A distributed feature-based environment for collaborative design
Li, W.D.; Lu, Y-Q.; Zhou, Hong; Ong, S.K.; Nee, A.Y.C.; Fuh, Y.H.; Wong, Y.S.

Published in:
Journal of Systemics, Cybernetics and Informatics

Published: 01/01/2003

Document Version
Publisher's PDF, also known as Version of Record (includes final page, issue and volume numbers)

Please check the document version of this publication:
• A submitted manuscript is the author's version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher's website.
• The final author version and the galley proof are versions of the publication after peer review.
• The final published version features the final layout of the paper including the volume, issue and page numbers.

Link to publication

Citation for published version (APA):
A Distributed Feature-based Environment for Collaborative Design

Wei-Dong Li, Yi-Qiang Lu, Hong Zhou
Singapore Institute of Manufacturing Technology
71 Nanyang Drive, Singapore, 638075
†Email: wdli@gintic.gov.sg, Tel: +65-6793-8354, Fax: +65-6791-6377

Soh-Khim Ong, Andrew YC Nee, Jerry YH Fuh, Yoke-San Wong
Department of Mechanical Engineering, National University of Singapore
10 Kent Ridge Crescent, Singapore, 119260

ABSTRACT

This paper presents a client/server design environment based on 3D feature-based modelling and Java technologies to enable design information to be shared efficiently among members within a design team. In this environment, design tasks and clients are organised through working sessions generated and maintained by a collaborative server. The information from an individual design client during a design process is updated and broadcast to other clients in the same session through an event-driven and call-back mechanism. The downstream manufacturing analysis modules can be wrapped as agents and plugged into the open environment to support the design activities. At the server side, a feature-feature relationship is established and maintained to filter the varied information of a working part, so as to facilitate efficient information update during the design process.

Keywords: Distributed environment, collaborative design, feature-based modelling

1. INTRODUCTION

For a complex design task, a design team is usually engaged, and the communication and collaboration among members in the team are crucial to enable the design to be carried out effectively. With the rapid development of IT, it is possible and imperative to develop a distributed design environment, in which geographically-distributed design systems can be integrated and a design team can be set up within the Internet/Intranet. In this environment, design information among the team can be exchanged across physical and temporal boundaries efficiently. With the integration of manufacturing analysis modules in the environment, downstream manufacturing processes of the product life-cycle can be considered and evaluated in the initial design phase, which will lead to better engineered products with higher quality, less iterations and more cost competitiveness.

Recently, some research and developments have been carried out in this area. Different collaborative design mechanisms and system architectures have been designed. A summary is listed in Table 1.

In this paper, a distributed feature-based modelling environment to support collaborative design is described. A designed part is modelled at the server side, and the clients are responsible for design parameter input, visualization and selection of operations for the part. The information communicated between the clients and the server is wrapped as events, and an event-driven and call-back mechanism is developed to efficiently update information in the environment. The environment is open and scalable, and it can dynamically integrate downstream manufacturing analysis agents without re-initializing the whole system. At the server side, a feature-feature relationship is established to differentiate the varied information of a part during its creation or edition process, so as to efficiently update information in the network.

2. SYSTEM FRAMEWORK

The system framework is illustrated in Figure 1. The three parts in the system are as follows:

(1) Design clients;
(2) A collaborative server; and
(3) Intelligent downstream manufacturing analysis and optimization agents.

The detailed functions of each part in the environment are described in Table 2. The organization for designing a part collaboratively in the environment is depicted in Figure 2.

3. DISTRIBUTED MECHANISM

3.1 Remote Method Invocation (RMI) protocol

Java RMI is a simple and yet powerful protocol for distributed object design. Java RMI-based objects can be quickly deployed and managed across networks, and the RMI mechanism is a much easier and lighter weight approach to distributed object design. In this research, the establishment of the distributed design environment is based on the Java RMI mechanism.

According to the Java RMI mechanism, through defining remote interfaces, methods and objects can be used for remote calling and transmission. The remote interfaces defined in the environment are given in Figure 3.

3.2 Event-driven and call-back mechanism

In order to enable clients to update design information only when the server has a new event to communicate, instead of routinely pinging the server for information and creating a network backlog, a call-back mechanism is employed to develop a high-performance and robust server. The working process for the call-back mechanism is described as follows:

(1) In a working session, a list is created to store the references of clients that have joined the session.
(2) With an input of parameters for a feature, a parameter event is generated in a client. Through invoking one of the server methods - push_Event(), such an event is received and handled by the server. After an object event is created and
ready for broadcasting from the server, each client recorded in the reference list is activated to receive the event by invoking one of the clients’ methods - receive_Event(). A similar process can be carried out between the server and the required CAPP agent.

