Reasoning Procedure and Implementation for Logic Programs as Managing Schemes to Extract Demand

Susumu Yamasaki, Kenichi Iwata, and Mariko Sasakura

Abstract— ***We apply a nonmonotonic reasoning based on logic programming to the managing scheme to get information on some composition of assumed materials and demanded ones. This scheme can be applied to the infrastructure on the internet by extracting the demand for supply. To extract the demand for a composition, the well-renowned negation as failure rule is available in logic programming. Explicit negation may be included for expressive capabilities, for which some contradiction-removal procedure must be fairly implemented, to avoid the contradiction of a predicate competing with its explicit negation.***

Index Terms—***Logic Programming, Managing, Nonmonotonic Reasoning***

1. INTRODUCTION

THERE has been a growth of order-made PC compositions to make use of modules to be combined for the PC customer needs. The PC composition should get modules (materials) from the more original producer and make variations of PCs for the customer (consumer). The composition process is regarded as a methodology, or a management in material compositions for making PCs. The composition induces a managing scheme, being an interface between the producer and the consumer.

For the interface, reasoning is of much use as [3] presents in wide variations of areas. In this paper, a managing scheme with reasoning facilities is examined as regards the PC composition domain. As an automated interface between the producer and consumer, a managing scheme is of use, where the interface is required to reason the process of getting materials from the producer, and composing the materials to supply goods to the consumer. Through the composition, some extraction of demanded materials is to be made by reasoning so that the demand is passed to the producer, and some possible supply is made. The managing of compositions can be organized with the supply goal and inferred demand of materials, which is illustrated in Fig. 1.

Manuscript received November 5, 2004.

Authors are with Department of Intelligence Computing and System, Graduate School of Natural Science and Technology, Okayama University, 3-1-1 Tsushima-Naka, Okayama, Japan (e-mail: yamasaki@momo.it.okayama-u.ac.jp).

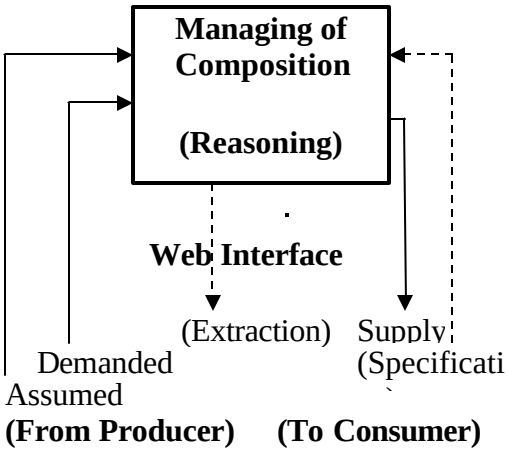

Figure 4 A managing scheme and Web-interface.

The scheme is constructed as a function or a procedure with some reasoning capabilities. For the extraction of demand to be made reasonably, logic programming is effective, because the extraction of demand can be fairly implemented by explanation reasonings, which originated from [5],[8]. As [6] presents the basis, a logical system (a class of extended logic programs) containing two kinds of negation, explicit negation and negation as failure, is regarded as one of standard frameworks with both simplicity and expressive capability. Based on the class, the abduction framework in [4] might offer a good model of managing scheme. With the role of negation as failure to extract information, this paper rather deals with a contradiction-free procedure for an extended logic program with some model theory. For simplicity, we adopt only negation as failure to extraction, apart from the treatment (as in [4]) of both positive abducibles and negative defaults.

To cope with a possible contradiction involved in a managing scheme, we make use of even contradiction-removal techniques as regards explicit negation for expressive facilities. There have been studies on proof procedures for extended logic programs (containing both explicit negation and negation as failure), as in [1],[12],[13], with an introduction of the coherence principle that the explicit negation implies the negation as failure. In accordance with the exception notion of [10] for removing any contradiction between a pair of a predicate and its explicit negation, a proof procedure is constructed in [18]. It is adopted for implementation. In this paper, a more abstract representation of the proof procedure is given for the reasoning capability. The realization of a managing scheme by means of logic programming is shown in Table 1.

