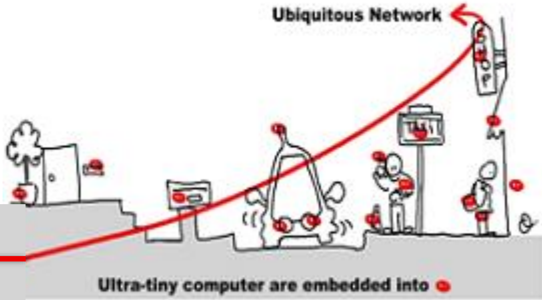
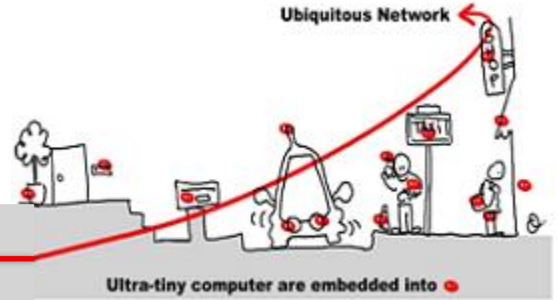


Lecture 7
Aspects of Assemblies

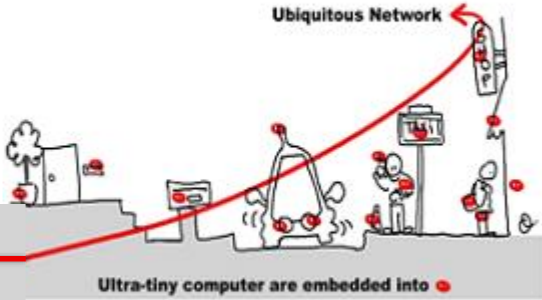
Aspects of Assemblies for structural self-adaptation

Aspect of Assembly Concept for self-adaptation

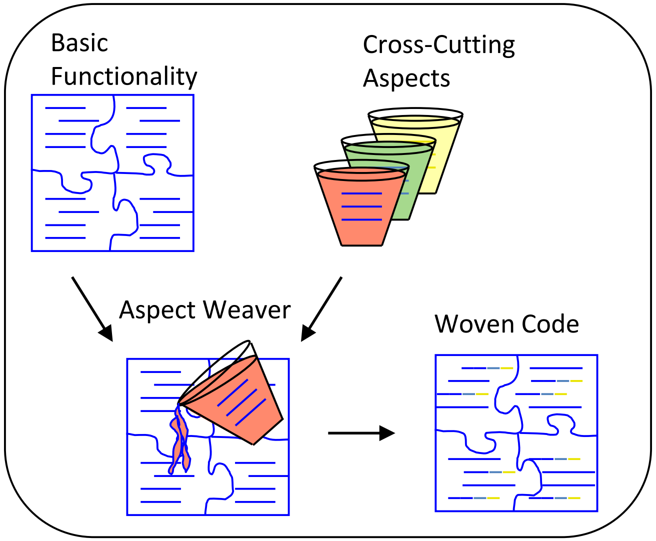


- From AOP Principles
- Aspect of Assembly Principles
- Complete AA Weaving Cycle
- Different kinds of conflict resolution

From Aspect-Oriented Programming principles

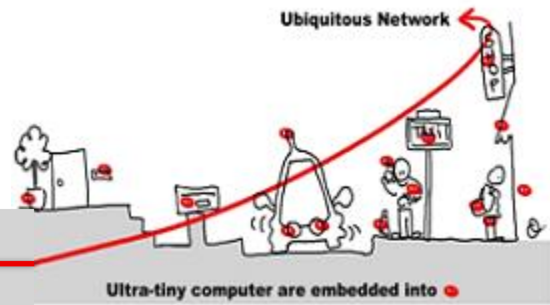


- Complex programs are composed of different intervened cross-cutting concerns.
- Cross-cutting concerns:
  - Properties or areas of interest such as QoS, energy consumption, fault tolerance, and security.
- Terminology
  - Aspect
  - Basic Functionality
  - Aspect Language
  - Aspect Weaver
    - Static
    - Dynamic
  - Woven Code



AOP in action.

Reminder : AOP Principles




```
public class HelloWorld {
        public static void main (String[ ] args)  {
                new HelloWorld().sayHello();
        }
        public void sayHello () {
                system.out.println("Hello World!");
        }
}
```

```
pointcut Two():
        execution(*HelloWorld.sayHello(..));
```

```
before():Two() {
        System.out.println( "Hello One ...");
    }
```

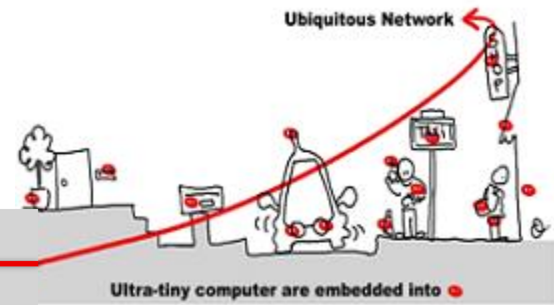
Weaver

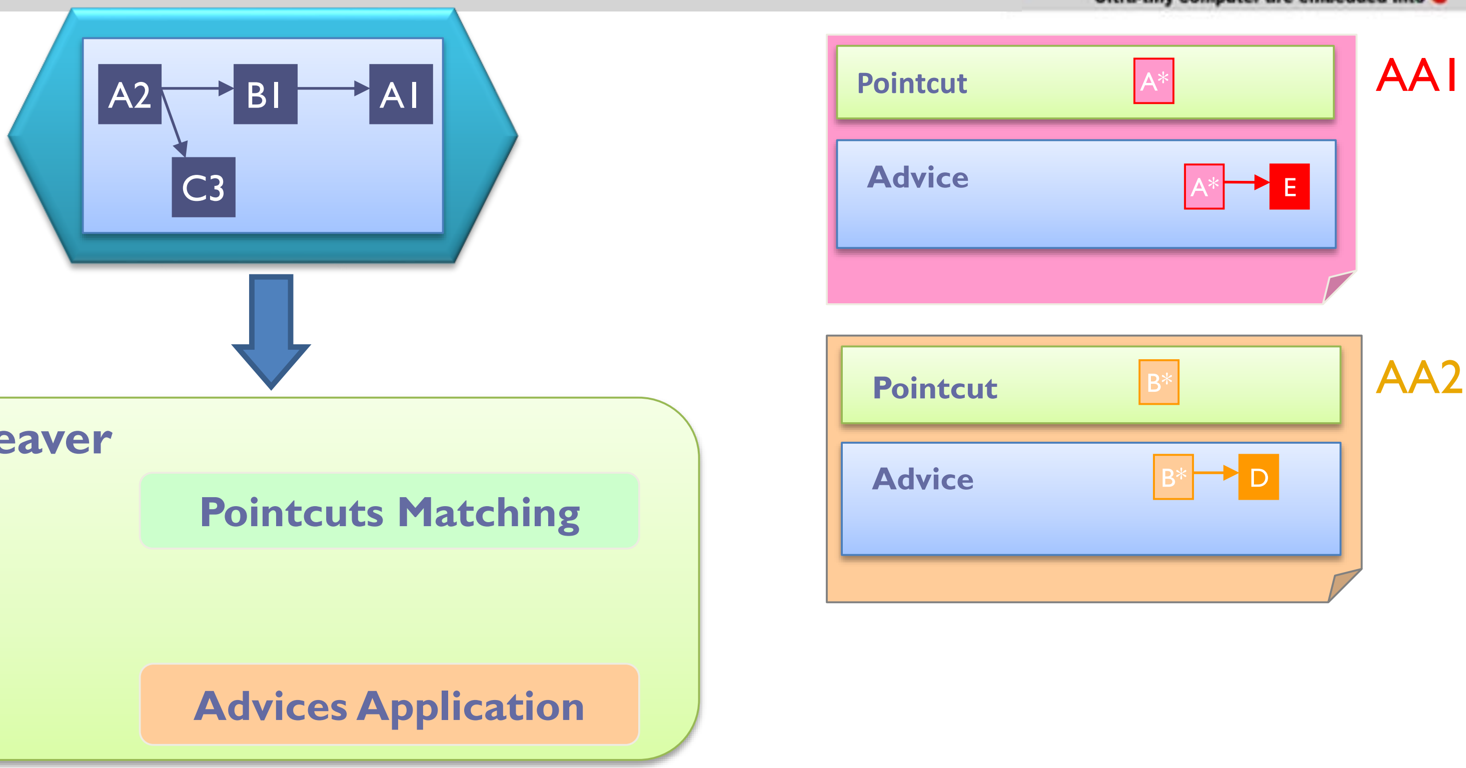
Pointcuts Matching 1

Advices Application 2

```
public class HelloWorld {
        public static void main (String[ ] args) {
                System.out.println("Hello One...");
                new HelloWorld().sayHello();
                System.out.println("Hello Two...");
        }
        public void sayHello () {
                system.out.println("Hello World!");
        }
}
```

