

Title	Computer-acceptable Set Theory for Model Construction
Author(s)	Takahara, Yasuhiko
Citation	
Issue Date	2005-11
Type	Conference Paper
Text version	publisher
URL	<a href="http://hdl.handle.net/10119/3882">http://hdl.handle.net/10119/3882</a>
Rights	2005 JAIST Press
Description	The original publication is available at JAIST Press <a href="http://www.jaist.ac.jp/library/jaist-press/index.html">http://www.jaist.ac.jp/library/jaist-press/index.html</a> , IFSR 2005 : Proceedings of the First World Congress of the International Federation for Systems Research : The New Roles of Systems Sciences For a Knowledge-based Society : Nov. 14-17, 2002, Kobe, Japan, Symposium 3, Session 4 : Intelligent Information Technology and Applications Models and Systems Engineering

# Computer-acceptable Set Theory for Model Construction

Yasuhiko Takahara

Department of Management Information Science, Chiba Institute of Technology  
Tsudanuma, Narashino, Chiba, Japan  
takahara@pf.it-chiba.ac.jp

## Abstract

We have proposed a new information system development approach which is called model theory approach [1]. The approach specifies a model in set theory. Once a model is specified in set theory, it can be translated into an executable system by set compiler provided by the approach. This paper investigates set theory for the model specification, which is called computer-acceptable set theory.

**Keywords:** system development, systems approach, model theory, set theory

## 1. MODEL THEORY APPROACH

We have proposed a new information system development approach which is called model theory approach. The model theory approach is based on recognition that due to rapid progress of the information technology, in particular, the hardware technology, many systems are now running on PCs and hence a new systems development methodology must be proposed. The model theory approach addresses the request. Its features are summarized as below.

1. The model theory approach is not concerned with software development or algorithm development but systems development of an information system.

2. It asserts that systems development must be done based on the systems theory. In fact model construction of the approach is based on GST (general systems theory) systems concepts.

3. It covers not only a transaction processing system but also a problem solving system. Since the skeleton of a management information system is constituted by the two systems, the approach can address the entire structure of an MIS.

4. A system model is specified in set theory based on a formal representation of an information system.

5. An executable system, which is implemented in extProlog, is automatically generated from the model specification.

## 2. User Model Construction in Model Theory Approach

This paper uses TPS (transaction processing system) development as illustration of the model theory approach. Fig. 1 shows an implementation structure of a TPS, which is called modTPS (model-based TPS).

The modTPS is decomposed into two sub-systems, standardized interface and user model. The standardized user interface consists of two sub-interfaces, external UI built in PHP on a regular browser and internal UI built in extended Prolog (extProlog). The sub-interfaces are

prepared by the system as black boxes for modTPS development.

The real activity of the modTPS is performed by the user model. It consists of a family of atomic processes. A user command is represented by a pair (action name, action parameter). Since the function implied by an action name is carried out by an atomic process, each action name is associated with by one atomic process. The input to the process is an action parameter provided by the user.

An atomic process consists of two components, interface and implementation. When an action name and its parameter are specified by an user as an input, an interface component corresponding to the action name is activated by the internal UI. The interface component prepares the action parameter as a list (paralist) and sends it to the implementation component.

The implementation component triggers state transition in the file system using the paralist. The content of the file system is consequently modified by the component. In Fig. 1 the modification is illustrated as transition of the current state  $c$  to the next state  $c2$ . A proper response is also prepared according to the state transition.