To realize a managing scheme, we deal with logic programming as follows:

(i) Both explicit negation and negation as failure are treated.

(ii) A proof procedure is presented, based on negation as failure to extract the demand of subjects. The proof procedure is concerned with dynamically removing the contradiction (through a procedure execution) between a pair of a predicate and its explicit negation. The contradiction removal is required, because the scheme may involve some contradiction.

(iii) An abstract representation of the proof procedure is given for its demonstration.

Table 1 Realization of Managing Scheme by Logic Programming

A managing Scheme	An Extended Logic Program
Reasoning Facilities for Composition and Extraction	Proof Procedure for Nonmonotonic reasoning with Negation as Failure

2. LOGICAL EXPRESSIONS

D. Notations

We now present the terminology for propositional logic programming.

(1) A set of symbols to stand for propositions is assumed.

(2) Two kinds of negation signs are taken: the explicit negation "$\neg$" and the negation as failure "not".

(3) A literal l is either an atom a or an explicit negation $\neg a$ of an atom a. An atom is an expression consisting of a symbol to denote a proposition.

We deal with the extended logic program. An extended logic program (ELP, for short) is a set of clauses of the form

$$l \leftarrow l_1,...,l_m, notl_{m+1},...,notl_n \quad (0 \le m \le n)$$

where l and l_i are literals, and "not" stands for the negation as failure (NAF). The expression $notl$ is also called an NAF literal. The literal l of the clause is said to be its head, and the literal sequence $l_1,...,l_m, not\ l_{m+1},...,notl_n$ is its body. The explicit negation $\neg l$ means

(i) $\neg a$ if $a = l$, and

(ii) a if $l = \neg a$,

for an atom a. The pair of literals a and $\neg a$ is said to be complementary.

The expressions $L, L_1, ..., L_m, ...$, $M, M_1, ..., M_n, ...$ are reserved to denote literals or NAF literals. The expressions $a, a_1, ..., a_m, ..., b, b_1, ..., b_n, ...$ are reserved to denote sequences of literals or NAF literals.

The ELP is said to be a general logic program if it does not contain any explicit negation. For details of ELPs, see [1],[2],[4],[10],[12],[13]. As a means of treating consistency for a pair of a literal and its explicit negation, an exception notion is introduced in [10]. In [1], a proof procedure with coherence principle is given, while some condition for the program to be consistent is shown. In [4], the abduction with the integrity constraint is formulated.

A goal is an expression of the form

$$\leftarrow l_1,...,l_m, not\ l_{m+1},...,not\ l_n \quad (0 \le m \le n),$$

where l_i are literals. The goal of the form $\leftarrow not\ m_1,...,not\ m_q$ $(q \ge 0)$ is said to be a negative goal. The negative goal is the empty clause, denoted by $empty$, if it contains no literal. Note that the goal $\leftarrow l$ may be interpreted as a proposition $\neg l$. We briefly mention SLD resolution and negation as failure (see [11],[15] for details), in which the reasoning capability consists of SLD resolution and negation as failure in logic programming.

The SLD resolution (deduction) is a derivation formed by finitely many times applying a rule to infer a goal $\leftarrow L_1,...,L_{i-1}, M_1,..., M_n, L_{i+1},..., L_m$ from a goal $\leftarrow L_1,...,L_{i-1}, l, L_{i+1},..., L_m$ and a clause $l \leftarrow M_1,...,M_n$ in the given ELP. That is, a literal l of a goal may be replaced by the body $M_1,...,M_n$ of a clause $l \leftarrow M_1,...,M_n$, whose head is just the literal l. Negation as failure is a rule to infer an NAF literal $notl$, when the literal l is not derived by some proof procedure. We next refine negation as failure in relation to SLD resolution such that the succeeding and failing (derivations) of a goal (with reference to a given ELP) are defined recursively as follows:

(i) The goal $\leftarrow l$ succeeds if $\leftarrow l$ is reduced to $empty$ by applying SLD resolution.

