Transformation of Executable Platform Independent Models

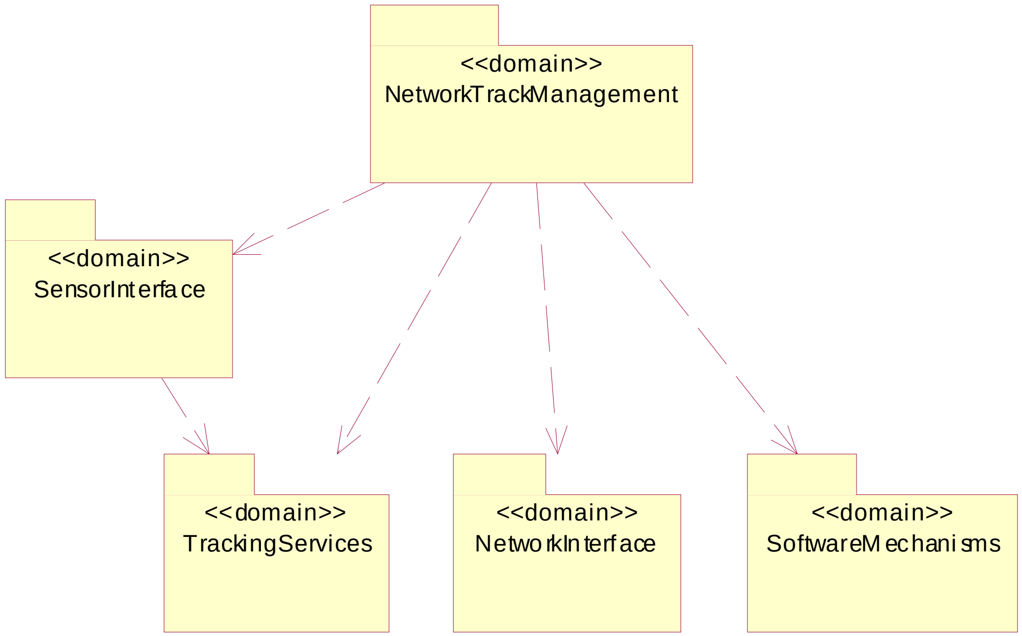
Bruce Levkoff
Pathfinder Solutions

Agenda

- Executable Models
- Transformation Environment
- Transformation Rule Language
  - Model Navigation
  - Flow of Control
    - Expanding other templates
    - Branches and Loops
  - Conversion to text
- Implementation patterns/Mapping Rules
  - Attribute, Set/Get
  - State Machine

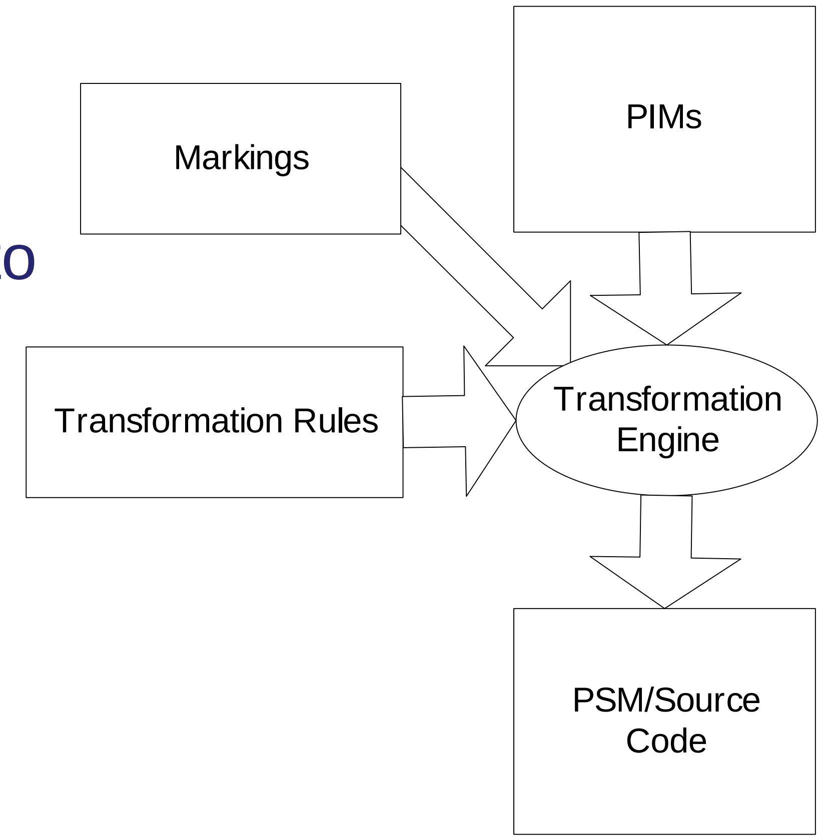
Executable Models

- Logical Separation
- Well defined boundaries
- Modeled domains consist of:
  - API
  - Classes
  - Statecharts
  - Action Language



