A Programming Model and Language for Concurrent and Distributed Object-Oriented Systems

Jan Schäfer

University of Kaiserslautern
Department of Computer Science

Supervisors

Prof. Dr. Arnd Poetzsch-Heffter
University of Kaiserslautern

Prof. Dr. Einar Broch Johnsen
University of Olso

5. November 2010

“The Free Lunch is Over”

- Since about 5 years computer processors are **not** getting **faster** anymore, instead they are getting **more parallel**.
- Software must be parallelizable to profit from new processor generations.
- Leading to **“A Fundamental Turn Toward Concurrency in Software”** – Herb Sutter
- The **Internet** is everywhere ⇒ **distributed systems**
- Non-expert programmers have to handle concurrency and distribution

State-of-the-Art in Practice

Concurrency in Mainstream OOLs (e.g. Java and C#)

- Concurrently running **threads**
- **Shared** state
- Synchronization with locks and monitors

State-of-the-Art in Practice

Concurrency in Mainstream OOLs (e.g. Java and C#)

- Concurrently running **threads**
- **Shared** state
- Synchronization with locks and monitors

Problems

- Error-prone (data races, deadlocks)
- Not very modular
- Not suitable for components
- Not suitable for distributed programming
- **Too difficult for most (all) software developers**

State-of-the-Art in Practice

Concurrency in Mainstream OOLs (e.g. Java and C#)

- Concurrently running **threads**
- **Shared** state
- Synchronization with locks and monitors

> "Avoid threads whenever possible"
> – John Ousterhout (Sun Microsystems, 1996)

Problems

- Error-prone (data races, deadlocks)
- Not very modular
- Not suitable for components
- Not suitable for distributed programming
- **Too difficult for most (all) software developers**

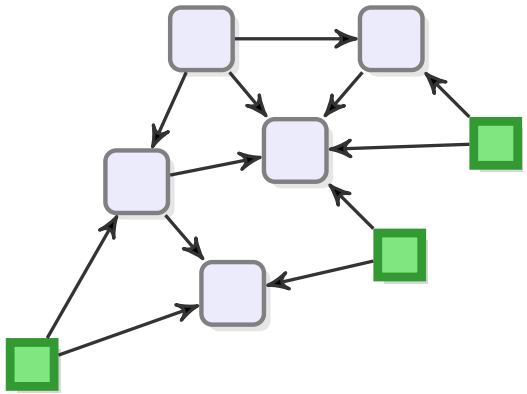
Goal

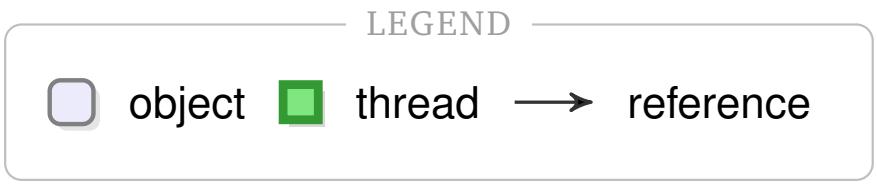
Goal

Develop, formalize, and implement a programming model and language that

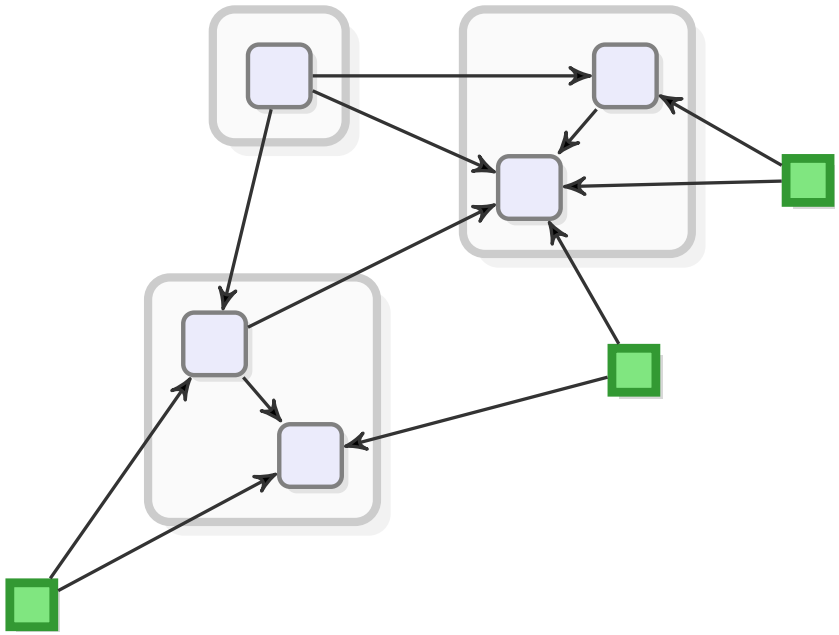
1. Forbids data races
2. Integrate with Object-Orientation
3. Provide a notion of OO-Components
4. Support modularity
5. Suited for distributed programming
6. Support sequential OOP
7. Efficiently implementable
8. Easy to learn and to use