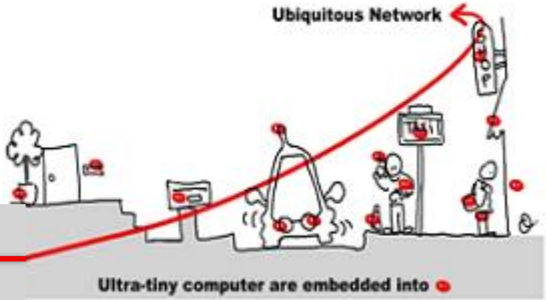
Aspect of Assembly Principles

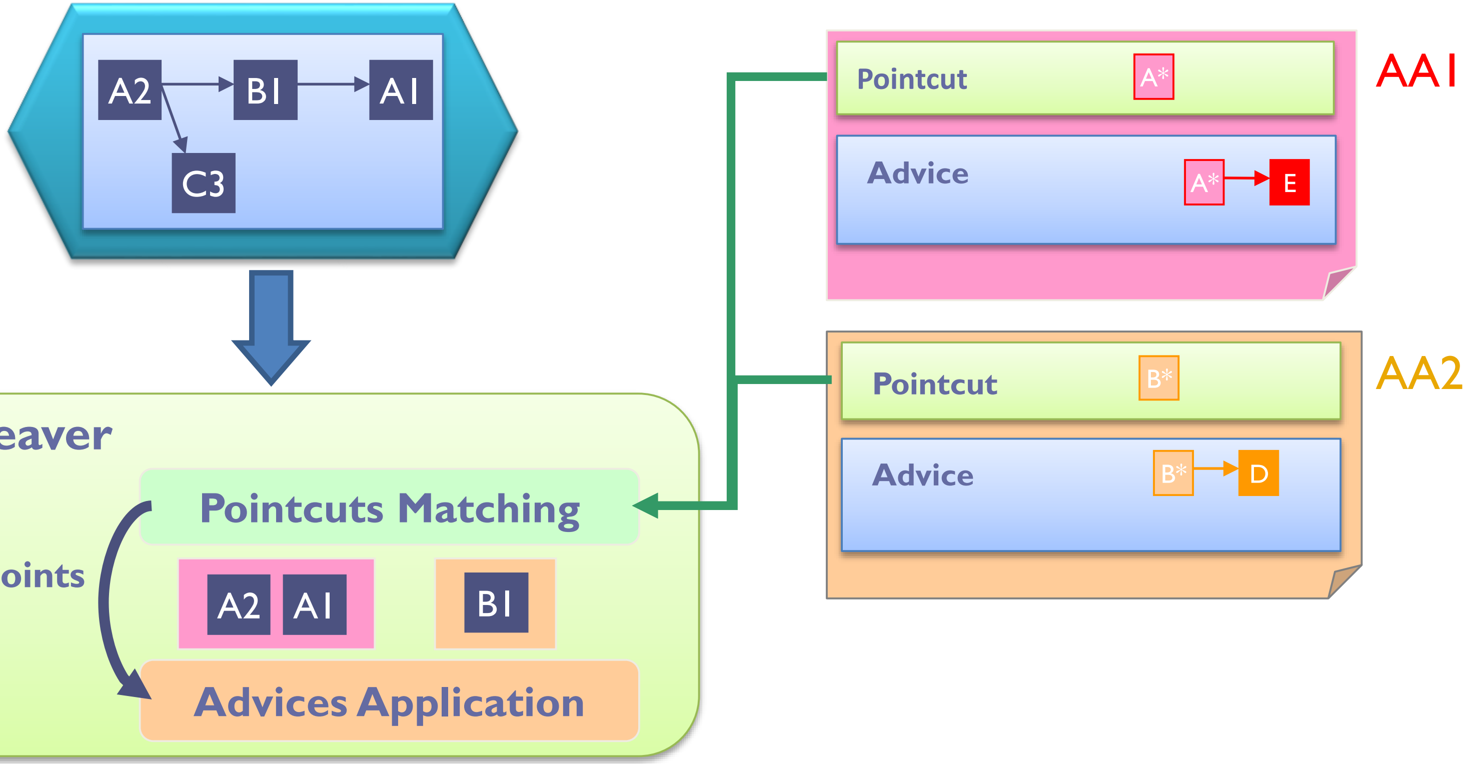




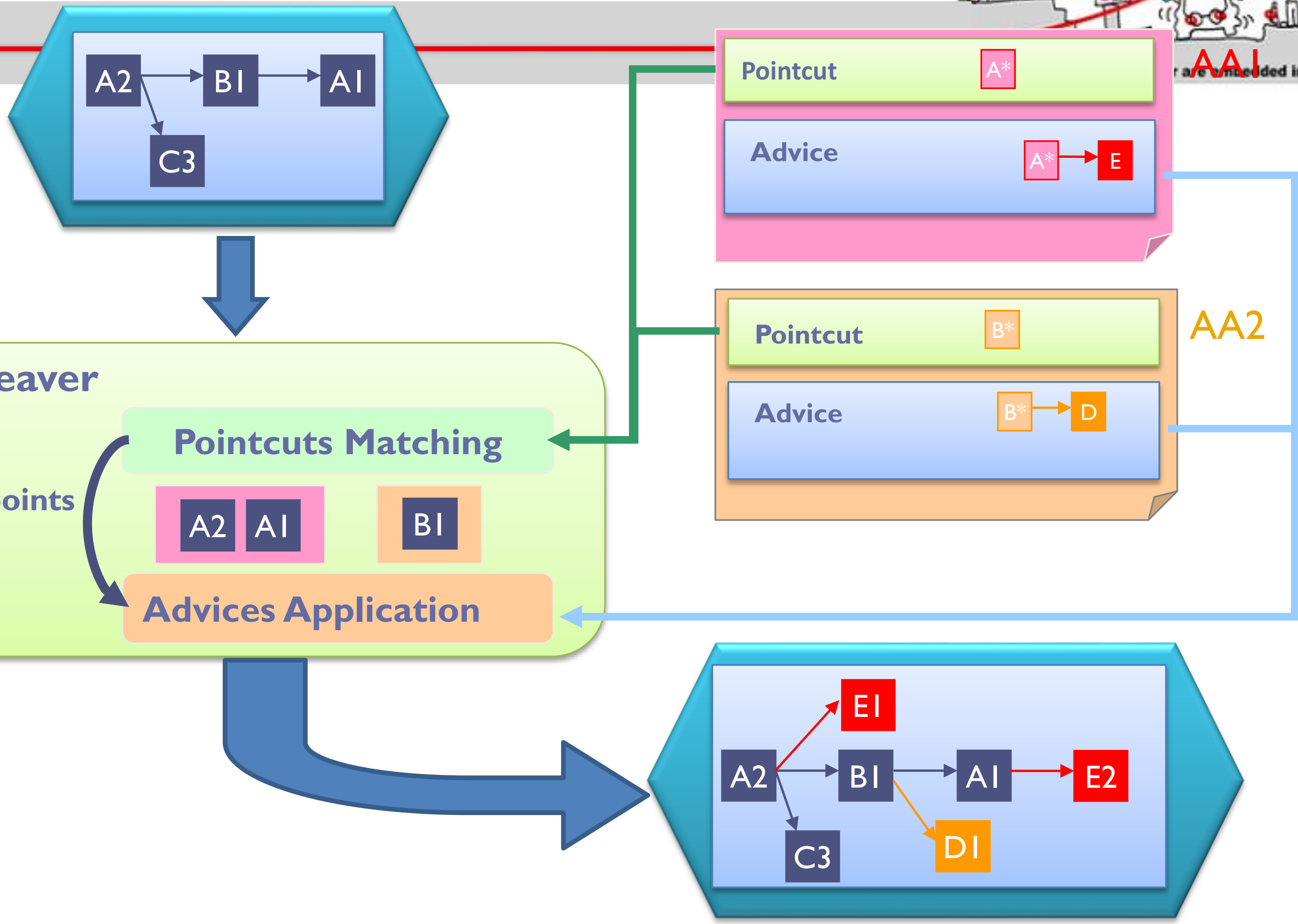
AOP inspired for Component based approach (like LCA)

