



# Observing component behaviors

## Motivation

Temporal logic

Observing behaviors

Conclusion



# Motivation

## Component model

Structural consistency

Behavioral consistency ?

## Temporal logic

Specifying correct behaviors

(software) Implementation ?

## Goal

**Conception** : specifying component behaviors (static checking)

**Debug/supervision** : Observing component behaviors (dynamic checking)



# Temporal Logic

## Model

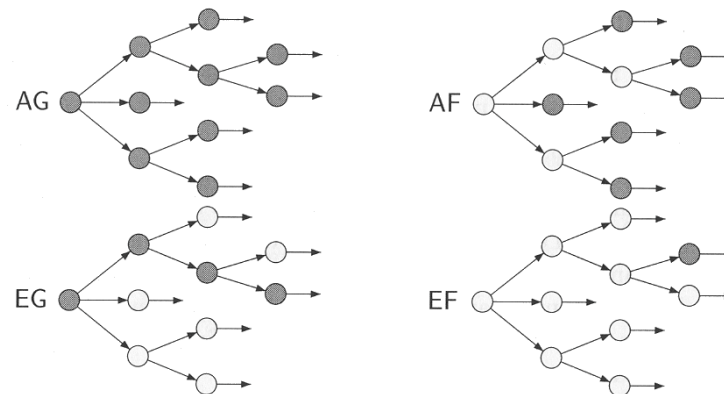
State machine : behaviors

## Temporal properties

Formula (TLA, CTL, Unity...) : specifies corrects behaviors

Safety (liveness) : violation can (cannot) be detected on finite behaviors

Composition, implementation... : conjunction, implication...



## Model-Checking :

Checks if a (finite) model satisfies the specification



# Observing behaviors

## Participants

**Specification / target** : correct behaviors (model + properties) / software implementation

**Observer** : on-line behaviors

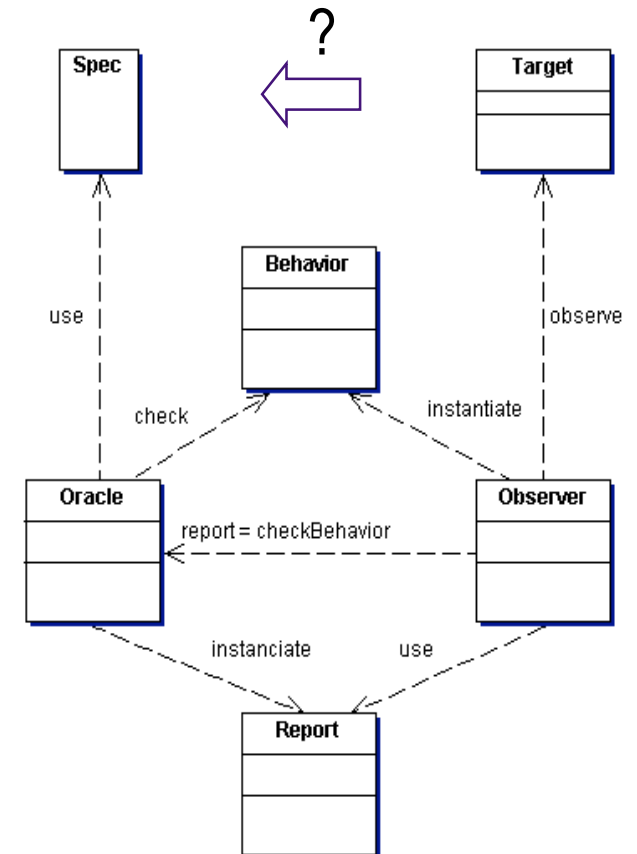
**Oracle** : model / run-time checker

## API

**neutral** (formalism/tool) & **minimal**

Observation : **pull** / **push**

Impl. : ex. TLA-TLC (code / specif.)





# Temporal Logic : ex. TLA

Temporal Logic of Action (Lamport : <http://lamport.org/>)

TLA : first order predicate logic & set theory

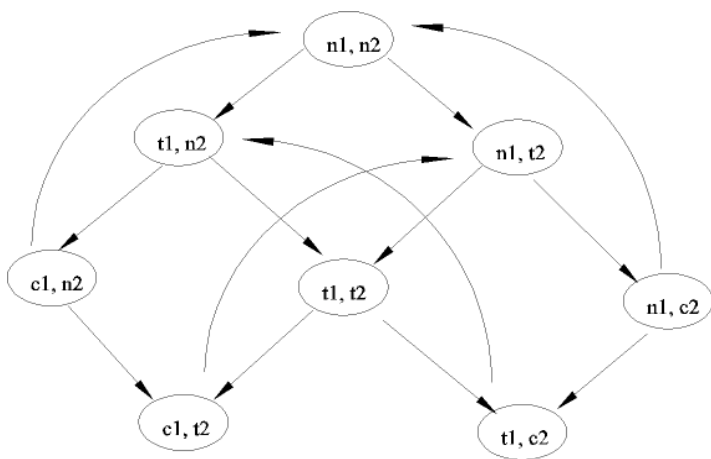
TLA : Temporal operators ( $\square$ ,  $\lozenge$ , ...)