A common event interface, which extends Serializable
class for communication in the network, is defined. Based on this interface, four types of events have been implemented as follows:

1. Parameter event for a design feature. This event, which is generated in a client, is used to wrap the parameters for a feature or a set of selected entities for local operations on an existing feature. This event is dispatched to the server for creating a feature represented as a B-Rep object.

2. Object event for design features. This event wraps the features in the server to be sent back to the clients for visualization and manipulation.

3. Object event for a design part. The feature objects of a design part are wrapped in this event, which is dispatched from a client to the CAPP service provider for analysis.

4. Process plan event for a design part. This event is generated by the CAPP service provider to bind the generated process plans for a requested client.

The four types of events are defined in Table 3.

4. INTELLIGENT ANALYSIS AGENTS

The downstream manufacturing analysis modules can be wrapped as intelligent agents to be plugged in the system to support concurrent engineering design. A three-layer architecture, including remote interfaces, abstract classes and class implementation, is designed for this purpose. With the definitions of the abstract classes for agents, some agents can be integrated later without re-initializing the whole system.

Currently, a hybrid genetic algorithm and simulated annealing CAPP optimization agent has been designed and integrated [Li et al., 2002(a)]. With such an agent, the activities of selecting machining resources, determining setup plans, and sequencing machining operations can be considered simultaneously so as to achieve the global lowest machining cost according to a combined evaluation criterion of machine costs, cutting tool costs, machine changes, tool and setup changes.

The other two agents, namely machining feature recognition and manufacturability analysis [Li, 2001], will be integrated later.

5. FEATURE RELATIONSHIP AND MANIPULATION

Table 2. The functions in each part of the distributed environment

<table>
<thead>
<tr>
<th>Clients</th>
<th>Collaborative Server</th>
<th>Analysis Agents</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Parameter Input/Edition Interface</strong></td>
<td><strong>Session Manager</strong></td>
<td>Intelligently analysis and optimization modules</td>
</tr>
<tr>
<td>• Input/edition of parameters for features</td>
<td>• Dynamical generation of working sessions;</td>
<td>• Machining feature recognition</td>
</tr>
<tr>
<td>• Selection of entities in features</td>
<td>• Sharing of parts for clients within a working session</td>
<td>• Manufacturability analysis.</td>
</tr>
<tr>
<td>• Information queries for features and parts</td>
<td></td>
<td>• CAPP optimization</td>
</tr>
<tr>
<td><strong>Visualization Environment</strong></td>
<td><strong>Modelling workspace</strong></td>
<td></td>
</tr>
<tr>
<td>• JFC-swing components to build the environment</td>
<td>• Acceptance of parameters or entities from the clients</td>
<td>• Providing abstract classes of machining feature recognition</td>
</tr>
<tr>
<td>• Visualization and manipulation of parts</td>
<td>• Modelling parts based on OpenCasCadeTM solid kernel</td>
<td>and manufacturability analysis for future integration</td>
</tr>
<tr>
<td></td>
<td>• Through Java Native Interface (JNI), the native API functions of the OpenCasCadeTM can be invoked and manipulated by Java application</td>
<td></td>
</tr>
<tr>
<td><strong>Discussion Community</strong></td>
<td><strong>Table 2. The functions in each part of the distributed environment</strong></td>
<td></td>
</tr>
<tr>
<td>• Provision of e-room for discussion and information (text and multimedia) exchange among clients</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Communication
The communication between each part is through an event-driven and call-back mechanism based on the Java RMI protocol.

Figure 2. Organization of a design task in the environment

5.1 Feature-to-feature relationships

In the environment, a client dynamically and interactively edits features in a part. Since a feature is usually associated with other features in the part, the edition operation on a feature might cause variations in other features in the part. In order to speed up the transmission of the designed model efficiently via the network with limited bandwidth, the varied information during the edition process should be differentiated in the server side and synchronized with the unchanged information in the client side. For this purpose, a feature-feature relationship should be studied.

The relationships between features can be categorized as interacting and non-interacting relationships. The interacting relationships between features include adjacency, overlapping, nesting and constraining [Li et al., 2002(b)]. The relationships are defined in Table 4, in which the symbols are as follows. Several cases for these definitions are illustrated in Figure 4.

FE1, FE2 - Two volumetric features in a part P
I ⊆ C - Regularized Boolean interaction, a proper subset, not a proper subset
5.2 Feature manipulation operations

For the different edition operations on a feature (adding features, deleting features, or modifying parameters of features), the feature-feature relationships associated with this feature are set up and maintained. The processes are described as follows:

(1) When a feature is New Added, its interacting and non-interacting sets are set up according to the definitions. The features in the interacting set are Updated.

