CEA LIST- LISE

Cyber Physical Systems: Speeding up validation thanks to design

Christophe Gaston

CEA | July 2014

christophe.gaston@cea.fr
Lots of systems handle critical aspects of our lives

(engine cruise control in cars, train or plane automatic control)

In critical domains, norms are more and more constraining concerning the level of confidence to reach about those system behaviors

Formal methods are good candidates to reach such levels of confidence
However some limitations have to be addressed:

• Systems are often heterogeneous which may limit the scope of formal methods to be used or requires mixing the usage of several ones (e.g. pieces of software communicating on distributed architectures with sensors or piloting mechanical parts)

• The formal analysis may be explosive in time or space

• Assessing the correctness of a system may require to precisely control and observe its executions which may be complicated

*Model Driven Engineering (MDE) may help to overcome those limitations*
Some benefits of MDE:

**MDE permits to represent systems to be studied in a common modelling framework**

⇒ permits to get rid of heterogeneity in early phase analysis

**MDE enables abstract denotations of system behaviors**

⇒ helps in controlling space or time explosion problems in formal analysis

**Models may be analyzed for the purpose of simplifying some formal treatments**

(based on some modularization or preservation principles such as compositionality)
A focus on Model Based Testing (MBT)

System To Test (STT) is seen as an unknown set of executions (traces = sequences of inputs and outputs). You can discover them by experimenting.

Entry data

Test cases are generated from M

Observable reactions

Conformance relation = a mathematical relation between Traces of SUT and M

is an executable specification of the SUT (characterizes a set of traces)

Test cases are generated from M

Observable reactions

Conformance relation = a mathematical relation between Traces of SUT and M

System To Test (STT) is seen as an unknown set of executions (traces = sequences of inputs and outputs). You can discover them by experimenting.
Systems are often heterogeneous which may limit the scope of formal methods to be used or requires mixing the usage of several ones.

MBT not so much impacted by this limitation at SUTs level

(SUTs are black boxes)

MBT can be impacted at the level of models:

System heterogeneity impacts the choice of modelling languages (i.e. UML for software parts, MATLAB SIMULINK for parts involving environment modelling, SDL for communication parts…):

⇒ may result on system models consisting of collections of heterogeneous models with low coupling

(defined by different teams and with no consistency analysis)

However by raising the level of abstraction it is possible to define models getting rid heterogeneity (e.g. using sequence diagrams models)
A sequence diagram model:

Each process $p_1..p_3$ is seen as a black (or grey) box.

The model focuses on intended interaction between sub-systems and not on how they are realized.
The formal analysis may be explosive in time or space

MBT processes include automatic test generation phases

In order to ensure high level of model coverage, test generation generally involves:
• solving techniques to analyze the data part of models (in our case *symbolic execution*)
• graph exploration techniques for the control part

Both kinds of techniques are explosive

*To limit explosion we propose to use collections of sequence diagram models (defining different scenarios or modes of usage) that can be explored and analyzed separately at the test generation steps*
Assessing the correctness of a system may require to precisely control and observe its executions which may happen to be complicated.

In the MBT domain this corresponds to two classical and well identified problems that occur at the test execution phase, namely:

**Observability** and **Controllability**
Observability: it may be hard to build an efficient testing environment because traces corresponding to real executions of STT may be hard to identify:

- e.g. when STT is distributed it may be hard or even impossible to find an occurrence order between atomic communication actions (as soon as they occur on distributed interfaces)
- e.g. because it would require to “instrument” STT (some actions are not observable)

Controllability: an efficient testing process should permit to “cover” (i.e. test) behaviors selected in the model $M$ and this may be complex because $M$ and/or STT may be non-deterministic:

- e.g. because we want to abstract away from implementation details in $M$ (about some computations for example)
- e.g. because STT is distributed / parallel / concurrent
**Compositionality:**

If it is hard to assess the correctness of:

\[ \text{Sub System 1} \quad \text{Sub System 2} \]

let us try to assess the correctness of:

\[ \text{Sub System 1} \quad \text{and the correctness of:} \quad \text{Sub System 2} \]

and take benefits from the following compositionality result:

\[ \text{If } X \quad \text{and } Y \quad \text{are correct then } X \cdot Y \quad \text{is correct} \]

By partitioning the system into more observable and controllable subsystems, one simplifies the problem of assessing the correctness of the system.
Additionnal benefits of compositionality:

Debugging is eased: when a failure is observed while executing a big system, it may be hard to identify the sub sub sub... sub system that caused the failure. Testing subsystems per subsystems simplifies the problem.