Fig. 2 shows the development procedure of a modTPS. A system developer must provide a user model of a target system in set theory. The model is constructed based on a DFD of the target system following the formal representation of a user model. Then, the setcompiler supported by the approach translates it into an executable system in extProlog. On translation the internal UI is attached to the system. Finally, the generated model is executed under control of the external

UI, which accepts a user's request and outputs a response while updating the file system

An abstracted user model of modTPS is represented by the following structure:

user model structure =

$\langle \text{MactionName}, \text{ActionName}, \text{ResName}, \text{AttrName}, \text{FName}, \text{maction}, \text{homefile}, \text{fstruct}, \text{para}, \{\text{delta\_lambda}, \text{actionN}_i\}_i \rangle$

where

1. MactionName: set of macro action names;
2. ActionName: set of action names;
3. ResName: set of response names;
4. AttrName: a set of attribute names;
5. FName: set of file names;
6. maction:  $\text{MactionName} \rightarrow \wp(\text{ActionName})$ ;  
maction  $\in \text{MactionName}$ ;
7. homefile:  $\text{MactionName} \rightarrow \text{Fname}$ ;
8. fstruct:  $\text{FName} \rightarrow \wp(\text{AttrName})$ ; file structure representation function;
9. para:  $\text{ActionName} \rightarrow \wp(\text{AttrName})$ : parameter specification function for an action;
10. delta\_lambda:  $\text{ActionName} \rightarrow (\text{realization of ResName})$ : interface of atomic process; it is specified as below:

$\text{delta\_lambda}(\text{actionN}_i) = \text{res} \leftrightarrow \text{paralist} := \text{stdUI\_para.lib},$   
 $\text{res} := \text{actionN}_i(\text{paralist}), \text{stdUI\_res.lib}2 := \text{res};$

where paralist  $\in (\text{realization of para}(\text{actionN}_i))$ .

11. actionN<sub>i</sub>:  $\text{ActionName} \rightarrow (\text{realization of ResName})$ :  
implementation of atomic process; it is specified as below:

$\text{actionN}_i(\text{paralist}) = \text{res} \leftrightarrow c2 := \phi(c, \text{paralist}),$   
 $\text{res} := \phi(c, \text{paralist});$

$c$  and  $c2$  are states of the file system (see Fig. 1).

stdUI\_para.lib and stdUI\_res.lib are special variables representing files used for communication between the interfaces and the user model.

Macro actions correspond to modules of TPS development. Each macro action is implemented by a set of micro actions which perform real activities. A micro action will be called simply an action. The set of actions which implement a macro action is specified by the function,  $\text{maction:MactionName} \rightarrow \wp(\text{ActionName})$ . Since actions, which are collected to implement a macro action, should be coherent, it is usual that the target of their file processing activities is represented by one file. The file is called homefile for the macro action. The file is given by the function,  $\text{homefile:MactionName} \rightarrow \text{FName}$ .

A slide which is displayed on selection of a macro action is called a main slide. A main slide shows the hierarchical structure of the target TPS displaying three components, the list of macro actions,  $\text{maction}(\text{selected macro action})$  (=a set of actions constituting a selected macro action) and  $\text{homefile}(\text{selected macro action})$ . Fig. 3 shows an example of a main slide.

An executable system is obtained by automatic system generation if a user model is specified in the above structure in computer-acceptable set theory.

### 3. Well-Formed Formula of First-order Predicate Calculus

To compile the user model, it must be described in a form that can be accepted by a computer. At the same time system defined predicates and functions are introduced in order to efficiently describe  $\phi$  and  $\varphi$  of an atomic process. Set theory created in this way is called

computer-acceptable set theory.

The basic syntax of computer-acceptable set theory for the model theory approach is based on first-order predicate calculus [2].

- A relational structure is given by

$$\text{ST} = \langle A, \{r_i | i \in I\}, \{f_j | j \in J\}, \{c_k | k \in K\} \rangle$$

where  $A$  = the domain of ST, is a non empty set;  $r_i$  = a relation on A, i.e.,  $r_i \subset A \times \dots \times A$ ;  $f_j$  = a function on A, i.e.,  $f_j: A \times \dots \times A \rightarrow A$ ;  $c_k$  = a constant element of A, i.e.,  $c_k \in A$ .

