ELSEVIER

Decision Support Systems 22 (1998) 379–390

Decision Support Systems

# Inheritance rules for flexible model retrieval [1]

Hai Zhuge [*]

*Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing, China*

Received 30 April 1997; revised 30 September 1997; accepted 30 November 1997

**Abstract**

Convenient model management requires flexible model retrieval. This paper presents a new flexible retrieval approach for mathematical model bases. The approach defines a multi-valued model inheritance relationship among models at a signature level. The inheritance provides a rich semantic information for the retrieval mechanism to refine inexact retrieval requirements. An inheritance rules reasoning system is proposed to enhance the ability and the efficiency of the model retrieval. The interface of the approach includes an SQL-like command, which enables users to retrieve their required models with inexact requirement expressions. The approach has been implemented in a rule-based mathematical model base system RMMBS. Application examples demonstrate the retrieval approach. © 1998 Elsevier Science B.V. All rights reserved.

*Keywords:* Inheritance; Model management; Model retrieval; Rules; Signature

## 1. Introduction

Model management plays an important role in decision support systems [2]. An efficient and convenient model retrieval mechanism is a key part of model management. Topics on raising the efficiency of model retrieval have been investigated in various aspects like model base (repository) organisation [1,12,15,16], model management language [4,10], and knowledge representation and reasoning [7,13,20]. A convenient model retrieval mechanism requires flexibility. For example, some users may not know the exact model they need, or can only present an inexact requirement. A flexible model retrieval mechanism will bring convenience for them. Incorporating domain knowledge and reasoning into a model retrieval mechanism is a way to realise flexible model retrieval. Inheritance relationship among models is another kind of knowledge that can be used to increase the flexibility of model retrieval.

Inheritance has been widely studied in fields of object-oriented programming (OOP) [5,19], object-oriented software methodology [3,17], knowledge-based model construction [11], functional programming [18], and formal semantics [6]. The conventional class-based inheritance [19] is commonly regarded as an incremental definition means, which organises classes in an inheritance hierarchy. The

[*] Corresponding author. Hai 10, Qinghe Building, 100085, Beijing, P.R.China; E-mail: zhuge.hai@bj.col.com.cn

[1] The research work was supported by the National Science Foundation of China (NSFC) under Grant No. 69773002.

hierarchy establishes a kind of abstraction and similar relationship among classes. The motivation of this paper is to establish a multi-valued abstraction and similar relationship (called multi-valued model inheritance) among models for supporting flexible model retrieval.

Considering the computational characteristic of mathematical models, this paper defines the model inheritance from the point of view of functional programming. A model is described by a signature that reflects both the behaviour classification and the abstract type information about the interface of the model being described. Model inheritance is defined as a specialisation-based and multi-valued relationship among models at the signature level. Model retrieval is regarded as a heuristic graph searching on the multi-valued model inheritance hierarchy. Rules are proposed to formalise the logic relationship among different values of the model inheritance. These rules together with a set of heuristic rules are used to support an efficient and flexible model retrieval on the model base with the multi-valued model inheritance hierarchy.

The paper proceeds with an overview of a rule-based mathematical model base system RMMBS, where the flexible model retrieval plays a key role in the model management mechanism. Section 3 defines the multi-valued model inheritance based on the definition of function specialisation and a signature representation of models. Section 4 proposes a set of inheritance rules that forms a reasoning basis for flexible model retrieval. Section 5 presents the flexible model retrieval with four parts: the characteristic of the multi-valued model inheritance hierarchy, heuristic rules and searching strategies, an SQL-like retrieval command, as well as a retrieval example. The closely related works and discussions are presented in Section 6. Section 7 summarises the work.

## 2. Overview of rule-based mathematical model base system, RMMBS

Current mathematical model bases include a large amount of standard and FORTRAN-based mathematical models. These model bases are passive and flat [8,19], without model abstraction and flexible retrieval means. Current OOP Languages (OOPLs, for example, C + + ) support class abstraction by class-based inheritance relationships. However, these standard mathematical models in large amounts are difficult to be carried out 're-engineering' in OOPL or in OOP paradigm because it is hard to keep the understandability, correctness and completeness of original models. The solution of the presented system RMMBS is to separate a model base into an abstract signature level and an implementation level. The implementation level keeps the original flat structure. The multi-valued model inheritance relationship is established at the signature level for supporting model abstraction and flexible retrieval means. Models at the signature level will finally be mapped into the models at the implementation level when composing applications.

The architecture of RMMBS is shown in Fig. 1. The model base consists of a signature base, a module base and a mechanism of one-to-one and onto mapping from the signature base (abstraction level) into the module base (implementation level). The signature base is represented as: SignatureBase $= \langle \mathit{SignatureSet}, \mathit{InheritanceRelationship} \rangle$. Currently, the model base includes mathematical models such as matrix operations, solving linear equations,

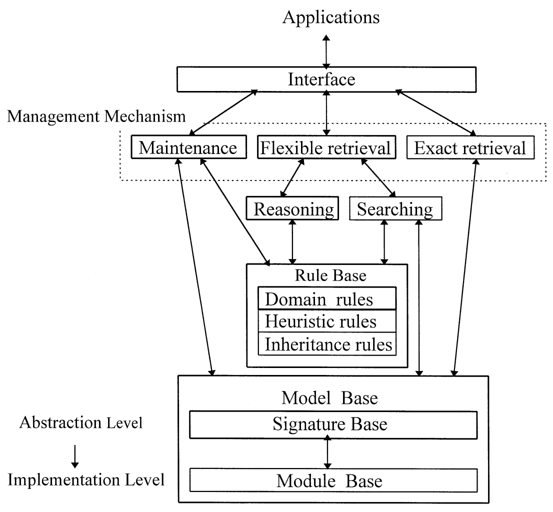


Fig. 1. Architecture of RMMBS.

