Architecture: modeling and mapping
- Metropolis approach -

Yoshi Watanabe
Cadence Berkeley Labs
Guang Yang
EECS, UC Berkeley

for EE249 class in September 2003
Outline

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

- **Methodologies**
  - Multi-media, wireless communication, mechanical controls, processors

- **Meta-model Library**
  - Models of computation

- **Infrastructure**
  - Metropolis meta-model
    - language
    - modeling mechanisms
  - Meta-model compiler

- **Tools**
  - Simulator
  - QSS
  - PIG
  - STARS
  - SPIN
  - ...
Metropolis meta-model

Concurrent specification with a formal execution semantics:

- process
- port
- interface
- medium
- netlist
- constraint
- quantity
**Meta-model : netlist**

- **MyNetlist**

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

- thread() generates a sequence of events for the process
- communication only through ports:
  - no global variable
  - no update on static fields outside yourself
- a port is specified with an interface type
  - all and only the methods of the interface can be called through the port
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(){ ... }
}
process P{
    port reader X;
    port writer Y;
    thread(){
        while(true){
            ...
            z = f(X.read());
            Y.write(z);
        }}}}
Meta-model: execution semantics

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

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

- An execution is *feasible* 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;

\begin{align*}
\text{y} & = \text{x} + 1 \\
\text{B y} & = \text{x} + 1 \\
\text{B x} & = \text{x} + 1 \\
\text{E x} & = \text{x} + 1 \\
\text{C} & = \text{y} = \text{x} + 1 \\
\text{y} & := \text{V}_{\text{x} + 1} \\
\text{E y} & = \text{x} + 1 \\
\text{E y} & = \text{any} \\
\text{E} & = \text{any} \\
\text{V} & = \text{x} + 1 \\
\text{V} & = \text{any} \\
\text{x} & = \text{any} \\
\text{x} & = \text{any} \\
\text{B y} & = \text{x} + 1 \\
\text{N} & = \text{write} y
\end{align*}
Action automata

- \( y = x + 1; \)

\[
\begin{align*}
\text{y} &= x + 1 \\
\text{B} &\quad \text{y} = x + 1 \\
\text{C} &\quad \text{B} x + 1 \\
\text{E} &\quad \text{E} x + 1 \\
\text{C} &\quad \text{E} y = x + 1 \\
\end{align*}
\]

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

\[
\begin{align*}
\text{x} + 1 &\quad \text{B} x + 1 \\
\text{E} &\quad \text{E} x + 1 \\
\text{C} &\quad \text{E} x + 1 \\
\text{C} &\quad \text{E} y = x + 1 \\
\text{E} &\quad \text{y} = \text{any} \\
\end{align*}
\]

\( \ast = \text{write x} \)

\[
\begin{align*}
\text{V}_{x + 1} &\quad 0 \\
\text{y} &\quad 0 \\
\text{x} &\quad 0 \\
\end{align*}
\]

\[
\begin{align*}
\text{B} y &= x + 1 \\
\text{B} x + 1 &\quad \text{N} \\
\end{align*}
\]
Action automata

- $y = x + 1$;

\[
\begin{align*}
y &= x + 1 \\
x + 1 &\xrightarrow{\text{write } x} \text{write } x + 1 \\
V_{x+1} &\xrightarrow{\text{write } x} y := y + 1 \\
&\xrightarrow{\text{write } y} y := V_{x+1} \\
&\xrightarrow{\text{write } any} y := \text{any}
\end{align*}
\]