- The first-order language  $L(\text{ST})$  for the structure  $\text{ST} = \langle A, \{r_i | i \in I\}, \{f_j | j \in J\}, \{c_k | k \in K\} \rangle$ , then consists of individual variables  $v_0, v_1, \dots$ ; individual constant symbol  $c_k$  for each  $k \in K$ ; a  $\lambda(i)$ -arity predicate symbol  $r_i$  for each  $i \in I$ ; a  $\mu(j)$ -ary function symbol  $f_j$  for each  $j \in J$ ; logical connectives  $\neg$  (not) and  $\&$  (and); universal quantifier  $\forall$ ; brackets  $()$ .

- The set of terms of the first-order language L,  $\text{Term}(L)$ , is the smallest set X such that all individual variables  $v_0, v_1, \dots$  and constant symbols,  $c_k$  are members of X; if  $t_1, \dots, t_{\mu(j)} \in X$ , then  $f_j(t_1, \dots, t_{\mu(j)}) \in X$  for each  $j \in J$

- The set of atomic formulas of L,  $\text{Atom}(L)$ , consists of all elements of the form  $r_i(t_1, \dots, t_{\lambda(i)})$  where  $t_1, \dots, t_{\lambda(i)} \in \text{Term}(L)$ .

- The set of well-formed formulas (or simply formulas) of L,  $\text{Form}(L)$ , is defined as the smallest set Y such that  $\text{Atom}(L) \subset Y$ ; if  $\phi, \psi \in Y$ , then  $\neg\phi, \phi \& \psi, \forall v_i \phi \in Y$ .

- Additional logical connectives  $\vee$ ,  $\rightarrow$ , and  $\leftrightarrow$  are defined in terms of the primitives as follows:  $A \vee B \equiv \neg(\neg A \& \neg B)$ ;  $A \rightarrow B \equiv \neg A \vee B$ ;  $A \leftrightarrow B \equiv A \rightarrow B \& B \rightarrow A$ ;  $\exists v_i \phi \equiv \neg \forall v_i \neg \phi$ .

The above is a standard definition of a wff. The connectives  $\rightarrow$  and  $\leftrightarrow$  are the main ingredients for a wff in the model theory approach.

#### 4. Basic Notation for Computer-acceptable

##### Set Theory

The following are extensions of the first-order language for computer-acceptable set theory. A symbol is an alphanumeric string starting with an alphabetical character.

(1) The universal quantifiers are not explicitly used in this approach. Square brackets '[' and ']' are used to represent a list structure.

(1) The conventional expression of numbers is used for numeric constant symbols. Arithmetic and relational operators are used in the usual way. The absolute operator is given by a function `abs()`.

(3) A constant symbol is given by the form `<symbol>`.

(4) "`<string>`" is used as a text-type constant symbol

(5) A symbol that is not a constant is a variable.

(6) A variable with the suffix ".g" is treated as a special variable called a global variable. A global variable is a variable that is valid over the entire model, whereas a regular variable is valid only within the statement in which it is used.

(7) A file is denoted by the form `<filename>.lib` and is treated as a global variable. As a file is a list of records, it is treated as a set in the model theory approach.

(8) The equality symbol '=' is used as a binary predicate symbol.

(9) The symbol ':=' is used as a binary predicate symbol to indicate the assignment of a value *v* to a variable *x*, i.e., `x:=v`.

(10) Due to restrictions on keyboard input, logical operator symbols are replaced by computer-acceptable

symbols (see Fig. 4). Here, "and" is replaced by ';' as per convention.

(11) A set is represented as a list term. For example, `[1,[2,3],x]` is a set expression, where "[...]" is treated as a function.

(12) Let *A*, *B* and *C* be formulas. Then, an expression (left-hand side) may be replaced as follows.

$(A \rightarrow B) \& (\neg A \rightarrow C) \equiv (A) \rightarrow (B) \text{ .otherwise } (C)$ .

(13) The following four types of formulas terminated by ';' are called statements.

(i) `<global variable>=<term>`. Used to define global variables.

(ii) `<predicate symbol>(<argument list>) <-> <formula>`. Used to define predicates.

(iii) `<function symbol>(<argument list>)=<variable> <-> <formula>`. Used to define functions.

(iv) atomic formula. Used to define a record of data.

