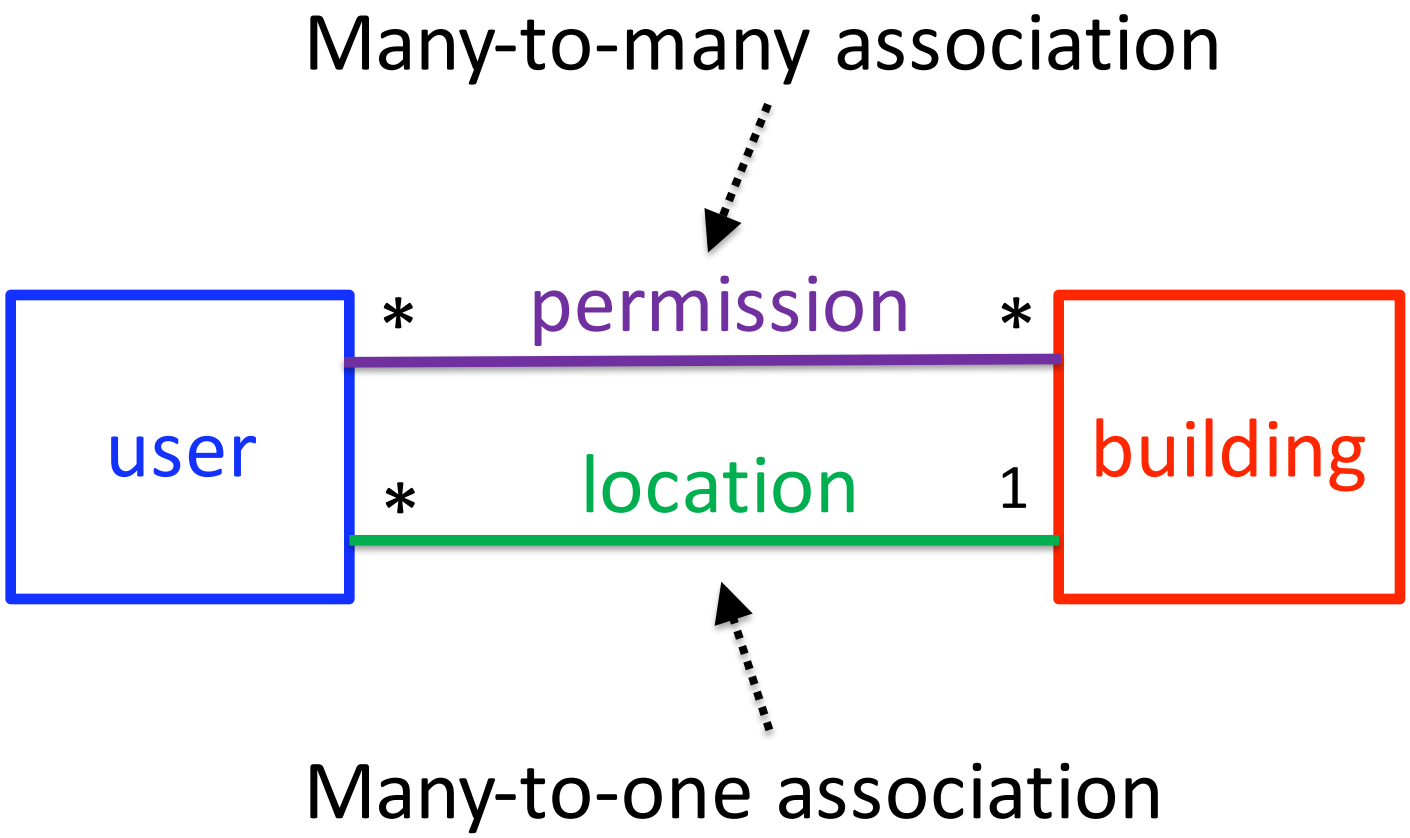
Modelling Classes and Associations in Event-B

© Michael Butler

University of Southampton

March 24, 2016

Class diagram abstraction



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
  - $dom(permission) \subseteq register$
    // only register users may have permissions
  - $location \in USER \nrightarrow BUILDING$
    // 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

$$
\begin{aligned}
\mathit{directory} \quad = \quad & \{\ \mathit{mary} \mapsto 287573, \\
& \ \ \mathit{mary} \mapsto 398620, \\
& \ \ \mathit{john} \mapsto 829483, \\
& \ \ \mathit{jim} \mapsto 398620\ \}
\end{aligned}
$$

$$
\begin{aligned}
\mathrm{dom}(\mathit{directory}) \quad &= \quad \{\mathit{mary}, \mathit{john}, \mathit{jim}\} \\
\mathrm{ran}(\mathit{directory}) \quad &= \quad \{287573, 398620, 829483\}
\end{aligned}
$$

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:

$$\boxed{T \leftrightarrow S} \quad = \quad \mathbb{P}(T \times S)$$

So we can write:

$$directory \quad \in \quad Person \leftrightarrow PhoneNum$$

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:

$$\boxed{f \in X \pfun Y}$$

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

Each domain element is mapped to one range element:

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

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

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

If $\mathit{card}(\ f[\{x\}]\ ) > 1$, then $f(x)$ is undefined.