solving approximate problems,..., etc. The rule base supports model base maintenance and searching. It consists of three subsets: domain rules, heuristic rules, and inheritance rules. The management mechanism consists of a maintenance mechanism, a flexible retrieval mechanism and a traditional exact retrieval mechanism. The maintenance mechanism is responsible for: 1)model naming, 2)checking redundancy and inconsistency in the model base and the rule base, and 3)maintenance operations on the model base and the rule base like appending, deleting and updating. The exact retrieval mechanism retrieves models in terms of exact requirements. The flexible retrieval mechanism retrieves models through heuristic graph searching on the multi-valued model inheritance hierarchy with the help of rules reasoning. The searching returns a set of models that are similar to each other corresponding to user's inexact requirement. The interface supports the internal applications of decision support systems and users' direct model retrievals by an SQL-like command and a graph-based navigator. The SQL-like command will be discussed in Section 5.3.

## 3. Multi-valued model inheritance

Model description is the first step for formalising model inheritance. Abstraction frame structure [7] and signature [21] are practicable approaches for model description. This paper utilises the signature approach, which focuses on the type information about the interface of the model being described. The ML signature is similar to Modula-3 interface modules and Ada definition modules [14,21]. A signature herein is defined by a model name (a coded identity) and a set of function types. A function type takes the form of $\sigma \to \tau$ [9,21], where $\sigma$ (input type) and $\tau$ (output type) can be basic-types (for example, *Integer*, *Real* and *Boolean*) or constructed-types (for example, *Record*, *Set*, *List*, and *Function*). $\sigma$ can be the Cartesian product of $n$ types (i.e., $\sigma_1 \times \ldots \times \sigma_n$). Each function type associates with a function name (a coded identity). For example, the function computing complex matrix minus (coded as *MMinC* in RMMBS) is described as $MMinC$: $MC \times MC \to MC$, where *MMinC* is the function name, $MC \times MC \to MC$ is the function type, a mapping from two complex matrix types (coded as $MC$) into a complex matrix type.

Type restriction, $\tau \subseteq \tau'$, means: 1) if $\tau$ and $\tau'$ are basic-types, then they are lexical identical type variables (for example, *Real* and *Real* are lexical identical types) or $\tau \subset \tau'$ (for example, $PositiveReal \subset Real$); 2) if $\tau$ and $\tau'$ are constructed-types, $\tau = TOp(\tau_1, \ldots, \tau_n)$ and $\tau' = TOp'(\tau'_1, \ldots, \tau'_n)$, where $TOp$ and $TOp'$ represent type operators, then $TOp = TOp'$ and $\tau_i \subseteq \tau'_i$ (for $i = 1, 2, \ldots, n$). For example, let $\sigma = \sigma_1 \times \ldots \times \sigma_m$ and $\sigma' = \sigma'_1 \times \ldots \times \sigma'_n$, $\sigma' \subseteq \sigma$ means $m = n$ and $\sigma'_i \subseteq \sigma_i$ for $i = 1, 2, \ldots, m$. A function can be specialised through the restriction of its input type and output type.

DEFINITION 1. Let $f$: $\sigma \to \tau$ and $f'$: $\sigma' \to \tau'$ be two functions. If 1) $\sigma' \subseteq \sigma$ and $\tau' \subseteq \tau$, 2) for every $\xi \in \tau$ and $\xi' \in \tau'$, if $\xi = \xi'$, then $f(\xi) = f'(\xi')$, and 3) the axioms of $f'$ imply the axioms of $f$, then $f'$ is called a function specialisation from $f$, denoted as $f'\text{-F} \to f$.

The definition of function specialisation implies: 1) reflexive: $f\text{-F} \to f$; 2)transitive: if $f''\text{-F} \to f'$ and $f'\text{-F} \to f$, then $f''\text{-F} \to f$; 3) if $f'\text{-F} \to f$, then $f'$ is behaviour compatible with $f$ (i.e., the function specialisation satisfies the behaviour compatibility [19]).

In practice, formal approaches for axiom expression and checking can not be easily manipulated by ordinary programmers and users. RMMBS uses name coding rules (belong to the domain rule base) to classify functions. The name of the specialised function $f'$ is coded by extending or revising a sub-string of the original function name $f$ (assume $f'\text{-F} \to f$). So the existence of the specialisation relationship between $f$ and $f'$ can be judged by comparing their name strings.

From the point of view of functional programming, model can be regarded as a set of functions. Hence the signature of a model can be represented as $ModelName = \{f_1\text{: } \sigma_1 \to \tau_1,\ f_2\text{: } \sigma_2 \to \tau_2, \ldots,\ f_n\text{: } \sigma_n \to \tau_n\}$, where *ModelName* is a coded string reflecting the behaviour category of the model being described, and $f_i$ $(i = 1, \ldots, n)$ is unique within the set. Thereafter, model is discussed at the signature level.

DEFINITION 2. Let M and M′ be models. If there exist $\mathrm{X} \subseteq \mathrm{M}$, $\mathrm{X}' \subseteq \mathrm{M}'$, and a one-to-one and onto mapping $\theta$: $X' \to X$, for every $f' \in X'$, $\theta(f') \in X$ and $f'$-F $\to \theta(f')$, then there exists a multi-valued model inheritance between M and M′: 1) if $X = M$ and $X' = M'$, then M′ inherits from M with a specialisation value, denoted as M′-S → M; 2) if $X = M$ and $X' \subseteq M'$, then M′ inherits from M with a total value, denoted as M′-T → M; 3) if $X \subseteq M$ and $X' = M'$, then M′ inherits from M with a partial value, denoted as M′-P → M; 4) if $X \subseteq M$ and $X' \subseteq M'$, then M′ inherits from M with a revision value, denoted as M′-R → M; 5) if both X and X′ are empty (i.e., there doesn't exist an $f \in M$ and an $f' \in M'$, such that $f'$-F $\to f$), then M′ inherits from M with an empty value, denoted as M′-\ → M.

