The Omega project and IF simulation and verification tool for UML

Susanne Graf
Verimag

http://www-verimag.imag.fr/~async/IF/
http://www-omega.imag.fr/
Plan

- Projet Omega (8)
- Langage IF (12)
- IF outils de simulation et de validation (10)
- IF pour UML
  - Profile UML (6)
  - Mapping UML vers IF (4)
  - Interface utilisateur et études de cas (6)
Partners

Academic (tool and technology providers)

- **Verimag**, France – coordinator
- Christian-Albrechts University **Kiel**, Germany
- **CWI** (Centrum voor Wiskunde en Informatica), Netherlands
- University of **Nijmegen**, Netherlands
- **OFFIS**, Germany
- **Weizmann** Institute, Israel

**Users**

- **EADS** Launch Vehicles, France
- **France Telecom R&D**, France
- **Israeli Aircraft Industries**, Israel
- **NLR** (Nationaal Lucht- en Ruimtevaartlaboratorium), Netherlands

Supporters (UML tool providers)

- I-Logix --- Rational Software, IBM --- Telelogic
Model based development and validation of real-time systems

Model (UML)
- Structure (classes, components, …)
- Behaviour (state machines)

Requirements
- + time

System and environment
- architecture
- platform

Update

Code generation
- Test cases
- Running implementation
- System

Simulation
- Semantic models
- Validation tools

Update

System \models Requirements
How well does UML fit?

Strong points of UML

- Support of requirement level and design level notations, including architecture and components, which made their proofs
- User acceptance
- Existence of Case tools and model interchange format XMI
- Integration in development cycle possible

Weak points of UML (for validation of dynamics)

- Concepts are defined at syntax level, many issues in semantics are left to tools to fix them
- Weak support of real-time concepts (improved by UML 2.0)
1. Define a profile: select a suitable subset of UML
   - adapt and extend where needed with a special emphasis on timing
   - define an appropriate semantics

2. Propose a development methodology, based on
   - the user’s development methodology
   - the UML modeling and specification capabilities
   - verification methods and tools developed in the project

3. Provide methods and tools for formal specifications and verification covering the chosen profile
   - Model interchange via standard XMI

4. Proposed methodology and tool-set evaluated on four industrial case studies
Choice of a UML profile

- Fact: validation is only one aspect, we need a profile for modelling real-time embedded systems
  - Do not restrict the considered UML profile too much, so as to make it just match the concepts in the validation tools

- Fact: validation requires semantics
  - It is not enough to define keywords (tag values and stereotypes as in SPT or QoS profile) for defining a profile

- Fact: building validation tools is expensive
  - Reuse existing state-of-the-art methods and tools
  - Be open to many UML tools
    - standard exchange format (XMI) and standard UML extension mechanisms
  - Be open to different methodologies
  - Be open to a variation of semantics
Results

1. Omega UML profile for real time
   - A rich subset, useful for development
   - Notations for design and requirement specifications
     - **Kernel model:** close to operational subset of profiles of main tool providers with real-time in the spirit of SPT
     - Component and architecture description
     - Requirement notations of different nature:
       - Operational: Live sequence charts, Observers
       - Declarative: OCL
   - Positive feedback from users
   - Some concepts may influence standard evolution
     - Timed extensions
     - Observers for expression of requirements
     - Live sequence charts
   - Existence of formal semantics helpful for tool builders
Results: tools

**XMI**
Omega exchange format

**XML format (SXMI)**
Intermediate representation

**PVS based validation**
UML models and OCL with time in PVS
Infinite and parameterized models

**XML based execution**
Rule based tool for execution of XMI

**XML compliance check**

**Timed model-checking (IF)**
OMEGA models with time extensions and observers
Internal: timed automata with priority rules
Enumerative MC
UML oriented feedback
Visual representation of properties and error traces

**Untimed Model-Checking (UVE)**
OMEGA models with discrete time
Internal: symbolic transition relation
BDD based MC against LSC and temporal logic
Error traces = sequence charts

**LSC tools**
Extract class information from XMI
Editing of LSC with time (play-in)
Consistency of LSC
Export of LSC to XML
State machine synthesis (play-out)

**XML Representation of LSC**

**Extract class information from XMI**

**XML based execution**
Rule based tool for execution of XMI
Results: tools

- A set of tools covering all notations of the Omega profile
  - Different aspects of a model are checked by different tools, abstracting from other aspects
  - Different kinds of properties are checked
  - Problem: some variations on common parts
  - No other validation tools cover such a large profile

