Toward Validated Composition in Component-based Adaptive Middleware

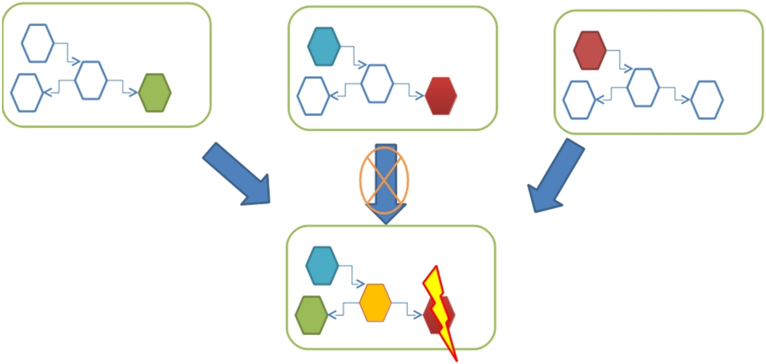
Annie Ressouche[1] and Jean-Yves Tigli[2] and Oscar Carrillo[1]

[1]Inria Sophia-Antipolis Méditerranée (Pulsar team)
[2]Nice Sophia Antipolis University and CNRS (Rainbow team)

SC 2011

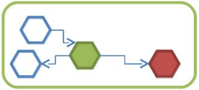
Motivation

- Challenge in adaptive middleware : How to manage interaction and sometimes conflicts between multiple ambient applications ?



Our Approach

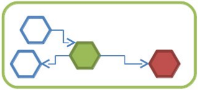
1. Need for validation on the critical component
   - Introduction of a synchronous monitor to manage such a component



2. Need for formal and sound composition operation

Our Approach

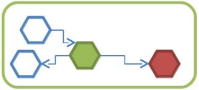
1. Need for validation on the critical component
   - Introduction of a synchronous monitor to manage such a component



2. Need for formal and sound composition operation

Our Approach

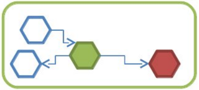
1. Need for validation on the critical component 
    - Introduction of a synchronous monitor to manage such a component



2. Need for formal and sound composition operation

Our Approach

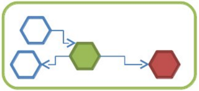
1. Need for validation on the critical component 
    - Introduction of a synchronous monitor to manage such a component



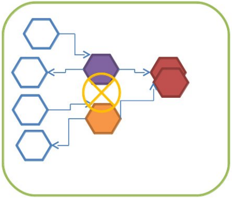
2. Need for formal and sound composition operation

Our Approach

1. Need for validation on the critical component 
   - Introduction of a synchronous monitor to manage such a component

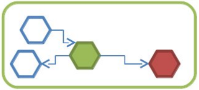


2. Need for formal and sound composition operation
   - Synchronous composition of monitors

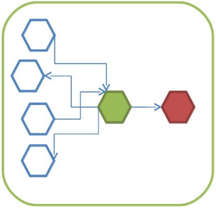


Our Approach

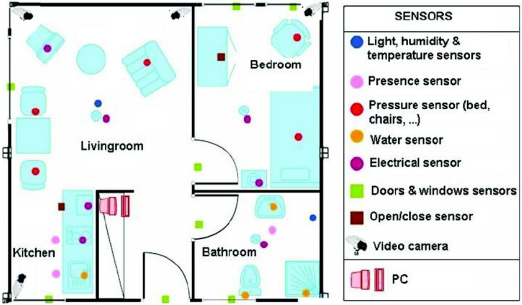
1. Need for validation on the critical component
   - Introduction of a synchronous monitor to manage such a component



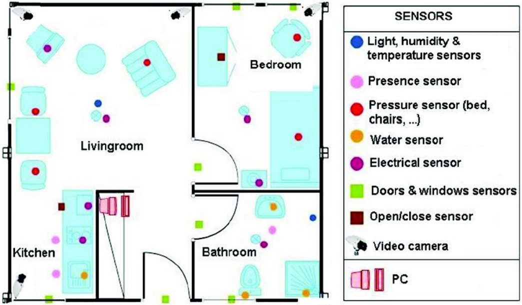
2. Need for formal and sound composition operation
   - Synchronous composition of monitors



Outline



- Monitor old adults in an instrumented home
- Use case : observe kitchen usage with :
  1. a camera sensor (to locate the person)
  2. a fridge sensor (contact sensor on the door)
  3. a timer sensor
  4. a posture sensor (accelerometers)
- Goal : send the appropriate alarm (warning, weak_alarm, strong_alarm



