DESIGN OBJECT DECOMPOSITION IN A PRODUCT DEVELOPMENT CHAIN

Internet Lab Technical Report TR 2001-12

Wen-Chieh Chuang Peter O'Grady

Department of Industrial Engineering
Seamans Center
University of Iowa
Iowa City, Iowa 52242, USA

Tel: +1 319 335 5939 Fax: +1 319 335 5424 E-mail: peter-ogrady@uiowa.edu
http://www.iil.ecn.uiowa.edu/internetlab

Draft: July 12, 2001

Design Object Decomposition in a Product Development Chain

ABSTRACT

While supplier involvement in the design of a product (in what can be called a product development chain) is generally regarded as being of substantial benefit, there has been little work done on developing methodologies that can be used to support such supplier involvement. The need for work in this area of cooperative product development is given urgency by three developments: the continual business pressures of reducing costs and lead times, the tendency for assemblers to outsource an increasing proportion of their manufacturing and the development of communications technologies (particularly Internet technologies) that allow for a much greater degree of cooperation than has been possible hitherto. This paper aims to address the area of product development chains by developing methodologies to decompose overall requirements into design requirements for components or modules. These are the final design requirements that are ready to be sent to the part suppliers. Associated with these design requirements are sets of interface specifications that specify the interfaces between components or modules. First, a new algorithm is presented that can distribute design requirements into sub-entities. Second, another algorithm is described that generates the interface requirements between the different sub-entities. Third, the initial implementation of the algorithms in a system called DODA is described. This work lays the foundation for a research program in the important area of product development chains.

1. INTRODUCTION

Supplier involvement in the design of a product is generally regarded as being of substantial benefit. For example, Dowlatshahi (1998) suggests that early supplier involvement is a "means of integrating suppliers' capabilities in the buying firm's supply chain system and operations". Carlisle and Parker (1989) state that "the suppliers must be involved in the product design process in order to optimize their special skills and processes". Although more research is needed to quantify the benefits, it is generally agreed that early supplier involvement in the design provides an environment in which product components can be designed separately and simultaneous by different suppliers using the suppliers design expertise. As a result, the manufacturing cost and production time for individual parts can be reduced and the overall cost and time from product design to market can be lowered. In addition, later problems in the operation of the supply chain can be diminished.

The relationship between a product assembler and their part suppliers has been called a design chain (Clark and Starkey, 1988; Twigg, 1997) but, more properly, can be called a *product development chain*, since all the activities associated with product development are carried out in the chain. These activities include design, prototyping, and testing.

A suitable definition of a product development chain can be adapted from that given by Poirier and Reiter (1996) for a supply chain, as follows:

> *A product development chain is a system through which organizations develop products and services to meet customer requirements.*

The product development chain encompasses the suppliers and the customers, as shown in Figure 1, where customer requirements are passed to the suppliers and sub-suppliers, who then develop parts to be assembled and configured for the customer.

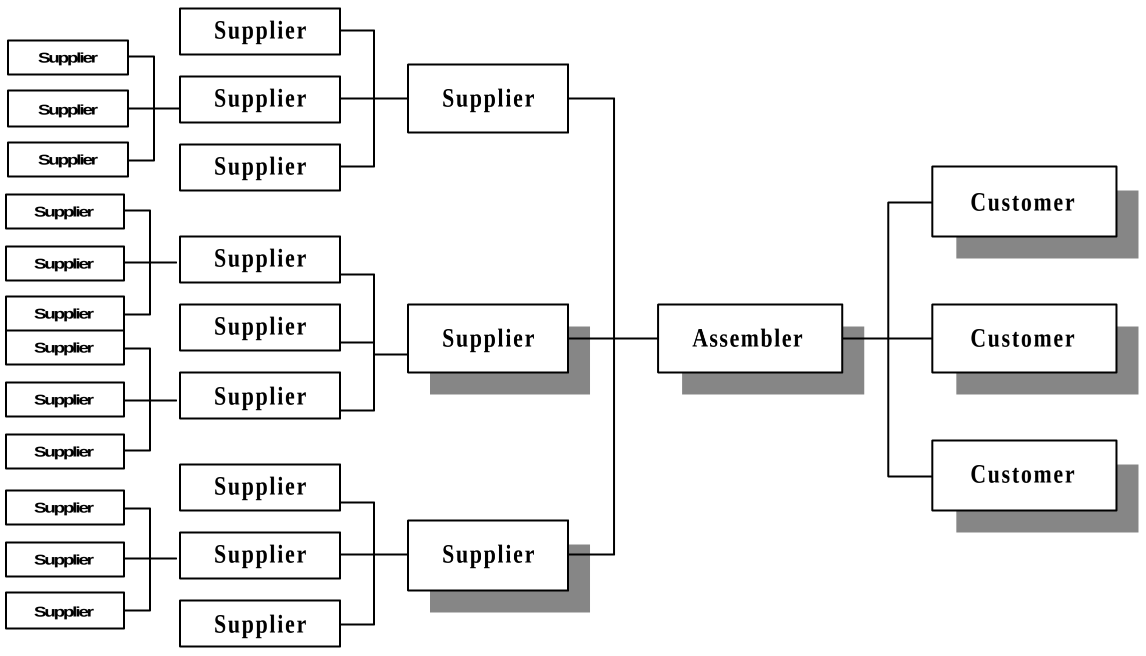


Figure 1: A Simple Product Development Chain

In practice, a product development chain is likely to be much more complex than that shown in Figure 1. There may be a considerable mesh of suppliers to the suppliers, sometimes called second tier suppliers, and so on to third, fourth and perhaps fifth tier suppliers.

To compound the complexity, each participant in the product development chain may be involved in a multitude of other product development chains, each vying for attention. Each manufacturer may, for example, be actively producing several hundred end-products, each requiring their own product development chain, probably with considerable overall between suppliers. Meeting customer requirements under these circumstances can be very difficult. Furthermore, each product development chain is dynamic in nature, with variations over time in such aspects as the participants and in customer requirements.

