Verification of UML models with timing constraints using IF

Susanne Graf
Verimag

http://www-if.imag.fr/
http://www-omega.imag.fr/

ARTIST summer school, Naesslingen, 30-09-2005
IST OMEGA: validation in the context of model-based development of real-time systems

Model (UML/XMI)

System
Environment
Requirements/assumptions

Behaviour
Time

Platform

Semantic models

Validation tools
System \models Requirements

Running implementation

Test

Update

Feedback

UML CASE tools

Feedback
The IF toolbox: approach

High-level programming and modeling notations (SDL, UML, SCADE, Java ...)

High-level semantics: structured notation, reduced number of general concepts (communication, coordination, time)

Low-level semantics: transition systems

Static analysis: model extraction, abstraction, ...

state explosion

simulation

verification1

test

verification2

verification3
IF tool-set: overview

- **Specifications**
  - Rational Rose, Rapsody, Argos, OMEGA
  - ObjectGeode
- **IF Exploration Engine**
  - Objecteering
  - RT
  - aml2if
  - uml2if
  - sdl2if
- **IF Static Analyzer**
  - TGV based TC generation
  - model construction
  - model checking observers µ-calculs
  - observer verification
  - guided simulation
- **Guided simulation**
  - LASH
  - RMC
  - SPIN
- **Min-cost path extraction**
  - TRex
  - LASH
  - RMC
  - SPIN

**Tools**
- Rational Rose
- Rapsody
- Argos
- OMEGA
- ObjectGeode
- Objecteering
- ObjectGeode
- Aldebaran
- SPIDER
- Aldebaran
Outline

- IF notation and tool-set (8)
- Omega Real-time profile (7)
- IFx: IF frontend for UML (5)
- Case studies (x)
System =
Set of **concurrent processes**
- **timed automata with urgency**
- hierarchical automata
- complex + abstract data types
- dynamic creation
- non-determinism

**Communication**
- asynchronous channels
- various routing / delay / loss models
- shared variables

**Execution control**
- dynamic priorities

**Assumptions and Requirements**
- observers (weak synchronization)

```
{ prio1 : x < y if x.t < y.t }
```
System description

Processes (components)
- Extended *hierarchical timed automata*
  (non-determinism, dynamic creation)

Data
- predefined data types
  (basic types, arrays, records)
- abstract data types

Interactions
- asynchronous channels
- shared variables

Execution control
- priority rules
- *resources* (mutex, preemption)
// processes
process P1(N1)
...
endprocess;
...
process P3(N3)
...
endprocess;

// signalroutes
signalroute sr1(1) ...
from P1 to P3 ;

// signals
signal s1(t1)
signal s2(t1, t2),
Process = hierarchical timed automaton

process P1(N1);
  fpar ... ;
  // types, variables, constants, procedures
state s0 ...
  ...  // transition t1
endstate;
state s1
  ...  // transitions t2, t3
endstate;
...  // states s2, s3, s4
endprocess;

IF: process description
transition = urgency + trigger + body

state s₀
...

urgency eager provided x! = 10;
when c₂ >= 4;
input update(m);
body ....
nextstate s₁;
...
endstate;

statement = data assignment
message sending,
process or signalroute creation or destruction, ...

IF: transitions
**IF: signal routes**

**signal route** = connector = process to process communication channel with **attributes**, can be **dynamically** created

- **route name**: signalroute
- **initial instance number**: s1(1)
- **attributes**: #unicast #lossy #fifo
- **signal set**
- **endpoints**

**attributes:**
- queuing policy: **fifo** | **multiset**
- reliability: **reliable** | **lossy**
- delivery policy: **peer** | **unicast** | **multicast**
- delay policy: **urgent** | **delay[l,u]** | **rate[l,u]**

**from server to client with** grant, fail;
priority order between process instances p1, p2
   ( free variables ranging over the active process set)

\[
\text{priority\_rule\_name: } p_1 < p_2 \text{ if condition}(p_1,p_2)
\]

- semantics: only maximal enabled processes can execute

- examples of scheduling policies
  - **fixed priority**: \( p_1 < p_2 \) if \( p_1 \) instanceof T and \( p_2 \) instanceof R
  - **EDF**: \( p_1 < p_2 \) if \( \text{Task}(p_2).\text{timer} < \text{Task}(p_1).\text{timer} \)
  - **run-to-completion**: \( p_1 < p_2 \) if \( p_2 = \text{manager}(0).\text{running} \)
**IF: observer for the expression of properties**