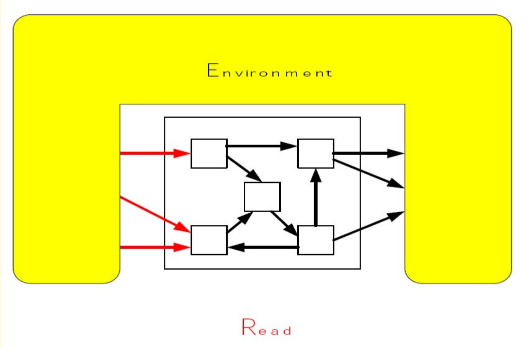
- Monitor old adults in an instrumented home
- Use case : observe kitchen usage with :
  1. a camera sensor (to locate the person)
  2. a fridge sensor (contact sensor on the door)
  3. a timer sensor
  4. a posture sensor (accelerometers)
- Goal : send the appropriate alarm (warning, weak_alarm, strong_alarm

Synchronous Modeling

- time model : monitors listen to events and provide output events in reaction They could be response time sensitive and should support formal validation($\Rightarrow$ determinism)
- component behavior models = synchronous models
- Synchronous models can be expressed as Mealy Machine

Synchronous Modeling

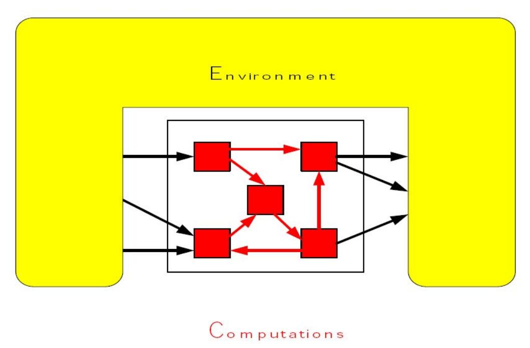
- time model : monitors listen to events and provide output events in reaction They could be response time sensitive and should support formal validation($\Rightarrow$ determinism)
- component behavior models = synchronous models



- Synchronous models can be expressed as Mealy Machine

Synchronous Modeling

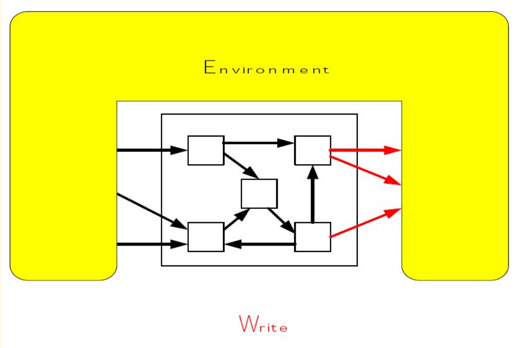
- time model : monitors listen to events and provide output events in reaction They could be response time sensitive and should support formal validation($\Rightarrow$ determinism)
- component behavior models = synchronous models



- Synchronous models can be expressed as Mealy Machine

Synchronous Modeling

- time model : monitors listen to events and provide output events in reaction They could be response time sensitive and should support formal validation($\Rightarrow$ determinism)
- component behavior models = synchronous models



- Synchronous models can be expressed as Mealy Machine

Synchronous Modeling

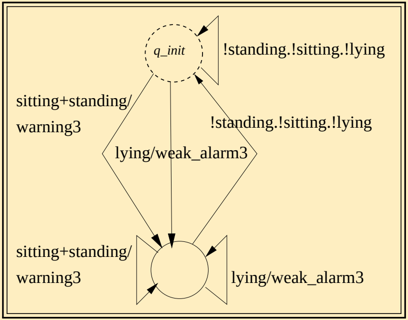
- time model : monitors listen to events and provide output events in reaction They could be response time sensitive and should support formal validation($\Rightarrow$ determinism)
- component behavior models = synchronous models
  Synchronous models respect the *synchronous hyothesis*
  - Succession of reactions $\Rightarrow$ logical time
  - Broadcasting of events (non blocking communication)
  - Reactions are **atomic** : input and resulting output events are simultaneous
  - Synchronous models are deterministic
- Synchronous models can be expressed as Mealy Machine

Synchronous Modeling

- time model : monitors listen to events and provide output events in reaction They could be response time sensitive and should support formal validation($\Rightarrow$ determinism)
- component behavior models = synchronous models
  Synchronous models respect the *synchronous hyothesis*
  - Succession of reactions $\Rightarrow$ logical time
  - Broadcasting of events (non blocking communication)
  - Reactions are **atomic** : input and resulting output events are simultaneous
  - Synchronous models are deterministic
- Synchronous models can be expressed as Mealy Machine

Mealy machines

- both finite automata and synchronous models
- model-checking techniques apply



$< Q, q^{init}, I, O, \mathcal{T}, \lambda >$:

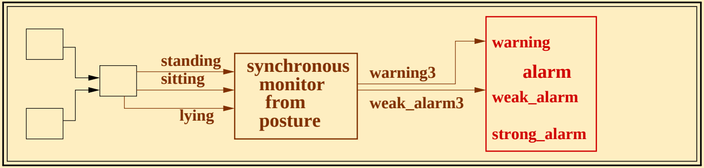
- Q : finite set of states
- $q^{init} \in Q$: initial state
- $\mathcal{T} \subseteq Q \times Q$: transition relation
- $\lambda : \mathcal{T} \times I^B \mapsto 2^{O_\epsilon}$: labeling function

