Research Report ISIS-RR-96-10E

An Intelligent Software Package Distributing System

Takanori Ugai and Hisayuki Horai

June, 1996

Institute for Social Information Science (*ISIS*)
at Makuhari

FUJITSU LABORATORIES LTD.

1-9-3 Nakase Mihama-ku Chiba 261, Japan
Telephone: (Makuhari) +81-43-299-3211 Fax: +81-43-299-3075

An Intelligent Software Package Distributing System

Takanori Ugai and Hisayuki Horai

Institute for Social Information Science (*ISIS*)
at Makuhari

FUJITSU LABORATORIES LTD.

1-9-3 Nakase Mihama-ku Chiba 261, Japan
Email: {ugai,horai}@iias.flab.fujitsu.co.jp

Abstract

In this paper we propose a software package distributing system, which receives requirements from both developers and customers. The system provides services to customers whether they connect with developer through a network or not. Furthermore, the system controls software package distribution in according to customers' requirements and optimizes the distribution. Our flexible system also allows for future extendibility.

Keywords: software distribution, requirement analysis

1 Introduction

Software maintenance and upgrade are becoming important stages in the software life cycle. Public and private networks allow software developers to maintain software packages at the remote site and provide new distributions directly to the target system. Customers often complain about how difficult to set up software and the potential conflict between software updates and patches often confuses them. Whereas, developers would like to determine maintenance consists for the software. There are currently some software distribution systems available, e.g. mirror [1], rdist on UNIX, and SDM [2]. These systems accept requirements only from developer or users. Version control systems for managing modules of large software systems have also been proposed [3].

In this paper we propose a framework for a software distribution system. This system manages software packages over a network, and accepts requirements from both developers and users. We also show an implementation of the software distribution system based on the framework.

A system which accepts only requirements from customers does not allow developers to manage the revision of the software. Similarly, a system which accepts requirements only from the developer will not satisfy the customers' requirements, e.g., the user does not require some software functions. We have to consider that all computers will not necessarily be connected to networks in the future.

Our software distribution framework has following characteristics.

- The system complies to the request of both the customer and the developer.
- The system supports distribution to customers who are not connected to a network (off-line user) as well as distribution to connected users (online user).
- Setting of the distribution frequency according to the users' specifications, the dependency between software packages, and the amount of software to be distributed.

In this paper, the composition of the entire system is described in Chapters 2 and 3. The database for user information management and the agent which distributes the software are shown in Chapter 4.

2 Framework for software distribution systems

Our system consists a the software packages database, a software distribution agent (called "agent"), and of course the customers, as shown in the Figure 1.

The software packages database contains an inference engine, to infer the dependency between software packages. The database controls the software group as a unit with the package and the customer's information on the requirements and functional specifications for the packages. The database sends the information and specification s to the agent as instructions for the software distribution.

The agent then distributes the software packages and updates information according to the information and specifications in the software packages database. The agent is a software system for an online user and the "Shop", represents the distribution channel for an off-line user. The user specifies the distributed packages interactively to the software packages database or the agent, and the specifies the method of distribution for the update. The user can specify the received distribution of packages according to their environment. For instance, if the user has enough hard disk capacity, they could receive the package which contains extended functions, and user without extra capacity could receive the minimum required distribution. An automatic update can be performed for online connected users, and off-line users can receive a guide for the update of the packages. Even when users are connected by a network, the frequency of the update can be limited according to the capacity of the network line. The system allows users to select the method according to their capacity using the inference engine. The developer wants to avoid maintaining many different software packages, even when they are only different versions of

