Context Aware Model Exploration with OBP tool to Improve Model-Checking

Philippe Dhaussy[1], Frédéric Boniol[2],

Jean-Charles Roger[1]. Luka Leroux[1]

(philippe.dhaussy@ensta-bretagne.fr)

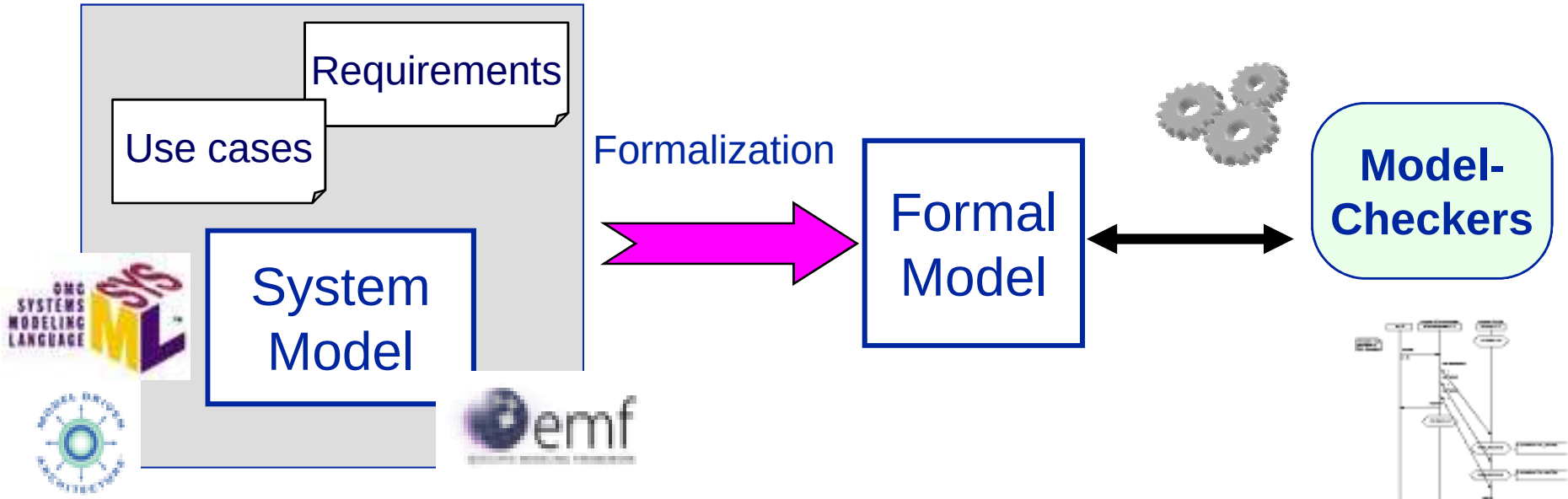
[1]: UEB, Laboratory Lab-STICC, ENSTA-Bretagne, Brest.

[2]: ONERA, CERT, Toulouse.

www.obpcdl.org

Motivation : Integration of formal methods in the engineering processes

Our research group is involved on testing of model-checking techniques for validation of software models of embedded systems.



Many barriers exist for an effective use of formal methods.

General idea of our work :

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

Context Aware Model Exploration with OBP tool to Improve Model-Checking

Currently, we study a technique of reduction of state explosion with context modeling to improve model-checking. It is the goal of this talk.

Motivation for Context aware verification

Proposition : Toolset OBP and CDL

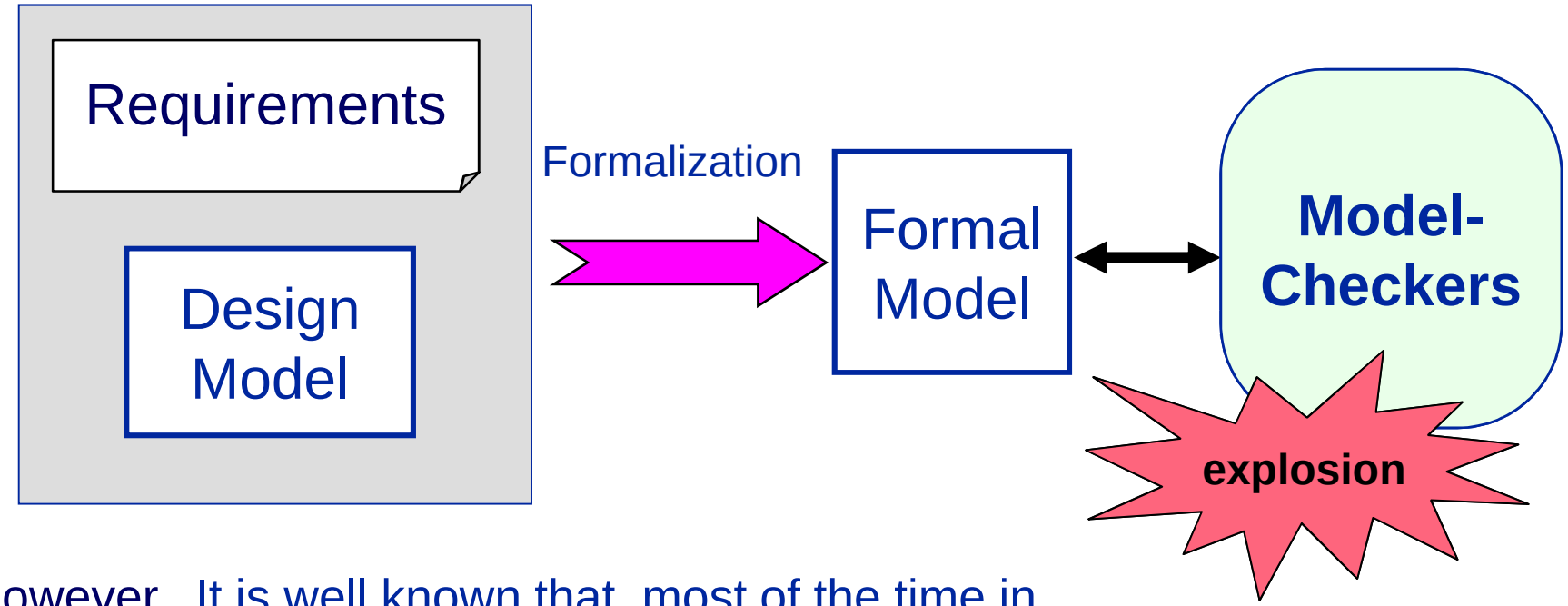
Context and requirement modeling (CDL)

Some results

Discussion and future work

Barrier : combinatorial explosion problem

For model-checking, several model checkers have been developed to help the property verification on software models.

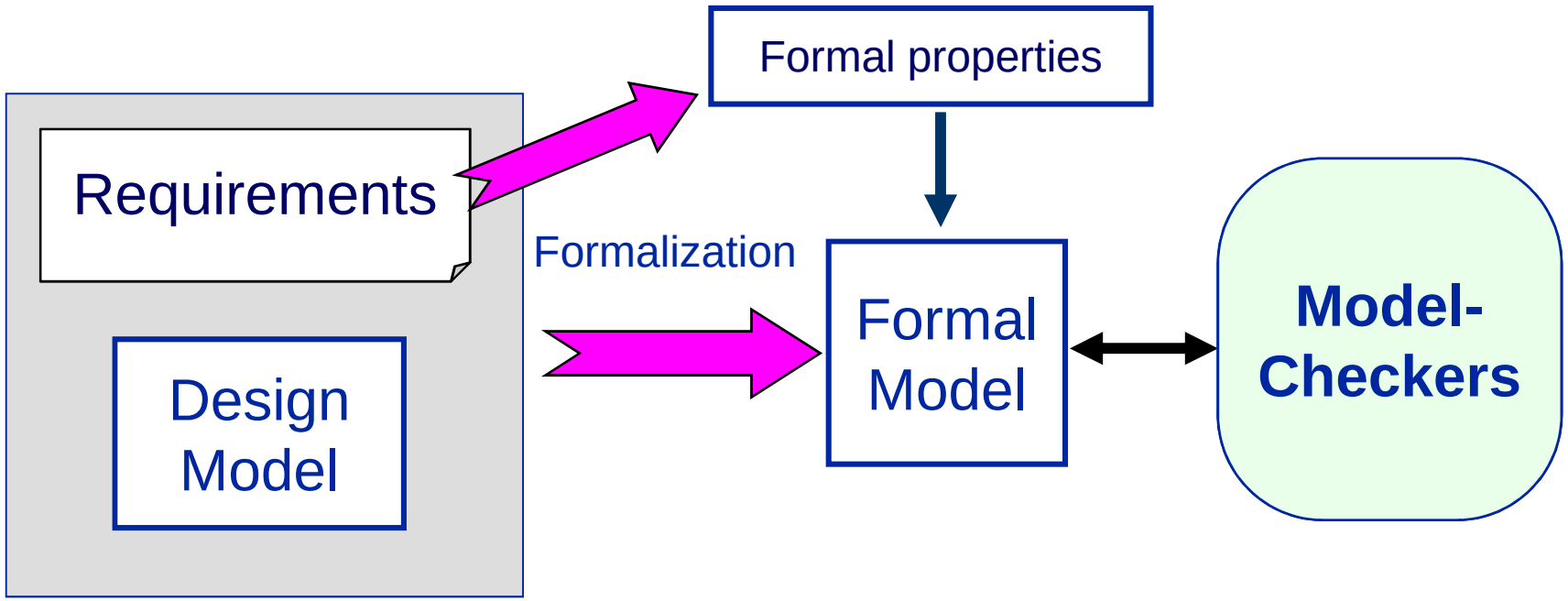


However, It is well known that most of the time in industrial context, we face state explosion problem.

It is an important issue that limits the application of model checking techniques

Barrier : Requirement formalization

Another barrier is the difficulty to formalize the requirements.



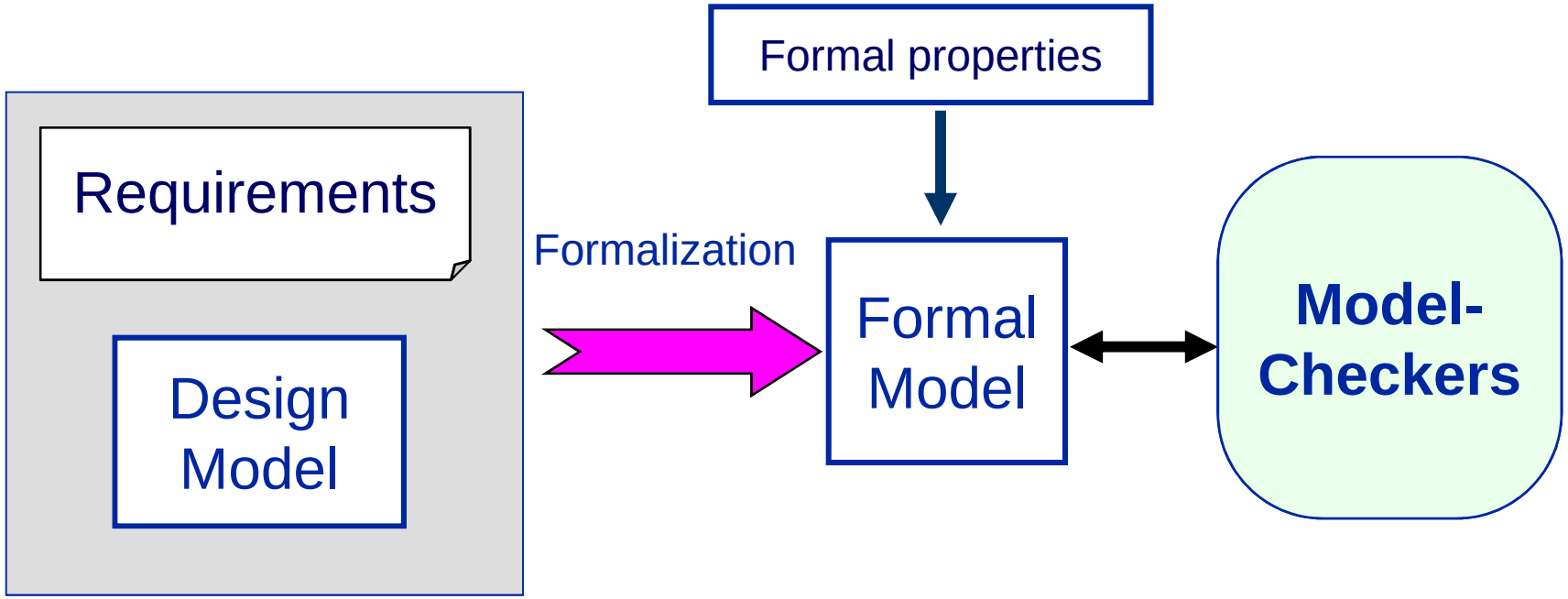
Because the semantic gap between

- the system model to be validated and
- the formal model needed for the verifications.

Barrier : Formal property expression

About requirement formalization :
the difficulty concerns property expression.