Synchronous Monitors

- Critical components (C) will provide a synchronous model of their behaviors as a Mealy machine (M)
- If $M =< Q, q^{init}, I, O, \mathcal{T}, \lambda >$ and I_C is the input event set of C, there is an injective mapping : $in : O \mapsto I_C$

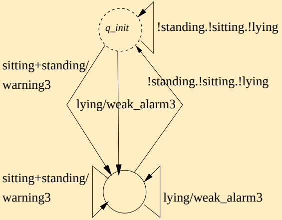
Synchronous Monitors

- Critical components (C) will provide a synchronous model of their behaviors as a Mealy machine (M)
- If $M =< Q, q^{init}, I, O, \mathcal{T}, \lambda >$ and I_C is the input event set of C, there is an injective mapping : $in : O \mapsto I_C$



Synchronous Monitors

- Critical components (C) will provide a synchronous model of their behaviors as a Mealy machine (M)
- If $M =< Q, q^{init}, I, O, \mathcal{T}, \lambda >$ and I_C is the input event set of C, there is an injective mapping : $in : O \mapsto I_C$



Component Behavior Validation

- model-checking techniques apply in our approach
- properties = $\forall CTL^*$ formulas
- formulas interpreted over Kripke structure
  detail
- $M \mapsto \mathcal{K}(M)$.

Definition

$M \models \psi$ iff $\mathcal{K}(M) \models \psi$ and iff each initial state of $\mathcal{K}(M)$ satifies ψ

Component Behavior Validation

- model-checking techniques apply in our approach
- properties = $\forall CTL^*$ formulas
- formulas interpreted over Kripke structure
  detail
- $M \mapsto \mathcal{K}(M)$.

Definition

$M \models \psi$ iff $\mathcal{K}(M) \models \psi$ and iff each initial state of $\mathcal{K}(M)$ satifies ψ

Component Behavior Validation

- model-checking techniques apply in our approach
- properties = $\forall CTL^*$ formulas
- formulas interpreted over Kripke structure

  detail

- $M \mapsto \mathcal{K}(M)$.

Definition

$M \models \psi$ iff $\mathcal{K}(M) \models \psi$ and iff each initial state of $\mathcal{K}(M)$ satifies ψ

Component Behavior Validation

- model-checking techniques apply in our approach
- properties = $\forall CTL^*$ formulas
- formulas interpreted over Kripke structure
  detail
- $M \mapsto \mathcal{K}(M)$.

Definition

$M \models \psi$ iff $\mathcal{K}(M) \models \psi$ and iff each initial state of $\mathcal{K}(M)$ satifies ψ

Component Behavior Validation

- model-checking techniques apply in our approach
- properties = $\forall CTL^*$ formulas
- formulas interpreted over Kripke structure
  detail
- $M \mapsto \mathcal{K}(M)$.

Definition

$M \models \psi$ iff $\mathcal{K}(M) \models \psi$ and iff each initial state of $\mathcal{K}(M)$ satifies ψ

Component Behavior Validation

- model-checking techniques apply in our approach
- properties = $\forall CTL^*$ formulas
- formulas interpreted over Kripke structure

  detail
- $M \mapsto \mathcal{K}(M)$.

Definition

$M \models \psi$ iff $\mathcal{K}(M) \models \psi$ and iff each initial state of $\mathcal{K}(M)$ satifies ψ

Component Behavior Validation

- model-checking techniques apply in our approach
- properties $= \forall CTL^*$ formulas
- formulas interpreted over Kripke structure

  detail
- $M \mapsto \mathcal{K}(M)$.

Definition

$M \models \psi$ iff $\mathcal{K}(M) \models \psi$ and iff each initial state of $\mathcal{K}(M)$ satifies ψ

Component Behavior Validation

- model-checking techniques apply in our approach
- properties $= \forall CTL^*$ formulas
- formulas interpreted over Kripke structure
  detail
- $M \mapsto \mathcal{K}(M)$.

Definition

$M \models \psi$ iff $\mathcal{K}(M) \models \psi$ and iff each initial state of $\mathcal{K}(M)$ satifies ψ

Component Behavior Validation

- model-checking techniques apply in our approach
- properties $= \forall CTL^*$ formulas
- formulas interpreted over Kripke structure
  detail
- $M \mapsto \mathcal{K}(M)$.

