Software Architecture

Lecture 06
Event-B

Néstor Cataño
Innopolis University

Spring 2016

Plan Driven Vs. Agile Software Development

Plan Driven Software Development

- Complete list of requirements up-front
- Changes in the requirements at any time aren't welcome

Plan Driven Development with Event-B

- Programs are correct-by-construction
- No need of testing software implementation

Event-B

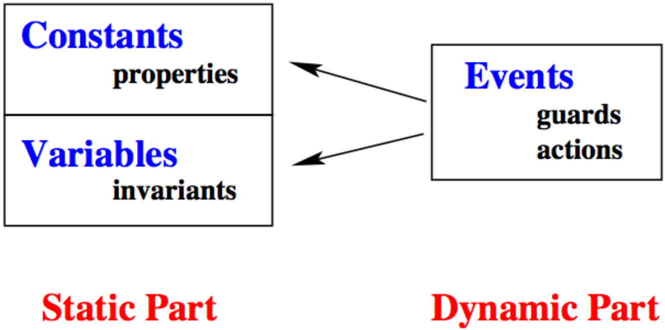
Notation

- modeling language: predicate logic and set theory
- models: state transition systems
  - states: constants and variables
  - transitions: events composed of a guard and actions
    - guard: logical predicate built on constant and variables
    - actions: describe how variables are modified

Event-B Components

- machines: variables, invariants, events
- contexts: constant, carrier sets

Static Vs. Dynamic



Events

```
event upload
any  parameters
where
  @grd  guard
then
  @act v := E(c,v)
end
```

parameters	set of variables
guard	predicate in predicate logic
actions	assignments changing the state of the machine
v	set of disjoint machine variables
E(c,v)	expression that depends on contants and variables

Contexts

```
context ctx
sets PERSON CONTENT
end
```

Machines

```
machine Facebook sees ctx

variables person content owner page

invariants
  @inv1 person ⊆ PERSON
  @inv2 content ⊆ CONTENT
  @inv3 owner ∈ content ↠ person
  @inv4 page ∈ content ⤀ person

event upload
any c p
where
  @grd1 p ∈ person
  @grd2 c ∈ CONTENT \ content
then
  @act1 content := content ∪ c
  @act2 owner := owner ∪ {c ↦ p}
  @act3 pages := pages ∪ {c ↦ p}
end
```

Operational Interpretation

machines

- when some event guards hold, exactly one of the events executes and the state is modified accordingly
  - subsequently, all the guards are checked again
- when no guards hold, then the model execution stops (the machine is deadlocked)

determinism vs. non-determinism

- if only one guard holds at some time then the model is deterministic
- machines exhibit external non-determinism as several guard can be true at the same time

Operational Interpretation

events

- an event describes an observable transition of the state variables
- an event is considered to take no execution time (atomic)
- no two events can occur simultaneously

termination

- termination is not is not mandatory, and indeed most machines run forever

The Event-B Method

Which Kinds of Systems?

- Complex: file transfer protocol, airline seat booking system, operating system, SmarCard electronic purse, vehicle flight controller, Android application
- Discrete: state transition systems, which can be executed by different entities concurrently.
- Reactive: react in response to the context or environment, e.g. a light bulb with a switch operated by a human

Correct by Construction

Claimed Difficulties

1. you have to be a mathematician
2. the proposed formalism is hard to master
3. the proposed formalism is not visual enough (boxes and arrows are missing)
4. people won't be able to conduct proofs

Correct by Construction

Actual Difficulties

1. you have to think a lot before coding
2. modeling is not the same as programming
3. modeling has to be accompanied by reasoning
4. the model of a program should also contain consistency proofs
5. frequent lack of a good requirements document

What is Correct by Construction?

Benefits

1. You do not need to test your implementation, yet you may want to animate it (is this contradictory?)
2. You do not need to test your implementation, yet you may want to test your requirements (is this contradictory?)

Development in Mature Engineering Disciplines

Blueprints

- represent a future system
- give insight on some but not all the aspects of a system: you cannot drive the blueprint of a car
- are organised into hierarchies in which more detailed blueprints are related to less detailed blueprints of the same thing
- allow us to prove that the system enjoys some desirable properties.