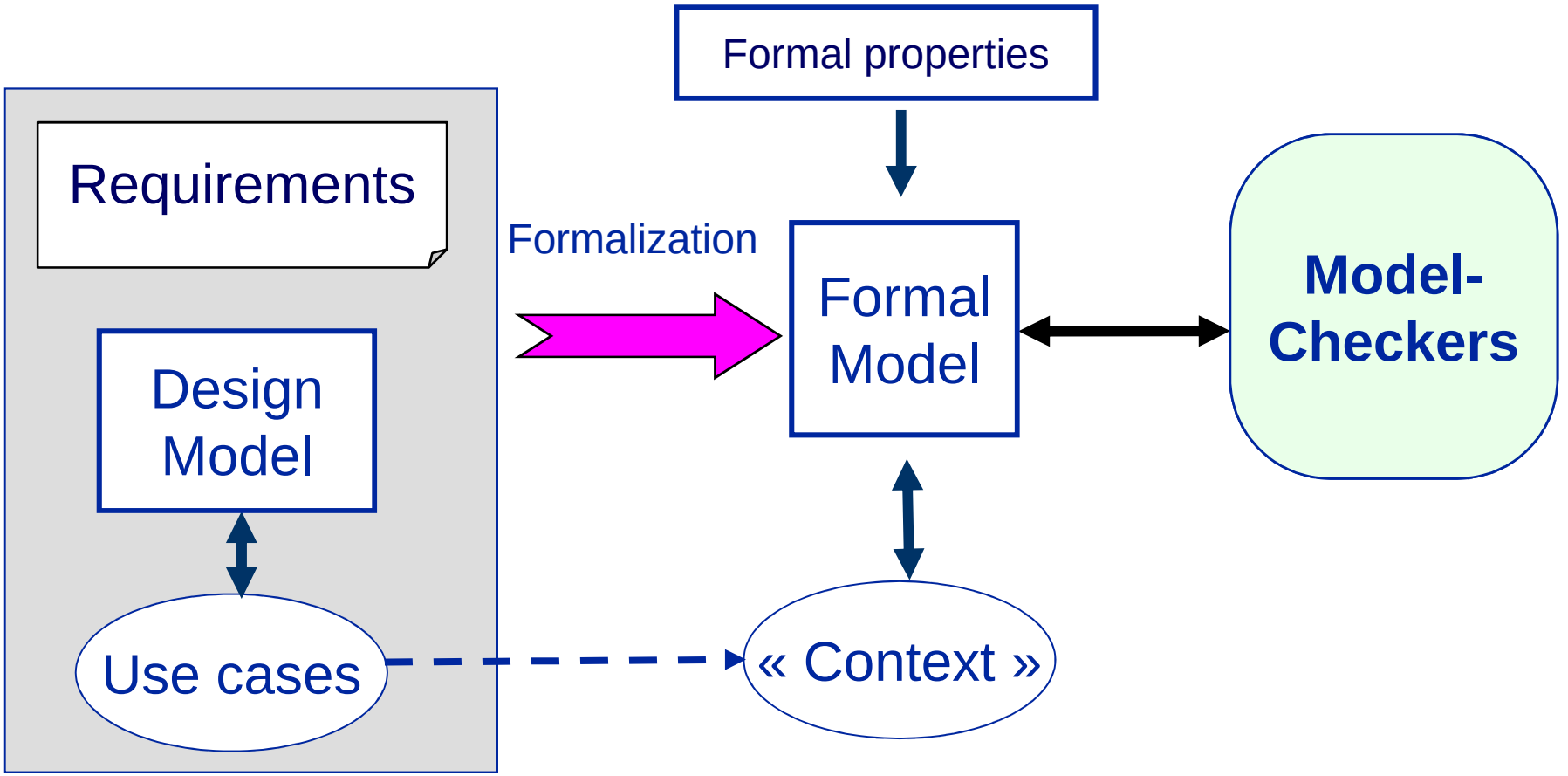
Temporal logic formula (as LTL or CTL) are not very acceptable in industry.



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



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

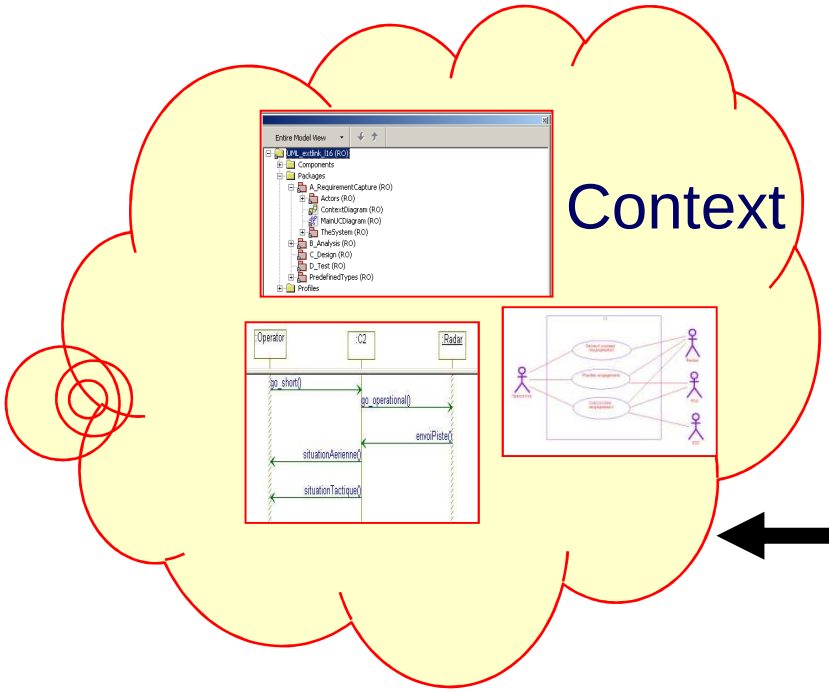
So, it's not necessary to verify them over all the environment scenarios.

And we propose to identify and to formalize these "contexts".

Context expression

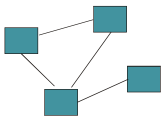
Contexts : represent behaviors of the environment

They correspond to well-defined operational phases such as, for example, initialization, reconfiguration, degraded modes, error scenarios, etc.



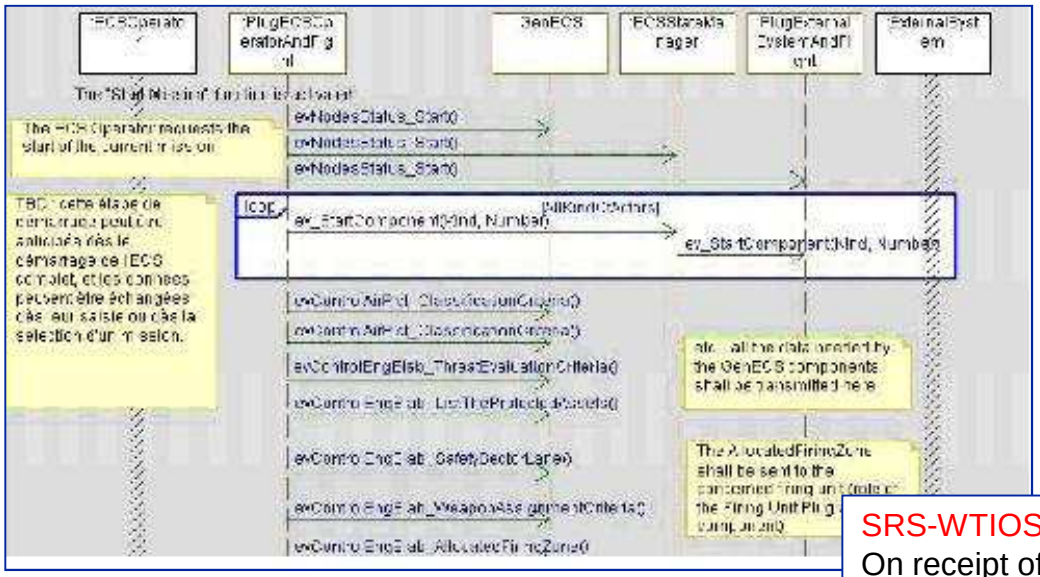
But, in reality, ,in the specifications, useful information about the context execution is very often implicit or disseminated in several documents

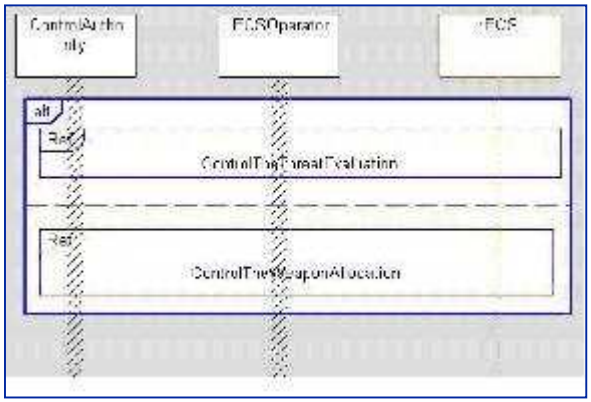
Model to be Validated

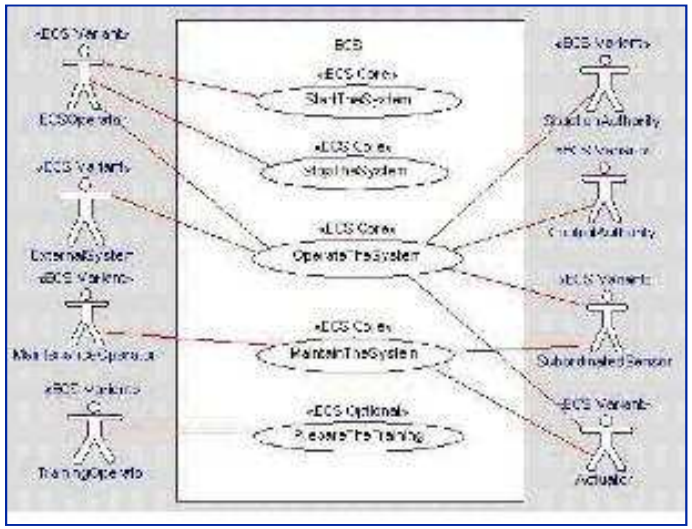


Context expression

For example, for context expression we need documents describing use cases, message or interaction diagrams and requirements.



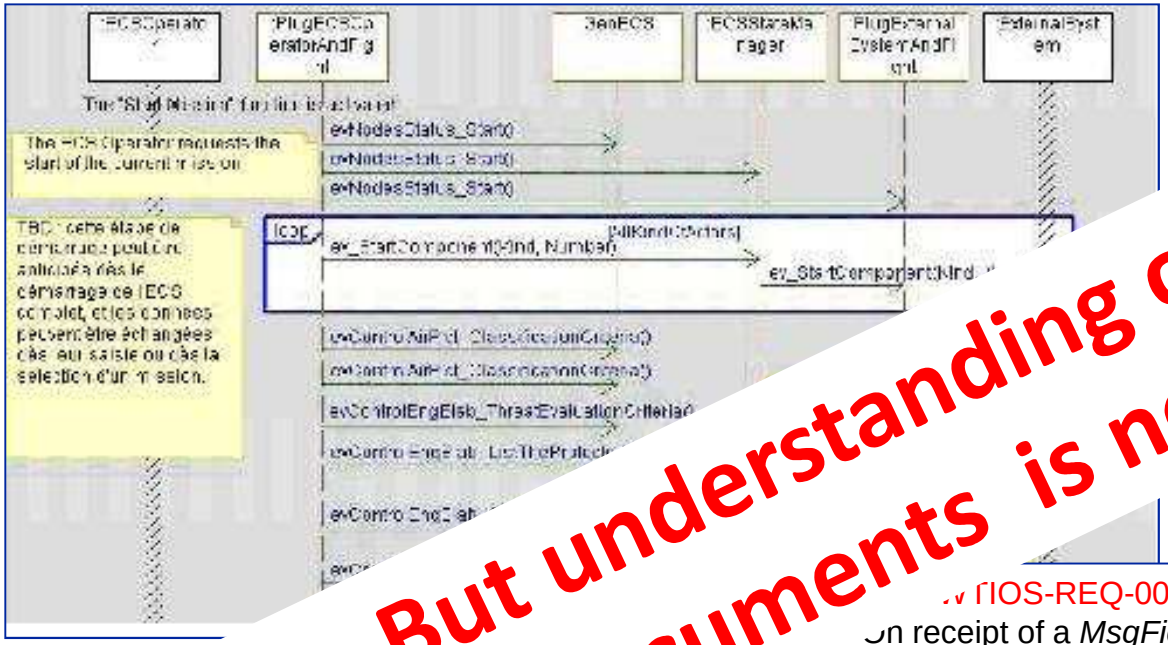


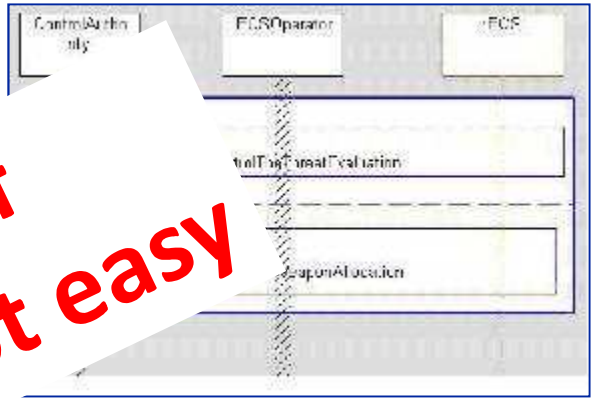


SRS-WTIOS-REQ-004

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

Context expression



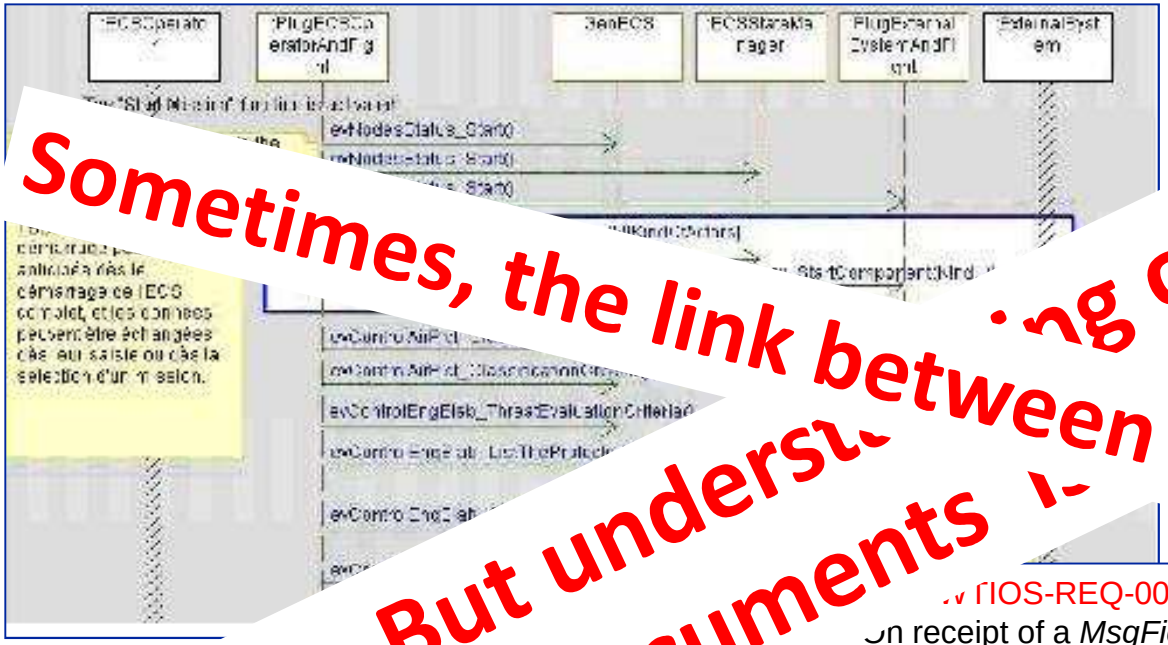


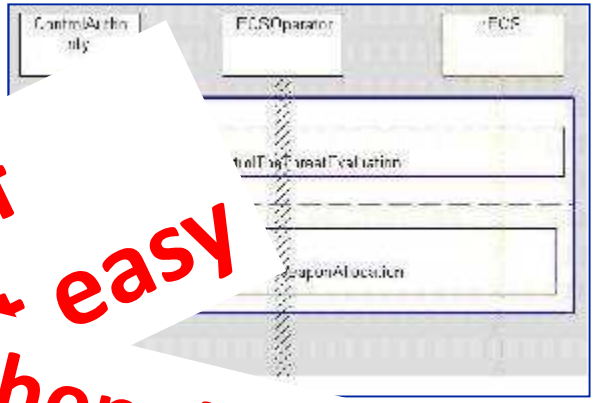
But understanding of these documents is not easy