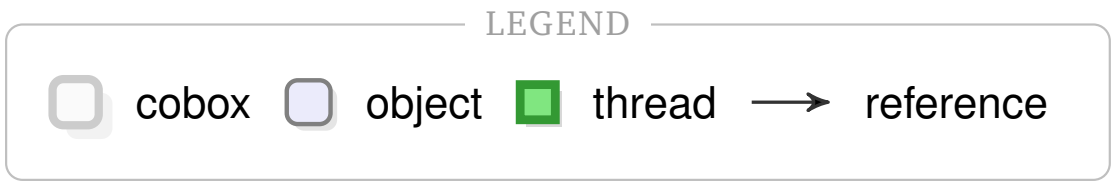
Basic Idea



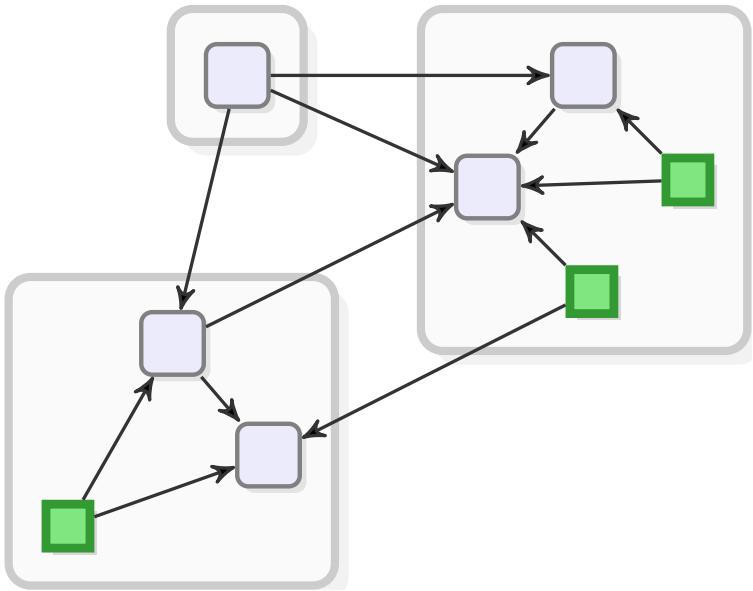


Object Structuring

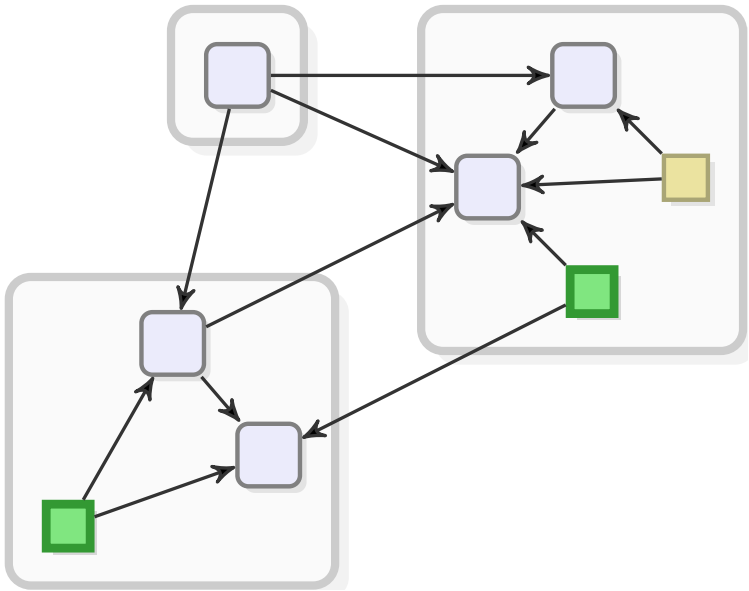


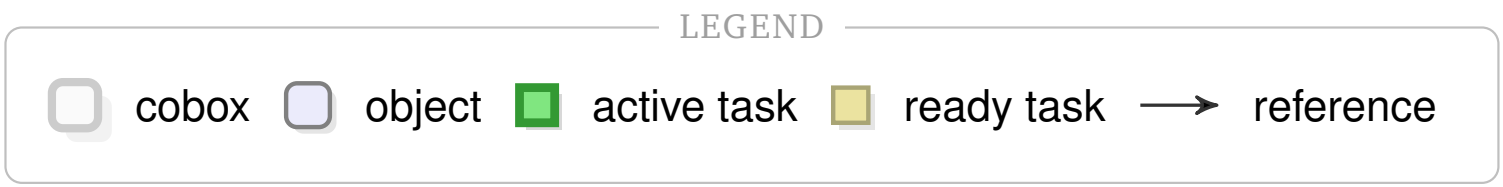


Thread Structuring

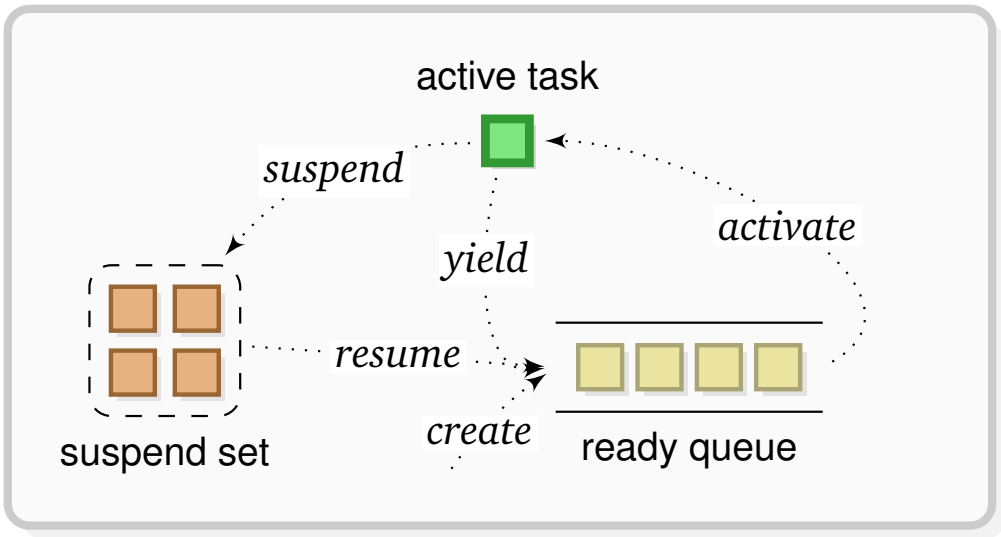


Cooperative Multi-Tasking





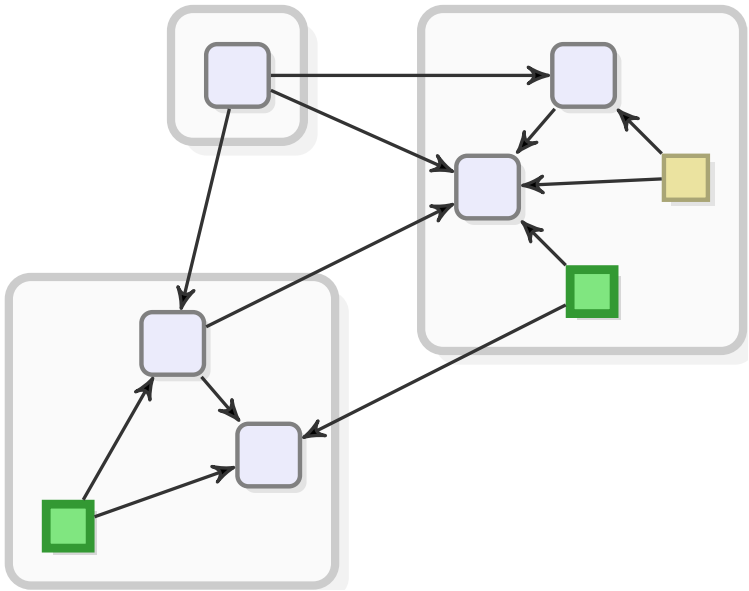
Cooperative Multi-Tasking

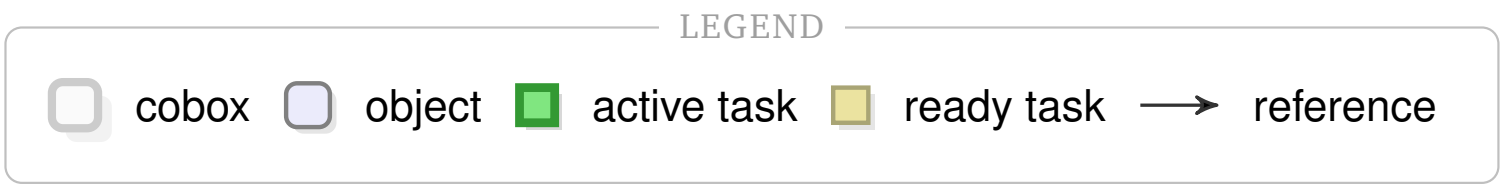


LEGEND

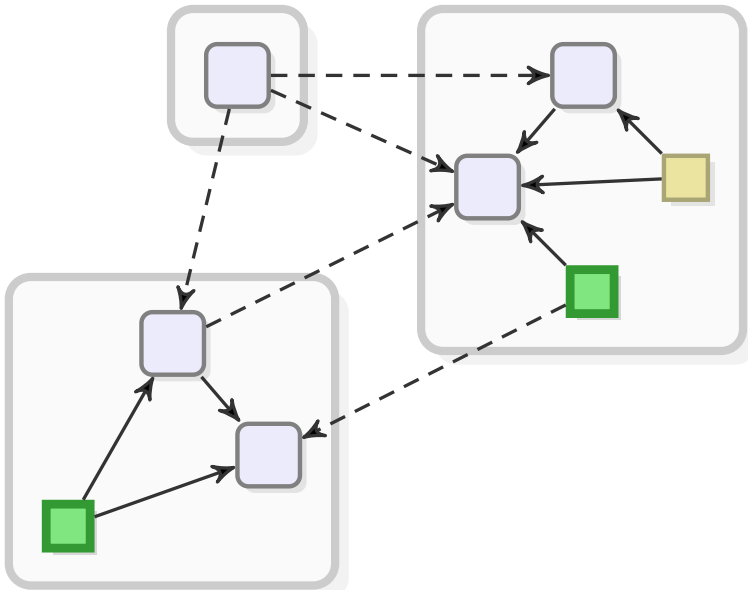
active task ready tasks suspended Tasks task scheduling

Cooperative Multi-Tasking





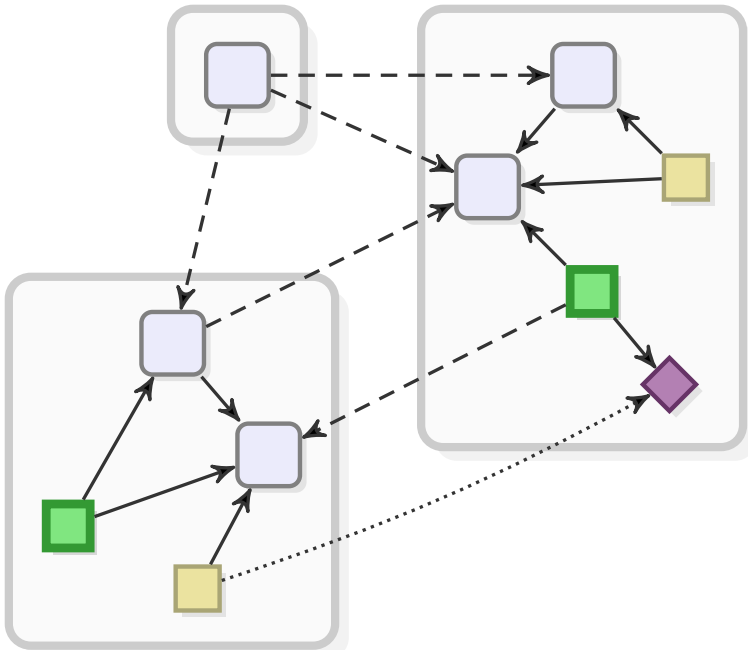
Far References



LEGEND

cobox object active task ready task near ref. far ref.