Aspect of Assembly Principles

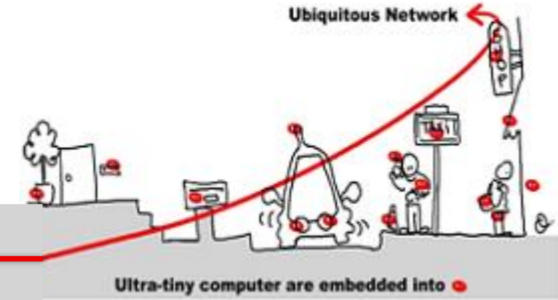


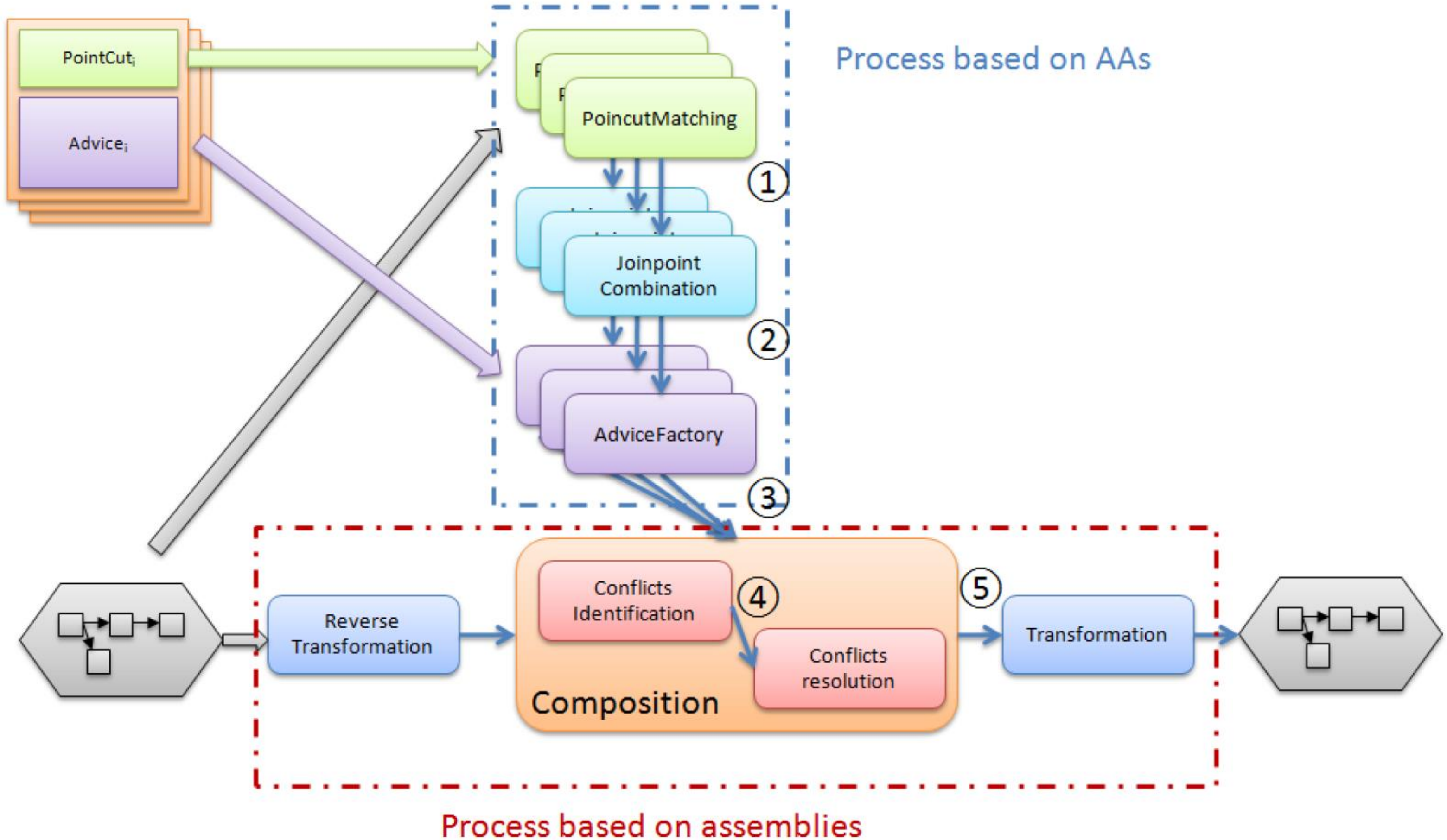


Aspect of Assembly Principles



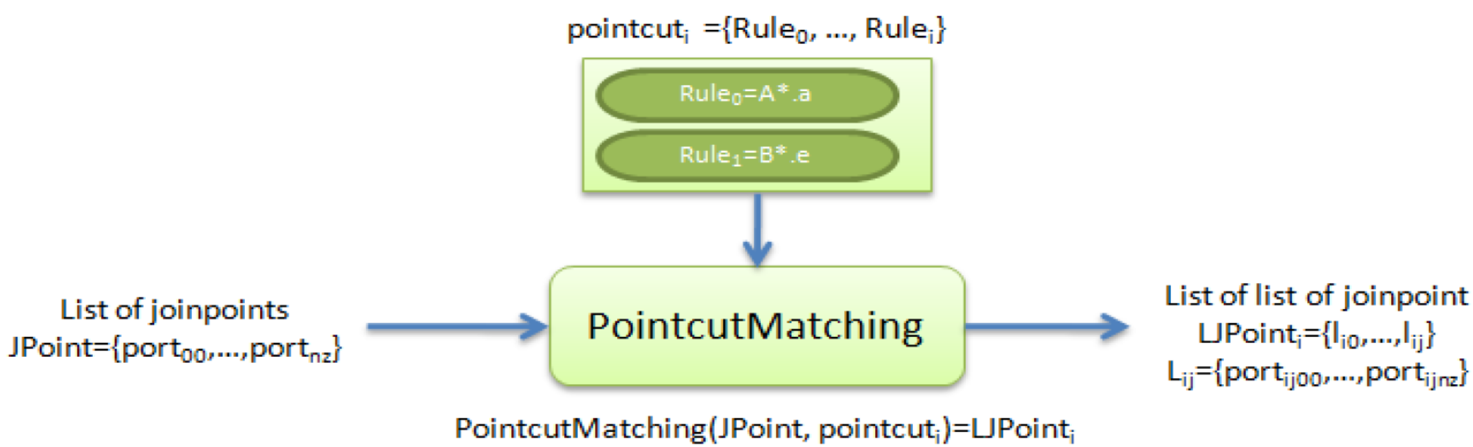
Complete AA Weaving Cycle



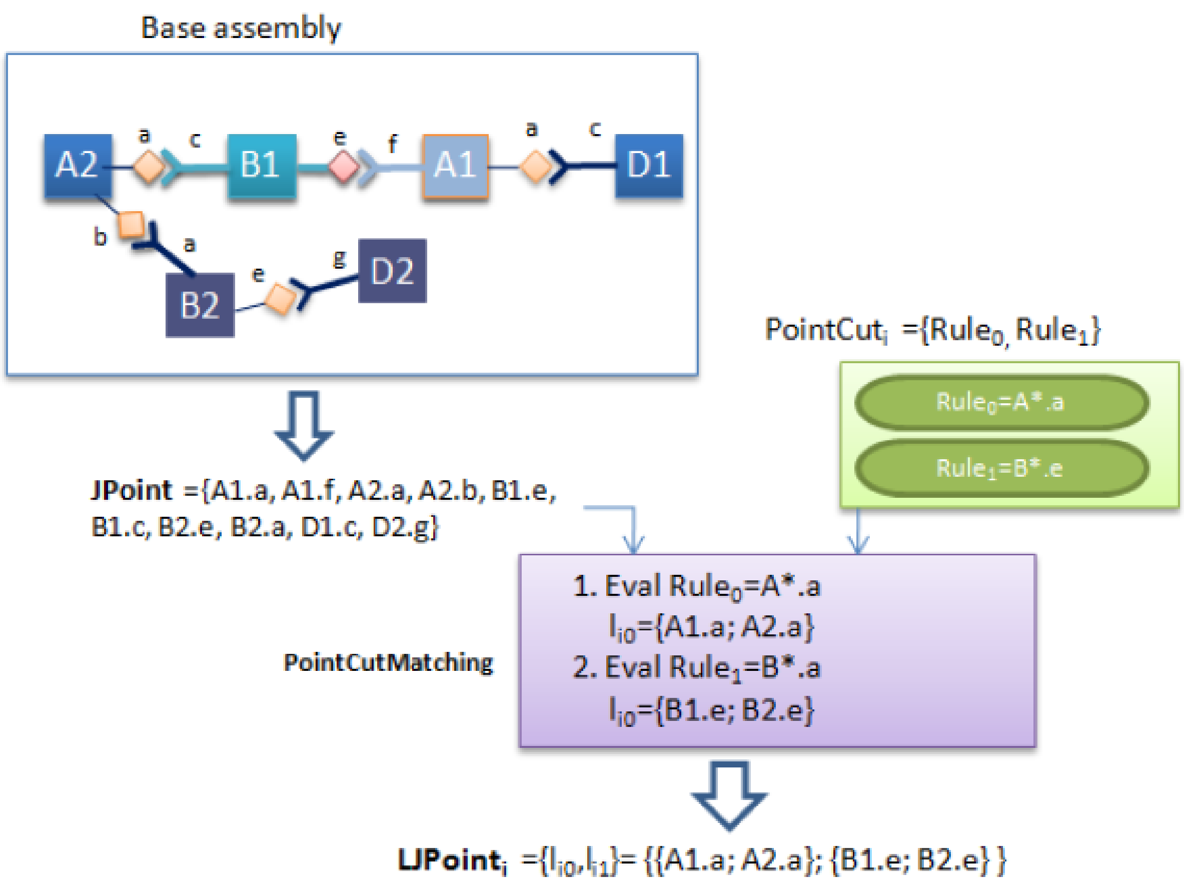


Pointcut Matching (1)

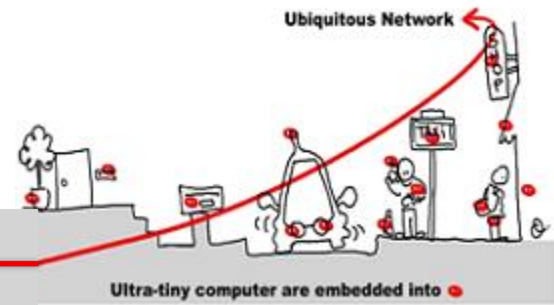
- Pointcut Matching aims to determine in the base assembly all areas where changes described in an AA can be applied.
- Indeed, it is a filter that takes as input all the ports present in the application.
- It is parametrized by the rules defined in the pointcut section of the AA.
- It produces some lists of joinpoints that satisfy each rule and more precisely, a list for each rule.