A product development chain can be classified according to who owns, and how much it owns, the design authority, that is the authorities to change the design. For example, Clark & Fujimoto (1991) classify suppliers into three different categories:

1. Suppliers that provide "supplier proprietary parts", in which suppliers own all of the part design authority.
2. Suppliers that provide "black box parts", in which suppliers own partial part design authority. Typically, part functions and general features are specified by the assembler, while the suppliers own the authority to design part details based on these pre-specified functions/features.
3. Suppliers that provide "detail-controlled parts", in which the assembler owns the whole part design authority. This would be the case where the supplier acts as a sub-contractor to the assembler, manufacturing the part to the detailed design received from the assembler and then delivering the completed part to the assembler.

In a similar vein to that of Clark & Fujimoto, Twigg (1998) compares research on suppliers to the automotive industry and classifies these suppliers into eight categories. They are (1) assembler in-house design, (2) concept and specialist engineering (3) proprietary parts, (4) black box: critical specifications, (5) detail-controlled: functional parts, (6) detail-controlled: aesthetic parts (7) less complex parts and (8) standard catalogue parts.

The research of both Clark & Fujimoto and Twigg would indicate that, in a product development chain, design tasks can be decomposed and different levels of design authority can be distributed to the suppliers. The actual distribution depending on such aspects as the characteristics of the part and the capabilities of the suppliers.

In terms of Concurrent Engineering, design object decomposition has been used as a basis for exploiting concurrency in product development (Arai and Iwata, 1990; Kusiak and Wang, 1993). Prasad (1996a) suggests that the level of concurrency can be maximized if a product can be decomposed into components in a way that their inherent dependencies are minimized. Prasad (1996b) also suggests that a product can be decomposed in a number of ways including physical-based decomposition, function-based decomposition, and activity-based decomposition.

Recently, there has been emphasis on the use of modularity where products are configured by combining modules together in different permutations (O'Grady, 1999). Modularity "involves the assembly of products from a set of modules. Each module can be extremely complex internally, but externally must have a set of clearly defined interfaces, which specify how that module can link to other modules" (O'Grady, 1999). For design task decomposition, design tasks are to be decomposed into several "design components", or modules, and the development of these modules can be performed by individual suppliers. Search algorithms can be used to search for a set of compatible modules that address customer requirements, while operating within constraints (O'Grady and Liang, 1998).

While much attention has recently been focused on issues in the supply chain, the area of cooperative product development with a chain of suppliers remains virtually ignored. The need for a research program in the product development chain is given urgency by three developments:

- The continual pressure to reduce product development time, to reduce costs and to improve product offerings.
- The tendency for assemblers to "outsource" an increasing proportion of their manufacturing (Kavanagh,1997; PA Consulting Group,1996).
- The development of communications technologies, particularly Internet technologies, that allow for a much improved level of communication between assemblers, customers, and their product development chain.

The promise is that of *integrated product development (IPD)* involving assemblers, customers and suppliers. Such a level of integration would offer improved designs, with substantially reduced product development times, while significantly reducing costs.

However there are significant research issues to be addressed before such integration becomes a viable proposition, certainly for more complex products. Perhaps the central research issue is that of devising a suitable methodology to divide the design task into independent but compatible smaller design tasks that can be performed separately. This is important in that the division of design tasks provides the basis for operating a product development chain. This paper aims to address this issue.

The central problem addressed in this paper can be stated as:

Given a design target object and a set of associated customer requirements, how can we divide this design object into several smaller but independent design objects (or design entities), each associated with a set of design requirements, so that their design can be performed separately and simultaneously by different designers in a product development chain. These independently designed objects, when brought together, can be assembled to meet the given customer requirements.

More specific research issues that are encompassed by this central problem are:

1. How can a design object be decomposed into several smaller, independent but compatible design objects (or design entities)?
2. How can the design requirements for a design object be divided into the decomposed design objects as their individual design requirements?
3. How can suitable interfaces between the design objects be devised?

The format of this work is as follows. First, a new algorithm is presented that distributes design requirements into sub-entities. Second, another algorithm is developed that generates the interface requirements between the different sub-entities. Third, the initial implementation of the combined algorithms in a system called DODA is described.

2. PROPOSED DESIGN OBJECT DECOMPOSITION

2.1. SYMBOLS AND NOTATION

ξ	requirement operator: inherit
ψ	requirement operator: distribute
ϖ	requirement operator: generate
E	a design entity
R	a design requirement
$E\{R\}$	design requirements for design entity E
e_i	a sub-level design entity for design entity E
$e_i\{R\}$	design requirements for design entity e_i
$\{e_i\}$	all sub-level design entity for design entity E

I	an interface requirement
$e_i\{I\}e_j$	interface requirements for design entity e_i to connect to e_j
$M\{e_i\}$	interface requirement matrix for $\{e_i\}$

2.2. DESIGN OBJECT DECOMPOSITION

Conceptually, a design object can be viewed as a design problem that seeks a solution that satisfies certain requirements. Design object decomposition divides this design problem into several sub-level problems where each of these problems can be solved separately. In a product development chain, design object decomposition refers to the distribution of design authority to suppliers so that components (or modules) can be developed separately by the suppliers.

Typically, a design object can be decomposed into several design entities, and these design entities can be further decomposed into lower-level design entities. By repeating this process, a design entity tree can be constructed (Figure 2). Note that a design object can be decomposed into many different design entity trees. To determine how a design object can be decomposed depends on a number of factors including the physical characteristics of the object, the functional characteristics of the object and the suppliers' capabilities.

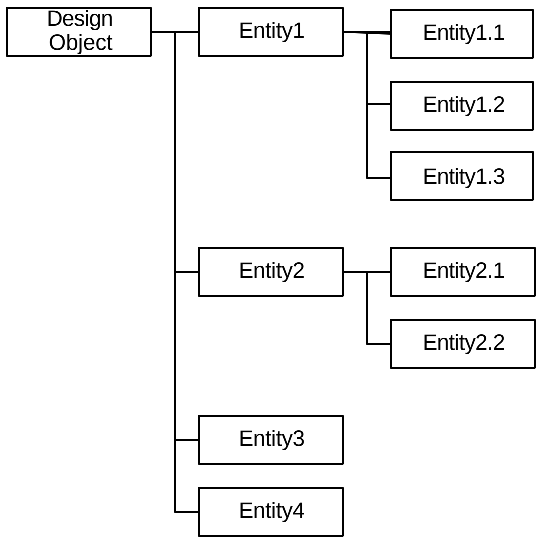


Figure 2: An Example of a Design Entity Tree

One major challenge in carrying out this decomposition is the decomposition and distribution of design requirements. The top-level design requirements (usually the customer requirements) need to be decomposed and distributed to lower-level design entities as their design requirements. Another challenge is that of compatibility. Since design is to be performed separately, it is important to make sure that individually designed components can be put together properly. A set of design requirements for the interface design between each connected design entity is therefore needed. The following two sections discuss these two challenges.