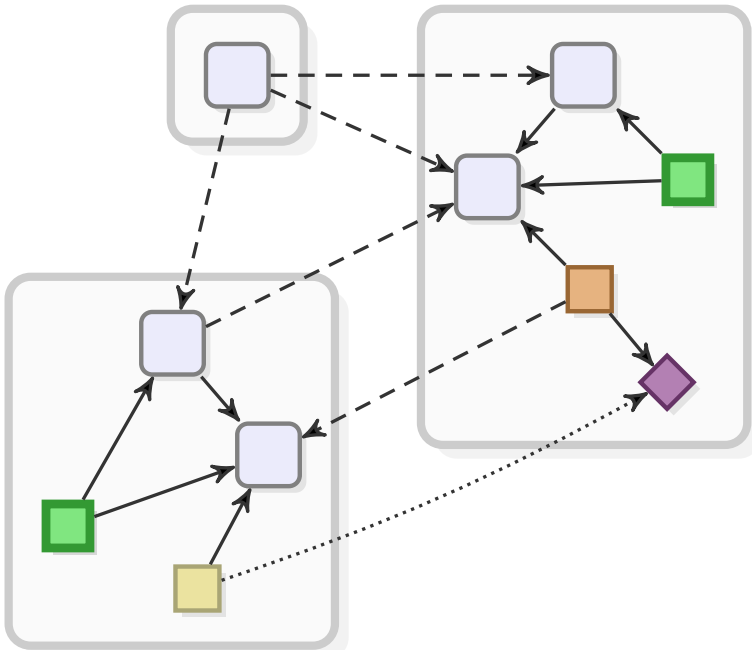
Asynchronous Calls and Futures



LEGEND

future active task ready task near ref. far ref. resolves

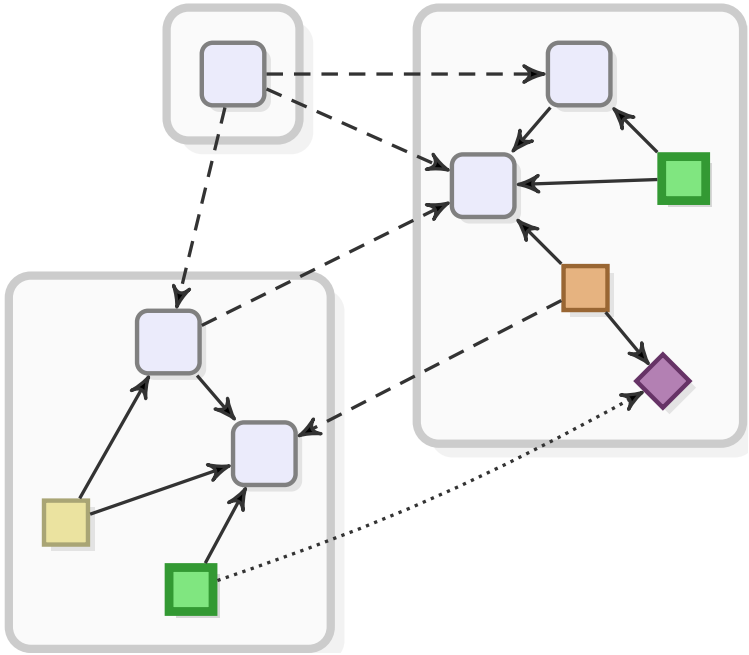
Suspending on a Future



LEGEND

future active task ready task suspended task resolves

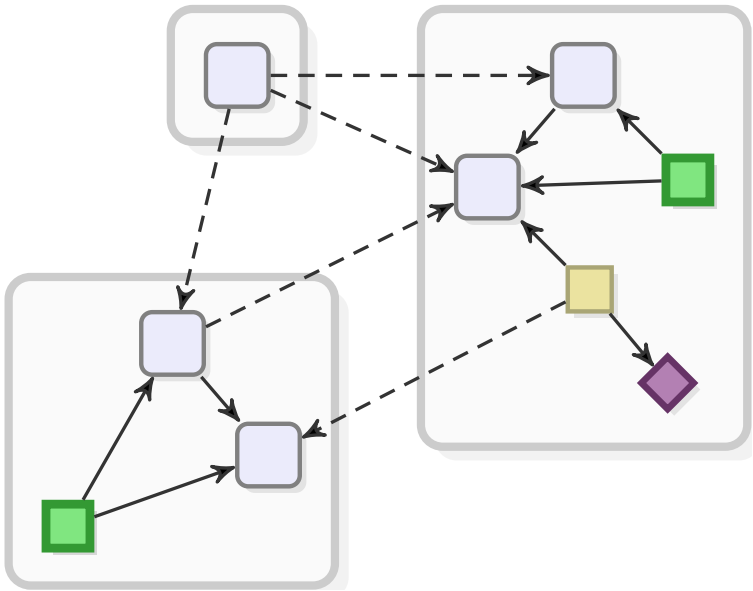
Suspending on a Future

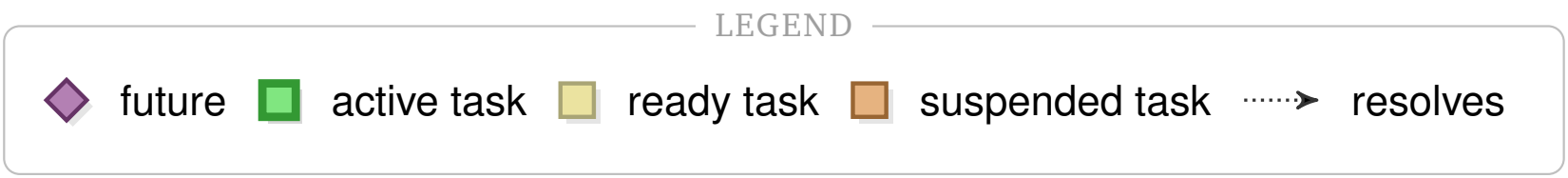


LEGEND

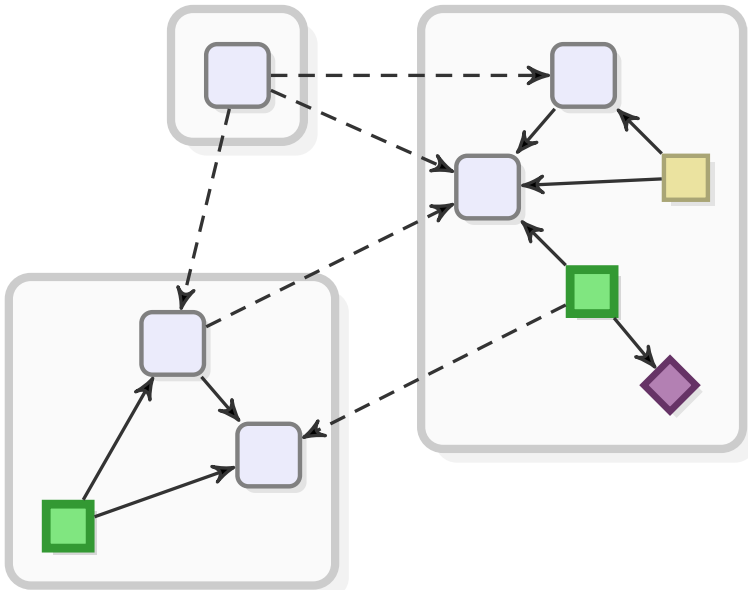
future active task ready task suspended task resolves

Suspending on a Future





Suspending on a Future



LEGEND

future active task ready task suspended task resolves

Further Features

- Promises
- Immutable Objects (passed-by-reference)
- Transfer Objects (passed-by-copy)