(ii) The goal $\leftarrow l$ fails if one of the following conditions holds:

(a) There is no clause whose head is the literal l.

(b) The goals $\leftarrow a_1, L_1, b_1, ...,$ $\leftarrow a_n, L_n, b_n$ $(n \ge 1)$ are derived by SLD resolution for the goal $\leftarrow l$ such that all the goals $\leftarrow L_1, ...,$ $\leftarrow L_n$ fail.

(c) The goal $\leftarrow \neg l$ succeeds.

(iii) The goal $\leftarrow notl$ succeeds if the goal $\leftarrow l$ fails.

(iv) The goal $\leftarrow notl$ fails if the goal $\leftarrow l$ succeeds.

Note the sense that if the goal $\leftarrow l$ may succeed, then $\neg l$ may be concluded.

In relation to the definition (ii)(c) above, we will present a method of removing contradictory succeeding derivations of both the goal $\leftarrow a$ and the goal $\leftarrow \neg a$. If both goals may succeed, contradictory conclusions $\neg a$ and a (that is, $\neg\neg a$) are regarded as inferred.

E. Illustration of a Program ELP

We present a simple example as regards managing a PC composition for the conditions.

(1) The PC consists of modules Processor, HDD, and CD-ROM.
(2) The module Processor is a Standard-type of 512 MB, or no Standard-type of 256MB.
(3) A processor ProcessorA is a Standard-type.
(4) No processor ProcessorB is not a Standard-type.
(5) The module HDD contains a disk of 80 GB with no case Overcost, or a disk of 40 GB.
(6) If there is no disk of 40 GB, then it is in the case Overcost.
(7) The module CD-ROM has a driver DriveA but not a driver DriveB, or is in no case of not containing a driver DriveB.

The conditions are expressed by an ELP :

(a) $PC \leftarrow Proc, HDD, CD\text{-}ROM$
(b) $Proc \leftarrow M512, Standard$
(c) $Proc \leftarrow M256, not\ Standard$
(d) $Standard \leftarrow ProcA$
(e) $\neg Standard \leftarrow not\ ProcB$
(f) $HDD \leftarrow HD80, Overcost$
(g) $HDD \leftarrow HD40$
(h) $Overcost \leftarrow not\ HD40$
(i) $CD\text{-}ROM \leftarrow DriveA, \neg DriveB$
(j) $CD\text{-}ROM \leftarrow not\ \neg DriveB$

If there is a processors *ProcA* and no processor *ProcB*, that is, there is a clause $ProcA \leftarrow$ and a clause $\neg ProcB \leftarrow$, there is a contradiction by means of the clauses (d) and (e).

The cases with hard disks of 80 GB and 40 GB are regarded as in "exclusive-or" for the module HDD, that is, one of two clauses $HD80 \leftarrow$ and $HD40 \leftarrow$ is exclusively selected for the clause $HDD \leftarrow$ to be reasoned.

The cases with drivers *DriveA* and *DriveB* are in "exclusive or". Although the clauses (f), (g) and (h) take a different combination with a pair of clauses (i) and (j), we see from the pair clauses (i) and (j) that one of two clauses $DriveA \leftarrow$, and $DriveB \leftarrow$ is exclusively selected for the clause $CD\text{-}ROM \leftarrow$ to be reasoned.

3. FORMAL REPRESENTATION OF REASONING PROCEDURE

The managing scheme can be expressed by an ELP with reasoning facilities. The reasonings are presented in [18]. For a brief demonstration, a relational form of the ground version procedure is herein given. We have two sets to be kept, which are to be transformed through succeeding and failing derivations:

(i) the set of literals to remove contradictory succeeding derivations
(ii) the set of NAF literals

Note that the first set is in accordance with a semantic view of [18], and the second is primarily a means of information extraction.

