Refinement

© Michael Butler

University of Southampton

May 22, 2010

Abstraction

- Abstraction can be viewed as a process of simplifying our understanding of a system.
- The simplification should
  - focus on the intended purpose of the system
  - ignore details of how that purpose is achieved.
- The modeller needs to make judgements about what they believe to be the key features of the system.

Abstraction

- Abstraction can be viewed as a process of simplifying our understanding of a system.
- The simplification should
  - focus on the intended purpose of the system
  - ignore details of how that purpose is achieved.
- The modeller needs to make judgements about what they believe to be the key features of the system.
- If the purpose is to provide some service, then
  - model what a system does from the perspective of the service users
  - 'users' might be computing agents as well as humans.

Abstraction

- Abstraction can be viewed as a process of simplifying our understanding of a system.
- The simplification should
  - focus on the intended purpose of the system
  - ignore details of how that purpose is achieved.
- The modeller needs to make judgements about what they believe to be the key features of the system.
- If the purpose is to provide some service, then
  - model what a system does from the perspective of the service users
  - 'users' might be computing agents as well as humans.
- If the purpose is to control, monitor or protect some phenomenon, then
  - the abstraction should focus on those phenomenon
  - in what way should they be controlled or protected?
  - why should they be monitored?

Refinement

- **Refinement** is a process of enriching or modifying a model in order to
  - augment the functionality being modelled, or
  - explain how some purpose is achieved

Refinement

- **Refinement** is a process of enriching or modifying a model in order to
  - augment the functionality being modelled, or
  - explain how some purpose is achieved
- In a refinement step we refine one model $M1$ to another model $M2$:
  - $M2$ is a refinement of $M1$
  - $M1$ is an abstraction of $M2$

Refinement

- **Refinement** is a process of enriching or modifying a model in order to
  - augment the functionality being modelled, or
  - explain how some purpose is achieved
- In a refinement step we refine one model $M1$ to another model $M2$:
  - $M2$ is a refinement of $M1$
  - $M1$ is an abstraction of $M2$
- We can perform a series of refinement steps to produce a series of models $M1$, $M2$, $M3$, ...
- Facilitates abstraction: we can postpone treatment of some system features to later refinement steps

Refinement

- **Refinement** is a process of enriching or modifying a model in order to
  - augment the functionality being modelled, or
  - explain how some purpose is achieved
- In a refinement step we refine one model $M1$ to another model $M2$:
  - $M2$ is a refinement of $M1$
  - $M1$ is an abstraction of $M2$
- We can perform a series of refinement steps to produce a series of models $M1$, $M2$, $M3$, ...
- Facilitates abstraction: we can postpone treatment of some system features to later refinement steps
- Event-B provides a notion of consistency of a refinement:
  - We use proof to verify the consistency of a refinement step
  - Failing proof can help us identify inconsistencies in a refinement step

Refinement

- **Refinement** is a process of enriching or modifying a model in order to
  - augment the functionality being modelled, or
  - explain how some purpose is achieved
- In a refinement step we refine one model $M1$ to another model $M2$:
  - $M2$ is a refinement of $M1$
  - $M1$ is an abstraction of $M2$
- We can perform a series of refinement steps to produce a series of models $M1$, $M2$, $M3$, ...
- Facilitates abstraction: we can postpone treatment of some system features to later refinement steps
- Event-B provides a notion of consistency of a refinement:
  - We use proof to verify the consistency of a refinement step
  - Failing proof can help us identify inconsistencies in a refinement step
- Abstraction and refinement together should allow us to manage system complexity in the design process

Forms of Event-B Refinement

1. Extension:
   - Add variables and invariants
   - Extend existing events to act on additional variables
   - Add new events to act on additional variables

Forms of Event-B Refinement

1. Extension:
   - Add variables and invariants
   - Extend existing events to act on additional variables
   - Add new events to act on additional variables
2. Extension with Guard Modification:
   - Similar to model extension, except that we modify guards of existing events

Forms of Event-B Refinement

1. Extension:
   - Add variables and invariants
   - Extend existing events to act on additional variables
   - Add new events to act on additional variables
2. Extension with Guard Modification:
   - Similar to model extension, except that we modify guards of existing events
3. Variable Replacement / Data Reification:
   - Replace some variables with other variables, i.e., replace abstract variables with concrete variables
   - Modify existing events, add new events

Forms of Event-B Refinement

1. Extension:
   - Add variables and invariants
   - Extend existing events to act on additional variables
   - Add new events to act on additional variables
2. Extension with Guard Modification:
   - Similar to model extension, except that we modify guards of existing events
3. Variable Replacement / Data Reification:
   - Replace some variables with other variables, i.e., replace abstract variables with concrete variables
   - Modify existing events, add new events
4. Variable Removal:
   - Remove variables that have become redundant through earlier introduction of other variables.