The multi-valued model inheritance is graphically explained by Fig. 2, where arrows denote inheritance direction, the shadow of M and the shadow of M′ represent X and X′ respectively. The multi-valued model inheritance can be used to form a hierarchy in a model base where the low level models (descendants) inherit from the direct high level models (ancestors) through the multi-valued inheritance arrows. The characteristic of the hierarchy will be discussed in Section 5.1. The following example presents three model inheritance relationships selected from the model base of RMMBS.

EXAMPLE: 1) The model carrying out real matrix binary operations is represented as $MBinOpR = \{MAddR: MR \times MR \to MR,\ MMinR: MR \times MR \to MR,\ MMulR: MR \times MR \to MR\}$, the model carrying out complex matrix binary operations is represented as $MBinOpC = \{MAddC: MC \times MC \to MC,\ MMinC: MC \times MC \to MC,\ MMulC: MC \times MC \to MC\}$; where *MAddR*, *MMinR*, *MMulR*, *MAddC* and *MMulC* are functions that compute real matrix addition, real matrix minus, real matrix multiplication, complex matrix addition and complex matrix multiplication respectively. Since *MAddR*-F → *MAddC*, *MMinR*-F → *MMinC*, and MMulR-F → *MMulC*, we have: *MBinOpR*-S → *MBinOpC*. 2) The model computing symmetric real matrix minus and multiplication is represented as $MMinMulRS = \{MMinRS: MRS \times MRS \to MRS,\ MMulRS: MRS \times MRS \to MRS\}$. Comparing with *MBinOpR*, we have: *MMinRS*-F → *MMinR* and *MMulRS*-F → *MMulR*, thus *MMinMulRS*-P → *MBinOpR*. 3) The model computing real matrix addition and multiplication is represented as $MAddMulR = \{MAddR: MR \times MR \to MR,\ MMulR: MR \times MR \to MR\}$. Comparing with *MMinMulRS*, we have *MMulRS*-F → *MMulR*, thus *MMinMulRS*-R → *MAddMulR*.

The model inheritance with the specialisation value ('-S → ') is the strongest one of all the values of the model inheritance. It guarantees that all the functions of the ancestor model are behaviour compatible with all the functions of the descendant model. Thus, the model inheritance with the specialisation value has the behaviour compatibility [19]. The model inheritance with the empty value ('-\ → ') is the weakest one. It does not have the behaviour compatibility. The model inheritance with the revision value ('-R → ') is stronger than the model inheritance with the empty value. It also does not have the behaviour compatibility. Both the model inheritance with the total value ('-T → ') and the model inheritance with the partial value ('-P → ') are stronger than the model inheritance with the revision value, and are weaker than the model inheritance with the specialisation

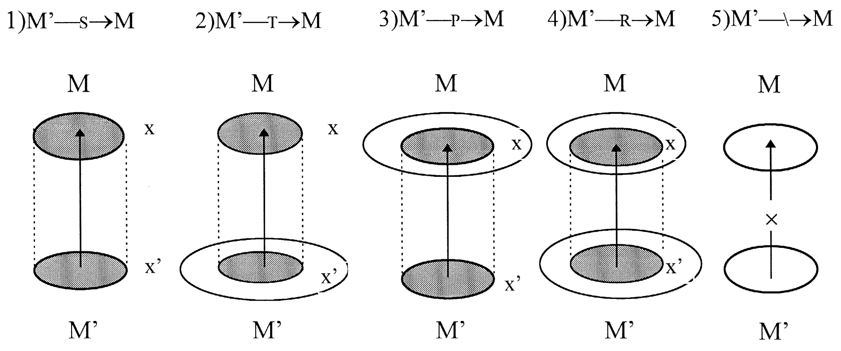


Fig. 2. Graphical explanation of the multi-valued model inheritance.

value. They can be regarded as the relaxation of the model inheritance with the specialisation value or the special cases of the model inheritance with the revision value. They do not have the behaviour compatibility. The multi-valued model inheritance relationship is established on the signature base of RMMBS, which can be further represented as: SignatureBase $= \langle$SignatureSet, $\{M_i\text{-I} \rightarrow M_j \mid M_i, M_j \in$ *ModelNameSet*, $I \in \{S, T, P, R, \backslash\}\}\rangle$.

## 4. Inheritance rules

Logic relationships exist among some values of the model inheritance. These logic relationships can be formalised by rules for supporting model retrieval. This section establishes a rule reasoning system by first establishing a basic set of rules, then deducing more rules from this set. A set of rules is listed as shown in Table 1, where ‘ $\Rightarrow$ ’ denotes the logical deduction, and ‘,’ means logic AND.

The model inheritance with the specialisation value, the total value, the partial value, and the empty value have transitive characteristic, shown as Rule1–4 in Table 1. The model inheritance with the revision value does not have transitive characteristic. For example, let $M_3\text{-R} \rightarrow M_2$ and $M_2\text{-R} \rightarrow M_1$. For $M_2$, the function set inherited by $M_3$ may disjoin the function set inherited from $M_1$, thus $M_3\text{-R} \rightarrow M_1$ can not be concluded.

