Modelling Classes and Associations in Event-B

© Michael Butler

University of Southampton

March 24, 2016
Many-to-many association

user

permission

location

building

Many-to-one association
Buildings Access System

- Types: $USER, BUILDING$
- Variables: $register, permission, location$
- Invariants:
  - $register \subseteq USER$  // register is a set of users
  - $permission \in USER \leftrightarrow BUILDING$
    // relates users to the buildings they can access
  - $\text{dom}(permission) \subseteq register$
    // only register users may have permissions
  - $location \in USER \mapsto BUILDING$
    // user is located in at most one building
  - $location \subseteq permission$
    // user located in a building must have permission
    // for that building
Domain and Range

\[\text{directory} = \{ \text{mary} \rightarrow 287573, \]
\[\text{mary} \rightarrow 398620, \]
\[\text{john} \rightarrow 829483, \]
\[\text{jim} \rightarrow 398620 \}\]

\[\text{dom(directory)} = \{\text{mary, john, jim}\}\]

\[\text{ran(directory)} = \{287573, 398620, 829483\}\]
Relations

A relation is a set of ordered pairs.

A relation is a common modelling structure so Event-B has a special notation for it:

\[ T \leftrightarrow S = \mathcal{P}(T \times S) \]

So we can write:

\[ \text{directory} \in \text{Person} \leftrightarrow \text{PhoneNum} \]

Do not confuse the arrow symbols:

\( \leftrightarrow \) combines two sets to form a set.
\( \mapsto \) combines two elements to form an ordered pair.
Partial Functions

Special kind of relation: each domain element has at most one range element associated with it.

To declare $f$ as a partial function:

$$f \in X \rightarrow Y$$

This says that $f$ is a many-to-one relation

Each domain element is mapped to one range element:

$$x \in dom(f) \implies \text{card}(f[\{x\}]) = 1$$

More usually formalised as a uniqueness constraint

$$x \mapsto y_1 \in f \land x \mapsto y_2 \in f \implies y_1 = y_2$$
Function Application

We can use function application for partial functions.

If $x \in \text{dom}(f)$, then we write $f(x)$ for the unique range element associated with $x$ in $f$.

If $x \notin \text{dom}(f)$, then $f(x)$ is undefined.

If $\text{card}(f[\{x\}]) > 1$, then $f(x)$ is undefined.
A total function is a special kind of partial function. To declare $f$ as a total function:

$$f \in X \rightarrow Y$$

This means that $f$ is well-defined for every element in $X$, i.e., $f \in X \rightarrow Y$ is shorthand for

$$f \in X \rightarrow Y \land \text{dom}(f) = X$$
Classes and attributes

model of a birthday book:

**variables**  \( birthday, person \)

**invariants**

\[
\begin{align*}
\text{person} & \subseteq \text{PERSON} \\
\text{birthday} & \in \text{person} \rightarrow \text{DATE}
\end{align*}
\]

Representing \( birthday \) as a simple class diagram:

![Diagram](image)
Multiple attributes

Suppose we want to model a person’s address as well. Multiple attributes of an entity (e.g., person) are modelled as separate total functions on the same domain:

variables \( birthday, \ person, \ address \)

invariants

\[
\begin{align*}
\text{person} & \subseteq \text{PERSON} \\
\text{birthday} & \in \text{person} \rightarrow \text{DATE} \\
\text{address} & \in \text{person} \rightarrow \text{ADDRESS}
\end{align*}
\]

The common domain for both functions means every element of the set \( r \ \text{person} \), has both a birthday and an address.
Class diagram for the birthday/address book

PERSON -> birthday -> DATE

PERSON -> address -> ADDRESS
Making variable set explicit

- **person** \(\subseteq\) **PERSON**
- **DATE**
- **ADDRESS**

- **birthday**
- **address**
Secure database example

We consider a secure database. Each object in the database has a data component.

Each object has a classification between 1 and 10.

Users of the system have a clearance level between 1 and 10.

Users can only read and write objects whose classification is no greater than the user’s clearance level.