Economical issues: waiting until the entire system is implemented to test it may be economically harmful (early design decisions are questioned too late). Thanks to compositionality each subsystem can be tested as soon as it exists.
1. UML MARTE sequence diagram models
2. TIOSTS: Timed Input Output Symbolic Transition Systems
3. Operational semantics of UML MARTE sequence diagrams
4. Testing framework and algorithm
5. Focus on compositional testing
6. Related works
7. Conclusion and perspectives
Messages and lifelines:

Vertical dotted lines are *Lifelines*

They are associated to ports

They represent successions of instants from the point of view of the port *(going down means going to the future)*

*One may associate events to instants: for example emissions or receptions of values*

*Values transit through messages symbolized by horizontal arrows*

*Instants on different lifelines cannot be compared, except that the causality between the reception and the emission of a message induce a partial order*
**loop operator:**

The loop operator is used to specify iterations.

As instants on different lifelines cannot be compared, the number of iterations on each lifeline occurring in a loop is not necessarily the same at given absolute time.
The alt operator is used to specify a choice (here between behaviors specified respectively in region o1 and region o2)
The strict operator is used to specify synchronization points between executions on lifeline

Here no lifeline may switch from o1 to o2 unless all lifelines completed executions in o1
Constraints on data and time:

\( p_1 < 10 : \)
the value carried by
message \( m_1 \) and
received on \( p_1 \) should
be less than 10

\( \{ p_1 < 10 \} \)

\( @t_1 \)

\( @t_2 \)

\( \{ t_2[i] - t_1[i] < 3 \} \)

t1 and t2 are *time stamps*: they can be understood as tables storing instants.

Each time they occur with the prefix @, the current value of time is added at the last place of the table.

The constraint \( t_2[i] - t_1[i] < 3 \) specifies that whenever a value goes through \( m_2 \), its takes
less than 3 unit of time to go from \( p_2 \) to \( p_3 \).
Pour personnaliser « nom événement et auteur » :

"Insertion / En-tête et pied de page"

Personnaliser la zone de pied de page

Cliquer sur appliquer partout
1. UML MARTE sequence diagram models

2. TIOSTS: Timed Input Output Symbolic Transition Systems

3. Operational semantics of UML MARTE sequence diagrams

4. Testing framework and algorithm

5. Focus on compositional testing

6. Related works

7. Conclusion and perspectives
**Data variables** are used to symbolically reason on system states and to perform computations.

**Time variables** can be declared to store instants (time is passing by itself) and symbolically reason about them.

One may condition the execution of a transition by properties on time called *time guards*, and properties on data called *data guards*.

The execution of a transition can be associated to a communication action which can be an input (of the form c?x) or an output (c!t).

Data variables can be updated by means of substitution (permits to denote state evolutions).

- **Data variables**
  - set of time variables
  - time guard
  - data guard
  - Communication action
  - Substitution on data variables

- **Time guard**
  - \( t[i] - t[i-1] < 0.3 \)

- **Data guard**
  - \( y < 500 \)

- **Communication action**
  - speed ! y

- **Substitution on data variables**
  - \( i \leftarrow i + 1 \)
1. UML MARTE sequence diagram models

2. TIOSTS: Timed Input Output Symbolic Transition Systems

3. Operational semantics of UML MARTE sequence diagrams

4. Testing framework and algorithm

5. Focus on compositional testing

6. Related works

7. Conclusion and perspectives
In order to test SUT against sequence diagrams we need to associate them with formal semantics.

The approach consists in applying translation schemas by associating TIOSTS to messages and lifelines.