Pointcut Matching Example



Pointcut Matching Algorithm



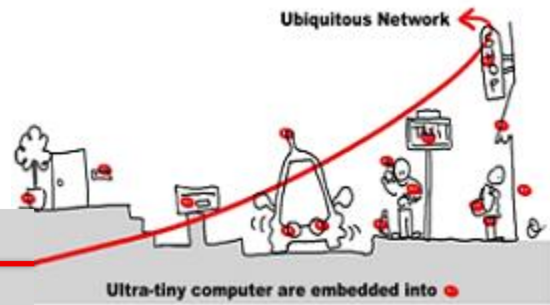
Algorithm 1 $PointcutMatching(JPoint, PointCut_i)$

---

l_{ij} : a list of ports (joinpoint) where $l_{ij} = port_{ij00}, ..., port_{ijnz}$ and j is the number of list which is equal to the number of rules in $PointCut_i$
$LJPoint_i$: a set of joinpoint lists where $LJPoint_i = \{l_{io}, ..., l_{ij}\}$
$JPoint$: the set of ports from the base assembly $port_{00}, ..., port_{nz}$ y.

create $LJPoint_i$
for $s = 0$ to j **do**
  Add a new list l_{is} to $LJPoint_i$
  for $t = 0$ to card(JPoint) **do**
    if JPoint[t] satisfy the rule $Rule_{is}$ **then**
      Add JPoint[t] to the list l_{is}
    end if
  end for
end for

Jointpoint Combination (2)



- Joinpoint combination and filters
- Join Point Combination aims to combine joinpoints that satisfy the pointcut matching according to various policies in order to dene how and where will be duplicated the AA.
- Joinpoints lists created identify all ports that check pointcut rules, in fact a list for each rule. To be applied, advices require at least an element of each list : a combination.
- Thus, an advice can be applied as many times as there are combinations of joinpoints between these lists.

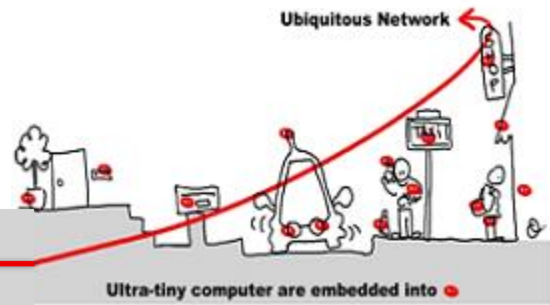
List of list of joinpoint
$LJPoint_i=\{l_{i0},...,l_{ij}\}$
$L_{ij}=\{port_{ij00},...,port_{ijnz}\}$

→ Joinpoint Combination →

$JPointComb_i=\{Comb_{i0},...,Comb_{ij}\}$

$JPCombination(LJPoint_i)=JPointComb_i$

Jointpoint Combination Example



LJPoint$_i$ $=\{l_{i0}, l_{i1}\} = \{\{A1.a; A2.b; A2.c\}; \{B1.e; B2.f\}; \{c1.g\}\}$

Number of combination
Product=3*2*1=6

A1.a	A2.b	A2.c
B1.e	B2.f	
C1.g		

Number of list beginning with A1.a
Product/Card(LJPoint **[0]*)=6/3=2***

$\{A1.a\}\{A1.a\}\{A2.b\}\{A2.b\}\{A2.c\}\{A2.c\}$

Number of list beginning with A1.a and B1.e
Product/Card(LJPoint **[1]*)=2/2=1***

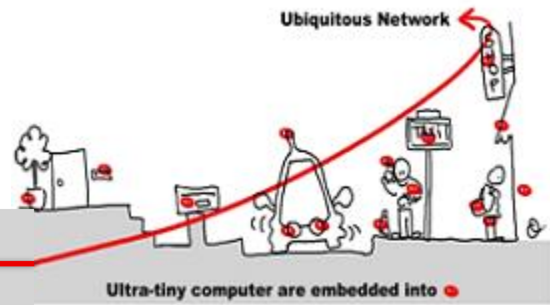
$\{A1.a, B1.e\}\{A1.a, B2.f\}\{A2.b, B1.e\}\{A2.b, B2.f\}\{A2.c, B1.e\}\{A2.c, B1.f\}$

Number of list beginning with A1.a and B1.e and c1.g
Product/Card(LJPoint **[2]*)=1/1=1***

$\{A1.a, B1.e, C1.g\}\{A1.a, B2.f, C1.g\}\{A2.b, B1.e, C1.g\}\{A2.b, B2.f, C1.g\}\{A2.c, B1.e, C1.g\}\{A2.c, B1.f, C1.g\}$

JPointComb$_i$ $=\{Comb_{i0}, Comb_{i1}, Comb_{i2}, Comb_{i3}, Comb_{i4}, Comb_{i5}\}=$
{(A1.a,B1.e,C1.g); (A1.a,B2.f,C1.g); (A2.b,B1.e,C1.g); (A2.b,B2.f,C1.g)
(A2.c,B1.e,C1.g);(A2.c,B1.f,C1.g)}

Jointpoint Combination Algorithm

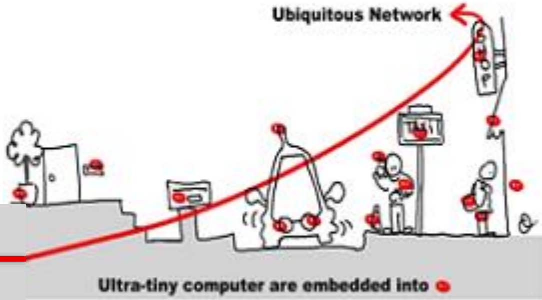


Algorithm 2 JPCombination($LJPoint$)

```
ACombination: list of joinpoint
Product: Integer : number of possible combination
mult : Integer : number of combination using the joinpoint
lcomb: list of combination

mult=1;
create JPointComb
for i = 0 to card(LJPoint) do
   Create lcomb
   ACombination.Clean
   product = product/(card(LJPoint[i]) − 1)
   for j = 1 to card(LJPoint[i]) do
      for k = 0 to product do
         ACombination.Add(LJPoint[i][j])
      end for
   end for
   for j = 1 to mult do
      lcomb.Add(ACombination)
   end for
   JPointComb[i]= lcomb
   mult = mult × (card(LJPoint[i]) − 1)
end for
return JPointComb
```

Filter Algorithm



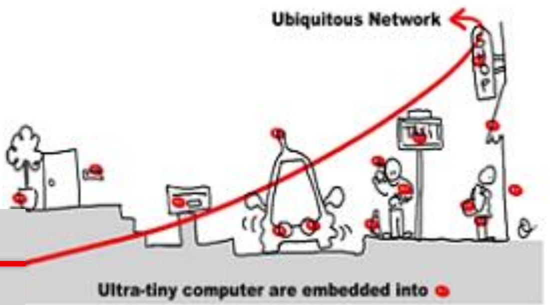
- To Poincut Matching and combination mechanisms may be associated some filters.
- The filter associated to the pointcut matching can withdraw some identified joinpoints.

Algorithm 3 Filter

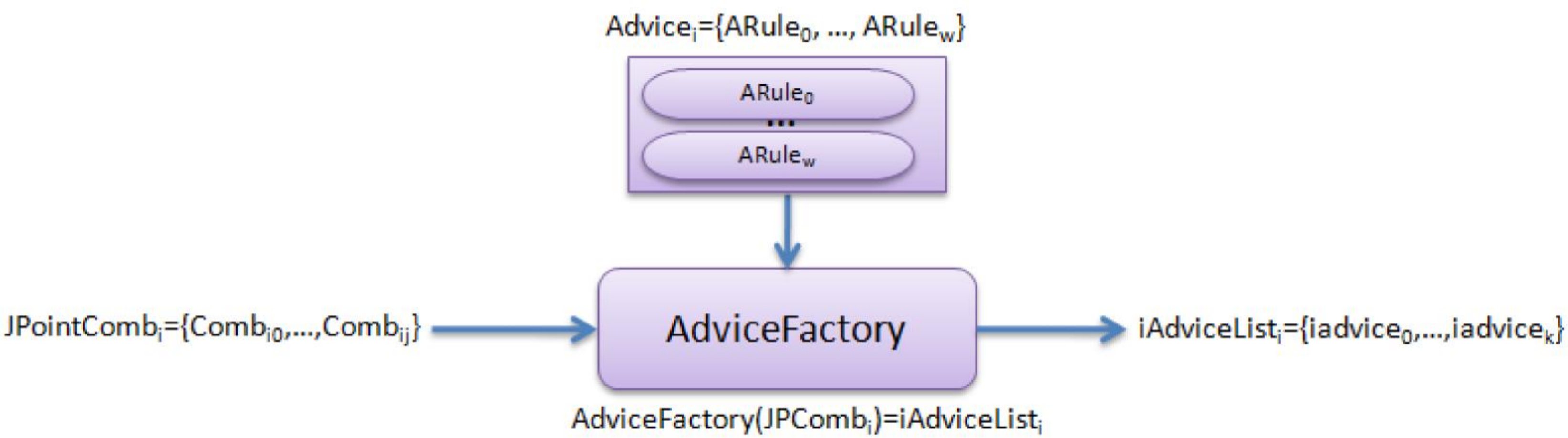
j : number of combination

for $s = 0$ to j **do**
    for $t = 0$ to $card(LJPoint_i[j])$ **do**
        if filtre($l_{is}[t]$) **then**
            $l_{is}.remove(t)$
        end if
    end for
end for

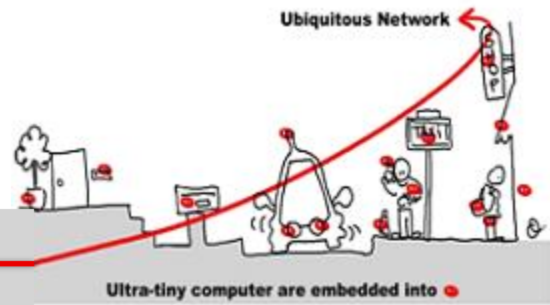
Advice Factory (3)



- AdviceFactory aims to build, from the list of joinpoint combination, instances of advice.
- Thus it create as many instances of advice as possible according to the list of combinations.
- It consist in replacing variables from advice rules with the joinpoint from each combinations.