PROOF of Rule1. Let $M_1\text{-S} \rightarrow M_2$ and $M_2\text{-S} \rightarrow M_3$. According to definition 2, there exists one-to-one and onto mappings $\theta$: $X_1 \rightarrow X_2$ and $\theta'$: $X_2 \rightarrow X_3$; $X_1 = M_1$, $X_2 = M_2$, and $X_3 = M_3$; and for every $f_1 \in X_1$, $f_2 \in X_2$, $f_1\text{-F} \rightarrow \theta(f_1)$, $f_2\text{-F} \rightarrow \theta'(f_2)$. Since ‘ $=$ ’ and ‘$\text{-F} \rightarrow$ ’ are transitive, there exist a one-to-one and onto mapping $\theta' \cdot \theta$: $X_1 \rightarrow X_3$, $X_1 = M_1$ and $X_3 = M_3$, for every $f_1 \in X_1$, $f_1\text{-F} \rightarrow \theta' \cdot \theta(f_1)$. Hence $M_1\text{-S} \rightarrow M_3$. QED.

Similarly, we can prove Rule2–4. Rule5–9 represent the implication relationships among some values of the model inheritance.

PROOF of Rule5. Let $M_1\text{-S} \rightarrow M_2$, $\theta$: $X_1 \rightarrow X_2$ is a one-to-one and onto mapping. According to definition 2, $X_1 = M_1$ and $X_2 = M_2$. Since $X_1 = M_1$ implies $X_1 \subseteq M_1$, according to definition 2, $M_1\text{-T} \rightarrow M_2$ hold. QED.

Similarly, we can prove Rule6–9. The transitivity characteristics and the implication relationships enable different values of the model inheritance to be connected for deduction purposes. Rule10 represents the logic deduction relationship between ‘$\text{-R} \rightarrow$ ’ and ‘$\text{-}\backslash \rightarrow$ ’. It can be proved directly by definition 2. Rule1–10 form a basic set of the model inheritance rules. More rules can be got through reasoning on this set. Rule11–15 reflect the logic deduction rela-

Table 1
Model inheritance rules

| No. | Rules | Classification |
|---|---|---|
| Rule1 | $M_1\text{-S} \rightarrow M_2, M_2\text{-S} \rightarrow M_3 \Rightarrow M_1\text{-S} \rightarrow M_3$ | Transitive |
| Rule2 | $M_1\text{-T} \rightarrow M_2, M_2\text{-T} \rightarrow M_3 \Rightarrow M_1\text{-T} \rightarrow M_3$ | Transitive |
| Rule3 | $M_1\text{-P} \rightarrow M_2, M_2\text{-P} \rightarrow M_3 \Rightarrow M_1\text{-P} \rightarrow M_3$ | Transitive |
| Rule4 | $M_1\text{-}\backslash \rightarrow M_2, M_2\text{-}\backslash \rightarrow M_3 \Rightarrow M_1\text{-}\backslash \rightarrow M_3$ | Transitive |
| Rule5 | $M_1\text{-S} \rightarrow M_2 \Rightarrow M_1\text{-T} \rightarrow M_2$ | Implication |
| Rule6 | $M_1\text{-S} \rightarrow M_2 \Rightarrow M_1\text{-P} \rightarrow M_2$ | Implication |
| Rule7 | $M_1\text{-S} \rightarrow M_2 \Rightarrow M_1\text{-R} \rightarrow M_2$ | Implication |
| Rule8 | $M_1\text{-P} \rightarrow M_2 \Rightarrow M_1\text{-R} \rightarrow M_2$ | Implication |
| Rule9 | $M_1\text{-T} \rightarrow M_2 \Rightarrow M_1\text{-R} \rightarrow M_2$ | Implication |
| Rule10 | $M_1\text{-R} \rightarrow M_2, M_2\text{-}\backslash \rightarrow M_3 \Rightarrow M_1\text{-}\backslash \rightarrow M_3$ | Deduction |
| Rule11 | $M_1\text{-S} \rightarrow M_2, M_2\text{-T} \rightarrow M_3 \Rightarrow M_1\text{-T} \rightarrow M_3$ | Deduction |
| Rule12 | $M_1\text{-S} \rightarrow M_2, M_2\text{-P} \rightarrow M_3 \Rightarrow M_1\text{-P} \rightarrow M_3$ | Deduction |
| Rule13 | $M_1\text{-S} \rightarrow M_2, M_2\text{-R} \rightarrow M_3 \Rightarrow M_1\text{-R} \rightarrow M_3$ | Deduction |
| Rule14 | $M_3\text{-T} \rightarrow M_1, M_3\text{-T} \rightarrow M_2 \Rightarrow M_3\text{-R} \rightarrow M_1, M_3\text{-R} \rightarrow M_2$ | Deduction |
| Rule15 | $M_3\text{-P} \rightarrow M_1, M_3\text{-P} \rightarrow M_2 \Rightarrow M_3\text{-R} \rightarrow M_1, M_3\text{-R} \rightarrow M_2$ | Deduction |

tionship among some values of the model inheritance. They can be deduced from the basic set.

PROOF of Rule11. Since $M_1\text{-S} \rightarrow M_2 \Rightarrow M_1\text{-T} \rightarrow M_2$ (by Rule5), $M_1\text{-S} \rightarrow M_2$, $M_2\text{-T} \rightarrow M_3 \Rightarrow M_1\text{-T} \rightarrow M_2$, $M_2\text{-T} \rightarrow M_3 \Rightarrow M_1\text{-T} \rightarrow M_3$ (by Rule2). QED.