The former set is expressed by $\Sigma, \Sigma_1, ...$, and the second is by $\Delta, \Delta_1, ...$ Given an ELP P, the predicate $suc_p(G; \Sigma_1; \Delta_1; \Sigma_2; \Delta_2)$ is true, when a goal G succeeds with the assumed sets Σ_1 and Δ_1 to acquire the sets Σ_2 and Δ_2. The predicate $fail_p(G; \Sigma_1; \Delta_1; \Sigma_2; \Delta_2)$ is true when a goal G fails with the assumed sets Σ_1 and Δ_1 to acquire the sets Σ_2 and Δ_2. The set $\Delta_2 - \Delta_1$ is interpreted as extracted. Following the derivations in [18], we have these rules:

(0) The goal $empty$ trivially succeeds.
(1) SLD resolution is applicable to the succeeding derivation.
(2) The goal $\leftarrow not\ a$ succeeds if the NAF literal $not\ a$ is in the assumed set Δ_1.
(3) The goal $\leftarrow not\ a$ succeeds if $not\ a \notin \Delta_1$ and the goal $\leftarrow a$ fails with the assumed sets Σ_1 and $\Delta_1 \cup \{not\ a\}$.
(4) The goal $\leftarrow a$ fails when there is no clause for SLD resolution to apply to.
(5) SLD resolution is applied for the goal to exhaust the failing (sub-)derivations.
(6) The goal $\leftarrow \neg a$ fails if the goal $\leftarrow a$ succeeds.
(7) The NAF literal $not\ a$ is removed from a goal in the failing derivation when the

NAF literal $not\ a$ is included in the assumed set Δ_1.

(8) The goal $\leftarrow not\ a$ fails when $not\ a \notin \Delta_1$ and the goal $\leftarrow a$ succeeds with the assumed set Δ_1.

We assume an ELP P and we now have the relational representations of succeeding and failing derivations, in which the relations suc_p and $fail_p$ are defined simultaneously by recursion to be the least set satisfying the following closure. The relational representation of derivations for general logic programs is given in 0. The relations demonstrate the implementation as procedural methods for the given ELP. The subscript P for the ELP P may be omitted if it is clear in the context.

(0) $suc_p(empty;\Sigma;\Delta;\Sigma;\Delta)$ for any Σ and Δ.

(1) $suc_p(\leftarrow L_1,...,L_{i-1},M_1,...,M_m,L_{i+1},...,L_n;\Sigma_1\cup\{l\};\Delta_1;\Sigma_2;\Delta_2)$ for $\neg l\notin\Sigma_1$ and $(l\leftarrow M_1,...,M_m)\in P\Rightarrow$
$suc_p(\leftarrow L_1,...,L_{i-1},l,L_{i+1},...,L_n;\Sigma_1;\Delta_1;\Sigma_2;\Delta_2)$.

(2) $suc_p(\leftarrow L_1,...,L_{i-1},L_{i+1},...,L_n;\Sigma_1;\Delta_1;\Sigma_2;\Delta_2)$ and $not\ l\in\Delta_1\Rightarrow$
$suc_p(\leftarrow L_1,...,L_{i-1},notl,L_{i+1},...,L_n;\Sigma_1;\Delta_1;\Sigma_2;\Delta_2)$.

(3) $fail_p(\leftarrow l;\Sigma_1;\Delta_1\cup\{notl\};\Sigma_2';\Delta_2')$ for $not\ l\notin\Delta_1$, and
$suc_p(\leftarrow L_1,...,L_{i-1},L_{i+1},...,L_n;\Sigma_2';\Delta_2';\Sigma_2;\Delta_2)\Rightarrow$
$suc_p(\leftarrow L_1,...,L_{i-1},notl,L_{i+1},...,L_n;\Sigma_1;\Delta_1;\Sigma_2;\Delta_2)$.