Advice Factory Algorithm

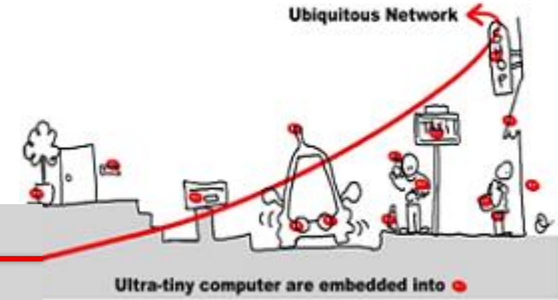


Algorithm 4 AdviceFactory($JPointComb_i$)

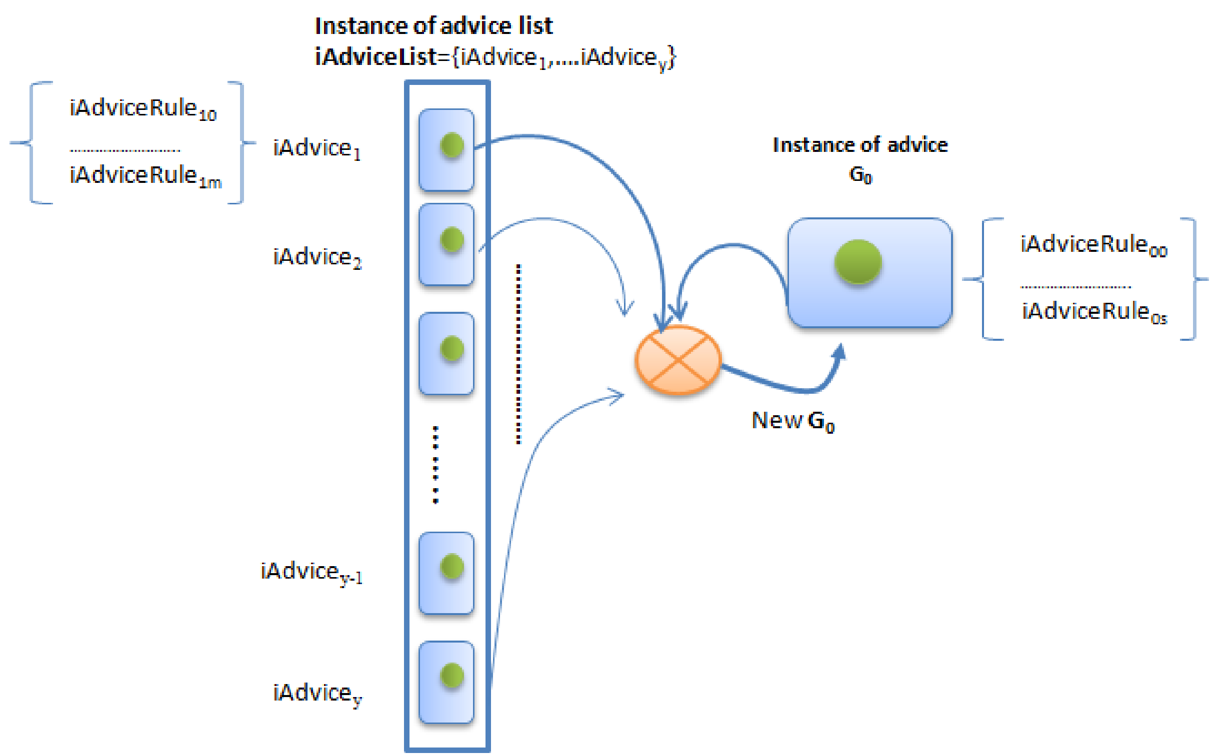
k : number of combination
w : number of advice rules

for $s = 0$ to k **do**
  for $t = 0$ to w **do**
    Replace variable from $ARule[t]$ using $JPointComb[s]$
  end for
end for

Conflict Identification (4)



- Superimposing component assemblies is a mechanism that builds a unique assembly from several intermediates component assemblies (and thus instances of advices).



Superimposition Algorithm

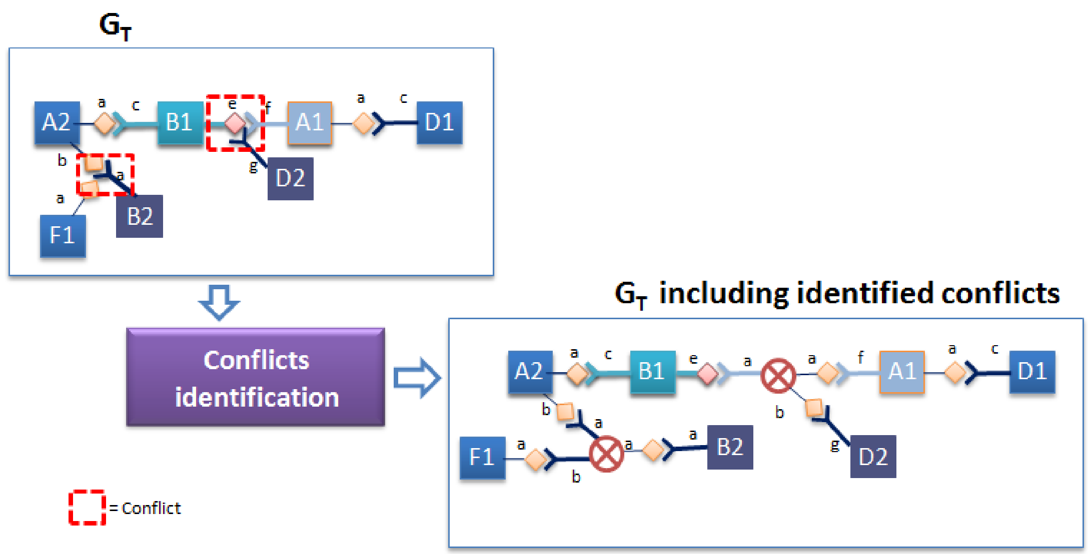
Algorithm 5 Superimpose(iAdviceList)

y : number of instance of advice

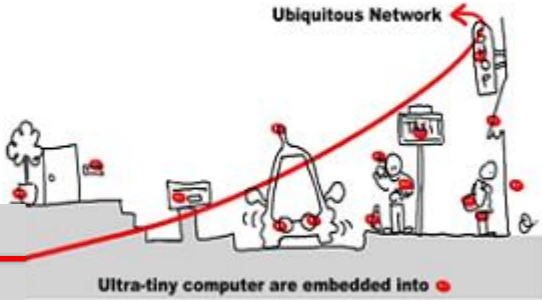
for $d = 0$ to y **do**
    for $t = 0$ to $card(iAdvise_d)$ **do**
        if $iAdvise_d[t] NotInG_0$ **then**
            Add $iAdvise_d[t]$ to G_0
        end if
    end for
end for

Conflict Resolution (5)

- Conflict resolution Conflict resolution aims to solve conflicts occurring when several instances of advices are woven on the same joinpoint (shared joinpoints)



Conflict Resolution Algorithm

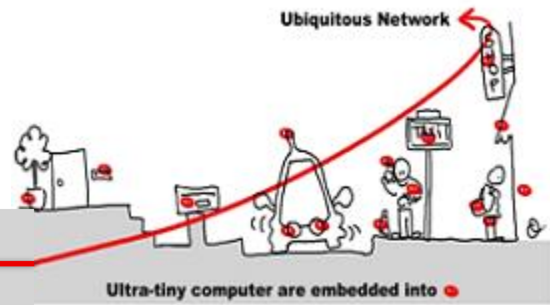


- Depends on the merge strategy
- Then depends on the Merge function

Algorithm 6 ConflictResolution(iAdvice)

for $s = 0$ to $card(List\otimes)$ **do**
  $Merge(List \otimes [s])$
end for

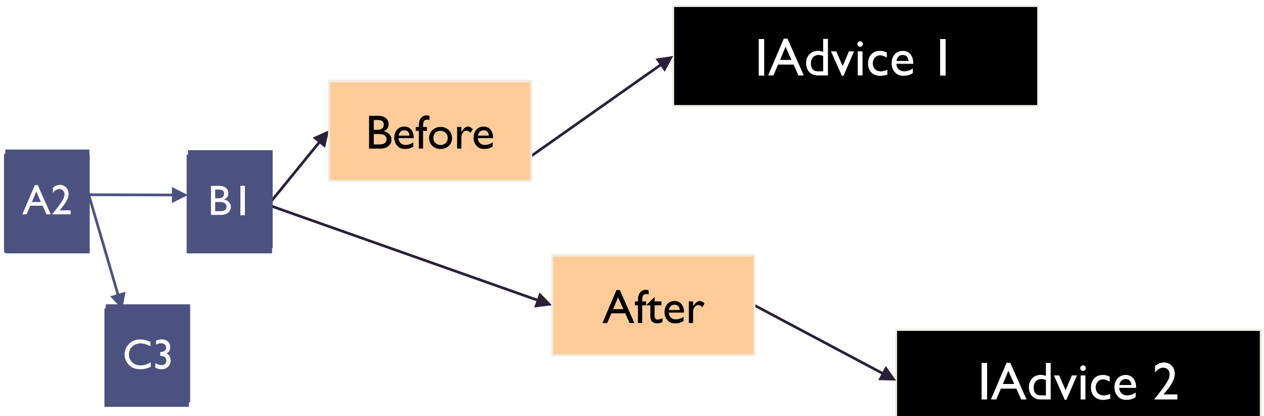
Different kinds of Conflicts Resolution



- External resolution for conflicts
- Internal resolution for conflicts (merge)
  - Example of language to describe advice : ISL4WComp
  - ISL4WComp operators merging matrix
  - Merging logic and its properties

