Agent-Based Distributed Design System Architecture for Basic Ship Design

Kikuo Fujita
Department of Computer-Controlled Mechanical Systems
Osaka University
Suita, Osaka 565-0871, Japan

Shinsuke Akagi
Department of Computer-Controlled Mechanical Systems
Osaka University
Suita, Osaka 565-0871, Japan

Abstract
This paper proposes an agent-based distributed design system architecture for complicated, large engineering systems and develops its experimental implementation for basic ship design. First, dependency and concurrency in design activities of engineering systems are discussed from the viewpoints of hierarchical system structure and entity-and-attribute relationship, and it reveals the possibilities of parallel execution of design operations under systematic viewpoint. Second, the design operations are classified into generating system structures, configuring spatial arrangement, assigning system attributes and selecting system components. In the architecture, design operations for attributes that are major parts are modeled with object orientation, and they are distributed to agents by extending an object oriented programming technique to an agent approach. The others are encapsulated as independent task agents. After describing key concepts for its computer implementation, this paper shows an application to basic ship design.

1 Introduction
Complicated, large engineering systems concern with many items and multiple disciplines, and their design activities have been and should be natively distributed within design teams or else. Requirements for enhancement and speed-up of such activities lead the challenges in concurrent engineering, for instance toward integration between marketing, design and manufacturing. The research field of computer supports for concurrent engineering has been an active area for a decade. Since total management of design data, that are interpreted into manufacturing and other activities, is effective for seamless integration of those activities, database technologies have been central in the direction. However, design activities include both what are design inputs and outputs and how those design data are emerged. This means that synthetic design activities should be incorporated with design data management. Further, while central databases facilitate the integration, such technologies are naive against huge integration, design activity distribution, engineering technology updates and so forth.

This paper proposes an agent-based distributed design system architecture for some engineering systems design such as for ships, aircraft, etc., under the above viewpoints. The distribution and concurrency of design activities must be followed with systematic structure of systems in both representation and synthesis. This implies that the computer implementation of those distribution and concurrency should be also matched with the underlying structure of engineering systems. The architecture proposed in this paper introduces an object orientation as a fundamental representation scheme, and combines it with an agent based distribution of computational components. The entity and attribute information are represented as objects and manipulated through message passing mechanism between objects within/across agents. The generation activities of system structure and arrangement are also encapsulated as agents. This paper shows an implementation of the architecture as an experimental computer system with a workstation cluster connected through Ethernet LAN, where the basic design of ships are used as an application field.

2 Design Problem of Engineering Systems

2.1 Systematic structure of engineering systems
Since practical engineering systems concern with many items and multiple disciplines, their design activities have been and should be performed in a way of concurrent engineering. As a nature of large systems, the whole
design representation of a design object can be recursively divided into a set of modules in various granularity levels. That is, the whole system can be decomposed into several subsystems, and such subsystems can be further decomposed into micro subsystems. Each system (subsystem) can be represented by a set of attributes, that are typically geometric dimensions. For instance, a ship is preliminarily represented with principal particulars such as length, depth, width, draft; it has several representations for hull form, hold arrangement, configuration of a power plant, each hold has beam structure, components of a plant should be arranged in a machinery hold, and so forth.

Under such a system structure, since attributes correspond to respective levels of system representation, ones for a supersystem constrain others for their subsystems. The design problems of established engineering systems can be viewed that such attributes should be determined so as to totally satisfy and optimize design requirements. For instance, with ship design, the top-level system can be represented with principal particulars. At least, their values must be ascertained by preliminary estimation of several functional requirements and design criterion. On the other hand, they assign a required propulsion power, and it enables the selection of a main engine. Simultaneously and separately, hull form can be arranged within these principal particulars. The selected engine must be ascertained against the propulsion resistance that can be calculated from the hull form. Further, the arrangements of bulk heads, machine rooms, etc. can be continued after these items respectively and concurrently. Among these design activities, standard system structures and spatial arrangement have been well established for a class of engineering systems, but some adjustments are necessary for individual design situations. Thus, the design process must include the tasks for configuring a set of system components, for slightly arranging system structures within prescribed templates, and so forth.