(4) There is no clause in P, which contains the literal l in the head $\Rightarrow$
$fail_p(\leftarrow L_1,...,L_{i-1},l,L_{i+1},...,L_n;\Sigma;\Delta;\Sigma;\Delta)$ for any Σ and Δ.

(5) For any clause $l\leftarrow M_1^j,...,M_{n_j}^j$ in P $(1\le j\le k)$ of all the clauses that contain the literal l in the head,
$fail_p(\leftarrow L_1,...,L_{i-1},M_1^j,...,M_{n_j}^j,L_{i+1},...,L_n;\Sigma_j;\Delta_j;\Sigma_{j+1};\Delta_{j+1})\Rightarrow$
$fail_p(\leftarrow L_1,...,L_{i-1},l,L_{i+1},...,L_n;\Sigma_1;\Delta_1;\Sigma_{k+1};\Delta_{k+1})$.

(6) $suc_p(\leftarrow\neg l;\Sigma_1;\Delta_1;\Sigma_2;\Delta_2)\Rightarrow$
$fail_p(\leftarrow L_1,...,L_{i-1},l,L_{i+1},...,L_n;\Sigma_1;\Delta_1;\Sigma_2;\Delta_2)$.

(7) $fail_p(\leftarrow L_1,...,L_{i-1},L_{i+1},...,L_n;\Sigma_1;\Delta_1;\Sigma_2;\Delta_2)$ and $not\ l\in\Delta_1\Rightarrow$
$fail_p(\leftarrow L_1,...,L_{i-1},not\ l,L_{i+1},...,L_n;\Sigma_1;\Delta_1;\Sigma_2;\Delta_2)$.

(8) $suc_p(\leftarrow l;\Sigma_1;\Delta_1;\Sigma_2;\Delta_2)$ for $not\ l\notin\Delta_1\Rightarrow$
$fail_p(\leftarrow L_1,...,L_{i-1},not\ l,L_{i+1},...,L_n;\Sigma_1;\Delta_1;\Sigma_2;\Delta_2)$.

We present the soundness of the reasoning procedure for the managing scheme. The extraction of demand may be correctly implemented because of the procedural soundness. The paper [18] suggests a semantics which supports a contradiction-free procedure.

In the following sense, the relation suc_p, which may involve the effect of the relation $fail_p$, is sound. This is paraphrased to the sense of consistency that if a goal $\leftarrow l$ succeeds, then $notl$ cannot be included in the set of NAF literals.

Definition 1. For a set Σ, we define the set $\Gamma[\Sigma]$ to be $\{not\ \neg l \mid l\in\Sigma\}$.

Note in the following theorem that we regard the set $\Delta_2\cup\Gamma[\Sigma_2]$ as extracted.

Theorem 1. Assume the predicate $suc_p(\leftarrow l;\Sigma_1;\Delta_1;\Sigma_2;\Delta_2)$ such that $\Sigma_1=\Delta_1=\varnothing$. Then $not\ l\notin\Delta_2\cup\Gamma[\Sigma_2]$.

Proof (Outline) (1) By the definition of the relation suc_p, $\neg l\notin\Sigma_2$. It follows that $not\ l\notin\Gamma[\Sigma_2]$.

(2) Contrary to the assumption that

$not\ l \notin \Delta_2$, suppose that $not\ l \in \Delta_2$. Then, by the construction of Δ_2, $fail_p(\leftarrow l;\Sigma_1';\Delta_1';\Sigma_2';\Delta_2')$ for $\Sigma_1', \Delta_1', \Sigma_2', \Delta_2'$ such that $l \in \Delta_1' \subseteq \Delta_2' \subseteq \Delta_2$. We next examine the cases supporting this relation $fail_p$.

(i) When $suc_p(\leftarrow \neg l;\Sigma_1';\Delta_1';\Sigma_2';\Delta_2')$, this is a contradiction to the first assumption that $suc_p(\leftarrow \neg l;\Sigma_1;\Delta_1;\Sigma_2;\Delta_2)$.