TLA : Action operators (state / successor state)

TLC : Model-checking tool for TLA

Ex. simple « Mutual Exclusion » : 1 process :  $n \rightarrow t \rightarrow c \rightarrow n \rightarrow t \rightarrow c \rightarrow \dots$

Safety : one process (max) in critical section



```

MODULE MutExclu
EXTENDS Naturals, TLC
CONSTANT N
VARIABLE prc

Proc ≙ 1..N
Error: Invariant Check is violated.
The behavior up to this point is:
STATE 1: prc = (1 :> "n" @@ 2 :> "n")
STATE 2: prc = (1 :> "t" @@ 2 :> "n") . distinct state generated.
STATE 3: prc = (1 :> "c" @@ 2 :> "n") has been found.
STATE 4: prc = (1 :> "c" @@ 2 :> "t") es found, 0 states left on queue.
STATE 5: prc = (1 :> "c" @@ 2 :> "c")

Critic(i) ≙ prc[i] = "c"
Free(i) ≙ prc[i] ≠ "c"

Next ≙ ∃ i ∈ Proc : Try(i) ∨ Critic(i) ∨ Free(i)

Spec ≙ Init ∧ □[Next]p

THEOREM Spec ⇒ □TypeInvariant
  
```



# Observing component behaviors

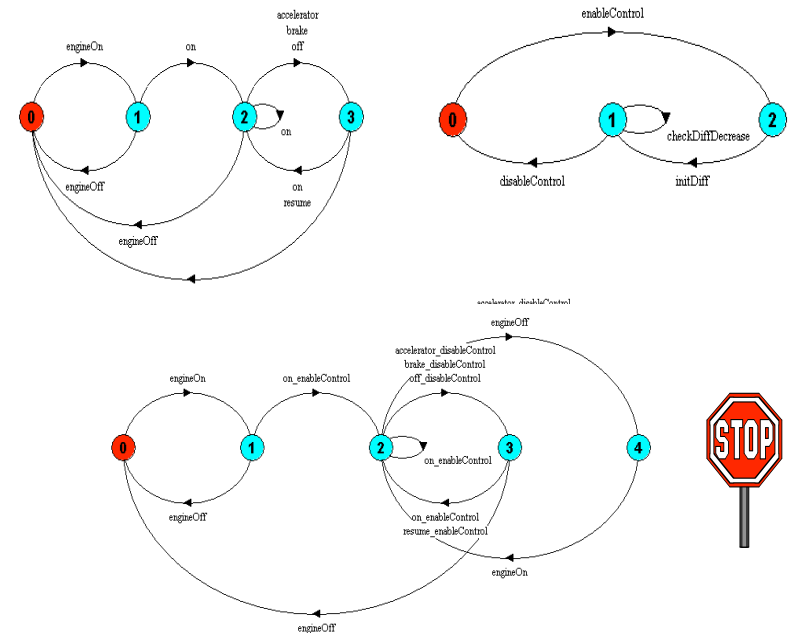
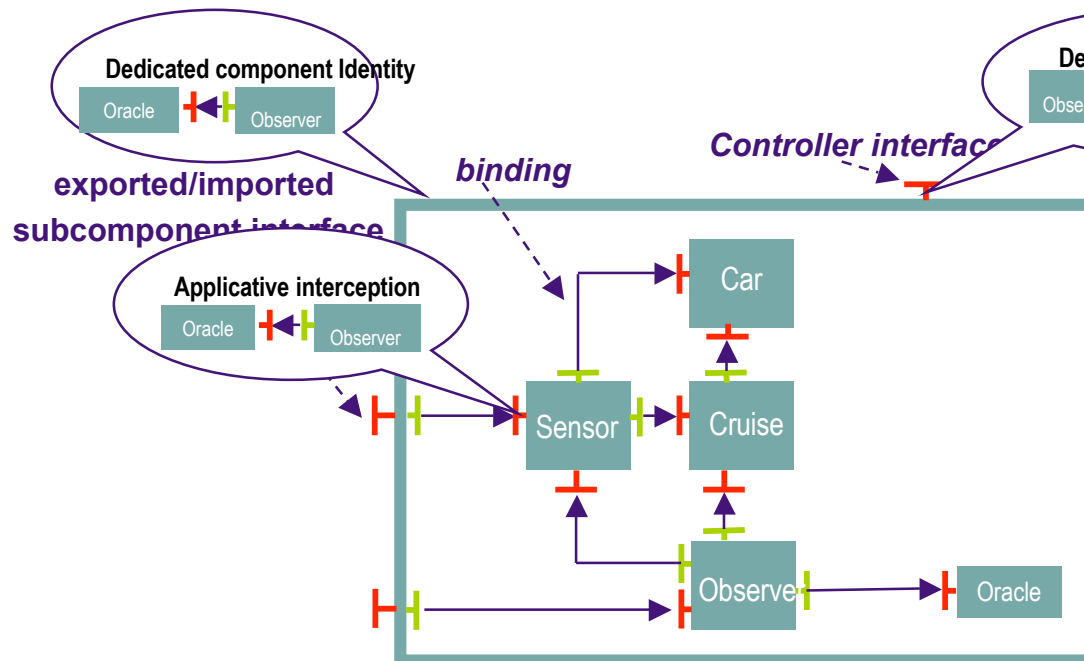
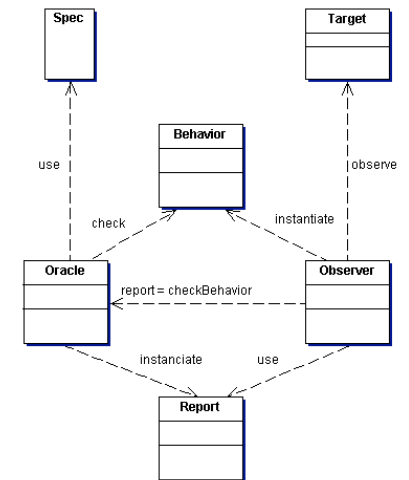
Fractal