2.3. DESIGN REQUIREMENT DISTRIBUTION

Design requirements are "demands" and "wishes" that clarify the design task in the space of needs (Pahl and Beitz, 1988). Top level design requirements are usually obtained from customers. For example, a customer might ask for certain required functions (functional requirements), for certain performance (performance requirements), or for requirements associated with cost and reliability. In design decomposition, these design requirements have to be decomposed and distributed to lower level design entities.

2.3.1. Specifying Requirements

Specifying requirements is a difficult task due to the complicated nature of requirements. Top level requirements are usually very general and ambiguous and therefore need to be specified in more detail before being distributed to lower level entities. For example, to design a computer system for intense graphic applications, customers could ask for a "premiere" graphic performance as a requirement. A designer, by using his/her expertise and experience, should be able to translate this requirement into more specific requirements, such as, for example, (1) support of OpenGL and DirectX (2) fill rates greater than 300 Mpixel/s (3) onboard memory greater than 128 MB (4) support for AGP interface, and so on.

Specifying requirements therefore involves translating customer requirements into specific design requirements that can be directly used for product design. Expertise and experience on both the market and manufacturing are required to perform this task. In addition, this interpretation needs to be updated rapidly with the most recent market/technology information, especially for those products that change rapidly. Work in this area is beyond the scope of this paper, instead the paper concentrates on the decomposition of the specific requirements.

2.3.2. Proposed Design Requirement Decomposing Operator

Once the general, ambiguous design requirements have been translated into more specific requirements, they can be decomposed and distributed to sub-level design entities by design decomposition operators. Two different requirement decomposition operators can be identified:

1. **Inherit ($\mathbf{x}$)**: where a design requirement is inherited by one or more sub-level entities. For example, in the design of a computer system, a customer requirement “minimum storage capacity = 10 GB” can be inherited to the sub-level entity consisting of a hard drive.
2. **Distribute ($\mathbf{y}$)**: where a requirement is separated into several sub-level requirements and distributed to several sub-level entities. For example, in designing a laptop computer system, a customer requirement “maximum weight = 4 kg.” can be distributed to the weight requirements of its sub-level entities such as a display, a keyboard, a hard drive, and so on, so that the total weight in these sub-entities is equal to, or less than, 4 kg.

   Applying the distribute operator requires the designer to determine how (a *distribution scheme*) the requirement is to be separated and how much (*weights*) to distribute to each of the sub-level entity. A set of *distribution regulators* can be derived based on the distribution scheme, and these serve as constraints for applying those weights. For example, to distribute a requirement “Cost less or equal than 1000” to five sub-level design entities $\{e_1, e_2, ... e_5\}$, the distribution scheme “addition” can be used. Table I shows an example of applying the addition scheme.

Table I: An Example of Applying Distribution Scheme “Addition”

Requirement: Cost ≤ 1000
Distribution Scheme: Addition
Regulator: w1+w2+w3+w4+w5 ≤ 1

Sub-level Entities		**Requirement**
e_1	w1 = 0.1	Cost ≤ w1 × 1000
e_2	w2 = 0.2	Cost ≤ w2 × 1000
e_3	w3 = 0.1	Cost ≤ w3 × 1000
e_4	w4 = 0.2	Cost ≤ w4 × 1000
e_5	w5 = 0.4	Cost ≤ w5 × 1000

As another example, to distribute a requirement "reliability greater than 99%" to five sub-level design entities {e_1, e_2, ... e_5}, the distribution scheme "multiplication" can be used. Table II shows an example of applying the multiplication scheme.

Table II: An Example of Applying Distribution Scheme "Multiplication"

Requirement: Reliability $\geq$ 98%
Distribution Scheme: Multiplication
Regulator: $r1 \times r2 \times r3 \times r4 \times r5 \geq 98\%$

Sub-level Entities		Requirement
e_1	r1 = 0.99	Reliability $\geq$ r1
e_2	r2 = 0.999	Reliability $\geq$ r2
e_3	r3 = 0.999	Reliability $\geq$ r3
e_4	r4 = 0.995	Reliability $\geq$ r4
e_5	r5 = 0.995	Reliability $\geq$ r5

Every top-level requirement can be decomposed and distributed, and these lower level requirements can be further decomposed and distributed. As a result a requirements network for each top-level design requirement can be established, as shown in Figure 3. A requirement network provides a framework to illustrate how a top-level requirement is to be decomposed, and can help in modifying or optimizing the decomposition.

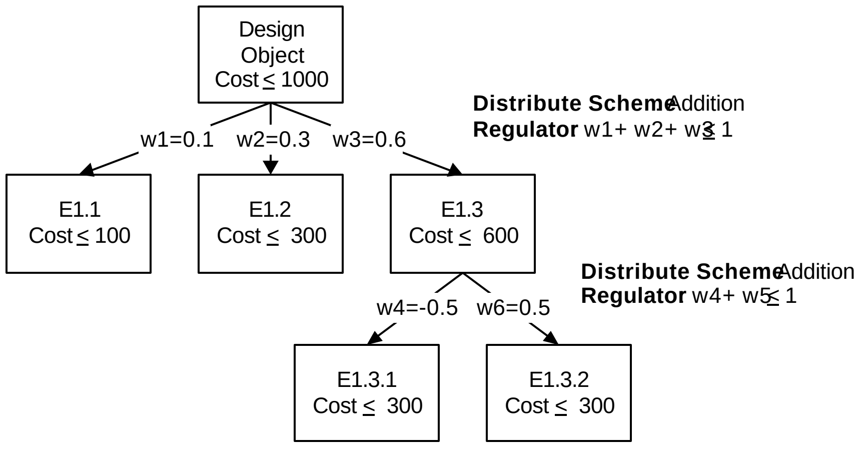


Figure 3: An Example of a Requirements Network

2.4. Proposed Interface Requirements Generation

In contrast to the decomposing operators that define relationships between design entities at different levels, the interface design requirements operator "generate (ϖ)" defines the relationship between design entities at the same level. In other words, it specifies the interfaces between design entities.

