Architecture: modeling and mapping
- Metropolis approach -

Yoshi Watanabe
Cadence Berkeley Labs

for EE249 class in October 2003
Outline

- Metropolis meta-model
- Meta-model execution semantics
- Architecture: what do we care about?
- Mapping
- Platforms: recursive paradigm of platform based design
Metropolis meta-model

Concurrent specification with a formal execution semantics:

- **Computation** : $f : X \rightarrow Z$

- **Communication** : state evaluation and manipulation

- **Coordination** : constraints over concurrent actions
process P{
    port reader X;
    port writer Y;
    thread()
    while(true){
        ...
        z = f(X.read());
        Y.write(z);
    }
}
process P{
    port reader X;
    port writer Y;
    thread(){
        while(true){
            ... 
            z = f(X.read());
            Y.write(z);
        }
    }
}

interface reader extends Port{
    update int read();
    eval int n();
}

interface writer extends Port{
    update void write(int i);
    eval int space();
}

medium M implements reader, writer{
    int storage;
    int n, space;
    void write(int z){
        await(space>0; this.all ; this.all)
        n=1; space=0; storage=z;
    }
    word read(){ ... }
}

Meta-model : network
process P{
    port reader X;
    port writer Y;
    thread(){
        while(true){
            ...
            z = f(X.read());
            Y.write(z);
        }
    }
}

medium M implements reader, writer{
    int storage;
    int n, space;
    void write(int z){
        await(space>0; this.all ; this.all)
        n=1; space=0; storage=z;
    }
    word read(){ ... }
}
Meta-model: execution semantics

- Processes take *actions*.
  - statements, function calls, function arguments, top-level expressions (e.g. assignment, conditional, RHS of an assignment):
    
    $$y = z + \text{port.f}();, \ y = z + \text{port.f}(), \ z + \text{port.f}(), \ \text{port.f}(), \ i < 10, \ldots$$

- An *execution* of a given network is a sequence of vectors of *events*.
  - *event*: the beginning of an action, e.g. B(process, port.f()),
    - the end of an action, e.g. E(process, port.f()), or null N
  - the *i*-th component of a vector is an event of the *i*-th process

- An execution is *legal* if
  - it satisfies all coordination constraints, and
  - it is accepted by all action automata.
**Execution semantics**

Action automaton:

- one for each action of each process
  - defines the set of sequences of events that can happen in executing the action

- a transition corresponds to an event:
  - it may update shared memory variables:
    - process and media member variables
    - values of actions-expressions
  - it may have guards that depend on states of other action automata and memory variables

- each state has a self-loop transition with the null N event.

- all the automata have their alphabets in common:
  - transitions must be taken together in different automata, if they correspond to the same event.
Action automata

- \( y = x + 1; \)

\( y = x + 1 \)

\( x + 1 \)

\( V_{x+1} \)

\( y \)

\( x \)

\( B\ y = x + 1 \)

\( B\ x + 1 \)

\( E\ x + 1 \)

\( E\ y = x + 1 \)

\( y := V_{x+1} \)

\( y := any \)

\( * = \text{write } y \)

\( V_{x+1} := x + 1 \)

\( V_{x+1} := any \)

\( B\ y = x + 1 \ N \)
Action automata

- $y = x + 1;$

$y = x + 1$

- $B \ y = x + 1$
- $B \ x + 1$
- $E \ x + 1$
- $E \ y = x + 1$
- $y := V_{x+1}$

$E \ y = x + 1$
- $y := \text{any}$

$x + 1$

- $B \ x + 1$
- $E \ x + 1$
- $V_{x+1} := x + 1$

- $E \ x + 1$
- $V_{x+1} := \text{any}$

$V_{x+1}$
- $0$

$y$
- $0$

$x$
- $0$

$B \ y = x + 1$
- $N$

$B \ x + 1$
- $N$

$N$

$N$

$*$ = write $y$
Action automata

- $y = x + 1$

```
y = x + 1
```

```
x + 1
```

```
V_{x+1} := x + 1
```

```
V_{x+1} := any
```

```
B y = x + 1 N B x + 1 N N E x + 1
```

* = write $y$
Action automata

- \( y=x+1; \)

\[
\begin{array}{c}
\text{y=x+1} \\
\text{B y=x+1} \\
\text{C} \\
\text{B x+1} \\
\text{E x+1} \\
\text{C} \\
\text{E y=x+1} \\
\text{y:=any} \\
\end{array}
\]