Similarly, we can prove Rule12–13. Rule14–15 mean that ''$M_3$ inherits from $M_1$ and $M_2$ with the total value or the partial value'' has a stronger restriction than ''$M_3$ inherits from $M_1$ and $M_2$ with the revision value''. We can prove Rule14 and Rule15 by Rule8–9.

More rules can be formed by definition 2 or deducing from existing rules. For example, new rules: Rule16: $M_1\text{-S} \rightarrow M_2$, $M_2\text{-}\backslash \rightarrow M_3 \Rightarrow M_1\text{-}\backslash \rightarrow M_3$ and Rule17: $M_1\text{-P} \rightarrow M_2$, $M_2\text{-}\backslash \rightarrow M_3 \Rightarrow M_1\text{-}\backslash \rightarrow M_3$ can be deduced from Rule7, Rule8 and Rule10. Rule reasoning enables the model base maintenance mechanism to complete some useful inheritance relationship that are not apparently described in the model base. The model inheritance rules are a kind of potential knowledge among models, which can be used to improve the ability of model retrieval. In Section 5.4, we show that inheritance rules and rule reasoning can raise the efficiency of model retrieval.

## 5. Flexible model retrieval

The presented flexible model retrieval approach is a heuristic graph searching on the model base with the multi-valued model inheritance hierarchy.

### *5.1. Characteristic of multi-valued model inheritance hierarchy*

Model bases with flat file structure provide little semantic information among models. Conventional class-based inheritance relationships organise models (coded in OOPLs) in an inheritance hierarchy shown in the left graph of Fig. 3, where $M_i \rightarrow M_j$ means that $M_i$ inherits from $M_j$. The inheritance hierarchy includes a kind of model abstraction semantic information, so it improves the flat model base structure.

The proposed multi-valued model inheritance hierarchy is shown in the right graph of Fig. 3. Models can inherit from their ancestors with multiple values. The hierarchy includes a more detailed model abstraction semantic information than the conventional inheritance hierarchy. Descendants of an ancestor are classified by different values with which descendants inherit from the ancestor. For example, descendants of $M_1$, $\{M_2, M_3, M_4, M_5\}$ is classified into three subsets: $\{M_2\}$, $\{M_3, M_5\}$, and $\{M_4\}$ according to the inheritance values: '$\text{-S} \rightarrow$ ', '$\text{-T} \rightarrow$ ', and '$\text{-R} \rightarrow$ '. With the multi-valued model inheritance semantic, the searching mechanism first decides which set the candidate node falls into by checking the inheritance relationship between $M_1$ and the target, then carries out the next step searching top-down within the candidate set (i.e., the set including the candidate node). Assume $\{M_3, M_5\}$ is the candidate set, the next step searching will carry out within the set. The space of searching from a root node to a target node is narrowed within the most relevant candidates. Hence the multi-valued model inheritance hierarchy

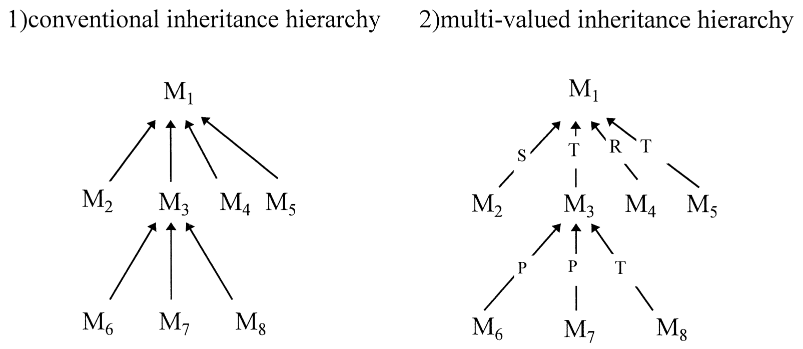

1)conventional inheritance hierarchy
2)multi-valued inheritance hierarchy
M1
M2
M3
M4
M5
M6
M7
M8
S
T
R
T
P
P
T

Fig. 3. Two kinds of model base hierarchies.

Table 2
Heuristic rules

| No. | Rules | Classification |
|---|---|---|
| HRule1 | Target-S → CurrentNode ⇒ CandidateNode-S → CurrentNode | Navigation |
| HRule2 | Target-T → CurrentNode ⇒ CandidateNode-T → CurrentNode or CandidateNode-S → CurrentNode | Navigation |
| HRule3 | Target-P → CurrentNode ⇒ CandidateNode-P → CurrentNode or CandidateNode-S → CurrentNode | Navigation |
| HRule4 | Target-R → CurrentNode ⇒ CandidateNode-R → CurrentNode or CandidateNode-P → CurrentNode or CandidateNode-T → CurrentNode or CandidateNode-S → CurrentNode | Navigation |
| HRule5 | Target-\ → CurrentNode ⇒ CandidateNode-\ → CurrentNode | Navigation |
| HRule6 | $M_1$ depends on $M_2$, $M'_1$-T → $M_1$, $M'_2$-T → $M_2$ ⇒ $M'_1$ depends on $M'_2$ | Dependence |
| HRule7 | $M_1$ depends on $M_2$, $M'_1$-S → $M_1$, $M'_2$-S → $M_2$ ⇒ $M'_1$ depends on $M'_2$ | Dependence |
| HRule8 | $M'_1$ depends on $M'_2$, $M'_1$-P → $M_1$, $M'_2$-P → $M_2$ ⇒ $M_1$ depends on $M_2$ | Dependence |

can refine inexact retrieval requirements and raise the efficiency of model retrieval.