Typically, the generate (ϖ) operator is needed when an entity is divided into several entities, and a set of interface requirements then needs to be generated in order to specify how these sub-level entities can be properly assembled. For example, to divide a computer system into an entity "computer" and another entity "monitor", interface requirements such as "power connection requirements" and "data connection requirements" have to be generated for both the computer and monitor. These requirements serve as a guideline in developing compatible components.

When an entity is divided into n sub-level entities, there is a need to generate totally $n^2 -1$ sets of interface requirements. An interface requirement matrix (M_I) can be used to illustrate these requirements, as shown below.

$$M\{e_i\} = \begin{array}{c} \\ e_1 \\ e_2 \\ : \\ e_n \end{array} \begin{array}{c} \begin{array}{cccc} e_1 & e_2 & \dots & e_n \end{array} \\ \begin{bmatrix} null & e_2\{I\}e_1 & \dots & e_n\{I\}e_1 \\ e_1\{I\}e_2 & null & \dots & e_n\{I\}e_2 \\ : & : & null & : \\ e_1\{I\}e_n & e_2\{I\}e_n & \dots & null \end{bmatrix} \end{array}$$

Where $M\{e_i\}$ is the interface requirement matrix for $\{e_i\}$ and
$e_i\{I\}e_j$ is the interface requirements for e_i to connect to e_j

3. DESIGN OBJECT DECOMPOSITION ALGORITHM (DODA)

The previous sections have described the processes of defining interfaces between entities and of distributing requirements to sub-entities. These process can now be included in a design object decomposition algorithm (DODA) that specifies how a design object can be decomposed to form a design entity tree and how interfaces between the entities can be determined. An overview of DODA is given in this section while the operation of DODA is more fully described in an illustrative example in a following section.

Step 1: In a design entity tree, select a design entity E with its design requirement E{R} to be decomposed.

Step 2: For each design requirement R where $R \in E\{R\}$, if required, specify this requirement and update E{R} with the specified design requirements.

Step 3: For this selected design entity E, determine its sub-level entities $\{e_i\} = \{e_1, e_2, e_3, ..., e_n\}$, and add these sub-level entities into the design entity tree as the branches of E.

Step 4: Define interface requirements (as described in section 2.4)

Step 4-1: Generate interface requirement matrix for $M\{e_i\}$.

Step 4-2: Determine the interface requirements $e_i\{I\}e_j$ in $M\{e_i\}$.

Step 5: Distribute E{R} (as described in section 2.3)

Step 5-1: Select a design requirement R where $R \in E\{R\}$

Step 5-2: For this selected R,

1. Determine which requirement decomposition operators (ξ or ψ) to apply
2. Determine distribution scheme (if applicable)
3. Determine distribution weightings (if applicable)
4. Add these newly generated R into its associated $e_i\{R\}$.

Step 5-3: Repeat step **5-1** to **5-2** until all $R \in E\{R\}$ has been processed.

Step 6: Repeat **step 1** to **step 5** until no design entity in the design entity tree could be further decomposed

3.1. ILLUSTRATIVE EXAMPLE USING DODA ALGORITHM

A simple example of decomposing a design task for a new computer system is now used to illustrate the operation of the DODA algorithm. In order to reduce space requirements, this example is necessarily simplified. This example decomposes the design task into four design entities, and distributes customer requirements into these entities. The example is of a personal computer system that has the following design (or customer) requirements:

Design Object: A new computer system

Design (Customer) Requirements: { Premiere Graphic Performance;
Premiere Office Performance;
Premiere Network Performance;
Good Portability
}

The DODA algorithm can now proceed with the 6 steps, as outlined above.

Step 1: Select design entity (E where E = the computer system) to be decomposed:

E = A new design of computer system
E{D} = { Premiere Graphic Performance;
Premiere Office Performance;

Premiere Network Performance;
Good Portability
}

Step 2: By examine an expertise database, or by using the expertise of the designer, E{D} can be further specified and updated as:

E{D} = { Support OpenGL and DirectX
Fill Rates > 300 Mpixel/s
Onboard Memory > 128 MB
Support AGP interface
Minimum Clock Frequency: 550 MHz
High Speed Communication Port
weight < 4 kg
}

Step 3: For this selected design entity (E where E = Computer System), four sub-level entities (e_1, e_2, e_3 and e_4) are determined based on the product characteristics and suppliers' condition , they are:

e_1 : Mother board/Memory/Communication Port combo
e_2 : CPU
e_3 : Monitor
e_4 : Video Card

Step 4: Interface requirements:

Step 4-1: Generate interface requirement matrix for M{e_i}.

$$M\{e_i\} = \begin{array}{c|cccc} & e_1 & e_2 & e_3 & e_n \\ e_1 & null & e_2\{I\}e_1 & e_3\{I\}e_1 & e_4\{I\}e_1 \\ e_2 & e_1\{I\}e_2 & null & e_3\{I\}e_2 & e_4\{I\}e_2 \\ e_3 & e_1\{I\}e_3 & e_2\{I\}e_3 & null & e_4\{I\}e_3 \\ e_4 & e_1\{I\}e_4 & e_2\{I\}e_4 & e_3\{I\}e_4 & null \end{array}$$

Where M{e_i} is the interface requirement matrix for {e_i} and
e_i\{I\}e_j is the interface requirements for e_i to connect to e_j

Step 4-2 : Determine the interface requirements e_i\{I\}e_j in M{e_i}.