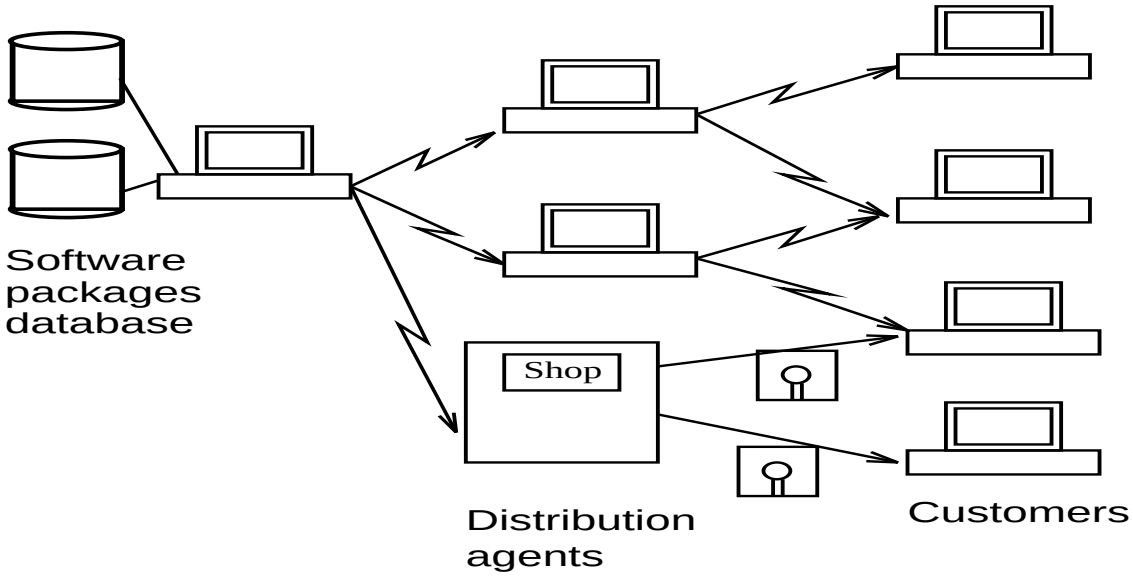

Figure 1: Structure of software distribution systems.

the same program. Therefore, developers should promote users to shift to the new system. This system promotes a prompt shift by an automatic update with the agent.

3 Software packages database

As shown in Figure 2, this system accumulates software packages in a database with each package stored as an element. Two or more candidates of the achievement of the distribution requirements of a certain package should be considered because of a complex dependence relation between the different version packages. This system chooses the best distribution scheme based on the user's system and the specified requirements of the user.

The database contains information on each user, it includes:

- Licence of each package
- Various settings of each package
- Hard disk capacity
- Location of existing packages

Generally when the user's system is connected by a network, the information about the user's system is automatically acquired from the control software, and is stored at the user's site — though it is also possible to obtain the information interactively. The basic method for each package distribution can be calculated by using the acquired information. Specifications such as "Replace the relevant package with the latest one", " Keep costs to a minimum", and "Set available disk capacity," are set interactively in the database by the user.

The following settings can be specified and managed.

- Dependency relationship between packages

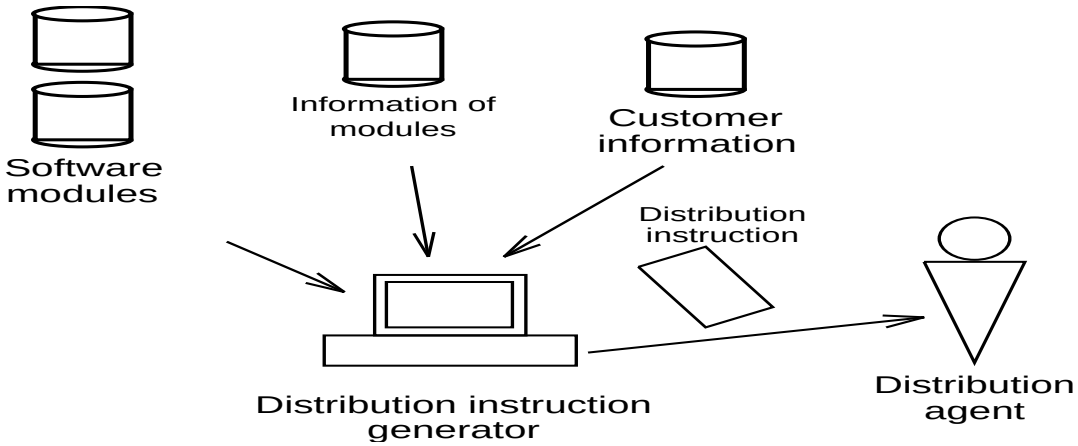

Figure 2: Software packages database

- Available disk capacity
- Price
- Function specification

3.1 Dependency between packages

The package has information on the version and the required capacity of the disk. Moreover, the operational dependency with other packages is also indispensable because certain packages cannot coexist. The user inputs this information using expanded AND-OR graphs.