Definition

$M \models \psi$ iff $\mathcal{K}(M) \models \psi$ and iff each initial state of $\mathcal{K}(M)$ satifies ψ

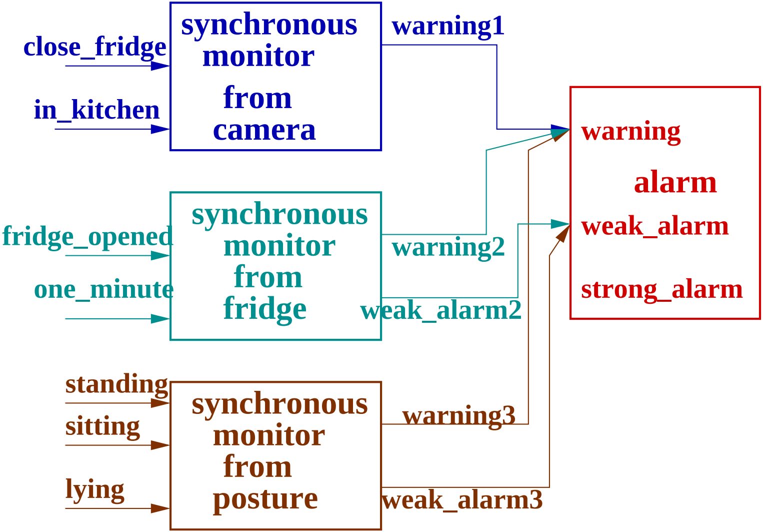
Component Behavior Validation

- model-checking techniques apply in our approach
- properties = $\forall CTL^*$ formulas
- formulas interpreted over Kripke structure
  detail
- $M \mapsto \mathcal{K}(M)$.

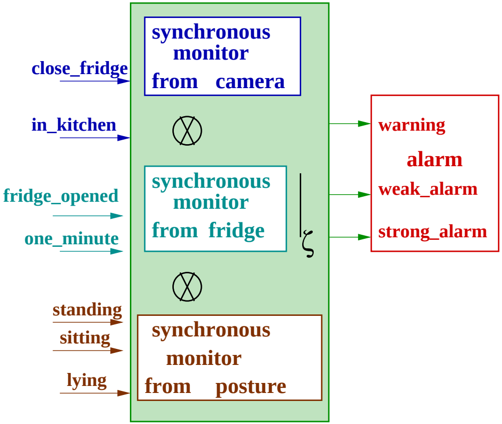
Definition

$M \models \psi$ iff $\mathcal{K}(M) \models \psi$ and iff each initial state of $\mathcal{K}(M)$ satifies ψ

A critical component may have multiple synchronous monitors :



Composition under constraints



Composition with constraints

- synchronous product ($\otimes$)
- constraint function (ζ)

Composition with constraints

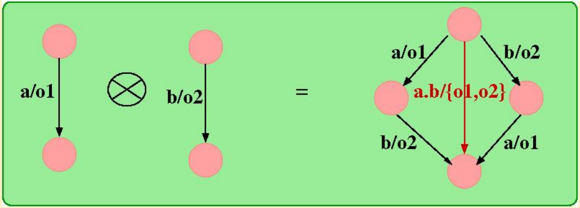
- synchronous product ($\otimes$)
- constraint function (ζ)

$M_1 =< Q_1, q_1^{init}, I_1, O_1, \mathcal{T}_1, \lambda_1 >$
$M_2 =< Q_2, q_2^{init}, I_2, O_2, \mathcal{T}_2, \lambda_2 >$
$M_1 \otimes M_2 = < Q_1 \times Q_2, (q_1^{init}, q_2^{init}), I_1 \cup I_2, O_1 \cup O_2, \mathcal{T}, \lambda > :$

- $\mathcal{T} = \{((q_1, q_2), (q_1', q_2')) \mid (q_1, q_1') \in \mathcal{T}_1, (q_2, q_2') \in \mathcal{T}_2\}$;
- $\lambda(((q_1, q_2), (q_1', q_2')), i_1 \cdot i_2) = o_1 \cup o_2)$ if there is $(q_1, q_1') \in \mathcal{T}_1 \mid \lambda_1((q_1, q_1'), i_1) = o_1)$ and $(q_2, q_2') \in \mathcal{T}_2 \mid \lambda_2((q_2, q_2'), i_2) = o_2)$

Composition with constraints

- synchronous product ($\otimes$)
- constraint function (ζ)



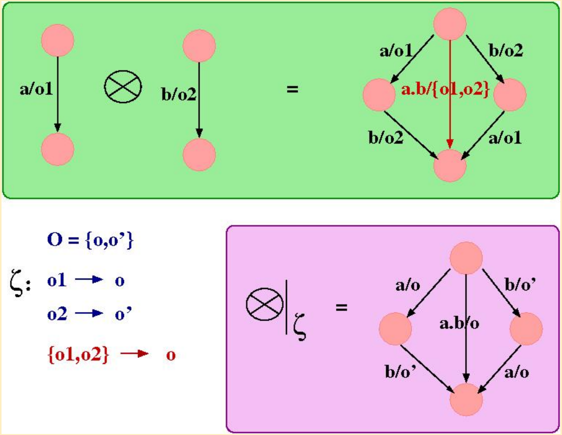
Composition with constraints

- synchronous product ($\otimes$)
- constraint function (ζ)