Interface requirements for e_1 (motherboard combo)
e_1\{I\}e_2 = {Interface specification to connect to CPU}
e_1\{I\}e_3 = {}
e_1\{I\}e_4 = {standard AGP slot}

Interface requirements for e_2 (CPU)
e_2\{I\}e_1 = {Interface specification to connect to motherboard}

$e_2\{I\}e_3 = \{\}$
$e_2\{I\}e_4 = \{\}$

Interface requirements for e_3 (Monitor)
$e_3\{I\}e_2 = \{\}$
$e_3\{I\}e_3 = \{\}$
$e_3\{I\}e_4$ = {Standard Analog RGB (DB15c male) }

Interface requirements for e_1 (Video card)
$e_4\{I\}e_1$ = {Stand AGP Interface}
$e_4\{I\}e_2 = \{\}$
$e_4\{I\}e_3$ = {Standard Analog RGB (DB15c female)}

Step 5: Distribute E{D} to $\{e_i\}$

Step 5-1 : Select a D from E{D} where D = "weight < 4 kg"

Step 5-2 : For this D:

(1) Apply requirement decomposition operator ψ (distribution)
(2) Determine distribution scheme = addition
(3) Distribution weighting {w1, w2, w3, w4} = {0.2, 0.1, 0.6, 0.1}
(4) Add D = "weight < 0.8 kg" to $e_1\{D\}$;
Add D = "weight < 0.4 kg" to $e_2\{D\}$;
Add D = "weight < 2.4 kg" to $e_3\{D\}$;
Add D = "weight < 0.4 kg" to $e_4\{D\}$;

Step 5-3 : Other $D \in E\{D\}$ can be decomposed as follows:

- Apply requirement decomposition operator "inherit (ξ)" to D = "Support OpenGL and DirectX". Add D = "Support OpenGL and DirectX" to $e_1\{D\}$, $e_2\{D\}$ and $e_4\{D\}$.
- Apply requirement decomposition operator "inherit (ξ)" to D = "Fill Rates > 300 Mpixel/s". Add D= "Fill Rates > 300 Mpixel/s" to $e_1\{D\}$ and $e_4\{D\}$.
- Apply requirement decomposition operator "inherit (ξ)" to D = "Onboard Memory > 128 MB". Add D = "Onboard Memory > 128 MB" to $e_1\{D\}$.
- Apply requirement decomposition operator "inherit (ξ)" to D = "Support AGP interface". Add D = "Support AGP interface" to $e_1\{D\}$ and $e_4\{D\}$.
- Apply requirement decomposition operator "inherit (ξ)" to D = "Minimum Clock Frequency = 550 MHz". Add D = "Minimum Clock Frequency = 550 MHz" to $e_1\{D\}$, $e_2\{D\}$ and $e_4\{D\}$.
- Apply requirement decomposition operator "inherit (ξ)" to D = "High Speed Communication Port". Add D = "= "High Speed Communication Port" to $e_1\{D\}$.

Step 6: Stop since all $D \in E\{D\}$ has been processed.

An example result of this decomposition can be:

Design requirements for e_1 (motherboard combo) = {
Fill Rate > 300 Mpixels
High Speed Communication Port
Interface specification to connect to CPU
Minimum Clock Frequency = 550 MHz
Onboard Memory > 128 MB
Standard AGP slot
Support AGP interface
Support OpenGL and Direct X
Weight < 0.8 kg
: }

Design requirements for e_1 (CPU) = {
Interface specification to connect to motherboard
Minimum Clock Frequency = 550 MHz
Support OpenGL and Direct X
Weight < 0.4 kg
: }

Design requirements for e_3 (Monitor) = {
Weight < 2.4 kg
Standard 15-pin monitor connector;
: }

Design requirements for e_4 (Video Card) = {
Fill Rate > 300 Mpixels
Minimum Clock Frequency = 550 MHz
Standard AGP Interface
Standard Analog RGB (DB15c female)
Support AGP interface
Support OpenGL and Direct X
Weight < 0.4 kg
: }

This illustrative example, though necessarily simplified, shows how the DODA algorithm can be used to progress from the initial, and often vague, customer requirements to the more detailed design requirement for each entity, with interfaces being defined for each entity.

4. INITIAL IMPLEMENTATION OF DODA

The DODA algorithm has been implemented in a prototype form in a computer system conveniently called DODA. This is implemented as a client-server system with a web browser as a client. The server part of DODA runs under Windows NT Server with the Internet Information Server as the web server. The

DODA algorithms are coded as Active Server Pages on the server. The architecture of the prototype implementation of DODA is shown in Figure 4.

