Evaluating Context Descriptions and Property Definition Patterns for Software Formal Validation

Philippe Dhaussy[1], Pierre-Yves Pillain[1], Stephen Creff[1], Amine Raji[1],

Yves Le Traon[2], Benoit Baudry[3]

1 : Lab. LISyC-DTN, ENSIETA, BREST

2 : Université du Luxembourg

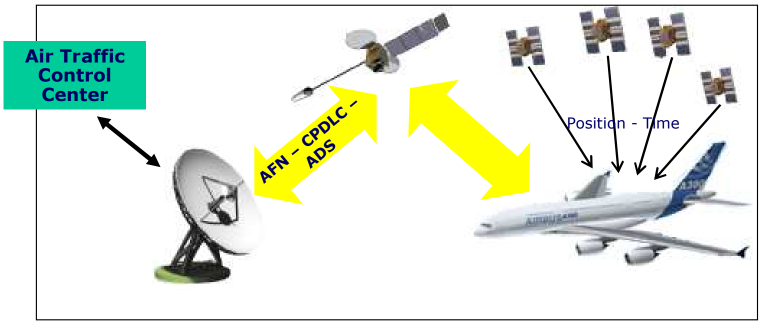
3 : Equipe Triskell, RENNES

Our research group is involved
in the experimentation of formal technologies
for validation of software models of embedded systems.

The general idea of our work is :

to study the hypothesis and the operational conditions
to make easier the integration of formal methods
in the engineering processes.

ATC (Air Traffic Control) needs Aircraft-Ground communications
ATC Data-Link applications



Work is in progress
First experience feedback with our industrial partners

This work is in progress and

we present here a first experience feedback

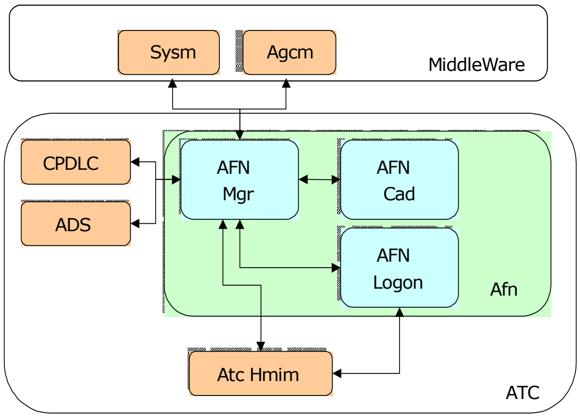
with our industrial partners.

Our partners are Airbus, Thales, Cnes, the French spatial agency,
CS-SI, this company works for Airbus, And DGA, french ministry of defense.

This is an example of a case study in the Air traffic Control applications.
We focus on a part of this application,
a software component of ATC Data-Link protocol

The AFN Model (2/2)

- AFN (Aircraft Facilities Notification) is an ATC Data-Link application
  - Allows pilot to send to ATC Center its identifier (Logon)
  - Then, ground may use other applications
  - Can change control from one ATC Center to another (Contact ADvisory procedure - CAD)



AFN User Model : SDL Model
3 processes
13400 lines (96 states, 107 transitions)

AFN (Aircraft Facilities Notification) is a part of ATC Data-Link application

AFN allows pilot to send to ATC Center its identifier (Logon)

Then, Ground may use other applications

Can change control from one ATC Center to another

AFN is modeled with SDL (Specification and Description Language)

It is composed of 3 processes (an hundred states and transitions)

Corresponding to thirteen thousand and 4 hundred lines of SDL

Evaluating Context Descriptions and Property Definition Patterns

- **Motivation**
- **Proposition**
- **Context and requirement modeling (CDL)**
- **Experimentation : toolset (OBP) and results**
- **Discussion and future work**

In this talk :

First of all, I present our motivation and proposition.

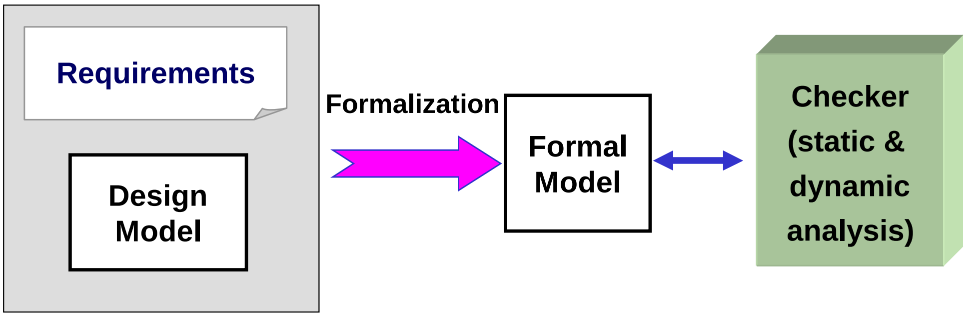
After, I show how we formalize, with a prototype language, contexts and properties

Then, I present some experimentations and some results.

I finish with a conclusion and future work.

Motivation : Integration of formal methods in the engineering processes

Many barriers



Difficulty to formalize the requirements

Semantic gap between

- the system to be validated
- the formal model needed for the validation.

Many barriers exist for an effective use of formal methods.

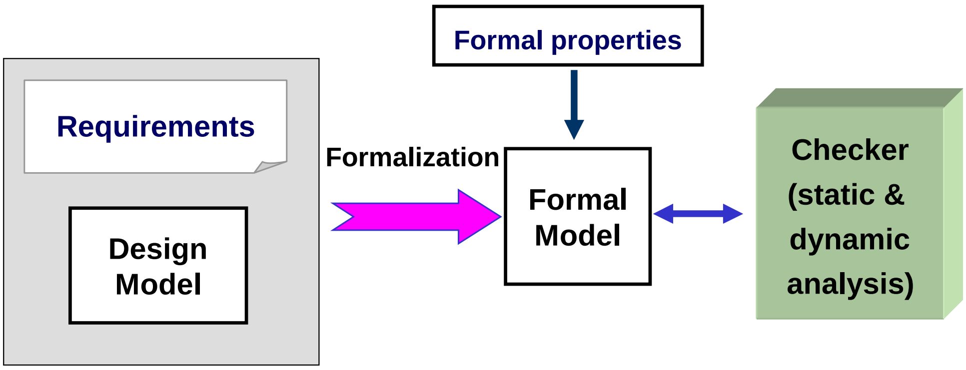
One of these barriers is the difficulty to formalize the requirements

because the semantic gap between the system to be validated

and the formal model needed for the validation.

Barrier : Formal property expression

Temporal logic formula (as LTL or CTL) : not acceptable in industry.



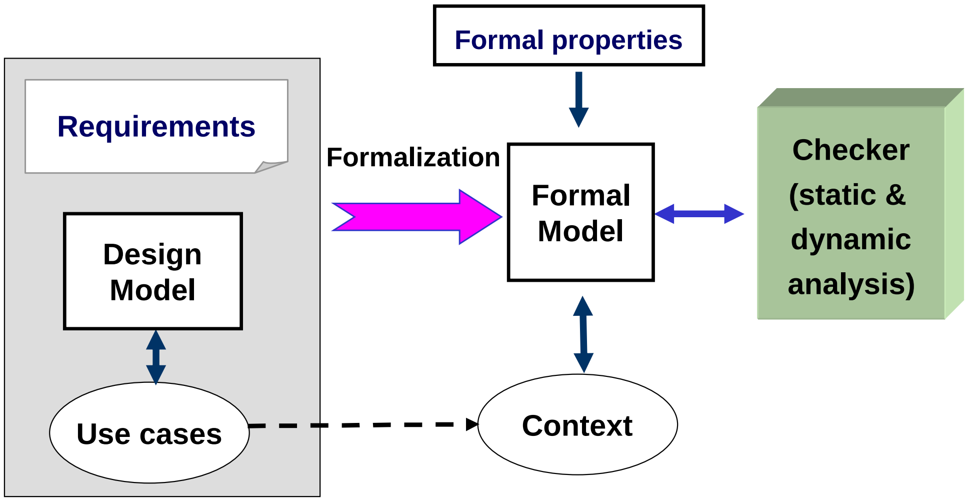
High expressiveness, but not easily readable
Not easy to handle by the engineers in industrial projects.

Property expression with temporal logic formula (as LTL or CTL) is not acceptable in industry.

These languages have a high expressiveness

but they are not easily readable and not easy to handle by the engineers in industrial projects.

Barrier : Formal property expression



Properties : often related to specific use cases
Not necessary to verify them over all the environment scenarios.

Moreover, properties are often related to specific use cases of the system.

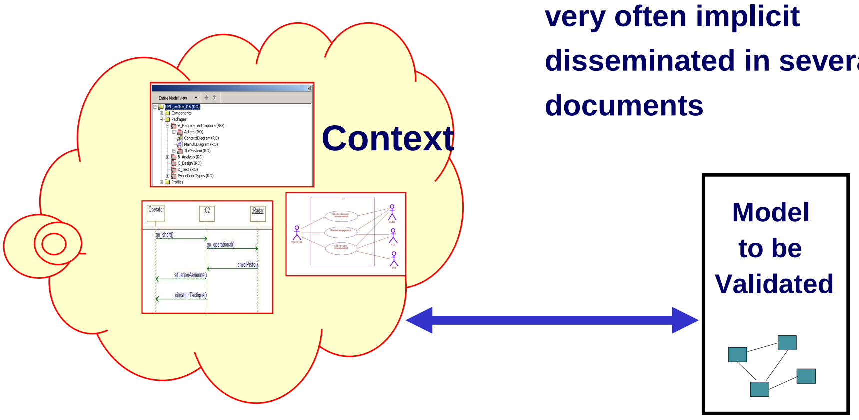
So, it is not necessary to verify them over all the environment scenarios. We want to identify these contexts.

Context expression

Contexts : to well-defined operational phases

initialization, reconfiguration, degraded modes, error scenarios, etc.

In reality, useful information :
very often implicit
disseminated in several documents



These contexts correspond to well-defined operational phases,

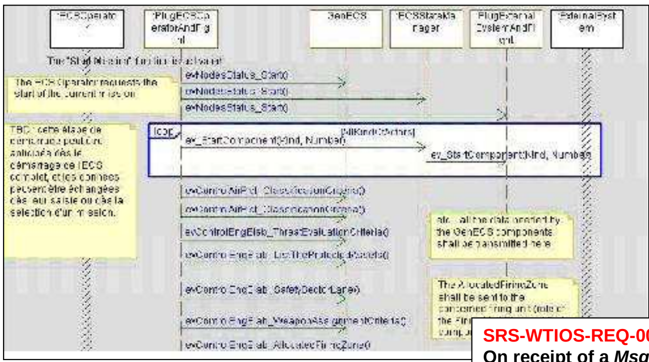
such as, for example, initialization, reconfiguration, degraded modes, error scenarios, etc.

