# From Plain Prolog to Logtalk Objects: Effective Code Encapsulation and Reuse

Paulo Moura
Dep. of Computer Science, Univ. of Beira Interior, Portugal
Center for Research in Advanced Computing Systems
INESC Porto, Portugal

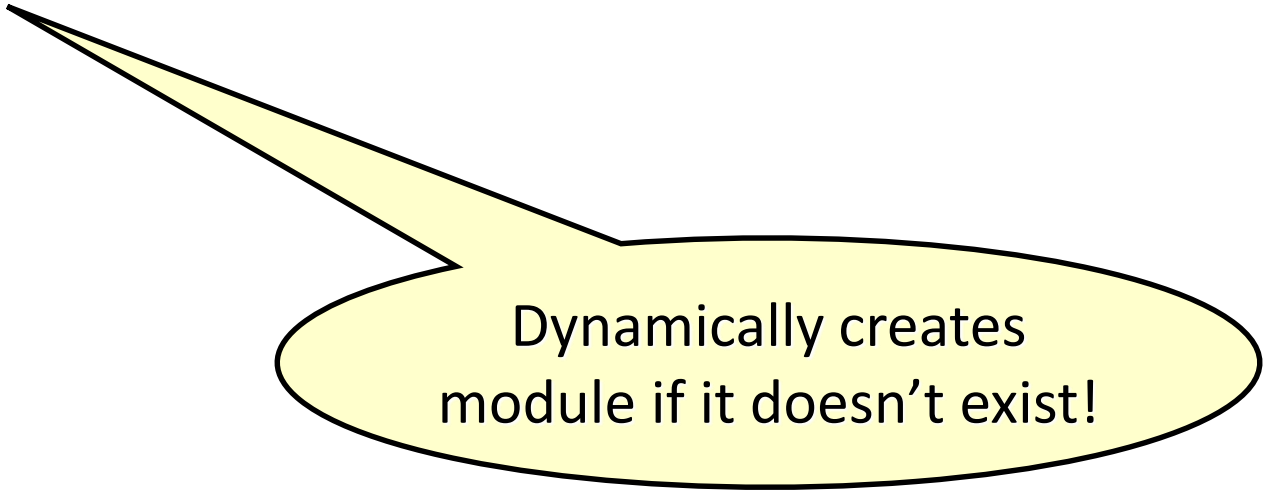
# Logtalk – Objects in Prolog

Modules as Objects???

# Objects are dynamically created!

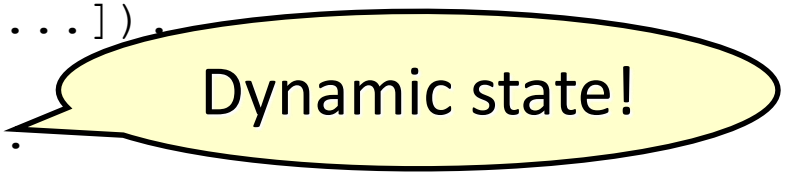
- Module = Object?
  - Modules are typically static, simply loaded from source files...
  - But access to a non-existing module implicitly creates it at run-time:

```
?- p(X), X:assertz(something).
yes
```



# Objects have identifiers and dynamic state!

- Module = Object?


```
:- module(broadcast, [...]).

:- dynamic(listener/4).
...

assert_listener(Templ, Listener, Module, TheGoal) :-
      asserta(listener(Templ, Listener, Module, TheGoal)).
retract_listener(Templ, Listener, Module, TheGoal) :-
      retractall(listener(Templ, Listener, Module, TheGoal)).
```

# Objects can inherit

- Module import = Inheriatance?

```
:- module(aggregate, [...]).
:- use_module(library(ordsets)).
:- use_module(library(pairs)).
:- use_module(library(error)).
:- use_module(library(lists)).
:- use_module(library(apply))....
```

# Why not stick to modules?!?

Prolog modules fail to:

- enforce encapsulation
  - you can call any module predicate using explicit qualification, module:p(X)
- implement namespaces
- provide a standard mechanism for import conflicts
- provide a clean separation between loading and importing
- support separation of interface from implementation
- provide the same semantics for implicitly and explicitly qualified calls to meta-predicates

# Why not improve module systems?!?

- No one wants to break backward compatibility
- Good enough mentality (few users working on large apps)
- Instant holy wars when discussing modules

# Logtalk

# Design Goals

- **Code encapsulation**
  - objects
  - protocols (= interfaces)
  - categories (fine-grained units of code reuse)
- **Code reuse**
  - message sending
  - inheritance
  - composition
- **Multi-paradigm language**
  - predicates
  - objects
  - events
  - threads
- **Multi-paradigm objects**
  - prototypes
  - classes
- **Compatibility**
  - ISO Prolog Core standard
  - Widely-used Prolog compilers

# Categories of object-oriented languages

## Class-based

- Immutable object structure
  - Only attribute values can change
  - Set of attributes is fixed
  - Set of methods is fixed
- Classes as object templates
  - Object creation by instantiation
- Limited inheritance
  - Relation between classes
  - Static!!!

## Prototype-based

- Dynamic object structure
  - Everything can change
  - … at run-time!!!
- No classes
  - Object creation by cloning
- More powerful inheritance
  - Relation between objects!!!
  - Dynamic!!!