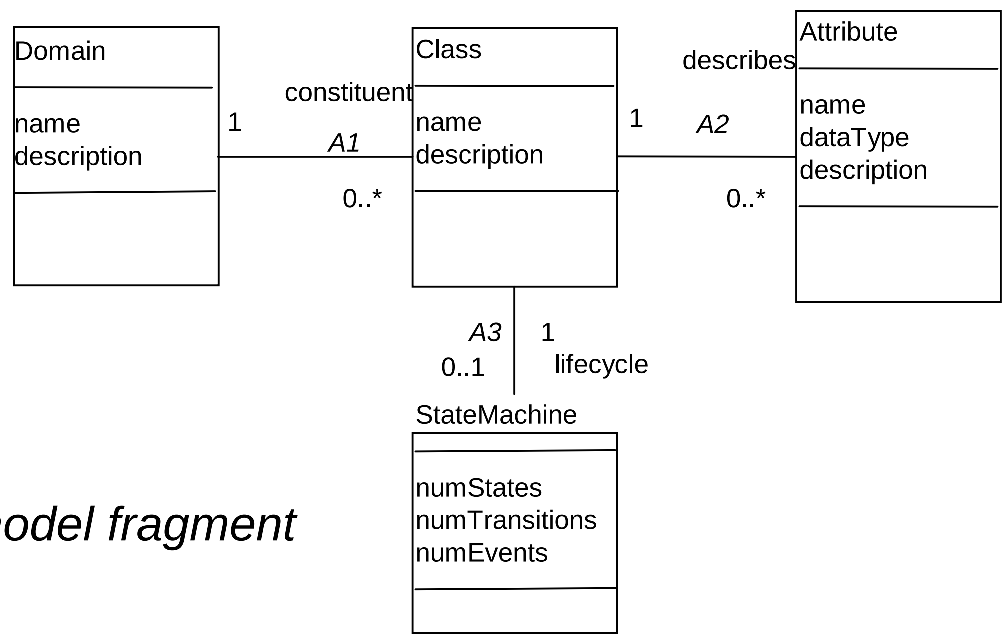
MDA Transformation

- Model components independent of implementation technologies
- Transformations map model to implementation
- Leverage models into integration testing



UML™ metamodel

- **The UML™ metamodel:**
  - is a Class Model of the analysis elements
  - used to organize analysis information for input to transformation engine



metamodel fragment

PIM elements

- **Platform Independent Models are made up of the following work products:**
  - Domain Models
  - Class Models
  - Scenario Models
  - State Models
  - Action Models
- **Design and Construction define how we map these analysis elements to implementation**

MDA execution

Pathfinder Solutions

- **MDA defines a very specific execution context:**
  - provides a dynamic view of the notation
  - has well defined run-time semantics of the models
  - abstract enough to allow multiple designs and implementations
- **a valid MDA implementation must support all of the rules of the execution engine**