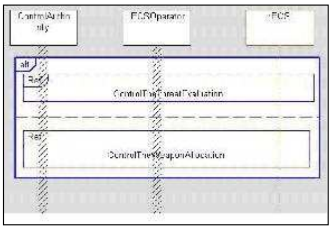
But, in reality, in the specifications,

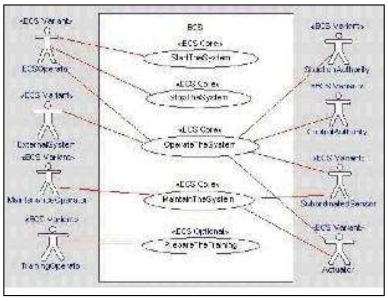
useful information about the context execution

is very often implicit or disseminated in several documents

Context expression







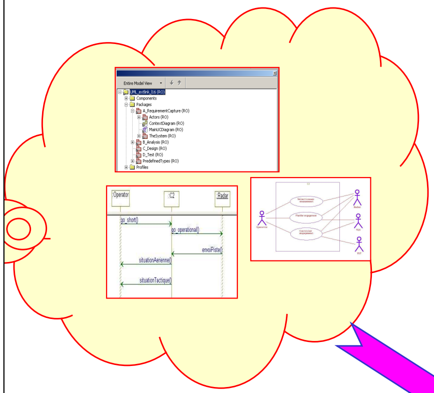
SRS-WTIOS-REQ-004
On receipt of a *MsgFieldMask* message from the COMM_WT, the WT_IOS shall set the WT_State to 'STANDBY' and transmit the *EvtTechnicalStateIos* message to the ECDP_DP with the following parameters:
equipmentId = equipmentId of the WT_IOS
roleId = roleId of the WT_IOS
state = STANDBY
If the requested WT_State is OPERATIONAL, the WT_IOS shall transmit the *MsgControlNetwork* message to the COMM_WT with the following parameters:
orderId
command = 'READY'
End Requirement

The understanding of these documents is not easy

And sometimes the link between them is implicit.

Context expression

Method to support the formal specification of these contexts



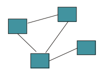
Model : tightly synchronized with its environment.

Circumvent the combinatorial explosion

Formalization

Context
(scenarios)

Model
to be
Validated



We study a method to support the formal specification of these contexts

in which the model will be validated.

In this condition, the model is tightly synchronized with its environment.
It is a way to circumvent the problem of combinatorial explosion

A strong hypothesis

The proof relevance : based on a strong hypothesis :

It is possible to specify the sets of bounded behaviors in a complete way.

Not formally justified

The essential idea :

The designer can correctly develop a software only if he knows the constraints of its use.

The proof relevance is based on a strong hypothesis :

It is possible to specify the sets of bounded behaviors in a complete way.

This hypothesis is not formally justified in our work.

But, the essential idea is :

The designer can correctly develop a software only if he knows the constraints of its use.

It is particularly true in embedded systems domain.

Pragmatic approach for integration in engineering processes

Adoption of a pragmatic approach.

Currently, we focus on :

- a formalization of use cases (contexts) and requirements
- a construction of a methodology for model validation without changing in deep their practices.

We adopt a pragmatic approach

Currently, we focus on :

a formalization of use cases (contexts) and requirements

a construction of a methodology for model validation without changing in deep the industrial practices.

Evaluating Context Descriptions and Property Definition Patterns

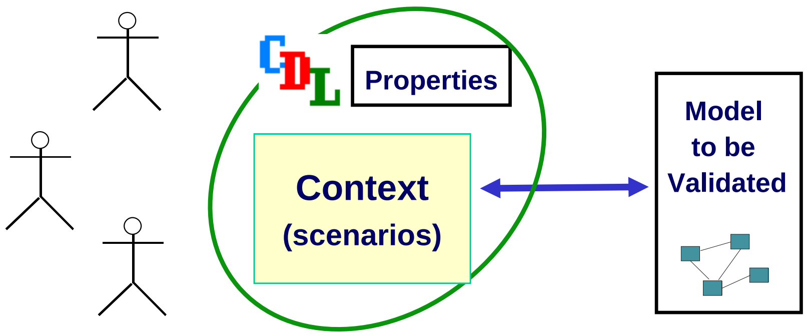
- **Motivation**
- **Proposition**
- **Context and requirement modeling (CDL)**
- **Experimentation : toolset (OBP) and results**
- **Discussion and future work**

Context Description Language (CDL)

Description of the environmental context : difficult task.

Context Description Language (CDL) :

- **to specify the context with scenarios and temporal properties**
- **to link each property to a limited scope of the system behavior.**



In the case of an environment composed of several parallel actors :

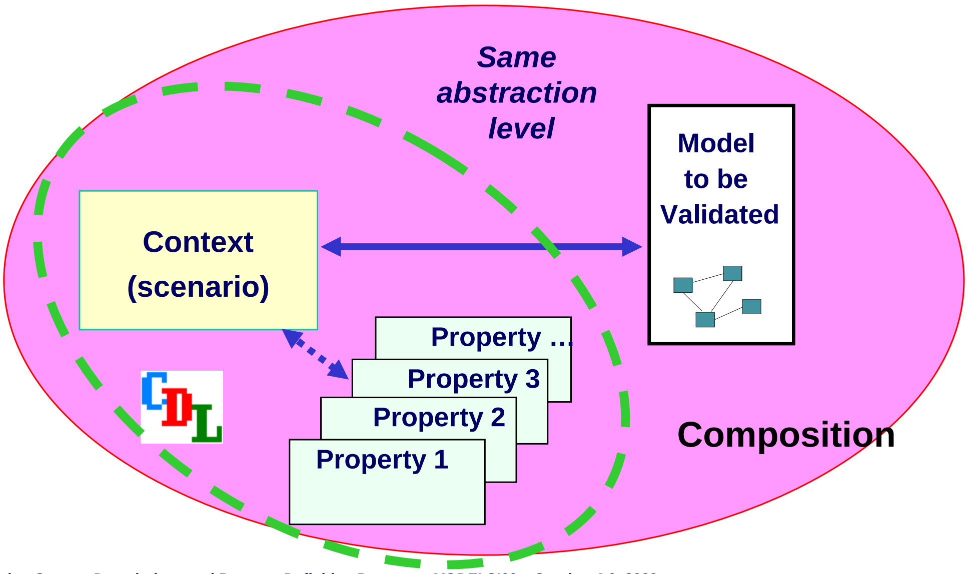
describing the environmental context can be a difficult task.

We proposed the Context Description Language (CDL) :

to specify the context with scenarios and temporal properties
and to link each property to a limited scope of the system behavior.

Verification principles

Based on a composition of the context, a set of requirements and the model to be validated.



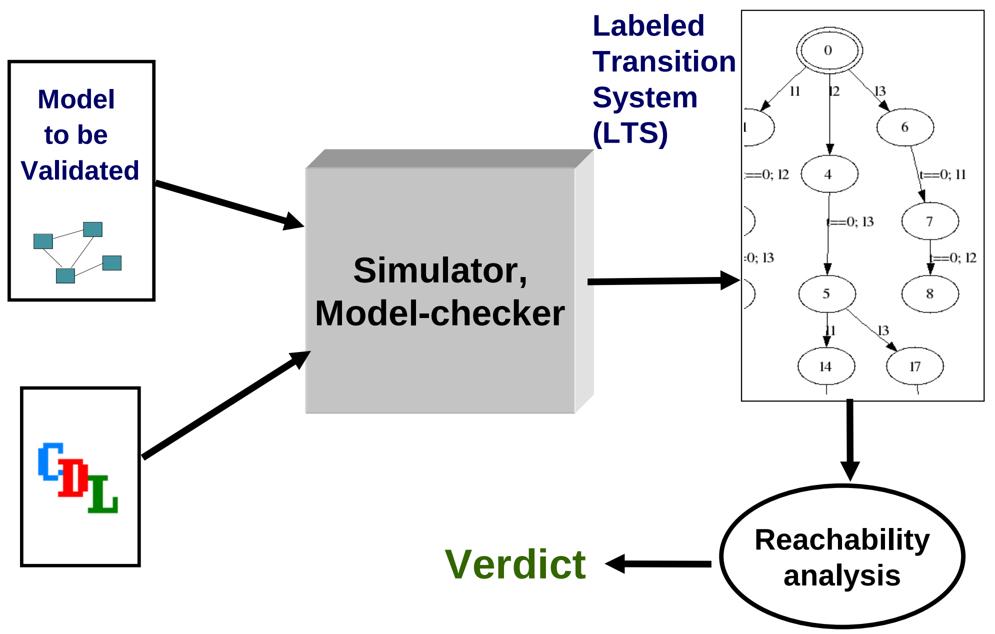
The verification process is based on a composition of

- the context,
- a set of requirements
- and the model to be validated.

The elements of CDL models and model to be validated

should be at the same abstraction level

Verification principles : Composition



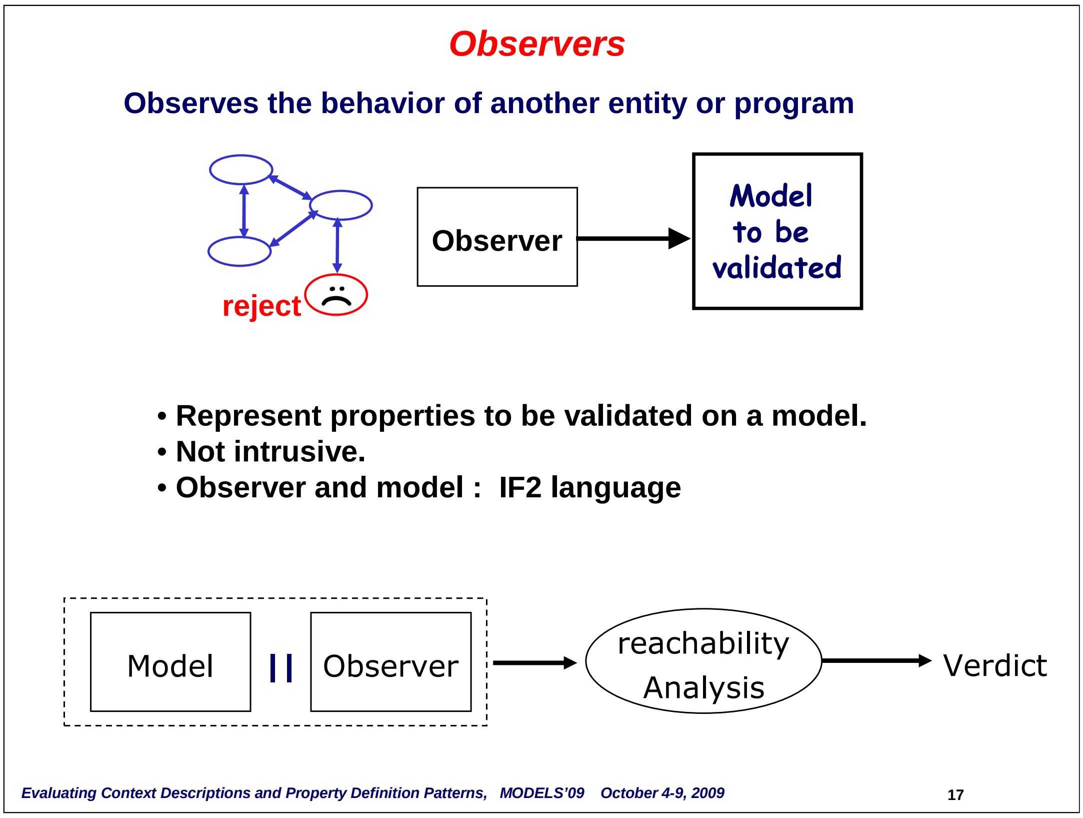
The composition is executed by a simulator or a model-checker.