Definition 1 *A dependency is a set of Expression*

Definition 2 *syntax of Expression*

Expression = *Module_name* :- *Module_name AND Expression*
| *Module_name* :- *Module_name OR Expression*
| *Module_name* *NOT* *Module_name*

Module_name = *Alpha*
| *Alpha (X)*
| *Alpha (>X)*
| *Alpha (<X)*

where "Alpha" is a package name and "X" is a number.

Intuitionally α(X) stands that the module name of version X of the package α. α(>X) and α(<X) represent module names and each module name shows newer one and old or more than version X of the package α respectively.

Definition 3 *a module is a version of a software package.*

The system manages package information and the dependency relationships between the packages by converting specifications into the form of the this AND-OR graph and by using the database.

One example of the dependence relations is given as follows.

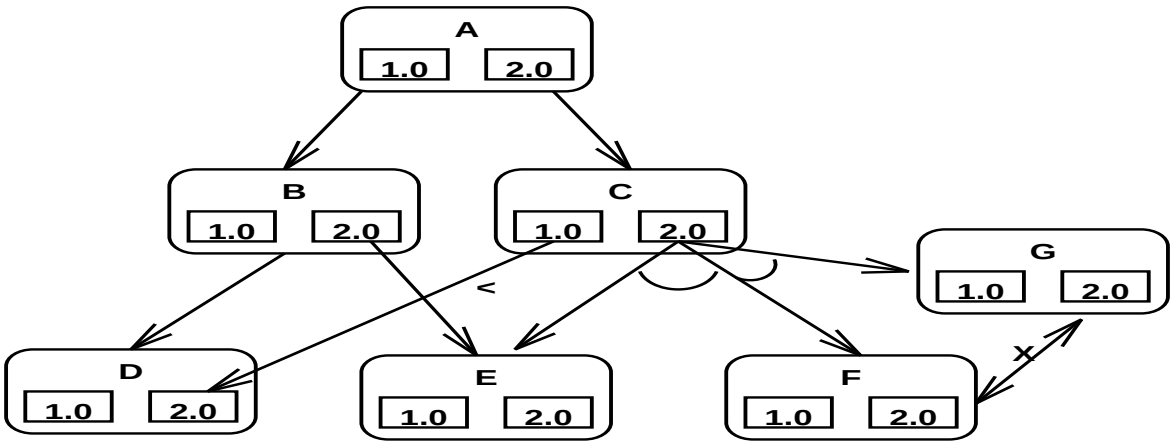

Figure 3: Dependency among the packages.

A :- B AND C
B :- D
B(2.0) :- E
C(1.0) :- D(>2.0)
C(2.0) :- E OR F OR G
G NOT F

Figure 3 shows information related to the dependence between packages In this example, all packages are either version 1.0 or 2.0. Package A needs package B and package C. Package B needs package D and version 2.0 of package B needs package E. Version 2.0 of package C does not need package D, instead it needs package E or F or G However, package F and G cannot be used at the same time.

3.2 Distribution instruction inference engine

Figure 4 shows the outline of the distribution instruction inference engine. Two or more candidates of the instruction schema for the software package distribution are enumerated because of a complex dependency relationship between the packages and the different versions.

Definition 4 *Notation of a instruction*
{*A1.0, B1.0, C1.0*} *is a instruction. The system distribute version 1.0 of package A, version 1.0 of package B, and version 1.0 of package C.*

For instance, the candidate of distribution instruction of package A are 74 in all such as { A1.0, B1.0, C1.0, D2.0 }, { A1.0, B2.0, C1.0, D2.0, E1.0 }, and { A2.0, B2.0, C2.0, D1.0, E2.0, F2.0 } in Figure 3. This system first extracts the candidates of all schema based on the dependency between packages, and chooses the best distribution method based on the user's system and the requirements of the users. A set of the candidate schema are constructed by tracing the requirements (shown by arrows in Figure 3) related to the dependency packages. The dependency is as follows.