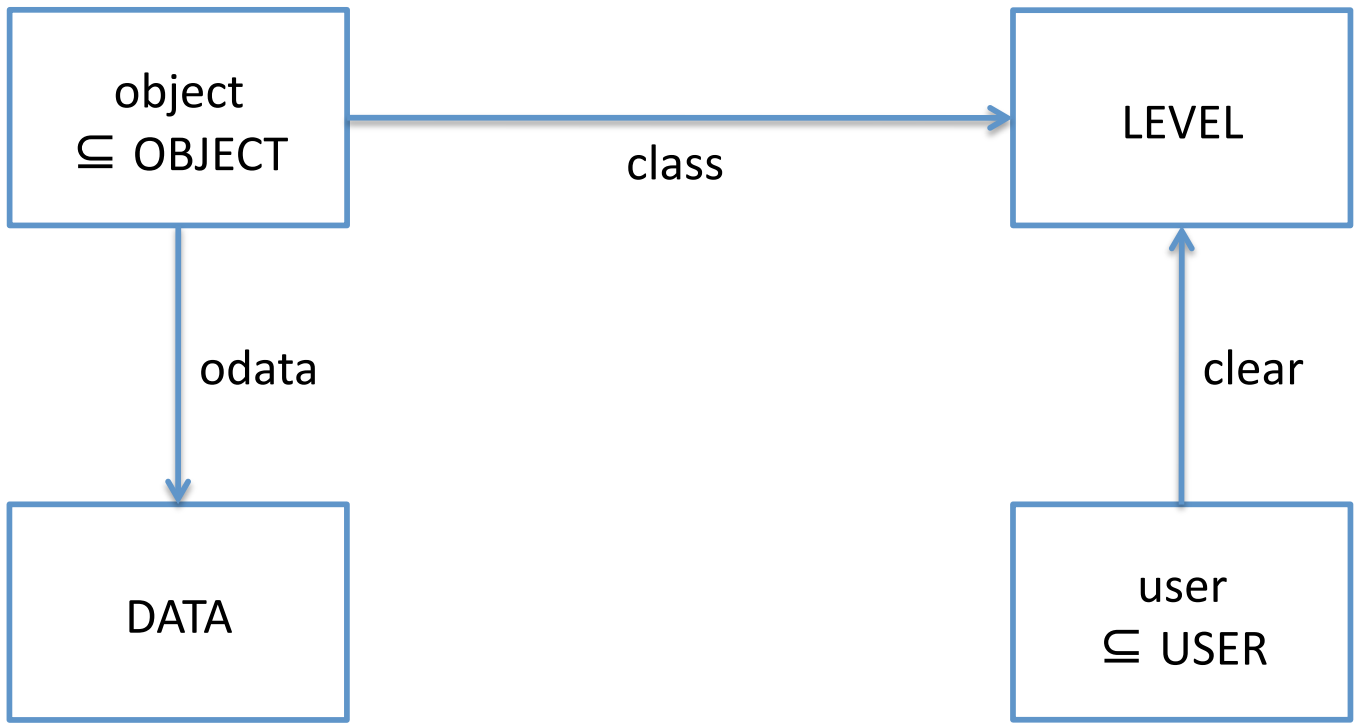
Forms of Event-B Refinement

1. Extension:
   - Add variables and invariants
   - Extend existing events to act on additional variables
   - Add new events to act on additional variables
2. Extension with Guard Modification:
   - Similar to model extension, except that we modify guards of existing events
3. Variable Replacement / Data Reification:
   - Replace some variables with other variables, i.e., replace abstract variables with concrete variables
   - Modify existing events, add new events
4. Variable Removal:
   - Remove variables that have become redundant through earlier introduction of other variables.

- Verification of 2, 3 and 4 requires gluing invariants that link abstract and concrete variables.

- Extension example: add ownership to secure database
- Extension with Guard Modification example: add tokens to access control system
- Variable replace example: replace disjoint sets by status function in building control system

Class diagram for secure database



Types and variables

sets $OBJECT \quad DATA \quad USER$
constants $LEVEL$
axioms $LEVEL = 1..10$

variables $object,\ user,\ odata,\ class,\ clear$
invariants

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

initialisation

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

Adding object ownership

Extend the database specification so that each object has an owner.