## 5.2. Heuristic rules and searching strategies

Many AI problem-solving approaches can be regarded as a graph searching. Heuristic functions are the keys to heuristic graph searching algorithms. In the presented approach, heuristic rules are used to navigate the searching besides general heuristic methods. Some heuristic rules (HRule1–5) for the searching navigation of RMMBS are listed in Table 2. For example, if the target model indirectly inherits from the current model with the total value, HRule2 navigates the searching following the inheritance path consisting of '-T → ' or '-S → '.

Heuristic rules can also be used for dealing with model dependence (HRule6–8 shown in Table 2). A model M is said dependent on another model M′, if the functions of M are realised by calling the functions of M′. In this case, if M is selected for use in a new environment, M′ should be available in the same environment. Heuristic rules can help a model management mechanism to judge whether there exists a model dependence relationship between two models, so as to decide whether include the relevant models or not before composing an application.

The searching of this approach is controlled generally by strategies. Basic strategies include: 1) if the current visiting model is more general than the target model, then carries out a top-down searching from general to special; 2) if the current visiting model is more special than the target model, then carries out a bottom-up searching from special to general; 3) if the current visiting model is more general than a model and is more special than another model, then carries out a two-way searching top-down and bottom-up.

The model inheritance hierarchy in the model base of RMMBS is formed by expanding the candidate nodes at each step during the searching process. Heuristic graph searching algorithms have been well studied in AI research field. They are suitable for this approach after incorporating the heuristic rules into heuristic functions. To avoid redundancy, we do not discuss here in detail.

## 5.3. SQL-like retrieval command

The user interface of the flexible model retrieval approach includes an SQL-like command. Users can use the command to retrieve models with inexact requirement expressions. The basic form of the select command is described as follows:

$$\begin{aligned}
&\langle \text{SelectCommand}\rangle ::= \text{Select}\,\langle\text{Quantifier}\rangle\,[\text{From MB}|(\langle\text{SelectCommand}\rangle)]\,[\text{Where}]\\
&\qquad\qquad\langle\text{Condition}\rangle\,[\text{GroupInto C}]\\
&\langle\text{Quantifier}\rangle ::= \text{all}|\text{unique}|\text{direct}|\text{alldirect}|\text{unique direct}\\
&\langle\text{Condition}\rangle ::= M_1 - \text{S} - > M_2 | M_1 - \text{T} - > M_2 | M_1 - \text{P} - > M_2 | M_1 - \text{R} - > M_2 |\\
&\qquad M_1 - \backslash - > M_2 | \text{like} M | M_1 < M_2 | M_1 > M_2 | M_1 = M_2 |\\
&\qquad M = \langle\text{String}\rangle \sim | \text{SD}(M_1, M_2, s) > \beta | (\langle\text{Condition}\rangle) |\\
&\qquad \langle\text{Condition}\rangle\ \text{and}\ \langle\text{Condition}\rangle | \langle\text{Condition}\rangle\ \text{or}\ \langle\text{Condition}\rangle,
\end{aligned}$$

where $M_i$ ($i = 1, 2$) can be a reference model name or a required model name (denoted as '$*$' in default case). The reference model name can be given by users, or be selected by an embedded command. The brackets, [...], mean the content by default. 'GroupInto C' is used to group the selected models into a

Table 3
Example models in RMMBS

| Model | Behaviour |
|---|---|
| *LSARG* | solve real general system of linear equation with iterative refinement |
| *LSACG* | solve complex general system of linear equation with iterative refinement |
| *LSLRT* | solve a real triangular system of linear equations |
| *LSLCT* | solve a complex triangular system of linear equations |
| *LSLSF* | solve a real symmetric system of linear equations without iterative |

component C. 'direct' refers to the direct ancestor(s) or descendant(s). 'Like M' means that the required model inherits from M (or is inherited by M) with all the values except the empty value. '$M_1 < M_2$' means that $M_1$ is more special than $M_2$. '$M_1 > M_2$' means that $M_1$ is more general than $M_2$. '$M = \langle\text{String}\rangle \sim$' means that the name of the required model begins with a given string. '$SD(M_1, M_2, s) > \beta$' means that the similarity degree between the required model $M_1$ and the reference model $M_2$ is bigger than $\beta$, $\beta \in [0, 1]$. $s$ is a variable representing: $M_1 < M_2$ when $s = -1$, $M_1 > M_2$ when $s = 1$, and $M_1 = M_2$ when $s = 0$.

The command is used to select the quantifier indicated model(s) from a model set satisfying a given condition. The model set can be a model base or a view of a model base formed by an embedded command. A simple form of the embedded command is: Select ⟨Quantifier⟩ From (Select ⟨Quantifier⟩ From MB Where condition1) Where condition2. Embedded commands support top-down refinement strategy. For example, users can use 'like M' as the first condition to select an initial set of candidate models from a model base, then use '$*$-P $\rightarrow$ M' as the second condition to refine the initial candidate set.

## 5.4. Retrieval example

Example models shown in Table 3 are selected from the model base of RMMBS. They originate from the IMSL Math/Library. The signatures of these models are described as: *LSARG*: $Int \times MR \times Int \times VR \times Int \rightarrow VR$, *LSACG*: $Int \times MC \times Int \times VC \times Int \rightarrow VC$, *LSLRT*: $Int \times MRT \times Int \times VR \times Int \rightarrow VC$, *LSLCT*: $Int \times MCT \times Int \times VR \times Int \rightarrow VC$, and *LSLSF*: $Int \times MRS \times Int \times VR \times Int \rightarrow VC$; where *VR*, *VC*, *MRT*, *MCT* and *MRS* represent the real vector type, the complex vector type, the real triangular matrix type, the complex triangular matrix type and the real symmetric matrix type respectively, and satisfy: $VR \subseteq VC$, $MRT \subseteq MCT$, $MRS \subseteq MR \subseteq MC$, $MCT \subseteq MC$, and $VR \subseteq VC$.