# Encapsulated Prototype Objects

## Defining prototype objects

```
:- object(list).

  :- public(append/3).
  append([], L, L).
  append([H| T], L, [H| T2]) :-
      append(T, L, T2).

  :- public(member/2).
  member(H, [H| _]).
  member(H, [_| T]) :-
      member(H, T).

  :- private(secret/2).
  ...

:- end_object.
```

## Sending messages

```
?- list::append(L1, L2, [1, 2]).
L1 = [], L2 = [1, 2];
L1 = [1], L2 = [2];
L3 = [1, 2], L2 = []
yes

?- list::member(X, [a, b, c]).
X = a;
X = b;
X = c
Yes

?- list::secret(A,B).
Error: Message secret/2 undefined
```

Encapsulation enforced! ☺

# Protocols (= Intrfaces)

## Defining protocols

```
:- protocol(listp).

  :- public(append/3).
  :- public(member/2).
  ...

:- end_protocol.
```

## Implementing protocols

```
:- object(list,
      implements(listp)).

  append([], L, L).
  append([H| T], L, [H| T2])
  :-
      append(T, L, T2).

  member(H, [H| _]).
  member(H, [_| T]) :-
      member(H, T).

:- end_object.
```

# Objec-level Inheritance

## A self-contained "prototype" object

```
:- object(state_space).

  :- public(initial_state/1).
  :- public(next_state/2).
  :- public(goal_state/1).
  ...

:- end_object.
```

## An object defined by extension

```
:- object( heuristic_state_space,
     extends(state_space)).

  :- public(heuristic/2).
  ...


:- end_object.
```

# Classes

## A class

```
:- object(person,
      instantiates(class),
      specializes(object)).

  :- public(name/1).
  :- public(age/1).
  ...

:- end_object.
```

## An instance

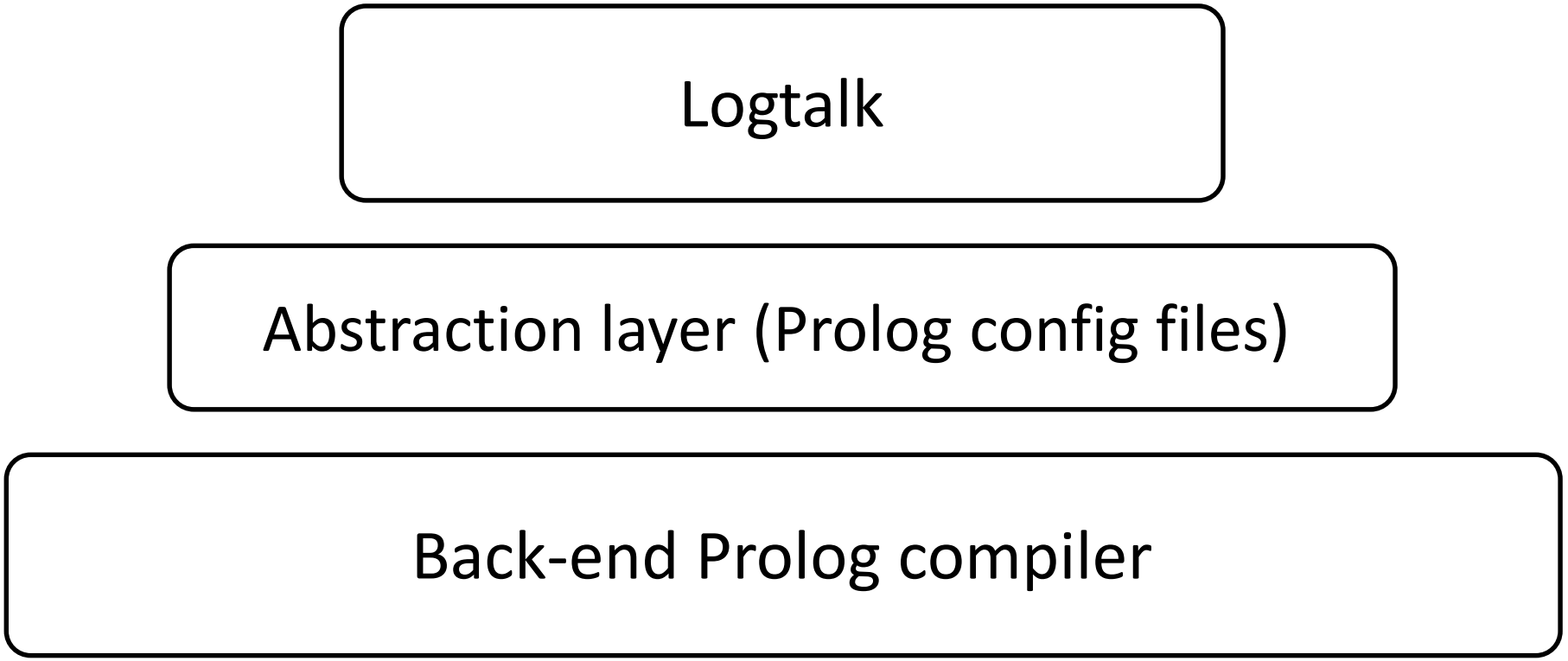
```
:- object( paulo,
      instantiates(person)).


  name('Paulo Moura').
  age(41).
  ...

:- end_object.
```

# Logtalk as a portable Prolog application

# Logtalk Architecture



# Supported Prolog compilers

## Runs out-of-the box using

- B-Prolog
- CxProlog
- ECLiPSe
- GNU Prolog
- Qu Prolog
- SICStus Prolog
- SWI-Prolog
- XSB
- YAP

## Supported older versions

- Ciao
- IF/Prolog
- JIProlog
- K-Prolog
- Open Prolog
- ALS Prolog
- Amzi! Prolog
- BinProlog
- LPA MacProlog, LPA WinProlog
- Prolog II+
- Quintus Prolog

# Logtalk in Eclipse

Full Eclipse integration on top the PDT and SWI-Prolog

- Editor
- Outline
- Search
- Console
- Graphical debugger

See http://sewiki.iai.uni-bonn.de/research/pdt/docs/v2.1/start

- Most functionalities of the PDT work for Logtalk too