Goal Review

Develop, formalize, and implement **a programming model** and language that

1. Forbids data races
2. Integrates with Object-Orientation
3. Provides a notion of OO-Components
4. Supports modularity
5. Suited for distributed programming
6. Supports sequential OOP
7. Efficiently implementable
8. Easy to learn and to use

Goal Review

Develop, formalize, and implement **a programming model** and language that

1. **Forbids data races** ✓
2. Integrates with Object-Orientation
3. Provides a notion of OO-Components
4. Supports modularity
5. Suited for distributed programming
6. Supports sequential OOP
7. Efficiently implementable
8. Easy to learn and to use

Goal Review

Develop, formalize, and implement **a programming model** and language that

1. Forbids data races ✓
2. **Integrates with Object-Orientation** ✓
3. Provides a notion of OO-Components
4. Supports modularity
5. Suited for distributed programming
6. Supports sequential OOP
7. Efficiently implementable
8. Easy to learn and to use

Goal Review

Develop, formalize, and implement **a programming model** and language that

1. Forbids data races ✓
2. Integrates with Object-Orientation ✓
3. **Provides a notion of OO-Components** ✓
4. Supports modularity
5. Suited for distributed programming
6. Supports sequential OOP
7. Efficiently implementable
8. Easy to learn and to use

Goal Review

Develop, formalize, and implement **a programming model** and language that

1. Forbids data races ✓
2. Integrates with Object-Orientation ✓
3. Provides a notion of OO-Components ✓
4. **Supports modularity** ✓
5. Suited for distributed programming
6. Supports sequential OOP
7. Efficiently implementable
8. Easy to learn and to use

Goal Review

Develop, formalize, and implement **a programming model** and language that

1. Forbids data races ✓
2. Integrates with Object-Orientation ✓
3. Provides a notion of OO-Components ✓
4. Supports modularity ✓
5. **Suited for distributed programming** ✓
6. Supports sequential OOP
7. Efficiently implementable
8. Easy to learn and to use

Goal Review

Develop, formalize, and implement **a programming model** and language that

1. Forbids data races ✓
2. Integrates with Object-Orientation ✓
3. Provides a notion of OO-Components ✓
4. Supports modularity ✓
5. Suited for distributed programming ✓
6. **Supports sequential OOP** ✓
7. Efficiently implementable
8. Easy to learn and to use

Goal Review

Develop, **formalize**, and **implement** a programming model and **language** that

1. Forbids data races ✓
2. Integrates with Object-Orientation ✓
3. Provides a notion of OO-Components ✓
4. Supports modularity ✓
5. Suited for distributed programming ✓
6. Supports sequential OOP ✓
7. Efficiently implementable **?**
8. Easy to learn and to use **?**

Actor-Based Programming (Active Objects)

Actor-Based Programming (Active Objects)

Concept

- Every object has its own thread
- State is encapsulated
- Communication by asynchronous messages

Actor-Based Programming (Active Objects)

Concept

- Every object has its own thread
- State is encapsulated
- Communication by asynchronous messages

Problems

- Local state is limited
- Concurrency too fine-grained
- Only one control flow per object

Active Object Extensions

Active Object Extensions

ASP (Caromel et al., 2004)

- Active objects with **multiple passive** objects
- But passive objects not accessible from outside

Active Object Extensions

ASP (Caromel et al., 2004)

- Active objects with **multiple passive** objects
- But passive objects not accessible from outside

E (Miller et al., 2005)

- **Multiple accessible objects**
- But only **one control flow**

Active Object Extensions

ASP (Caromel et al., 2004)

- Active objects with **multiple passive** objects
- But passive objects not accessible from outside

E (Miller et al., 2005)

- **Multiple accessible objects**
- But only **one control flow**

Creol (Johnsen and Owe, 2007)

- **Multiple cooperatively** scheduled control flows
- Integrated with **futures**. Only **single objects**.

Formal Semantics

Rationale

- Precise Semantics
- Provable Properties

Formal Semantics

Rationale

- Precise Semantics
- Provable Properties

Approach

- Formal core language with coboxes
- Static semantics
- Small-step operational semantics
  - Implemented in Maude
- Properties

Formal Core Language

$$
\begin{aligned}
p \in \textbf{Prog} &::= D\ e \\
d \in D \subseteq \textbf{ClassDecl} &::= \mu \text{ class } c \text{ extends } c' \ \{\ \overline{\tau\ f};\ H\ \} \\
\mu \in \textbf{Modifier} &::= \text{cobox} \mid \text{transfer} \mid \text{plain} \\
h \in H \subseteq \textbf{MethDecl} &::= \tau\ m(\overline{\tau\ x})\{e\} \\
e \in \textbf{Expr} &::= x \mid \text{null} \mid \text{let } x = e \text{ in } e' \mid e.f \mid e.f = e' \mid \text{new } c\ [\text{in } e] \mid \\
&\qquad e.m(\overline{e}) \mid e!m(\overline{e}) \mid e.\text{get} \mid e.\text{await} \mid \text{yield} \mid \text{promise } \tau \mid e.\text{fut} \mid e.\text{resolve } e' \\
\tau \in \textbf{Type} &::= c \mid \mathsf{F}\langle\tau\rangle \mid \mathsf{P}\langle\tau\rangle
\end{aligned}
$$

Dynamic Entities

Config	$::= K$	configurations
$k \in K \subseteq$ **Comp**	$::= b \mid \rho$	components
$b \in B \subseteq$ **CoBox**	$::= \mathsf{B}\langle \kappa_\mathrm{b}, O, T, \bar{t} \rangle$	coboxes
$\rho \in P \subseteq$ **Prom**	$::= \mathsf{P}\langle \kappa_\mathrm{p}, O, v_\epsilon \rangle$	promises
$o \in O \subseteq$ **Obj**	$::= \mathsf{O}\langle \iota, c, \overline{v} \rangle$	objects
	$\mid \mathsf{F}\langle \iota, \kappa_\mathrm{p}, v_\epsilon \rangle$	futures
$t \in T \subseteq$ **Tsk**	$::= \mathsf{T}\langle e \rangle$	tasks
$v \subseteq$ **Value**	$::= r \mid \mathsf{null}$	values
$v_\epsilon \subseteq$ **OptValue**	$::= v \mid \epsilon$	optional values
$r \in R \subseteq$ **Ref**	$::= \kappa.\iota \mid \kappa_\mathrm{p}$	references
$\kappa \in$ **CompId**	$::= \kappa_\mathrm{b} \mid \kappa_\mathrm{p}$	component identifiers
$e \in E \subseteq$ **Expr**	$::= \ldots \mid v$	extended expressions
$\iota \in$ **ObjId**		object identifiers
$\kappa_\mathrm{b} \in$ **CoBoxId**		cobox identifiers including MAIN
$\kappa_\mathrm{p} \in$ **PromId**		promise identifiers

Reduction Rules

Global Rules

$$K \longrightarrow K'$$

CoBox-Local Rules

$$b \longrightarrow_{\mathrm{b}} b'$$

Global Integration Rule

$$\frac{b \longrightarrow_{\mathrm{b}} b'}{K \dot{\cup} b \longrightarrow K \cup b'}$$

CoBox-Local Reduction Rules (subset)

(R-Let)

$$\frac{}{\mathsf{B}\langle \kappa_b, O, T, \overline{t} \cdot \mathsf{T}\langle e_\square[\text{let } x = v \text{ in } e]\rangle\rangle \longrightarrow_b \mathsf{B}\langle \kappa_b, O, T, \overline{t} \cdot \mathsf{T}\langle e_\square[[v/x]e]\rangle\rangle}$$

(R-NewObjLocal)

$$\frac{e = \text{new } c \vee e = \text{new } c \text{ in } \kappa_b.\iota' \qquad \neg \mathit{coboxcl}(c) \qquad \iota \notin \mathit{oids}(O)}{\mathsf{B}\langle \kappa_b, O, T, \overline{t} \cdot \mathsf{T}\langle e_\square[e]\rangle\rangle \longrightarrow_b \mathsf{B}\langle \kappa_b, O \cup \mathsf{O}\langle \iota, c, \mathit{init}(c)\rangle, T, \overline{t} \cdot \mathsf{T}\langle e_\square[\kappa_b.\iota]\rangle\rangle}$$

