View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by Okayama University Scientific Achievement Repository

Engineering

Industrial & Management Engineering fields

Okayama University *Year* 2005

Visualization for management of electronics product composition

Iwata Kenichi
Okayama University

Mariko Sasakura
Okayama University

Susumu Yamasaki
Okayama University

This paper is posted at eScholarship@OUDIR : Okayama University Digital Information Repository.

http://escholarship.lib.okayama-u.ac.jp/industrial_engineering/8

Visualization for Management of Electronics Product Composition

Kenichi Iwata, Mariko Sasakura, Susumu Yamasaki
Department of Computer Science,
Graduate School of Natural Science and Technology, Okayama University
{iwa, sasakura, yamasaki}@momo.it.okayama-u.ac.jp

Abstract

There are some systems called Supply Chain Management system or Value Chain Management system that manage production. It is a powerful tool in normal cases, but in a problem such that some parts are out of stock, it can solve the problem only by simple solutions, like postponing shipping of the product. Because it does not have enough information about production and functions to use the various information. Our research is concerned with a system to integrate information about production and show a solution to help users to judge which way is better to solve the problem. We implemented a prototype system. It takes inputs of some information that were not integrated in one place in former systems, but distributed among systems, people, or sections. It shows a solution for a problem making use of the information integrated in the system. The solution comes as process of reasoning to help user to judge what is the best to do in the case. We also implemented the user interface to show the process of reasoning.

1. Introduction

We already have several commercial systems to support production in factories. They are called Supply Chain Management system (SCM) or Value Chain Management System. They manage accepting an order, arranging various parts, composing a production, ensuring the date of delivery and shipping it to a customer. They reduce inventories, provide productions to market speedy and improve customer's satisfaction.

These systems support the best suited production when everything is going well. However, when a problem is occurred in any place of production process, it will be hard to judge what is the best solution for the problem.

Now we think about an example. If a kind of parts were out of stock, how does it solved? The factory can just postpone to compose a product, or they also can use other part that is compatible with the part. The decision have to be done by the business policy, but it was almost impossible and tend to be stuck into a local optimality, because of lack of information about production for a manager.

In this research, we propose a system to stock and manage information about production, to help judging the best solution in the production process. We also provide an interface to show the result visually to help users understand the reason why the system propose the solution as the best one. We now present a prototype system based on our proposal in this paper.

Our research is based on reasoning procedure is given in [14], which follows the notation of [6] in relation to the fixpoint semantics as in [13] and [16]. We have studied an application of the reasoning [9]. It assists to choose a way of parallelizing a program based on knowledge for parallelizing compilers [1].

The construction of a managing system with reference to PC composition processes is briefly formulated in [15]. The paper demonstrates abstract structures of its semantics and visualization. In this paper, we develop a prototype system based on the structure with another visualization.

In our system, the user interface is important. To let users judge if the result is reasonable, and choose the best solution that is suited to the administrative policy, we demonstrate the process of reasoning as a set of trees.

There are various researches about visualization of trees [2], [7], [8], [10], [11], [12]. In the system that we describe in this paper, we use GraphViz [3], [4] to visualize trees. It is a group of tools implemented by various methods of visualization. The figures of a tree we show in this paper are displayed by ZGRViewer[1].

[1] It is developed by Dr. Emmanuel Pietriga.
http://zvtm.sourceforge.net/zgrviewer.html

Proceedings of the Ninth International Conference on Information Visualisation (IV'05)
1550-6037/05 $20.00 © 2005 **IEEE**

2. Management of Electronics Product Composition

2.1. Aspects of Electronics Production Area

In this paper, we think about the electronics production area. A specific characteristic in this area is that semiconductor parts need huge investment to be produced. Therefore, the most of vendors of the parts are big companies, compared with other area like the car production.

There is a method called "kanban" system [5] in order to compress inventories, and also reduce the risk to run out of parts. It is mainly used in the car production area. A car producer has only few parts stocks, usually just for one or two days production. Every part comes with a tag called "kanban", and when a part is used on a production line, the kanban is collected. The collected kanbans will be sent to parts vendors, then they receive kanbans and start producing parts, bringing them to the production line very often to avoid shortage of parts.