* = write \( y \)

\[
\begin{array}{c}
x+1 \\
\text{B x+1} \\
\text{C} \\
\text{write x} \\
\text{E x+1} \\
\text{V_{x+1}:=x+1} \\
\text{C} \\
\text{E x+1} \\
\text{V_{x+1}:=any} \\
\end{array}
\]

\[
\begin{array}{c|c|c}
V_{x+1} & 0 & 1 \\
y & 0 & 1 \\
x & 0 & 1 \\
\end{array}
\]

\[
\begin{array}{c|c|c|c|c|c|c}
& \text{B y=x+1} & \text{N} & \text{B x+1} & \text{N} & \text{N} & \text{E x+1} & \text{E y=x+1} \\
\end{array}
\]
**Meta-model: network**

process P{
    port reader X;
    port writer Y;
    thread(){
        while(true){
            ...
            z = f(X.read());
            Y.write(z);
        }
    }
}

medium M implements reader, writer{
    int storage;
    int n, space;
    void write(int z){
        await(space>0; this.all ; this.all)
        n=1; space=0; storage=z;
    }
    word read(){ ... }
}
Outline

- Metropolis meta-model
- Meta-model execution semantics
- Architecture: what do we care about?
- Mapping
- Platforms: recursive paradigm of platform based design
Meta-model: architecture components

An architecture component specifies services, i.e.

- what it *can* do:
  - interfaces, methods, coordination (awaits, constraints), network
- how much it *costs*:
  - quantities, annotated with events, related over a set of events

```java
interface BusMasterService extends Port {
    update void busRead(String dest, int size);
    update void busWrite(String dest, int size);
}
```

```java
medium Bus implements BusMasterService ...{
    port BusArbiterService Arb;
    port MemService Mem; ...
    update void busRead(String dest, int size) {
        if(dest== ...) Mem.memRead(size);
    }
    ...
}
Meta-model: quantities

- The domain $D$ of the quantity, e.g. $real$ for the global time,
- The operations and relations on $D$, e.g. subtraction, $<$, $=$,
- The function from an event instance to an element of $D$,
- Axioms on the quantity, e.g.

  the global time is non-decreasing in a sequence of vectors of any feasible execution.

```java
class GTime extends Quantity {
    double t;
    double sub(double t2, double t1){...}
    double add(double t1, double t2){...}
    boolean equal(double t1, double t2){ ... }
    boolean less(double t1, double t2){ ... }
    double A(event e, int i){ ... }
    constraints{
        forall(event e1, event e2, int i, int j):
            GXI.A(e1, i) == GXI.A(e2, j) -> equal(A(e1, i), A(e2, j)) &&
            GXI.A(e1, i) < GXI.A(e2, j) -> (less(A(e1, i), A(e2, j)) || equal(A(e1, i), A(e2. j))));
    }
```
**Meta-model: architecture components**

```plaintext
interface BusMasterService extends Port {
    update void busRead(String dest, int size);
    update void busWrite(String dest, int size);
}
```