MDA execution

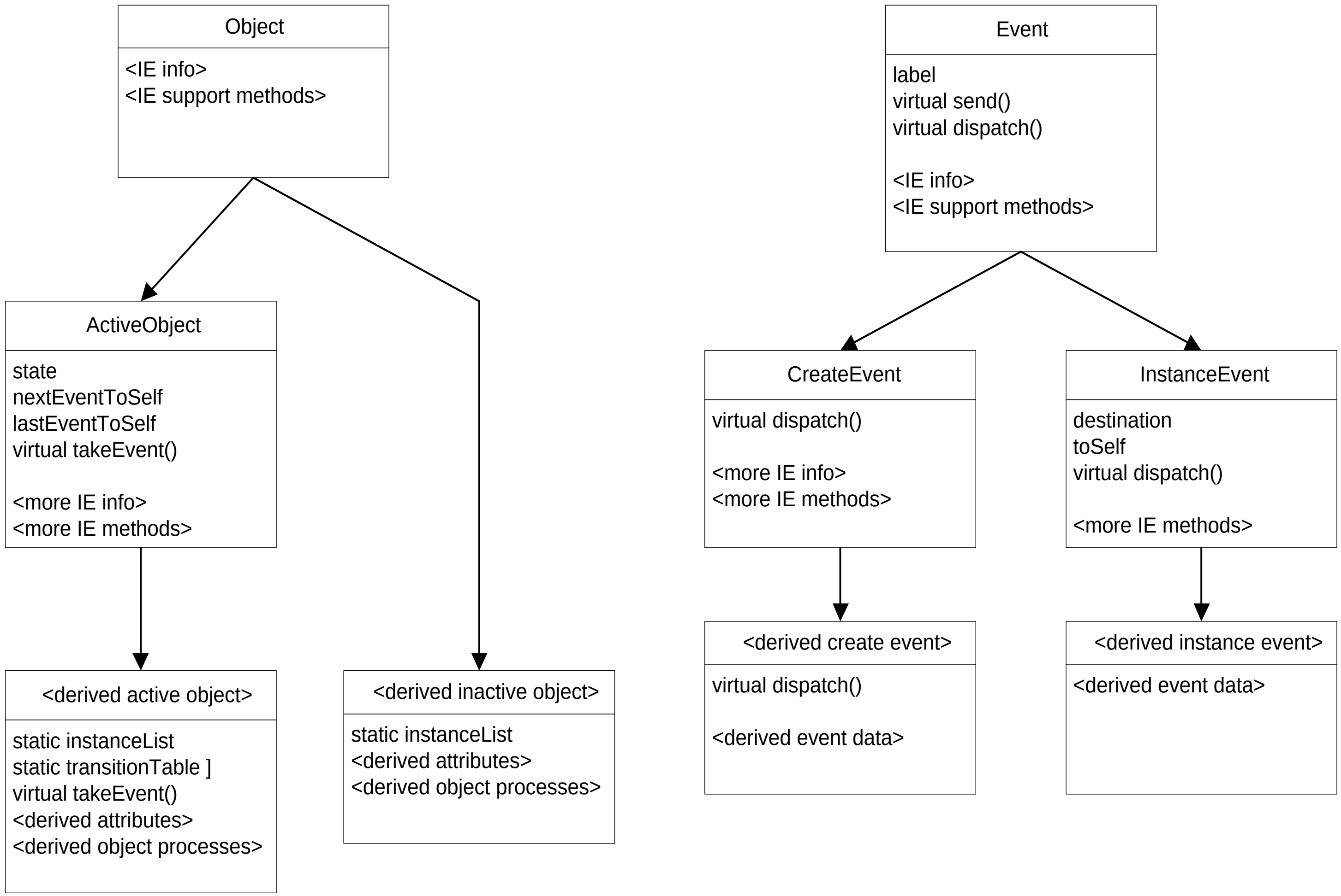
- **rules - a valid MDA implementation must:**
  - maintain tracking structures of all class instances where needed
  - maintain tracking structures for association populations where needed
  - support state machine semantics
  - enforce Event ordering
  - preserve Action atomicity
  - provide a mapping for all analysis elements, including:
    - Domain, Domain Service
    - Class, Attribute, Association, Inheritance, Associative Class, Class Service
    - State, Event, Transition, Superstate, Substate
    - all Action Modeling elements

SW Mechanisms

- **the Software mechanisms are realized code elements that support the execution engine**
- **common base mechanisms:**
  - Task: manage event queue and control overall OOA execution
  - Object/ActiveObject: parent classes for derived object classes
  - Event: parent class for derived event classes
  - String, List, Array, ErrorLog, other base mechanisms to establish execution environment

SW Mechanisms

example mechanisms:



Transformation

- **Transformation is the mapping of Analysis elements through code templates to implementation**
  – Implementation code patterns and transformation rules are captured in “code templates”.
  – Platform-specific marking sets can be applied as input to transformation rules.

Transformation Language

- Model Navigation
  - Extract the model into associations with metamodel elements.
  - Navigate the associations using analysis element fields
    - system.domains
    - domain.classes
  - Emit the fields into the target document as specified by the templates.