Software Development in Event-B

The Parachute Strategy

1. one starts writing an abstract model (blueprint) of the system in Event-B
2. the modeled system goes through a series of stages, named refinements (blueprints), each of which adds more details to the system
3. each refinement is provably consistent with the previous one (consistency proof obligations are discharged)
4. code is generated for the last refinement

But, ... Before Creating the Blueprints

Requirements Document (RD)

- most of the time is missing or poorly written
- does not consider many aspects of the system

RD for the Event-B Method

- separation between explanations (the **prose**) and requirements
  - reader is not acquainted with the system: explanations are more important
  - reader is acquainted with the system: requirements are more important

Example: Facebook

Brief and Vague System Description

A social network web-site features personal information (e.g. gender, birthday, family situation), media content (personal photos and videos), and a continuously stream of activity logged from actions taken on the site (such as messages sent, status updated, games played).
The privacy and security of this information is a significant concern, e.g. users may upload media they wish to share with specific friends, but do not wish to be widely distributed to their network as a whole.

Requirements Document

Notation

FUN	requirements about functionality of the system
SEC	requirements about privacy and security
INV	invariant properties

More Notation

- prose in blue
- requirements in red

Requirements Document

formalisation of the state of the system

FUN-1: $\boxed{\text{The social network shall have users and data (content)}}$

Requirements Document

a user that uploads content to her page owns that content

FUN-2: $\boxed{\text{The user who uploads data shall be classified as the owner of that data (content)}}$

Requirements Document

users can publish content on their web pages

FUN-3: $\boxed{\text{The users of the social network shall upload data}}$

Requirements Document

users must have a controlled access to information

SEC-1: $\boxed{\text{The users shall have controlled access to the data on the network based on permissions}}$

Requirements Document

users must be able to view or edit data

SEC-2:	The following permissions (privileges) over a given data may be given to a user : *i*. the permission to view the data *ii*. the permission to edit the data

Requirements Document

What's Missing?

What well-structuredness properties can be added to the previous SEC and FUN definitions?

Requirements Document

there is hierarchy of permissions between view and edit content permissions

INV-1: $\boxed{\text{Users that can edit data must also be able to view it}}$

Requirements Document

definition of being the owner of some data

INV-2: $\boxed{\text{The owner of some data has all the permissions on it}}$

Requirements Document

users must be able to make their data visible or invisible

FUN-4: $\boxed{\text{Users might choose what data available to them is viewed by them}}$

Requirements Document

FUN-1: The social network shall have users and data (content)

FUN-2: The user who uploads data shall be classified as the owner of the data

FUN-3: The users of the social network shall upload data

SEC-1: The users shall have controlled access to the data on the network based on permissions

SEC-2: The following permissions (privileges) over a given data may be given to a user:
i. the permission to view the data
ii. the permission to edit the data

INV-1: Users that can edit data must also be able to view it

INV-2: The owner of some data has all the permissions on it

FUN-4: Users might choose what data available to them is viewed by them

The Parachute Refinement Strategy

Basic Model

Model	What does the model see?	Requirements
Abstract	users, content, owner	FUN-1, FUN-2, FUN-3
Refinement 1	permissions	SEC-1, SEC-2, FUN-4 INV-1, INV-2

Additional Refinements

Model	What does the model see?
Refinement 2	friendship relations (best-friends, social friends, acquaintances)
Refinement 3	the wall
Refinement 4	the chatting room

Event-B Models

Event-B models are composed of two parts, contexts and machines.

Context

- constants, uninterpreted sets, and their properties (axioms)

Machine

- variables, invariants, state transitions (called events).
- an initialisation event computes the initial state of the machine.

RD – Abstract Machine (static part)

FUN-1: $\boxed{\text{The social network shall have users and data (content)}}$

RD – Abstract Machine (static part)

FUN-1: $\boxed{\text{The social network shall have users and data (content)}}$

sets PERSON CONTENT

variables person content

invariants

@inv1 person $\subseteq$ PERSON

@inv2 content $\subseteq$ CONTENT

RD – Abstract Machine (static part)

FUN-2: The user who uploads data shall be classified as the owner of that data

RD – Abstract Machine (static part)

FUN-2: | The user who uploads data shall be classified as the owner of that data |

variables owner

invariants

@inv3 owner $\in$ content $\twoheadrightarrow$ person

RD – Abstract Machine (static part)

FUN-3: $\boxed{\text{The users of the social network shall upload data}}$

RD – Abstract Machine (static part)

FUN-3: $\boxed{\text{The users of the social network shall upload data}}$

variables page

invariants

@inv4 page $\in$ content $\twoheadleftrightarrow$ person

RD – Abstract Machine (dynamic part)