$* = \text{write } y$
Action automata

- $y = x + 1$;

\[ y = x + 1 \]

\[ x + 1 \]

\[ V_{x+1} \]

\[ 0 \]

\[ 0 \]

\[ 0 \]

\[ B y = x + 1 \]

\[ B x + 1 \]

\[ E x + 1 \]

\[ E y = x + 1 \]

\[ y := V_{x+1} \]

\[ * = \text{write } y \]

\[ x \]

\[ V_{x+1} := x + 1 \]

\[ \text{write } x \]

\[ E x + 1 \]

\[ E x + 1 \]

\[ V_{x+1} := \text{any} \]

\[ 1 \]

\[ 1 \]

\[ 1 \]

\[ 0 \]

\[ 1 \]

\[ 0 \]

\[ 0 \]

\[ B y = x + 1 \]

\[ B x + 1 \]

\[ N \]

\[ N \]

\[ N \]

\[ E x + 1 \]

\[ E y = x + 1 \]
Action automata

- \( y = x + 1; \)

\[ y = x + 1 \]

\[ x + 1 \]

\[ V_{x+1} 0 \]
\[ y 0 \]
\[ x 0 \]

\[ B \ y = x + 1 \]

\[ B \ x + 1 \]
\[ E \ x + 1 \]
\[ E \ y = x + 1 \]

\[ * = \text{write } y \]

\[ B \ x + 1 \]
\[ E \ x + 1 \]
\[ V_{x+1} := x + 1 \]

\[ E \ x + 1 \]
\[ V_{x+1} := \text{any} \]
Action automata

- $y = x + 1$;