The simulator generates a Labelled Transition System as the result of the composition

The LTS corresponds at the semantics of the model.

The accessibility analysis is carried out on this Transition System.

Before the composition, each property is transformed into an observer automaton.



An observer is an entity or a program which observes the behavior of another entity.

An observer represents a property to be validated on a model.

In that case, observers are not intrusive.

In our work, the observer and the model to be validated are described in the same language : IF2 language, based on timed automata, from Verimag laboratory.

The accessibility analysis consists of checking if some observer state is reached.

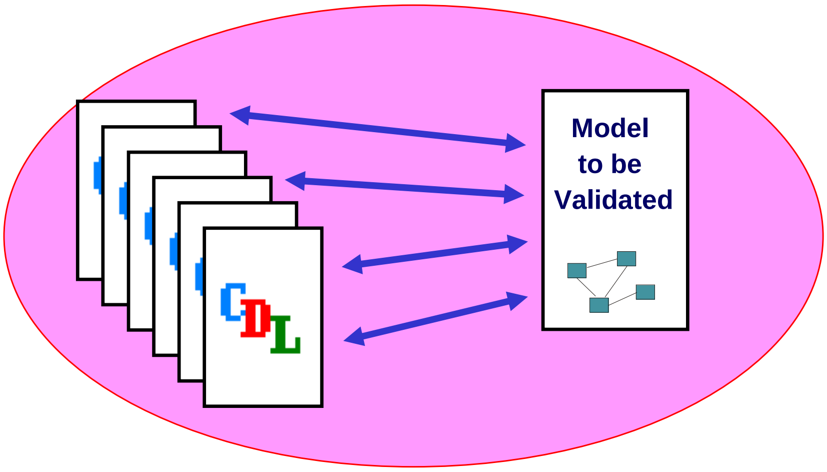
For example, if a *reject* state is reached for an observer,

then the property is considered as false.

Verification principles

Identification of many contexts.

Validation with a set of CDL models

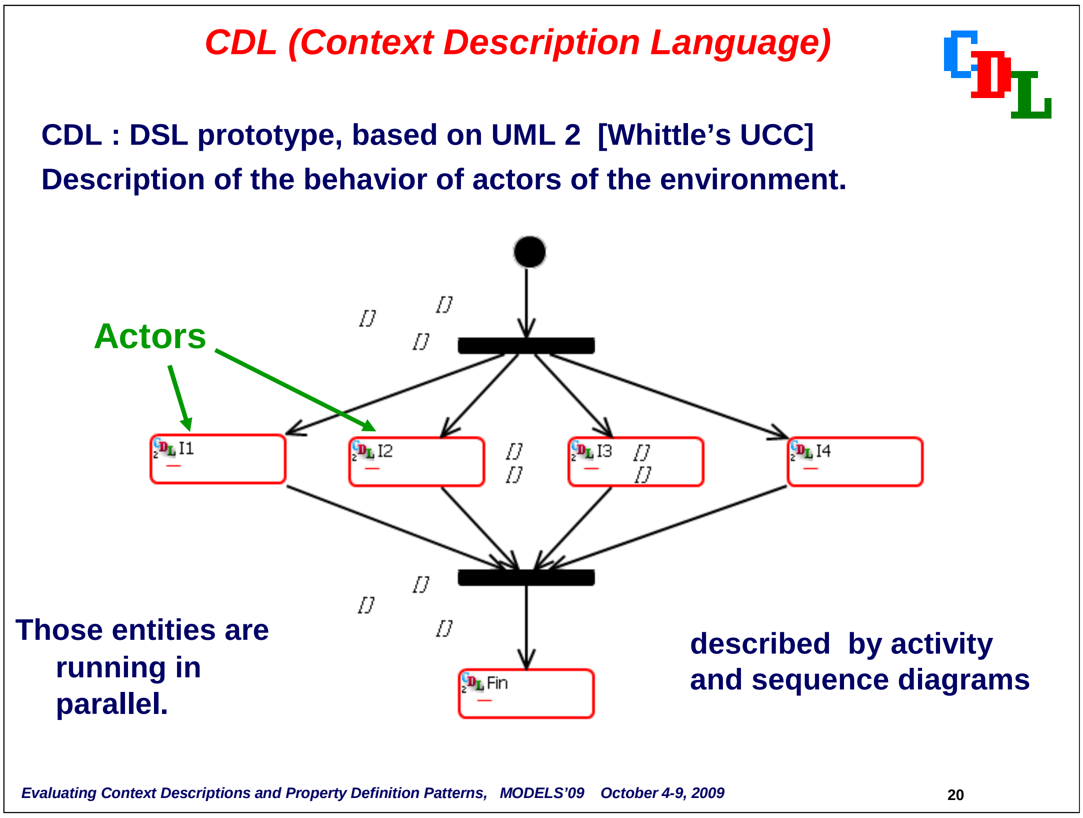


For one software component, we have to specify many contexts.

The proof of a set of properties implies the validation with a set of CDL models

Evaluating Context Descriptions and Property Definition Patterns

- **Motivation**
- **Proposition**
- **Context and requirement modeling (CDL)**
- **Experimentation : toolset (OBP) and results**
- **Discussion and future work**

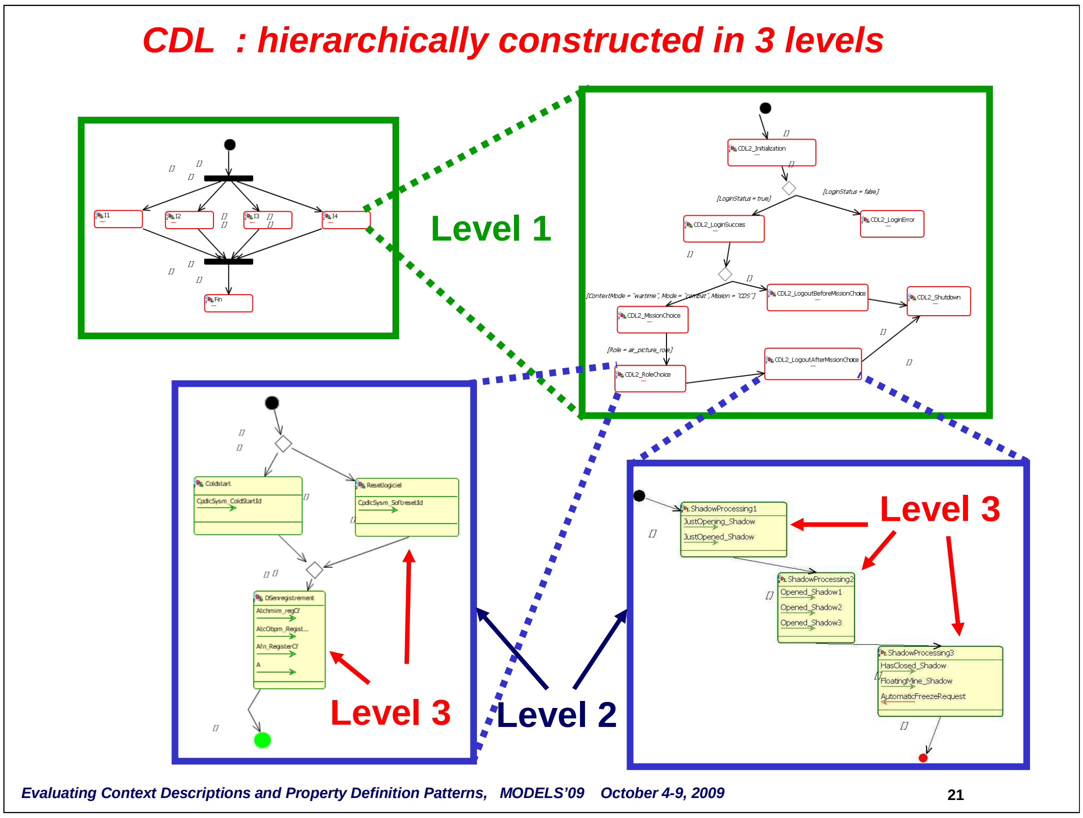


CDL is a DSL prototype, based on UML 2 and inspired by Jon Whittle's Use Case Charts .

A CDL model describes the behavior of actors of the environment.

Those entities are running in parallel.

The behavior of the actors is described by activity and sequence diagrams



CDL is constructed in three levels:

Level-1 and level 2 with activity diagrams

And level 3 with sequence diagrams.

Context – properties linking

ECDP-DP-REQ-006

During initialization procedure, the ECDP_DP shall associate a generic equipment identifiers to one or several role in the system (MainSensor, OtherSensor, IFF, Actuator, ...). It shall also associate an identifier to each console.
The ECDP_DP shall send an evtEquipmentRole message, in preparation mode, for each connected generic equipment, to each connected console.
Initialization procedure shall end successfully, when the ECDP_DP has set all the generic equipment identifiers and all console identifiers and all evtEquipmentRole message have been sent.

End

ECDP-DP-REQ-008

Once initialization is achieved, the ECDP_DP shall send to each console an evtCurrentMission with curMission set to IDLE, to set current mission to idle, followed by an evtCurrentActivity with curActivity to LOGIN and status to TRUE to activate login.

End

Industrial projects :

Requirements :

- **not associated to the entire lifecycle of software**
- **only to specific steps in its lifecycle.**

In several industrial projects :

all the requirements are not associated to the entire lifecycle of software, but only to specific steps in its lifecycle.

This is an example from Thales.

Context – properties linking

ECDP_DP-REQ-006

During initialization procedure, the ECDP_DP shall associate a generic equipment identifiers to one or several role in the system (MainSensor, OtherSensor, IFF, Actuator, ...). It shall also associate an identifier to each console.
The ECDP_DP shall send an evtEquipmentRole message, in preparation mode, for each connected generic equipment, to each connected console.
Initialization procedure shall end successfully, when the ECDP_DP has set all the generic equipment identifiers and all console identifiers and all evtEquipmentRole message have been sent.

End

ECDP_DP-REQ-008

Once initialization is achieved, the ECDP_DP shall send to each console an evtCurrentMission with curMission set to IDLE, to set current mission to idle, followed by an evtCurrentActivity with curActivity to LOGIN and status to TRUE to activate login.