1. Define the output set O of the composition monitor such that there is an injection $in : O \mapsto I_C$
2. Define a surjective function $\gamma : O_1 \cup O_2 \cup O_1 \times O_2 \mapsto O_\epsilon$ according to the respective injection from monitor output events and I_C :
   - $\forall o_1 \in O_1,\ \gamma(o_1) = o$ and $in(o) = in_1(o_1)$
   - $\forall o_2 \in O_2,\ \gamma(o_2) = o$ and $in(o) = in_2(o_2)$
3. Deduce the constraint function $\zeta : 2^{O_1 \cup O_2} \mapsto 2^O$:
   $\forall o \in 2^{O_1 \cup O_2}$, if $\exists o_1, o_2 \in o$ such that $\gamma(o_1, o_2) \neq \epsilon$ then $\gamma(o_1, o_2) \in \zeta(o)$; else $\gamma(o_1) \in \zeta(o)$ and $\gamma(o_2) \in \zeta(o)$

Composition with constraints

- synchronous product ($\otimes$)
- constraint function (ζ)



$\forall CTL*$ formula preservation

- Goal : ensure that $\forall CTL*$ properties are preserved through composition under constraints ;
- Means :
  1. Show that $\mathcal{K}(M_1 \otimes |_\varsigma M_2)$ (K_ς) approximates $\mathcal{K}(M_1)$ (K_1) ;
  2. Define a translation τ_ς to map $\forall CTL*$ properties related to M_1 to properties related to $(M_1 \otimes |_\varsigma M_2)$;
  3. Prove that $K_1 \models \phi \Rightarrow K_\varsigma \models \tau_\varsigma(\phi)$;
  4. Deduce the result for M_1 and M_ς.

$\forall CTL*$ formula preservation

- Goal : ensure that $\forall CTL*$ properties are preserved through composition under constraints ;
- Means :
  1. Show that $\mathcal{K}(M_1 \otimes |_\zeta M_2)$ (K_ζ) approximates $\mathcal{K}(M_1)$ (K_1) ;
  2. Define a translation τ_ζ to map $\forall CTL*$ properties related to M_1 to properties related to $(M_1 \otimes |_\zeta M_2)$;
  3. Prove that $K_1 \models \phi \Rightarrow K_\zeta \models \tau_\zeta(\phi)$;
  4. Deduce the result for M_1 and M_ζ.

$\forall CTL*$ formula preservation

- Goal : ensure that $\forall CTL*$ properties are preserved through composition under constraints ;
- Means :
  1. Show that $\mathcal{K}(M_1 \otimes |_\zeta \ M_2)$ (K_ζ) approximates $\mathcal{K}(M_1)$ (K_1) ;
  2. Define a translation τ_ζ to map $\forall CTL*$ properties related to M_1 to properties related to $(M_1 \otimes |_\zeta \ M_2)$;
  3. Prove that $K_1 \models \phi \Rightarrow K_\zeta \models \tau_\zeta(\phi)$;
  4. Deduce the result for M_1 and M_ζ.

$\forall CTL*$ formula preservation

- Goal : ensure that $\forall CTL*$ properties are preserved through composition under constraints ;
- Means :
  1. Show that $\mathcal{K}(M_1 \otimes |_\zeta\ M_2)$ (K_ζ) approximates $\mathcal{K}(M_1)$ (K_1) ;
  2. Define a translation τ_ζ to map $\forall CTL*$ properties related to M_1 to properties related to $(M_1 \otimes |_\zeta\ M_2)$;
  3. Prove that $K_1 \models \phi \Rightarrow K_\zeta \models \tau_\zeta(\phi)$;
  4. Deduce the result for M_1 and M_ζ.

$\forall CTL*$ formula preservation

- Goal : ensure that $\forall CTL*$ properties are preserved through composition under constraints ;
- Means :
  1. Show that $\mathcal{K}(M_1 \otimes |_\zeta\ M_2)$ (K_ζ) approximates $\mathcal{K}(M_1)$ (K_1) ;
  2. Define a translation τ_ζ to map $\forall CTL*$ properties related to M_1 to properties related to $(M_1 \otimes |_\zeta\ M_2)$;
  3. Prove that $K_1 \models \phi \Rightarrow K_\zeta \models \tau_\zeta(\phi)$;
  4. Deduce the result for M_1 and M_ζ.

$\forall CTL*$ formula preservation

- Goal : ensure that $\forall CTL*$ properties are preserved through composition under constraints ;
- Means :
  1. Show that $\mathcal{K}(M_1 \otimes |_\zeta\ M_2)$ (K_ζ) approximates $\mathcal{K}(M_1)$ (K_1) ;
  2. Define a translation τ_ζ to map $\forall CTL*$ properties related to M_1 to properties related to $(M_1 \otimes |_\zeta\ M_2)$;
  3. Prove that $K_1 \models \phi \Rightarrow K_\zeta \models \tau_\zeta(\phi)$;
  4. Deduce the result for M_1 and M_ζ.