$y = x + 1$

$\mathcal{B} y = x + 1$

$\mathcal{B} x + 1$

$\mathcal{E} x + 1$

$\mathcal{E} y = x + 1$

$y := V_{x+1}$

$\mathcal{E} y = x + 1$

$y := \text{any}$

$x + 1$

$\mathcal{B} x + 1$

$\mathcal{E} x + 1$

$\mathcal{E} x + 1$

$\mathcal{E} y = x + 1$

$\mathcal{E} y = x + 1$

$\mathcal{E} y = x + 1$

$\mathcal{E} y = x + 1$

$\mathcal{E} y = x + 1$

$\mathcal{E} y = x + 1$

$\mathcal{E} y = x + 1$

$\mathcal{E} y = x + 1$

$\mathcal{E} y = x + 1$

$\mathcal{E} y = x + 1$

$\mathcal{E} y = x + 1$

$\mathcal{E} y = x + 1$

$\mathcal{E} y = x + 1$

$\mathcal{E} y = x + 1$

$\mathcal{E} y = x + 1$

$\mathcal{E} y = x + 1$

$\mathcal{E} y = x + 1$

$\mathcal{E} y = x + 1$

$\mathcal{E} y = x + 1$

$\mathcal{E} y = x + 1$

$\mathcal{E} y = x + 1$

$\mathcal{E} y = x + 1$

$\mathcal{E} y = x + 1$

$\mathcal{E} y = x + 1$

$\mathcal{E} y = x + 1$

$\mathcal{E} y = x + 1$

$\mathcal{E} y = x + 1$

$\mathcal{E} y = x + 1$

$\mathcal{E} y = x + 1$

$\mathcal{E} y = x + 1$

$\mathcal{E} y = x + 1$

$\mathcal{E} y = x + 1$

$\mathcal{E} y = x + 1$

$\mathcal{E} y = x + 1$

$\mathcal{E} y = x + 1$

$\mathcal{E} y = x + 1$

$\mathcal{E} y = x + 1$

$\mathcal{E} y = x + 1$

$\mathcal{E} y = x + 1$

$\mathcal{E} y = x + 1$

$\mathcal{E} y = x + 1$

$\mathcal{E} y = x + 1$

$\mathcal{E} y = x + 1$

$\mathcal{E} y = x + 1$

$\mathcal{E} y = x + 1$

$\mathcal{E} y = x + 1$

$\mathcal{E} y = x + 1$

$\mathcal{E} y = x + 1$

$\mathcal{E} y = x + 1$

$\mathcal{E} y = x + 1$

$\mathcal{E} y = x + 1$

$\mathcal{E} y = x + 1$

$\mathcal{E} y = x + 1$

$\mathcal{E} y = x + 1$

$\mathcal{E} y = x + 1$

$\mathcal{E} y = x + 1$

$\mathcal{E} y = x + 1$

$\mathcal{E} y = x + 1$

$\mathcal{E} y = x + 1$

$\mathcal{E} y = x + 1$

$\mathcal{E} y = x + 1$

$\mathcal{E} y = x + 1$

$\mathcal{E} y = x + 1$

$\mathcal{E} y = x + 1$

$\mathcal{E} y = x + 1$

$\mathcal{E} y = x + 1$

$\mathcal{E} y = x + 1$

$\mathcal{E} y = x + 1$

$\mathcal{E} y = x + 1$

$\mathcal{E} y = x + 1$
Action automata

- \( y = x + 1; \)

\[
\begin{align*}
y &= x + 1 \\
x + 1 &\xrightarrow{B} \quad y = x + 1 \\
\text{write } y &\xrightarrow{*} \\
x + 1 &\xrightarrow{E} \\
\end{align*}
\]

- \( x + 1 \)

\[
\begin{align*}
x + 1 &\xrightarrow{B} \\
\text{write } x &\xrightarrow{C} \\
V_{x+1} &\xrightarrow{E} : = x + 1 \\
\end{align*}
\]

- \( y \)

\[
\begin{align*}
y &\xrightarrow{B} \\
y &\xrightarrow{E} : = \text{any} \\
\end{align*}
\]

- \( x \)

\[
\begin{align*}
x &\xrightarrow{B} \\
\text{write } x &\xrightarrow{C} \\
V_{x+1} &\xrightarrow{E} : = \text{any} \\
\end{align*}
\]

\[
\begin{align*}
V_{x+1} &= 0 \\
y &= 0 \\
x &= 0 \\
B \ y &= x + 1 \quad N \\
B \ x &= 1 \quad N \quad N
\end{align*}
\]
Action automata

- y = x + 1;

\[
y = x + 1
\]

\[
x + 1
\]

\[\text{write } x
\]

\[V_{x+1} := x + 1
\]

\[V_{x+1} := \text{any}
\]

\[\star = \text{write } y
\]
Action automata

- \( y = x + 1; \)

\[
\begin{align*}
V_{x+1} & \quad 0 \quad 5 \quad 5 \\
y & \quad 0 \quad 0 \quad 5 \\
x & \quad 0 \quad 0 \quad 0 \\
B \ y=x+1 & \quad N \quad B \ x+1 & \quad N \quad N \quad E \ x+1 & \quad E \ y=x+1
\end{align*}
\]

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

```
await {
  (guard_1; test_1; set_1){ statements_1; }
  ...
  (guard_n; test_k; set_n){ statements_n; }
}
```

- `guard_i`: expression with no side effect
- `test_i`, `set_i`: `{ (object, port interface) }

- the only operational construct to coordinate one process with others.
- it blocks until exactly one `statements_i` starts; a choice is non-deterministic.
- `statements_i` can start only if `guard_i` is true and no action in `test_i` is taken.
- `statements_i` is executed under the guarantee that no action in `set_i` is taken.
- if `statements_i` ends, await needs to end.

actions defined by `(object M, port interface I)` in `test_i`:
- calls of functions of `I` implemented in `M`,
- `statement_x` in `some await` with `(M, I)` in `set_x`
Semantics of await

```plaintext
await {
    (guard_i; test_i; set_i) stmt_i;
    (guard_j; test_j; set_j) stmt_j;
}
```

B await...