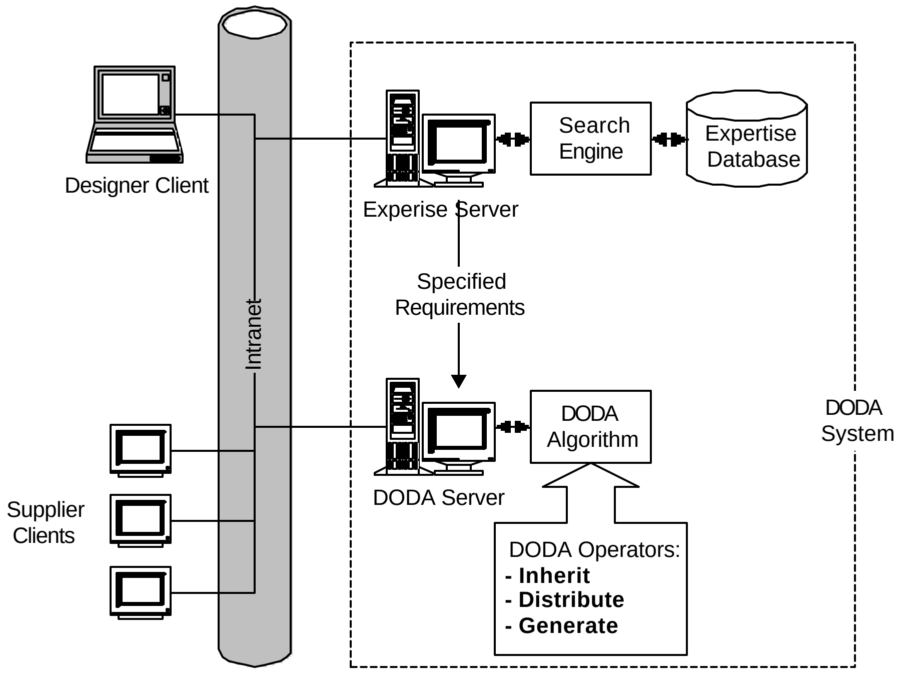
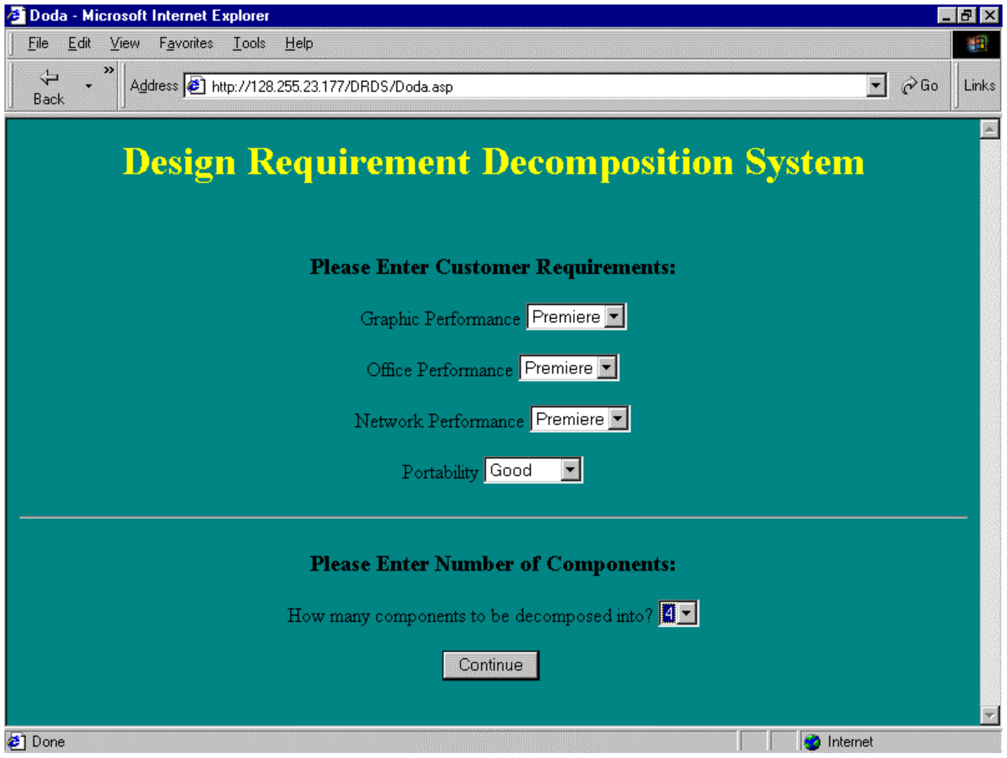


Figure 4: Architecture of the Initial Implementation of DODA

Designers access the DODA system as a client using a web browser and search though the expertise database containing customer requirements. These are compiled over a period of time. The expertise database responses with specified design requirements and posts them to the DODA server. An alternative is to use the expertise and experience of the designer to complete this initial mapping. Designers are then able to decompose these design requirements, using the DODA operators, into sub-level design requirements and to then distribute them to the component suppliers.

Let us now consider a scenario where we are still concerned with the design of the computer system as described in the previous section. The customer requirements are, as described above, classified into the four categories of graphic performance, office performance, network performance and portability. The designers can select settings for these four categories using the client web browser. The selected options then go to an expertise server and search through an expertise database for the specified requirements. This setting gives the designers the ability to respond quickly to diversified demands and interpret those demands into specific requirements.

In addition to the customer requirement options, the designer also needs to determine the number of sub-level components that the design object is to be decomposed into. This determination depends on the characteristics of the design object and the conditions of the suppliers. For example, a designer might want to have an integrated video display/capture card instead of separate video display and video capture cards. Figure 5 shows the interface in the DODA system for entering these options.


Doda - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back
Address http://128.255.23.177/DRDS/Doda.asp
Go
Links
Continue
Done
Internet

Design Requirement Decomposition System

Please Enter Customer Requirements:

Graphic Performance Premiere

Office Performance Premiere

Network Performance Premiere

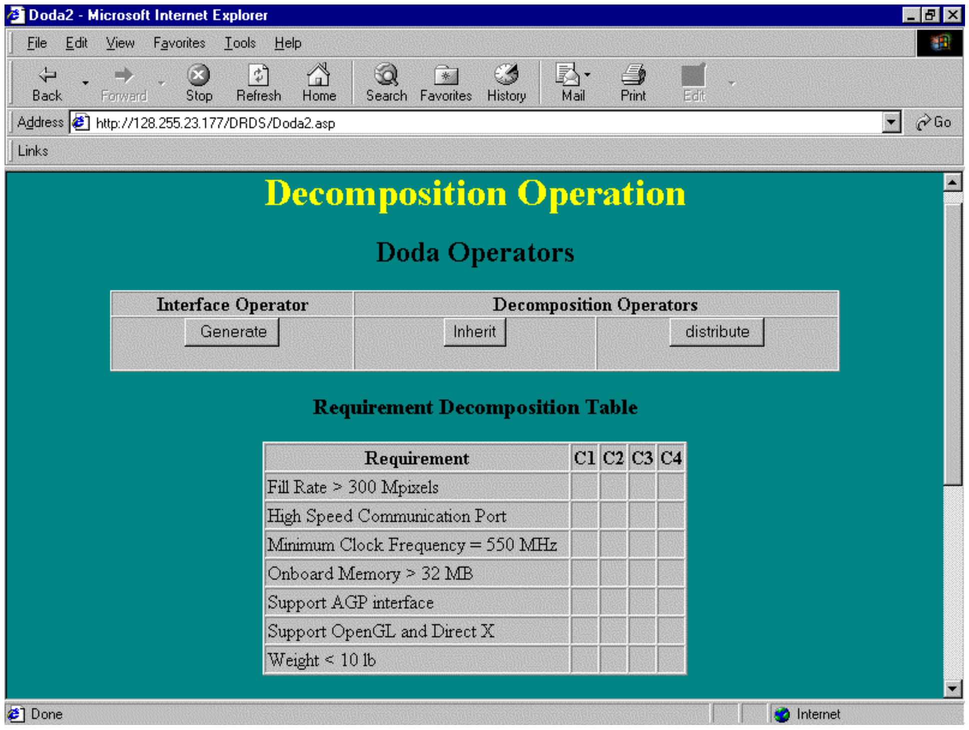
Portability Good

Please Enter Number of Components:

How many components to be decomposed into? 4

Figure 5: Entering Customer Requirements in DODA

Once the customer requirements and the number of component have been determined, specified requirements can be retrieved from the expertise database. The DODA algorithm then can be applied to decompose these specified requirements. Figure 6 shows the interface of this DODA system and the retrieved requirements.


