Checking consistency between architectural models using SPIN

Paola Inverardi, Henry Muccini, Patrizio Pelliccione
University of L’Aquila (Italy)
{inverard, muccini, pellicci}@univaq.it
**Objective:**
To validate Software Architectural models with respect to Requirements

**How to do this:**

1) defining a development process that explicitly identifies and manages coordination.
   [Coordination2000]

2) validating consistency among scenarios and statecharts...
   for instance

   validating SA models of dynamics (statecharts) with respect to the expected behaviors (scenarios)
Our approach to gain objective 1

- **Step 1**: Requirement Engineering + Coordination
- **Step 2**: drives
- **Step 3**: validates
- **Step 4**: drives

**The Unified Software Development Process**
Step 1: Identification and representation of Coordination Requirements

Requirements

- Use Case Diagram
- Analysis model
- Interaction Diagrams
- Activity Diagrams

SA + Coordination drives Specifications Coordination validates
In detail (2/4)

Step 2: From Requirements to Software Architectures

Requirements

- Analysis model
- Interaction Diagrams
- Activity Diagrams

Software Architecture

- Static view
- Dynamic view
- SA description
- LTS model

Diagram showing the relationships between requirements and software architecture, with arrows indicating 'drives' relationships.
In detail (3/4)

Step 3: Validating Software Architectures

Requirements

- Dynamic view
- Interaction Diagrams
- Activity Diagrams

Software Architecture

- Dynamic view
- LTS model
- Specifications Coordination

Drives and Validates relationships are shown in the diagram.
Is the SA model correct with respect to the Requirements? I.e., is the SA dynamics conform to the Coordination Requirements?
In detail (4/4)

Step 4: From SA to Coordination Models

Requirements and Software Architectures

Software Architecture  Coordination Models

**Static View**
- SA description
- LTS model

**Dynamic View**

- Drives
- Validates

Requirement Engineering + Coordination

IWIM Specification

Henry Muccini
Our approach to gain objective 2...
Validate statecharts with respect to the scenarios

Requirements and Software Architectures

Statecharts, LTS, Automaton

Process P

Statechart:
- a
- b
- c
- \(?ch1\)
- \(!ch2\)

Process Q

Statechart:
- x
- y
- \(!ch1\)
- \(?ch2\)

Promela Specification

LTL Formulae

Scenarios

UML Sequence, MSC, Scenarios


Step 1: State -> Promela

Component P

a

b

!ch1

?ch1

!ch2

c

Component Q

x

?ch2

!ch1

y

Statecharts

# define N 10
# define NC 2

chan channel [NC] = [0] of { bit };
typedef matrix {int pos[N]};
matrix ch [2*NC];
int position [2*NC];
int cont;

proctype P()
{<proctype description>}

proctype Q()
{s0:<state description>;goto si ;
...}

sn:<state description>;goto sj ;}

init
{atomic { run P(); run Q();}}

Henry Muccini
In detail (2/2)

Step 2: Scenario -> LTL Formula

P sends m1 before P sends m2 before Q receives m2 before Q receives m1
AND Send m1 is the first operation
AND Send m2 is the second operation
AND Receive m2 is the third operation
AND Receive m1 is the fourth operation

\[
\begin{align*}
& (\text{ch[ch1_s].pos}[0] < \text{ch[ch2_s].pos}[0] \\
& < \text{ch[ch2_r].pos}[0] < \text{ch[ch1_r].pos}[0]) \\
& \land (\text{ch[ch1_s].pos}[0] = 1) \\
& \land (\text{ch[ch2_s].pos}[0] = 2) \\
& \land (\text{ch[ch2_r].pos}[0] = 3) \\
& \land (\text{ch[ch1_r].pos}[0] = 4)
\end{align*}
\]

Scenarios
Integrating the approaches

Requirements

- Use Case Diagram
- Analysis model
- Interaction Diagrams

Software Architecture

- SA description
- LTS model
- static view
- dynamic view

Coordination Models

- IWIM Specification
- drives

LTL Formulae

Promela Spec.

Validates using SPIN

Henry Muccini
Applying the Approach

TRMCS Case Study
Requirements and Software Architectures

Analysis model

User -> AlarmInput -> AlarmRequest UI

sendAlarm: AlarmRequest UI -> AlarmHandler
receiveAlarm: AlarmHandler -> Server
write&read: Server -> UserDbase

AlarmInput: User -> AlarmRequest UI

sendCheck: CheckRequest UI -> CheckHandler
receiveCheck: CheckHandler -> Router
write&read: Router -> RouterDbase

CheckRequest UI: User -> CheckHandler

RouterDbase: Server -> Log File
UserDbase: User -> Log File

Server: AlarmHandler -> Error Handler

ErrorHandler: AlarmHandler -> Server
write&read: Server -> UserDbase

LTL Formula

Dynamics

User -> Alarm Handler -> Router -> Alarm Handler -> Server -> Check Handler

Alarm1: User -> Alarm Handler
Check: User -> Alarm Handler
Ack1: User -> Alarm Handler

Flow diagram with LTL Formula.
SA topology

User_i

sendCheck
sendAlarm
receiveAck
receiveAlarm

Check Coordinator

Timer Coordinator

Alarm Coordinator

Clock

Router

sendAlarm
receiveAlarm
sendAck
receiveAck

Server

SA dynamics

Promela

Requirements and Software Architectures

Henry Muccini
An architectural Error we found:

**Req:**
An User can send Alarms and Checks **whenever** he wants

**SA statechart:** An User can send a second check (Check2) only if the first check (Check1) as been forwarded to the Router Component
Requirements and Software Architectures

Ongoing and Future Works

✧ Tool Support

✧ Step1 Refinement (in [ConCoord’01])

✧ Enriched Statecharts and Scenarios

✧ Mapping

✧ Case Study

Henry Muccini
... and after your presentations...

✧ Use Case Diagrams   Vs.   Actors and Goals

✧ Our process   Vs.   Goal Oriented Req.
Henry Muccini
Ph-D Student in Computer Science
University of L’Aquila - Italy
muccini@univaq.it
http://www.dm.univaq.it/~muccini