Operator that covers several lifelines (alt, loop…) are taken into account at the lifeline level in the translation.
Semantics of sequence diagrams

```
ctrl:Controller
  intensity:Integer

calc:Calculator
  intensity:Integer

loop

@t₁ — m₂ — @t₂

<<TimedConstraint>>
{ t₂[i] - t₁[i] < 0.1 }

q

f_{m₂} ← empty()
i_{t₂} ← 0

{t₁}
ctrl.intensity?x_{m₂}
f_{m₂} ← push(f_{m₂}, x_{m₂})

q'

{t₂}
t₂[i_{t₂}] - t₁[i_{t₂}] < 0.1
f_{m₂} ≠ emptyQueue
calc.intensity!top(f_{m₂})
f_{m₂} ← pop(f_{m₂})
i_{t₂} ← i_{t₂} + 1
```
Semantics of sequence diagrams

```
ctrl:Controller
intensity:Integer

loop

@t₁

<<TimedConstraint>>
{ t₁[i] - t₁[i-1] = 0.5 }

m₂

q

\{t₁\}
t₁[i₁] - t₁[i₁-1] = 0.5

ctrl.intensity!x_{ctrl.intensity}
i₁₁ ← i₁₁ + 1
```
1. UML MARTE sequence diagram models
2. TIOSTS: Timed Input Output Symbolic Transition Systems
3. Operational semantics of UML MARTE sequence diagrams
4. Testing framework and algorithm
5. Focus on compositional testing
6. Related works
7. Conclusion and perspectives
Interaction Models

\[ C_1 \quad C_2 \]

Implemented in UML sequence diagrams (+ UML MARTE profile timing constraints)

conforms to ?
ioco [krichen2006]

Symbolic behavior computation

Verdict computation

Diversity

\[
\begin{align*}
\text{delay}_1 + \\
\text{delay}_2 + \text{delay}_3 & \leq 6 \\
x_1 & < 10 \\
x_2 & < 10 \\
y & \geq 8
\end{align*}
\]

Temporized trace of the system execution

Temporized sequence of inputs

Log

Verdict: correct
- Transition with symbolic data and time

\[
q \rightarrow q' \quad \begin{array}{c}
\text{clock} < 3 \\
x > 6 \\
\{\text{clock}\} \\
\text{c!x} \\
x := x + 1
\end{array}
\]

- Symbolic execution of a TIOSTS transition

\[
\eta \quad \begin{array}{c}
q \\
\text{pc}_t^0 \\
\text{pc}_d^0 \\
T_0 \\
x := x_0 \\
\text{clock} := \text{clock}_0
\end{array} \quad \xrightarrow{\text{delta}_0 \cdot \text{c!x}_0} \quad \eta'
\]

\[
\eta' \quad \begin{array}{c}
q' \\
\text{pc}_t^1 = \text{pc}_t^0 \land \text{clock}_0 + \text{delta}_0 < 3 \\
\text{pc}_d^1 = \text{pc}_d^0 \land x_0 > 6 \\
T_0 + \text{delta}_0 \\
x := x_0 + 1 \\
\text{clock} := 0
\end{array}
\]

\text{symbolic snapshot}

\text{delta}_0 \quad \text{new fresh variable for the delay of the transition}
Testing framework and algorithm

\[
\begin{align*}
\text{Init} & : (q_0, \theta_0, \pi_0, \theta_0, \lambda_0) \\
& \downarrow z_0 \cdot \text{location } ? \text{loc}_1 \\
\eta_1 & : (q_1, \theta_0, \pi_0, \theta_1, \lambda_1) \\
& \downarrow z_1 \cdot \text{plan } ? \text{ask } F \rho_0 \\
\eta_2 & : (q_2, \theta_1, \pi_0, \theta_2, \lambda_2) \\
& \downarrow \text{error } ! \text{f Timeout}_0 \\
& \downarrow z_3 \cdot \text{plan } ? \text{f Plan}_1 \\
\eta_3 & : (q_0, \theta_2, \pi_0, \theta_3, \lambda_3) \\
\eta_4 & : (q_3, \theta_3, \pi_0, \theta_4, \lambda_4) \\
& \downarrow z_4 \cdot \text{location } ? \text{loc}_2 \\
\eta_5 & : (q_1, \theta_2, \pi_0, \theta_5, \lambda_5) \\
\eta_6 & : (q_4, \theta_4, \pi_0, \theta_6, \lambda_6) \\
& \downarrow \text{error } ! \text{c Param } s_0 \\
& \downarrow z_6 \cdot \text{notif } ! \text{c Param } s_0 \\
\eta_7 & : (q_5, \theta_5, \pi_0, \theta_7, \lambda_7) \\
\eta_8 & : (q_5, \theta_6, \pi_0, \theta_8, \lambda_8) \\
& \downarrow z_7 \cdot \text{param } ? \text{d Param } s_1 \\
& \downarrow z_8 \cdot \text{calc } ! \text{data}_1 \\
\eta_9 & : (q_6, \theta_7, \pi_0, \theta_9, \lambda_9) \\
& \downarrow z_9 \cdot \text{error } ! \text{c Timeout}_0 \\
& \downarrow z_{10} \cdot \text{calc } ? \text{c md } s_1 \\
\eta_{10} & : (q_0, \theta_10, \pi_0, \theta_10, \lambda_{10}) \\
\eta_{11} & : (q_7, \theta_9, \pi_0, \theta_{11}, \lambda_{11}) \\
& \downarrow z_{11} \cdot \text{c md } ! \text{c md } s_1 \\
\eta_{12} & : (q_0, \theta_{10}, \pi_0, \theta_{12}, \lambda_{12})
\end{align*}
\]
1. UML MARTE sequence diagram models
2. TIOSTS: Timed Input Output Symbolic Transition Systems
3. Operational semantics of UML MARTE sequence diagrams
4. Testing framework and algorithm
5. Focus on compositional testing
6. Related works
7. Conclusion and perspectives
Focus on compositional testing