What are the entities, associations, events?
Class diagram for secure database

- **OBJECT**
  - data
- **LEVEL**
  - clear
- **DATA**
- **USER**
Making variable set explicit

object \subseteq OBJECT \quad \text{class} \quad \text{level} \subsetneq USER

data \quad \text{clear}
Types and variables

sets  OBJECT  DATA  USER
constants  LEVEL
axioms  LEVEL = 1..10

variables  object, user, data, class, clear
invariants

\[
\begin{align*}
object & \subseteq \text{OBJECT} \\
user & \subseteq \text{USER} \\
data & \in \text{object} \rightarrow \text{DATA} \\
\text{class} & \in \text{object} \rightarrow \text{LEVEL} \\
\text{clear} & \in \text{user} \rightarrow \text{LEVEL}
\end{align*}
\]

The invariant \( data \in \text{object} \rightarrow \text{DATA} \) means that \( data(o) \) is well-defined whenever \( o \in \text{object} \). Why is this important?

initialisation

\[
\begin{align*}
\text{object} := \{\} & \quad \text{user} := \{\} \\
data := \{\} & \quad \text{class} := \{\} \\
\text{clear} := \{\}
\end{align*}
\]
Adding users

\[
\text{AddUser} \triangleq \\
\text{any } u, c \text{ where } \\
u \in \text{USER} \\
u \not\in \text{user} \\
c \in \text{LEVEL} \\
\text{then} \\
u \text{ser} := \text{user} \cup \{u\} \\
\text{clear}(u) := c \\
\text{end}
\]

The new user must not already exist.
We need to provide the initial clearance level for the new user.
Adding objects

$$AddObject \triangleq$$

**any** $o, d, c$ where

- $o \in OBJECT$
- $o \notin object$
- $d \in DATA$
- $c \in LEVEL$

then

- $object := object \cup \{o\}$
- $data(o) := d$
- $class(o) := c$

end

The new object must not already exist.
We need to provide the initial classification level and data value for the new object.
Reading objects

\[
\text{Read } \hat{=} \\
\text{any } u, o, \text{ result where} \\
\quad u \in \text{user} \\
\quad o \in \text{object} \\
\quad \text{clear}(u) \geq \text{class}(o) \\
\quad \text{result} = \text{data}(o) \\
\text{end}
\]

The user must exist
The object must exist
The clearance must be ok
The data associated with the object
Writing objects

\[
\text{Write} \overset{\hat{=}}{=} \\
\text{any } u, o, d \text{ where} \\
u \in \text{user} \\
o \in \text{object} \\
clear(u) \geq \text{class}(o) \\
\text{then} \\
data(o) := d \\
\text{end}
\]

The write operation overwrites the data value associate with the object with a new value.
Changing classification and clearance levels

\[\text{ChangeClass} \triangleq \]
\[\text{any } o, c \text{ where } o \in \text{object} \quad c \in \text{LEVEL} \]
\[\text{then} \]
\[\text{class}(o) := c \]
\[\text{end} \]

\[\text{ChangeClear} \triangleq \]
\[\text{any } u, c \text{ where } u \in \text{user} \quad c \in \text{LEVEL} \]
\[\text{then} \]
\[\text{clear}(u) := c \]
\[\text{end} \]
Making classification changes more secure

Include constraints on the user who is changing the object classification:

\[ \text{ChangeClass} \triangleq \]
\[
\text{any } o, c, u \text{ where}
\]
\[
o \in \text{object}
\]
\[
c \in \text{LEVEL}
\]
\[
clear(u) \geq \text{class}(o)
\]
\[
clear(u) \geq c
\]
\[
\text{then}
\]
\[
\text{class}(o) := c
\]
\[
\text{end}
\]
Making clearance changes more secure

Include constraints on the user who is changing the object classification:

\[
\text{ChangeClear} \triangleq \\
\text{any } u, a, c \text{ where} \\
\begin{align*}
u & \in \text{user} \\
a & \in \text{user} \\
clear(a) & \geq clear(u) \\
clear(a) & \geq c \\
c & \in \text{LEVEL} \\
\end{align*}
\text{then} \\
\begin{align*}
clear(u) & := c \\
\end{align*}
\text{end}
Removing users and objects

\[\text{RemoveUser} \triangleq \]
\[
\text{any } u \text{ where } u \in \text{user} \\
\text{then} \\
\text{user} := \text{user} \setminus \{u\} \\
\text{clear} := \{u\} \triangleleft \text{clear} \\
\text{end}
\]

\[\text{RemoveObject} \triangleq \]
\[
\text{any } o \text{ where } o \in \text{object} \\
\text{then} \\
\text{object} := \text{object} \setminus \{o\} \\
\text{class} := \{o\} \triangleleft \text{class} \\
\text{data} := \{o\} \triangleleft \text{data} \\
\text{end}\]