- **Observers** specify safety properties (assumptions and requirements)
- Event language acceptors: processes with specific triggers for monitoring events, system state, elapsed time
- 3 types of states: normal / error / success
- **Semantics:**
  - Transitions triggered by monitored events are executed with highest priority
  - Reaching a success state = reaching an uninteresting part (assumption)
IF tool-set: overview

Rational Rose, Rapsody, Argos, OMEGA

ObjectGeode

Rational Rose, Rapsody, Argos, OMEGA

Objecteering

aml2if

uml2if

sdl2if

Rational Rose, Rapsody, Argos, OMEGA

ObjectGeode

TGV based TC generation

ATS

model construction

model checking observers µ-calculs

observer verification

guided simulation

guided simulation

mincost path extraction

schedules

TReX

LASH

RMC

SPIN

IF Static Analyzer

IF Exploration Engine

IF Specifications

ARTIST summer school, Naesslingen, 30-09-2005
IF: core components

- Syntactic transformation tools:
  - static analyser
  - code generator

- Static analyser
- Code generator

- C/C++ code
  - Application specific process code
  - Predefined modules (time, channels, etc.)

- Compiler

- Interaction model
- Dynamic scheduling
- State space representation

- LTS exploration tools
  -- debugging
  -- model checking
  -- test generation
IF: exploration engine

asynchronous execution (max. concurrency)

dynamic scheduling (dynamic priorities)
state storage is completely done by the simulator

structural representation of configurations offering maximal sharing

unique tables implemented as hash tables with collision or search trees (splay trees or 2-3 trees)
IF: representation of time

Time represented by a dedicated process instance handling:
- dynamic clock allocation (set, reset)
- representation of clock valuations
- checking time constraints (time guards)
- computation of time progress conditions w.r.t. actual deadlines
- firing time progress transitions, if enabled

Two concrete implementations are available (others can be easily added)

i) discrete time
   - clock valuations represented as integer values
   - time progress by an explicit tick transition to the next deadline

ii) symbolic time
   - clock valuations represented by (varying size) difference bound matrices (DBMs)
   - time progress is implicit: State = state + time constraint
   - non convex time zones may arise due to urgency: represented implicitly by unions of DBMs
IF tool-set: overview

IF Static Analyzer

IF Specifications

IF Exploration Engine

Rational Rose, Rhapsody, Argos, OMEGA
ObjectGeode

Objecteering

aml2if
uml2if
sdl2if

TGV based
TC generation

model construction

model checking observers μ-calculs

observer verification

guided simulation

mincost path extraction

schedules

TReX
LASH
RMC
SPIN

ATS
AUT

SPIDER
Aldebaran

Verimag

ARTIST summer school, Naesslingen, 30-09-2005
**IF: Static analysis**

- **Approach**
  - source code transformations for model reduction
  - code optimization methods

- **Particular techniques implemented** so far
  - live variable analysis: remove dead variables and/or reset variables when useless in a control state
  - slicing: remove unreachable code, model elements w.r.t. a property, e.g. assumptions about the environment
  - variable abstraction: extract the relevant part after removing some variables
  - queue reduction: static analysis of queues

- **Result:** usually, *impressive state space reduction*
Outline

- IF notation and tool-set (8)
- Omega Real-time profile (7)
- IFx: IF frontend for UML (5)
- Case studies (x)
Omega UML profile: general features

Structure
• class diagrams distinguishing active and passive classes
• structuring concepts: inheritance, associations, compositions
• architecture and components (UML 2.0-like, not available in UML 1.4)

Behavior
• state machines with action language (compatible to UML1.4 A.S.)
• operations defined by methods (action body) → polymorphic
• concurrency: active/passive objects → activity groups
• interactions: primitive/triggered operations, asynchronous signals

Requirements and assumptions
• operational: observers, Live Sequence Charts
• declarative: OCL constraints on event histories

Timing constraints (in requirements, structure and design)
• declarative: timed events, linear (duration) constraints
• imperative: timers, clocks
Omega UML profile: interaction model & semantics