FUN-3: $\boxed{\text{The users of the social network shall upload data}}$

RD – Abstract Machine (dynamic part)

FUN-3: The users of the social network shall upload data

event upload
any c p
where
 @grd1 $p \in person$
 @grd2 $c \in CONTENT \setminus content$
then
 @act1 $content := content \cup \{c\}$
 @act2 $owner := owner \cup \{c \mapsto p\}$
 @act3 $pages := pages \cup \{c \mapsto p\}$
end

Proof Obligations - Correctness Conditions

Which INV correctness condition is generated?
Which property should **@act2** guarantee?
hint: look at **@inv3**

invariants
 @inv1 person $\subseteq$ PERSON
 @inv2 content $\subseteq$ CONTENT
 @inv3 owner $\in$ content $\twoheadrightarrow$ person

event upload
any c p
where
 @grd1 p $\in$ person
 @grd2 c $\in$ CONTENT $\setminus$ content
then
 @act1 content := content $\cup$ c
 @act2 owner := owner $\cup$ $\{$c $\mapsto$ p$\}$
 @act3 pages := pages $\cup$ $\{$c $\mapsto$ p$\}$
end

Proof Obligations - Invariant Preservation

Invariant Preservation

$Inv(c, v)$, G(c,v) $\vdash$ $I(c, E(c, v))$	INV

Proof Obligations - Invariant Preservation

Invariant Preservation

$Inv(c, v)$, G(c,v) $\vdash$ $I(c, E(c, v))$	INV

Invariant Preservation

$$
\begin{array}{l}
\text{owner} \in \text{content} \twoheadrightarrow \text{person} \\
\text{p} \in \text{person} \\
\text{c} \in \text{CONTENT} \setminus \text{content} \\
\vdash \\
\text{owner} \cup \{\text{c} \mapsto \text{p}\} \in (\text{content} \cup \{\text{c}\}) \twoheadrightarrow \text{person}
\end{array}
$$

Invariant Preservation

$$\boxed{\begin{array}{l} \mathsf{owner} \in \mathsf{content} \twoheadrightarrow \mathsf{person} \\ \mathsf{p} \in \mathsf{person} \\ \mathsf{c} \in \mathsf{CONTENT} \setminus \mathsf{content} \\ \vdash \\ \mathsf{owner} \cup \{\mathsf{c} \mapsto \mathsf{p}\} \in (\mathsf{content} \cup \{\mathsf{c}\}) \twoheadrightarrow \mathsf{person} \end{array}}$$

Invariant Preservation

$$
\begin{array}{|l|}
\hline
\text{owner} \in \text{content} \twoheadrightarrow \text{person} \\
\text{p} \in \text{person} \\
\text{c} \in \text{CONTENT} \setminus \text{content} \\
\vdash \\
\text{owner} \cup \{\text{c} \mapsto \text{p}\} \in (\text{content} \cup \{\text{c}\}) \twoheadrightarrow \text{person} \\
\hline
\end{array}
$$

$\text{owner} \cup \{\text{c} \mapsto \text{p}\} \in (\text{content} \cup \{\text{c}\}) \twoheadrightarrow \text{person}$

- is function because owner was initially a function and c is not in the domain of owner
- is total because it was initially total and it sets an image for the new element of the domain.
- is surjective because owner was surjective and the pre-image of the p is c

Invariant Preservation

What if ....

What if **@grd2** $c \in CONTENT \setminus content$ is not added as a pre-condition of event upload?

The Complete Context

```
context ctx
sets PERSON CONTENT
end
```

The Complete Abstract Machine

```
machine Facebook sees ctx

variables person content owner page

invariants
 @inv1 person ⊆ PERSON
 @inv2 content ⊆ CONTENT
 @inv3 owner ∈ content ↠ person
 @inv4 page ∈ content ⇹ person

event upload
any c p
where
 @grd1 p ∈ person
 @grd2 c ∈ CONTENT \ content
then
 @act1 content := content ∪ c
 @act2 owner := owner ∪ {c ↦ p}
 @act3 pages := pages ∪ {c ↦ p}
end
```

The Parachute Refinement Strategy

Basic Model

Model	**What does the model see?**	**Requirements**
Abstract	users, content, owner	FUN-1, FUN-2, FUN-3
Refinement 1	permissions	SEC-1, SEC-2, FUN-4 INV-1, INV-2

RD – Refinement 1 (static part)

SEC-1: The users shall have controlled access to the data on the network based on permissions