2.2 Hierarchical decomposition of systems and consequent concurrency of design activities

As shown by the ship design example, design operations on respective modules/subsystems are hierarchically dependent each other, but they can be performed quasi-independently. This circumstance leads concurrency of design activities, to which computer-based concurrent engineering systems should follow.

Figure 1 illustrates the above circumstance of engineering system design. That is, hierarchy of a system structure corresponds to dependency among activities on respective subsystems, and breadth of subsystems in a certain level means concurrency among design activities. Further, a pair of subsystems whose supersystems are different from each other can be concurrently proceeded. In the design process of well-established engineering systems, the determination of system attributes is major parts rather than configuring system structures and arranging spatial relationships among components. This enables that most design activities can be concurrently executed, except that a set of attributes that relates to a specific subsystem cannot have such a concurrency.

This paper proposes an architecture for parallel execution of the above types of concurrent design activities for complicated, large engineering systems. Besides, it seems that the ordinary understanding of concurrent engineering necessity relates to the marketing, design and manufacturing integration or to design process integration throughout conceptual, preliminary, embodiment and detail stages.

2.3 Types of design operations

Before discussing the system architecture for computer-supported concurrent design of engineering systems, we categorize design operations for well-established engineering systems based on the understanding of Fig. 1. The following are with the case for a ship design problem:

- Configuring system structures:
  - Template based generation of system structures ⋯ In the case of basic ship design, most parts of system structures are fixed beforehand, but, for instance, the number of holds must be determined based on ship size, i.e., required deadweight.
  - Spatial arrangement of components ⋯ In the case of basic ship design, for instance,
plant machinery such as a main propulsion engine, diesel engines and shaft generators for electricity, auxiliary boilers for heat supply must be arranged within a machinery hold, and so forth.

- Assigning attributes within a structure:
  
  - Parametric design ··· In the case of basic ship design, the principal particulars and the other attribute values of various subsystems must be determined. They are design variables of various systems, their consequent performance measures, etc.
  
  - Catalog selection ··· In the case of basic ship design, the above machinery of a plant must be selected from catalog data based on the required propulsion power, electricity and heat demands that are determined by ship size, etc.

Among these, parametric design parts are primal for preliminary design of engineering systems such as basic ship design.

As for the computational support or automation for respective tasks mentioned above, knowledge base techniques and search based artificial intelligence techniques can be applicable for individual ones [1, 2, 3, 4]. Especially, object orientation is useful for representation scheme for parametric design part [1].

3 From Object Orientation to Distributed Agents

3.1 Object orientation for design representation and parametric design

A system can be represented with entity-attribute relationships as shown in Fig. 1. A representation scheme for them is a fundamental issue for computerization. For the integration of operation with representation, the concept of object-oriented programming techniques has been appropriate, while it is a descendant of actor theory, frame representation, etc. [5]. As a usage of object orientation in design systems, Akagi and Fujita developed an expert system for parametric design [1], its integration for geometry design [3] and catalog selection of plant machinery [2] for basic ship design. One of the purposes of this paper is to extend these techniques toward computer-supported concurrent engineering.

Figure 2 shows the representation of entity-attribute relationships¹. Design operation requires design procedures for respective items as well as representation slots. In the figure, both entities and attributes are represented as instance objects, and the latter are linked with the corresponding former. Each attribute instance also has slots for representing its design procedure and value. Further, meta entity instances are introduced for representing how to generate entity instances based on design requirements. Such an instance object has slots for the entity class name and an entity generation procedure.

3.2 Dependency and concurrency in design computation

Under this representation scheme, most design activities are executed with attribute objects in the cases of engineering systems such as ships. The values of respective attributes depend on the ones of other attributes through the contents of ‘procedure’ slots, and the dependency and concurrency discussed in the previous section can correspond to this type of recursive relationships. For explanation simplicity, assume that there are three attribute variables A, B and C that belong to different entities and that the value of A can be determined with an equation ‘B + 2.0C’. In this case, while the operation for determining the value of A depends on the results of both B and C, the operations for B and C can be executed concurrently. Since well-established engineering systems include a massive number of attributes, this kind of concurrency dominates the dependencies among them. This is expected to be a promising point for design speedup through computer-supported concurrent engineering.