**Plasm** : Oracle as a Fractal component

**Membrane**

*Local* : e.g. **Interception (Pre/post)** ; dedicated **Controller (life-cycle...)**

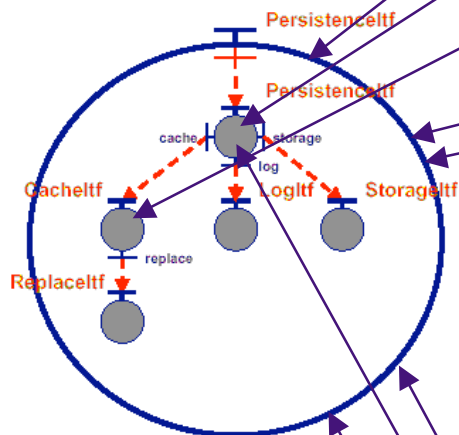
*Global* : **Component Identity**





# (Re)configuration

Ex. Persistence service



```

// 1
persistenceServiceLC.stopFc();
// 2
((BindingController)persistence
.getFcInterface(BindingController.BINDING_CONTROLLER))
.unbindFc("cache");
((BindingController)cache
.getFcInterface(BindingController.BINDING_CONTROLLER))
.unbindFc("replace");
// 3
persistenceServiceCC.removeFcSubComponent(cache);
persistenceServiceCC.removeFcSubComponent(replace);
// 4
// a
ComponentType newCacheType = typeFactory.createFcType(
new InterfaceType[] {
typeFactory.createFcItfType(
"server",
"Cache",
TypeFactory.SERVER,
TypeFactory.MANDATORY,
TypeFactory.SINGLE),
});
// b
ComponentIdentity newCacheTemplate =
templateFactory.createFcTemplate(
newCacheType, "PrimitiveTemplate", "Primitive", "NewCacheImpl");
// c
ComponentIdentity newCache =
((Template)newCacheTemplate.getFcInterface(Template.TEMPLATE))
.instantiateFc();
// 5
persistenceServiceCC.addFcSubComponent(newCache);
((BindingController)persistence
.getFcInterface(BindingController.BINDING_CONTROLLER))
.bindFc("cache", newCache.getFcInterface("server"));
// 6
persistenceServiceLC.startFc();

```

Temporal specification ?  
Composite policies



# Conclusion

## Pros

Continuum (conception/debug/supervision of behaviors)

*model / run-time checking*

Diagnostics on behaviors

Externalizes verifications (abstraction)

## Cons

Formalism !!!

*Temporal logic, process algebra, synchronous ; assertions...*

Safety only ; Target => specification ?

Observations (intrusive, global state...)

## Component model

Specification / contract ?

Software / formal composition ?

