Experiences with the Omega tool set
in the context of
the MARS case study

Yuri Yushtein                  Jozef Hooman
Radboud University Nijmegen
Embedded Systems Institute,
Eindhoven
Report about experiences with main part of Omega tool set on industrial case study, Medium Altitude Reconnaissance System (MARS) of Dutch National Aerospace Laboratory (NLR)

Note:
- Not addressing all features of the tools,
- Not showing full power of tools
- Not always based on latest, current version of tool, but often experience with preliminary version during development within Omega

See other talks and demos for more details of tools
Overview

- Introduction MARS case study
- Relevant part Omega tool set, used on MARS:
  - LSCs (Weizmann)
  - Untimed Verification using UVE (OFFIS)
  - Timed Verification using IF (Verimag)
  - Interactive Verification using PVS (RUN, Weizmann, CAU)
- Redesign to facilitate compositional techniques and to investigate combined use of tools
- Summary
Purpose: counteract image quality degradation caused by forward motion of aircraft
Case study scope

DatabusManager

1 :ControllerMonitor
- currentStatusOK : int
- previousStatusOK : int

1 :DatabusController
- controllerStatus : int
+ controllerStatusOK():
  <<Actor>>

1 :AltitudeDataSource
- msgERROR : int
- msgCount : int
- timeoutCount : int
  AltMsgTimeoutCount : int
  AltMsgCount : int

1 :NavigationDataSource
- msgERROR : int
- msgCount : int
- timeoutCount : int
  NavMsgTimeoutCount : int
  NavMsgCount : int

1 :MessageReceiver
- AltMsgTimeoutCount : int
- NavMsgTimeoutCount : int
- NavMsgCount : int
- AltMsgCount : int
MARS environment constraints

- Data sources provide data with the 25 ms cycle and a jitter of ±5 ms
- The data sources are independent and are not synchronised
- The data messages may occasionally be lost due to the transmission errors
- The Data-bus Controller may exhibit built-in-test errors
MARS properties subject to verification

- Timely detection of Data-bus Controller error, based on built-in-test facility of controller, and proper recovery

- Timely detection of Data-bus error, based on data message arrival monitoring, and proper recovery
Relevant Part Omega Tool Set

- UML-based CASE tool
- Timed Omega Kernel
- XMI
- Untimed Model Checking UVE
- Timed Model Checking IF
- Interactive Verification PVS
- LSC Play-in Play-out
Scenario-based requirements modelling

Tool support

- Modelling with LSCs, PlayEngine tool
- Play-In facility – automated LSC capturing
- Play-Out facility – model simulation
- Model verification based on SMV model checker
- Property specification with existential or universal LSCs
Scenario-based requirements modelling (2)

- UML Sequence Diagram example (camera control)

ordering

atomic sequences

relating diagrams

timing, conditions, constraints, etc.
Scenario-based requirements modelling (3)

- **External data sources**

  - Pre-chart
  - Non-deterministic choice
  - Timing constraint

Diagram:
- **NavDataSource**
- **Tick**
- **NavSend = NDS_Time Mod ND_Cycle**
- **Time < NDS_Time + 5**
- True
- False
- True
- False
- False
Scenario-based requirements modelling (4)

- Data processing and transfer

![Diagram of data processing and transfer]

- Condition
- Forbidden element
Scenario-based requirements modelling (5)

- LSC representation of the properties
Scenario-based requirements modelling (6)

Conclusions

- Possibility to verify high-level (timed) requirements
- More effective for transaction-based systems
- Play-Out and verification for autonomous systems
- Play-Out – early system simulation
- Play-In effective for “human-in-the-loop” systems
- GUI Play-In is artificial for autonomous systems
- Play-In can be used to capture anti-scenarios
- No model export/connection to other tools
- GUI development for Play-In relies on VB
Untimed UML modelling and verification

Tool Support

- UML modelling with Rhapsody tool
- Verification with UML Verification Environment (UVE)
- UVE performs untimed model checking
- A system run is seen in terms of run-to-completion steps
- Property specification with propositional logic and temporal logic patterns
- Counterexamples are given as discrete-time timing diagrams and LSC traces
Untimed UML modelling and verification (3)