End

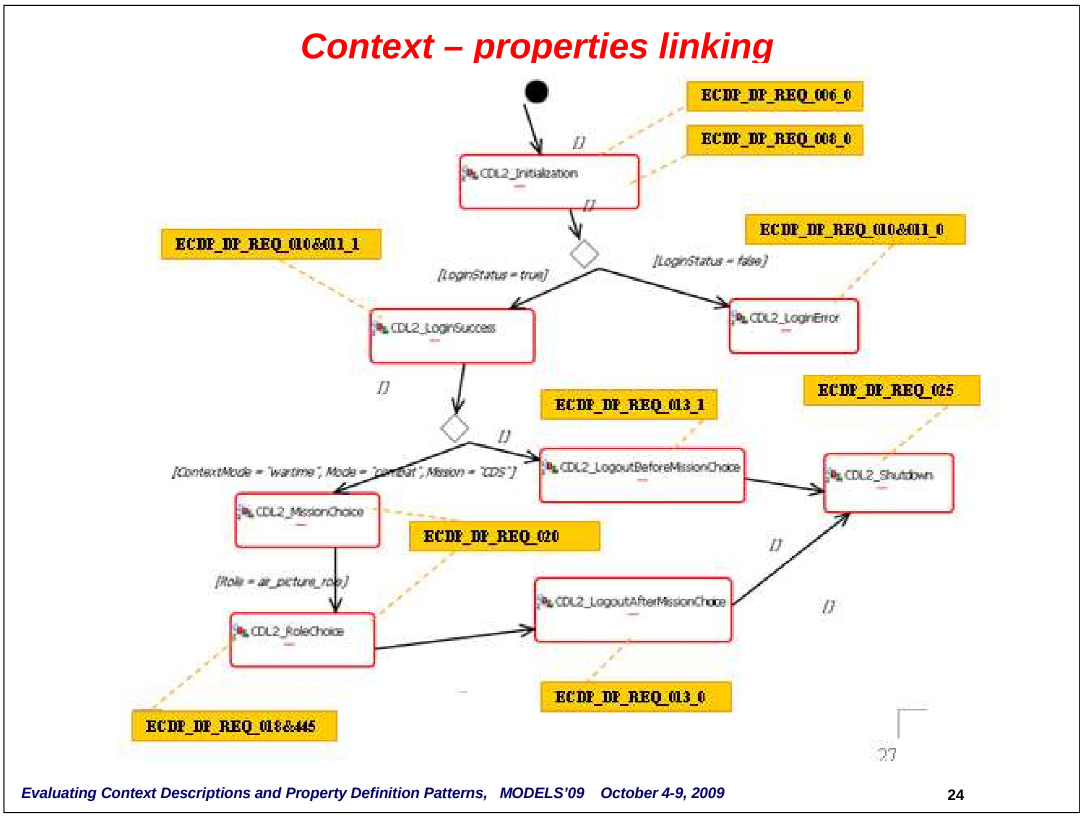
Context

Link
- properties
- specific context

In the system specification documents,

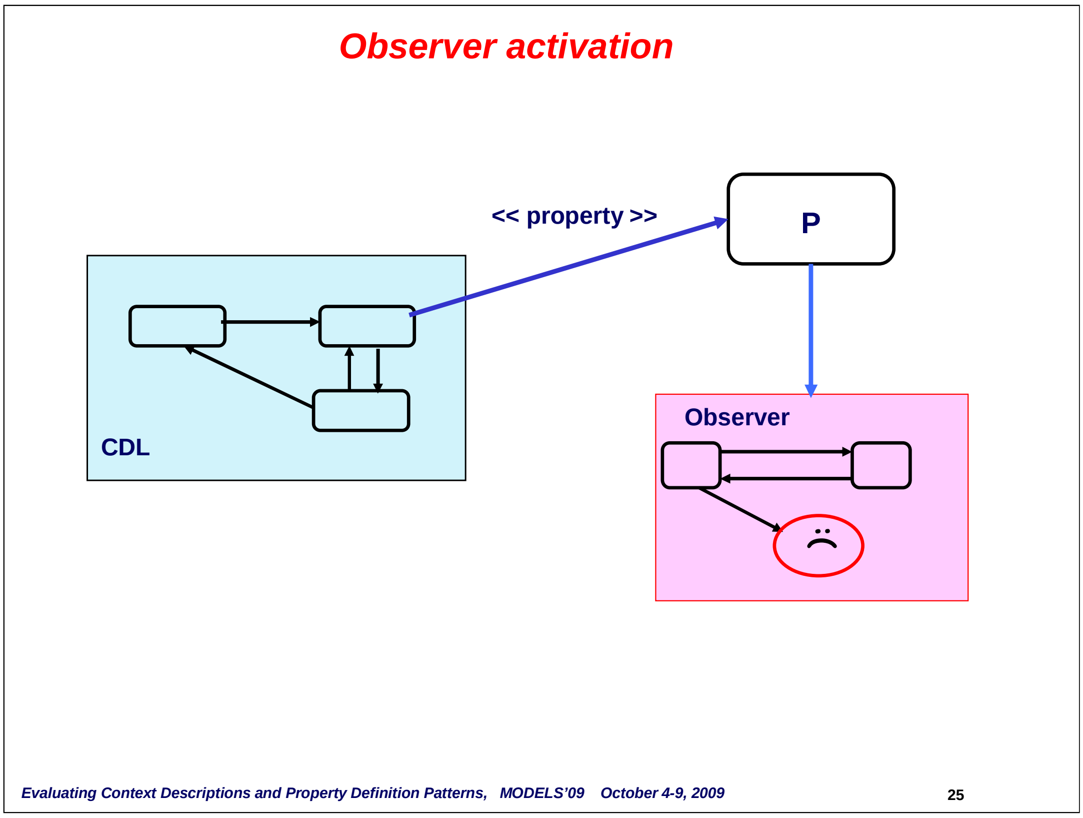
requirements are often expressed in a context of the system execution.

For this reason :
we propose to link formalized properties
to a specific execution context
and thus to limit the scope of the properties.



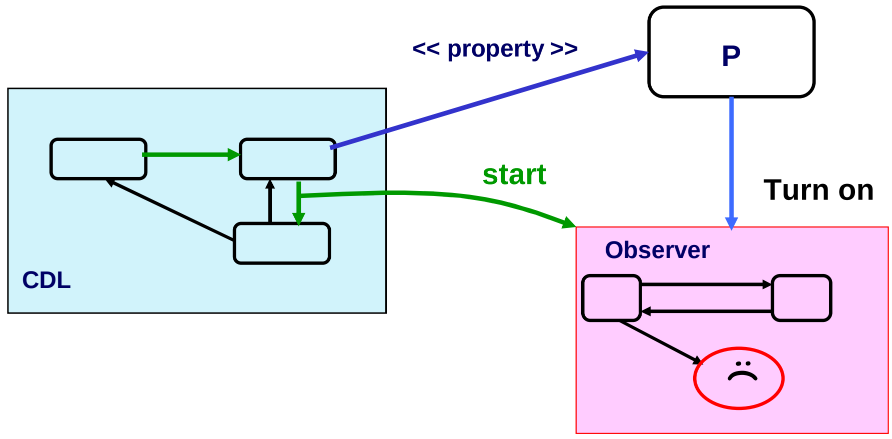
The originality of CDL is its ability
to link each property to a context diagram
(at level 1 or 2)

One property can be associated to a node of activity diagram.
The interest is to link a property to an execution of the context

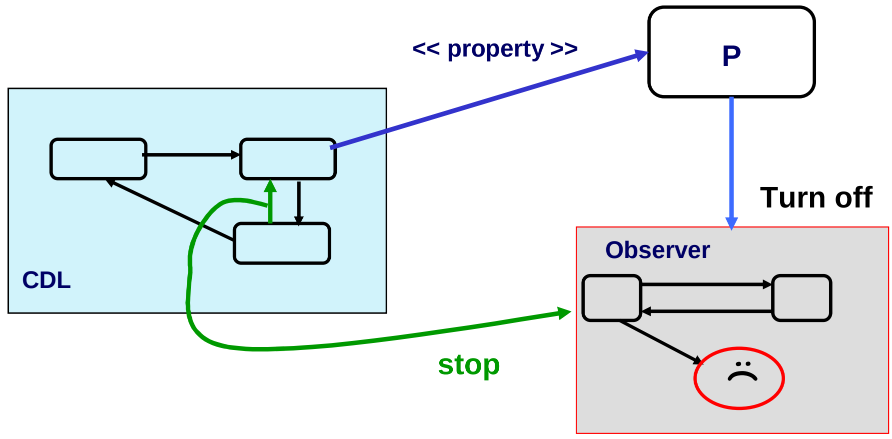


We implemented a mechanism to allow the observers to be activated during the execution of a activity diagram.
During the generation of an observer automaton, we add transitions for synchronization with the context.
So, the context can start et stop the observer.

Observer activation



Observer activation



Context – properties linking : benefits

The benefits are :

- **to explicit the conditions under which a given property is checked.**

- **Easier property specification**

The benefits of linking are :

- **to explicitly specify the conditions for application of this property**

- **The properties are specified easier.**

Requirement formalization

ECDP-DP-REQ-006
During initialization procedure, the ECDP_DP shall associate a generic equipment identifiers to one or several role in the system (MainSensor, OtherSensor, IFF, Actuator, ...). It shall also associate an identifier to each console.
The ECDP_DP shall send an evtEquipmentRole message, in preparation mode, for each connected generic equipment, to each connected console.
Initialization procedure shall end successfully, when the ECDP_DP has set all the generic equipment identifiers and all console identifiers and all evtEquipmentRole message have been sent.
End

Extracted from industrial documentation.

Textual requirement

→ ambiguous and complex.

About the requirement formalization :

On this requirement extracted from industrial documentation.

We observe that this textual requirement is ambiguous and too complex.

Requirement decomposition

After a discussion with industrial partners, we had to rewrite it :

→ to decompose in a set of requirements.

→ to formalize them.

ECDP-DP-REQ-006-*1*

During initialization procedure, the ECDP_DP shall associate an identifier to NC console (IHM), before dMax_cons time units.

ECDP-DP-REQ-006-2

After initialization, in preparation mode, the ECDP_DP shall shall send an evtEquipmentRole for each connected generic equipment, to each connected console, before dMax_dev time units.

ECDP-DP-REQ-006-3

Each device returns a statusRole message to ECDP_DP before dMax_ack time units.

ECDP-DP-REQ-006-4

The ECDP_DP shall send an notifyRole message for each connected generic device, to each connected console. Initialization procedure shall end successfully, when the ECDP_DP has set all the generic device identifiers and all console identifiers and all notifyRole messages have been sent.

After a discussion with industrial partners,

we had to rewrite it :

→ to decompose in a set of requirements.

→ to formalize them.

Property formalization with definition patterns

ECDP-DP-REQ-006-1
During initialization procedure, the ECDP_DP shall associate an identifier to NC console (IHM), before dMax_cons time units.