Object-oriented programming has concurrency in computation as its nature, but ordinary implementations do not have parallel execution of such concurrency. In order to establish a computational parallel implementation

---

¹The contents of object orientation are arranged from ones described in the original papers [1, 2, 3] for some extensions and CLO5 use.
of the above type of concurrency, we need distribute them to appropriate chunks of operations, and such distribution must be matched with underlying engineering disciplines and system structure. As well, the distribution of parametric design must be able to cooperate with other types of design activities such as catalog selection, spatial arrangement and so forth.

### 3.3 Agent based distribution of design computations

This paper introduces an agent technique for the distribution purpose of the above object orientation. While agent techniques have been introduced into computer-supported concurrent engineering, the meanings of agents are widely spread from a large number of simple single-function agents to a small number of task based agents. For instance, Brown discussed agents as primitive media for design knowledge and reasoning [6], and Park et al. developed a computer system for cable harness design by using agent technology [7]. While a solution expected through agent technologies is to facilitate collaborative design activities through synchronization of different design tasks with communication and coordination, etc., fundamentally it can be understood as a mean for distributing representation and computation.

The agent concept that is used in this paper belongs to the small number of task based agents, and its purpose is to distribute computational units to different computation resources. On the basis of design operation type categorization, the parametric design sources that can be represented with object orientation are encapsulated as instance objects in fine grain, and their chunks that correspond to system-level decomposition of a design object respectively are distributed to different agents in coarse grain. The other types of design operations are encapsulated as agents, and they are distributed to different computational resources respectively as well.

### 4 Architecture for Agent-Based Distributed Design System

An agent technique consists of several layers from an inter-agent communication mechanism to a task based coordination mechanism and so forth. This section describes several concepts used in our agent-based distributed design system.

#### 4.1 Agent communication

As aforementioned, the parametric design tasks can be effectively implemented with an object-oriented programming technique [1]. Since it is a powerful programming paradigm and it is suitable for agent programming in the aspects of encapsulation, message passing mechanism and so forth, we use object orientation as a foundation. In order to expand an object-oriented programming beyond a barrier among individual computational resources, we need to make message-passing go over the barrier. This requires the following two kinds of messages over agents as an agent communication mechanism under the object orientation, while low-level communication channels depend on computer hardwares and primitive softwares that are used:

- **Send** type message ... This message (agent message) is used to transmit a message (object-oriented message) from an object to another object that exists in another agent. This can be implemented with few mechanisms that are to put a message with a tag and a body to a communication channel, to wait for a reply message against it, and to translate messages between agent messages and object-oriented messages at both sides of the channel.
- **Broadcast** type message ... This message is used, for instance, to find an object that has a certain name in another agent. For this purpose, we provide a mechanism to send a message to all agents. On the other hand of this type of broadcast messages, every agent must have an ability to immediately respond against them.

Besides of the above, the practical issues on the agent-based computation require the following two types of messages additionally:

- **Agent setup** type message ... This is to create a new agent within a computation resource cluster on demand.
- **Remote procedure call** type message ... This is to execute a specific procedure described in a practical computer language at another agent, for instance for initializing activities on it.

### 4.2 Objects over agents for parametric design

Under the above agent communication architecture, the objects that represent entities and attributes (Fig. 2) can be encapsulated and distributed through agents as shown in Fig. 3. The design operations of objects are executed based on the procedure described in their slots. The following methods perform them through message passing:

- **Decide** ... to request the values of argument variables (attributes) for the sender’s value determination. This
message is recursively sent to related variables, if they do not have values.

- **Cancel** ... to cancel the receiver’s value and to recursively cancel the values of dependent variables (attributes) before cancellation of the sender’s value.

- **Query** ... to query the receiver’s value if it has a value. Otherwise, this operation returns a flag to the sender.

Iterative and recursive executions of *decide* and *cancel* methods principally form a design process [1]².

In order to parallelize the concurrency of design operations, first, the requests of *decide* and *cancel* methods should be simultaneously sent to corresponding objects, while this mechanism is implemented by the multitasking mechanism of a computer language. Further, the agent based distribution of objects can parallelize such executions with different computational resources for respective agents.