(14) A user model in set theory is a set of statements.

(15) Quantifiers are not used explicitly in the model theory approach. However, every statement is assumed to be quantified by the universal quantifier.

(16) A meta statement "`.func([....]);`" is used to declare function symbols. If a predicate is listed in "`.func([....]);`", it can be used as a function.

(17) There are system-defined functions and predicates that are used to construct and manipulate sets, predicates and functions [3]. The following is a typical example: `project():binary function; x:=project([1,2,3],2)` implies `x=2`.

#### 5. Set

- A set is represented as a list in the model theory approach.

- A set is constructed by identifying elements of a list directly or by an extension of a predicate. A set  $Ys$  given by  $Ys = \{y | p(y, x, \langle \text{parameter list} \rangle), x \in Xs\}$  is constructed as  $Ys := \text{defSet}(p(y, x, [\text{parameter list}]), [x, Xs]); p(y, x, [\text{parameter list}]) \leftrightarrow \text{definition of } p();$
- Due to restrictions on keyboard input, set theoretic operator symbols are replaced by the computer-acceptable terms and atomic formulas given in Fig. 5.
- A new element  $x$  is added to a set (list)  $Xs$  by  $Xs2 := \text{union}(Xs, [x]).$
- An element  $x$  is deleted from a set (list)  $Xs$  by  $Xs2 := \text{minus}(Xs, [x]).$
- An element  $x$  of a set  $Xs$  is replaced by another element  $y$  by  $\text{pos} := \text{invproject}(Xs, x), Xs2 := \text{replaceList}(Xs, \text{pos}, y);$   
The predicate  $\text{invproject}(Xs, x)$  gives the position  $\text{pos}$  of  $x$  in  $Xs$ .

## 6. Predicate (Relation)

- A relation is simply represented as a predicate, which is defined in the following form:  
 $p(v_1, \dots, v_n) \leftrightarrow \text{conditions on } \{v_1, \dots, v_n\};$
- A relation is modified in two ways.
  - (1) If a relation is given by a set (its extension), the modification method of the set is applicable to the relation.
  - (2) If a relation is given by a predicate  $p(x)$ , it can be modified as follows:  $P2(x) \leftrightarrow (\text{condition}(x)) \rightarrow \text{modified statement} \text{ otherwise } p(x);$
- Quantifiers are not used explicitly in the model theory approach. However, if the target set  $Z$  is finite, the functions of the quantifiers are handled by  $\text{defSet}()$  in the

following way: (1)  $(\forall x \in Z)(p(x))$  is true iff  $Z = \text{defSet}(p2(x, x, []), [x, Z])$  where  $p2(x, x, []) \leftrightarrow p(x);$   
(2)  $(\exists x \in Z)(p(x)) = \text{true}$  iff  $[] \in \text{defSet}(p2(x, x, []), [x, Z])$  where  $p2(x, x, []) \leftrightarrow p(x);$

## 7. Functions

- A function is defined in several ways.

(1) A function may be represented as a set or a relation.

(3) Suppose  $f: X \rightarrow Y$  is specified as

$$f(x) = \begin{cases} z & \text{if condition}(x) = \text{true} \\ z' & \text{if condition}'(x) = \text{true} \\ \bullet \\ \bullet \\ z'' & \text{if condition}''(x) = \text{true}. \end{cases}$$

Then,  $f: X \rightarrow Y$  is described in the model theory approach as follows:  $f(x) = y \leftrightarrow (\text{condition}(x)) \rightarrow (y = z), (\text{condition}'(x)) \rightarrow (y = z'), \dots (\text{condition}''(x)) \rightarrow (y = z'');$

In particular, if  $f$  is specified as

$$f(x) = \begin{cases} y_1 & \text{if conditionA is true} \\ y_2 & \text{otherwise,} \end{cases}$$

$f$  is described as follows:  $f(x) = y \leftrightarrow (\text{conditionA}) \rightarrow (y = y_1) \text{ otherwise } (y = y_2);$