Pattern-based approach.
[Dwyer, Cheng]

Response, Precedence, Absence, Existence

Property ECDP-DP-REQ-006-1;

```
AN
          exactly one occurrence of chgt_state_ECDP
end

eventually leads-to [ 0 .. dmax_cons [

ALL combined
          exactly one occurrence of  send_1_cons
          exactly one occurrence of  send_2_cons
end

chgt_state_ECDP may never occur
one of send_1_cons cannot occur before the first one of chgt_state_ECDP
one of send_2_cons cannot occur before the first one of chgt_state_ECDP
repeatability : true
```


For the property formalization,
we use a pattern-based approach.
We reuse the categories of Dwyer's and Cheng's patterns.

Patterns are classified in basic families :
Response, Precedence, Absence, Existence

Property definition pattern : detectable events

Detectable events :

transmissions or receptions, actions, model state changes.

Property ECDP-DP-REQ-006-1;

AN
exactly one occurrence of chgt_state_ECDP
end

eventually leads-to [0 .. dmax_cons [

ALL combined
exactly one occurrence of send_1_cons
exactly one occurrence of send_2_cons
end

chgt_state_ECDP may never occur
one of send_1_cons cannot occur before the first one of chgt_state_ECDP
one of send_2_cons cannot occur before the first one of chgt_state_ECDP
repeatability : true

The properties refer to detectable events like

transmissions or receptions of signals, actions, model state changes.

Property definition pattern : sets of events

Possibility of handling **sets of events,** ordered or not ordered.

Property ECDP-DP-REQ-006-1;

AN
exactly one occurrence of chgt_state_ECDP
end

eventually leads-to [0 .. dmax_cons [

ALL combined
exactly one occurrence of send_1_cons
exactly one occurrence of send_2_cons
end

chgt_state_ECDP may never occur
one of send_1_cons cannot occur before the first one of chgt_state_ECDP
one of send_2_cons cannot occur before the first one of chgt_state_ECDP
repeatability : true

AN **: an event**
ALL **: all the events**

There is a possibility of handling sets of events, ordered or not ordered.

With key-words as AN and ALL

Property definition pattern : options

Enrichment with Options using annotations [Smith].
To produce distinct variations on a property pattern.

Property ECDP-DP-REQ-006-1;

```
AN
        exactly one occurrence of chgt_state_ECDP
end

eventually leads-to [ 0 .. dmax_cons [

ALL combined
        exactly one occurrence of  send_1_cons
        exactly one occurrence of  send_2_cons
end

chgt_state_ECDP may never occur
one of send_1_cons cannot occur before the first one of chgt_state_ECDP
one of send_2_cons cannot occur before the first one of chgt_state_ECDP
repeatability : true
```

Pre-arity,
Post-arity,
Immediacy,
Precedence,
Nullity,
Repeatability

These basic forms are enriched by different options using annotations [Smith]. These options combine to produce distinct variations on a property pattern.

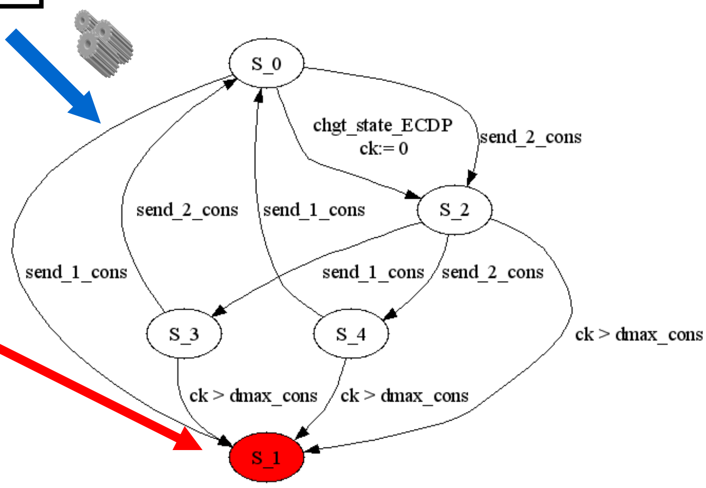
Transformation into an observer automaton

Each property is transformed into an observer automaton (*reject* node)
Safety and bounded liveness properties.

Pty ECDP-DP-REQ-006-1

Accessibility analysis :

Checking on the global LTS if a *reject* state is reached



Currently, our toolset transforms each property into an observer automaton, including a *reject* node.

With observers, the properties we can handle are of safety and bounded liveness type.

The accessibility analysis consists of checking on the global LTS if a *reject* state is reached

Evaluating Context Descriptions and Property Definition Patterns

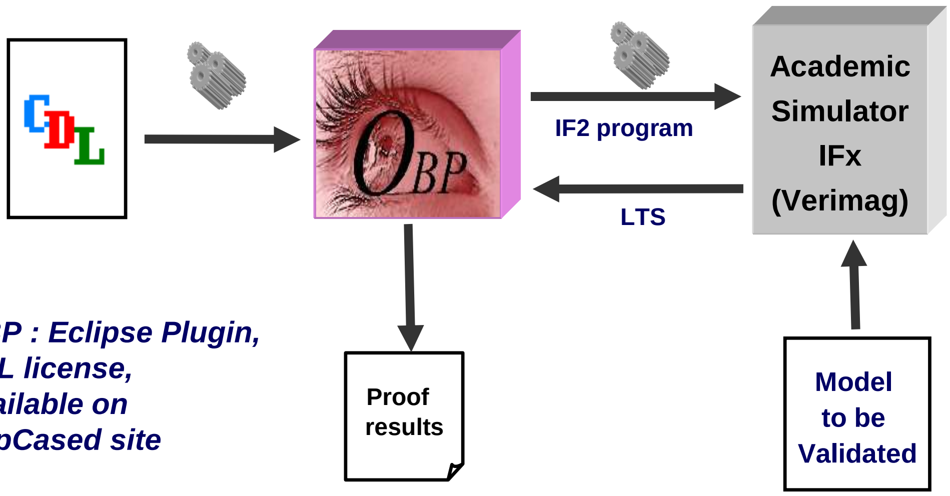
- **Motivation**
- **Proposition**
- **Context and requirement modeling (CDL)**
- **Experimentation : toolset (OBP) and results**
- **Discussion and future work**

To carry out our experiments, we implemented OBP tool.

Toolset : Observer-Based Prover

OBP takes as input the model to be validated and each CDL model.

Connection to an existing academic simulator IFx (with IF2 language)



OBP : Eclipse Plugin, EPL license, available on TopCased site

OBP takes as input the model to be validated and each CDL model.

Currently, OBP is connected to an existing academic simulator IFx, from Verimag Laboratory,

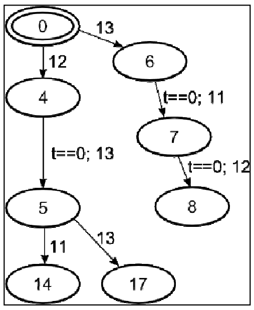
OBP generates IF2 code for IFx

And IFx return a Transition System.

This LTS is analyzed by OBP to provide a result about the properties

Generation of context paths

Unfolding
Interleaving



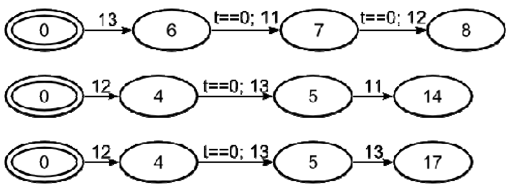
From CDL diagrams :

Generation of a set of context path automata.

Each path :

one environment run

Partionning



From CDL context diagrams,
OBP tool generates a set of context path automata.

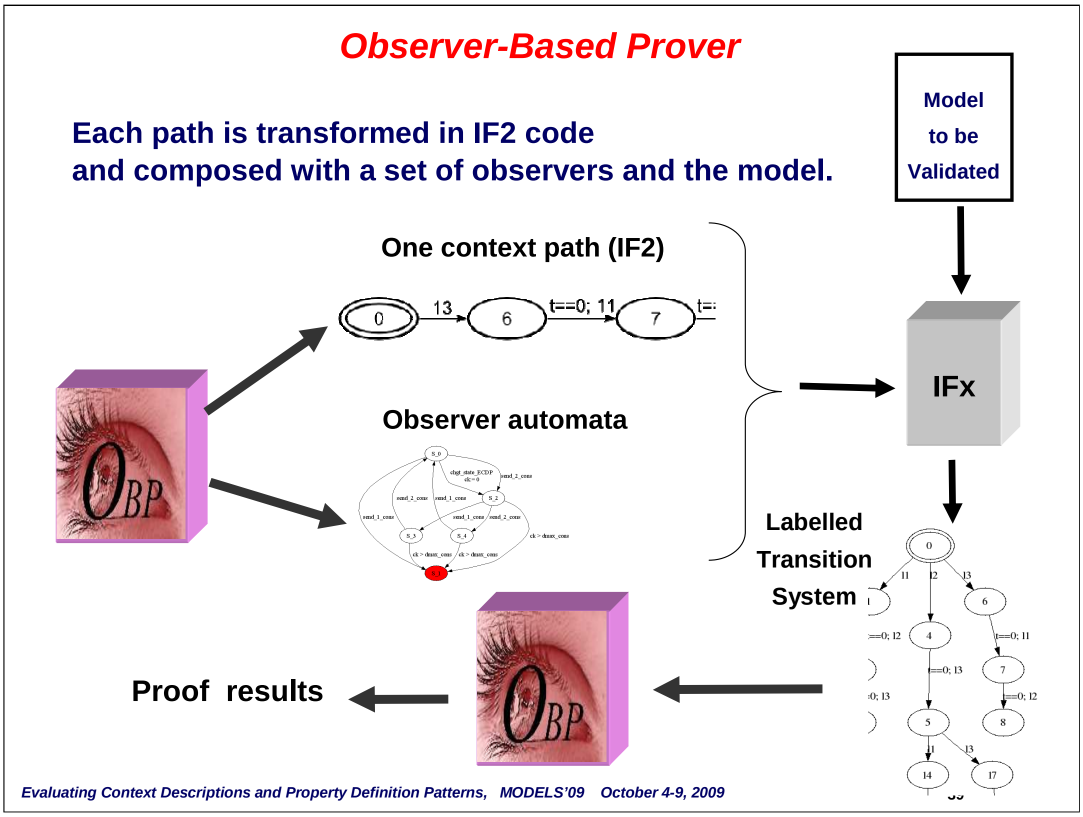
For that, the contexts diagrams are unfolded to automata.

OBP generates one automaton from the interleaving of them with the taking into

account of their parallel executions.

Then, this automaton is partitioned into a set of path automata.

Each path represents one environment run, one possible interaction between model and context.



Each path is transformed into IF2 code

which is composed with the model to be validated and the generated observer automata.

OBP execute the accessibility analysis on the global LTS

to check the properties

Some industrial experiments results (schema)

Several industrial software components of embedded systems.

For each component :

- **requirement documents**
  - **Use cases,**
  - **Requirements (natural language)**
- **component executable model (UML or SDL).**
- **translated in IF2 models (manually or semi-automatically)**

Our approach was applied to several industrial software components of embedded systems.