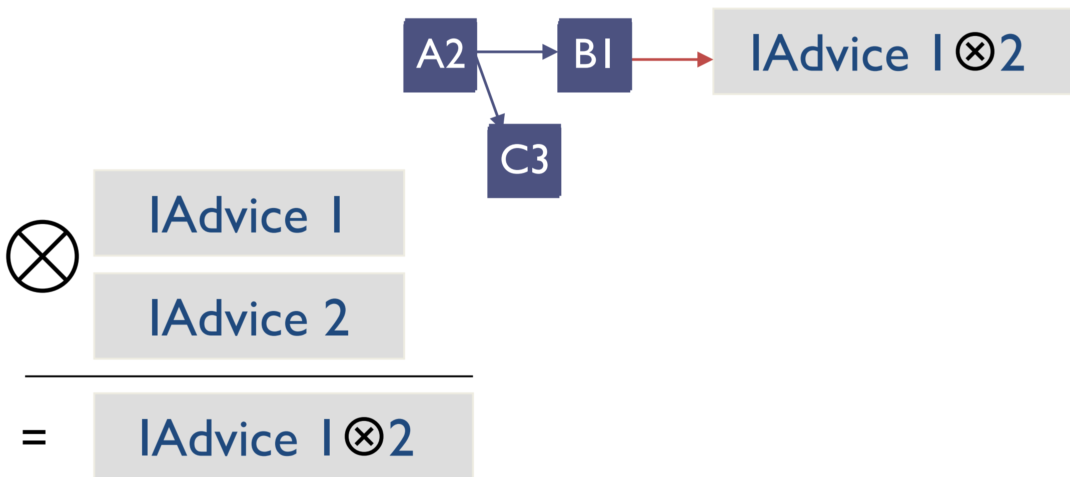
External Composition

- I-Advices are « blackbox »
- I-Advices are scheduled
- Before, After, Around …

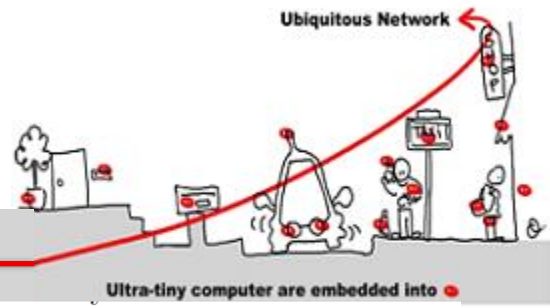


Internal Composition with Merge

- I-Advice are « whitebox »
- Conflicted I-Advices can be merged according to a specific logic and its properties (ex. ISL, ISL4WComp, BSL …)



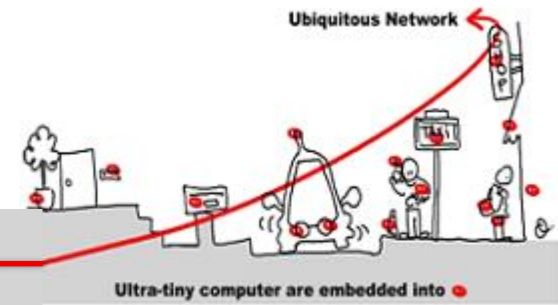
Example of language to describe advice : ISL4WComp



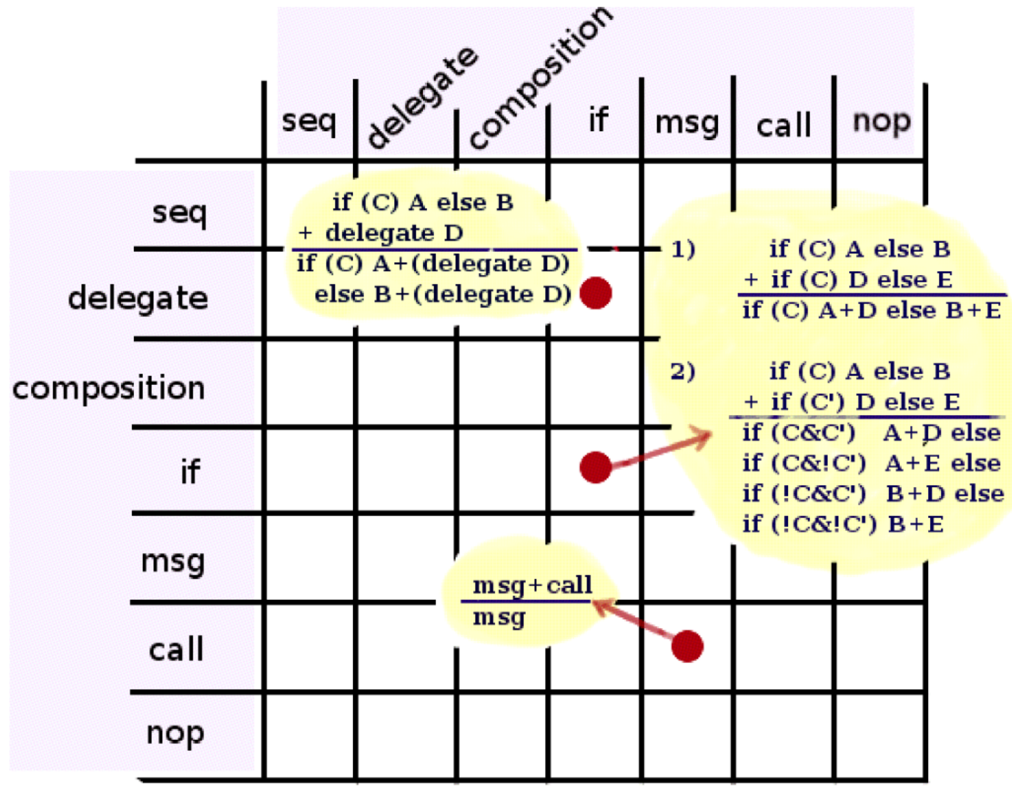
- Operators are :
  - ; (seq)
  - || (par)
  - If / else
  - Nop
  - Call
  - delegate

	Keywords / Operators	Description
port types	*comp.port*	'.' is to separate the name of an instance of component from the name of a port. It describes a provided port.
	comp.^port	'^' at the beginning of a port name describes a required port.
Rules for structural adaptations	*comp : type*	To create a black-box component
	comp : type (prop = val, . . .)	To create a black-box component and to initialize properties
	required_port → (required_port)	To create a link between two ports. The keyword → separates the right part of the rule from its left part
	provided_port → (required_port)	To rewrite an existing link by changing the destination port
Operators (symmetry property, conflicts resolution)	. . . ; . . .	Describes the sequence
	. . . || . . .	To describe that there is no order (parallelism)
	`if (condition) {...} else {...}`	`condition` is evaluated by a black-box component
	`nop`	Nothing to do
	`call`	Allow to reuse the left part of a rule in a rewriting rule
	`delegate`	Allow to specify that an interaction is unique in case of conflict

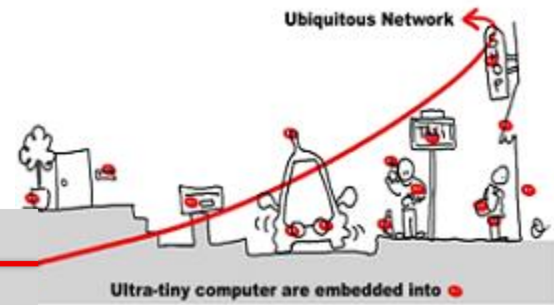
ISL4WComp Operators Merging Matrix



- Merging logic is based on rules to merge semantic trees of the advices
- Each rule gives the result of merging of one operator with another



Merging Logic and its Properties

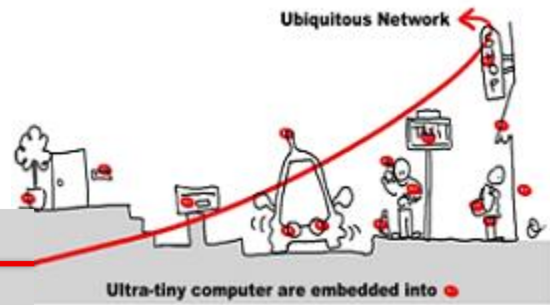


- Example of prooved properties for a composition / merging logic :

Commutativity : $AA0 \otimes AA1 = AA0 \otimes AA1$
Associativity : $(AA0 \otimes AA1) \otimes AA2 = AA0 \otimes (AA1 \otimes AA2)$
Idempotence : $AA0 \otimes AA0 = AA0$

- Weaving mecanism becomes « Symmetric »
- It can apply a set of AA without caring of their order.