- Prohibition of using together of same module in a instruction.
- Prohibition (by bidirectioned arrow with a X shown in Figure 3) of using two modules, together.
- Dependency between packages.

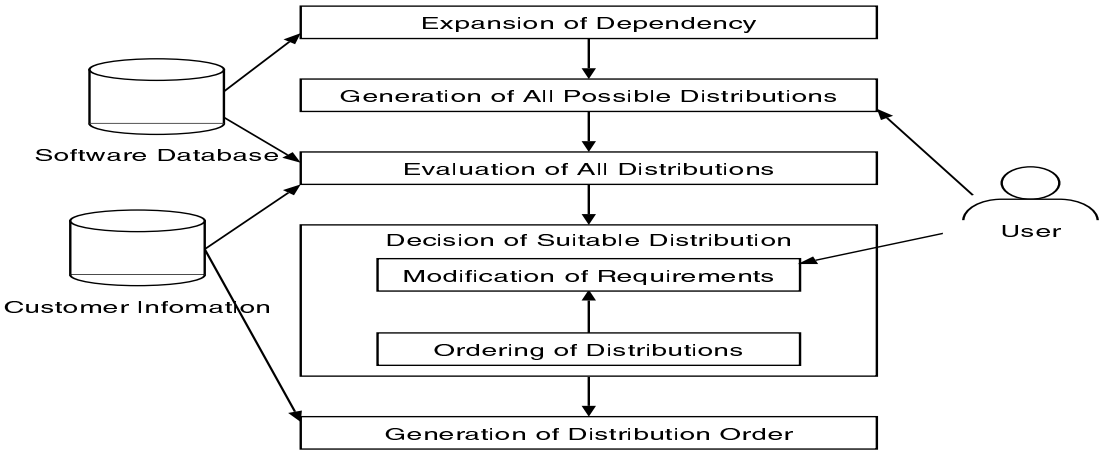

Figure 4: Distribution Instruction Inference Engine

Table 1: Indirect arranging method

Specification Item	Indirect method
Version	It should be newer
Licence fee	Expense should be minimized
Disk Capacity	The disk space should be minimized

- Dependency between package and module.

The system develops from the above-mentioned relationship to the dependency between modules and develops with two or more necessities concerning a pertinent module of the necessity that the version attribute in addition (an arrow with a “>” mark in Figure 3) at any time. We can prevent dependency relationship looping, by closing the relationship.

It is necessary to express the user requirements by a pair showing that "A certain candidate is better than another candidate" and arrange the order among the requirements for the selection of the distribution method. The direct specification method of, "Candidate X is better than candidate Y" and the indirect specification method of using attributes such as, "It should be new" and "Expense should be minimized" are provided for specifying the candidates. This system not only uses direct specifications as it is for paired relationship but also expands indirect specifications. In this expansion, the system use the capacity of the disk and version number and licence fee of each module defined in the database as well as the information on the user’s system. For instance, the specification "Expense should be minimized" is expanded by calculating the expense of all candidates and sorting them in order of expense. The user can also specify requirements for each candidate, such as "A candidate alone who does not exceed the capacity of an existing disk". In the inference after this, the candidate that contradicts this requirement is discarded. If the user specifies too severe a limitation and no candidate can calculates, the system informs the user to lelax the requirements. The available requirements for the candidate limitation and the indirect specification methods are shown in Table 1 and 2.

In general, a user requirements often contradict each other. For instance, when a user require that "It should be new" and "The capacity of the disk should be minimized", a relationship converted from these requirements might have contradictions. In this case the newer version may take up more disk space

Table 2: Candidate limitation.

Specification Item	Candidate limitation
Version	An old version is not used
Licence fee	Do not exceed a specific amount of money
Capacity of the disk	Do not exceed existing disk capacity

than the older version.