Total Functions

A total function is a special kind of partial function. To declare f as a total function:

$$\boxed{f \in X \to Y}$$

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

$$f \in X \pfun Y \quad \land \quad dom(f) = X$$

Classes and attributes

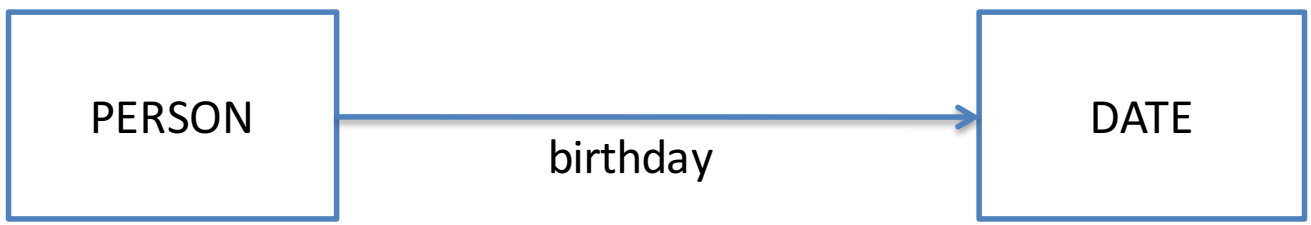
model of a birthday book:

variables $birthday, person$

invariants

$$person \subseteq PERSON$$
$$birthday \in person \rightarrow DATE$$

Representing *birthday* as a simple class diagram:



Multiple attributes

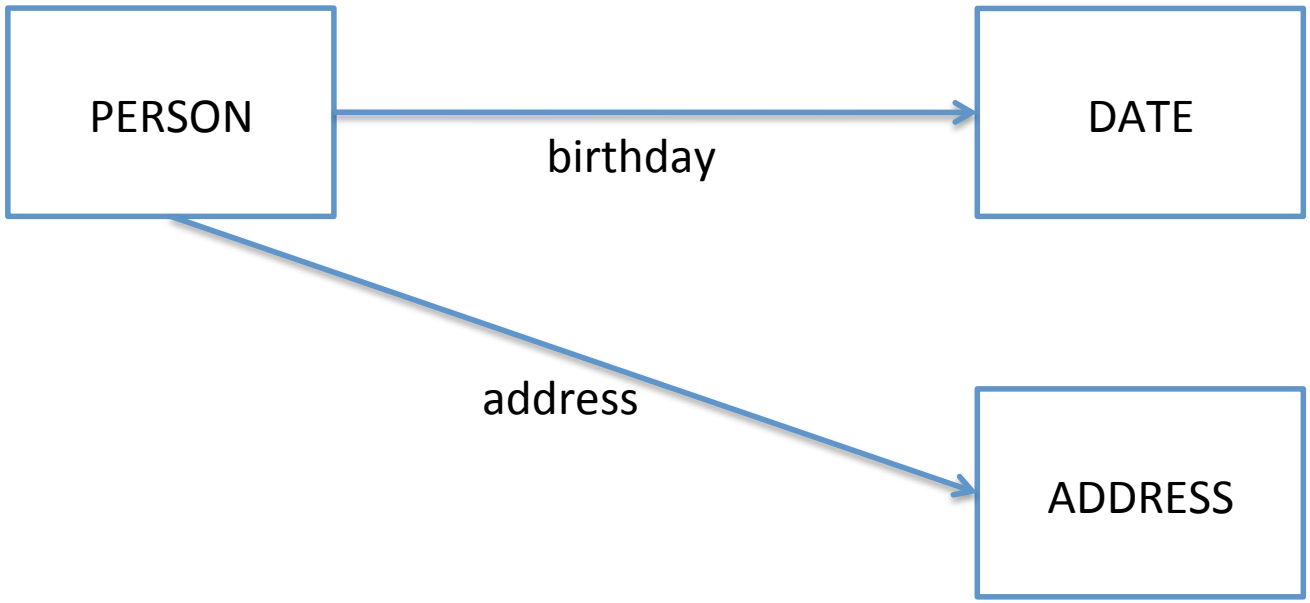
Suppose we want to model a person's address as well.
Multiple attributes of an entity (e.g., person) are modelled as separate total functions on the same domain:

variables $birthday, person, address$
invariants

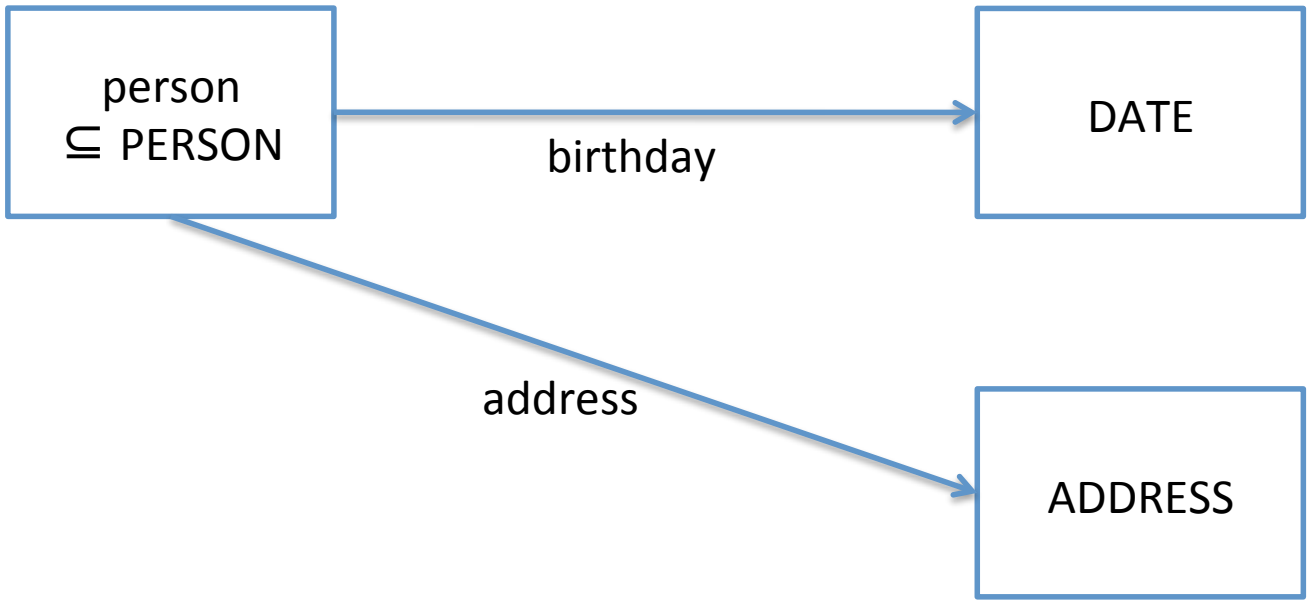
$$person \subseteq PERSON$$
$$birthday \in person \to DATE$$
$$address \in person \to ADDRESS$$

The common domain for both functions means every element of the set r *person*, has both a birthday and an address.