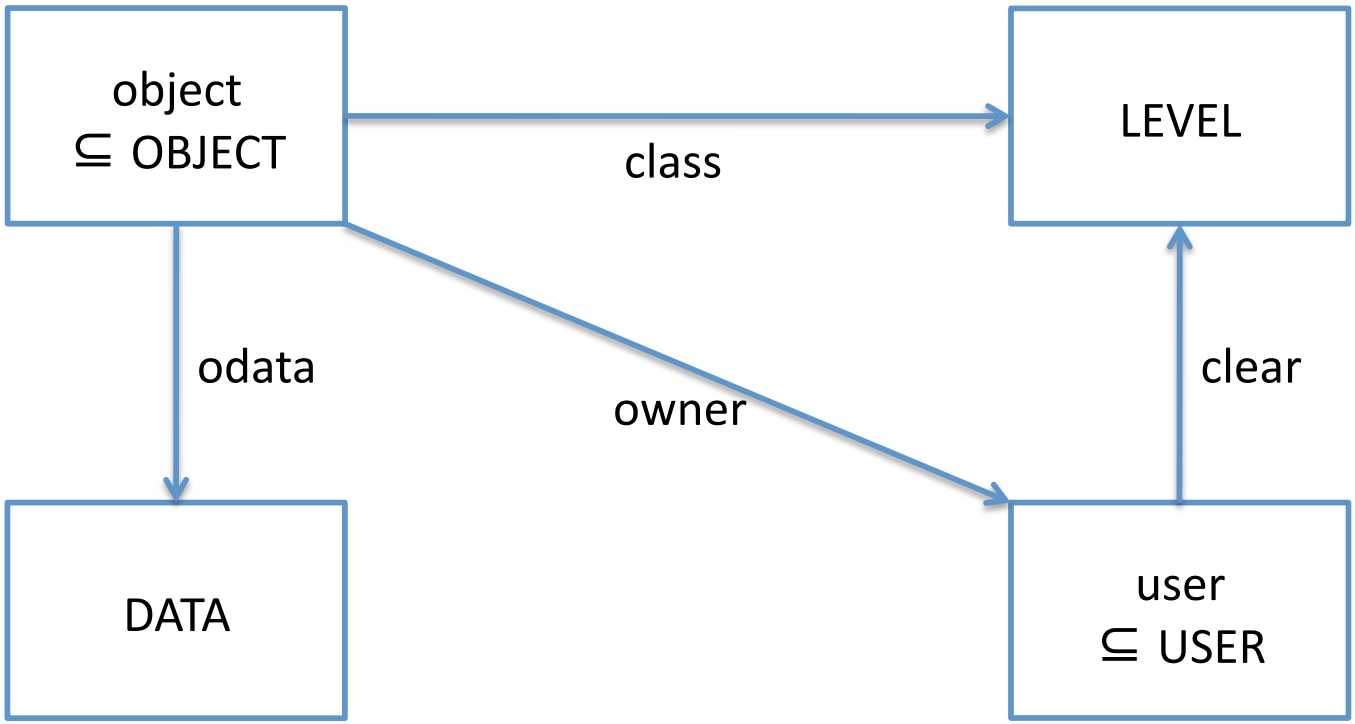
The clearance associated with that owner must be at least as high as the classification of the object.

Only the owner of an object is allowed to delete it.

What additional variables are required?

What events are affected?

Class diagram with ownership



Refinement

machine $m2$
refines $m1$
variables $object$, $user$, $odata$, $class$, $clear$, $owner$
invariants

$$owner \in object \rightarrow user$$

initialisation

$$\ldots \quad owner := \{\}$$

Adding users

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

Do we need to modify this?

Adding objects

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

Do we need to modify this?

Event Extension

$$
\begin{array}{l}
\mathit{AddObject}\ \ \textbf{extends}\ \ \mathit{AddObject}\ \ \hat{=} \\
\quad \textbf{any}\ u\ \textbf{where} \\
\quad\quad u \in \mathit{user} \\
\quad\quad \mathit{clear}(u) \geq \mathit{class}(o) \\
\quad \textbf{then} \\
\quad\quad \mathit{owner}(o) := u \\
\quad \textbf{end}
\end{array}
$$

Equivalent to

$$
\begin{array}{l}
\mathit{AddObject}\ \ \textbf{refines}\ \ \mathit{AddObject}\ \ \hat{=} \\
\quad \textbf{any}\ o, d, c, u\ \textbf{where} \\
\qquad o \in \mathit{OBJECT} \\
\qquad o \notin \mathit{object} \\
\qquad d \in \mathit{DATA} \\
\qquad c \in \mathit{LEVEL} \\
\qquad u \in \mathit{user} \\
\qquad \mathit{clear}(u) \geq \mathit{class}(o) \\
\quad \textbf{then} \\
\qquad \mathit{object} := \mathit{object} \cup \{o\} \\
\qquad \mathit{odata}(o) := d \\
\qquad \mathit{class}(o) := c \\
\qquad \mathit{owner}(o) := u \\
\quad \textbf{end}
\end{array}
$$

Other events to consider

- *Read*
- *Write*
- *ChangeClass*
- *ChangeClear*
- *RemoveUser*, *RemoveObject*

Do we need new events?