### 4.3 Method synchronization and redesign mechanism

Under the parallel execution of design operations through *decide* and *cancel* messages, duplicated or conflicted messages may be sent to an identical object. For instance, the object that is going to determine its value may receive an extra *decide* message, such an object may receive a *cancel* message that should result in termination of the *decide* operation, and so forth. For managing these kinds of situations, we define four states indexes in every object:

- **Unbound** state ... This means that it does not have a value and no operation is performing.

These states shift one to another as shown in Fig. 4 with triggers of receiving messages. This transition includes several waits for synchronization of method operations as shown in the figure. Besides, sequential execution of design operations does not require the specific states of determining and canceling, since an operation in progress itself means such a state and any other operation never happens during it.

Redesign is an important phase of parametric design. Fundamentally, it is performed with precedent *cancel* operations and succeeding *decide* operations. Under the state transition, once a designer decides to modify an attribute, first a cancel message is sent to the corresponding instance object and then the dependent attributes are automatically canceled based on the relationships in their procedures, and second a designer can assign a new value to the attribute. After these, when the instance object receives a *decide* message, its related attributes are automatically determined again by recursive sending of *decide* messages.

### 4.4 Independent tasks for configuring system structures and arrangement

The tasks for configuring system structures and spatial arrangement are other parts of engineering systems design, and the distributed design system architecture must link such tasks with the above parametric design framework.
For this purpose, we encapsulate each configuration task as an agent and provide a mechanism to feed parametric design result to encapsulated task agents. This facilitates to build specific tasks through independence from other parts.

Figure 5 shows the mechanism for encapsulation of independent tasks for configuring systems and arrangement. In the figure, after an independent task agent is initiated, it creates receptor objects for booking design information on specified entities or attributes. First of all, a receptor sends an attention message to a corresponding entity or attribute object, and the corresponding object stores the receptor’s name in its attention list. After this, every time the design information stored in the corresponding object is changed, the receptor receives such update information. Based on this mechanism, the encapsulated agent can perform its task at appropriate timing. While the contents of independent tasks depend on a specific design subject, a basic ship design application requires encapsulated agents for energy plant configuration, machinery room arrangement, etc.

4.5 Design process management

Since all operations described above are executed on demand as a nature of object orientation, some triggers for starting design operations must be expressly provided. As aforementioned, an engineering system has hierarchical structure, and the design process has corresponding dependency and concurrency. This implies a categorization of design items into several chunks that can be called as design steps respectively, and each of them includes few representative items. Thus, the overall design process can be organized by declaring such items as ones that should receive a decide message at the beginning of respective steps of the design process. Design process management through the above mechanism can further parallelize design operations by simultaneously activating multiple design steps, while decide messages parallelize design operations essentially in fine granularity.

4.6 Design diagnosis

Since design process consists of design, verify, critique and modify phases [8], checklists for design diagnosis are useful for design support. To provide this mechanism, we use the receptor mechanism as well as encapsulated independent tasks (Fig. 5). When checklists are itemized as rules, each rule can be described as an inequality or equality among a set of attributes, and it can be evaluated when all of related items have values. By using receptors for design items of a rule respectively, design information of all items is automatically fed to the rule, and each rule becomes to be evaluatable on time.

In the coordination with design process management, a set of rules that is related to a specific discipline are evaluated at the ends of respective design steps, not at every time when each rule becomes evaluatable. This is better for interaction with a designer who is responsible for the discipline.

5 Computer Implementation of Distributed Design System

5.1 Computer Environments

We use a cluster of UNIX workstations that are connected with Ethernet LAN for an experimental implementation of the agent-based distributed design system, and use Allegro Common Lisp [9] as a base programming language that is in conformity to ANSI Common Lisp and Common Lisp Object System (CLOS) [10] and that has a multi-tasking functionality and a foreign-function interface. While object orientation is a fundamental concept in the architecture, symbolic manipulation functionality of Lisp and generic function for method binding mechanism are effective for the implementation. The multi-tasking functionality, that can fork subprocesses, wait for a termination of subprocesses and lock the other process execution, is also necessary for parallelizing computations within an agent, and it enables to receive and to handle agent messages coming from other agents anytime. As for the network programming, we use the ARPA Internet domain socket communication protocol with the C language from Common Lisp through the foreign-function interface.