(R-DirectCall)

$$\frac{\mathsf{O}\langle \iota, c, \_\rangle \in O \qquad e' = \mathit{mexpr}(c, m, \kappa_b.\iota, \overline{v})}{\mathsf{B}\langle \kappa_b, O, T, \overline{t} \cdot \mathsf{T}\langle e_\square[\kappa_b.\iota.m(\overline{v})]\rangle\rangle \longrightarrow_b \mathsf{B}\langle \kappa_b, O, T, \overline{t} \cdot \mathsf{T}\langle e_\square[e']\rangle\rangle}$$

(R-FieldSelect)

$$\frac{\mathsf{O}\langle \iota, c, \overline{v}\rangle \in O \qquad \mathit{fields}(c) = \overline{\tau\ f}}{\mathsf{B}\langle \kappa_b, O, T, \overline{t} \cdot \mathsf{T}\langle e_\square[\kappa_b.\iota.f_i]\rangle\rangle \longrightarrow_b \mathsf{B}\langle \kappa_b, O, T, \overline{t} \cdot \mathsf{T}\langle e_\square[v_i]\rangle\rangle}$$

(R-FieldUpdate)

$$\frac{o = \mathsf{O}\langle \iota, c, \overline{v}\rangle \qquad \mathit{fields}(c) = \overline{\tau\ f} \qquad o' = \mathsf{O}\langle \iota, c, \overline{v}[v_i = v]\rangle}{\mathsf{B}\langle \kappa_b, O \uplus o, T, \overline{t} \cdot \mathsf{T}\langle e_\square[\kappa_b.\iota.f_i = v]\rangle\rangle \longrightarrow_b \mathsf{B}\langle \kappa_b, O \cup o', T, \overline{t} \cdot \mathsf{T}\langle e_\square[v]\rangle\rangle}$$

Global Reduction Rules (subset)

(R-NewObjFar)

$$\frac{\mathit{plaincl}(c) \qquad b = \mathsf{B}\langle \kappa_b', O', T', \bar{t}' \rangle \qquad \iota \notin \mathit{oids}(O') \qquad b' = \mathsf{B}\langle \kappa_b', O' \cup \mathsf{O}\langle \iota, c, \mathit{init}(c)\rangle, T', \bar{t}' \rangle}{K \uplus b \uplus \mathsf{B}\langle \kappa_b, O, T, \bar{t} \cdot \mathsf{T}\langle e_\square[\text{new } c \text{ in } \kappa_b'.\iota']\rangle\rangle \longrightarrow K \cup b' \cup \mathsf{B}\langle \kappa_b, O, T, \bar{t} \cdot \mathsf{T}\langle e_\square[\kappa_b'.\iota]\rangle\rangle}$$

(R-NewCoBox)

$$\frac{\mathit{coboxcl}(c) \qquad \kappa_b' \text{ fresh} \qquad b = \mathsf{B}\langle \kappa_b', \{\mathsf{O}\langle \iota, c, \mathit{init}(c)\rangle\}, \varnothing, \bullet \rangle}{K \uplus \mathsf{B}\langle \kappa_b, O, T, \bar{t} \cdot \mathsf{T}\langle e_\square[\text{new } c]\rangle\rangle \longrightarrow K \cup \mathsf{B}\langle \kappa_b, O, T, \bar{t} \cdot \mathsf{T}\langle e_\square[\kappa_b'.\iota]\rangle\rangle \cup b}$$

(R-PromNew)

$$\frac{\kappa_p \text{ fresh}}{K \uplus \mathsf{B}\langle \kappa_b, O, T, \bar{t} \cdot \mathsf{T}\langle e_\square[\text{promise } \tau]\rangle\rangle \longrightarrow K \cup \mathsf{B}\langle \kappa_b, O, T, \bar{t} \cdot \mathsf{T}\langle e_\square[\kappa_p]\rangle\rangle \cup \mathsf{P}\langle \kappa_p, \varnothing, \epsilon \rangle}$$

(R-PromResolve)

$$\frac{\rho = \mathsf{P}\langle \kappa_p, \varnothing, \epsilon \rangle \qquad (O', v') = \mathit{copy}(\kappa_b, O, v, \kappa_p, \varnothing) \qquad \rho' = \mathsf{P}\langle \kappa_p, O', v' \rangle}{K \uplus \rho \uplus \mathsf{B}\langle \kappa_b, O, T, \bar{t} \cdot \mathsf{T}\langle e_\square[\kappa_p.\text{resolve } v]\rangle\rangle \longrightarrow K \cup \rho' \cup \mathsf{B}\langle \kappa_b, O, T, \bar{t} \cdot \mathsf{T}\langle e_\square[\text{null}]\rangle\rangle}$$

(R-PromFut)

$$\frac{\mathsf{P}\langle \kappa_p, \_, \_ \rangle \in K \qquad \iota \notin \mathit{oids}(O)}{K \uplus \mathsf{B}\langle \kappa_b, O, T, \bar{t} \cdot \mathsf{T}\langle e_\square[\kappa_p.\text{fut}]\rangle\rangle \longrightarrow K \cup \mathsf{B}\langle \kappa_b, O \cup \mathsf{F}\langle \iota, \kappa_p, \epsilon \rangle, T, \bar{t} \cdot \mathsf{T}\langle e_\square[\kappa_b.\iota]\rangle\rangle}$$

Rules in Maude (subset)

```
rl [R-Let] :
   let x = v in e => toCxt([ v / x ] e) .

crl [R-NewObjLocal] :
      conf(p, K cb(k,O,T, TL : tsk( ep [ new c ] )))
   => conf(p, K cb(k,O obj(i,c,init(p,c)),T,TL : tsk( ep [ oref(k,i) ] )))
   if i := freshObjId(O) /\
      not isCoBoxClass(p,c) .

crl [R-NewObjLocalIn] :
      conf(p, K cb(k,O,T, TL : tsk( ep [ new c in oref(k,i')] )))
   => conf(p, K cb(k,O obj(i,c,init(p,c)),T,TL : tsk( ep [ oref(k,i) ] )))
   if i := freshObjId(O) /\
      not isCoBoxClass(p,c) .

crl [R-DirectCall] :
     conf(p, K cb(k,O,T,TL : tsk( ep [ oref(k, i) . m ( V ) ])))
  => conf(p, K cb(k,O,T,TL : tsk( ep [ mexpr(p,c,oref(k,i),m,V) ])))
  if O' obj(i,c,vm) := O .

rl [R-FieldSelect] :
      cb(k,O obj(i,c,vm),T, TL : tsk( ep [ oref(k,i) . x ] ))
   => cb(k,O obj(i,c,vm),T, TL : tsk( ep [ vm[x] ]  )) .

rl [R-FieldUpdate] :
      cb(k,O obj(i,c,(vm,x |-> v')),T, TL : tsk( ep [ oref(k,i) . x = v'' ] ))
   => cb(k,O obj(i,c,(vm,x |-> v'')),T,TL : tsk( ep [ v'' ] )) .
```

Properties

- Type-Soundness
  - Preservation
  - Progress
- Data race freedom
- Single-cobox determinism
- Sequential program determinism
- Sequential Java equivalence

JCoBox - Design Decisions

- How to realize sequential programing?
- How to create coboxes?
- How to assign objects to coboxes?
- How to express asynchronous method calls?
- How to express task scheduling?