- According to recursive function theory, a complicated function can be defined in three ways; composition, primitive recursion, and minimization. They can be realized by the meta command  $\text{func}()$  and the system defined function  $\text{defSet}()$  [3].
- A function is evaluated in two ways: (1) If a function  $f: X \rightarrow Y$  is defined as a set, that is,  $f = \{(x, f(x)) | x \in X\}$ ,  $y$  is obtained by execution of  $\text{member}([x, y], f)$  for a given  $x$ .
- (2) If a function  $f: X \rightarrow Y$  is defined by  $f(x) = v \leftrightarrow$

(specification of  $v$ ),  $y$  is simply obtained by  $y:=f(x)$  if  $f$  is declared in `func()`.

- A function can be modified in two ways: (1) If a function  $f:X \rightarrow Y$  is defined as a set,  $f=\{(x,f(x))|x \in X\}$ , a new function  $f1: X \rightarrow Y$  where

$$f1(x) = \begin{cases} f(x) & \text{if } x \neq x1 \\ y1 & \text{otherwise,} \end{cases}$$

is created as follows:  $Ix:=\text{invproject}(f,[x,y0])$ ,  $f1:=\text{replaceList}(f,Ix,[x,y1])$ ; where the function  $\text{replaceList}(f,Ix,[x,y1])$  replaces the  $Ix$ -th element by  $[x,y1]$  in  $f$ ; (2) If  $f:X \rightarrow Y$  is defined as

```
f(x)=y <->
    (condition(x)) ->
        (y:=z),
    (condition'(x)) ->
        (y:=z'),
    .
    .
    (condition''(x)) ->
        (y:=z'');
```

$f1(x)=y <-> (x=x1) \rightarrow (y:=y1) .\text{otherwise } (y:=f(x));$

## 8. Example

A simple example illustrating set theoretic description is given below. The example is the development of a registration system for a workshop [4]. Figure 6 shows a part of a DFD for the registration system. From the DFD in Fig. 6, the following user model is constructed in computer-acceptable set theory.

```
func([delta_lambda,register]);
MactionName.g=["quit","registration","participant"];
;
registration.g=["register","reg_update","account"];
registration.lib.g=["participant","data"];
```

```
registrationHomeFile.g=["participant.lib"];
/*atomic process of register*/
register.g=["name","institute"];
delta_lambda([.register])=res <->
    paralist:=stdUI_para.lib,
    res := register(paralist),
    stdUI_res.lib2:=res;
register([name,institute])=res <->
    (notmember'[[name,institute],y], participant.lib)) ->
        (participant.lib2 :=
        union(participant.lib,[[[name,institute],[0,0,0]]]),

        res:=[name,institute,new_registration,.fee,.track])
    .otherwise

    ([fee,track,date] := participant.lib([name,institute]),
    (fee < 0 and track < 0) ->

    (res:=[name,institute,pre_registration,.receipt,.proceeding])

    .otherwise

    (res:=[name,institute,incomplete_registration,.fee,.track]));
```

`MactionName.g`, `registration.g`, and `registrationHomeFile.g` correspond to `MactionName`, `maction(registration)` and `homefile(registration)` in the user model structure. Figure 3 is the main slide of “registration”.

## REFERENCES

- [1] Takahara, Y and Liu, Y (2005), *Management Information Development: Theoretical Foundation-Model Theory Approach*, internal report, Chiba Inst. of Tech., Tsudanuma, Japan.
- [2] Bridge, J (1978), *BEGINNING MODEL THEORY*, Oxford UP.
- [3] Takahara et al (2003), *Manual for extProlog*, internal report, Chiba Inst. Of Tech., Tsudanuma, Japan.
- [4] Koizumi, T., Yoshida, K. and Nakajima, T. (2003) *Software Development*, Ohmusha (in Japanese).