SEC-2: The following permissions (privileges) over a given data may be given to a user:
i. the permission to view the data
ii. the permission to edit the data

RD – Refinement 1 (static part)

SEC-1: The users shall have controlled access to the data on the network based on permissions

SEC-2: The following permissions (privileges) over a given data may be given to a user:
i. the permission to view the data
ii. the permission to edit the data

variables viewp editp

invariants
@invr1 $viewp \in content \leftrightarrow persons$
@invr2 $editp \in content \leftrightarrow persons$

RD – Refinement 1 (static part)

INV-1: $\boxed{\text{Users that can edit data must also be able to view it}}$

RD – Refinement 1 (static part)

INV-1: $\boxed{\text{Users that can edit data must also be able to view it}}$

variables viewp editp

invariants
 @invr3 editp $\subseteq$ viewp

RD – Refinement 1 (static part)

INV-2: $\boxed{\text{The owner of some data has all the permissions on it}}$

RD – Refinement 1 (static part)

INV-2: $\boxed{\text{The owner of some data has all the permissions on it}}$

invariants

@invr4 owner $\subseteq$ viewp

@invr5 owner $\subseteq$ editp

RD – Refinement 1

Gluing Invariant

- it relates the state of the abstract machine with the state of the refinement machine
- it is stated as a (refinement) machine invariant
- it must be preserved by the (events of the) refinement machine, so proof obligations are generated and must be discharged

RD – Refinement 1

Refinement machine invariants

@invr1 viewp $\in$ content $\leftrightarrow$ persons
@invr2 editp $\in$ content $\leftrightarrow$ persons
@invr3 editp $\subseteq$ viewp
@invr4 owner $\subseteq$ viewp
@invr5 owner $\subseteq$ editp

RD – Refinement 1

Gluing invariants

@invr6 viewp $\subseteq$ page

RD – Refinement 1 (dynamic part)

FUN-4: $\boxed{\text{Users might choose what data available to them is viewed by them}}$

RD – Refinement 1 (dynamic part)

FUN-4: Users might choose what data available to them is viewed by them

event grant_view_permission
any p c
where
@grd1 $p \in persons$
@grd2 $c \in content$
@grd3 $c \mapsto p \notin viewp$
@grd4 **<GluingInvPres>**
then
@act1 $viewp := viewp \cup \{c \mapsto p\}$
end

RD – Refinement 1 (dynamic part)

FUN-4: Users might choose what data available to them is viewed by them

event grant_view_permission
any p c
where
 @grd1 $p \in persons$
 @grd2 $c \in content$
 @grd3 $c \mapsto p \notin viewp$
 @grd4 $c \mapsto p \in page$
then
 @act1 $viewp := viewp \cup \{c \mapsto p\}$
end

Granting Permissions

Proof Obligations

Pre-Condition	What is that needed for?
`@grd1` and `@grd2`	Typing of parameters
`@grd3`	Defensive specification style
`@grd4`	Preserving gluing invariant

RD – Refinement of Abstract Events

```
event upload extends upload
 then
  @act1r1 viewp := viewp ∪ {c ↦ p}
  @act2r1 editp := editp ∪ {c ↦ p}
end
```

The Complete Refinement Machine

```
machine permissions refines Facebook sees ctx

variables person content owner page viewp editp

invariants
 @invr1 viewp ∈ content ↔ persons
 @invr2 editp ∈ content ↔ persons
 @invr3 editp ⊆ viewp
 @invr4 owner ⊆ viewp
 @invr5 owner ⊆ editp
 @invr6 viewp ⊆ page

event upload extends upload
 then
  @act1r1 viewp := viewp ∪ {c ↦ p}
  @act2r1 editp := editp ∪ {c ↦ p}
end
```

Readings

Texts

- Modeling in Event-B: System and Software Engineering, by Jean-Raymond Abrial, ISBN 0521895561.

Papers and Docs

- Code Generation for Event-B, STTT, 2015, by Nestor Catano.
- Formal Methods: Theory Becoming Practice, by J.-R. Abrial.
- Comparison of Software Specification Methods Using a Case Study, b M. Yusufu and G. Yusufu.
- Rodin, User's Handbook, version 2.8, available at `handbook.event-b.org/current/pdf/rodin-doc.pdf`

The B Method

Tool Support

- Rodin platform, writing Event-B, discharging proofs
- EventB2Java Rodin plug-in for generating Java implementations of reactive systems