Transformation Language

- Flow of Control
  - IF/ELSE
  - FOREACH
  - EXPAND

```
[FOREACH svc IN obj.services]
        [IF (svc.polymorphism != POLYMORPHIC_INTERFACE)]
/* [svc.name]: [svc.description] */
        [EXPAND act_svc_profile (svc, FALSE, FALSE)];
        [ENDIF /* !POLYMORPHIC_INTERFACE */]
        [IF ((obj.subTypes > 0) && (svc.polymorphism != NOT_POLYMORPHIC) && (svc.instanceBased))]
/* Dispatch virtual invocations to subtype implementations of [svc.name] */
[EXPAND act_svc_profile (svc, TRUE, FALSE)];
        [ENDIF /* !NOT_POLYMORPHIC */]
[ENDFOREACH /* svc */ ]
```

Transformation

Pathfinder Solutions

- **a transformation engine is used to:**
  - extract the analysis model semantics
  - load model markings
  - load and parse the code templates (including transformation rules)
  - produce the target documents (code, reports, documentation, etc) based on the models, markings, and templates.

Transformation

Pathfinder Solutions

- **code templates tie the models, structural design, and mechanisms together**
- **some examples of Analysis-level semantic element available through templates are:**
  - analysis class/C++ class
  - attribute/member variable
  - state action/member function
  - Action Language element/function call
  - event generation/event enqueing
  - event processing

template

```
  . . .
public:
    // Attributes:
[FOREACH attribute in object.attributes]
    /* [attribute.description] */
    [EXPAND x_a_typ (attribute)]  [attribute.langId];

[ENDFOREACH /*attribute */ ]
  . . .
```

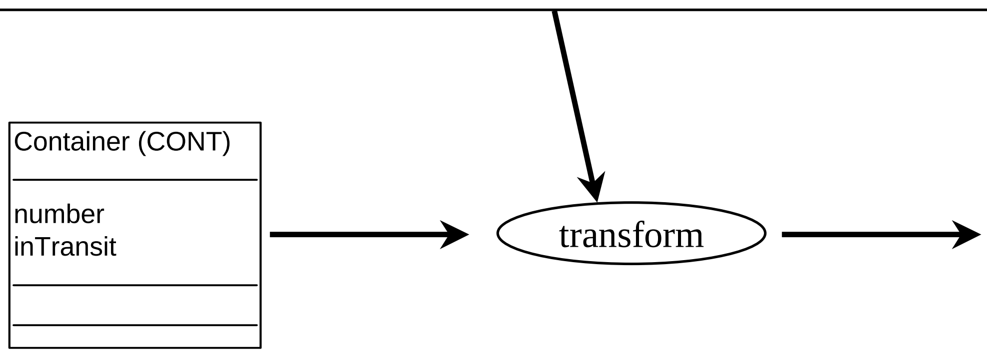
Transformation

• Analysis Class -> C++ class

template

```
//============================================================
class [class_name] : public \
    [FOREACH parent IN object.superTypes SEPARATOR ", public "] \
    [object.domain.prefix]_[parent.prefix][ENDFOREACH]
{
  /* [object.description]  */

public:
  // This is the list of all instances of objects of this type
  static PfdBaseList instanceList;
  . . .
```



generated C++ code

```
//=========================================
class FP_CONT : public   FP_PA
{
  /* A bowl, disk, pan, baking sheet,
    or other mixing or cooking vessel.  */

public:
  // This is the list of all instances of
  //objects of this type
  static PfdBaseList instanceList;
. . .
```

Transformation

- **Attributes -> Class Member Variables**
  – each Attribute maps to a public data member of it's C++ class

template

```
  . . .
public:
    // Attributes:
[FOREACH attribute in object.attributes]
    /* [attribute.description] */
    [attributedataType]  [attribute.name];

[ENDFOREACH /*attribute */ ]
  . . .
```

transform

Container (CONT)