Class behaviour modelling

Statechart of the MessageReceiver

OMEGA Workshop - Grenoble, 17 February 2005
### Verification example

#### Property specification

<table>
<thead>
<tr>
<th>Specification of Properties</th>
<th>Assumptions</th>
<th>Non-deterministic External Event List</th>
<th>Verification Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>inv ( P \implies \text{finally } Q _B _immediately ) with ( P = \text{root} \rightarrow \text{Data bus Manager} \rightarrow \text{Data bus Controller} \rightarrow IS _IN(\text{Error}) ) ( Q = \text{root} \rightarrow \text{Data bus Manager} \rightarrow \text{Message Receiver} \rightarrow IS _IN(\text{Controller Error}) )</td>
<td>first ( P \implies \text{globally } Q _immediately ) with ( P = \text{root} \rightarrow \text{Data bus Manager} \rightarrow \text{Data bus Controller} \rightarrow IS _IN(\text{Error}) ) ( Q = \text{root} \rightarrow \text{Data bus Manager} \rightarrow \text{Data bus Controller} \rightarrow IS _IN(\text{Error}) )</td>
<td>evPolController(ENV, \text{root} \rightarrow \text{Data bus Manager} \rightarrow \text{Controller Monitor})</td>
<td>does not hold.</td>
</tr>
</tbody>
</table>

A counterexample trace is generated. Please standby...
** The Property 'inv_P implies finally Q B immediate' with

P = root->p_DatabusManager->itsDataBusController->IS_IN(Error)
Q = root->p_DatabusManager->itsMessageReceiver->IS_IN(ControlError)

under the assumptions 'first_P implies globally Q immediately' with

P = root->p_DatabusManager->itsDataBusController->IS_IN(Error)
Q = root->p_DatabusManager->itsDataBusController->IS_IN(Error)

and assumption 'inv finally P B immediate' with

P = ES_eventContoller(ENV, root->p_DatabusManager->itsControllerMonitor)
max_X_Val = 5

with 'ndet'-mode external event list

root->p_DatabusManager->itsDataBusController, evControllerBIT_OK
root->p_DatabusManager->itsDataBusController, evControllerBIT_ERROR
root->p_DatabusManager->itsControllerMonitor, evPollController
root->p_DatabusManager->itsNavigationDataSource, evSendMsg (0)
root->p_DatabusManager->itsNavigationDataSource, evSendMsg (1)
root->p_DatabusManager->itsAltitudeDataSource, evSendMsg (0)
root->p_DatabusManager->itsAltitudeDataSource, evSendMsg (1)
Results of the experiments

- Property violation found and corrected
- Several new features added to the tool, e.g. transient properties
- Several tool issues identified and corrected
Conclusions

- Verification of high-level models, or partial models of critical parts only
- Verification of small UML models, with non-deterministic environment stimuli
- Significant decrease in performance for complex properties
- LSC facility for transaction-oriented designs
- Discrete timing – run-to-completion step varies depending on circumstances
Timed UML modelling and verification

Tool support

- UML modelling with Rhapsody tool
- IFx tool reads UML models in XMI format
- Model simulation and verification
- Timed model checking
- Uses OMEGA UML time extensions
- Based on IF tool from VERIMAG
- Property specification based on observers
- Counterexample scenarios can be simulated
Timed UML modelling and verification (2)

Environment behaviour modelling

Statechart of the AltitudeDataSource, NavigationDataSource is identical
Observer specification example

```plaintext
/match invoke : Default: DatabusManager: startDEM() on itsDEM/

wait_cm

/match informal "cmReady"

wait_ok

[itsDEM.itsDatabusController.controllerStatus <> 0]

wait_error

[itsDEM.itsDatabusController.controllerStatus = 0]t.set(0)

bc_error

[t >= 10]t.reset()

[not (itsDEM.itsMessageReceiver @ ControllerError)]

error

<<error>>
```
Compositional observer class diagram example
Results of the experiments

- Property violation found and corrected
- Modelling issue with start-up synchronisation
- Performance bottleneck in tool implementation found and resolved
- New features introduced: informal actions, inter-observer communication
- Current shortcoming: inability to generate shortest counterexample
Conclusions

- Timed UML modelling and property specification
- Specification of timing non-determinism
- Timed verification of small UML models
- Handled model with up to 8 non-deterministic elements
- Time-bounded non-determinism is the foremost cause of verification complexity growth
Interactive Verification

Tool support

- **PVS**: general purpose interactive theorem prover, tool developed by SRI; free available

- **PVS specification language:**
  - higher-order typed logic
  - hierarchies of parameterized theories, with declarations, definitions, axioms, theorems
  - large amount of predefined theories

- **PVS proof engine can be used to prove theorems:**
  to prove goal, user invokes proof commands `proof`, includes powerful decision procedures and rewrite strategies
UML formalization in PVS