IOS-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

Context expression





Sometimes, the link between them is implicit

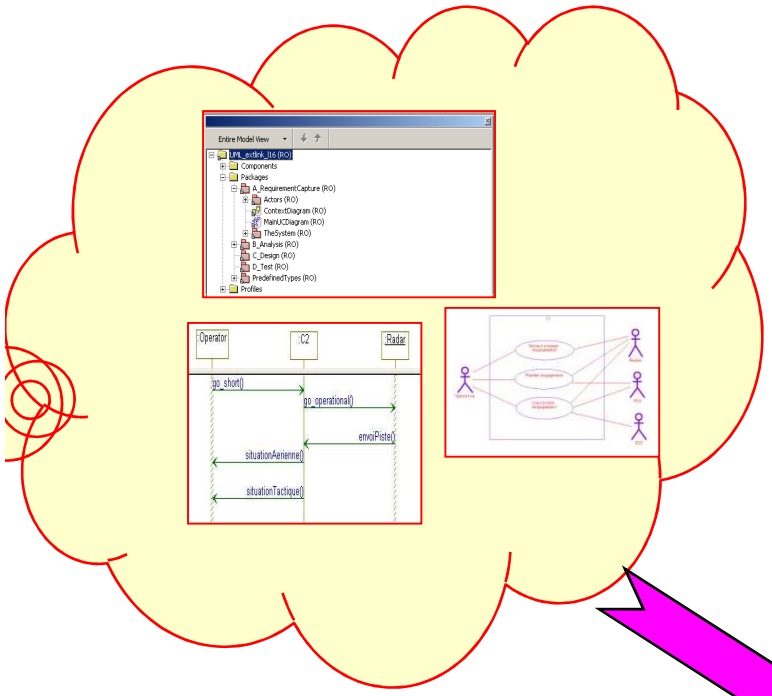
But understanding of these documents is easy

[V_IOS-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

Context expression

So, we study a method to support the formal specification of these contexts in which the model will be validated

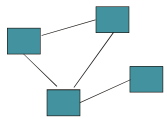


In these conditions, a model is tightly synchronized with its environment.
It is a way to circumvent the problem of combinatorial explosion

Formalization

Context (scenarios, properties)

Model to be Validated



In our approach :
we make *a first strong hypothesis*

In domain of embedded systems : we focus on deterministic systems with bounded environment

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 not formally justified in our work

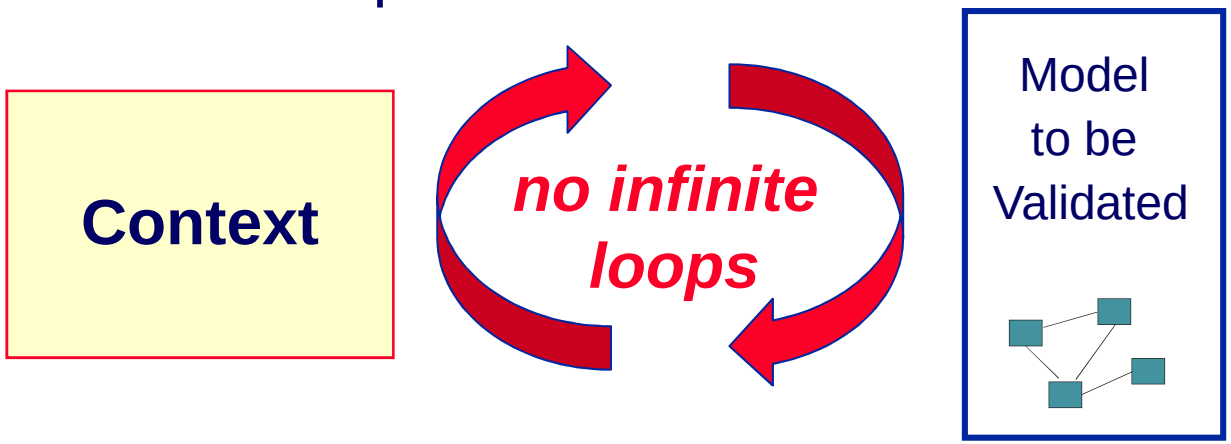
But the essential idea is :

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

Second hypothesis

A second hypothesis expresses that

- The contexts we describe are finite.
- There are no infinite loops in the interactions between the system and its environment.
- It is particularly true for instance with command systems or communication protocols.



And we focus our technique to verify safety and bounded liveness properties.

A pragmatic approach

So, we adopt a pragmatic approach for integration in engineering processes.

Currently, we focus on :

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

Context Aware Model Exploration with OBP tool to Improve Model-Checking

Motivation for Context aware verification

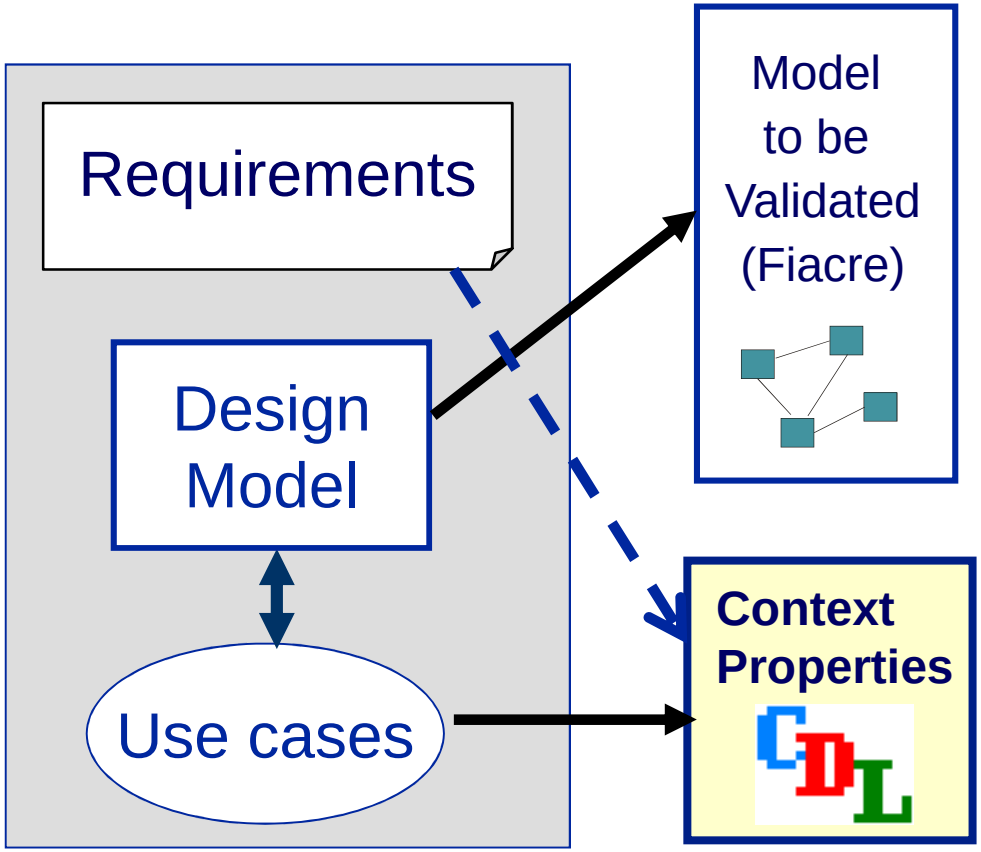
Proposition : Toolset OBP and CDL

Context and requirement modeling (CDL)

Some results

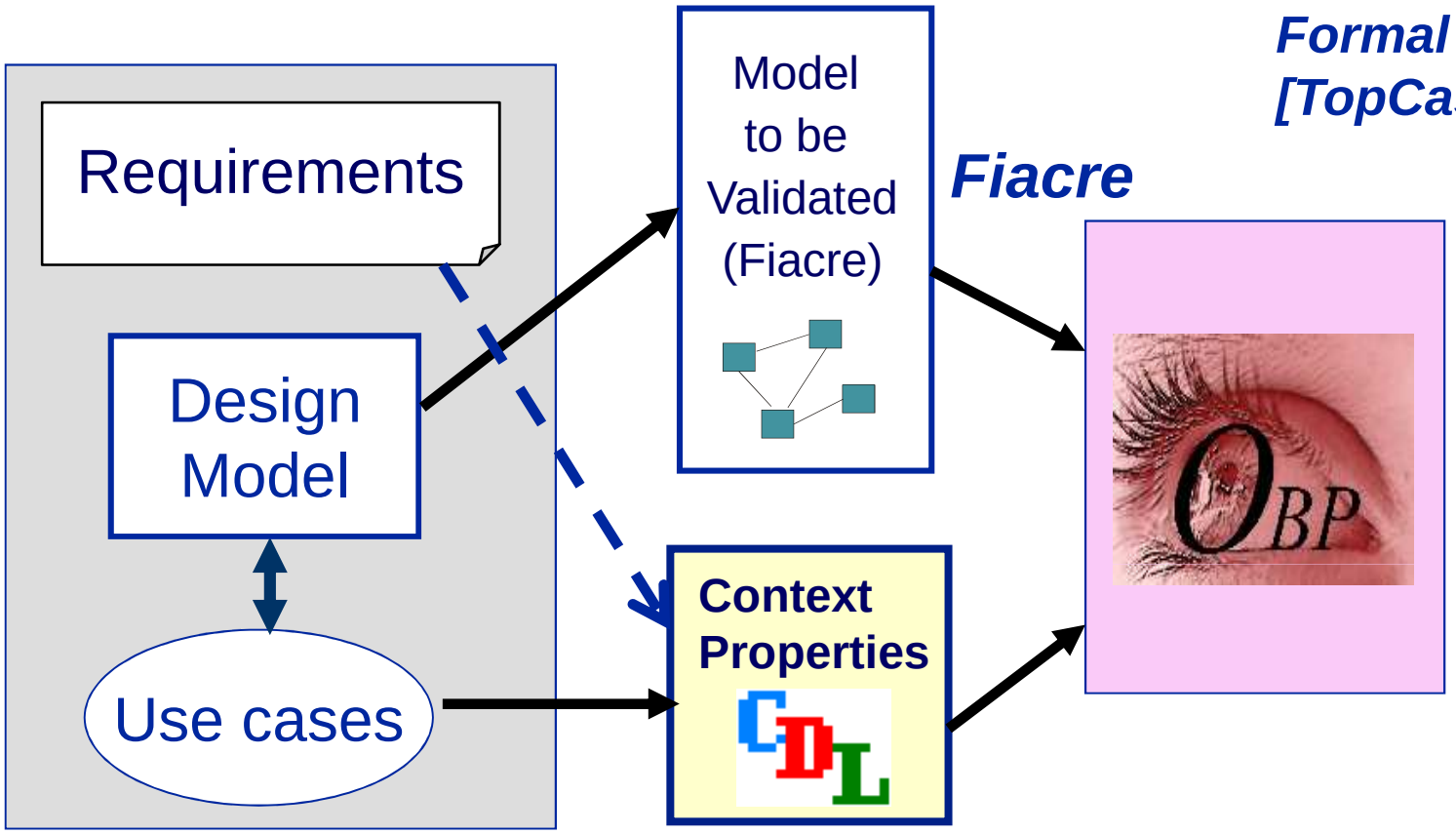
Discussion and future work

Our experimented tool : OBP and CDL



We implement our approach with a tool and a language
- a *Domain Specific Language* named Context Description Language.