Details on AA temporal Validation (response time)



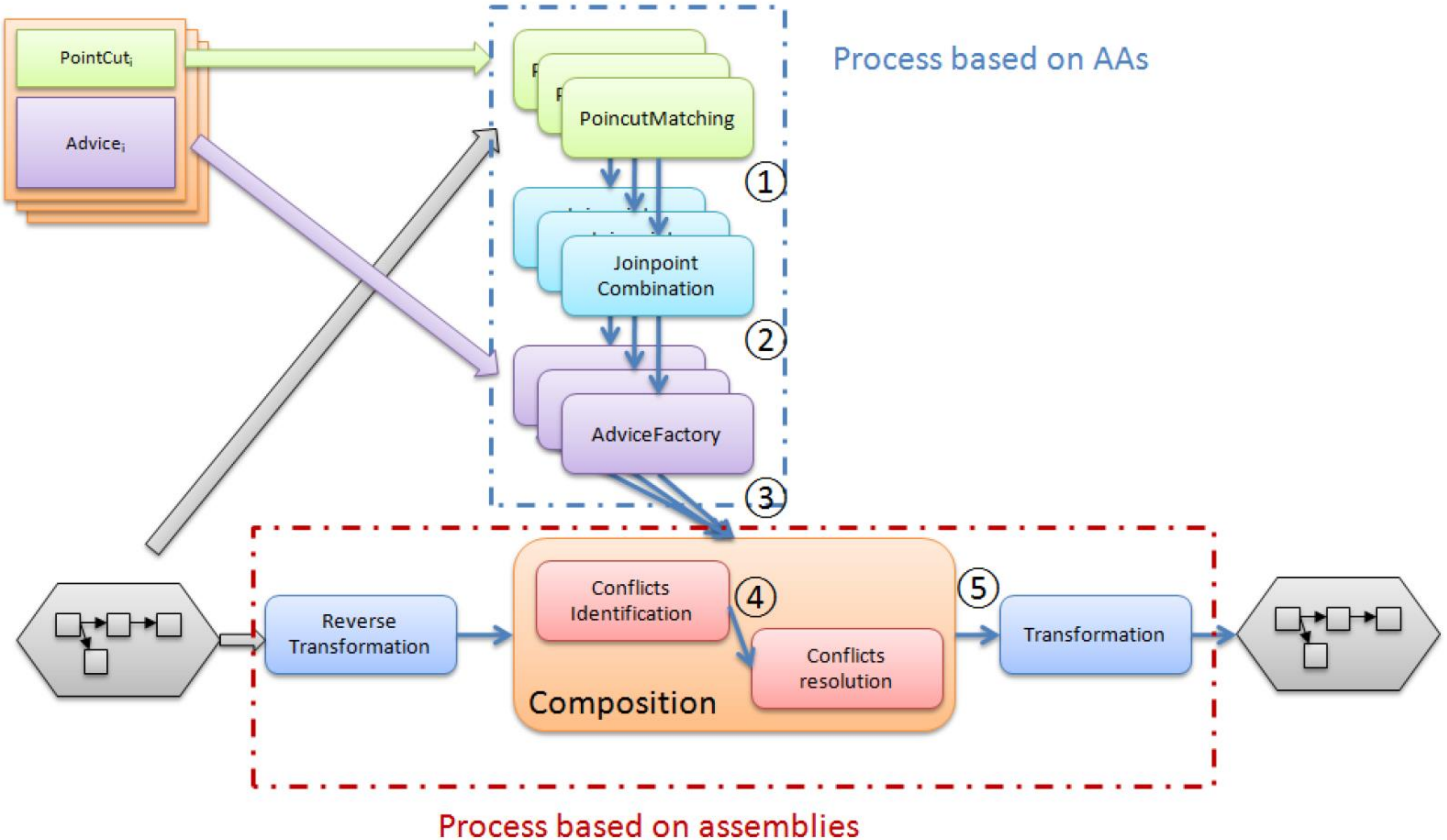
- To response in a timely fashion we need to garantee a minimum response time

- To study the response time of the overall adaptation process based on AA, we need to study :

  – Each algorithm and its complexity

  – Temporal model of the response time and the identification of its parameters

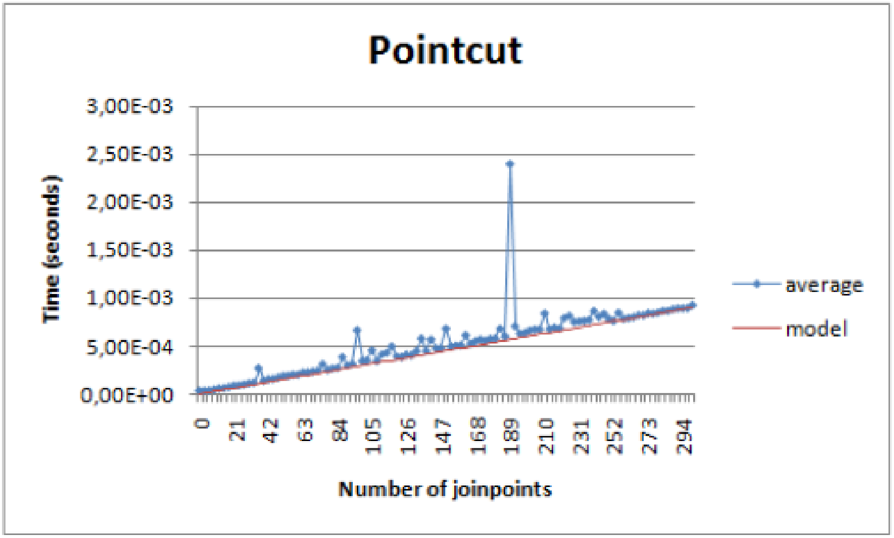
Complete AA Weaving Cycle



Pointcut Matching (1)

A : duration of the PointcutMatching process
$a1; a2$: model parameters
c : number of ports into the base assembly
i : number of AA
j : number of rules in the pointcut section of an AA

$$A = a1 \times \sum_{k=1}^{i} (j.c) + a2$$



Joinpoint Combination (2)

C : Duration of the joinpoint combination process

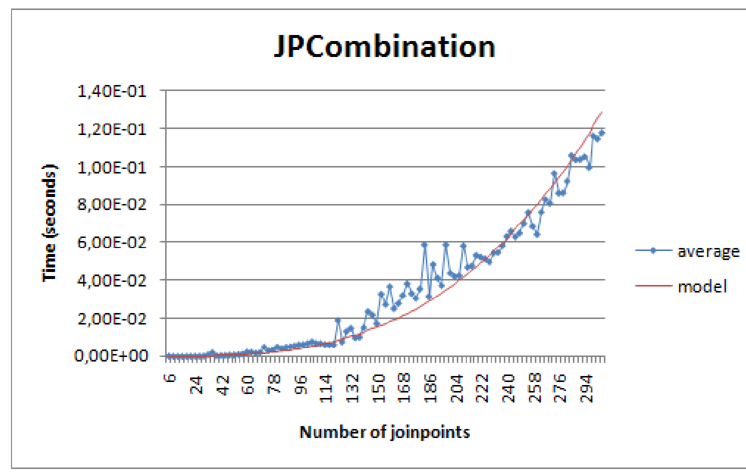
$a1; a2$: model parameters

$JPoint$: the set of joinpoints

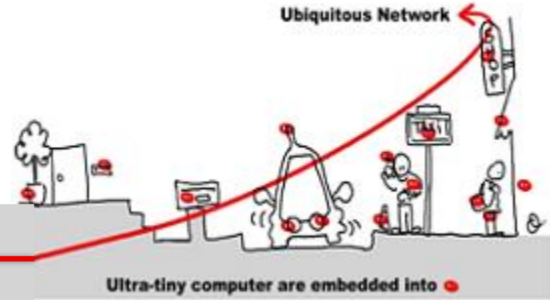
i : number of AA

j : number of rules in the pointcut section of an AA

$$C = a1 \times \sum_{k=1}^{i} (card(JPoint)^{j}) + a2$$



Advice Factory (3)



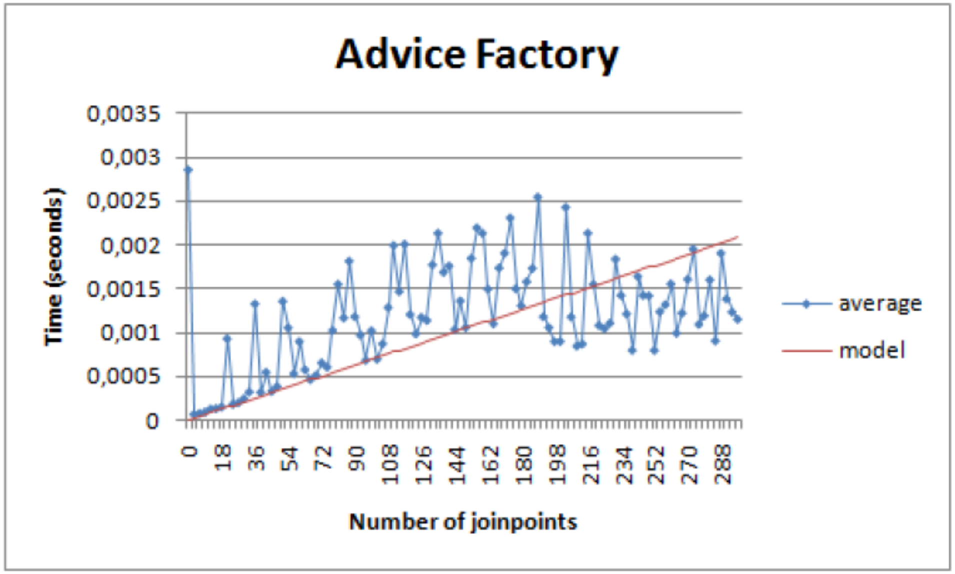
A : duration of instance of advice generation

k : number of combination

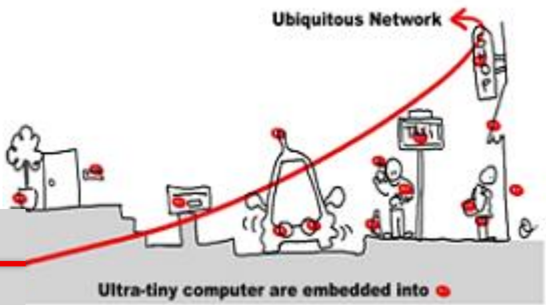
w : number of advice rule

$a1;a2$: model parameters

$$A = a1 \times \sum_{k=1}^{i}(kw) + a2$$



Conflict Identification (4)