- Tool integration
  - Tool interchange by sharing models via an identified exchange format (XMI/XML)
  - Requirement: all tools agree on the common features
  - No heavy integration
IF simulation and verification tool for UML

VERIMAG
(Marius Bozga, Laurent Mounier, Susanne Graf, Joseph Sifakis, Iulian Ober)

- motivation
- the IF notation
- toolbox architecture
- the UML front-end IFx
- conclusions
The IF toolbox: objectives

Model-based development of real-time systems

Use of high level modeling and programming languages
• Expressivity for faithful and natural modeling
• Cover functional and extra-functional aspects
• Openness

Model-based validation
• Combine static analysis and model-based validation
• Integrate verification, testing, simulation and debugging

Applications:
Protocols, Embedded systems, Asynchronous circuits, Planning and scheduling
The IF toolbox: approach

Modeling and programming languages (SDL, UML, SCADE, Java …)

IF: Intermediate Format, based on a general and powerful semantic model

Static Analysis

Transition systems

Simulation

Verification 1

Verification 2

Verification 3

State explosion

Test
The IF toolbox: challenges

Find an adequate intermediate representation

*Expressiveness*: direct mapping of concepts and primitives of high modeling and programming languages
• asynchronous, synchronous, timed execution
• buffered interaction, shared memory, method call …

Use information about structure for efficient validation and traceability

*Semantic tuning*: when translating languages to express semantic variation points, such as time semantics, execution and interaction modes
overview

• introduction
• the IF notation
• the IF validation tools
• the
• UML front-end (IFx)
• conclusions
System description

Processes
- Extended hierarchical timed automata (non-determinism, dynamic creation)
- Predefined data types (basic types, arrays, records)
- Abstract data types

Interactions
- Asynchronous channels
- Shared variables

Data
- Predefined data types
- Abstract data types

Execution control
- Priority rules
- "Resources" for mutex constraints
System description

A set of interacting processes

• A process instance:
  – executes asynchronously with other instances
  – can be dynamically created
  – owns local data (public or private)
  – owns a private FIFO buffer

• Inter-process interactions:
  – asynchronous signal exchanges (directly or via signalroutes)
  – shared variables
System description

// 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

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

IF simulation and verification tool for UML
transition = \textit{urgency} + \textit{trigger} + \textit{body}

\begin{itemize}
  \item \textit{state} \texttt{s0}
  \item \textit{urgency}
  \item \texttt{provided} \texttt{x!=10;}
  \item \texttt{when} \texttt{c2 \geq 4;}
  \item \texttt{input} \texttt{update(m);}
  \item \textit{body}
  \item \texttt{nextstate} \texttt{s1;}
  \item \texttt{endstate;}
\end{itemize}

\texttt{statement} = \textit{data assignment}
\textit{message emission},
\textit{process or signal route creation or destruction}, …
signal route = connector = process to process communication channel with attributes, can be dynamically created

signalroute s1(1) #unicast #lossy #fifo

from server to client with grant, fail;

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

peer

unicast

multicast

to one specific instance
to a randomly chosen instance
to all instances
The model of time [timed automata]

- global time \(\rightarrow\) same clock speed in all processes
- time progress in stable states only \(\rightarrow\) transitions are instantaneous
Timed behavior

- **operations on clocks**
  - set to value
  - deactivate
  - read the value into a variable

- **timed guards**
  - comparison of a clock to a time constant (integer)
  - comparison of a difference of two clocks to a duration constant (integer)

```plaintext
state send;
output sdt(self,m,b) to {receiver}0;
set t:= 10;

nextstate wait_ack;
endstate;

state wait_ack;
input ack(sender,c);
...
when 10 < t < 20 ;
...
endstate;
```
Dynamic priorities

- priority order between process instances p1, p2 (free variables ranging over the active process set)

\[
\text{priority_rule_name : p1 < p2 if condition(p1,p2)}
\]

- semantics: only maximal enabled processes can execute

- scheduling policies
  - fixed priority: p1 < p2 if p1 instanceof T and p2 instanceof R
  - run-to-completion: p1 < p2 if p2 = manager(0).running
  - EDF: p1 < p2 if Task(p2).timer < Task(p1).timer
overview

• introduction
• the IF notation
• the IF validation tools
• the UML front-end (IFx)
• conclusions
IF simulation and verification tool for UML

**Description**

- **UML**
  - aml2if
- **RT/UML OMEGA**
  - uml2if
- **SDL**
  - sdl2if

**Exploration Platform**

- **Objecteering**
  - IF
  - Static Analyzer
- **Rational Rose**
  - IF
  - Description
- **ObjectGeode**
  - IF
  - Exploration Platform

**Static Analyzer**

- **TGV**
  - test generation
  - Test Suites
  - SPIDER
- **model construction**
  - model checking
  - guided simulation
  - mincost path extraction
  - schedules

**SPIDER**

- **LTS**
- **CADP**

**Tools**

- TReX
- LASH
- RMC
Core components

IF specifications
- parser
- writer

IF AST

Syntactic transformation tools:
- static analyser
- code generator

Compiler

C/C++ code
- application specific process code
- predefined modules (time, channels, etc.)