Our experimented tool : OBP and CDL

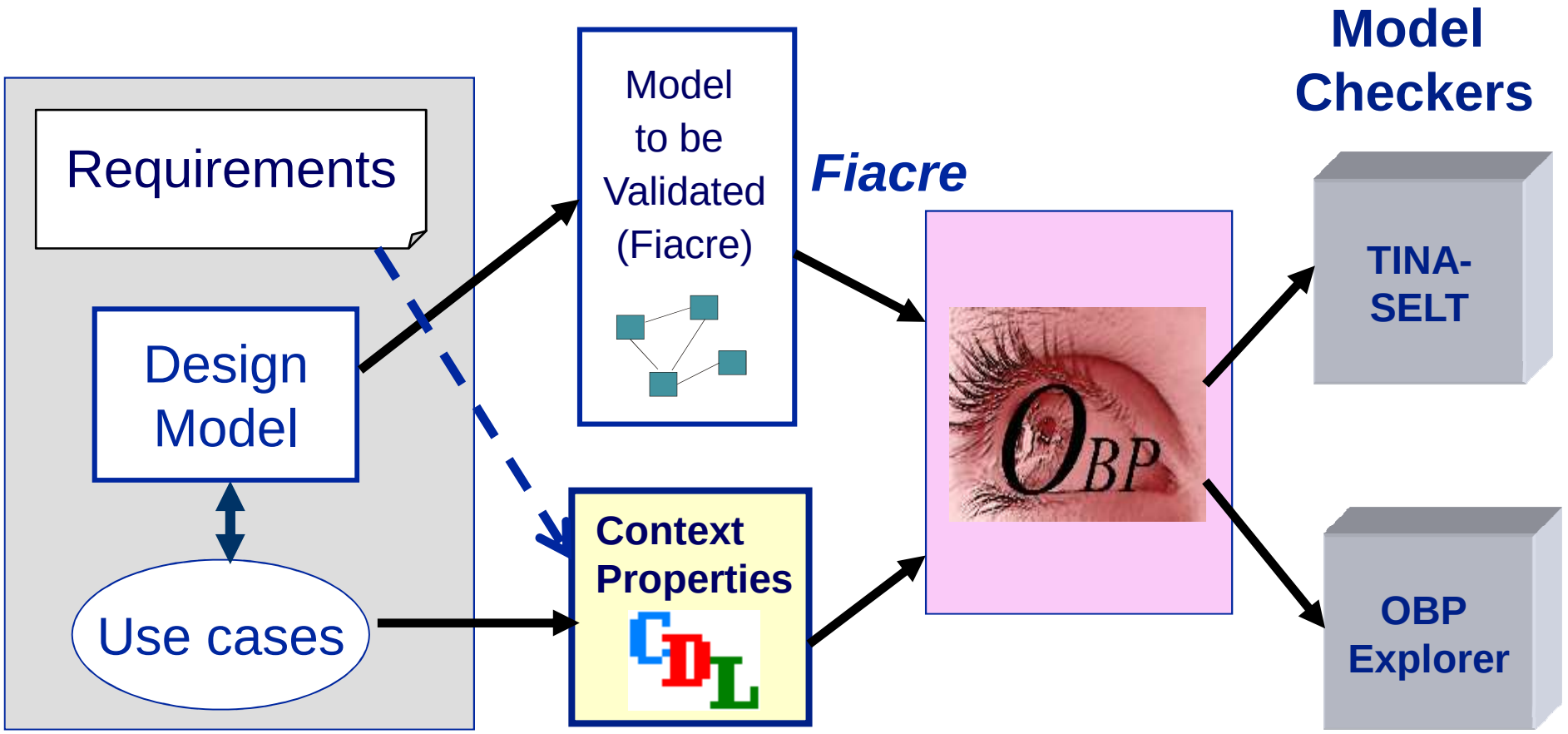


Formal language [TopCased project]

We implement our approach with a tool and a language

- a *Domain Specific Language* named Context Description Language.
- **a tool named Observer Based Prover.**
- **OBP imports models described with FIACRE language.**

Our experimented tool : OBP and CDL

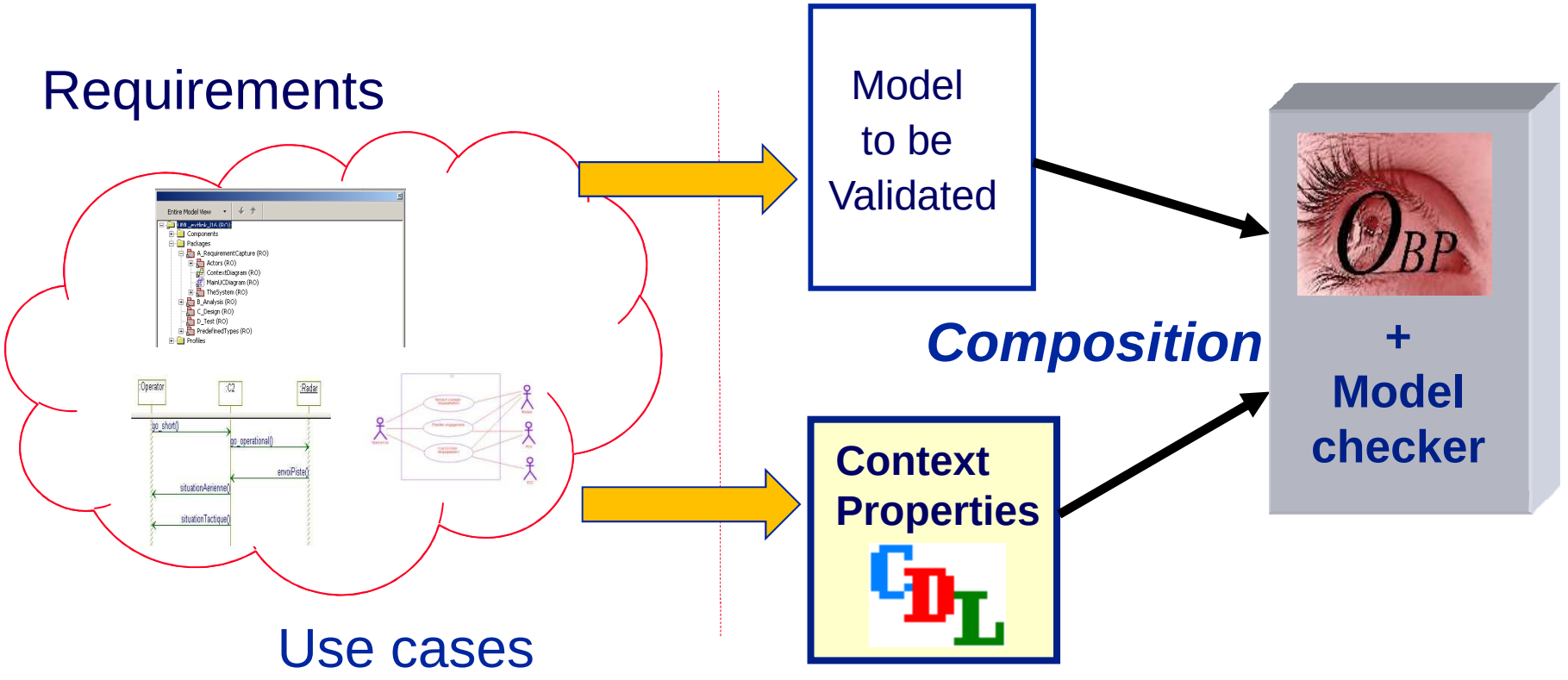


We implement our approach with a tool and a language

- a *Domain Specific Language* named Context Description Language.
- a tool named Observer Based Prover.
- OBP imports models described with FIACRE language.
- **Currently, OBP can be connected to TINA or OBP Explorer**

Methodology for reduction

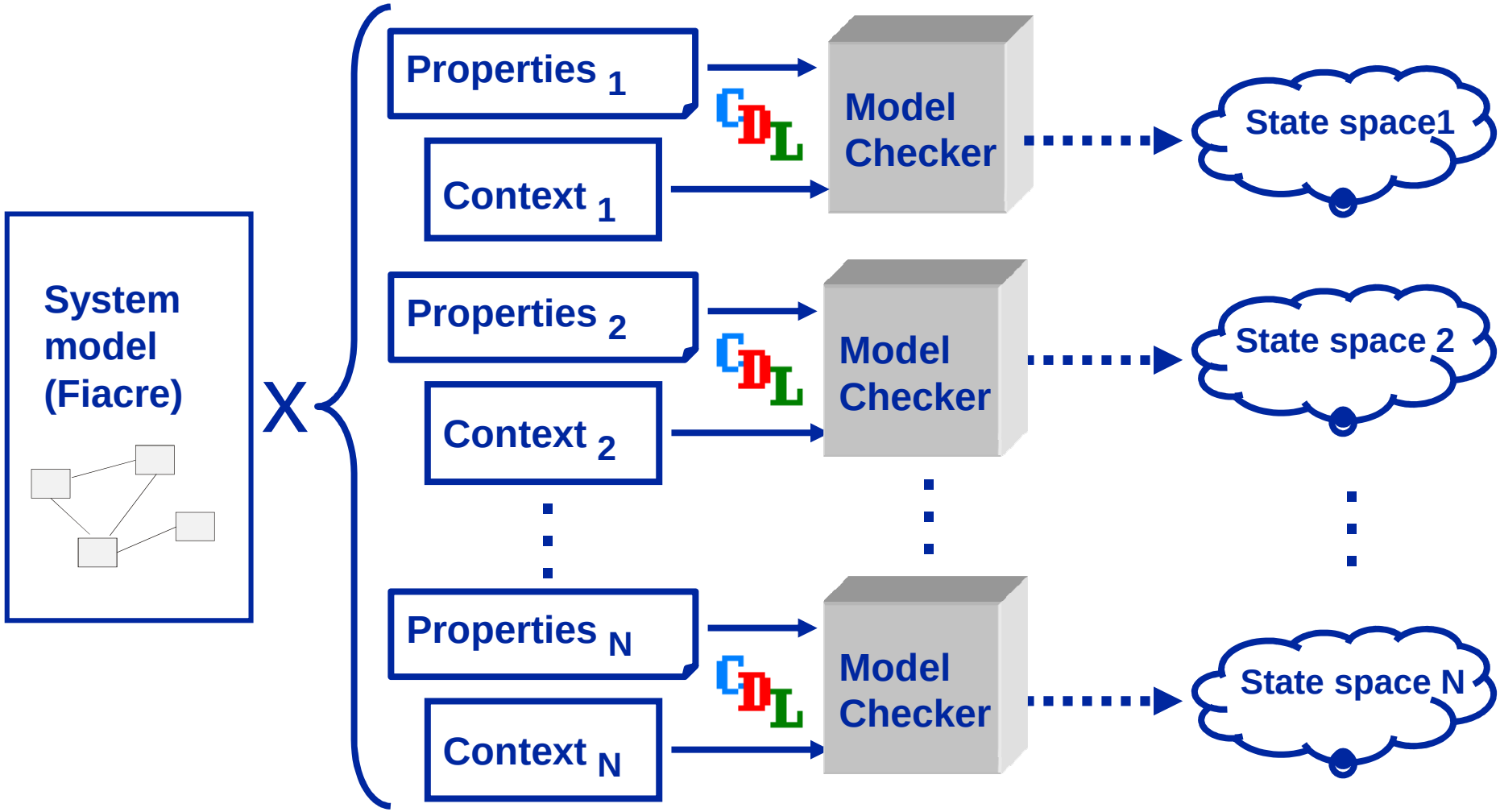
We propose to restrict model behavior by composing it
with an environment that interacts with the model.
This technique can reduce the complexity of the exploration during verification.



With the context specification, the reduction is computed in two steps.

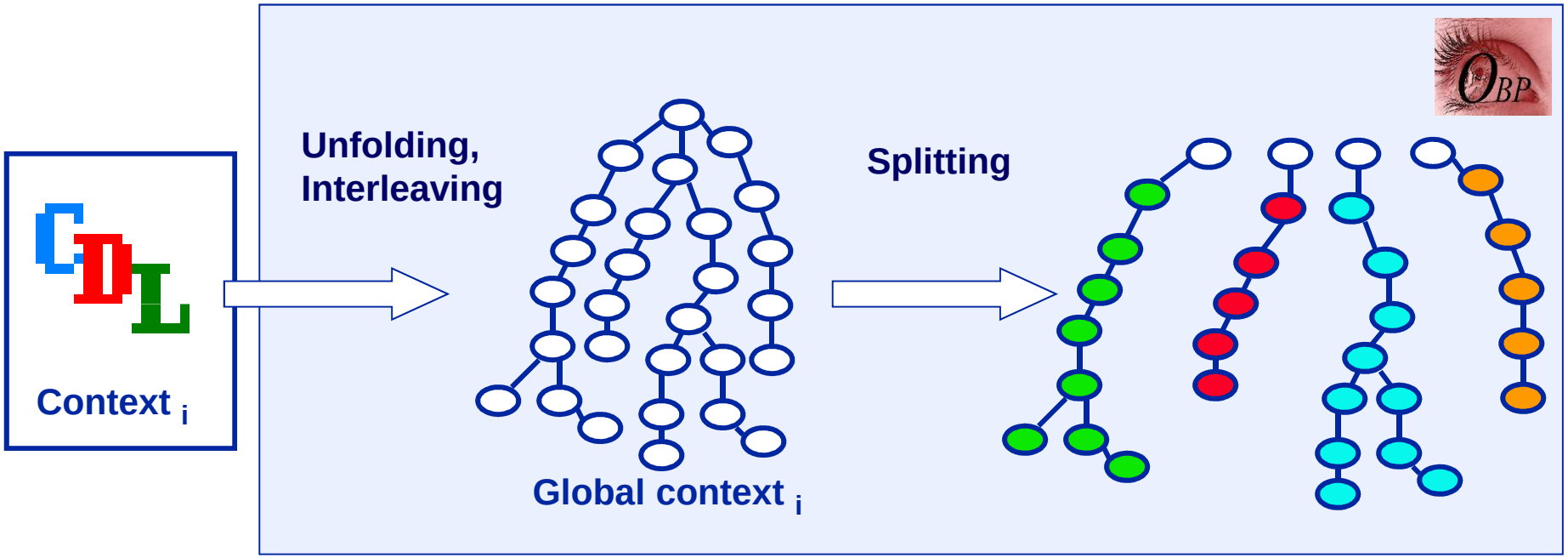
Step 1: Contexts are first identified by the user