```plaintext
medium Bus implements BusMasterService ...{
    port Bus Arbiter Service Arb;
    port Mem Service Mem; ...
    update void busRead(String dest, int size) {
        if(dest== ...) Mem.memRead(size);
        <![
            constraint{
                forall i: GTime.A(B(thisthread, this.memRead), i) ==
                GTime.A(B(thisthread, this.busRead), i)
                + BUSCLOCKCYCLE;
            }
        ]]
    }
    ...
```
Quantity resolution

- Suppose that quantities and constraints are specified; how do we use them?
  - In general, there are many event vectors that can happen next.
  - Some of the events may be subject to quantity constraints.

Ex:

A process P1 executes a method foo(), with
  \[\text{forall } i: \text{GTime}.A(E(P1, \text{foo}), i) = \text{GTime}.A(B(P1, \text{foo}), i) + 15.\]

A process P2 executes a method bar(), with
  \[\text{forall } i: \text{GTime}.A(E(P2, \text{bar}), i) = \text{GTime}.A(B(P2, \text{bar}), i) + 5.\]

In the current execution, suppose that
- the next possible events for P1 are E(P1, foo) or null N, and the latest B(P1, foo) took place with GTime.A(B(P1, foo), LAST) = 5.
- the next possible events for P2 are E(P2, bar) or null N, and the latest B(P2, bar) took place with GTime.A(B(P2, bar), LAST) = 10.

Then, because of the axiom on non-decreasing time, \([P1, P2] = [N, E(P2, bar)]\) is the only choice for the next vector among four possible event vectors, or P2 will block permanently.
Quantity resolution

The 2-step method to resolve quantities at each state of a network being executed:

1. quantity requests

   for each process $P_i$, for each event $e$ that $P_i$ can take, find all the quantity constraints on $e$.

   In the meta-model, this is done by explicitly requesting quantity annotations at the relevant events, i.e. Quantity.request(event, requested quantities).

2. quantity resolution
Meta-model: architecture components

interface BusMasterService extends Port {
    update void busRead(String dest, int size);
    update void busWrite(String dest, int size);
}

medium Bus implements BusMasterService {...
    port BusArbiterService Arb;
    port MemService Mem; ...
    update void busRead(String dest, int size) {
        if(dest== ...) Mem.memRead(size);
        [[
            GTime.request(B(thisthread, this.memRead),
            BUSCLKCYCLE +
            GTime.A(B(thisthread, this.busRead), LAST));
        ]]
    }
    ...
}
**Quantity resolution**

The 2-step approach to resolve quantities at each state of a network being executed:

1. **quantity requests**
   
   For each process $Pi$, for each event $e$ that $Pi$ can take, find all the quantity constraints on $e$.

   In the meta-model, this is done by explicitly requesting quantity annotations at the relevant events, i.e. `Quantity.request(event, requested quantities).`

2. **quantity resolution**
   
   Find a vector made of the candidate events and a set of quantities annotated with each of the events, such that the annotated quantities satisfy:
   - all the quantity requests, and
   - all the axioms of the Quantity types.

   In the meta-model, this is done by letting each Quantity type implement a `resolve()` method, and the methods of relevant Quantity types are iteratively called.
   - theory of fixed-point computation
Quantity resolution

- The 2-step approach is same as how schedulers work, e.g. OS schedulers, BUS schedulers, BUS bridge controllers.
- Semantically, a scheduler can be considered as one that resolves a quantity called execution index.
- Two ways to model schedulers:
  1. As processes:
     - explicitly model the scheduling protocols using the meta-model building blocks
     - a good reflection of actual implementations
  2. As quantities:
     - use the built-in request/resolve approach for modeling the scheduling protocols
     - more focus on resolution (scheduling) algorithms, than protocols: suitable for higher level abstraction models
**Meta-model: architecture components**

An architecture component specifies *services*, i.e.

- what it *can* do
- how much it *costs*

```java
interface BusMasterService extends Port {
    update void busRead(String dest, int size);
    update void busWrite(String dest, int size);
}
```

```java
interface BusArbiterService extends Port {
    update void request(event e);
    update void resolve();
}
```

```java
medium Bus implements BusMasterService {...
    port BusArbiterService Arb;
    port MemService Mem; ...
    update void busRead(String dest, int size) {
        if(dest== ... ) Mem.memRead(size);
        [[Arb.request(B(thisthread, this.busRead));
         GTime.request(B(thisthread, this.memRead),
                      BUSCLKCYCLE +
                      GTime.A(B(thisthread, this.busRead))));
         ]]
    }
...
```

```java
scheduler BusArbiter extends Quantity
    implements BusArbiterService {
    update void request(event e){ ... }  
    update void resolve() { //schedule }
    ...
}
```
**Meta-model: architecture network**

Architecture network specifies configurations of architecture components.
- instantiates arch. components,
- connects them,
- add processes that *use* the services

What do these processes represent?
**Meta-model: architecture network**

```plaintext
medium AbsCpu implements AbsCpuService {
  void cpuRead() { ... }
  void cpuWrite() { ... }
  void exec() { ... }
}

process Task1{
  void thread() {
    while (true) {
      await {
        (true; ; ;) cpuRead();
        (true; ; ;) exec();
        (true; ; ;) cpuWrite();
      }
    }
  }
}

process TaskN{
  thread() {
    while (true) {
      await {
        (true; ; ;) cpuRead();
        (true; ; ;) exec();
        (true; ; ;) cpuWrite();
      }
    }
  }
}
```
**Meta-model: architecture network**

Architecture network specifies configurations of architecture components.

- instantiates arch. components,
- connects them,
- add processes that *use* the services

What do these processes represent?

: mapping processes
**Meta-model: mapping processes**

**Function process**

```java
process P{
    port reader X;
    port writer Y;
    thread(){
        while(true){
            ...
            z = f(X.read());
            Y.write(z);
        }
    }
}
```

**Mapping process**