JCoBox

- Based on standard sequential Java 5
- CoBox classes: `@CoBox`
- Asynchronous method calls: `x!m()`
  - Alternative: `async(x).m()`
- Futures: `Fut<T>`
  - `fut.await()` – cooperative claiming
  - `fut.get()` – exclusive claiming
- Cooperative scheduling: `yield()`
- Immutable classes: `@Immutable`
- Transfer classes: `@Transfer`

Implementation

JCoBox Compiler

- Compiler based on Polyglot Java + JL5
- Generates Java code

Bytecode Rewriter

- Rewrites bytecode (compiled from Java using JCoBox annotations)
- Only supports encoded syntax

Performance Evaluation

Languages

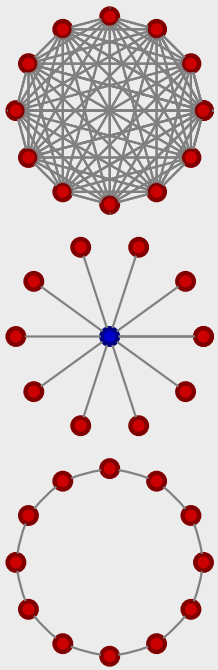
- Scala (v2.7.5, v2.7.7)
- Clojure v1.0.0
- JCoBox

Platforms

- Sun JDK 1.6.16 (-Xmx1024)
- Linux 2.6
- Five machines (1 to 4 cores)

Benchmarks

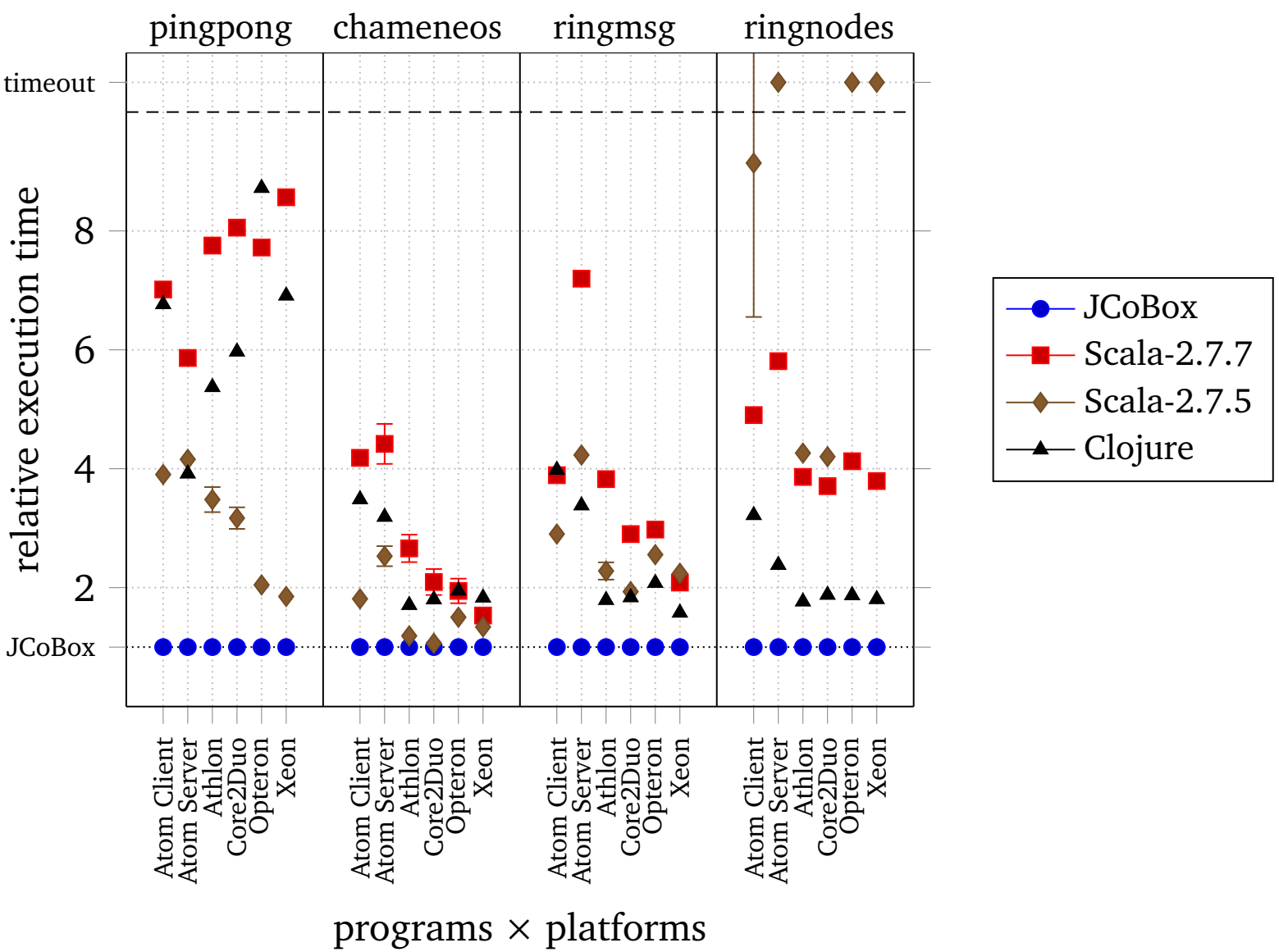
- Big Ping Pong[a]
- Chameneos[b]
- Actor Ring[b]



[a] Srinivasan and Mycroft, 2008

[b] The Computer Language Benchmarks Game

Results


pingpong
chameneos
ringmsg
ringnodes
timeout
relative execution time
8
6
4
2
JCoBox
Atom Client
Atom Server
Athlon
Core2Duo
Opteron
Xeon
programs × platforms
JCoBox
Scala-2.7.7
Scala-2.7.5
Clojure

Goal Review

Develop, formalize, and implement a programming model and language that

1. Forbids data races ✓
2. Integrates with Object-Orientation ✓
3. Provides a notion of OO-Components ✓
4. Supports modularity ✓
5. Suited for distributed programming ✓
6. Supports sequential OOP ✓
7. **Efficiently implementable ?**
8. Easy to learn and to use ?

Goal Review

Develop, formalize, and implement a programming model and language that

1. Forbids data races ✓
2. Integrates with Object-Orientation ✓
3. Provides a notion of OO-Components ✓
4. Supports modularity ✓
5. Suited for distributed programming ✓
6. Supports sequential OOP ✓
7. Efficiently implementable ✓
8. Easy to learn and to use ?

Practical Evaluation

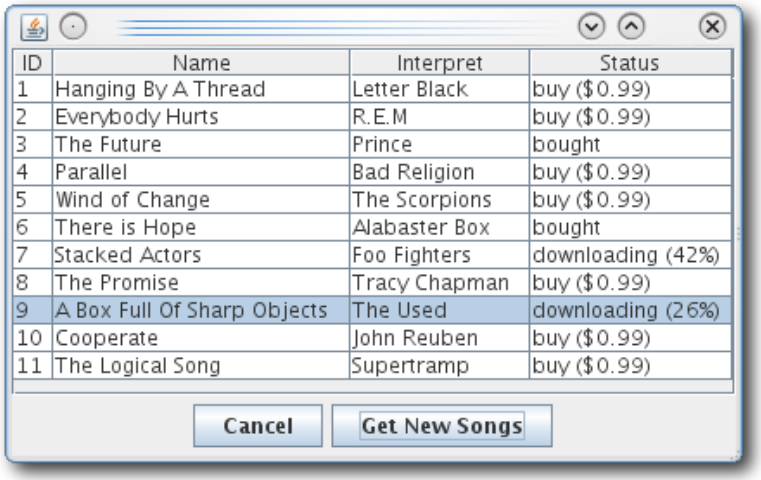
Case Studies

- CoMusic (0.5 kloc)
- FourWins (4 kloc)
- The CoCoME Example (2 kloc)
- Chat Application (0.9 kloc)

Practical Evaluation

Case Studies

- **CoMusic (0.5 kloc)**
- FourWins (4 kloc)
- The CoCoME Example (2 kloc)
- Chat Application (0.9 kloc)



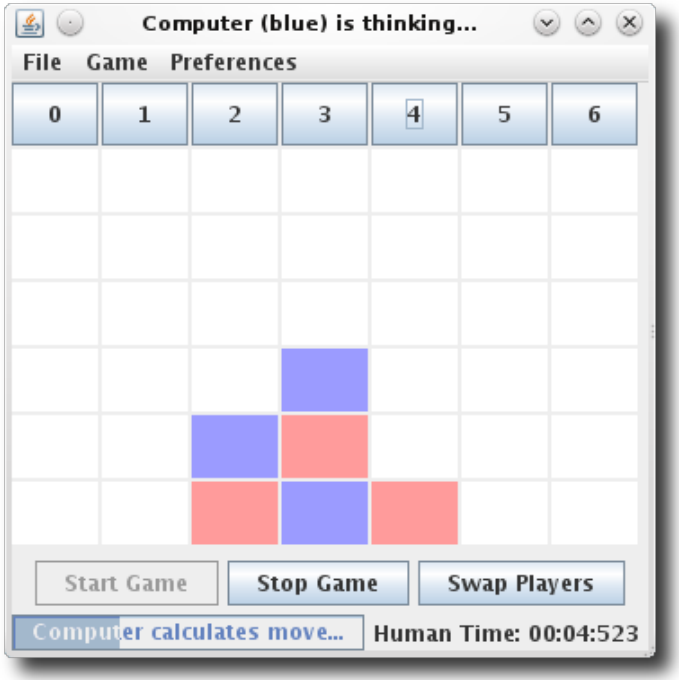
Key Features

- Legacy Java interaction
- Internet access

Practical Evaluation

Case Studies

- CoMusic (0.5 kloc)
- **FourWins (4 kloc)**
- The CoCoME Example (2 kloc)
- Chat Application (0.9 kloc)