5.2 Agent communication

This implementation introduces a simple communication server method for agent communication due to its sim-
plicity, while there would be other sophisticated methods. Figure 6 conceptually illustrates the communication server method. In this method, the communication server is a relay station for all agent messages, that simply stores and forwards every coming message to the agent(s) indicated with a tag. It is implemented in the C language with socket communication library. Each agent that is a client in the communication mechanism is fundamentally an Lisp process that is extended with the socket based communication function. At the client side, a subprocess running in Lisp is watching the low-level message buffer and managing the context-level message buffer from the low-level, and a subroutine implemented in the C language manages sends out and receives the messages from/to the communication server. The multi-tasking functionality of Lisp enables simultaneous execution of design operations and agent message handling in addition to parallel execution of methods.

5.3 Class and objects

Figure 7 shows the class hierarchy of object oriented programming used in the distributed design system. For the distribution of objects to different agents, the following classes are introduced:

- **Local object** … The object that originally exists in the agent.
  - **External local object** … The local object that can be exported to other agents.
  - **Internal local object** … The local object that cannot be exported to other agents.

- **Imported object** … The object that is imported from another agent and that has only its name and the identifier of the agent where the original object exists. Import of an object is performed after sending a broadcast message to find the location of the corresponding external local object.

Under these classes, an imported object class is necessary for each external local object class, respectively.

Under the architecture discussed in the previous section, meta-entity, entity and attribute classes are defined for parametric design, receptor and transmitted object classes are defined for interface with task agents, a design step class is defined for design process management, and a check item class is defined for design diagnosis. Among these, since each of meta-entity, attribute and design step instances must handle a procedure for determining its content, methods for its operations are commonly defined with a procedural object class.

Besides of these, other types of classes are necessary for individual encapsulated task agents, and they depend on their contents. By means of receptor and transmitted objects, such classes can be independently introduced apart from the classes shown in Fig. 7.
6 Experimental System for Basic Ship Design

6.1 Basic design problem of ships

In the basic design of ships, it is required to determine the size of ship and related principal particulars, to arrange hull geometry and cargo holds, to configure propulsion power plant and so forth, so as to optimize economy and to satisfy safety conditions, regulations, etc. [11]. The design procedures and knowledge have been well established and organized through past engineering efforts. Since a merchant ship is a large, complicated system, many design issues are involved for the design process, and concurrent engineering concept is essential for their integration and rapid engineering process.

In comparison with Fig. 1, a ship is primarily represented with a set of principal particulars, and under them it has several levels of subsystems such as hull geometry, cargo hold arrangement, a propulsion plant, etc. Figure 8 roughly shows chunks of design items and their dependency. Based on such relationships, we apply the agent-based distributed design system architecture and the experimental design system to basic ship design. This experimental implementation is done by translating the past development of several design systems for ships, etc. [1, 2, 3, 4] that did not have abilities for concurrent engineering.

6.2 Design knowledge for parametric design

Parametric design for particulars and subsystems’ attributes including parametric geometry are major parts of basic ship design. Design knowledge for these parts is transferred from the object-oriented design system [1, 3]. The number of instance objects is about 500 in total. The number of related check items is about 50 in total.

These instances on design knowledge are categorized as shown in Fig. 8 and distributed to the following agents:

- **Determination of principal particulars and primal performance confirmation** … This category must be prior to the other categories. This priority is managed by process management function of the system.

- **Main engine selection and propulsion resistance** … A main engine is selected from catalog data based on required propulsion power and shift revolution number that are estimated from the principal particulars. The selected engine is checked against the propulsion resistance that can be calculated from hull geometry parameters. This requires hull geometry.

- **Hull geometry, cargo hold arrangement and loading tables** … Hull geometry and cargo hold arrangement are determined under the confirmed principal particulars. Loading tables for several conditions such as full load, normal ballast, etc. are generated. This requires mass properties that can be calculated with the hull geometry.

