same context, while a ServiceJob is a composite exertion which can contain multiple ServiceTasks and even other nested ServiceJobs and can bind to multiple service providers, usually a large-scale federation. For a top-level ServiceJob, the shared data between the component ServiceTasks and ServiceJobs can be defined by their input-output mapping relationships [2]. When the top-level exertion is complete, the results are combined by the service provider binding to the top level exertion and the client application can have the results through the service context in the returned-back processed exertion. In this way, SORCER service providers and requestors can communicate via exertion passing—e.g., ServiceTasks and ServiceJobs. Therefore, programming in the SCoD application layer is mainly about writing and invoking exertions for specific applications.

Service Coordination in SCoD. As discussed above, a service provider can be discovered at runtime and invoked to execute an elementary exertion—a ServiceTask. A federation of service providers for a specific CE job can be initiated through executing a composite exertion—a ServiceJob. When multiple service providers are involved in CE job execution, the coordination service is needed for coordinating participating services, so the service providers can work in a collaborative way as specified by job exertions. A Jobber service in the SORCER environment is used

for this purpose here. A Jobber interprets and executes a service job's control context in terms of the job's nested exertions, and manages a federation of required service providers for component service tasks and jobs in runtime, and maintains a shared context for the job federation. Also, it provides a substitution for input context parameters, so service providers can work sequentially or in parallel as desired.

For example, to perform the CE job illustrated in Fig. 1, four tasks are needed: a task for Pro/E geometry modeling, a task for HyperMesh mesh generation, a task for Ansys FEA computation and a task for human expert decision making. The required invocation sequence of the related entities for one cycle of the iterations of this job in the SCoD environment is shown in Fig. 3. Though the four tasks are sequential in appearance, since the job may be performed iteratively before the final FEA results are satisfying, in reality, the tasks in the job are overlapped with each other on the time horizon, therefore concurrent activities can be achieved in the macro level. To form a service federation for this job, a composite exertion (i.e. a service job) is created to include four elementary exertions (service tasks in this case). When the static analysis job is executed in the SCoD application layer, the four underlying service tasks are dynamically bound to the four corresponding service providers through Jobber's coordination, and the outputs of the previous tasks are mapped to the required inputs of the service context of the subsequent task. Thus all the related service providers can work collaboratively on the shared context. In this example, the final result will be produced by the human expert service, and if this result does not satisfy some predefined standards in the static stress application, the static stress analysis job can be iterated until these standards are met.