(2) When a feature is Deleted, its nesting features are Deleted. Other features in its interacting set will be differentiated as two types: features with changes in volume or boundary are Updated, or features without changes in volume and boundary are Unchanged. The features in the non-interacting set are Unchanged.

(3) When the parameters of a feature are modified, the process consists of three steps:
   (a) The features in its non-interacting set that have adjacency, overlapping or nesting relationship with this feature due to the modification are Updated;
   (b) The features in its interacting set will be differentiated as two types: features with changes in volume or boundary are Updated, and features without changes in volume and boundary are Unchanged;
   (c) If this feature has a constraining feature, the interacting and non-interacting features of the constraining feature will be adjusted according to the above steps.

For features that are New Added and Updated differentiated from above process, their unique IDs, feature objects and other properties are wrapped into the object event for design features. For features that are Deleted, only their IDs are recorded in the event for clients to erase the relevant information. The Unchanged features will be kept the same in the clients. The scenario is shown in Figure 5.
### Table 4. The relationships between two interacting features - \( FE_1 \) and \( FE_2 \)

<table>
<thead>
<tr>
<th>Relationships</th>
<th>Definitions</th>
</tr>
</thead>
<tbody>
<tr>
<td>( FE_1 ) and ( FE_2 ) are adjacent</td>
<td>( FE_1 \cap FE_2 = \phi, \partial(FE_1) \cap \partial(FE_2) \neq \phi, \partial(FE_1) \cap \partial(F') = \phi, \partial(FE_2) \cap \partial(F') \neq \phi. )</td>
</tr>
<tr>
<td>( FE_1 ) and ( FE_2 ) are overlapped</td>
<td>( FE_1 \cap FE_2 \neq \phi, FE_1 \subsetneq FE_2 ) and ( FE_2 \subsetneq FE_1. )</td>
</tr>
<tr>
<td>( FE_1 ) nests ( FE_2 )</td>
<td>If both of ( FE_1 ) and ( FE_2 ) are either negative or positive: ( FE_1 \cap FE_2 = \phi, \partial(FE_1) \cap \partial(FE_2) \neq \phi, \partial(FE_1) \cap \partial(F') = \phi. ) If one of ( FE_1 ) and ( FE_2 ) is negative, and another one is positive: ( FE_1 \subsetneq FE_2. )</td>
</tr>
<tr>
<td>( FE_2 ) nests ( FE_1 )</td>
<td>If both of ( FE_1 ) and ( FE_2 ) are either negative or positive: ( FE_1 \cap FE_2 = \phi, \partial(FE_1) \cap \partial(FE_2) \neq \phi, \partial(FE_2) \cap \partial(F') = \phi. ) If one of ( FE_1 ) and ( FE_2 ) is negative, and another one is positive: ( FE_2 \subsetneq FE_3. )</td>
</tr>
<tr>
<td>( FE_1 ) and ( FE_2 ) are constrained</td>
<td>There are some constraints between the entities in ( FE_1 ) and ( FE_2. )</td>
</tr>
</tbody>
</table>

![Diagram](image)

**Figure 4.** Examples of relationships between interacting features - \( FE_1 \) and \( FE_2 \)

An example case is illustrated in Figure 6 (a). The feature-to-feature relationships for the part are shown in Figure 6(b). There is a constraint between the widths of slots 1 and 2. With the shrinking operation of slot 2 in Figure 6(c), the relationship between the feature and the features in its non-interacting set remain the same and are labeled as *Unchanged*. The features in its non-interacting set are *Updated*. Since slots 1 and 2 have the same interacting set, with the variation of slot 1, the features in the interacting set of slot 1 will not be handled further. The final result is shown in Figure 6(d).

### 6. CONCLUSION

A distributed design environment based on 3D feature-based modelling and Java technologies has been proposed and developed. The advantages of the approach include:

1. The environment can simulate a practical teamwork situation through creating and managing dynamic sessions, in which clients can play different roles in the design task.
A high-performance and robust server is established through an event-driven and call-back mechanism; (2) The design information based on 3D feature-based modelling technology can be exchanged and updated in the environment efficiently through a feature-to-feature relationship and manipulation method. The system is open and scalable, and additional analysis modules can be incorporated by implementing some pre-defined abstract classes.

REFERENCES

Inventor™ collaboration tools, 2001, Autodesk Inc. www.autodesk.com

Figure 5. Differentiation of features during a part edition process

A Client

Parameter Event

Feature-feature Relationship

A part

Object Event

Unchanged Features

A Client

A Client

A Client

New_Added Features

Updated Features

Deleted Features

Figure 6. A case part containing the differentiated features