The contexts correspond to patterns of use of the component being modeled.
We restrict the behavior with different configurations to check specific sets of properties

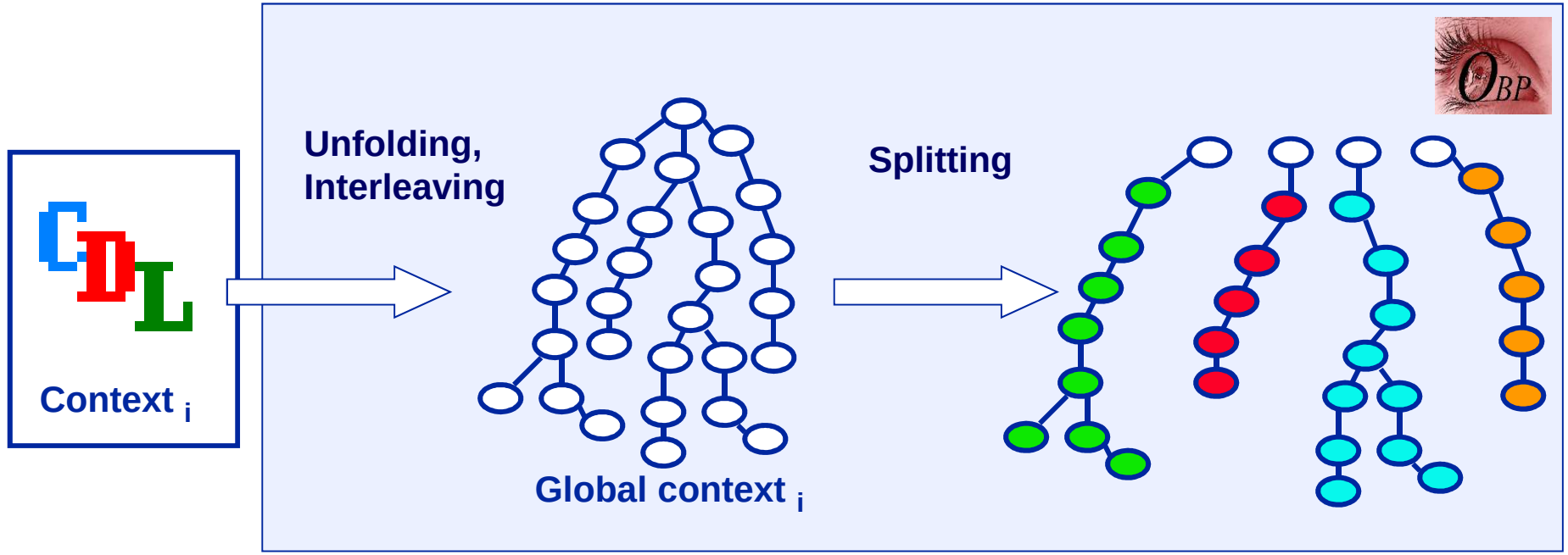


Step 2: Automatic context splitting

The second idea is to automatically split each identified context into a set of smaller sub-contexts.

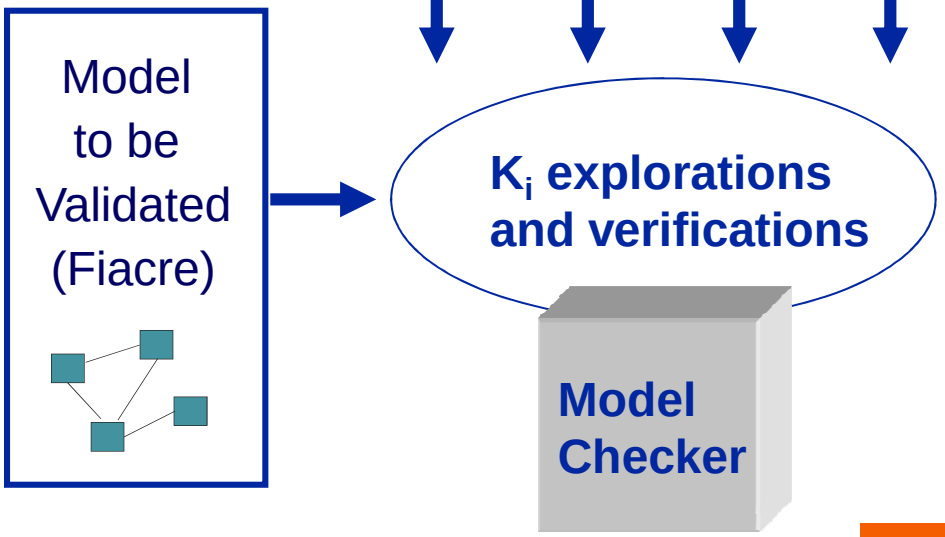


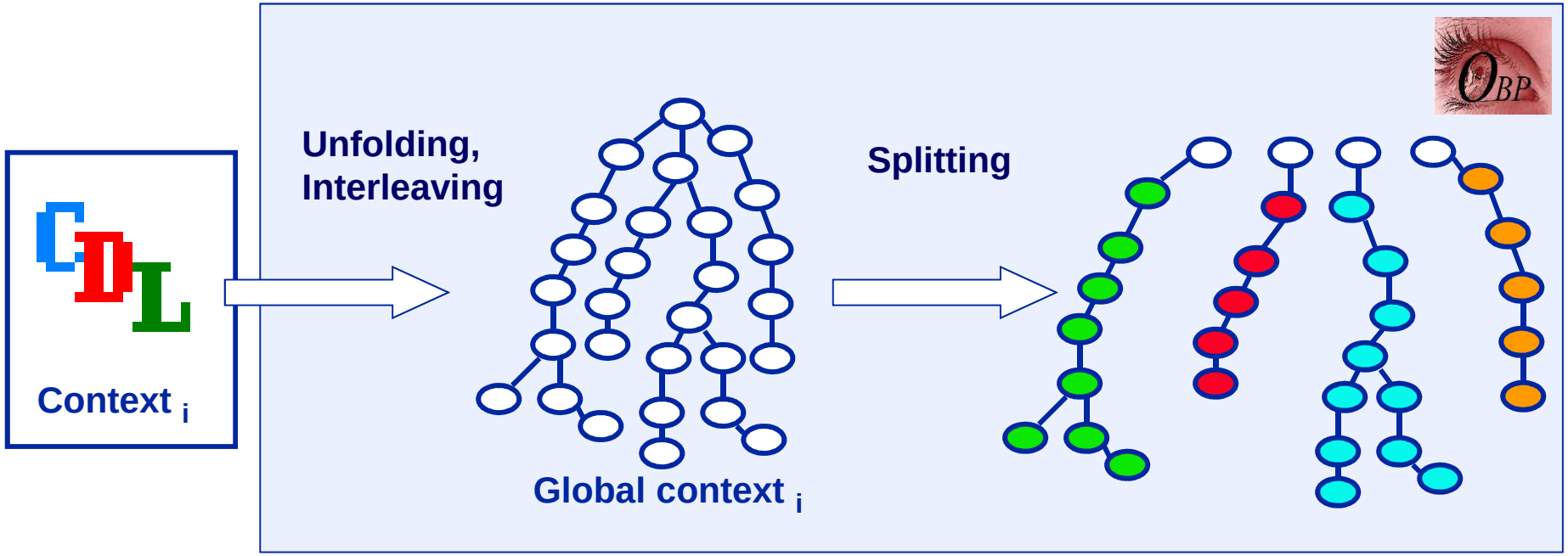
Contexts: Finite and acyclic



Set of K_i sub-contexts

Actually, we transform the global verification problem for one context into of several smaller verification sub problems.

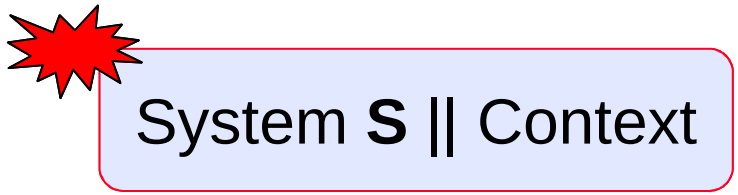




For that, we implemented a recursive splitting algorithm in our OBP tool.

```
explore_mc (model, context ¡, pty, d)
{   //---- exploration -----
    lts = explore (model, context ¡ );
    //---- model-checking -----
    if  lts != error  model_check (lts, pty);
    else
    {  set_c = split (context, d); // splitting
       for k : 0 to sizeof set_c
           explore_mc (model, set_c_k, pty, d);
     }
}
```

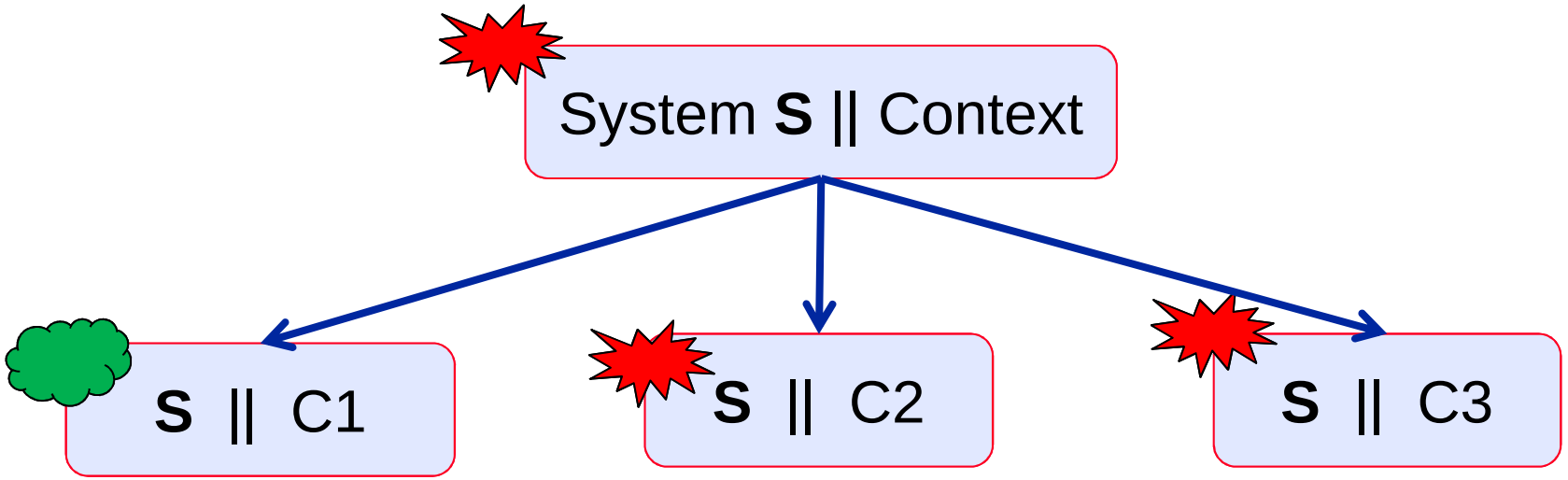
Automatic context split



Each context is represented by acyclic graph.

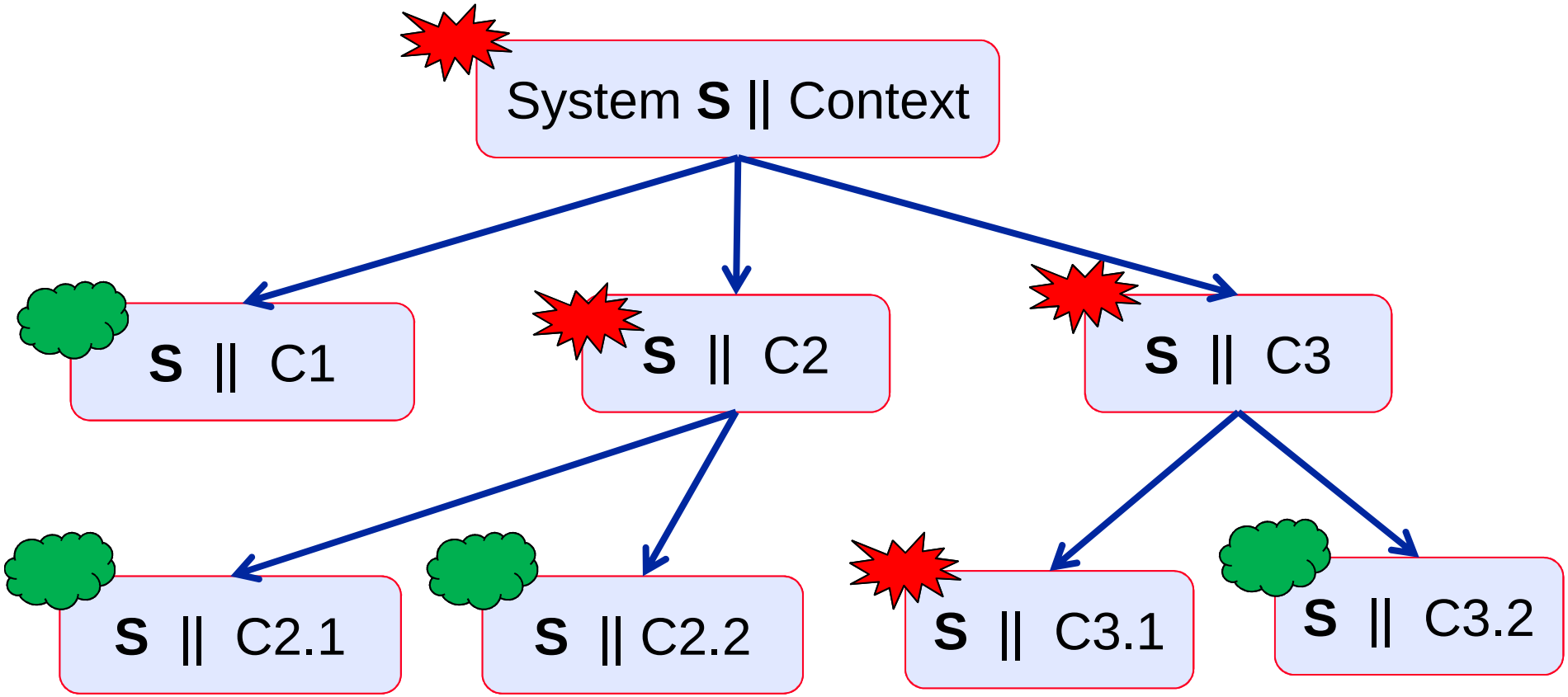
This graph is composed with the model for exploration.