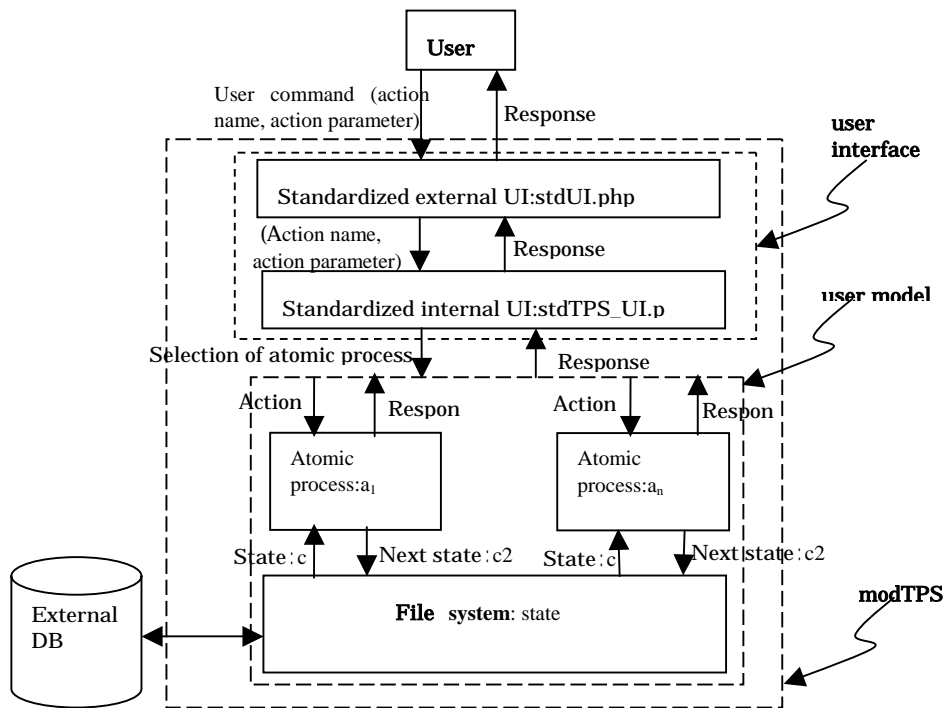


Fig. 1 Structure of modTPS

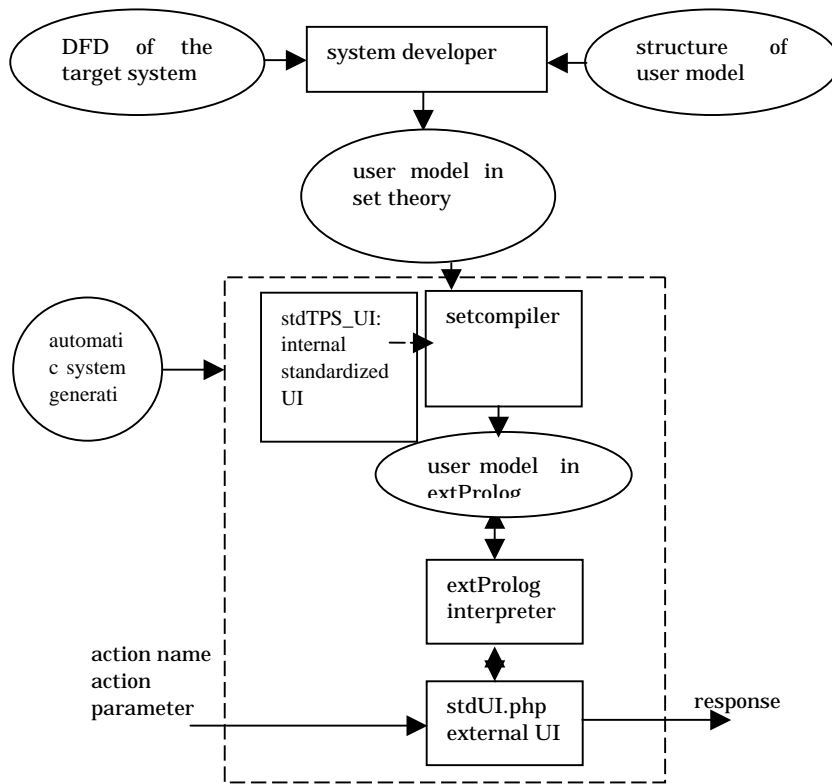


Fig. 2 Development procedure



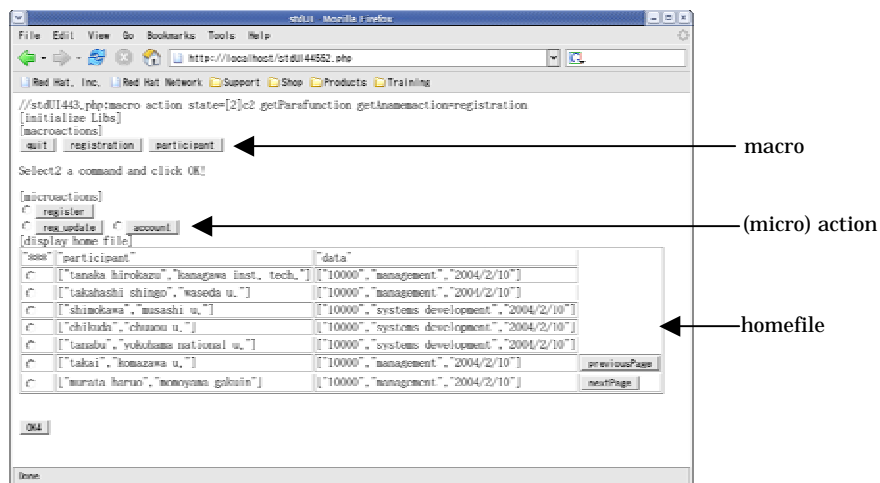


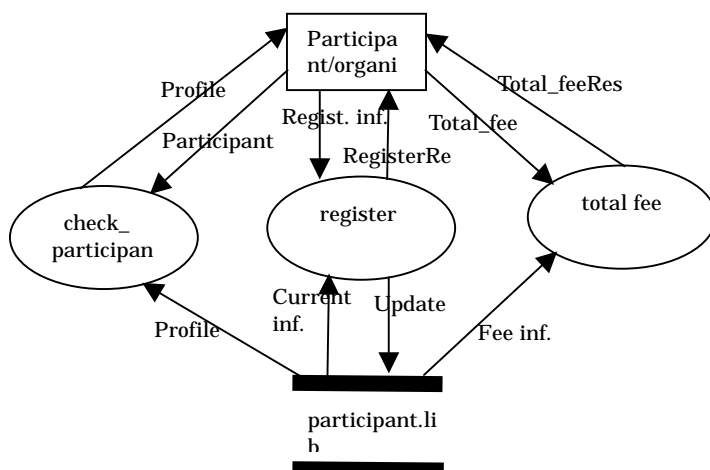
Fig. 3 Main slide

Connective	Computer-acceptable symbol	Example	Replaced example
$\neg$	not()	$\neg p$	not(p)
$\wedge$ (&)	and	p and q	p and q
$\vee$	or	P or q	p or q
$\leftrightarrow$	<->	$p \leftrightarrow q$	$p <-> q$
$\rightarrow$	->	$p \rightarrow q$	$p -> q$

Fig. 4 Connectives in set theory

Operator symbol	Computer-acceptable term	Example	Replaced example
$\cap$	intersection()	$A \cap B$	intersection(A,B)
$\cup$	union()	$A \cup B$	union(A,B)
$\times$	product	$A \times B$	product(A,B)
-	minus()	$A - B$	minus(A,B)
	cardinality()	A	cardinality(A)
$\in$	.in (member)	$x \in Xs$	$x$ .in $Xs$ (member(x,Xs))
$\notin$	.notin (notmember)	$x \notin Xs$	$x$ .notin $Xs$ (notmember(x,Xs))
$\subset$	subset()	$A \subset B$	subset(B,A)
$\not\subset$	notsubset()	$A \not\subset B$	notsubset(B,A)

Fig. 5 Operators in set theory



4. Fig. 6 DFD for registration system