- active/passive objects define **activity groups**
- interactions: primitive/triggered operations, asynchronous signals

Omega UML profile: Time extensions

Compatible SPT profile and UML 2.0

- **Basics**
  - A notion of global time, *time progress non-deterministic, but controllable* by the model
  - Time primitive *types*: *Time, Duration* with operations
  - *Timed Events*: instants of occurrences of identified state changes in executions

- **Operational time access** *(UML 2.0)*
  - *time dependent behavior*
  - Mechanisms for measuring durations: *timers, clocks*
  - Corresponding actions: *set, reset,…*
Omega UML profile: Time extensions

- Time constraints
  - Constraints on durations between occurrences of events
    - OCL based
    - Patterns for constraining durations between occurrences of 2 events
    - SPT like derived patterns associated with syntactic entities
      - response time, duration of actions $\rightarrow$ deadline constraints,
      - duration in state, delay of channel, ...
  - Observers with time guards

- Scheduling
  - Resources accessed in mut. excl. and consuming execution time and actions for associating behavior with resources (deployment)
  - Execution time of actions
  - Dynamic priorities for expressing scheduling policies
Omega UML profile: requirements as observers

- special objects monitoring the system state / events
- example (Ariane-5): "If the Pyro1 object enters state "Ignition_done", then the Pyro2 object shall enter the state "Ignition_done" in not less than TimeConstants.MN_5*2 + Tpstot and not more than TimeConstants.MN_5*2 + Tpstar time units.

```
<<Observer>>

 liftoff_performed_right2

 g : Ground
 mc : MissionConstants
 tc : TimeConstants

<<Observer>>

wait_start

match send ::EADS::Signals::Start(void) by g / begin mc := g.Acyclic.MissionConstants; tc := g.Acyclic.TimeConstants end

wait_ignition_p1

[ g.Acyclic.EAP.Pyro1 @ Ignition_done ]

p1_ignited

[ now >= (tc.MN_5 * 2 + mc.Tpstar_prep) ]

ok

<<error>>

ko

choice

[ g.Acyclic.EAP.Pyro2 @ Ignition_done ]

[ now < (tc.MN_5*2 + mc.Tpstat_prep) ]

<<error>>

end state

observer stereotype

observer variables

event observation (see time profile)

state observation (variables + control states of reachable objects)

error state

(see time profile)
Omega UML profile: observables

- **observable events**
  - for signals: send, receive, accept
  - for operations: invoke, receive, accept, invokereturn, …
  - for states: entry, exit
  - for actions: start, end, start-end (for instantaneous actions)

- **observable state**
  - all entities reachable by navigation from already known entities (e.g., obtained from events)
  - can be stored in the observer

- **observing time**
  - use clocks local to an observer
  - read clocks of visible part of the model
Define explicit events and constraints

**example (Ariane-5)**: If the Pyro1 object enters state “Ignition_done”, then the Pyro2 object shall enter the state “Ignition_done” in not less than `TimeConstants.MN_5*2 + Tpstot` and not more than `TimeConstants.MN_5*2 + Tpstar` time units.
Outline

- IF notation and tool-set (8)
- Omega Real-time profile (7)
- IFx: IF frontend for UML (5)
- Case studies (x)
IFx: overview

- Rhapsody
- Rose
- Argo
- Objecteering

XMI 1.0/1.1 (UML 1.4 + stereotypes)

UML2IF

- XMI reader
- IF 2.0 translator

IF 2.0 TOOLBOX

UML 1.4 repository

UML 1.4 API

IF spec

IF spec
Mapping OO concepts to (extended) communicating automata

- **Structure**
  - class → process type
  - attributes & associations → variables
  - inheritance → replication of features
  - signals, basic data types → direct mapping

- **Behavior**
  - state machines (with restrictions) → IF hierarchical automata
  - action language → IF actions, automaton encoding
  - operations:
    - operation call/return → signal exchange
    - procedure activations → process creation
    - polymorphism → untyped PIDs
    - dynamic binding → destination object automaton determines the executed procedure

- **Observers and events: direct mapping**
IFx: example of mapping

BeverageDispenser

CoffeeUnit

TeaUnit

BeverageDispenser

CoffeeUnit::prepare

TeaUnit::prepare

ARTIST summer school, Naesslingen, 30-09-2005
IFx: global architecture

UML front-end

IF

XMI
UML model + time annotations