Automatic context split



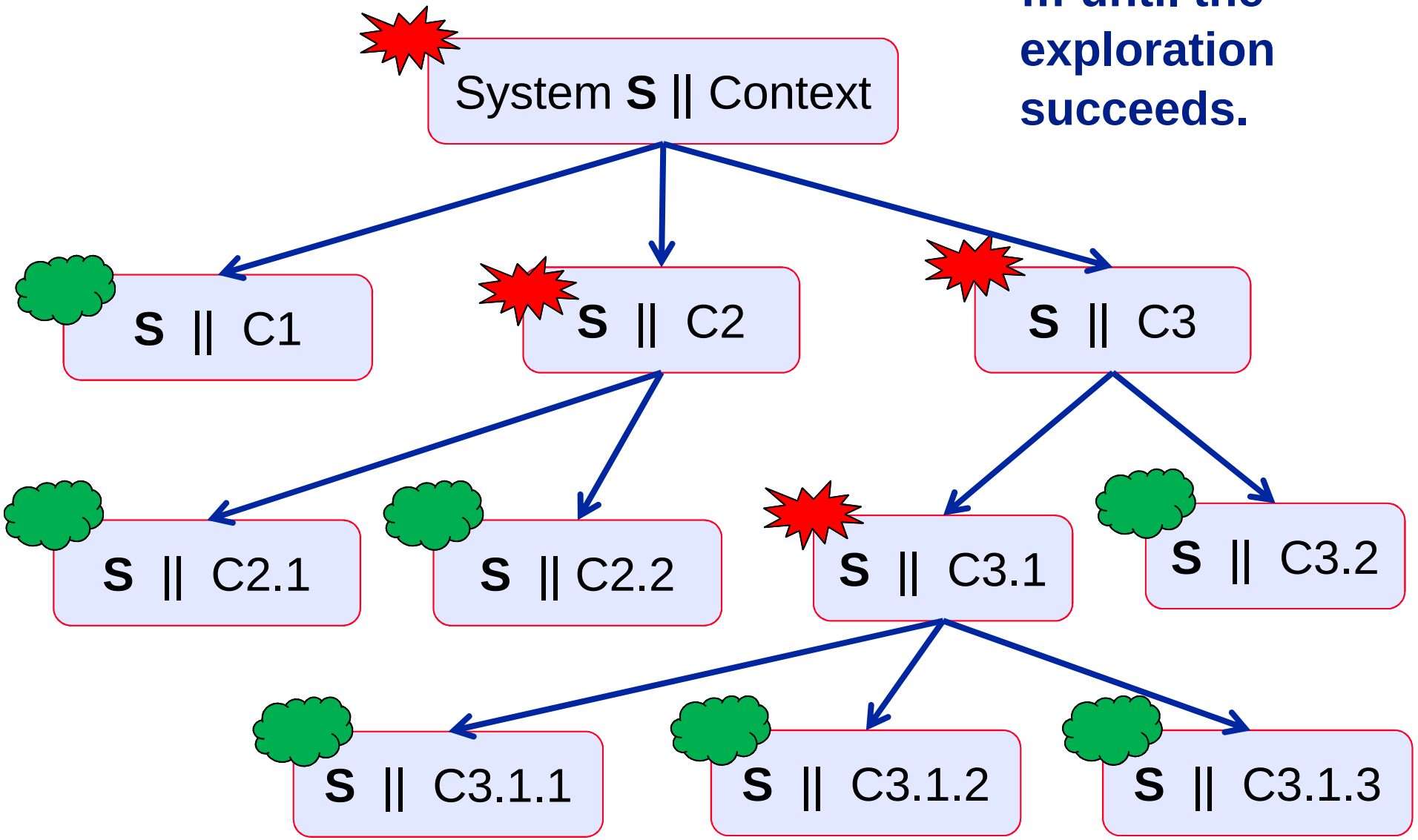
In case of explosion, this context is automatically split into several parts taking into account a parameter for the depth in the graph for splitting ...

Automatic context split



Automatic context split

… until the exploration succeeds.



Context Aware Model Exploration with OBP tool to Improve Model-Checking

Motivation for Context aware verification

Proposition : Toolset OBP and CDL

Context and requirement modeling (CDL)

Some results

Discussion and future work

Context Description Language

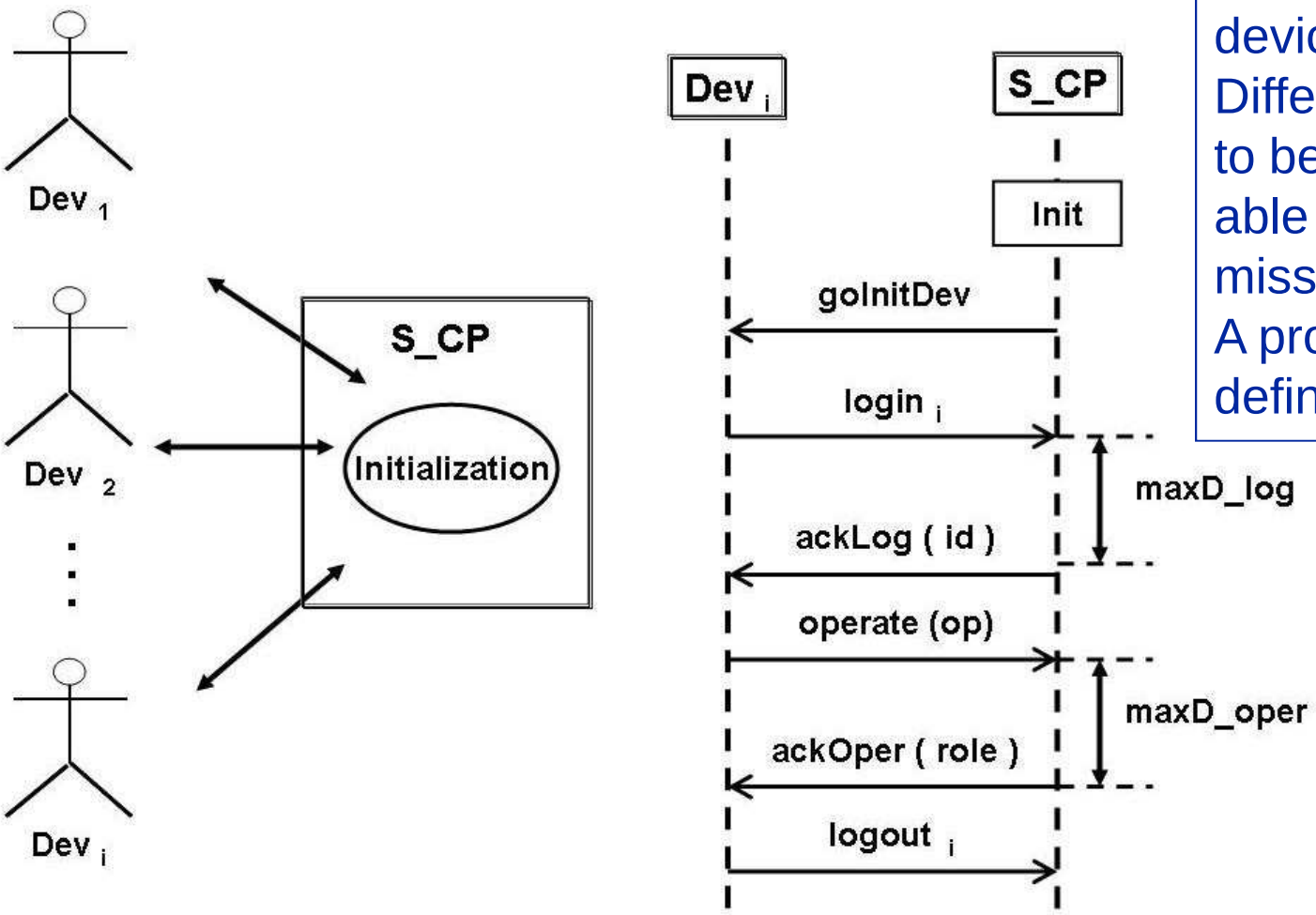
A CDL model is composed by :

- **Activity and sequence diagrams**
- **With a textual and graphical representation**

The contexts are described in hierarchical manner : Each context can be unfolded to a finite and acyclic graph.

CDL also contains description for properties as patterns and observers

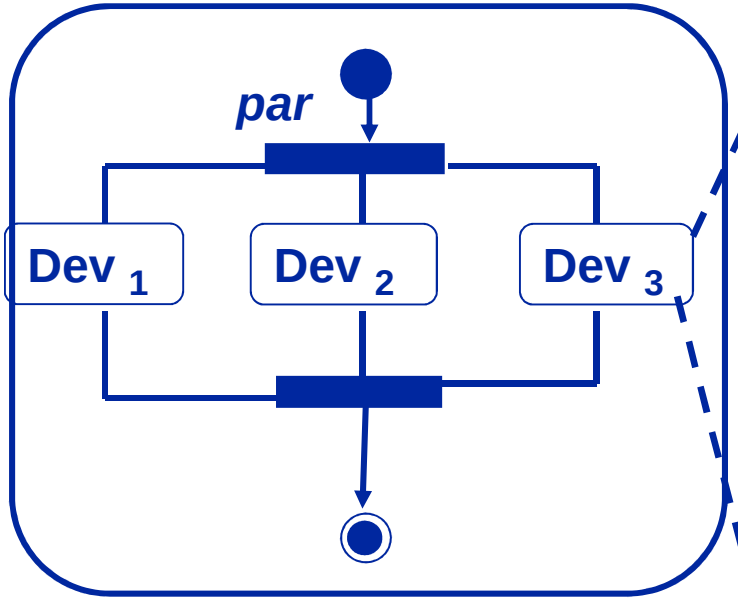
To present CDL: an example of simple case study



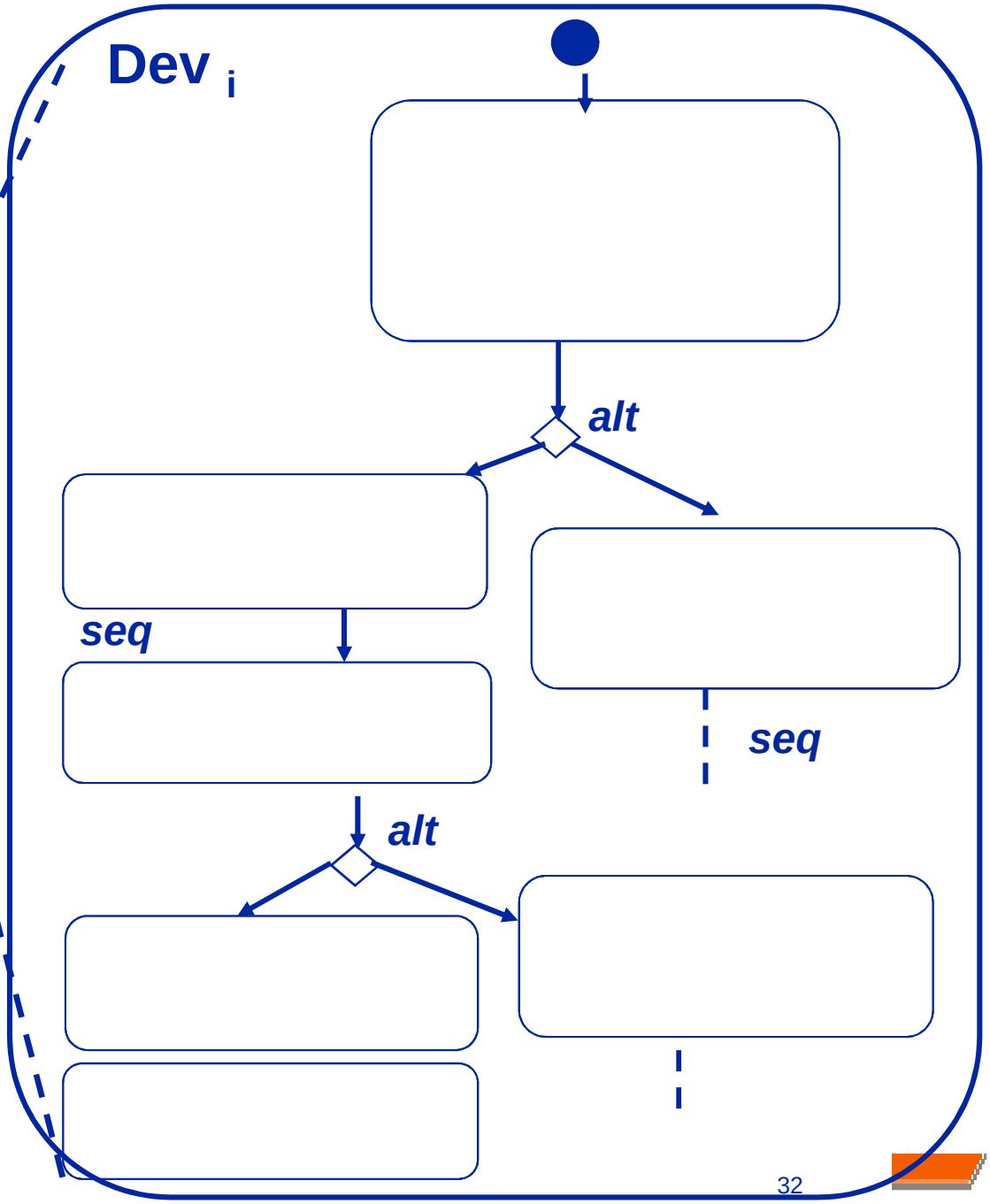
It's a software part of an industrial system.
This controller manages physical devices.
Different devices need to be registered to be able to operate missions.
A protocol has been defined for operations.