(ii) When there is no clause whose head is l for the relation $fail_p$, this is a contradiction to the first assumption that $suc_p(\leftarrow \neg l;\Sigma_1;\Delta_1;\Sigma_2;\Delta_2)$ such that there is some clause whose head is l.

(iii) When there is the predicate $fail_p(\leftarrow not\ m_1,...,not\ m_n;\Sigma_1'';\Delta_1'';\Sigma_2'';\Delta_2'')$

for some sets $\Sigma_1'', \Delta_1'', \Sigma_2'', \Delta_2''$, which is possibly caused by $\leftarrow not\ m_1,...,not\ m_n$ derivable from the goal $\leftarrow l$, it is concluded that $n \neq 0$. Otherwise, $\leftarrow not\ m_1,...,not\ m_n$ is *empty* and it contradicts $fail_p$. For $\leftarrow not\ m_1,...,not\ m_n$, there is some literal $m_i \notin \Delta_1$ $(1 \le i \le n)$ such that we have $suc_p(\leftarrow m_i;\Sigma_1'';\Delta_1'';\Sigma_2'';\Delta_2'')$, because of $fail_p$. On the assumption that $not\ m_i \in \Delta_2$, we repeat the same discussion. We finally reach the case in which $fail_p(\leftarrow not\ n_1,...,not\ n_k;\Sigma_{f_1};\Delta_{f_1};\Sigma_{f_2};\Delta_{f_2})$

for some sets $\Sigma_{f1}, \Delta_{f_1}, \Sigma_{f_2}, \Delta_{f_2}$, but there is no literal $not\ n_i$ such that $not\ n_i \notin \Delta_{f_1}$. This causes the case in which $\leftarrow not\ n_1,...,not\ n_k = empty$, which is a contradiction to the possible predicate that

$$fail_p(\leftarrow not\ n_1,...,not\ n_k;\Sigma_{f_1};\Delta_{f_1};\Sigma_{f_2};\Delta_{f_2})$$

.

q.e.d.

4. MANAGING SCHEME WITH WEB-INTERFACE

A. Architecture

The architecture of the presented managing scheme with Web-interface is shown as in Fig. 2. That is, a managing scheme is

$$M = (Reasoning\ system, Database, Web)$$

where:

(1) The subsystem Reasoning system operates on the program stored in Database and a goal given by a manager so that the reasoning result and the extracted information are provided to the manager. The demand may be communicated to the producer. It is constructed as a Java applet, following the procedure of [14],[18] in accordance with the relational representation of this paper.

(2) The subsystem Database contains an order list of the consumer (customer), a stock list of the producer and the parts information of the manager. Database takes a form of ELP. It is described by XML forms.

(3) Web-interfaces Web are installed for the manager, the consumer and the producer. They are constructed as Java servlets.

B. Implementation illustration

We have an experiment for the ELP which is presented in Section 2. B. Assume that:

(i) The consumer requests the statement (1) which is translated to the clause (a) in Database.

(ii) The manager keeps the statements (2)-(7), which are directly translated to the clauses (b)-(j), respectively, in Database.

(iii) The producer presents the parts *M512*, *HD80*, *DriveA*, but not *DriveB*. They are translated to the clauses :

$$M256 \leftarrow;\ HD40 \leftarrow;\ DriveA \leftarrow;$$

$$\neg DriveB \leftarrow .$$

The procedure is implemented as follows.

(i) The goal $\leftarrow PC$ is given by the manager to evoke the reasoning procedure.

(ii) The next goal is easily obtained: $\leftarrow Proc, HDD, CD\text{-}ROM$. The subgoal $\leftarrow Proc$ succeeds, because the goal sequence is formed as below:

$\leftarrow M256, not\ Standard; \leftarrow not\ Standard.$

The goal $\leftarrow Standard$ fails, because of succeeding goal $\leftarrow \neg Standard$. Note that the goal $\leftarrow not\ ProcB$ succeeds for the failure of the goal $\leftarrow ProcB$. Among the extracted, we have $notStandard$ and $not\ ProcB$.

