

---

# Component-Oriented Programming Languages: Messages vs. Methods, Modules vs. Types

**Peter H. Fröhlich**  
phf@acm.org

Department of Information and Computer Science  
University of California, Irvine  
Irvine, CA 92697-3425, USA

## **Acknowledgements**

Joint research with Dr. Michael Franz.

---

# Overview

- Trend towards component-oriented programming languages.
- Essential concepts and some terminology.
- Interface conflicts in object-oriented languages.
- Separating messages from methods and modules from types.
- The programming language Lagoona.

---

# Component-Oriented Programming Languages

- Programming languages are influenced by
  - software development paradigms (component-oriented)
  - design patterns (COM, CORBA, JavaBeans)
- Formalizing paradigms and patterns
  - reduces the “semantic gap” between problem and solution
  - helps to think and communicate at the right level
  - enables compilers to enforce constraints and optimize
- Recent research and development on
  - features (Compound Types for Java, Units for Scheme)
  - languages (Component Pascal, Components for SML)

---

# Essential Concepts and Some Terminology

- Polymorphism
  - dynamically substitute component instances for each other
- Modularity
  - completely isolate components except for explicit dependencies
- Independent Extensibility
  - ensure that independently developed extensions can be composed
- Terminology
  - Message (abstract operation), Method (concrete operation)
  - Interface (set of messages), Implementation (set of methods)
  - Module (static structure), Type (dynamic structure)

---

# Interface Conflicts in Object-Oriented Languages

## Original Interface

```
void push (Object o)
void pop ()
Object top ()
boolean empty ()
```

## Compatible Interface

```
void push (Object o)
void pop ()
Object top ()
int size ()
```

## Syntactic Conflict

?

## Semantic Conflict

```
void push (Object o)
Object pop ()
boolean empty ()
```

```
void push (Object o)
void pop ()
Object top ()
boolean empty ()
int size ()
```

---

# Messages $\neq$ Methods, Modules $\neq$ Types

- Object-oriented programming languages:

Conflicts	$\text{sig}(a) \neq \text{sig}(b)$	$\text{sig}(a) = \text{sig}(b)$
– $\text{id}(a) \neq \text{id}(b)$	No conflict	No conflict
– $\text{id}(a) = \text{id}(b)$	Syntactic conflict	Semantic conflict

- messages have unique identities within a type only
- combining types can lead to subtle conflicts

- Component-oriented programming languages:

Design Space	<b>Message <math>\in</math> Type</b>	<b>Message <math>\in</math> Module</b>
– <b>Method <math>\in</math> Type</b>	Object-Oriented	<i>Component-Oriented</i>
– <b>Method <math>\in</math> Module</b>	Useless?	Modular

- messages must be unique within a module to avoid conflicts
- methods must be unique within a type for polymorphism

---

# The Programming Language Lagoon

- Modules (similar to Oberon, unique by convention)
- Stand-alone messages (declared at module level)
- Object types (based on records, no type extension)
- Methods implement messages in object types (no inheritance)
- Message set types (references to objects, set-like operations)
- Type compatibility rules

$b: B \leftarrow a: A$	<b>Object Type B</b>	<b>Message Set B</b>
<b>Object Type A</b>	by name	by structure
<b>Message Set A</b>	n/a	by structure

---

# Summary

- Component-oriented software development
  - needs component-oriented programming languages
- Object-oriented programming languages
  - are unsuitable, in part due to interface conflicts
- Separating messages from methods and modules from types
  - leads to a simpler, more regular programming language
  - and supports component-oriented programming better