In the area of electronics production, the "kanban" system has difficulty to be applied. Because the most of parts vendors are too big to follow the system.

Therefore when producers compress inventories, they will have risks of running out of parts. This can be a problem on the production line.

2.2. The Work Flow of the System

Being out of stock of a part results in delay of finishing products composition. To solve this problem, there are some solutions at accepting an order, production or shipping.

The first solution is to absorb the problem at the accepting an order. It means asking the customer to agree the delay of the product. If the customer accepts the delay of the ordered product, the problem will be solved.

Another solution is to absorb it at the production section. When the design of the product is being done, designers make sets of compatible parts for a part. The set contains some parts which are compatible, such as some resistors that have the same resistance with different accuracy, memory boards that have various speed memory chips, and CPUs with different clock-speeds. In a normal case, the production section uses the "default" part, which is the cheapest one. When the default part is out of stock, they can choose another part from the set, usually it is of higher grade and more expensive than the default part.

Yet another solution is to absorb it at the shipping section. A higher speed shipping way, like express mail, can solve the delay of production.

In this research, we handle the case of problem that is out of stock.

The system has four subsystems which deal with the following information as their databases respectively.

- Design information
- Order
- Shipping information
- Inventory information

The design information contains a parts list for a product, and also information about part compatibility.

The order means what product is ordered.

The shipping information is the number of days and the cost that are needed to ship a product from the factory to the customer.

The inventory information is a quantity of every kind of parts that are stocked in the factory.

The output of the system is a set of solutions for an order. If there is no problem on producing the ordered product, the system shows a solution without condition. If there is a problem, the system shows the candidates for the solution with conditions which indicate a shortage of parts. A manager who has the responsibility for the product can know the situation about the production, and judge which solution is the best suited to the administrative policy. The administrative policy would be like the priority among many candidates such as the cheaper cost, the quicker delivery, and so on.

2.3. An Example of the System Flow

Our system handles information as a logic program. We use the way to write a logic program as in [14]. We give an example in this subsection.

The design information

We think of a personal computer. A PC has some parts to be built. The following clause states that:

$$\begin{aligned} PC \leftarrow \quad & board, power\ supply, \\ & body\ case, hard\ drive, \\ & CDROM\ drive, keyboard, mouse \end{aligned}$$

The each part has some parts to be built.

$$\begin{aligned} board \leftarrow \quad & bare\ board, A_1, A_2, chip\ resistor, \\ & CPU, bus\ bridge, memory, powerpod, \\ & USB\ controller \end{aligned}$$

where A_1 is a bypass condenser for a place on a board, and A_2 is a bypass condenser placed for another place on the board.

Some parts have a list including more than one part to be applicable. The top item of the list is the default part. In the following, the negation sign "$\sim$" is operationally interpreted as "negation as failure" as studied in [6]

$$\begin{array}{l} A_1 \leftarrow a_{21} \\ A_1 \leftarrow a_{22}, \sim a_{21} \\ A_1 \leftarrow a_{23}, \sim a_{22} \\ A_1 \leftarrow a_{24}, \sim a_{23} \end{array} \quad (1)$$

$$\begin{array}{l} A_2 \leftarrow a_{22} \\ A_2 \leftarrow a_{23}, \sim a_{22} \\ A_2 \leftarrow a_{24}, \sim a_{23} \end{array} \quad (2)$$

where a_{21},a_{22},a_{23},a_{24} are all chip condensers with different accuracy. The item in a lower place in the list has finer accuracy and higher price. a_{21} is the default for A_1, a_{22} is the default for A_2.

There must be similar rules for another part. However, we omit them in this paper.

Order

An order will be placed from the interface shown as Figure 1. It is implemented on the www system. We have a subsystem to translate the order from the www interface to a logic program. The order from Figure 1 is represented by a logic program as follows:

$$\begin{array}{rl} PC \leftarrow & SpaceSavingCase, ProcesserA3G, \\ & PC1300256MB, ATA120GB, \\ & CDRWwDVD, WirelessKBM \end{array}$$

Inventory information

About inventory information, we have a subsystem that handles numbers of stocked parts. It returns an answer to a query from the system. The answer is given like the following.

$$\begin{array}{l} a_{21} \leftarrow \\ a_{22} \leftarrow \end{array}$$

These are given only for parts in stock. Parts that are out of stock will not be shown in a list.