S : duration of instance of advice superposition

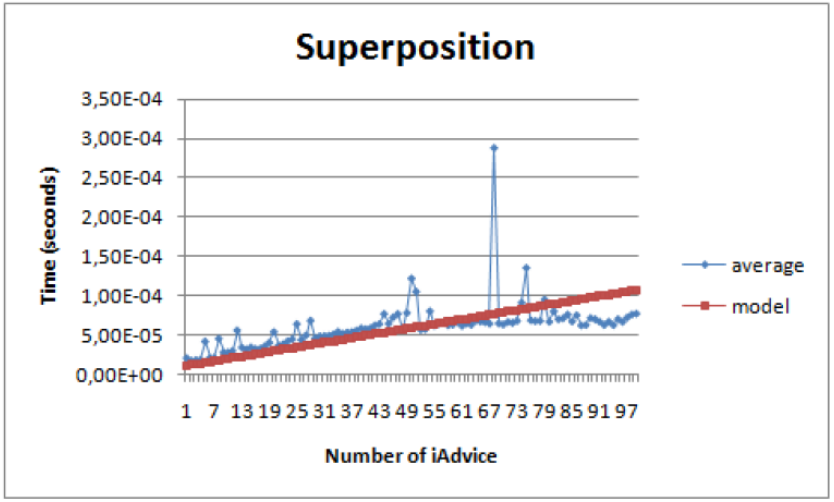
y : number of instance of advice

w : number of advice rule

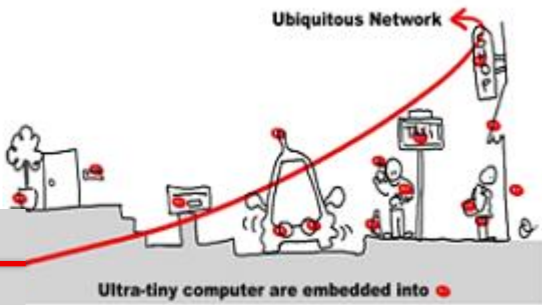
$g0$: number of rules in the initial instance of advice

$a1;a2$: model parameters

$$S = a1 \times \sum_{i=1}^{y}(w_i.g_0) + a2$$



Conflict Resolution, Example with ISL4WComp



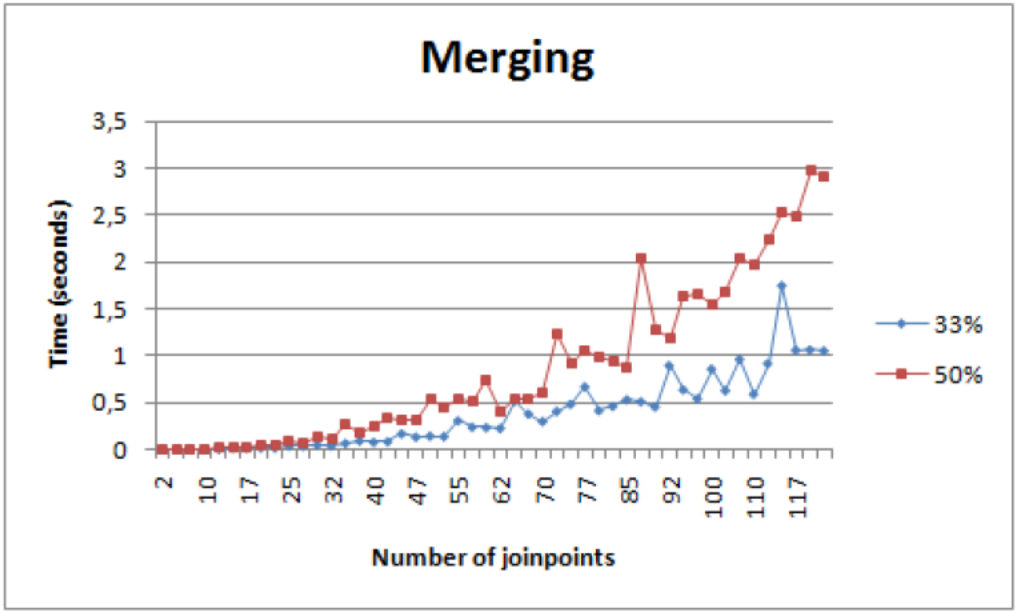
- Duration of instance of advice merging

$$F : \text{duration of instance of advice merging}$$
$$g_o : \text{number of rules in the base assembly}$$
$$y : \text{number of instance of advice}$$
$$w : \text{number of advice rule}$$
$$a1 : \text{model parameters}$$
$$p_i : \text{merging probability}$$
$$M : \text{Cost of merging}$$

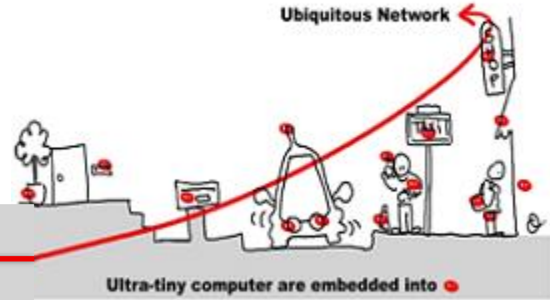
$$F = a1.g_0 \times \sum_{i=1}^{y} w_i.p_i.M$$

Conflict Resolution, Example with ISL4WComp

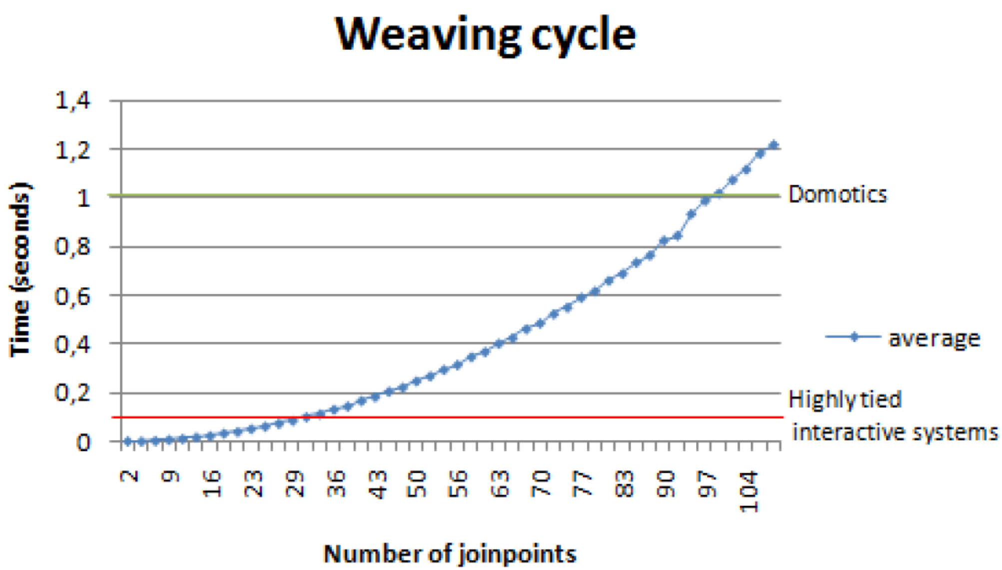
- Conflict resolution processing response time.
- Experiments : Response time average with C=33% and 50%



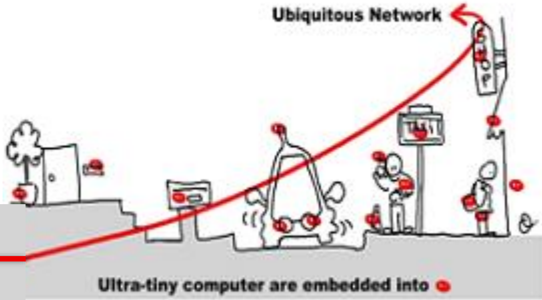
Synthesis : Overall Weaving Cycle



- Weaving cycles duration can be formally define as follows : W(n) = D(n)+C(n)+A(n)+S(n)+F(n) where n is the set of joinpoints from the base assembly.

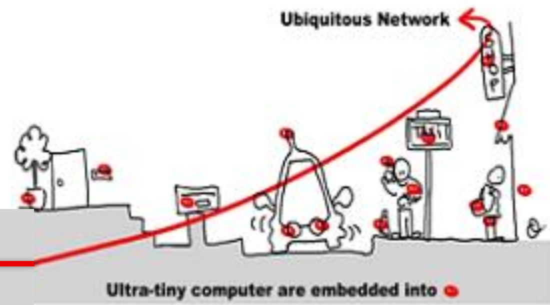


DEMO and future works



- Simple Demo : AA in WComp
- Other DEMO : AA in WComp

Future Works in WComp



- Multi-Domain weaving for AA to adapt Mobile Workers applications (Cf. CONTINUUM project of the French National Research Agency towards « Continuity of Service »)

- Adaptation trigered by physical environment variations

- Semantic adaptation : Improving of Pointcut Matching algorithms from Ontology-Based Metadata and mapping between ontologies (Cf. Continuum project of the French National Research Agency towards « Continuity of Service »)

7.4 Questions ?