When the transition closure of relationships is converted from the user's requirements.

Definition 5 *if $X > Y$ and $Y > X$ are true, the relationship between candidates concerning X and Y is defined as a contradiction.*

Definition 6 *if both of X>Y and Y>X are false, there is no Z that satisfies $Z > X$, and there is no W that satisfies $W > Y$, the relationship between candidates concerning X and Y is defined as the incomplete.*

This system allows th user remove contradicting relationships. Our system shows the set elements where there is a contradiction so that the user can approve or cancel individual relationships. If cancellation is selected, the relationship is removed. If approve is selected, the relationship that contradicts the approved specification is removed. An improper relationship can be indirectly removed by specifying priority levels between the user's requirements without directly selecting the relationship. For instance, if contradiction between relationships is caused and when the priority of "It should be new" is higher than "The capacity of the disk should be minimized", latter requirement is automatically removed. If the user's requirements do not narrow down the candidates, the system presents the candidates without order and lets the user input a direct or indirect requirement.

When the best distributed package is decided, the instruction for the distribution is made. This instruction can be composed of "License purchase", "Installation of the package", "Check the capacity of the disk", "Move the module location", and "Make aliases of module." An instruction is decided based on the best distribution method and the user's situation. If the user does not have the license of the module in the distribution method, "License purchase" is ordered to the module. If the module does not exist on user's disk, " Installation of the package" is ordered. In this case, if the available area on user's disk is insufficient, "Check the capacity of the disk" is ordered. If the module has already been installed on the user's disk and the module location is different from the requirements, "Make aliases of the module" or "Move of the location" is ordered. A series of instructions is passed by the software distribution agent and the distribution will be executed.

4 Software Distribution Agent

In the framework proposed by this paper, there are two kinds of system agents, an off-line agent and an online agent. The off-line agent include people who distribute package to people who are not online. The online agent supports distribution over a network. The online and off-line agents decide the distributed packages interactively with the user, and the software package is distributed to the user by the instructions from the software package database. The mechanism is different when instructions are done automatically by computers or manually by the users. The software distribution agent who automatically does all the distribution is called an online agent.

Online agents and off-line agents are basically governed by the following protocols.

1. Present version information on the managed package is received from the database or the user.
2. The requirements of the user are obtained.
3. The distributed packages are decided by using obtained information.
4. The packages are distributed.
5. The user side system updates the softwares in the distributed packages.

Information on the user's system can be automatically acquired and an automatic update from the development side is possible only for an online user. Therefore, when the system supports off-line users, it is necessary to notify the users of updates.

4.1 Off-line agent

An off-line agent is a software system which decides the distributed software package interactively to the user. The required software packages are stored on a distribution media such a floppy disk, along with an automatic installer, determined by the user's requirements in the software package database. For instance, notification of the update can be sent to the user by FAX from the shop using this system according to the instruction of the software package database. Then the user buys the software at the shop using this system and installs it at their computer system site.

5 Future work

In the software distribution system we have proposed, the method of collecting money for the software support is not considered. In future work we would like to model many possible contract methods and enable control with the distribution instruction inference engine. It is also necessary to develop the distribution software itself. Moreover, a consistent and user-friendly interactive user interface for specification of user and developer requirements is also necessary.

6 Conclusion

In this paper, we presented a system to manage software packages as a unit. We proposed a framework for a package distribution system that is controlled by update requests from both the user's side and the development side.

Acknowledgements

The authors have benefited from the discussions with the members in ISIS.

References

[1] Lee McLoughlin. *Manual of mirror, mirror packages on remote site*.

[2] Toru Nakagawa and Yuji Takada. A System for Softwere Maintenance through a Network. Research Memorandum ISIS-RM-95-1E, Institute for Social Information Science, Fujitsu Laboratories LTD, 1995.

[3] Walter F. Tichy. Tools for Software Configuration Management. In Winkler, J.F.H., editor, *Proceedings of the International Workshop on Software Version and Configuration Control*, pp. 1–20. ACM, B.G. Teubner, Stuttgart, West Germany, Jan 1988.