Shipping information

We also have a subsystem to handle shipping information. The system sends a query to the subsystem, and receives an answer. The answer is like the following.

$$\begin{array}{l} Ship\_a_{21} \leftarrow \\ Ship\_a_{22} \leftarrow \end{array}$$

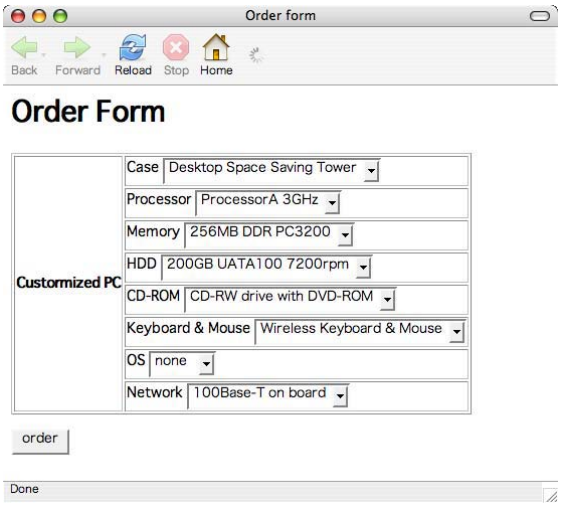

Figure 1. An interface to accept an order.

These mean that the product will reach a customer on the date promised when the customer made an order, even if the factory waits for a part a_{21} and a_{22}. If they cannot send a product on the date with waiting for a part a_{23}, the entry for a_{23} will not be generated.

When we receive an order for the normal case, that is, there are enough stock of parts to build an ordered product, the system just chooses default parts, then outputs an instruction to build a product with the default parts, reserves shipping and answers the customer the date of delivery.

On the other hand, if there are parts out of stock, it is another problem. In this case, the system shows candidates for solutions.

3. How to Visualize a Result of the System

We implement a management system to derive the solution from an order and other information. The system consists of the derivation engine and four subsystems to manage information. These engine and subsystems are implemented by Java. Its formal description is briefly described in Section 4. The system has an interface to accept an order, and has a function to show the result visually. The result will be shown as a process of reasoning, in a form of a tree.

An example is shown as in Figure 2. Nodes in the figure are goals. The final goal is the target to be built in a factory. It is shown as the root node which is placed at the top of Figure 2. It will be broken in subgoals that are conditions to make the final goal succeeded. The figure is an overview, each part of the tree can be magnified to see its detail on demand.

The system will break the final goal to subgoals, then it

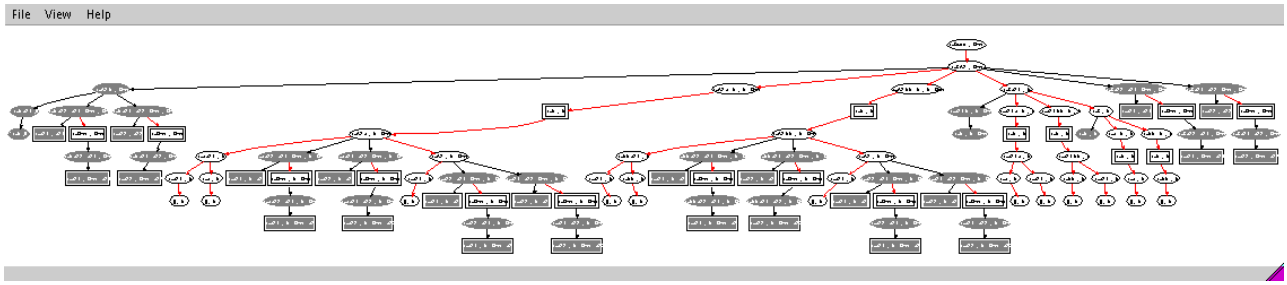

Figure 2. An overview of a result.