- UML model
- XMI representation
- PVS representation
- Semantics
- Preprocessing
- OCL property
- PVS property
- PVS proof engine
Desired (generalized) properties:
1) Receiver shall move to bus error location if and only if one of data sources misses $N$ consecutive signals
2) Receiver shall recover from error if and only if both data sources send $K$ consecutive signals

Implementation of NLR: $N = 3, K = 2$
Spec in PVS

Trace: sequence of events and time stamps

Prop1(tr) : bool = FORALL i, j : i < j AND
(LongTimeOut(d1, i, j)(tr) OR LongTimeOut(d2, i, j)(tr)) AND
NOT Error(d1, j)(tr) AND NOT Error(d2, j)(tr)
IMPLIES AfterWithin(err, j, DeltaErr)(tr)

Never(d2,i,j)(tr) AND
T(tr)(j) - T(tr)(i) >= K * C + 2 * J

EXISTS i: i >= j AND (err@i)(tr) AND
T(tr)(i) - T(tr)(j) <= DeltaErr
Verification of Message Receiver

Untimed version: absence of data as separate events
  - First basic verification in PVS, finding suitable invariants
  - Redone in TLPVS, reducing user interaction by strategies [UML’04]

Timed version: verification in PVS of simple case
  - Only sender and receiver \( N = 1 \) and \( K = 1 \)
  - One safety property proved (\( \sim 50 \) PVS lemmas)
  - Required relations between parameters identified:
    \[
    S < C/2, \\
    \max(P_N, P_A) < C - 2S, \\
    N*C + S < Tout < (N + 1)*C - S
    \]
Results of the experiments

- Combination of UML features, e.g. synchronous operation calls, asynchronous signals, threads, hierarchical state machines, makes interactive verification very complicated

- In PVS modularization of semantic definitions is important to allow use of minimal semantics for features needed

- For scalability, compositionality and abstraction are essential, but this often requires redesign
  ➔ see MARS example
To enable compositional verification on MARS: specify message receiver for single data source parameterized by events $d$, $ok$, and $err$
Message receiver for two data sources

Obtain original message receiver by composition

Note: allows generalization to more data sources
Declarative specifications of
- Message receiver for two data sources  TDS
- Message receiver for one source          MR
- Error Logic                             EL

To validate MR, specify various senders S and show in PVS:  S || MR  \rightarrow  desired properties

Correctness decomposition in PVS:
MR1 || MR2 || EL  \rightarrow  TDS

Next parts can be implemented and checked in isolation
Implementation Message Receiver

**Message Receiver** can be implemented by single state machine or by parallel composition of processes

For instance, $R \parallel M$ where

- **$R$** implemented by timed state machine (with time-out) suitable for timed model checking
- **$M$** implemented by untimed state machine (counts the number of messages needed to recover) suitable for untimed model checking

**Error Logic** is untimed; suitable for untimed model checking
Recovery Problem

Problem (found by Verimag):
*err* signal not good for recovery, because any data miss requires restart of counting

Note: condition for re-entering correct state is stronger than condition for staying

Solution 1: signals parameterized by bit string representing presence of last N data messages

Solution 2: additional *miss* signal to indicate miss of single data message
## Omega techniques on MARS

<table>
<thead>
<tr>
<th>MARS</th>
<th>Spec</th>
<th>Application to original model</th>
<th>Compositional approach</th>
</tr>
</thead>
<tbody>
<tr>
<td>UVE OFFIS</td>
<td>Patterns LSC Propositional formulas</td>
<td>Untimed verification of model with 2 senders</td>
<td>Modular verification of parts, using assume/promise approach, un timed</td>
</tr>
<tr>
<td>IF Verimag</td>
<td>Observers Abstract spec for simulations</td>
<td>Timed verification of model with 2 synchronized senders</td>
<td>Compositional approach using abstractions timed</td>
</tr>
<tr>
<td>PVS RUN Weizmann</td>
<td>Declarative, higher-order logic</td>
<td>Untimed verification, use of TLPVS Timed: small instance</td>
<td>Compositional verification, timed</td>
</tr>
<tr>
<td>LSC Weizmann</td>
<td>LSCs</td>
<td>Requirements modeling, untimed model checking, timing partly</td>
<td>Synthesis of parts from LSCs &amp; Generic specs</td>
</tr>
</tbody>
</table>
Concluding Remarks

- One of the main issues is scalability
- Also combinations of tools important; but
  - remodeling to meet different levels of abstraction
  - reformulation of specifications for different tools
  - additional burden for industrial development process

- Growing need for accessible industrial-grade
  UML-based formal verification tools:
  use of formal methods is expected to become mandatory
  in avionics development standards; but requires
  - better integration with commercial UML modelling tools
  - human-tool interaction on UML model level