Lemma

K_ζ approximates K_1

Approximation

1. there is a surjective mapping $h_a : A_\zeta \mapsto A_1$
2. there is a surjective mapping $h : \mathcal{K}Q_\zeta \mapsto \mathcal{K}Q_1$ such that $h(q_\zeta) = q_1 \Rightarrow \forall a_1 \in L_1(q_1), \exists a_\zeta \in L_\zeta(q_\zeta)$ and $h_a(a_\zeta) = a_1$.
3. $q_\zeta \longrightarrow q'_\zeta$ is a transition of K_ζ then $h(q_\zeta) \longrightarrow h(q'_\zeta)$ is a transition in K_1

Definition

τ_ζ :

- $\tau_\zeta(true) = true\,;\ \tau_\zeta(false) = false$
- $\forall a_1 \in A_1, \tau_\zeta(a_1) = \bigvee_{a_\zeta \in A_\zeta} a_\zeta \mid h_a(a_1) = a_\zeta$
- extended to formulas according to logic syntax

Theorem

Let M_1 and M_2 be two Mealy machines and ϕ a $\forall CTL*$ formula related to M_1, then $M_1 \models \phi \Rightarrow M_1 \otimes |_\zeta M_2 \models \tau_\zeta(\phi)$

Definition

τ_ζ :

- $\tau_\zeta(true) = true$; $\tau_\zeta(false) = false$
- $\forall a_1 \in A_1, \tau_\zeta(a_1) = \bigvee_{a_\zeta \in A_\zeta} a_\zeta \mid h_a(a_1) = a_\zeta$
- extended to formulas according to logic syntax

Theorem

Let M_1 and M_2 be two Mealy machines and ϕ a $\forall CTL*$ formula related to M_1, then $M_1 \models \phi \Rightarrow M_1 \otimes \mid_\zeta M_2 \models \tau_\zeta(\phi)$

WComp : our experimental middleware

- WComp, middleware for ubiquitous and ambient computing
- Based on services for devices software infrastructure
- Manage interactions between devices at runtime using a component-based architecture and event flows

WComp Synchronous Monitor Specification

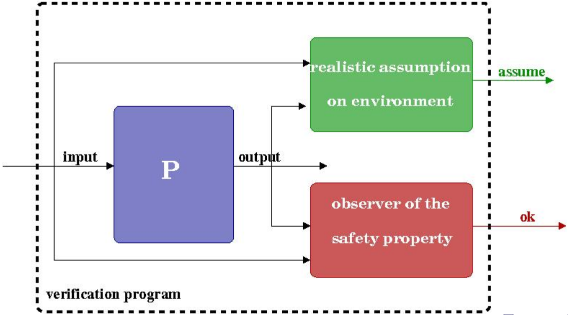
1. Lustre *synchronous language* to specify mealy machines :
    - respect of synchrony hypothesis
    - compilation generates mealy machines
    - synchronous product natural
    - constraint functions expressed as equations
    - well adapted to formal verification
2. Lesar model-checker to verify properties :
    - Bdd based model-checker
    - observers to express properties (in Lustre)

WComp Synchronous Monitor Specification

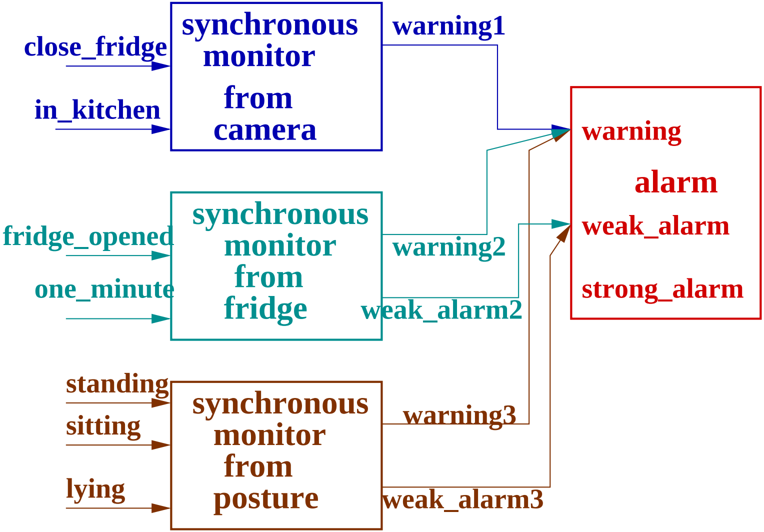
1. Lustre *synchronous language* to specify mealy machines :
   - respect of synchrony hypothesis
   - compilation generates mealy machines
   - synchronous product natural
   - constraint functions expressed as equations
   - well adapted to formal verification
2. Lesar model-checker to verify properties :
   - Bdd based model-checker
   - observers to express properties (in Lustre)

WComp Synchronous Monitor Specification

1. Lustre *synchronous language* to specify mealy machines :
   - respect of synchrony hypothesis
   - compilation generates mealy machines
   - synchronous product natural
   - constraint functions expressed as equations
   - well adapted to formal verification
2. Lesar model-checker to verify properties :
   - Bdd based model-checker
   - observers to express properties (in Lustre)