The multi-valued model inheritance relationships among these models are described in Fig. 4. The dashed arrows represent the deduced inheritance relationships: *LSLRT*-P $\rightarrow$ *LSACG* and *LSLSF*-P $\rightarrow$ *LSACG*, which are got by inheritance rules reasoning. Deduced inheritance relationships include more relevant models into the candidate sets at every step of searching. For example, the searching without the rule reasoning classifies the descendants of *LSACG* into two sets, {*LSARG*} and {*LSLCT*}, according to the inheritance values '-S $\rightarrow$' and '-P $\rightarrow$' respectively. Assume {*LSARG*} is the candidate at this step and our target is *LSLSF*. The searching will carries out another step to expand the target *LSLSF* into the candidate set of *LSARG*. Alternatively, if we use the rule reasoning, the descendants of *LSACG* are classified into two sets, {*LSARG*} and {*LSLSF*, *LSLCT*, *LSLRT*}, according to the inheritance values '-S $\rightarrow$' and '-P $\rightarrow$' respectively. The target *LSLSF* is expanded into the candidate set of *LSACG* at first step of searching. So the searching with the rule reason-

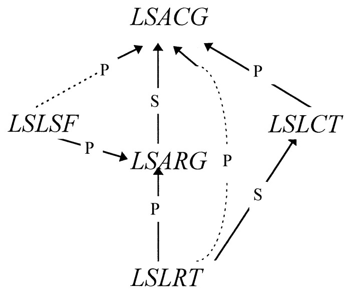


Fig. 4. A multi-valued model inheritance example.

ing is more efficient than the searching without the rule reasoning. Besides, the rule reasoning can let users know why and how the target is reached through showing the reasoning path, like the explanation process of expert systems.

Users can use various forms of the SQL-like command to retrieve their required models in an inexact way. For example: 1)users can use 'Select all $*$-S $->$ *LSLCT*' to retrieve all the models that inherit from *LSLCT* with the specialisation value. Since the target is more special than *LSLCT*, the searching carries out the top-down strategy. Heuristic rules navigate the searching following the inheritance path consisting of '-S $\rightarrow$ '. The command returns {*LSLRT*}. 2)Users can use 'Select all direct $*$-P $->$ *LSARG*' to retrieve all the direct descendants that inherit from *LSARG* with the partial value. The searching carries out the top-down strategy. Heuristic rules navigate the searching following the inheritance path consisting of '-P $\rightarrow$ ' or '-S $\rightarrow$ '. The command returns {*LSLSF*, *LSLRT*}. 3)Users can use 'Select all ($*$-S $->$ *LSACG*) and (*LSLRT*-P $->$ $*$)' to retrieve all the models that inherit from *LSACG* with the specialisation value, and are inherited by *LSLRT* with the partial value. The searching carries out two-ways (i.e., top-down from *LSACG* and bottom-up from *LSLRT*) and returns {*LSARG*}. 4)Users can use 'Select all like *LSARG*' to retrieve all the models like *LSARG*. The searching carries out two-ways (i.e., top-down from *LSARG* and bottom-up from *LSARG*) and returns {*LSACG*, *LSLRT*, *LSLSF*}.

## 6. Related works and discussions

The class-based inheritance [19] is an incremental definition mechanism for sharing code and behaviour. From the point of view of code reuse, subclasses (descendants) should be allowed to inherit by arbitrarily modifying their parent class (ancestor). But the arbitrarily modifying will break the behaviour compatibility between subclasses and their parents. Four kinds of compatibility of the inheritance are suggested from the point of view of OOP [19]. The strongest form is the behaviour compatibility, where subclasses are compatible with the axioms of their parent class. The signature (interface) compatibility is a weaker form where a subclass can substitute for any of its parents but the computing result may be different. The name compatibility is the more weaker form, where subclasses must preserve the names used in its parents. The weakest form is the cancellation, where subclasses can unrestrictedly modify their superclass.

A frame-based model representation approach for model management systems has been studied [7]. It supports both heuristic inference and deterministic inference. The suggested abstract model consists of data objects, procedures and assertions. The behaviours of a model are constrained by the assertions stated in the frame. The model abstraction is based on data abstraction, and can be regarded as a reversing relationship of subtype. It has been addressed in this work that abstraction can provide many flexible features.

Signature matching has been investigated at both the function level and the module level [21]. The relaxation of the matching is based on a kind of subtype specialisation and generalisation. However, impure specialisation or generalisation relationships play an important role in practice. New specialisation or generalisation relationships can also be deduced, for example, from the transitivity of existing specialisation or generalisation relationships.

Conventional signature expressions reflect only the type information about the interface. These expressions can not distinguish among those models that have different behaviours but share the same interface type. An ideal model base should include not only the signature description but also the behaviour specification for each model. However, formal specifications can not be manipulated with ease by domain users or ordinary programmers in practice.

The multi-valued model inheritance presented in this paper is different from the conventional class-based inheritance. The presented inheritance is defined in terms of how (totally or partially) a descendant inherits from an ancestor and whether new behaviours are added or not into the descendant. Behaviours of an ancestor to be inherited can be arbitrarily selected, and new behaviours can be arbitrarily added into the descendant, but the modifications of the inherited behaviours are restricted by the function specialisation that keeps behaviour compatibility. The conventional inheritance does not have such a restriction. For subtype inheritance, a subtype is defined by adding predicates into its parent, and behaviour modifications are heavily constrained. So, the presented inheritance has a weaker restriction than the subtype inheritance.