Class diagram for the birthday/address book



Making variable set explicit



Secure database example

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

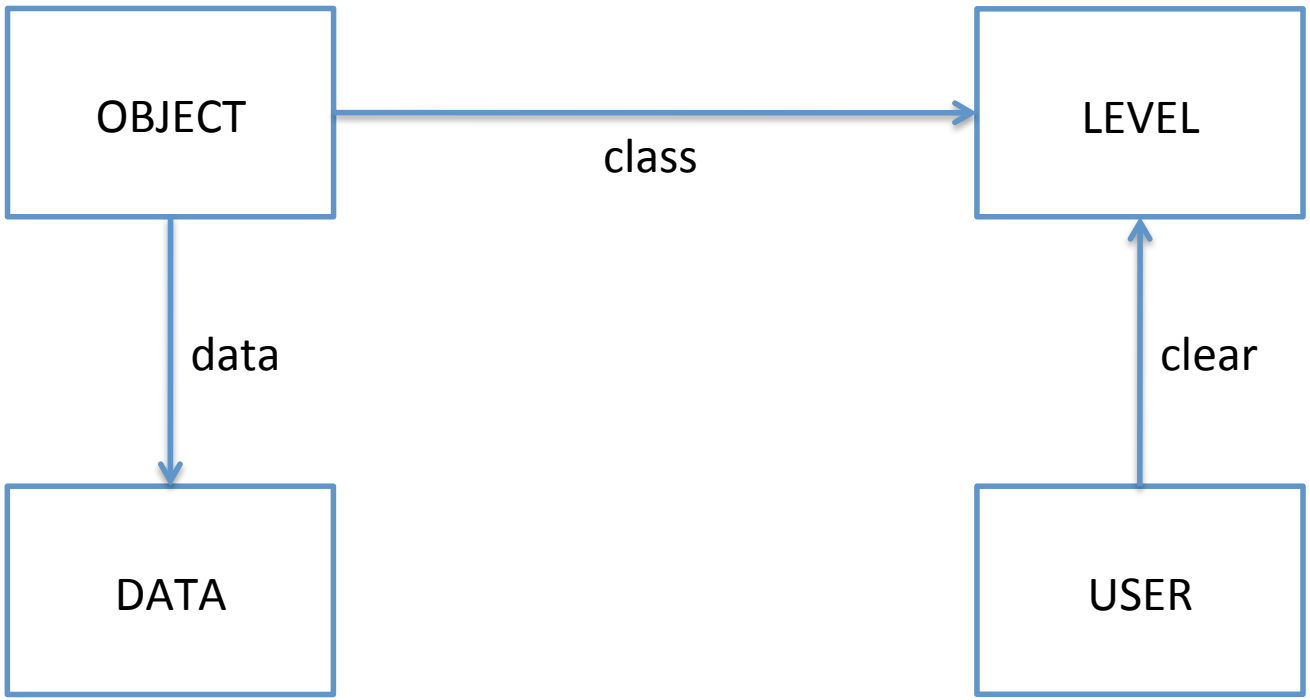
Each object has a classification between 1 and 10.

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

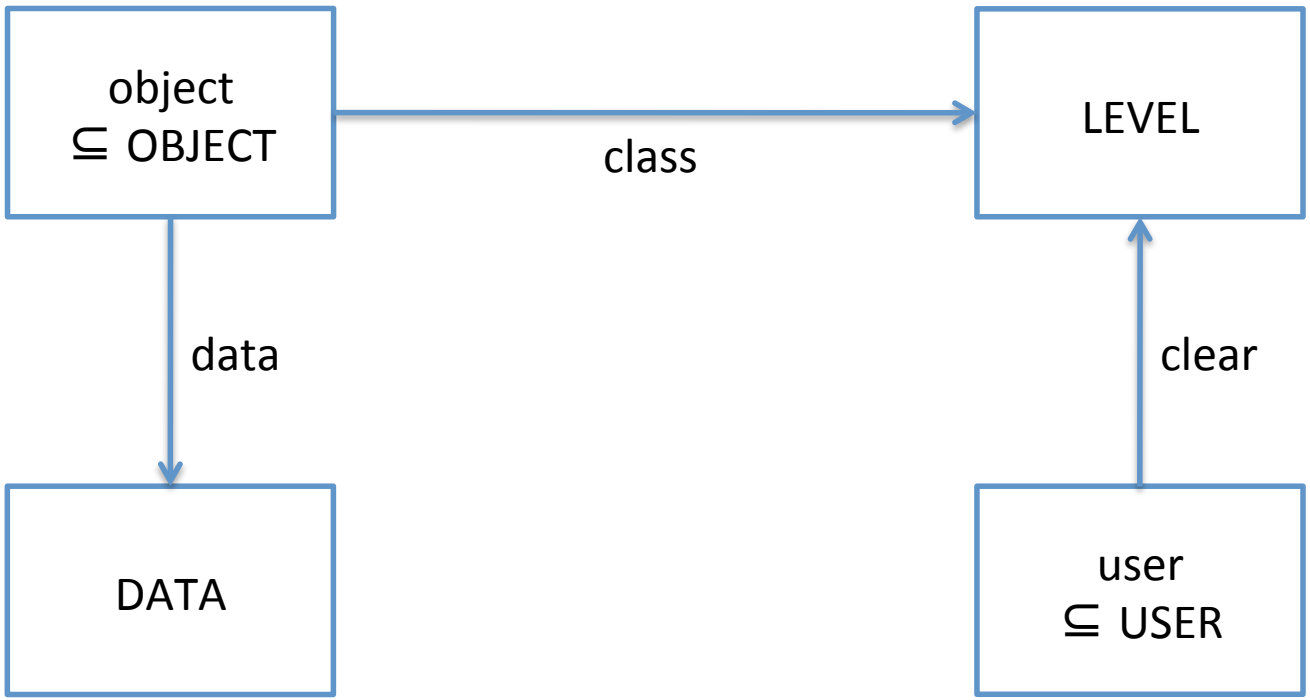
Users can only read and write objects whose classification is no greater than the user's clearance level.