Interaction model

Dynamic scheduling

State space representation

LTS exploration tools
- debugging
- model checking
- test generation

IF simulation and verification tool for UML
Exploration platform

active instances

process 1
I₁:P₁

process 2
I₁:P₂
I₂:P₂

process j
I_k:P_j

Signal route
S2

Time module

execution control

interaction model

dynamic scheduling

Succ?
Succ!

IF simulation and verification tool for UML
Dealing with Time

Dedicated module
- including clock variables
- handling dynamic clock allocation (set, reset)
- checking timing constraints (timed guards)
- computing time progress conditions w.r.t. actual deadlines and
- fires timed transitions, if enabled

Two implementations for discrete and continuous time (others can be easily added)

i) discrete time
  - clock valuations represented as varying size integer vectors
  - time progress is explicit and computed w.r.t. the next enabled deadline

ii) continuous time
  - clock valuations represented using varying size difference bound matrices (DBMs)
  - time progress represented symbolically
  - non-convex time zones may arise because of deadlines: they are represented implicitly as unions of DBMs
Model-Based Validation
- static analysis
- model checking
- test generation
- optimization
Static analysis

• approach
  – source code transformations for model reduction
  – code optimization methods

• techniques implemented so far
  – live variable analysis: remove dead variables and/or reset variables when useless in a control state
  – dead-code elimination: remove unreachable code w.r.t. assumptions about the environment
  – variable abstraction: extract the relevant part after removing some variables
  – queue reduction: static analysis of queues

• usually, impressive state space reduction
Model-checking using observers

- **Observers** are used to specify safety properties in an operational way
- They are described as the processes – specific command for monitoring events, system state, elapsed time
- 3 types of states: normal / error / success
- **Semantics:** Transitions triggered by monitored events and executed with highest priority

```
match output SDT(void, b)
[b <> R(0).flag]

match input ACK(void)
[x <= t_ack]

match set x := 0
[b = R(0).flag]

match wait
[x >= t_ack]
```
Behavioral equivalence checking

• LTS comparison:
  – equivalence relations (“behavior equality”):
    System ≈ Requirements
  – preorder relations (“behavior inclusion”):
    System ≤ Requirements

• LTS minimization:
  – quotient w.r.t an equivalence relation:
    (System / ≈)

• CADP can be used to check the following relations:
  weak/strong bisimulation, branching, safety, trace equivalence
• User defined costs associated to transitions of IF descriptions e.g., execution times

• problem: find the min-cost execution path leading from the initial state to some goal state

• three algorithms implemented:
  – Dijkstra algorithm (best first)
  – A* algorithm (best first + estimation)
  – branch and bound (depth-first)

• applications:
  – job-shop scheduling (find the makespan)
  – asynchronous circuit analysis (find the maximal stabilization time)
UML frontend for IF

- architecture
- the Omega profile
- the mapping UML to IF
- examples
the UML front-end (IFx)
Omega real-time profile for real-time systems

A semantics has been formally defined for this subset and implemented in several tools

- Aim: Consistent validation results by different tools (hard to achieve, in fact there are some variations in the choices made in different tools)

Structure

- Class diagrams
  - Distinguishing active objects (mono-threaded processes) and passive objects (local data)
  - Private and public attributes and methods
  - Several associations with defined interpretation (inheritance, aggregation, navigation)

- Components define visibility constraints on the sub-components (UML 1.4 has no diagrams for components → little used case studies)
Omega real-time profile

- **Behavior**
  - **Local Behaviour**
    - State machines
    - Action language (concrete syntax, compatible to UML1.4 A.S.)
      - Object creation / destruction
      - Interactions (point-to-point): primitive/triggered operations, asynchronous signals
  - **Global behavior (requirements, functional specifications)**
    - OCL for the expression of state invariants and invariants of event histories
    - Live Sequence Charts = extension of sequence diagrams (outside UML)
    - Observers = state machines triggered by semantic level events (constructive description of a set of event sequences = property)
Interaction model : semantics