(\text{true guard}_i \land \neg\text{active test}_i) / \neg\text{start set}_i

\text{B stmt}_i \land \neg\text{start set}_i \rightarrow \text{E stmt}_i

\neg\text{start set}_i \rightarrow \text{C}

\text{B await}... \rightarrow \text{C}

(\text{true guard}_j \land \neg\text{active test}_j) / \neg\text{start set}_j

\text{E stmt}_j \rightarrow \text{C}

\text{E await}... \rightarrow \text{C}

\text{E stmt}_j \land \neg\text{start set}_j \rightarrow \text{C}
Semantics summary

- Processes run sequential code concurrently, each at its own arbitrary pace.
- Read-Write and Write-Write hazards may cause unpredictable results
  - atomicity has to be explicitly specified.
- Progress may block at synchronization points
  - awaits
  - function calls and labels to which awaits or constraints refer.
- The legal behavior of a netlist is given by a set of sequences of event vectors.
  - multiple sequences reflect the non-determinism of the semantics: concurrency, synchronization (awaits and constraints)
**Architecture components**

An architecture component specifies services, i.e.

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

---

**Nexperia™ Hardware Architecture**

- **MIPS™**
  - MIPS CPU
  - D$ FRxxx
  - Device I/P Block
  - Device I/P Block
  - Device I/P Block
  - DVP System Silicon

- **TriMedia™**
  - TriMedia CPU
  - TM-xxxx
  - Device I/P Block
  - Device I/P Block
  - Device I/P Block

**Library of Device Blocks**
- Image coprocessors
- DSPs
- UART
- 1394
- USB

**GENERAL PURPOSE RISC PROCESSOR**
- 50 to 300+ MHz
- 32-bit or 64-bit

**VLIW MEDIA PROCESSOR:**
- 100 to 300+ MHz
- 32-bit or 64-bit

**Nexperia System Buses**
- 32-128 bit

---

*Let's make things better*
**Meta-model: architecture components**

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

- **what it can do:**
  - interfaces, methods, coordination (awaits, constraints), netlists
- **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)));
    }
}
```
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);
        [[
            constraint{
                forall i: GTime.A(B(thisthread, this.memRead), i) ==
                GTime.A(B(thisthread, this.busRead), i)
                + BUSCLOCKCYCLE;
            }
        ]]
    }
    ...
}
Meta-model: architecture components

- This modeling mechanism is generic, independent of services and cost specified.
- Which levels of abstraction, what kind of quantities, what kind of cost constraints should be used to capture architecture components?
  - depends on applications: on-going research

---

**Transaction:**
- Services:
  - fuzzy instruction set for SW, execute() for HW
  - bounded FIFO (point-to-point)
- Quantities:
  - #reads, #writes, token size, context switches

---

**Virtual BUS:**
- Services:
  - data decomposition/composition
  - address (internal v.s. external)
- Quantities: same as above, different weights

---

**Physical:**
- Services: full characterization
- Quantities: time
**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

```
forall i: GTime.A(E(P1, foo), i) = GTime.A(B(P1, foo), i) + 15.
```

A process `P2` executes a method `bar()`, with

```
forall i: GTime.A(E(P2, bar), i) = GTime.A(B(P2, 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 netlist 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

```java
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 netlist 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**
   
   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);
}

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

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))));
        ]]
    }
}

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

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

What do these processes represent?
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();
            }
        } }}
}

medium AbsCpu implements AbsCpuService {
    void cpuRead(){ ... }
    void cpuWrite(){ ... }
    void exec(){ ... }
}
**Meta-model: architecture netlist**

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

What do these processes represent?
: mapping processes
Function process

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

Mapping process

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) <-> B(MapP, readCpu); E(P, X.read) <-> E(MapP, readCpu);
B(P, f) <-> B(MapP, mapf); E(P, f) <-> E(MapP, mapf);
...

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

This is a compilation problem!

• 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 netlist**

MyMapNetlist

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

Remember: a meta-model netlist 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.
interface MyService extends Port {
  int myService(int d);
}

medium AbsM implements MyService{
  int myService(int d) {
      ...
  }
}

refine(AbsM, MyMapNetlist2)
refine(AbsM, MyMapNetlist1)

MyMapNetlist2
- B(P1, M.write) <-> B(mP1, mP1.writeCpu);
- B(P1, P1.f) <-> B(mP1, mP1.mapf);
- E(P1, P1.f) <-> E(mP1, mP1.mapf);
- B(P2, M.read) <-> B(P2, mP2.readCpu);
- E(P2, P2.f) <-> E(mP2, mP2.mapf);

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

refine(AbsM, MyMapNetlist2)
refine(AbsM, MyMapNetlist1)
Meta-model: platforms

A set of mapping netlists, together with constraints on event relations to a given interface implementation, constitutes a platform of the interface.