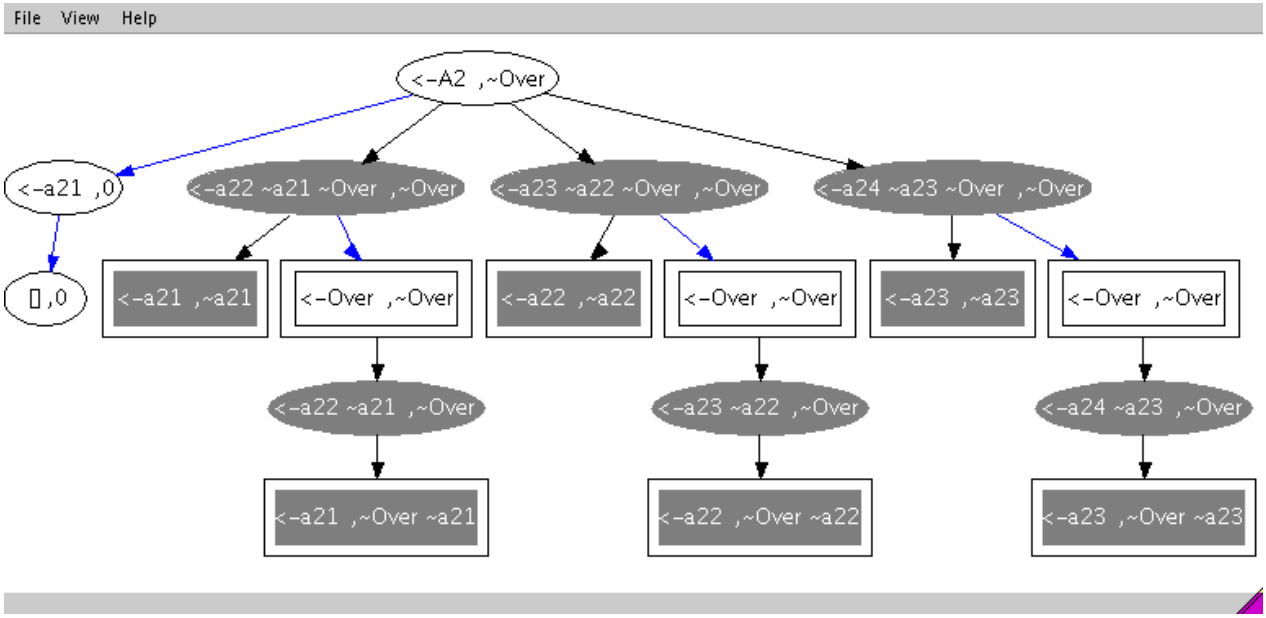

Figure 3. A case succeeded with a default part.

will reach the leaf that cannot be broken anymore. After it reached the leaf, the system will know if the subtree is succeeded or failed. The white nodes are succeeded goals. The colored nodes are failed goals. The oval node means that if the condition written on the node is succeeded, the goal of the node is also succeeded. Oppositely, the rectangle node means that if the condition written on the node is failed, the goal of the node is succeeded.

Now we see small examples in Figures 3, 4 and 5. Figure 3 shows an illustration that the goal is succeeded with a default part a_{21}. We can get the figure from the logic program (1) with A_1 as a goal. The root node is white, it indicates that the goal is succeeded, with the white child node on the left end.

Figure 4 shows that the default part is out of stock and failed with it, but the other solutions are shown. We can get the figure from the following logic program (3) with $A2$ as a goal.

$$\begin{array}{l} A2 \leftarrow a_{21} \\ A2 \leftarrow a_{22}, \sim a_{21} \\ A2 \leftarrow Ship\_a_{21}, \sim a_{21} \\ a_{22} \leftarrow \\ Ship\_a_{21} \leftarrow \end{array} \quad (3)$$

We can see the result, and know that both of the white child nodes are possible solutions. The manager can judge which one is better suited to the administrative policy and choose one of them. The system will issue an instruction with the solution the manager chooses.