For each component :

The partner provided requirement documents (use cases, requirements in natural language)

and the component executable model.

Theses executable models are described with UML,

completed by ADA or JAVA programs, or with SDL language.

Theses models were translated in IF2 models, often manually, sometimes semi-automatically.

Some industrial experiments results

In industrial documents, requirements :

- At different abstraction levels
  - extraction of requirements corresponding to the model abstraction level.
- Rewritten into a set of several properties
  - Decomposition
  - Pattern-based rewriting consequently to discussion with industrial partners.

In industrial documents :

Firstly, the requirements of different abstraction levels are mixed.

We extracted requirements corresponding to the model abstraction level.

Secondly, most of requirements had to be rewritten into a set of several properties.

Finally, the most of the textual requirements are ambiguous.

We had to rewrite them, with the pattern, consequently to discussions with industrial partners.

Some industrial experiments results

	AFN (Airbus)	ADS (Airbus)	CPDLC (Airbus)	A623 (Airbus)	PAAMS (Thales)	ECDP (Thales)	Average
Provable properties	38/49 (78%)	73/94 (78%)	72/136 (53%)	49/85 (58%)	155/188 (82%)	41/151 (27%)	428/703 (61%)
Non-computable properties	0/49 (0%)	2/94 (2%)	24/136 (18%)	2/85 (2%)	18/188 (10%)	48/151 (32%)	94/703 (13%)
Non-provable properties	11/49 (22%)	19/94 (20%)	40/136 (29%)	34/85 (40%)	15/188 (8%)	62/151 (41%)	181/703 (26%)

Number of properties translated from requirements.
Three categories of properties.

Here, we reports on six case studies.

Four of them come from AIRBUS and two from THALES.

This table shows the number of properties which are translated from requirements.

We consider three categories of properties.

Some industrial experiments results

	AFN (Airbus)	ADS (Airbus)	CPDLC (Airbus)	A623 (Airbus)	PAAMS (Thales)	ECDP (Thales)	Average
Provable properties	38/49 (78%)	73/94 (78%)	72/136 (53%)	49/85 (58%)	155/188 (82%)	41/151 (27%)	428/703 (61%)
Non-computable properties	0/49 (0%)	2/94 (2%)	24/136 (18%)	2/85 (2%)	18/188 (10%)	48/151 (32%)	94/703 (13%)
Non-provable properties	11/49 (22%)	19/94 (20%)	40/136 (29%)	34/85 (40%)	15/188 (8%)	62/151 (41%)	181/703 (26%)

captured with our patterns, translated into observers

Provable **properties can be captured with our patterns**

and can be translated into observers.

Some industrial experiments results

	AFN (Airbus)	ADS (Airbus)	CPDLC (Airbus)	A623 (Airbus)	PAAMS (Thales)	ECDP (Thales)	Average
Provable properties	38/49 (78%)	73/94 (78%)	72/136 (53%)	49/85 (58%)	155/188 (82%)	41/151 (27%)	428/703 (61%)
Non-computable properties	0/49 (0%)	2/94 (2%)	24/136 (18%)	2/85 (2%)	18/188 (10%)	48/151 (32%)	94/703 (13%)
Non-provable properties	11/49 (22%)	19/94 (20%)	40/136 (29%)	34/85 (40%)	15/188 (8%)	62/151 (41%)	181/703 (26%)

captured with our patterns but not translated into an observer (unbounded liveness)

Non computable **properties can be interpreted by a pattern**
but cannot be translated into an observer.
It is the case of liveness properties
which cannot be translated because they are not bounded.

Some industrial experiments results

	AFN (Airbus)	ADS (Airbus)	CPDLC (Airbus)	A623 (Airbus)	PAAMS (Thales)	ECDP (Thales)	Average
Provable properties	38/49 (78%)	73/94 (78%)	72/136 (53%)	49/85 (58%)	155/188 (82%)	41/151 (27%)	428/703 (61%)
Non-computable properties	0/49 (0%)	2/94 (2%)	24/136 (18%)	2/85 (2%)	18/188 (10%)	48/151 (32%)	94/703 (13%)
Non-provable properties	11/49 (22%)	19/94 (20%)	40/136 (29%)	34/85 (40%)	15/188 (8%)	62/151 (41%)	181/703 (26%)

not captured with our patterns.
(example : undetectable events for the observer)

Non provable **properties cannot be interpreted at all with our patterns.**
It is the case when a property refers to undetectable events for the observer.

Some industrial experiments results

	AFN (Airbus)	ADS (Airbus)	CPDLC (Airbus)	A623 (Airbus)	PAAMS (Thales)	ECDP (Thales)	Average
Provable properties	38/49 (78%)	73/94 (78%)	72/136 (53%)	49/85 (58%)	155/188 (82%)	41/151 (27%)	428/703 (61%)
Non-computable properties	0/49 (0%)	2/94 (2%)	24/136 (18%)	2/85 (2%)	18/188 (10%)	48/151 (32%)	94/703 (13%)
Non-provable properties	11/49 (22%)	19/94 (20%)	40/136 (29%)	34/85 (40%)	15/188 (8%)	62/151 (41%)	181/703 (26%)

For the PAAMS component :

the percentage (82%) of provable properties : very high.

Most of properties : written with a good property pattern matching.

For the PAAMS component, the percentage (82%) of provable properties is very high.

One reason is that the most of properties was written with a good property pattern matching.

Some industrial experiments results

	AFN (Airbus)	ADS (Airbus)	CPDLC (Airbus)	A623 (Airbus)	PAAMS (Thales)	ECDP (Thales)	Average
Provable properties	38/49 (78%)	73/94 (78%)	72/136 (53%)	49/85 (58%)	155/188 (82%)	41/151 (27%)	428/703 (61%)
Non-computable properties	0/49 (0%)	2/94 (2%)	24/136 (18%)	2/85 (2%)	18/188 (10%)	48/151 (32%)	94/703 (13%)
Non-provable properties	11/49 (22%)	19/94 (20%)	40/136 (29%)	34/85 (40%)	15/188 (8%)	62/151 (41%)	181/703 (26%)

For the ECDP component :

the percentage (27%) is very low.

difficult to re-write the properties from specification documentation.

For the ECDP component, the percentage (27%) is very low.

It was very difficult to re-write the properties from specification documentation.

We should have spent much time to interpret properties with our partners to formalize them with our patterns.

Some industrial experiments results

	AFN (Airbus)	ADS (Airbus)	CPDLC (Airbus)	A623 (Airbus)	PAAMS (Thales)	ECDP (Thales)	Average
Provable properties	38/49 (78%)	73/94 (78%)	72/136 (53%)	49/85 (58%)	155/188 (82%)	41/151 (27%)	428/703 (61%)
Non-computable properties	0/49 (0%)	2/94 (2%)	24/136 (18%)	2/85 (2%)	18/188 (10%)	48/151 (32%)	94/703 (13%)
Non-provable properties	11/49 (22%)	19/94 (20%)	40/136 (29%)	34/85 (40%)	15/188 (8%)	62/151 (41%)	181/703 (26%)

About forty properties have been formally verified

2 errors : detected in AFN and ECDP

In the case studies, about forty properties have been formally verified.

In average :

The 61% (provable) are translated and can be verified automatically.

About 13% (non-computable) of the requirements required adaptation.

For the others 26%, the requirements have to be discussed with the partners to improve their use.

In two case studies (AFN and PAAMS) : we found an error in the models

The classical simulation techniques could not permit to find these errors.

Approach benefits

- Requirements often partially described.
- CDL : formalization of contexts and properties.
- Contribution to overcome the combinatorial explosion
- Motivation of the partners for a more formal approach to express their requirements.
- Better appropriation of formal verification process by partners.
- Help to structure and formalize their specification.

During experiments, some requirements were often partially described in the available documentation.

CDL permit to formalize contexts and non ambiguous properties.

CDL contribute to overcome the combinatorial explosion by allowing partial verification on restricted scenarios specified by the context automata.

During this collaboration, the partners were motivated to consider a more formal approach to express their requirements.

CDL improve a better appropriation of formal verification process by partners.

It is a help to structure and formalize their specification.

Difficulties

Major difficulties are :

- **Scenarios : lack of complete and coherent description**

  -> Many discussions with experts required

  -> Long discussions for understanding and capture in a model

In case studies, during the building of CDL models, the major difficulties are :

For the scenarios : there is a lack of complete and coherent description of the environment's behavior.

-> CDL diagrams development required many discussions with experts

-> Long discussions with engineers are usually needed to precisely understand the different contexts for the system and capture them in a CDL model.

Difficulties

- Requirements : difficulty to formalize them into formal properties.

  -> Different abstraction levels.
  -> Several interpretations.
  -> Some refer to an applicable configuration, operational phase or history without defining it.

- Complexity : CDL programming / path set complexity.

For the requirements : it is difficult to formalize them into formal properties.

-> because the different abstraction levels.

-> Several interpretations.

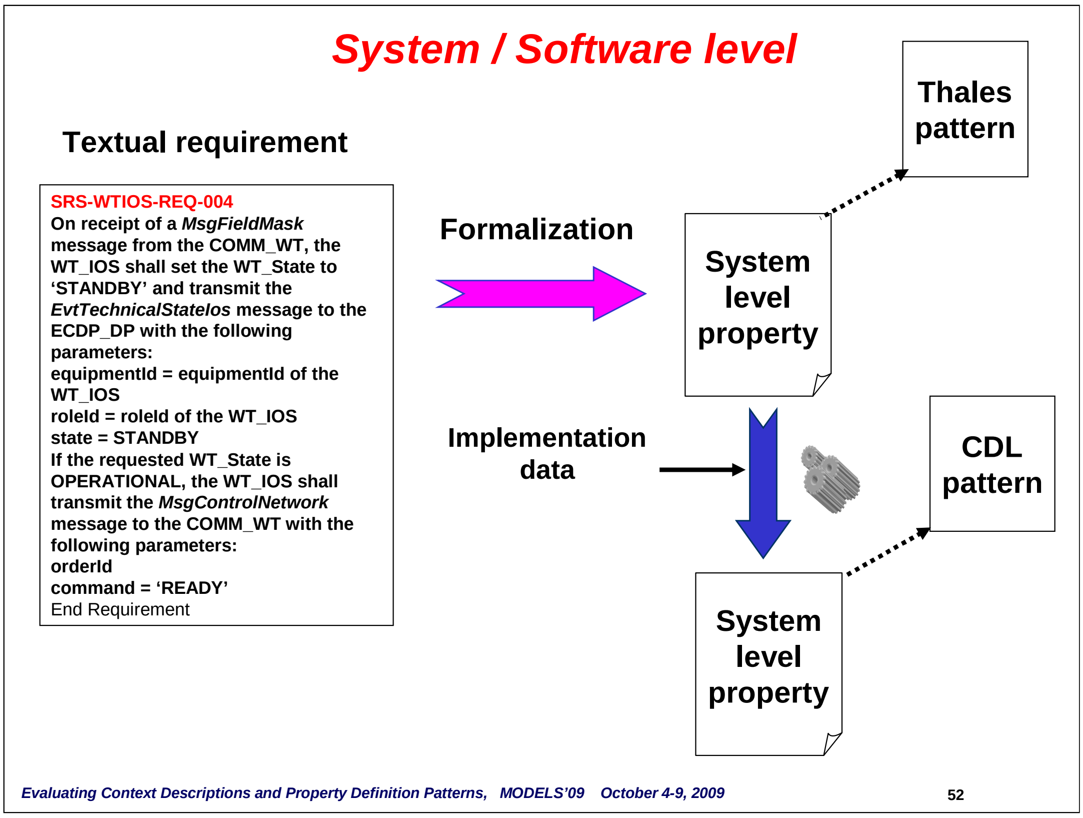
-> Some refer to an applicable configuration, operational phase or history without defining it.