Context description : Graphical version

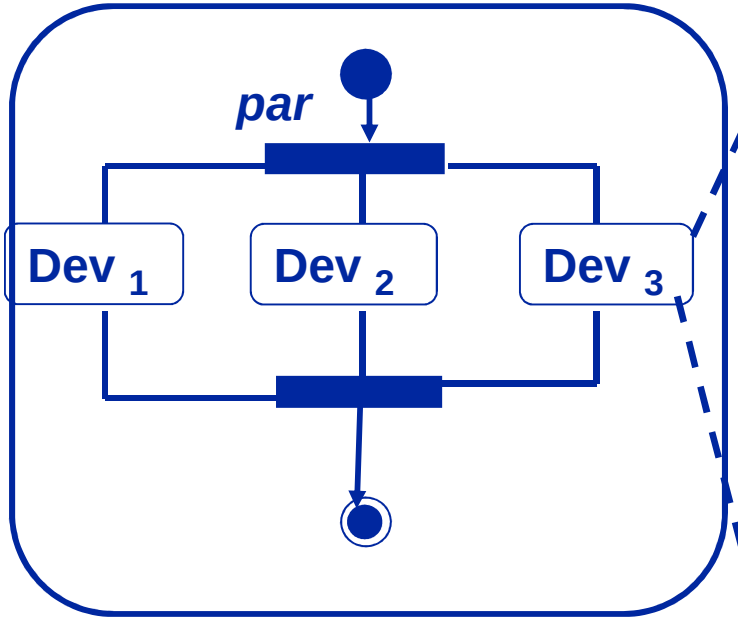
A CDL model is constructed with different levels

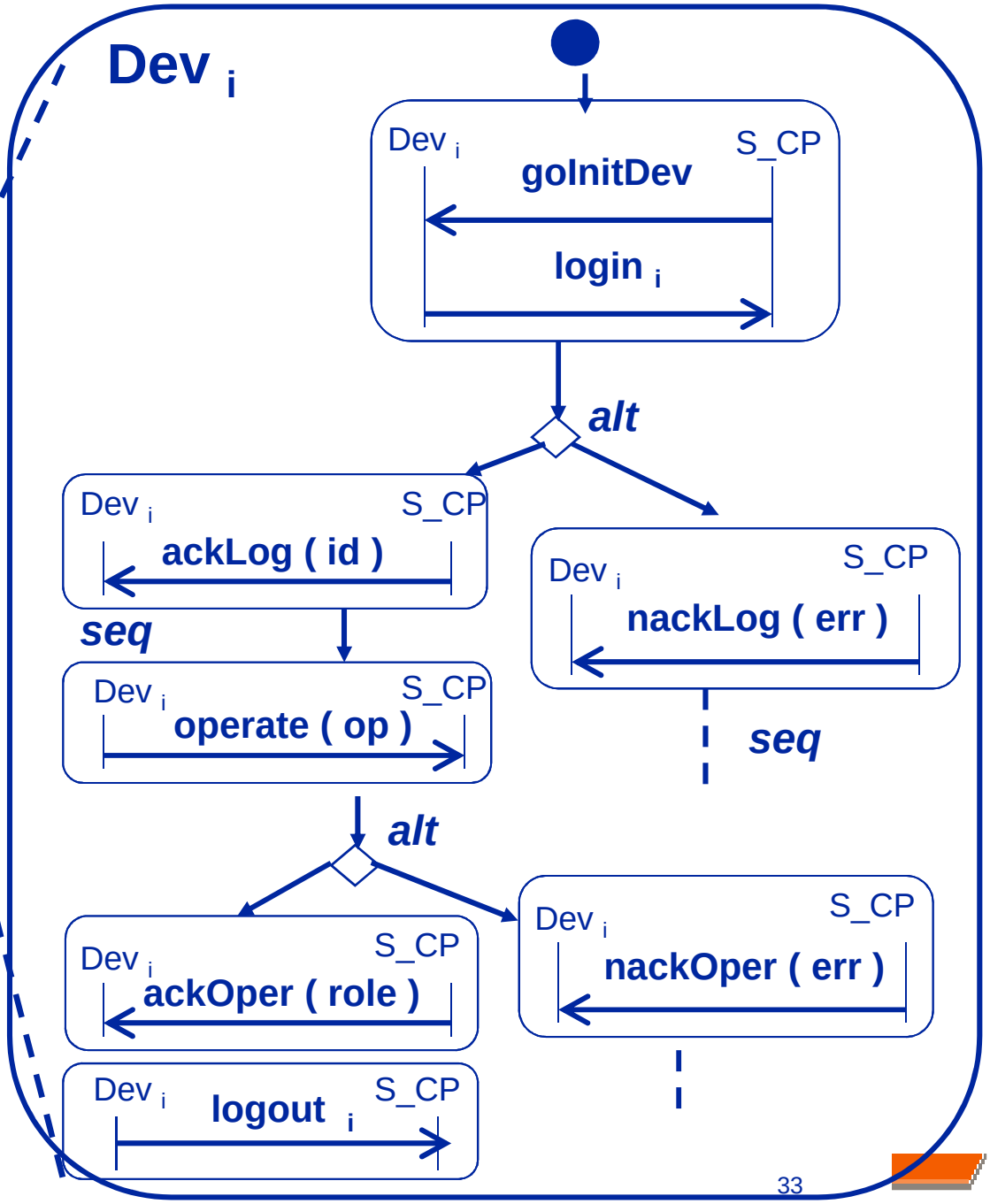


On first level : a set of use case diagrams is described by activity diagrams :
Either alternative between several executions or a parallelization of several executions is available.



Context description : Graphical version





Scenario diagrams are organized with sequences and alternatives.
Each scenario is fully described by sequence diagrams.

CDL Example (Textual)

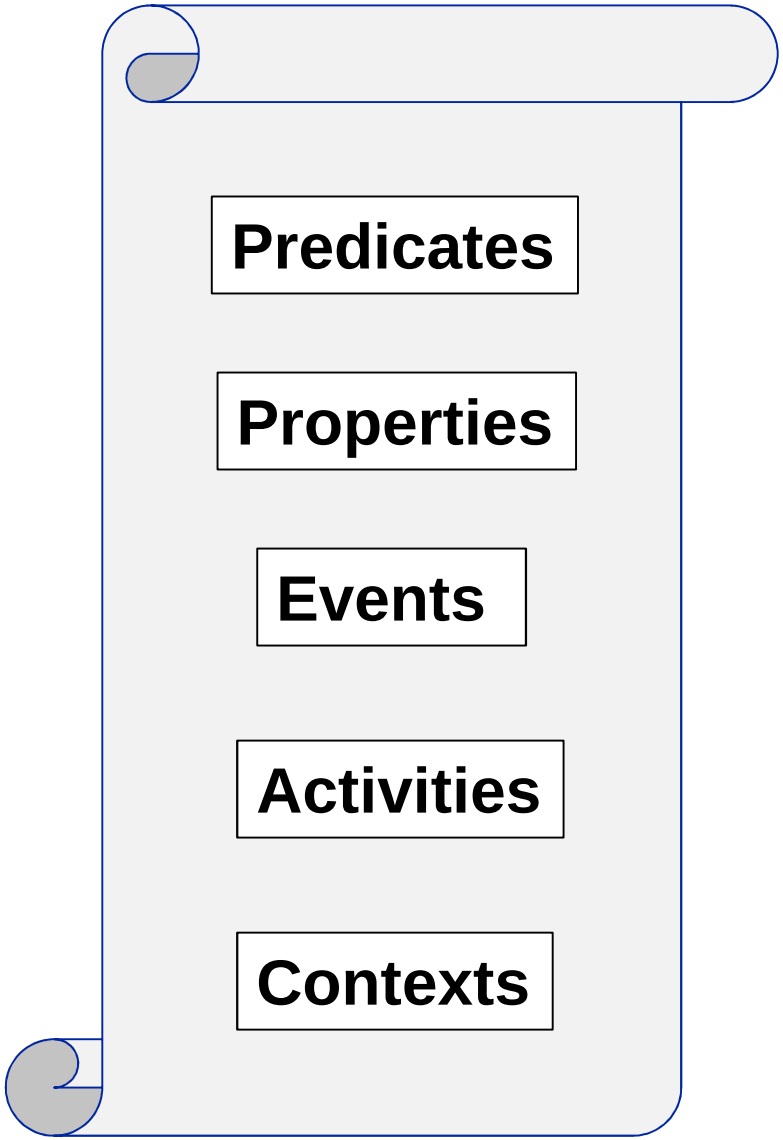
```
cdl context is {
        main is { dev1 || dev2 || dev3 }
}

activity dev1 is {
        { event goInitDev; event login1 };
        {
            event ackLog ; event operate;  {
                { event ackOperate ; event logout1 }
                  []   event nackOperate …
                }
            [] event nackLog  …
        }
}
```

But we chose to develop a textual version of CDL because we want to generate CDL models in the future.

Syntax and semantics in [HASE'11] or **http:// www.obpcdl.org**

A CDL model



A CDL model describes :

- **a set of predicates and properties**
- **a set of events and activities**
- **a set of contexts**

Event : communications (asynchronous, synchronous)

Asynchronous sending

event e **is** { **send** m **from** {env}1 **to** {p}1 }
event e **is** { **send** m **from** {p}1 **to** {q}1 }

Asynchronous reception

event e **is** { **receive** m **from** {p}1 **to** {env}1 }
event e **is** { **receive** m **from** {p}1 **to** {q}1 }

An activity references events :

An event is an elementary asynchronous interaction : *send, receive*

Theses events are useful to specify observers as matched events.

Event : communications (asynchronous, synchronous)

Asynchronous sending

event e **is** { **send** m **from** env **to** p }
event e **is** { **send** m **from** p **to** q }

Asynchronous reception

event e **is** { **receive** m **from** p **to** env }
event e **is** { **receive** m **from** p **to** q }

Synchronous communication

event e **is** { **sync** m **from** p **to** q }

We add synchronous events to specify observers as matched events.

Only for observers

Property expression

We integrate property patterns description in the CDL language.
(currently under testing)

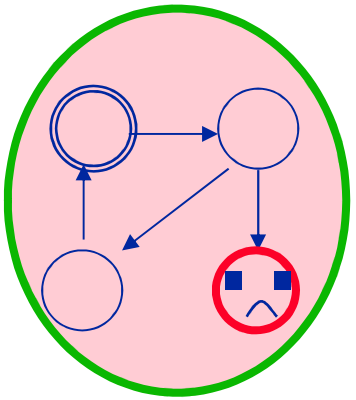
Different patterns : Response, Absence, Existence from [Dwyer], [Cheng]

The properties refer to detectable events :

- communication events
- predicate-based events

Currently, properties are formalized with automata *(timed or not)*

```
property p is
 {
      start    - - gard1 / pred1 / evt1 / update1 -> state1;
      …
      stateX  - - gardX / predX / evtX / updateX -> success;
      …
      stateY  - - gardY / predY / evtY / updateY -> reject
}
```



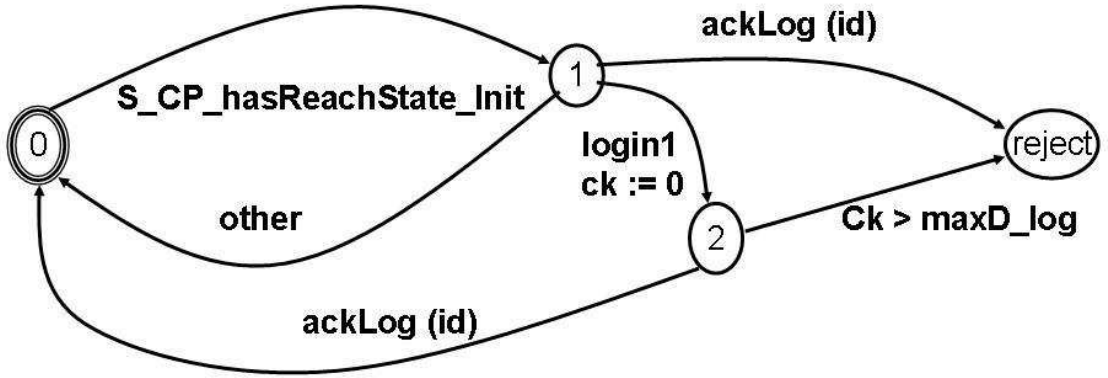
Observer

Pattern example (Response)

Property P;
ALL Ordered
exactly one **occurrence of** S_CP_hasReachState_Init
exactly one **occurrence of** login1
end
eventually **leads-to** [0..maxD_log]
AN
one or more **occurrence of** ackLog (id)
end
S_CP_hasReachState_Init may never **occurs**
login1 may never **occurs**
one of ackLog (id) cannot **occur before** login1
repeatability: true

We hare developed transformation algorithms to generate observer automata from pattern-based properties

An observer : entity or a program which observes the behavior of another entity or program.
Observers represent behavioral and timed properties to be validated on a model.