Doda2 - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search Favorites History Mail Print Edit
Address http://128.255.23.177/DRDS/Doda2.asp
Go
Links
Done
Internet

Decomposition Operation

Doda Operators

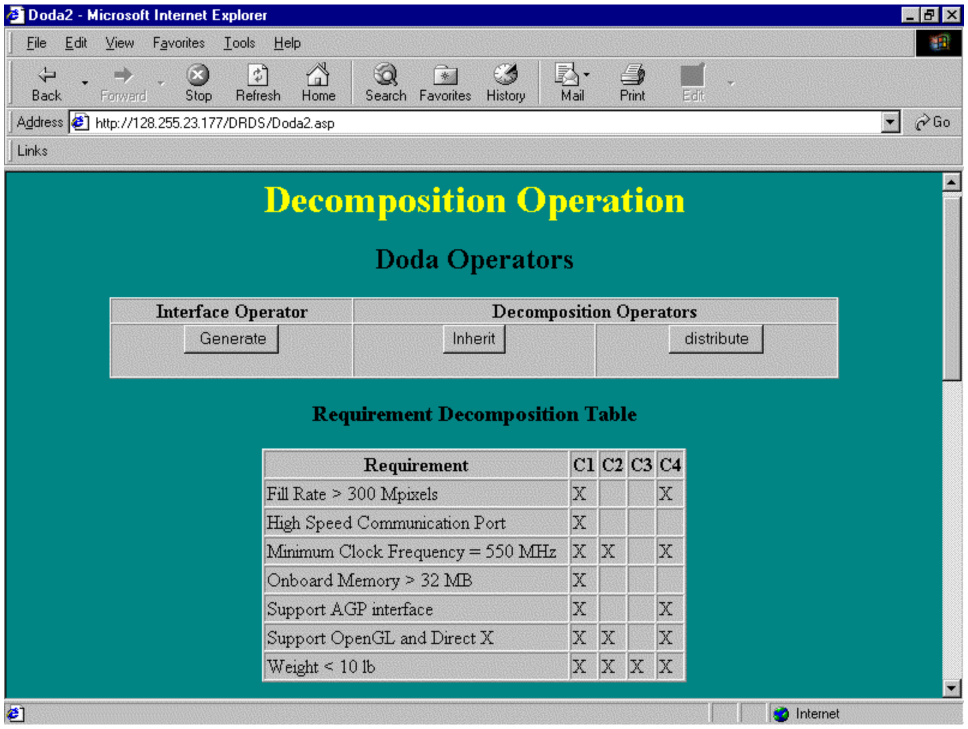
Interface Operator	Decomposition Operators	
Generate	Inherit	distribute

Requirement Decomposition Table

Requirement	C1	C2	C3	C4
Fill Rate > 300 Mpixels				
High Speed Communication Port				
Minimum Clock Frequency = 550 MHz				
Onboard Memory > 32 MB				
Support AGP interface				
Support OpenGL and Direct X				
Weight < 10 lb				

Figure 6 DODA Interface and Retrieved Requirements

Designers are able to apply DODA operators “Generate”, “Inherit” and “Distribute”, as well as to specify which decomposition scheme, and what weighting to apply on each specified requirement. For example, designers might decide to use operator “inherit” to decompose requirement “Support AGP interface” to component 1 (a motherboard) and component 4 (a video card). Or the designers might want to apply DODA operator “distribute” to distribute requirement “weight < 4 kg” to component 1, 2, 3, and 4 by using “addition” scheme with weighting “0.1”,”0.4”,”0.4” and “0.1”. An “X” is shown in the requirement-component matrix to indicate where a particular requirement is decomposed. Figure 7 shows this requirement-component matrix.


Doda2 - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search Favorites History Mail Print Edit
Address http://128.255.23.177/DRDS/Doda2.asp
Go
Links
Internet

Decomposition Operation

Doda Operators

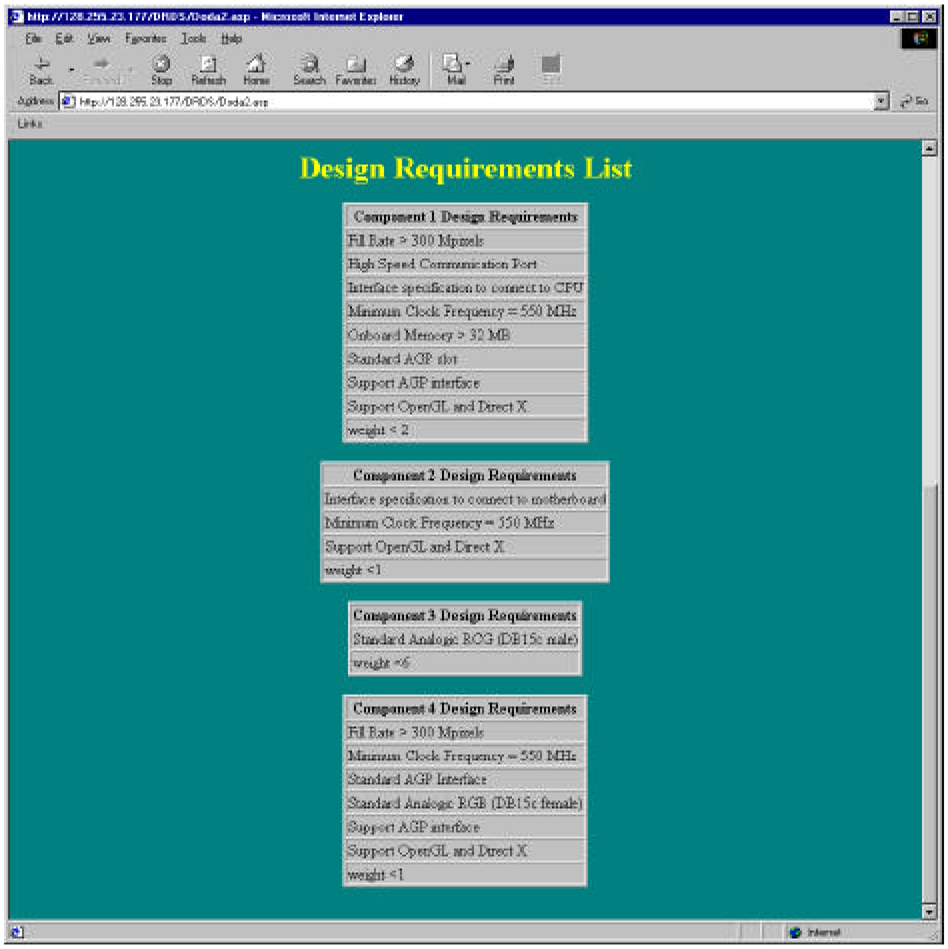
Interface Operator	Decomposition Operators	
Generate	Inherit	distribute