```java
process MapP{
    port CpuService Cpu;
    void readCpu(){
        Cpu.exec();  Cpu.cpuRead();
    }
    void mapf(){  ...
    ...
    thread(){
        while(true){
            await {
                (true; ; ;) readCpu();
                (true; ; ;) mapf();
                (true; ; ;) readWrite();
            }
        }
    }
}
```

\[ B(P, X.read) \leftrightarrow B(MapP, readCpu); \quad E(P, X.read) \leftrightarrow E(MapP, readCpu); \]
\[ B(P, f) \leftrightarrow B(MapP, mapf); \quad E(P, f) \leftrightarrow E(MapP, mapf); \]
\[ \ldots \]
Meta-model: mapping processes

How do we identify methods like readCpu(), mapf() in the mapping process?

This is a compilation problem!

<table>
<thead>
<tr>
<th>Function process</th>
<th>Mapping process</th>
</tr>
</thead>
<tbody>
<tr>
<td>process P{</td>
<td>process MapP{</td>
</tr>
<tr>
<td>port reader X;</td>
<td>port CpuService Cpu;</td>
</tr>
<tr>
<td>port writer Y;</td>
<td>void readCpu(){</td>
</tr>
<tr>
<td>thread(){</td>
<td>Cpu.exec(); Cpu.cpuRead();</td>
</tr>
<tr>
<td>while(true){</td>
<td>}</td>
</tr>
<tr>
<td>...</td>
<td>void mapf(){ ...}</td>
</tr>
<tr>
<td>z = f(X.read());</td>
<td>}</td>
</tr>
<tr>
<td>Y.write(z);</td>
<td>...</td>
</tr>
<tr>
<td>}}</td>
<td>}</td>
</tr>
</tbody>
</table>

- If the function and services can be decomposed into common objects (subject graphs), then conventional compilation techniques could be applied.
  - this happens for lower levels of abstraction.
- Often requires manual translation: a good research topic.
Meta-model: mapping network

MyMapNet

\[ B(P1, M.\text{write}) \leftrightarrow B(mP1, mP1.\text{writeCpu}); \quad E(P1, M.\text{write}) \leftrightarrow E(mP1, mP1.\text{writeCpu}); \]
\[ B(P1, P1.f) \leftrightarrow B(mP1, mP1.\text{mapf}); \quad E(P1, P1.f) \leftrightarrow E(mP1, mP1.\text{mapf}); \]
\[ B(P2, M.\text{read}) \leftrightarrow B(P2, mP2.\text{readCpu}); \quad E(P2, M.\text{read}) \leftrightarrow E(mP2, mP2.\text{readCpu}); \]
\[ B(P2, P2.f) \leftrightarrow B(mP2, mP2.\text{mapf}); \quad E(P2, P2.f) \leftrightarrow E(mP2, mP2.\text{mapf}); \]
**Meta-model: mapping network**

Remember: a meta-model network specifies a non-deterministic behavior.

- The non-determinism in the function:
  - event coordination by concurrency, awaits
- The non-determinism in the architecture:
  - order of services, data to be applied

Mapping reduces the non-determinism:

- The function specifies the control flow and data for each process
- The architecture defines coordination of concurrent events by quantity resolutions.
Meta-model: recursive paradigm of platforms

B(Q2, S.cdx) <-> B(Q2, mQ2.excCpu); E(Q2, M.cdx) <-> E(mQ2, mQ2.excCpu);
B(Q2, Q2.f) <-> B(mQ2, mQ2.mapf); E(Q2, P2.f) <-> E(mQ2, mQ2.mapf);

MyMapNetlist1
B(P1, M.write) <-> B(mP1, mP1.writeCpu);
B(P1, P1.f) <-> B(mP1, mP1.mapf); E(P1, P1.f) <-> E(mP1,)
B(P2, M.read) <-> B(P2, mP2.readCpu);
E(P2, P2.f) <-> E(mP2, mP2.mapf);
Meta-model: summary

- Concurrent specification with a formal execution semantics
- Feasible executions of a network: sequences of event vectors
- Quantities can be defined and annotated with events, e.g. time, power, global indices, composite quantities.
- Concurrent events can be coordinated in terms of quantities:
  - logic can be used to define the coordination,
  - algorithms can be used to implement the coordination.
- The mechanism of event coordination wrt quantities plays a key role:
  - architecture modeling as service with cost,
  - a mapping coordinates executions of function and architecture network,
  - a refinement through event coordination provides a platform.