- active/passive objects define activity groups
- interactions: primitive/triggered operations, asynchronous signals
Omega real time profile: Timing

Compatible SPT profile and UML 2.0

- Basics
  - A notion of global time, *external* to the system
  - Time primitive types: Time, Duration with operations
    - **Timed Events**: sequence of instants of occurrences of identified state changes in each execution:
      - “SendSignal”, “ReceiveSignal”, “ConsumeSignal”
      - “InvokeMethod”, …
      - “EnterState”, “ExitState”
      - “StartAction”, “EndAction”
      - …

- Operational time access (as in UML 2.0): *time dependent behavior*
  - Mechanisms for measuring durations: timers, clocks
  - And corresponding actions: set, reset,…
Omega real time profile: Timing

- **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 constraints for the expression of (local and *global*) properties implying several events

- **Scheduling related**
  - **Resources** accessed in mut. excl. and consuming execution time
  - Execution time of actions
  - **Dynamic priorities** for expressing scheduling policies
Time 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” after the time TimeConstants.MN_5*2 + Tpstat_prep and before the time TimeConstants.MN_5*2 + Tpstar_prep.

Omega UML profile
Coverage of the Omega-UML profile

- fully OO models: classes with operations, attributes, associations, generalization, statecharts; basic data types; the Omega Action Language (compatible to UML 1.4 A.S.)
- UML observers for specifying requirements
- timing constraints

Tool connection

- XMI 1.0 or 1.1 for UML 1.4: Rational Rose, I-Logix Rhapsody and others.
• structure
  – UML object → IF process
  – attributes & associations → variables
  – inheritance: replication of structural features

• behavior
  – state machines, actions → syntactic translation (almost)
  – operations $X::m(x,y,\ldots)$
    ⇒ one IF process for every invocation of $X::m$
    process $X::m(x, y, \ldots)$
    – lives the period of activation, implements behavior
    – encapsulates the "stack frame" variables
    ⇒ predefined signals
    call$_{X::m}$, return$_{X::m}$

• flexible: adaptable to different call semantics (async, with futures…)

compilation of UML elements
compilation of method calls

caller

\text{call}_{X::m}(\ldots)

: X

create

: X::m(\ldots)

\text{return}_{X::m}(\ldots)
polymorphism, concurrency...

polymorphism $\Rightarrow$ dynamic binding resolved with signals
- the object state machine decides the method implementation
  which is executed in response to $\text{call}_{X:m}$

concurrency $\Rightarrow$ activity group management
- each active object has an associated group manager
- it handles/dispatches external calls for objects of the group
- keeps track of the running object

run-to-completion
- implemented with dynamic priority rules
  e.g. : $\forall x,y. (x.\text{manager} = y) \Rightarrow x < y$
the UML front-end (IFx)
the front-end GUI

- Wrapper of IF tools → (partly) hidden from the UML user
- Functionality:
  - interactive simulation / diagnostics analysis
  - scenario rewind / reply / load /saves
  - source tracing
  - conditional breakpoints
  - UML and other customized views (XSLT stylesheets)
  - batch tool launcher (compilers, verification)
IF simulation and verification tool for UML

front-end
Ariane-5 case study (EADS) – developed in Rational Rose
• statically validate the well formedness of the model wrt the Omega profile,
• proved 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,
• proved a safety property concerning bus read/write coherence under this policy

MARS case study (NLR) – developed in 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 an Monitoring case study (IAI) – developed in 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.
Ariane 5 model architecture

- **Equipment**: Valves, Pyros, Bus
- **Regulation**: Sequencer, EAP stage, EPC stage, start(H0)
- **Ground**: start(Cyclic)
- **GNC**: Thrust monitor, SRI, Attitude
  - requestEAPPrep, requestEAPRelease
  - 23 classes
  - 29 run-time objects
  - 7000 LOC IF
  - 74 processes

- openValve, ignitePyro
Conclusions

• IF is an unique platform:
  – relates functional and non-functional aspects:
    • distribution, communication, external C/C++ code, dynamic creation,
    • real-time, deadlines, resources, dynamic priorities
  – integrates state of the art validation techniques and tools
  – provides front-end to SDL/UML and related tools
    – http://www-verimag.imag.fr/~async/IF/

• one step further, component-based modeling and validation
  – combination of synchronous and asynchronous systems (e.g, Lustre/Esterel/StateCharts and SDL/UML)
  – composability of models and compositional reasoning