

Multiple Dispatch for AmI

1st Workshop on
Object Technology for Ambient Intelligence

26th July 2005

Sebastián González
INGI / UCL / Belgium

Position Statement

Taking a step back from standard OO semantics can help us finding innovative programming abstractions for Aml

Multiple dispatch seems a particularly rich mechanism to explore

Presentation Plan

(20 minutes)

- The “Prototypes + Multiple Dispatch” model
- General and Aml-specific advantages
- Technical challenges

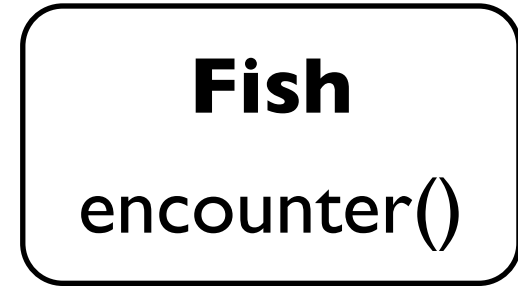
The PMD Model

- Uniform semantics
 - Everything is an object
 - All computation happens by message passing
- State as encapsulated as in Self †
 - An object consists of named slots ‡
 - Every slot references some object
 - Manipulation through accessor and mutator methods
- Polymorphic message passing
- Sharing through multiple delegation slots, traversed depth-first

Fish>>**encounter: animal**

[animal instanceof: Shark
ifTrue: [self swimAway: animal.
animal chase: self]]

1st person story



“The fish knows how”

animal@Fish encounter: **other@Shark**

[animal swimAway: other.
other chase: animal.]

3rd person story

- A “theatre play” script
- Roles played by actors
- Augmented notion of collaboration



“They know how”

General Advantages of MD

- **More declarative**

`_@True /\ _@True [True].`

`_@(Boolean traits) /\ _@(Boolean traits) [False].`

- **More concise**

enhanced code reuse

- **Easier to understand**

simplified control flow

- **More extensible**

methods can be added without changing existing objects

- **Subsumes procedures and single dispatching**

- **No Receiver-Centricity**

`a@Integer + b@Integer`

`a primitiveAdd: b`

Aml-specific Advantages

- Non-intrusive specialisation of behaviour

```
dev1@BTDevice pairWith: dev2@BTDevice  
  [ "generic pairing algorithm here" ]
```

```
dev1@BTSmartPhone pairWith: dev2@BTSmartPhone  
  [ "specialised pairing algorithm here" ]
```

- Multi-methods are fine-grained units of knowledge
- Subjective dispatch
(subject-oriented programming)

Supporting Multiple Dispatch in Mobile Systems (Challenges)

Message handling

send → receive → schedule → lookup → execute

Message sending, receiving and scheduling

- Messages have no particular sender and receiver (there is not a distinguished “self”)
- **Simple solution:** have per-VM message in/out queues.

Supporting Multiple Dispatch in Mobile Systems (Challenges)

Message lookup

A with: **B** and: **C**
L **M** **N**
(local)

Where to search for #with:and: ?
In L, M, N or elsewhere?

Supporting Multiple Dispatch in Mobile Systems

Message lookup

A with: B and: C

L M N

Suggestion:

emerging behaviour

- 1 Start by L (use local knowledge)
- 2 Then M, N (knowledge of acquaintances)
- 3 Then elsewhere (distributed knowledge)

Supporting Multiple Dispatch in Mobile Systems

Message lookup

A with: B and: C

L **M** **N**

What if M has a more specific implementation than L? (i.e. external knowledge is more refined)

Difficult issue... this **happens** IRL as well!

Possibility: use non-functional constraints

MD in Mobile Systems

Method execution

A with **B** and: C
L M N

Once an implementation has been chosen, where should it be executed?

- On the hosting device (where it was found)
- On the device which sent the message
- Receiver with most unary messages
- Programmer hints (semantic hints)
- Choice based on profiling / machine learning / ...

Key Points

(conclusion)

MD for AmI

brings opportunities...

- Distributed multi-methods enable **synergism** between devices
(emerging behaviour)
- Distributed multi-methods enable **fine-grained sharing of knowledge** between devices
- Distributed multi-methods enable **non-intrusive customisation** of object behaviour

Key Points

(conclusion)

MD for AmI

brings opportunities...

- Distributed multi-methods *facilitate* **dynamic adaptability** such as load balancing and resource-aware computing