The signature expression approach used in this paper reflects both the behaviour classification and the type information about the interface (input and output) of a model. The behaviour classification of models is realised through name coding. Coded names can differentiate among those models that have the same interface type but have different behaviours. The approach can be easily manipulated by domain users or ordinary programmers.

## 7. Summary

The presented flexible model retrieval approach is based on a heuristic searching on the model base with the multi-valued model inheritance hierarchy. The main contribution of this work concerns three aspects. Firstly, we propose a multi-valued model inheritance relationship. It has a richer semantic than the conventional class-based inheritance, thus supports retrieval mechanism to narrow the searching space for inexact retrieval expressions. Secondly, we propose an inheritance rules reasoning system. It supports a deep reasoning for flexible model retrieval as an extension to the domain rules reasoning. The inheritance rules reasoning can deduce more inheritance relationships from the existing inheritance relationships, raise the searching efficiency, and explain the retrieval result. We also present a set of heuristic rules for searching navigation. These inheritance rules and heuristic rules support a convenient and intelligent model retrieval. Thirdly, we provide a solution for establishing the inheritance relationship on flat and standard mathematical model bases. We realise the flexible model retrieval approach in the system RMMBS. Users can use an SQL-like command to express their inexact retrieval requirements and to retrieve models efficiently and conveniently.

Based on the proposed approach and the system RMMBS, we have developed a problem-oriented model base system PROMBS. Users can use a problem-oriented language to describe problems, and the system solves problems with components that are the encapsulations of basic models. A multi-valued component inheritance relationship and an inheritance rules reasoning system play a key role in the component management of PROMBS.

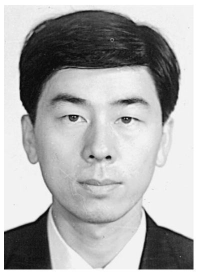
## Acknowledgements

The author thanks the editor and anonymous referees for their helpful comments on the earlier version of this paper.

## References

[1] D. Batory, S. O'Malley, The design and implementation of hierarchical software systems with reusable components, ACM Trans. Software Eng. Methodol. 1 (4) (1992) 355–398.
[2] R.W. Blanning, Model management systems: An overview, Decision Support Systems 9 (1993) 9–18.
[3] G. Booch, Object-oriented design with applications, Benjamin-Cummings, Redwood City, CA, (1990).
[4] G. Bradley, R. Clemence, A type calculus for executable modelling languages, IMA J. Math. Manage. 3 (1) (1988) 277–291.
[5] P.S. Canning, W.R. Cook, W.L. Hill, W.G. Olthoff, Interfaces for strongly-typed object-oriented programming, OOPSLA'89 Proceedings, New Orleans, Oct. (1989) 457–467.
[6] W. Cook, J. Palsberg, A denotational semantics of inheritance, OOPSLA'89 Proceedings, New Orleans, Oct. (1989) 433–443.
[7] D.R. Dolk, B.R. Konsynski, Knowledge representation for Model management systems, IEEE Trans. Software Eng. SE–10 (6) (1984) 619–628.
[8] P.D. Felice, Reusability of mathematical software: A contribution, IEEE Trans. Software Eng. 19 (8) (1993) 835–843.
[9] A.J. Field, P.G. Harrison, Functional programming, Addison-Wesley, Reading, MA, (1988).
[10] S.N. Hong, M.V. Mannino, B. Greenberg, Measurement theoretic representation of large, diverse Model bases—the unified modelling language $L_U$, Decision Support Systems 10 (1993) 319–340.
[11] R. Krishnan, Knowledge based aids for model construction, Ph.D. Dissertation, The University of Texas at Austin, Austin, TX, (Nov. 1987).
[12] M.L. Lenard, An object-oriented approach to Model management, Decision Support Systems 9 (1993) 67–73.
[13] M. Mannino, B. Greenberg, S. Hong, Model libraries: Knowledge representation and reasoning, ORSA J. Computing 2 (3) (1990) 287–301.

[14] R. Milner, M. Tofte, R. Harper, The definition of Standard ML, MIT Press, Cambridge, Mass, (1990).
[15] W.A. Muhanna, An object-oriented framework for Model management and DSS development, Decision Support Systems 9 (1993) 217–229.
[16] V. Rajlich, J.H. Silva, Evolution and reuse of orthogonal architecture, IEEE Trans. Software Eng. 22 (2) (1996) 153–157.
[17] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, W. Lorensen, Object-oriented modelling and design, Prentice-Hall, Englewood Cliffs, NJ, (1991).
[18] S. Thompson, Type theory and functional Programming, Addison-Wesley, Reading, MA, (1991).
[19] P. Wegner, Concepts and paradigms of object-oriented programming, OOPS Messenger 1 (1) (1990) 7–87.
[20] S.S. Yau, J.J. Tsai, Knowledge representation of software components interconnection information for large-scale software modifications, IEEE Trans. Software Eng. SE–13 (3) (1987) 355–361.
[21] A.M. Zaremski, J.M. Wing, Signature matching: A tool for using software libraries, ACM Trans. Software Eng. Methodol. 4 (2) (1995) 146–170.



Hai Zhuge is an associate professor at the Institute of Software, Chinese Academy of Sciences. He was a post-doctoral research fellow at the same institute from 1992 to 1994. He received the Ph.D. in computer science from Zhejiang University, China, in 1992. His current research interests include: problem-oriented and agent-based model base systems, object-based analogical reasoning, and multiple cognitive spaces inter-operation model for group decision. He is now leading a research group on distributed continue group decisions for dynamic environment.