- **Stability analysis** … Since a ship floats on water plane, hydrostatic stability, i.e., trim and metacentric height must be ascertained for several loading conditions. This requires the loading conditions and hydrostatic data that can be calculated from hull geometry.

Among these, some items are related to geometries of hull and cargo holds. Such geometries and their mass properties can be assigned to attribute variables of entities under the entity and attribute representation scheme shown in Fig. 2 [3].

Since all instances for parametric design are individually encapsulated under object orientation, any distributions are possible in theory. However, it is important to arrange them into closely related groups for efficient design (Improper distribution increases the number of agent messages rather than the number of object-oriented messages within each agent).

6.3 Independent task agents

In addition to parametric design knowledge, the following tasks are presently implemented for basic ship design (The tasks implemented here do not cover all design activities required for ship design):
(1) Catalog selection

As mentioned above, a main engine is selected from catalog data. While design conditions for engine selection are represented under the parametric design framework, the catalog data for engine candidates has another data structure. Thus, catalog data represented with frame-type knowledge representation method [2] is locally integrated at the agent for engine selection.

(2) Plant planning

A marine power plant consists of diesel engines and shaft generators for electricity, auxiliary boilers for heat supply. The design problem of this subsystem is a system configuration problem under several template structures, that is linked with catalog selection of respective equipment. Further, since the optimality of a plant depends on machinery operation against energy demand patterns, the optimal plant must be selected possible candidates based on the optimal initial and operation cost, that can be calculated with solving a mixed integer programming problem on plant operation [12].

This plant planning is also integrated as an agent with the distributed design system, as similar to the above main engine selection. The detail design procedure is same to an expert system for marine power plants [12].

(3) Machinery room arrangement

The components of a power plant that are configured by the above task agent must be arranged in a machinery room that is determined by parametric design agents. This type of spatial arrangement problems can be automated by a constraint-directed search procedure [4]. However, spatial arrangement of system components is an absolutely different design task from parametric design, while it is indispensable in systems design problems. Thus, we link the spatial arrangement task and the others by means of receptors and transmitted objects that were explained in Fig. 5. That is, after the task is initiated, it immediately sends attention messages to the plant configuration object and the machinery room object for reserving transmission of design information. By this mean, once the necessary conditions for spatial arrangement become ready, the encapsulated task can start its own design operation.

6.4 Example of system execution

Figure 9 shows a system execution image of the distributed basic ship design system. The figure includes some snapshots of computer screens, that are under the X Window System.

In the figure, a bulk carrier is designed for 38,000 ton deadweight and 14.9 knot service speed. Part ③ shows several design and redesign iterations for determining
principal particulars. Part ② shows the design results of main engine selection. Part ③ shows hull form design with sectional lines and its perspective view. Part ④ is marine plant planning, where several plant candidates are listed in the order of total cost and the configuration of the best one is also shown. Part ⑤ is stability analysis, where hydrostatic curves and a GZ curve are shown. Part ⑥ is machinery room arrangement, where several plant components are arranged around a main engine in the four floors.

7 Concluding Remarks

This paper proposed an agent-based distributed design system architecture for engineering systems, and showed its experimental computer implementation and application to basic ship design toward computer-supported concurrent engineering. The underlying concept of this architecture is that dependency and concurrency of design operations should correspond to hierarchical system structure and entity-and-attribute relationship. Based on this, tasks for configuring system structures and spatial arrangement are encapsulated as agents, since each of them is an isolated operation, and parametric design operations that are automated by an object oriented programming technique are also agentized in proper units.

While the application of proposed architecture to some simple parts of basic ship design shows its fundamental validity, some extensions must be necessary to expand the architecture to more practical use and wide range of ship design, for instance in the aspects of dynamic allocation of design activities for computational resources, collaborative negotiation for conflict resolution and so forth.

Acknowledgments

This research is partially supported by “Research for the Future” Program of the Japan Society for the Promotion of Science (JSPS) under “Methodology for Emergent Synthesis” Project (project number 96P00702) and The Ministry of Education, Science and Culture of Japan through Grant-in-Aid for Scientific Research 08405016.

References