Complexity : the style of CDL programming has an influence on the complexity of path sets.

With definition of contexts, the complexity is moved.

The sets of generated paths can be huge.

we have to study technics for their reduction.



If the software engineers can easily model the context with CDL,
the system engineers don't want formalize their properties with our CDL paterns.
at too low abstraction level for them.

For example at Thales, currently, we are working on definition of specific patterns at highter abstraction level.
We have to design transformations to generate CDL properties from these specific patterns.

System / Software level

hight level
Operator

Property ECDP_DP_REQ_006_1_3

(ECDP_DP is in state INITIALIZATION)
exactly one occurrences of
receive equipmentRoles () from IOS
leads to

(ECDP_DP is in state INITIALIZATION)

For each HMI : exactly one occurrences of
send process (evtEquipmentRole) to
HMI

equipmentRoles must occur

one of process cannot occur before
the first one of equipmentRoles

Here is an example of property conformed to specific pattern
With higther level operator.
But this work is in progress and is not mature.

Evaluating Context Descriptions and Property Definition Patterns

- **Motivation**
- **Proposition**
- **Context and requirement modeling (CDL)**
- **Experimentation : toolset (OBP) and results**
- **Conclusion and future work**

Conclusion

Work in progress.

Academic

Evaluating CDL on industrial context
Execution of proofs
Study of concepts and a methodology
CDL concepts can be implemented in another language

Industrial

Experimentations : rich in teaching
Contribution for a formal validation methodology

Users

Appropriation of formal verification process by partners
Not trivial activity

This work is still in progress.
On Academic aspect : We are evaluating the CDL on industrial case studies.
Formal proofs were executed on software components by the partners themselves.
CDL allows us to study concepts and a methodology for the formal validation in industrial context.
CDL is a prototype language.
But CDL concepts can be implemented in another language.

Conclusion

Work in progress.

Academic

Evaluating CDL on industrial context
Execution of proofs
Study of concepts and a methodology
CDL concepts can be implemented in another language

Industrial

Experimentations : rich in teaching
Contribution for a formal validation methodology

Users

Appropriation of formal verification process by partners
Not trivial activity

On Industrial aspect : These experimentations are rich in teaching for our partners and us.

They allows us to design a methodology, adapted to industrial processes. and the first feedback is encouraging.

Users : CDL allows partners to a better appropriation of formal verification process.

But it is obvious that specifying all these contexts is not a trivial activity.

It takes a great part of time and effort within a project.

But we are at the beginning of very long process.

Future works

Academic

- Path set reduction : equivalence
- Pattern improvement
- Improve diagnostics readability

Industrial

- Need of formal validation expertise capitalization
- Methodology of CDL models design

Users

- Methodology and guidelines

For futur w, Our work focuses on path reduction. We evaluate how paths can be equivalent with respect to a particular property.

This optimization aims at reducing the combinatorial explosion, allowing treating larger applications.

Extension of the property expression

Experiments shown that part of the requirements found in industrial specification documents

were not translatable into property patterns proposed by this approach.

Several directions are followed to face the problem,

one is to extend actual patterns,

and another is to create other patterns.

Future works

Academic

- Path set reduction : equivalence
- Pattern improvement
- Improve diagnostics readability

Industrial

- Need of formal validation expertise capitalization
- Methodology of CDL models design

Users

- Methodology and guidelines

Understanding feedbacks for diagnostics

During the proofs, the interpretation of the feedback obtained

by the analysis is difficult on the large LTS.

If one observer is in an error state, we have to understand why.

We would like to get validation feedbacks on user source models.

We should take advantages of model driven techniques and transformation traces in tooling

to have validation feedbacks on source models.

Future works

Academic

- Path set reduction : equivalence
- Pattern improvement
- Improve diagnostics readability

Industrial

- Need of formal validation expertise capitalization
- Methodology of CDL models design

Users

- Methodology and guidelines

We have to define a methodology to design CDL models in a complete way.

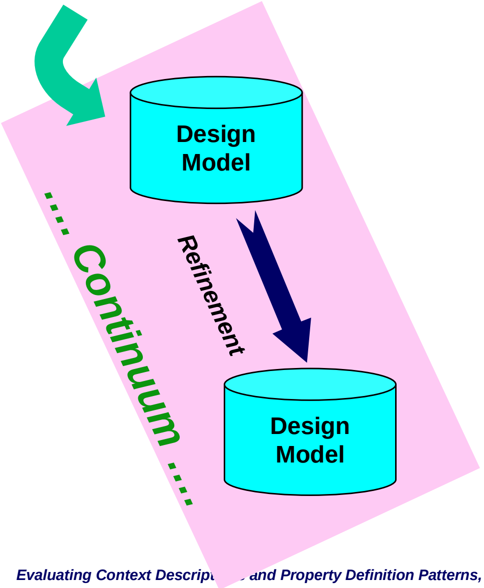
The development process must include a step of environmental specification making it possible to generate sets of bounded behaviors in a complete way.

This must be provided formally by the process analysis of the designed software architecture, using a framework of development process.

Thank for your attention

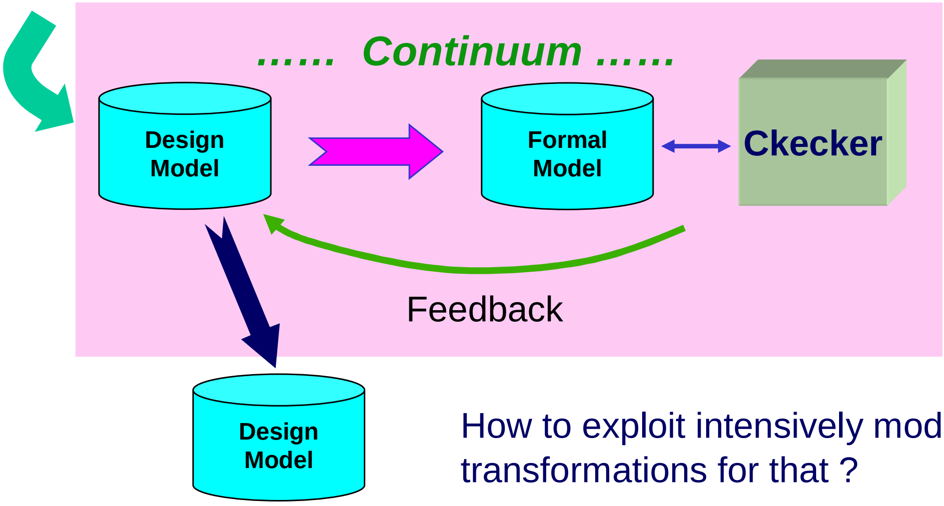
Integration of formal methods into engineering processes

MDE : a great opportunity to implement a continuum from design model to code.



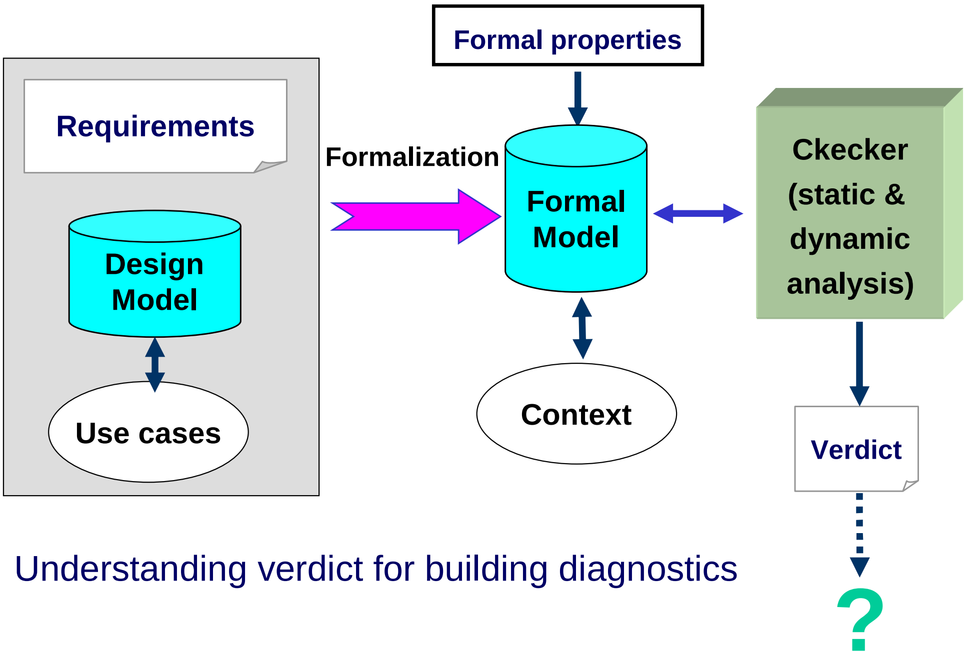
Integration of formal methods into engineering processes

Is MDE a great opportunity to implement a continuum from design model + requirements to diagnostics ?



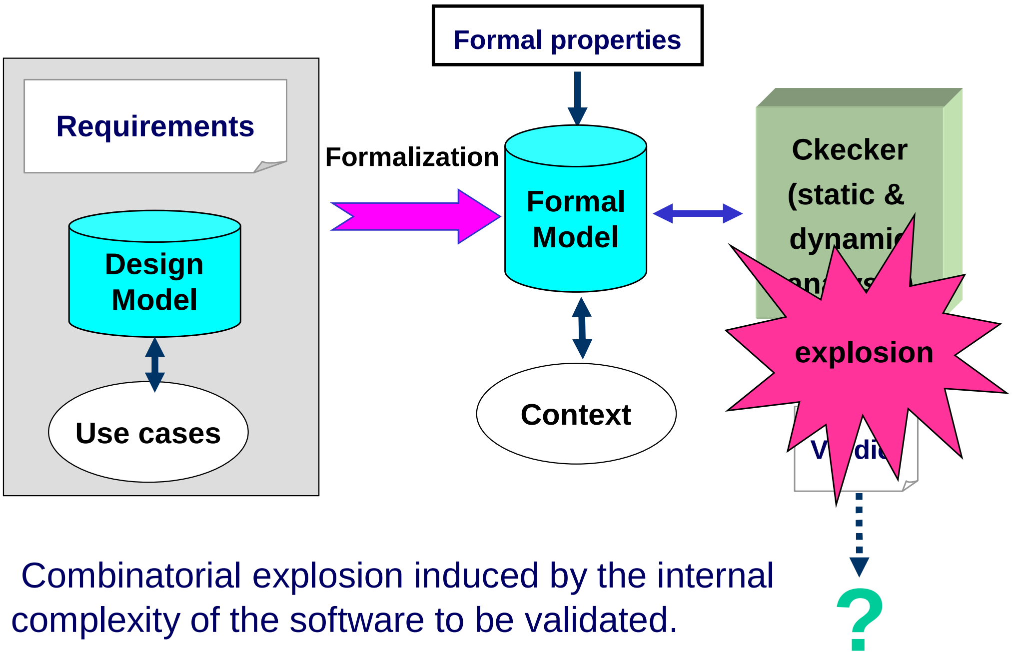
How to exploit intensively model transformations for that ?