(iii) As is straightforward, we have got the goal $\leftarrow HDD, CD\text{-}ROM$. Then we have the goal $\leftarrow HDD$, because of the clause $HD40 \leftarrow$.

(iv) We now have the goal $\leftarrow CD\text{-}ROM$. Because of the clauses $DriveA \leftarrow$ and $\neg DriveB$, $\leftarrow CD\text{-}ROM$ succeeds.

We finally have a demand of the processor $ProcB$. The result of reasoning for (i) and (ii) is shown as in Fig.3, where the set Total Sigma $\cup$ Total Delta without negation as failure sign ("wave" in the figure) is regarded as the extracted.

5. CONCLUDING REMARKS

We present a managing scheme as an automated interface between the producer and the consumer. It can be implemented as a Web system on the internet. The scheme involves reasoning facilities, which are caused by an extended logic program.

(1) A managing scheme of the material compositions can be represented by an extended logic program, in which the reasoning procedure for the extraction may be implemented. This paper presents an abstract representation for the reasoning of a scheme, which differs from an abduction of [4].

(2) The background of the soundness of succeeding and failing derivations may be given with reference to model theory, following [1], [12], [13], [16], [19]. The first-order version of derivations are presented in [18]. The non-ground extraction is expected in relation to the model theory of [7].

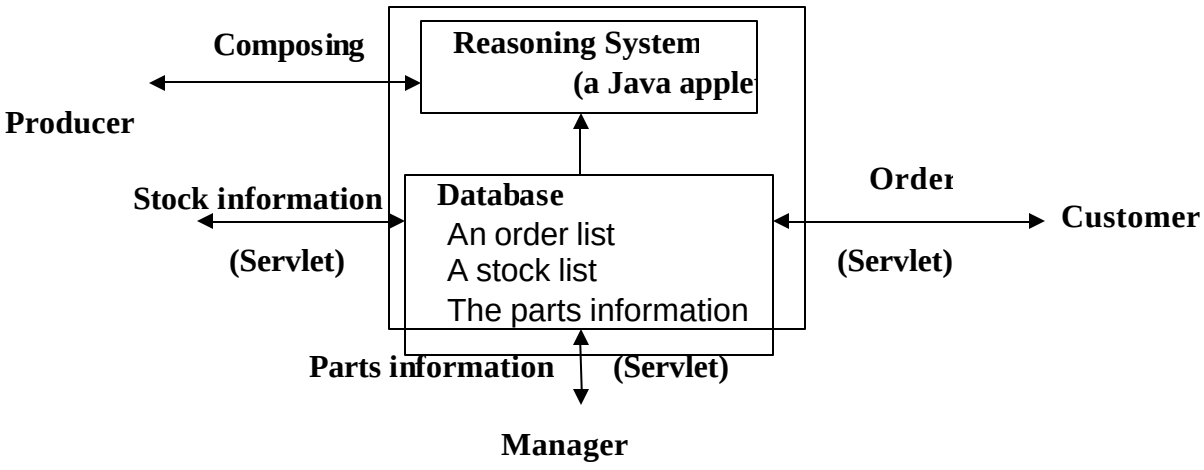

Figure 5 An architecture of the managing scheme.

(3) The notion of exceptions, as in [10], can be implemented in the derivations we present. We now apply a contradiction-free procedure to the implementation of a managing scheme on the internet, even if the managing scheme faces some contradictory state.

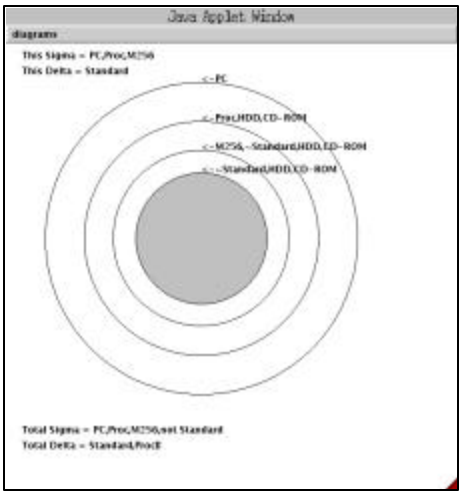

Figure 6 An output figure of the managing scheme.