Figure 5 shows that the all alternative solutions were failed. We can get the figure from the logic program (3)

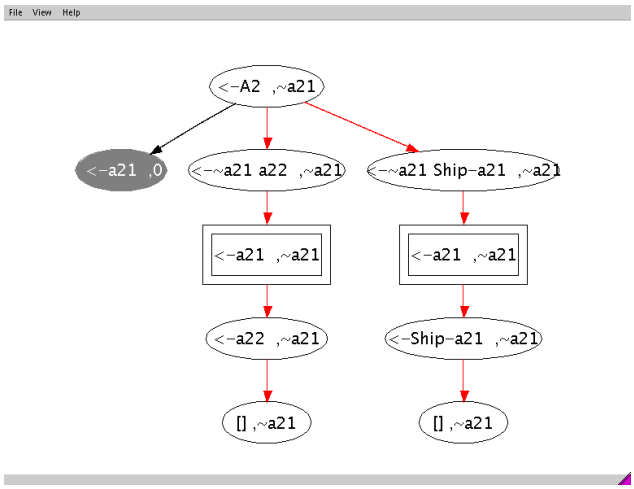

Figure 4. A case succeeded with an alternative part or rapid shipping.

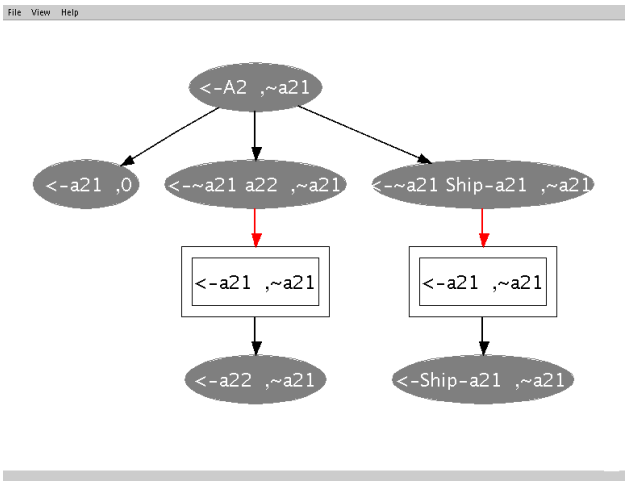

Figure 5. A case failed in all alternative solutions.

except the last two lines. The manager must postpone the shipping of the product, and ask the customer to be late to receive the product they ordered basically. But the system shows that the reason why the derivation was failed. In this case, a part a_{21} and a_{22} are both out of stock, and use a rapid shipment with waiting for a part a_{21} is also failed. Therefore the manager can know what should be fixed to solve this stuck situation. The manager can ask a part vendor if they can provide the part a_{21} or a_{22}. If it is possible, the manager can input the new condition to the system, and have a new solution after the system is derived again.

4. Design and Semantics of Management

In this section, we sketch a formal design of management in terms of semantic functions, with reference to the paper regarding the normal logic program semantics as in [14].

The management for the order of a customer is interpreted by the (semantic) functions, where:

(a) The customer ordering is captured by the function $Client$ whose income is for inventory and shipping information described as a database (by a logic program), and whose outcome is for the acquired knowledge that is a set of candidates for a solution.

(b) The management is regarded as the function $Consultant$ which gets design information described as a database (by a logic program) from an operation.

(c) The management implements the function $Acquisition$ for a query in accordance with the ordering and a database (which is a union of customer's inventory and shipping information, and management's design information).

With the notations in [14] for the normal logic program and the goal, and with the basic domains:

(i) $Database$: a set of databases (described by normal logic programs),

(ii) $Goal$: a set of goals, and

(iii) 2^S: the powerset of sets of negative literals (i.e. negated atoms),

we define the semantic functions as follows:

$$Client : Income \rightarrow Database,$$
$$Client : Outcome \rightarrow Database \rightarrow Goal \rightarrow 2^S,$$
$$Consultant : Operation \rightarrow Database,$$
$$Acquisition : Goal \rightarrow Database \rightarrow 2^S,$$

where on condition that we let

$$Client[\![Income]\!] = P_1,$$
$$Consultant[\![Operation]\!] = P_2,$$

we have:

$$\Delta \in Client[\![Outcome]\!](P_1 \cup P_2)g$$
$$\Leftrightarrow \Delta \in Acquisition[\![g]\!](P_1 \cup P_2).$$

Note that we must define the predicate:

$$\Delta \in Acquisition[\![g]\!]P$$

by means of the relation definition of [14] iff $suc_p(g; \emptyset; \Delta)$, that is, the goal g induces the set Δ (by negation as failure) for a given problem *P*.

Theorem [14]: If we have the predicate $suc_p(g; \emptyset; \Delta)$, then the reasoning is consistently sound.