Property descriptions : Predicates

In the properties, we use predicates and events based on predicates

```
predicate   pred1  is          //  variable value
{
     {  MyProc  }1: myVar  = 1
}

predicate   pred2  is          // process state
{
     {  MyProc  }1@stateX
}

predicate   pred3  is          //  message number in a buffer
{
     {  MyComponent  }1 : myfifo.length = 2
}
```

A predicate is a boolean expression refering

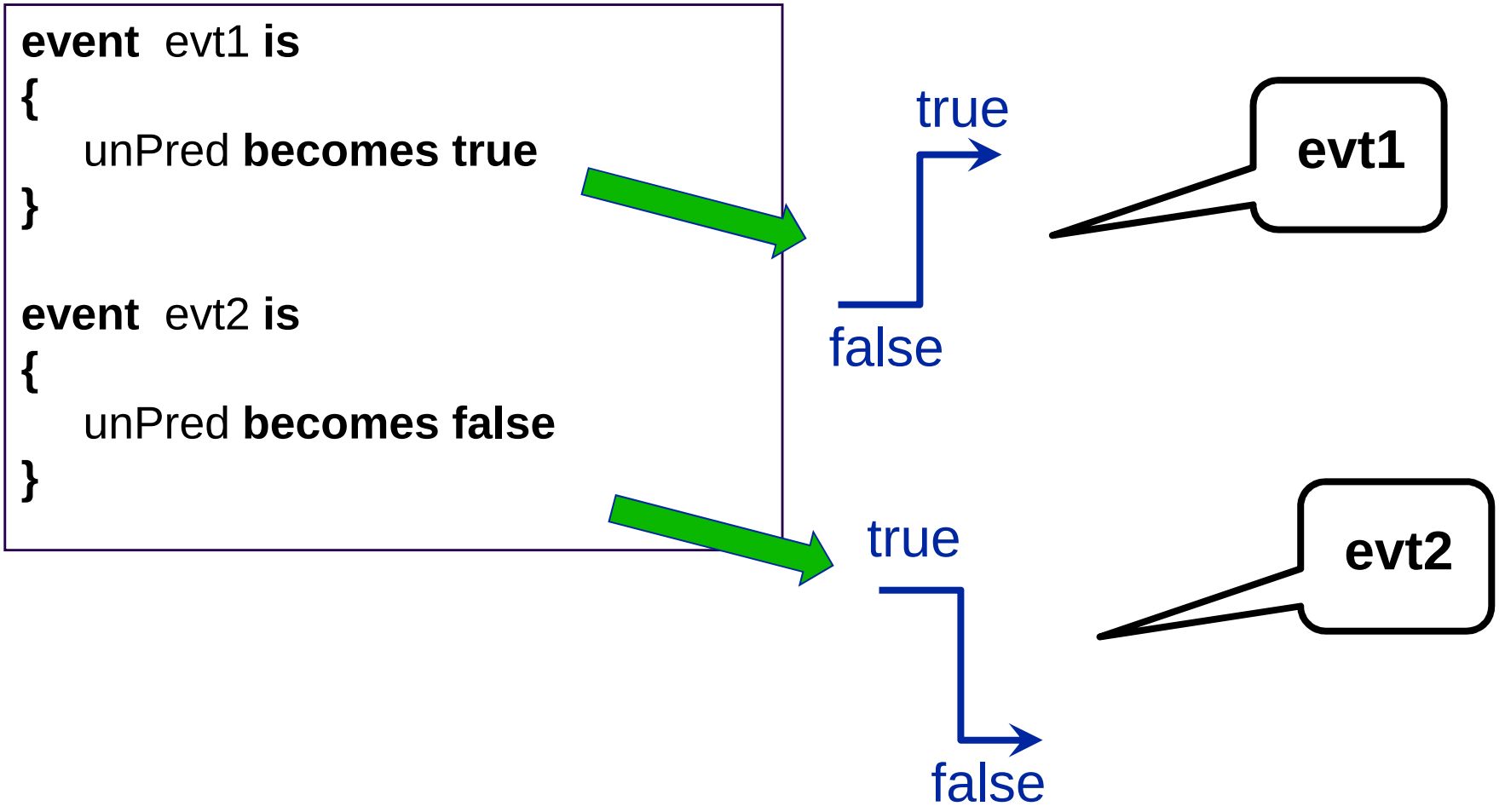
- a variable value,
- a process state
- a message number in a buffer.

Event : based on predicates

With predicates, we can specify events

```
event  evt1 is
{
    unPred becomes true
}

event  evt2 is
{
    unPred becomes false
}
```



Structure of CDL model

predicate aPredicat **is { … }**
…
event anEvent **is { … }**
…
property aPty **is { … }**
…
activity anAct **is { … }**
…
cdl context1 **is { … }**

cdl context2 **is { … }**

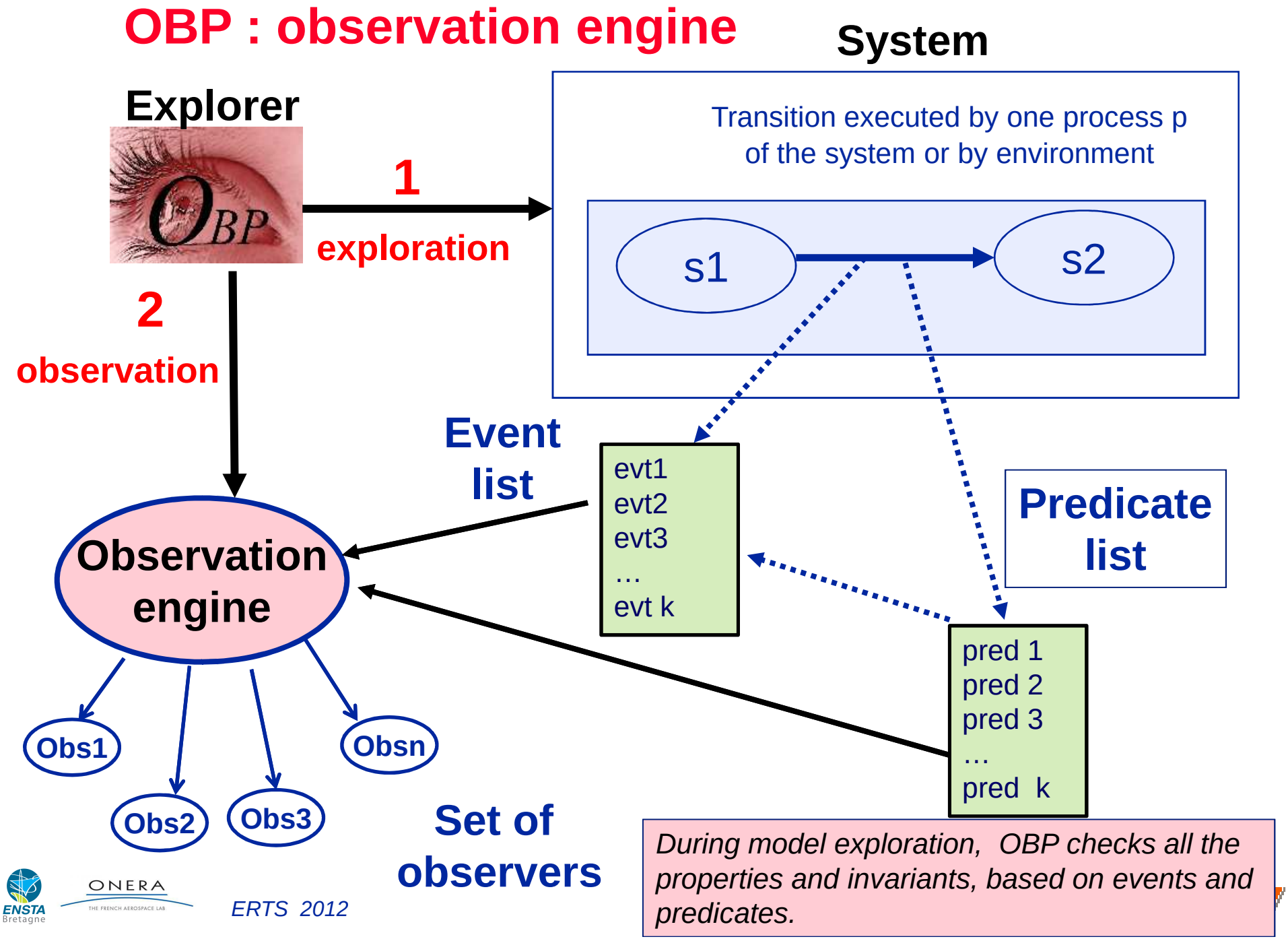
cdl context3 **is { … }**

cdl context4 **is { … }**
…

Set of properties.

A model CDL can be composed by several contexts.

OBP : observation engine



During model exploration, OBP checks all the properties and invariants, based on events and predicates.

Context Aware Model Exploration with OBP tool to Improve Model-Checking

Motivation for Context aware verification

Proposition : Toolset OBP and CDL

Context and requirement modeling (CDL)

Some results

Discussion and future work

Case study : some results (without CDL)

Some results of the exhaustive exploration for the case study **with OBP Explorer** and **without CDL** for different complexities**.**

N (Number of devices)	Exploration & analyze time (sec)	N.of LTS configurations	N.of LTS transitions
1	1	43 828	321 002
2	4	350 256	2 475 392
3	19	1 466 934	6 430 265
4	Explosion	–	–

Results obtained on 3 G. bytes memory computer.
When reaching 4 devices, the exploration fails.

Case study : some results (with CDL)

Some results of the exhaustive exploration for the case study **with OBP Explorer** and **with CDL**.

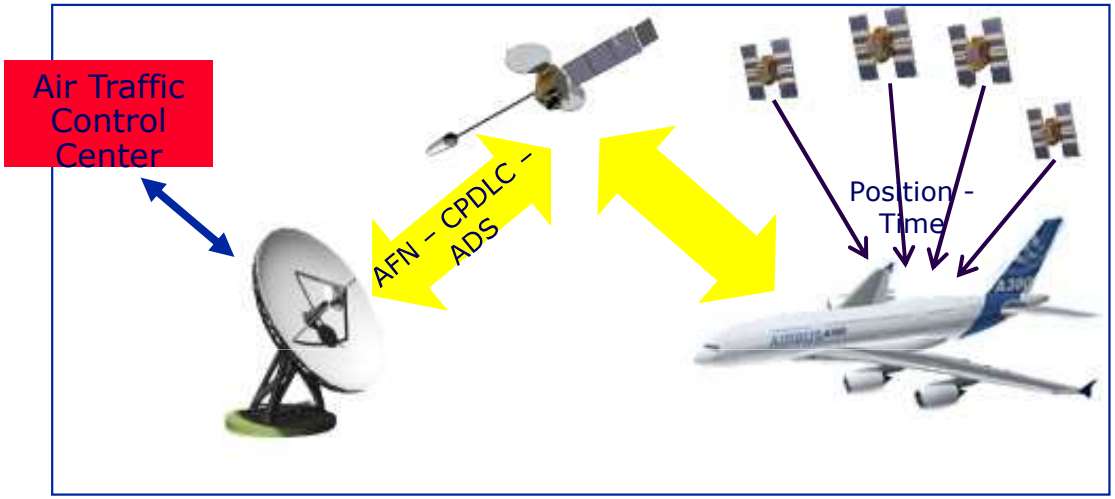
N. of devices	Exploration time (sec)	N. of sub-contexts	N. of LTS config.	N. of LTS trans.
4	954	22	16 450 288	75 362 832
5	1 256	28	33 568 422	156 743 290
6	3 442	242	68 880 326	368 452 864
7	6 480	344	126 450 324	634 382 590

Results obtained on 3 G. bytes memory computer.
These results show an exploration for 7 devices.

Some experiments

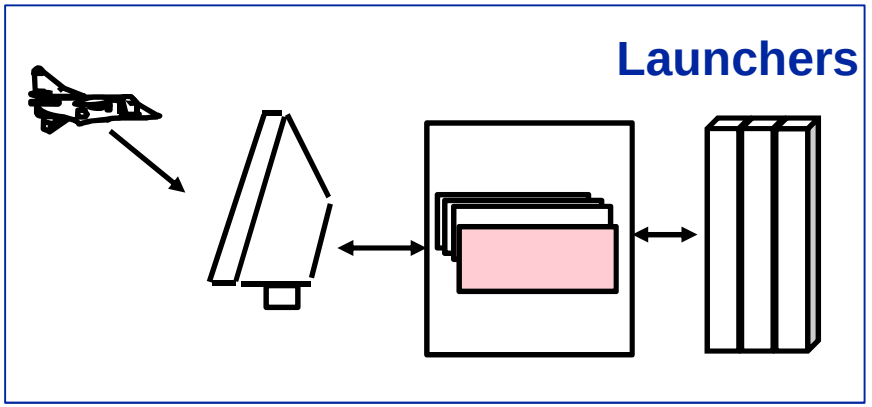
Currently, this approach is experimented with our industrial partners.

We are waiting for feedbacks from our partners about the experiments.



AIRBUS

CS COMMUNICATION & SYSTEMES



Context Aware Model Exploration with OBP tool to Improve Model-Checking

Motivation for Context aware verification

Proposition : Toolset OBP and CDL

Context and requirement modeling (CDL)

Some results

Discussion and future work

Conclusion

Work still in progress.

CDL allows to circumvent the state explosion
It can leverage different model-checkers

CDL contains an observation language:

- **fine grain observations**
- **property scoping**
- **property patterns to formalize properties**

OBP is available at www.obpcdl.org

Conclusion

Academic aspect :

- **We are evaluating CDL on industrial case studies**
- **Formal proofs are executed on software components by the partners themselves.**
- **With CDL : we can study concepts and a methodology for the formal validation in industrial context.**
- **We considerer CDL as a prototype language.**
- **CDL concepts could be implemented in another language or generated.**

Conclusion

Industrial aspect :

- **These experiments are rich in teaching for our partners and us**
- **They allows us to design a methodology, adapted to industrial processes**
- **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.**

Future work

An important issue is understanding feedbacks for diagnostics

We want to better understand the data returned from model-checker.

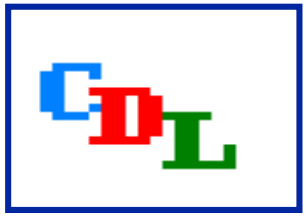
Relations between scenarios and properties can be exploited for understanding feedbacks for diagnostics

Another issue is the generation of CDL models

Parallel exploration on a computer network

Exploitation of CDL models to generate test sequences.

Thank you for your attention and your questions ?



www.obpcdl.org