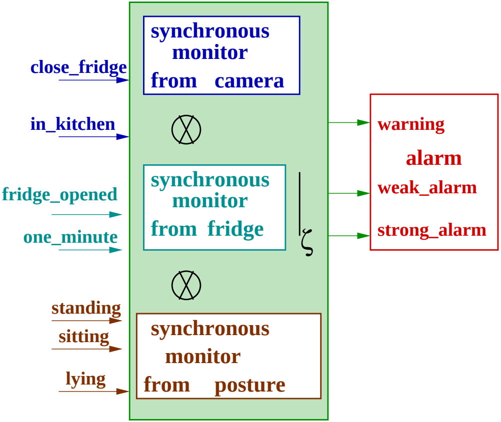


Use Case Implementation



```
node camera(in_kitchen,close_fridge:bool) returns(warning1:bool)
let warning1 = in_kitchen and close_fridge;
tel
node fridge(fridge_opened, one_minute: bool)
     returns (warning2, weak_alarm2: bool);
let warning2= fridge_opened and not one_minute;
    weak_alarm2= fridge_opened and one_minute;
tel
node posture(sitting, standing,lying:bool)
     returns(warning3,weak_alarm3:bool)
let  warning3 = (standing or sitting) and not lying;
     weak_alarm3 = not standing and not sitting and lying;
tel
```

Use Case Implementation : monitor composition



```
node alarm_comp (close_fridge, fridge_opened, one_minute, standing,
                 sitting, lying, in_kitchen : bool)
     returns (warning, weak_alarm, strong_alarm : bool)

var warning1, warning2, warning3, weak_alarm2, weak_alarm3 : bool;
let  warning1 = camera(in_kitchen, close_fridge);
    (warning2, weak_alarm2) = fridge(fridge_opened, one_minute);
    (warning3, weak_alarm3) = posture(standing, sitting, lying);

    warning = warning1 or warning2 or warning3 and not weak_alarm2
              and not weak_alarm3;
    weak_alarm = weak_alarm2 xor weak_alarm3;
    strong_alarm = weak_alarm2 and weak_alarm3;
tel
```

```
node verif (close_fridge, fridge_opened, one_minute, standing,
            sitting, lying, in_kitchen : bool) returns (prop: bool)
var warning, weak_alarm, strong_alarm : bool;
let (warning, weak_alarm, strong_alarm) =
           alarm_comp(close_fridge, fridge_opened, one_minute,
                standing, sitting, lying, in_kitchen);
    assert (not ((standing and lying) or (standing and sitting) or
               (lying and sitting)));
   prop = if (fridge_opened and one_minute and lying)
                   then strong_alarm  else true;
tel
```

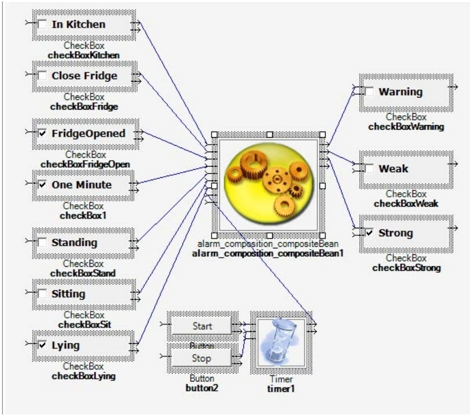
```
node verif (close_fridge, fridge_opened, one_minute, standing,
            sitting, lying, in_kitchen : bool) returns (prop: bool)
var warning, weak_alarm, strong_alarm : bool;
let (warning, weak_alarm, strong_alarm) =
           alarm_comp(close_fridge, fridge_opened, one_minute,
                standing, sitting, lying, in_kitchen);
    assert (not ((standing and lying) or (standing and sitting) or
                 (lying and sitting)));
   prop = if (fridge_opened and one_minute and lying)
                         then strong_alarm  else true;
tel
```

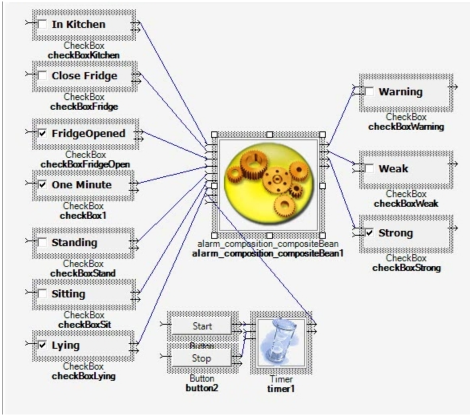
Property Preservation

In fridge synchronous monitor : $fridge\_opened \Rightarrow warning_2$

$\tau_\zeta(warning_2) = warning$

In alarm_comp monitor : $fridge\_opened \Rightarrow warning$


In Kitchen
CheckBox
checkBoxKitchen
Close Fridge
CheckBox
checkBoxFridge
FridgeOpened
CheckBox
checkBoxFridgeOpen
One Minute
CheckBox
checkBox1
Standing
CheckBox
checkBoxStand
Sitting
CheckBox
checkBoxSit
Lying
CheckBox
checkBoxLying
alarm_composition_compositeBean
alarm_composition_compositeBean1
Warning
CheckBox
checkBoxWarning
Weak
CheckBox
checkBoxWeak
Strong
CheckBox
checkBoxStrong
Start
Stop
Button
button2
Timer
timer1