Rose, Rhapsody, Argo, ...

UML-IF frontend GUI

UML2IF translator + compliance checker

UML validation driver

IF model

IF tools

IFx: global architecture

ARTIST summer school, Naesslingen, 30-09-2005
IFx: simulation/verification interface

- User friendly simulation
  - rewind/replay
  - conditional breakpoints
  - customizable presentation of results for UML users
Outline

- IF notation and tool-set (8)
- Omega Real-time profile (7)
- IFx: IF frontend for UML (5)
- Case studies (x)
IFx: case studies

**Ariane-5 flight program** (together with EADS) – Rational Rose
- statically validate the well formedness of the model wrt the Omega profile,
- 9 safety properties of the flight regulation and configuration components,
- analyzed the schedulability of the cyclic / acyclic components under the assumption of fixed priority preemptive scheduling policy,
- safety properties concerning bus read/write access under this policy

**MARS bus monitor** (together with NLR) – I-Logix Rhapsody
- static validation
- proved 4 safety properties concerning the correctness of the MessageReceiver,
- discover reactivity limits of the MessageReceiver and to fine-tune its behavior in order to improve reactivity.

**Sensor Voting** (together with IAI) – Rational Rose
- static validation
- proved 4 safety properties concerning the timing of data acquiring by the three Sensors: end-to-end duration, duration between consecutive reads, etc.

**A depannage service** specification (done FT) – Rational Rose and IF
- showed service level timing properties
Ariane 5 flight program

Joint work with EADS SPACE Transportation

flight program specification

built by reverse engineering by EADS
high level, non-deterministic, abstracts
the whole program as a OMEGA UML model

23 classes, 27 runtime objects
~7000 lines of IF code

flight program requirements

General requirements
– no deadlock, no timelock
– no implicit signal consumption

Overall system requirements
– flight phase order
– stop sequence order

Local requirements of components
– activation signals arrive in some predefined time interval
Ariane 5: Model architecture

- **Equipment**: Valves, Pyros, Bus
- **Regulation**: Sequencer, EAP stage, EPC stage
- **GNC**: Thrust monitor, SRI, Attitude

Start(H0): openValve, ignitPyro

StartCyclic: requestEAPPrep, requestEAPRelease

23 classes, 29 run-time objects, 7000 LOC IF, 74 processes
Ariane 5: techniques applied

**translation**
- Mapping of complete UML specification into IF with **uml2if**
- fixed static errors (typing, naming)

**model generation**
- partial order reduction needed
- the full state space cannot be constructed
  - use some conservative abstractions

**model exploration**
- random or guided simulation
  - several inconsistencies found

**static analysis**
- live variable analysis
  - 20% of all variables are dead in each state

**model checking**
- 9 safety properties about the correct sequencing of sub-phases
  - concern only the acyclic part
  - abstraction of GNC part
- schedulability analysis
  - concerns the entire system
  - abstraction of mission duration
9 safety properties about the correct sequencing of sub-phases:

- between any two commands sent by the flight program to the valves there should elapse at least 50ms
- a valve should not receive signal Open while in state Open, nor signal Close while in state Closed.
- if some instance of class Valve fails to open (i.e. enters the state Failed Open) then
  - No instance of the Pyro class reaches the state Ignition done.
  - All instances of class Valve shall reach one of the states Failed Close or Close after at most 2 seconds since the initial valve failure.
  - The events EAP Preparation and EAP Release are never emitted.
- ...

...
Informal description

- If the liftoff is performed, the boosters shall be released at due time.

Formal description

- Using an observer
- Liftoff = pyro1.ignition
- Boosters release = pyro2.ignition
pre-emptive fixed priority scheduling

- one processor
- three tasks:

**Regulation**
- sporadic
- $E = 2-5\text{ms (func)}$
- priority: 0

**NC**
- periodic 72ms
- $E = 37-64\text{ms (f)}$
- priority: 1

**Guidance**
- periodic 576ms
- $E = \text{? ms}$
- priority: 2
why we cannot abstract functionality

Worst case: 64ms (72%)
Average: 42ms