Key Features

- Multi-Core utilization
- Interactive

Practical Evaluation

Case Studies

- CoMusic (0.5 kloc)
- FourWins (4 kloc)
- **The CoCoME Example (2 kloc)**
- Chat Application (0.9 kloc)

Key Features

- Distributed
- Component-Based
- Implemented by a student

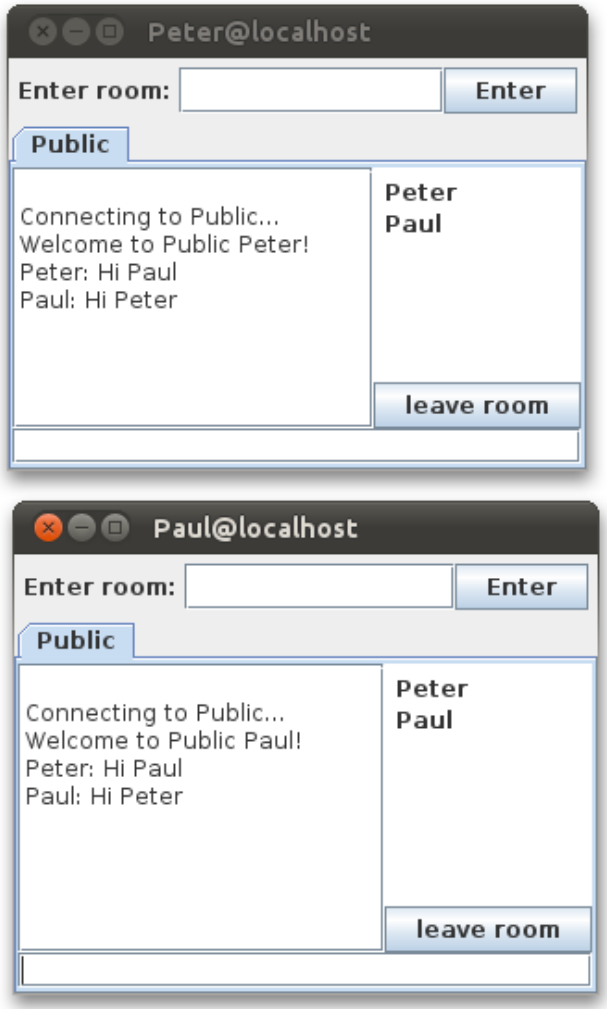
Practical Evaluation

Case Studies

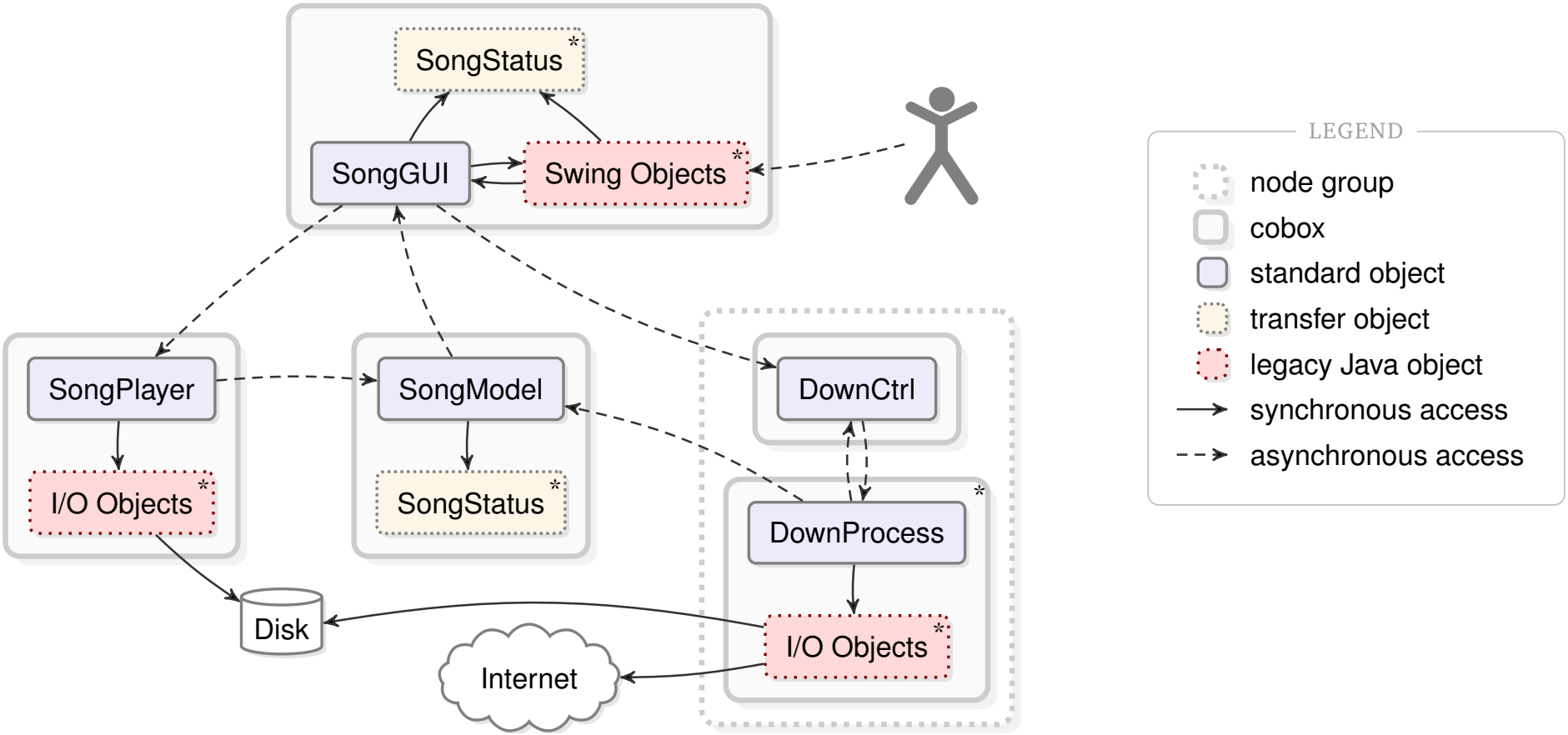
- CoMusic (0.5 kloc)
- FourWins (4 kloc)
- The CoCoME Example (2 kloc)
- **Chat Application (0.9 kloc)**

Key Features

- Distributed



CoMusic Runtime Structure



Goal Review

Develop, formalize, and implement a programming model and language that

1. Forbids data races ✓
2. Integrates with Object-Orientation ✓
3. Provides a notion of OO-Components ✓
4. Supports modularity ✓
5. Suited for distributed programming ✓
6. Supports sequential OOP ✓
7. Efficiently implementable ✓
8. **Easy to learn and to use ?**

Goal Review

Develop, formalize, and implement a programming model and language that

1. Forbids data races ✓
2. Integrates with Object-Orientation ✓
3. Provides a notion of OO-Components ✓
4. Supports modularity ✓
5. Suited for distributed programming ✓
6. Supports sequential OOP ✓
7. Efficiently implementable ✓
8. Easy to learn and to use ✓?

Contributions

1. Development of a new concurrency model for OO
2. Formalization of the model (executable in Maude)
3. Development and implementation of a practical programming language
4. Evaluation of performance and practicability

Thank You.