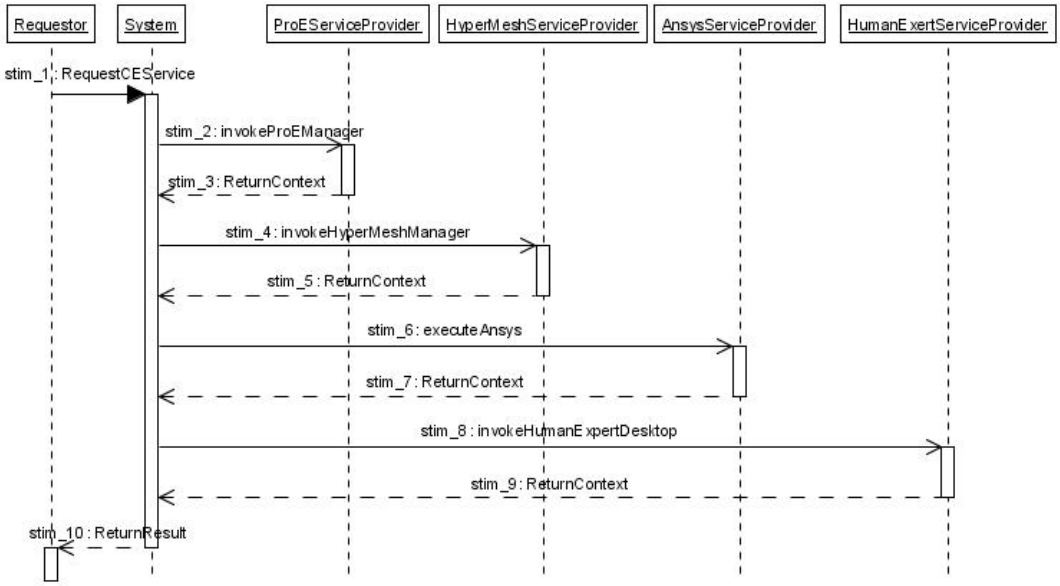

Fig. 3 Sequence of activities in a CE job

To further facilitate the utilization of the four service providers, a facade service provider can be further developed to provide integrated access to all needed service via a zero-install service UI [11]. That facade service used by service browsers [12] allows for accessing the service UI with needed integration of overlapping tasks during an engineering design process as depicted in Fig. 1. Obviously, the service UI role is to collect the user inputs and create the relevant job at runtime with all required tasks accordingly and submit it onto the network. This interactive process via the service UI can continue until the final result satisfies the design requirements.

Deployment of the SCoD Platform Prototype

Based on SORCER and the SCoD architecture presented above, a prototype for the SCoD platform

was developed. Four service providers have been deployed for the four tasks depicted in Fig. 1. Each service in the SCoD platform can be discovered and invoked dynamically by a requestor, and the four services can form a federation at runtime to carry out the static stress analysis job. Alternatively, a service can be invoked from a service browser interactively using a zero-install user agent attached to the SCoD façade.

Conclusions

SORCER provides a dynamic service oriented architecture and exertion-oriented programming for developing distributed applications on the Internet. Based on SORCER, we have built a service-oriented collaborative design platform for CE—SCoD, which can integrate CAD and CAE software tools as services for CE product development teams. Services can be distributed among different hosts to allow reusability, scalability, reliability, and load balancing. Service providers can be replicated and dynamically provisioned for reliability to compensate for network failures. They can be discovered dynamically in runtime by the service types they implement. Based on exertion-oriented programming, a requestor can write exertions to invoke the CAD and CAE services in a dynamic federation with no need to know the exact location of a provider beforehand, so flexibility can be well achieved, and collaborative design in the CE systems can be implemented in the distributed dynamic environment.

Acknowledgement

This paper is supported by Beijing Jiaotong University Research Fund (Project No. 2007RC035, No.2006XZ011). The SCoD Project is a collaborative effort between the Intelligent Engineering Lab, Beijing Jiaotong University and the SORCER Lab, Texas Tech University. We would like to thank all the members of SORCER Lab and Computer Science Department at Texas Tech University for their tremendous help and support in this collaborative project.

References

[1] W.I. Winner, IDA Report R-338. AD-A203/615 (1988)

[2] M. Sobolewski, in: Proceedings of the 2007 IMCSIT Conference, PTI Press (2007).

[3] W.A. Ruh, T. Herron, P. Klinker: IIOP Complete: Understanding CORBA and Middleware Interoperability, Addison-Wesley (1999).

[4] Information on http://www.globus.org

[5] Information on http://www.w3.org/2002/ws/

[6] Information on http://www.jini.org

[7] Information on http://en.wikipedia.org/wiki/Fallacies_of_Distributed_Computing

[8] M. Sobolewski, R. Kolonay: International Journal of Concurrent Engineering: Research & Applications, Vol. 14 (2006), p.55.

[9] S. Goel, S. Talya, and M. Sobolewski, in: Next Generation Concurrent Engineering: Smart and Concurrent Integration of Product Data, Services, and Control Strategies, ISPE, Inc. (2005).

[10] M. Sobolewski, in: Advances in Concurrent Engineering, A.A. Balkema Publishers (2002).

[11] Information on http://www.artima.com/jini/serviceui/index.html.

[12] Information on http://www.incax.com