REFERENCES

[1] Alferes,J.J., Damasio,C.V. and Pereira,L.M., A logic programming system for nonmonotonic reasoning, J. of Automated Reasoning, 14, pp.93-147, 1995.

[2] Analyti,A. and Pramanik,S., Reliable semantics for extended logic programs with rule priotization, J. of Logic and Computation, 5, pp.303-324, 1995.

[3] Baral,C., Knowledge Representation, Reasoning and Declarative Problem Solving, Cambridge University Press, 2003.

[4] Brogi,A., Lamma,P., Mancarella,P. and Mello,P., A unifying view for logic programming with non-monotonic reasoning, Theoretical Computer Science, 184, 1-2, pp.1-59, 1997.

[5] Dung,P.M., An argumentation-theoretic foundation for logic programming, J. of Logic Programming, 22, pp.151-177, 1995.

[6] Gelfond,M. and Lifschitz,V., Logic programs with classical negation, Proc. of th ICLP, pp.579-597, 1990.

[7] Gottlob,G., Marcus,S., Nerode,A., Salzer,G. and Subrahmanian, V. S., A non-ground realization of the stable and well-founded semantics, Theoretical Computer Science, 166, pp.221-262, 1996.

[8] Eshghi,K. and Kowalski,R.A., Abduction compared with negation by failure, Proc. of 6th ICLP, pp.234-255, 1989.

[9] Kunen,K., Signed data dependencies in logic programming, J. of Logic Programming, 7, pp.231-245, 1989.

[10] Kowalski,R.A. and Sadri,F., Logic programs with exceptions, Proc. of 7th International Conference on Logic Programming, pp.598-613, 1990.

[11] Lloyd,J.W., Foundations of Logic Programming, 2nd, Extended Edition, Springer-Verlag, 1993.

[12] Pereira,L.M., Alferes,J. J. and Aparicio,J.N., Contradiction removal within well-founded semantics, Proc. of 1st International Workshop on Logic Programming and Nonmonotonic Reasoning, pp.105-119, 1991.

[13] Pereira,P.M., Joaquim,N., Aparicio,J.N. and Alferes,J.J., Non-monotonic reasoning with logic programming, J. of Logic Programming, 17, pp.227-263, 1993.

[14] Sasakura,M. and Yamasaki,S., 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, pp.34-39, Helsinki, 2003.

[15] Shepherdson,J.C., Negation in Logic Programming, in: Minker,J. (ed.), Foundations of Deductive Databases and Logic Programming, pp.19-88, 1987.

[16] Van Gelder,A., The alternating fixpoint of logic programs with negation, J. of Computer and System Sciences, 47, pp.185-221, 1993.

[17] Yamasaki,S. and Kurose,Y., 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, pp.489-512, 2001.

[18] Yamasaki,S. and Sasakura,M.,A contradiction-free proof procedure with visualization for extended logic programs, Proc. of SSGRR02, 4, 2002.

[19] You,J.-H. and Yuan, L.Y., On the equivalence of semantics for normal logic programs, J. of Logic Programming, 22, pp.211-222, 1995.

Susumu Yamasaki received Eng.D. in information science from Kyoto University in 1980. In 1985/86, he was a visiting fellow at Dept. of Computer Science, University of Warwick, U.K. In 1987, he joined Okayama University as a professor.

Kenichi Iwata is a Ph.D. candidate at Graduate School of Natural Science and Technology, Okayama University.

Mariko Sasakura received Eng.D. in information science from Kyushu University in 2000. She is an assistant professor of Okayama University.