Requirement Decomposition Table

Requirement	C1	C2	C3	C4
Fill Rate > 300 Mpixels	X			X
High Speed Communication Port	X			
Minimum Clock Frequency = 550 MHz	X	X		X
Onboard Memory > 32 MB	X			
Support AGP interface	X			X
Support OpenGL and Direct X	X	X		X
Weight < 10 lb	X	X	X	X

Figure 7 Requirement-Component Decomposition Matrix

A final decomposed result is shown as Figure 8 to illustrate the design requirements for each sub-level entity. These sub-level entities can be individual components or modules. These are the final design requirements that are ready to be sent to the part suppliers.



Design Requirements List

Component 1 Design Requirements
Fill Rate > 300 Mpixels
High Speed Communication Port
Interface specification to connect to CPU
Minimum Clock Frequency = 550 MHz
Onboard Memory > 32 MB
Standard AGP slot
Support AGP interface
Support OpenGL and Direct X
weight < 2

Component 2 Design Requirements
Interface specifications to connect to motherboard
Minimum Clock Frequency = 550 MHz
Support OpenGL and Direct X
weight <1

Component 3 Design Requirements
Standard Analogic RGB (DB15c male)
weight <6

Component 4 Design Requirements
Fill Rate > 300 Mpixels
Minimum Clock Frequency = 550 MHz
Standard AGP Interface
Standard Analogic RGB (DB15c female)
Support AGP interface
Support OpenGL and Direct X
weight <1

Figure 8: Design Requirement List for Each Sub-Level Entity

5. SUMMARY AND CONCLUSIONS

While supplier involvement in the design of a product (in what can be called a product development chain) is generally regarded as being of substantial benefit, there has been little work done on developing methodologies that can be used to support such supplier involvement. Unfortunately, product development chains can be inordinately complex due to the often large mesh of suppliers, the fact that suppliers may be involved in many different product development chains, and due to the dynamic nature of the product development chain with wide variations in customer requirements over time.

This paper aims to address the area of product development chains by developing methodologies that can be used to translate overall design requirements into design requirements for components or modules.

These are the final design requirements that are ready to be sent to the part suppliers. Associated with these design requirements are sets of interface specifications that specify the interfaces between components or modules.

The key contributions of this work are three-fold. First, a new algorithm is presented that can distribute design requirements into sub-entities. Second, another algorithm is developed that generates the interface requirements between the different sub-entities. Third, the initial implementation of the combined algorithm in a system called DODA is described. Such work provides the basis for further work in this important area of product development chains.

6. REFERENCES

Arai, E., K. Iwata. (1990), *CAD with Design Specification Decomposition and Its Application.* Annals of the CIRP. Vol. 39 (Jan.), pp. 121-124.

Carlisle, J. A. And Parker, R. C. (1989), *Beyond Negotiation: Redeeming Customer-Suppliers Relationships*, John Wiley & Sons, Chichester, p. 127

Clark, K. B. And Fujimoto, T. (1991), *Product Development Performance,* Harvard Business School Press, Boston, MA.

Clark, P. A. and Starkey, K. (1988), *Organization Transitions and Innovation-Design*, Pinter, London.

Dowlatshahi, S. (1998), *Implementing Early Supplier Involvement: a Conceptual Framework,* International Journal of Operations & Production Management, Vol. 18 No. 2, pp. 143-167.

Kavanagh, J. (1997), *Trend towards contracting-out*, the Financial Times, 4 June.

Kusiak, A., and J. Wang. (1993), *Decomposition of the Design Process*, Journal of Mechanical Design, Transactions of the ASME, Vol. 115, No. 4 (December), pp. 687-695.

O'Grady, P. (1999), The Age of Modularity – Using the World of Modular Products to Revolutionize Your Corporation, Adams and Steele Publishers, U.S.A.

O'Grady, P; Liang,WY (1998), *An Internet-Based Search Formalism for Design with Modules*, Computers & Industrial Engineering, No. 35 : (1-2), pp. 13-16.

PA Consulting Group (1996), *Management Summary* – International Strategic Sourcing Survey.

Pahl, G. and Beitz, W. (1988), Engineering Design, Springer-Verlag, New York.

Poirier, C. C. and Reiter, S. E. (1996), *Supply Chain Optimization: Building the Strongest Total Business Network,* Berrett-Koehler Publishers, Inc.

Prasad, B., (1996a), Concurrent Engineering Fundamentals. Volume II, Prentice Hall, New Jersey.

Prasad, B., (1996b), Concurrent Engineering Fundamentals. Volume I, Prentice Hall, New Jersey.

Twigg, D. (1997), *Design chain management,* in Slack, N. (Ed.), *The Blackwell Encyclopaedic Dictionary of Operations Management*, Blackwell, Oxford.

Twigg, D. (1998), *Managing Product Development within a Design Chain,* International Journal of Operations & Production Management, Vol. 18, No. 5, pp. 508-524.