Barrier : diagnostic building



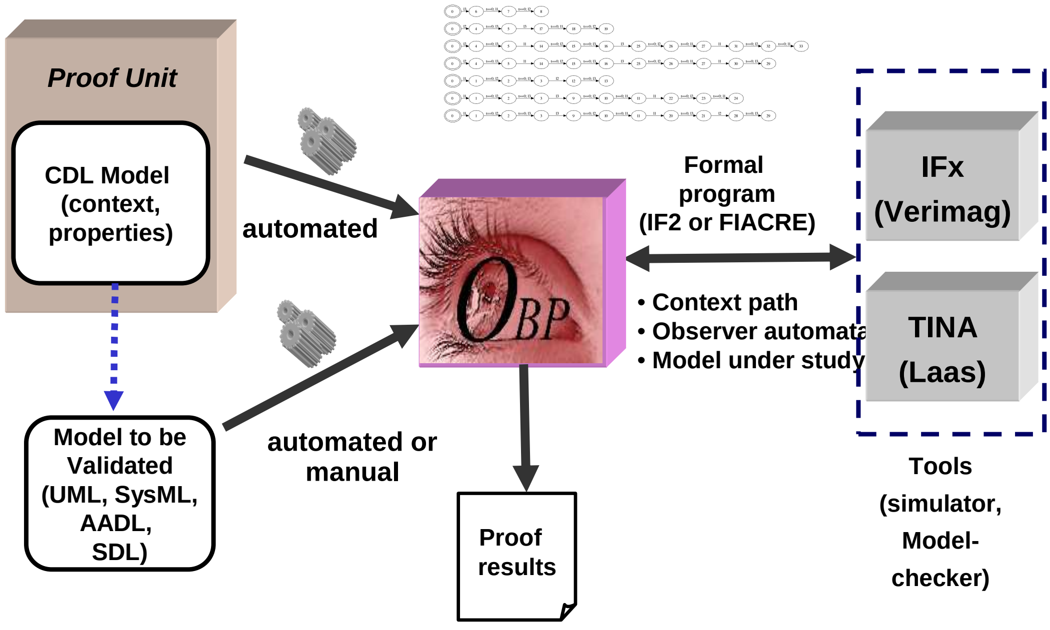
Understanding verdict for building diagnostics

Barrier : complexity

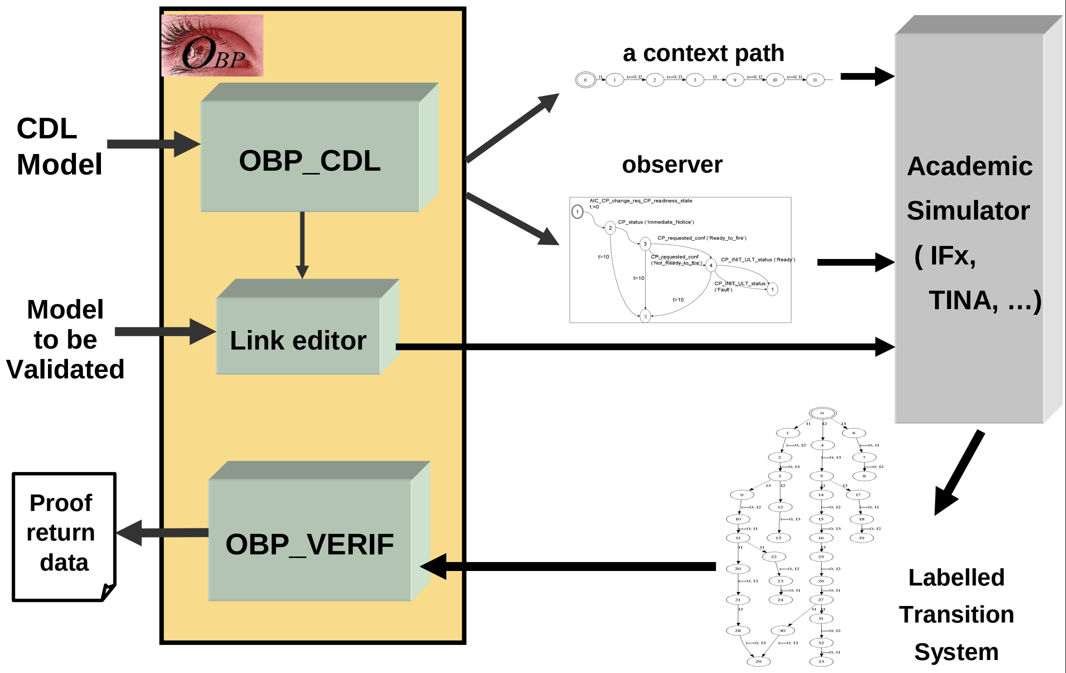


Combinatorial explosion induced by the internal complexity of the software to be validated.

Toolset : Observer-Based Prover
(Eclipse Plugin, EPL licence, TopCased project)



Generation, simulation, accessibility analysis



OBP generates the observer automata from the properties.

Each generated context path is transformed into an IF2 or Fiacre automaton.

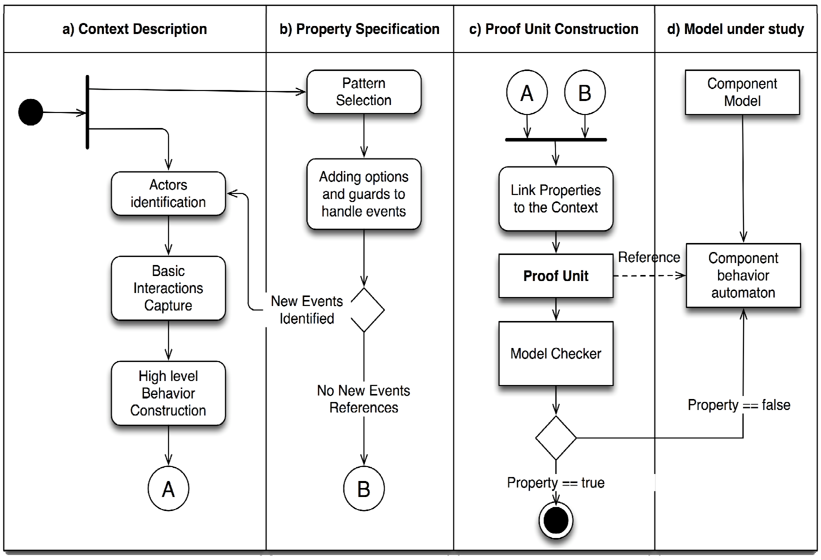
This path is composed with the Model to be validated and the observer automata by the IFx simulator or TINA model-checker.

Each property must be verified for all paths.

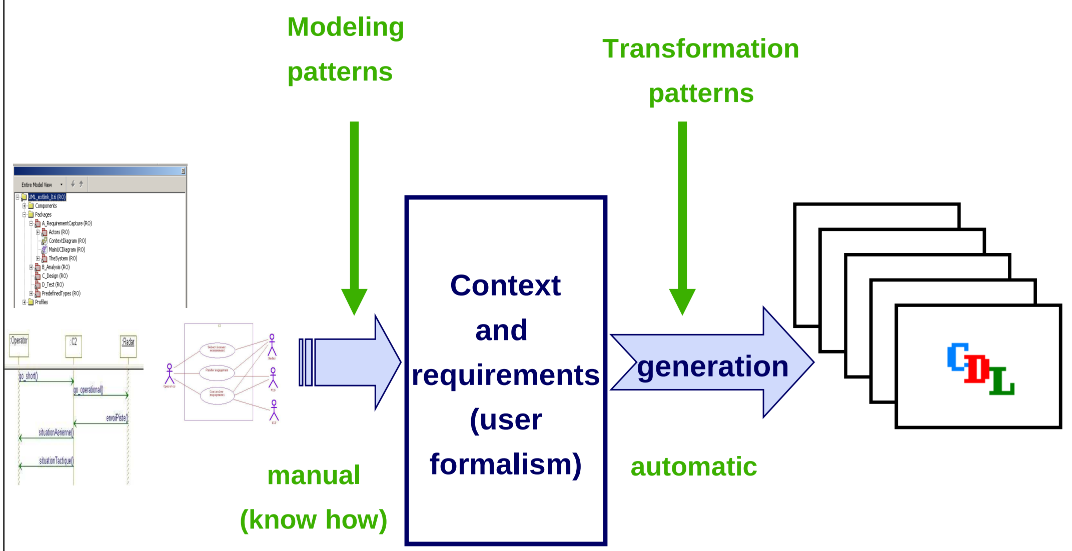
The accessibility analysis is carried out on the result of the composition between a path, a set of observers and the model.

If there is a *reject* state reached of a property observer for one of paths, then the property is considered as false.

Methodology : process



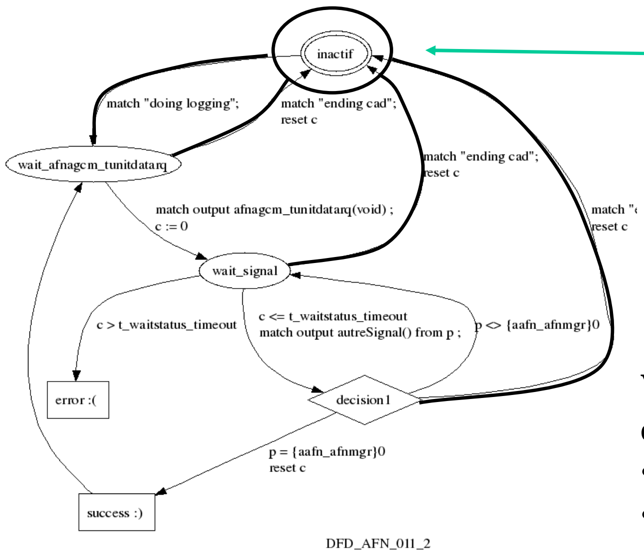
CDL model generation



An idea is to generate CDL models from high level data

The CDL models should be generated from

L'exigence DFD_11_2



Activation
(généré automatiquement)

Vérifiée sur les 24 chemins du contexte:

- **87279 états au total et**
- **96712 transitions**