What are the *entities*, *associations*, *events*?

Class diagram for secure database



Making variable set explicit



Types and variables

sets $OBJECT \quad DATA \quad USER$

constants $LEVEL$

axioms $LEVEL = 1..10$

variables $object,\ user,\ data,\ class,\ clear$

invariants

$$object \subseteq OBJECT$$
$$user \subseteq USER$$
$$data \in object \rightarrow DATA$$
$$class \in object \rightarrow LEVEL$$
$$clear \in user \rightarrow LEVEL$$

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

initialisation

$object := \{\} \quad user := \{\} \quad data := \{\} \quad class := \{\} \quad clear := \{\}$

Adding users

$$
\begin{array}{l}
\mathit{AddUser} \;\hat{=}\; \\
\quad \textbf{any}\ u, c\ \textbf{where} \\
\qquad u \in \mathit{USER} \\
\qquad u \notin \mathit{user} \\
\qquad c \in \mathit{LEVEL} \\
\quad \textbf{then} \\
\qquad \mathit{user} := \mathit{user} \cup \{u\} \\
\qquad \mathit{clear}(u) := c \\
\quad \textbf{end}
\end{array}
$$

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

Adding objects

$$
\begin{array}{l}
\mathit{AddObject} \;\; \hat{=} \\
\quad \textbf{any} \; o, d, c \; \textbf{where} \\
\qquad o \in \mathit{OBJECT} \\
\qquad o \notin \mathit{object} \\
\qquad d \in \mathit{DATA} \\
\qquad c \in \mathit{LEVEL} \\
\quad \textbf{then} \\
\qquad \mathit{object} := \mathit{object} \cup \{o\} \\
\qquad \mathit{data}(o) := d \\
\qquad \mathit{class}(o) := c \\
\quad \textbf{end}
\end{array}
$$

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

Reading objects

$Read \triangleq$

any $u, o, result$ **where**

$u \in user$	The user must exist
$o \in object$	The object must exist
$clear(u) \geq class(o)$	The clearance must be ok
$result = data(o)$	The data associated with the object

end

Writing objects

$$
\begin{array}{l}
\mathit{Write} \;\hat{=} \\
\quad \textbf{any}\ u, o, d\ \textbf{where} \\
\qquad u \in \mathit{user} \\
\qquad o \in \mathit{object} \\
\qquad \mathit{clear}(u) \geq \mathit{class}(o) \\
\quad \textbf{then} \\
\qquad \mathit{data}(o) := d \\
\quad \textbf{end}
\end{array}
$$

The write operation overwrites the data value associate with the object with a new value.

Changing classification and clearance levels

ChangeClass $\hat{=}$

any o, c **where**

$o \in object$

$c \in LEVEL$

then

$class(o) := c$

end

ChangeClear $\hat{=}$

any u, c **where**

$u \in user$

$c \in LEVEL$

then

$clear(u) := c$

end

Making classification changes more secure

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

$$
\begin{array}{l}
\mathit{ChangeClass} \;\hat{=}\; \\
\quad \textbf{any}\ o, c, u\ \textbf{where} \\
\qquad o \in \mathit{object} \\
\qquad c \in \mathit{LEVEL} \\
\qquad \mathit{clear}(u) \geq \mathit{class}(o) \\
\qquad \mathit{clear}(u) \geq c \\
\quad \textbf{then} \\
\qquad \mathit{class}(o) := c \\
\quad \textbf{end}
\end{array}
$$

Making clearance changes more secure

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

$$
\begin{array}{l}
\mathit{ChangeClear} \;\hat{=}\; \\
\quad \textbf{any}\ u, a, c\ \textbf{where} \\
\qquad u \in \mathit{user} \\
\qquad a \in \mathit{user} \\
\qquad \mathit{clear}(a) \geq \mathit{clear}(u) \\
\qquad \mathit{clear}(a) \geq c \\
\qquad c \in \mathit{LEVEL} \\
\quad \textbf{then} \\
\qquad \mathit{clear}(u) := c \\
\quad \textbf{end}
\end{array}
$$

Removing users and objects

RemoveUser $\triangleq$

any u **where**

$u \in user$

then

$user := user \setminus \{u\}$

$clear := \{u\} \lhd clear$

end

RemoveObject $\triangleq$

any o **where**

$o \in object$

then

$object := object \setminus \{o\}$

$class := \{o\} \lhd class$

$data := \{o\} \lhd data$

end