As well, a management is made by the semantic function $Acquisition$. By the definition, we have:
$Proposition$:

$$\Delta \in Client[\![Outcome]\!](P_1 \cup P_2)g \Leftrightarrow suc_{P_1 \cup P_2}(g; \emptyset; \Delta),$$

where

$$P_1 = Client[\![income]\!],$$
$$P_2 = Consultant[\![Outcome]\!].$$

Note that the predicate $\Delta \in Client[\![Outcome]\!]Pg$ means the suggestive set Δ for a program P and a goal g (in relation to the query).

The set Δ is the one gained as acquired knowledge by means of the management for the customer ordering.

5. Concluding Remarks

We show a system that integrates information about production, and help managers to solve problem cases, with showing candidates for solutions and the process by which the system derives the solution.

We introduced the system we implemented as a prototype. It contains

- a derivation engine,
- www-type user interface for ordering,
- four subsystems which handle order, design, inventories and shipping information, respectively, and
- a system to visualize the result.

The result is displayed as a tree of process of reasoning. We show also that the system is useful to help managers recognizing the situation of problems.

We also show through the system that it is important to show the process to derive the solution, to help the manager judge which is the best suited solution to their administrative policy. The manager can have a key to solve the problem to which the system cannot find the solution.

References

[1] D. F. Bacon, S. L. Graham, and O. J. Sharp. Compiler transformations for high-performance computing. *ACM Computing Survey*, 26(4):345–420, 1994.

[2] G. D. Battista, P. Eades, R. Tamassia, and I. G. Tollis. *Graph Drawing*. Prentice Hall, 1999.

[3] J. Ellson, E. Gansner, L. Koutsofios, S. North, and G. Woodhull. Graphviz - open source graph drawing tools. *Graph Drawing. 9th International Symposium, GD 2001. Revised Papers (Lecture Notes in Computer Science Vol.2265)*, pages 483–484, 2002.

[4] E. Gansner and S. North. An open graph visualization system and its applications to software engineering. *Softw. - Pract. Exp.*, 30(11):1203–1233, 2000.

[5] Japan Management Association, editor. *Kanban just-in-time at Toyota:management begins at the workplace*. Productivity Press Inc., 1989.

[6] K.Kunen. Signed data dependencies in logic programs. *Journal of Logic Programming*, 7:231–245, 1989.

[7] M. Raitner. HGV: A library for hierarchies, graphs, and views. *GD2002, LNCS 2528*, pages 236–243, 2002.

[8] M. Sasakura. Concentric circle diagrams for visualizing a reasoning process on an extended logic program. *PDPTA'02*, I:253–259, 2002.

[9] M. Sasakura and S. Yamasaki. An explanation reasoning procedure applicable to loop transformation in compiler. *Proc. of ACM ESEC/FSE International Workshop on Intelligent Technologies for Software Engineering, WITSE 03*, pages 34–39, 2003.

[10] M. Sasakura and S. Yamasaki. Visualization with hierarchically structured trees for an explanation reasoning system. *Proceedings of Eighth International Conference on Information Visualization(IV04)*, pages 893–898, 2004.

[11] R. Spence. *Information visualization*. Addison-Wesley, 2001.

[12] S. Sugiyama and K. Misue. Visualization of structural information: automatic drawing of compound digraphs. *IEEE Transaction on Systems, Man and Cybernetics*, 21(4):876–892, 1991.

[13] A. Van Gelder. The alternating fixpoint of logic programs with negation. *Journal of Computer and System Science*, 47:185–221, 1993.

[14] S. Yamasaki. Semantics of normal goals as acquisitors caused by negation as failure. *IEICE Transactions on Information and Systems*, E86-D(6):993–1000, 2003.

[15] S. Yamasaki, K. Iwata, and M. Sasakura. Reasoning procedure and implementation for logic programs as managing schemes to extract demand. *IPSI Transactions on Advanced Research*, 1(1):83–90, 2005.

[16] S. Yamasaki and Y. Kurose. A sound and complete procedure for a general logic program in non-floundering derivations with respect to the 3-valued stable model semantics. *Theoretical Computer Science*, 266:489–512, 2001.

1550-6037/05 $20.00 © 2005 **IEEE**