SynComp tool offering :

- facilities to design synchronous monitor and observers
- automatic generation of WComp components for synchronous monitors

Future Work

1. Improve constraint function expression (default rules) to get efficient adaptation
2. A dedicated language versus Lustre
3. Apply Abstract Interpretation methodology
   - To perform validation on complex value events
   - To strengthen runtime composition
4. Study how global properties can be decomposed into local ones (assume-guarantee paradigm)

Future Work

1. Improve constraint function expression (default rules) to get efficient adaptation
2. A dedicated language versus Lustre
3. Apply Abstract Interpretation methodology
    - To perform validation on complex value events
    - To strengthen runtime composition
4. Study how global properties can be decomposed into local ones (assume-guarantee paradigm)

Future Work

1. Improve constraint function expression (default rules) to get efficient adaptation
2. A dedicated language versus Lustre
3. Apply Abstract Interpretation methodology
   - To perform validation on complex value events
   - To strengthen runtime composition
4. Study how global properties can be decomposed into local ones (assume-guarantee paradigm)

Future Work

1. Improve constraint function expression (default rules) to get efficient adaptation
2. A dedicated language versus Lustre
3. Apply Abstract Interpretation methodology
   - To perform validation on complex value events
   - To strengthen runtime composition
4. Study how global properties can be decomposed into local ones (assume-guarantee paradigm)

Kripke Structure

A *Kripke structure* K is a tuple : $K =< Q, Q_0, A, R, L >$ where :

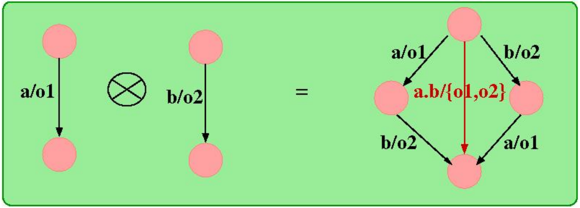
- Q is a finite set of states ;
- $Q_0 \subseteq Q$ is the set of initial states ;
- A is a finite set of atomic propositions ;
- $R \subseteq Q \times Q$ is a transition relation that must be total : for every state $q \in Q$, there is a state q' such that $R(q, q')$;
- $L : S \mapsto 2^A$ is a labeling function that labels each state by the set of atomic propositions true in that state.

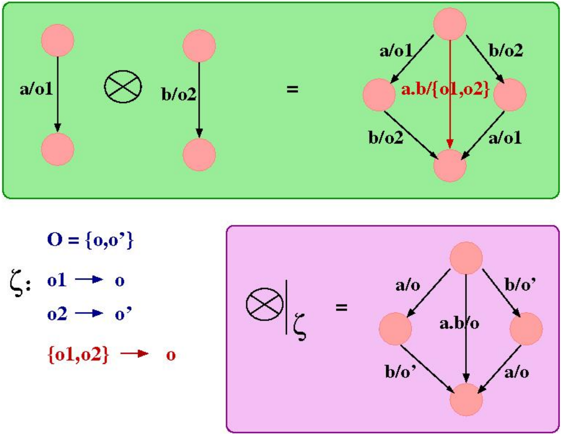
return

Definition

$M_1 =< Q_1, q_1^{init}, I_1, O_1, \mathcal{T}_1, \lambda_1 >$
$M_2 =< Q_2, q_2^{init}, I_2, O_2, \mathcal{T}_2, \lambda_2 >$
$M_1 \otimes M_2 = < Q_1 \times Q_2, (q_1^{init}, q_2^{init}), I_1 \cup I_2, O_1 \cup O_2, \mathcal{T}, \lambda > :$

- $\mathcal{T} = \{((q_1, q_2), (q_1', q_2')) \mid (q_1, q_1') \in \mathcal{T}_1, (q_2, q_2') \in \mathcal{T}_2\}$;
- $\lambda(((q_1, q_2), (q_1', q_2')), i_1 \cdot i_2) = o_1 \cup o_2)$ if there is $(q_1, q_1') \in \mathcal{T}_1 \mid \lambda_1((q_1, q_1'), i_1) = o_1)$ and $(q_2, q_2') \in \mathcal{T}_2 \mid \lambda_2((q_2, q_2'), i_2) = o_2)$




a/o1
b/o2
=
a/o1
b/o2
a.b/{o1,o2}
b/o2
a/o1
O = {o,o'}
ζ: o1 → o
o2 → o'
{o1,o2} → o
ζ
=
a/o
b/o'
a.b/o
b/o'
a/o

return