Compute symbolic behaviors

Compositional techniques

Extraction of unitary behaviors thanks to projection techniques

Validation thanks to a compositional result

**Result**: if each component (sub-system) conforms to its deduced behaviors then the system as a whole conforms to the sequence diagram [Bannour 2011]
Focus on compositional testing

(1) Chose the subsystem on which to project

(2) Hide observable actions (input/output) at the interface of other components

(3) Transform observable action of the system if it corresponds to an input at the interface of the component
Focus on compositional testing

Project FSF

Those techniques have been used on a system controlling doors opening and closing in the railway context (on simulation platform)

-> an error due to the latency of communication was identified
Some related projects

FSF, **Funding**: IRT program, **Key partners**: Alstom, LRI lab (Orsay University), **Key subject**: Timed distributed testing for safety critical systems

SESAMGRID, **Funding**: BGLE Project (french national program), **Key partners**: Cofely Ineo GDF Suez, MAS lab (Ecole Centrale Paris), **Key subject**: Design methodology for smart grids including test in the loop process to handle security issues

OpenES, **Funding**: CATRENE, **Key partners**: STMicroElectronics, Thales, NXP, Synopsis, **Key subject**: Design and validation for electronics systems
1. UML MARTE sequence diagram models
2. TIOSTS: Timed Input Output Symbolic Transition Systems
3. Operational semantics of UML MARTE sequence diagrams
4. Testing framework and algorithm
5. Focus on compositional testing
6. Related works
7. Conclusion and perspectives
Related works

- [Kruger1998], [Harel2000], [Harel2005], [Whittle2000] and [Alur2003] **generation of automata from basic MSC** – no timing aspects

- [Larsen2010] and [Zhu2010] **timed framework but for verification purpose**

- [Dinh-Trong2006] and [Roychoudhury2011] **use symbolic execution for testing from system scenarios** as MSC (Message Sequence Chart) – no compositionality study

- [Kruger2004] **derivation of models of the components to be designed/chosen using projection from MSC specifications** – not symbolic – not timing aspects
Related works


- GASTON C., HIERONS R., LE GALL P., “Model based testing of timed distributed systems”, ICTSS 2013, LNCS

1. UML MARTE sequence diagram models
2. TIOSTS: Timed Input Output Symbolic Transition Systems
3. Operational semantics of UML MARTE sequence diagrams
4. Testing framework and algorithm
5. Focus on compositional testing
6. Related works
7. Conclusion and perspectives
We proposed an compositional approach for testing from sequence diagram models denoting system models

Implementation in tools: Papyrus modeler + Diversity tool

Such an approach permits to ease model based testing by:

• using abstract system models (minimizing explosion of analysis) allowing one to partially get rid of heterogeneity of systems

• (almost…) reducing system testing process to subsystem testing thanks to compositionality

The approach can be used to test implementations or any executable subsystem models (it can be a basic technology to test consistency between heterogeneous models of different subsystems)
Distributed testing: extend the approach to the distributed conformance relation dtioco [Gaston - Hierons - Le Gall - ICTSS 2013]

Explore how to incorporate projection and testing in a complete design methodology (refinement driven by projection – ongoing work: activity diagram models refining sequence diagrams)