```java
interface MyService extends Port {
    int myService(int d);
}

medium AbsM implements MyService{
    int myService(int d) { ... }
}
```

refine(AbsM, MyMapNetlist)

MyArchNetlist
MyFncNetlist

refine(AbsM, MyMapNetlist1)

MyArchNetlist
MyFncNetlist

A set of mapping netlists, together with constraints on event relations to a given interface implementation, constitutes a platform of the interface.
Meta-model: recursive paradigm of platforms

\[
\begin{align*}
B(Q2, S.cdx) & \leftrightarrow B(Q2, mQ2.excCpu); & E(Q2, M.cdx) & \leftrightarrow E(mQ2, mQ2.excCpu); \\
B(Q2, Q2.f) & \leftrightarrow B(mQ2, mQ2.mapf); & E(Q2, P2.f) & \leftrightarrow E(mQ2, mQ2.mapf); \\
B(P1, M.write) & \leftrightarrow B(mP1, mP1.writeCpu); & B(P1, P1.f) & \leftrightarrow B(mP1, mP1.mapf); \\
B(P2, M.read) & \leftrightarrow B(P2, mP2.readCpu); & E(P2, P2.f) & \leftrightarrow E(mP2, mP2.mapf); \\
\end{align*}
\]
Meta-model: summary

• Concurrent specification with a formal execution semantics
• Feasible executions of a netlist: 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 netlists,
  - a refinement through event coordination provides a platform.
Metropolis Framework

Methodologies
- Multi-media, wireless communication, mechanical controls, processors

Meta-model Library
- Models of computation

Infrastructure
- Metropolis meta-model
  - language
  - modeling mechanisms
- Meta-model compiler

Meta-model Library
- Architecture platforms

Tools
- Simulator
- QSS
- PIG
- STARS
- SPIN
...
Metropolis design environment

- Load designs
- Browse designs
- Relate designs (refine, map etc)
- Invoke tools
- Analyze results
New to Metropolis? Type 'help' for information on available commands.

```bash
metropolis> load pkg -semantics DoubleStream
metropolis> elaborate DoubleStream.MyNet
    Choosing a temporal directory...
    Compiling elaboration code...
metropolis> refine IntM0 DoubleStream.WordMNet
metropolis> simulate systemc -net MyNet0
```
Summary

- Metropolis meta-model:
  - concurrent specification for functions, architecture, and mappings
  - libraries of models of computation, libraries of platforms

- Metropolis design environment:
  - meta-model compiler to provide an API to browse designs,
  - backend tools to analyze designs and produce appropriate models,
  - easy to incorporate simulation, verification, and synthesis tools
Install Metropolis Design Environment

- You need
  - An account on ic.eecs.berkeley.edu
  - To be in hwsw group
  - Linux (windows?)
  - CVS [http://www.cvshome.org](http://www.cvshome.org)
  - SSH \winsww
  - Java JDK 1.2 or 1.3 \winsww
  - SystemC for simulator [http://www.systemc.org](http://www.systemc.org)
  - JACL for metroshell

- Download metro
  CVSROOT=guyang@ic.eecs.berkeley.edu:/projects/hwsw/hwsw/common/src
  CVS_RSH=ssh
cvs checkout metro
cvs update metro

- Install metro
  Follow metro/src/metropolis/metamodel/README
Examples

- 2-producer-1-consumer

- YAPI library