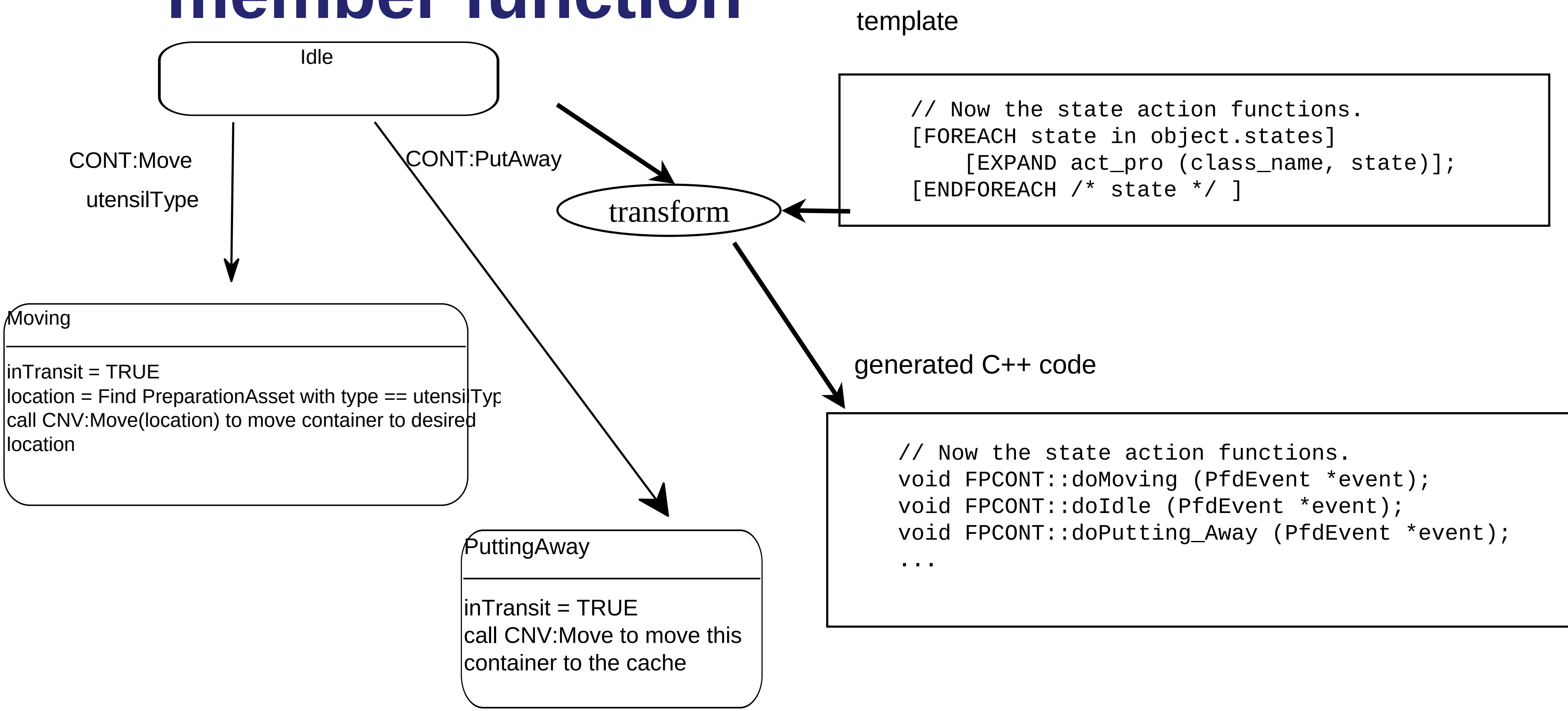
number
inTransit

generated C++ code

```
public:
    // Attributes:
    /* External identification number. */
    Integer  number;
    /* Flag indicating if the container is moving towards
       its location, or if it has reached it. */
    Boolean  inTransit;
```

Transformation

- **Analysis State action -> Class member function**



Transformation

Pathfinder Solutions

- **Analysis Event generation**
  - the event generator creates an event and queues it in the event queue

StartCooking

template

```
void [event_class_name]::generate([class_name] *dest\
[FOREACH evdi IN event.parameters]
, [EXPAND gen_type (evdi.dataType)] [evdi.langId]\
[ENDFOREACH /* params */]
)
{
    // Create a new instance of the event
    [event_class_name] *ev = new [event_class_name] (dest);
    [ENDIF /* create */]
    // Now fill in the parameters
    [FOREACH evdi IN event.parameters]
    ev->[evdi.langId] = [evdi.langId];
    [ENDFOREACH /* params */]

    // Send the event
    ev->send();
}
```

Analysis action language

```
GENERATE AR:StartCooking() TO (new_recipe);
```

transform

generated C++ code

```
void FP_AR::StartCooking::generate(FP_AR *dest)
{
    // Create a new instance of the event
    FP_AR::StartCooking *ev = new FP_AR::StartCooking (dest);
    // Now fill in the parameters

    // Send the event
    ev->send();
}
```

Transformation

- **design markings can be attached to analysis elements**
- **they are design-level input to the transformation step**
  - helps transformation engine choose which templates to apply
  - can be used to map classes in one domain to another
  - example: implementing a 1:M association:
    - options: array or linked list
    - example property “MaxPopulation” values: “<unbounded>”, “SYS_MAX_SENSOR_COUNT”

Conclusion

- Transformation through templates gives complete control over generated code.
- Platform independence of models is maintained.
- Implementation can be adjusted for various deployments.