Back Up

Publications

- ECOOP 2010 - JCoBox: Generalizing Active Objects to Concurrent Components
- IWMSE 2010 - Writing concurrent desktop applications in an actor-based programming model
- FMOODS 2008 - CoBoxes: Unifying Active Objects and Structured Heaps

Outlook

- Parallel State Access
- Different Local Execution Models
- Hierarchical Components
- Distributed Systems
- Efficient Multi-Core Execution
- Specification and Verification

Data Exchange

Immutable Objects

- Never change their state after construction
- Shared between coboxes

Transfer Objects

- Always referenced by near references
- Copied when passed to a different cobox

Preservation and Progress Lemmas

Lemma (Preservation)

Let p be the implicit, fixed program with $\vdash_p p$. *Let* K_n *be a configuration and* Σ *a reference typing with* $\Sigma \models K_n$. ***Assume*** $K_n \longrightarrow K_{n+1}$. *Then there exists a* Σ' *with* $\Sigma \subseteq \Sigma'$ *and* $\Sigma' \models K_{n+1}$.

Lemma (Progress)

Let p be the implicit, fixed program with $\vdash_p p$. *Let* K_n *be a configuration which is* ***not terminal****, and* Σ *be a reference typing with* $\Sigma \models K_n$. *Then there is a* K_{n+1} *with* $K_n \longrightarrow K_{n+1}$.

Type Rules

(T-Program)
$$\frac{p = D\ e \qquad \mathit{uniqueNames}(D) \qquad \mathit{noCycles}(D) \qquad \vdash^{*}_{\mathsf{d}} D \qquad \varnothing;\varnothing \vdash_{\mathsf{e}} e}{\vdash_{\mathsf{p}} p}$$

(T-Decl)
$$\frac{\mathit{noSelfExtend}(c, c') \qquad \mathit{noFieldHide}(c) \qquad \mathit{noOverload}(H) \qquad \mathit{modifierOk}(c, c') \qquad c \vdash^{*}_{\mathsf{h}} H \qquad \vdash^{*}_{\mathsf{t}} \tau}{\vdash_{\mathsf{d}} \_ \text{ class } c \text{ extends } c' \{\ \overline{\tau\ f};\ H\ \}}$$

(T-Method)
$$\frac{h = \tau\ m(\overline{\tau\ x})\{e\} \qquad \mathit{nodups}(\text{this} \cdot \overline{x}) \qquad \mathit{overrideOk}(c, m) \qquad \text{this} : c, \overline{x} : \overline{\tau}; \varnothing \vdash_{\mathsf{e}} e :_{<} \tau}{c \vdash_{\mathsf{h}} h}$$

(T-Null)
$$\frac{\vdash \tau}{\Theta \vdash_{\mathsf{e}} \text{null} : \tau}$$

(T-Var)
$$\frac{\Gamma(x) = \tau}{\Gamma; \Sigma \vdash_{\mathsf{e}} x : \tau}$$

(T-Let)
$$\frac{\Gamma; \Sigma \vdash_{\mathsf{e}} e : \tau \qquad \Gamma, x : \tau; \Sigma \vdash_{\mathsf{e}} e' : \tau'}{\Gamma; \Sigma \vdash_{\mathsf{e}} \text{let } x = e \text{ in } e' : \tau'}$$

(T-New)
$$\frac{\vdash_{\mathsf{t}} c}{\Theta \vdash_{\mathsf{e}} \text{new } c : c}$$

(T-NewIn)
$$\frac{\vdash_{\mathsf{t}} c \qquad \Theta \vdash_{\mathsf{e}} e : c' \qquad \mathit{plaincl}(c)}{\Theta \vdash_{\mathsf{e}} \text{new } c \text{ in } e : c}$$

(T-FieldSelect)
$$\frac{\Theta \vdash_{\mathsf{e}} e : c \qquad \mathit{fields}(c) = \overline{\tau\ f}}{\Theta \vdash_{\mathsf{e}} e.f_i : \tau_i}$$

(T-FieldUpdate)
$$\frac{\Theta \vdash_{\mathsf{e}} e.f : \tau \qquad \Theta \vdash_{\mathsf{e}} e' :_{<} \tau}{\Theta \vdash_{\mathsf{e}} e.f = e' : \tau}$$

(T-Yield)
$$\frac{\vdash_{\mathsf{t}} \tau}{\Theta \vdash_{\mathsf{e}} \text{yield} : \tau}$$

(T-FutAwait)
$$\frac{\Theta \vdash_{\mathsf{e}} e : \mathsf{F}\langle\tau\rangle}{\Theta \vdash_{\mathsf{e}} e.\text{await} : \tau}$$

(T-FutGet)
$$\frac{\Theta \vdash_{\mathsf{e}} e : \mathsf{F}\langle\tau\rangle}{\Theta \vdash_{\mathsf{e}} e.\text{get} : \tau}$$

(T-PromNew)
$$\frac{\vdash_{\mathsf{t}} \tau}{\Theta \vdash_{\mathsf{e}} \text{promise } \tau : \mathsf{P}\langle\tau\rangle}$$

(T-PromResolve)
$$\frac{\Theta \vdash_{\mathsf{e}} e : \mathsf{P}\langle\tau\rangle \qquad \Theta \vdash_{\mathsf{e}} e' :_{<} \tau}{\Theta \vdash_{\mathsf{e}} e.\text{resolve } e' : \tau}$$

(T-PromFut)
$$\frac{\Theta \vdash_{\mathsf{e}} e : \mathsf{P}\langle\tau\rangle}{\Theta \vdash_{\mathsf{e}} e.\text{fut} : \mathsf{F}\langle\tau\rangle}$$

(T-DirectCall)
$$\frac{\Theta \vdash_{\mathsf{e}} e : c \qquad \Theta \vdash^{*}_{\mathsf{e}} \overline{e} :_{<} \overline{\tau} \qquad \mathit{mtype}(c, m) = \overline{\tau} \triangleright \tau}{\Theta \vdash_{\mathsf{e}} e.m(\overline{e}) : \tau}$$

(T-AsyncCall)
$$\frac{\Theta \vdash_{\mathsf{e}} e.m(\overline{e}) : \tau}{\Theta \vdash_{\mathsf{e}} e!m(\overline{e}) : \mathsf{F}\langle\tau\rangle}$$

(T-Sub)
$$\frac{\Theta \vdash_{\mathsf{e}} e : \tau' \qquad \tau' <: \tau}{\Theta \vdash_{\mathsf{e}} e :_{<} \tau}$$

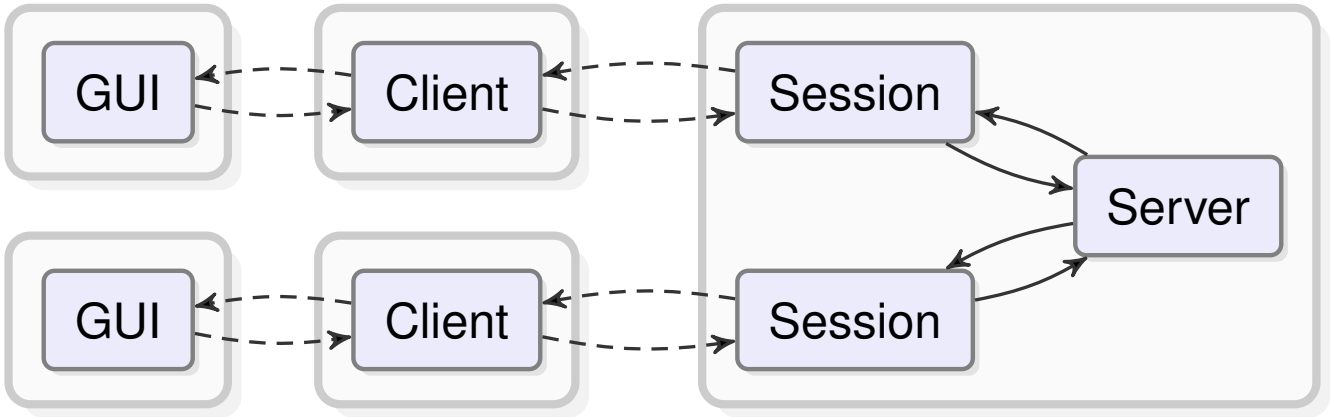
(T-TypeCl)
$$\frac{\mathit{definedcl}(c)}{\vdash_{\mathsf{t}} c}$$

(T-TypeFut)
$$\frac{\vdash_{\mathsf{t}} \tau}{\vdash_{\mathsf{t}} \mathsf{F}\langle\tau\rangle}$$

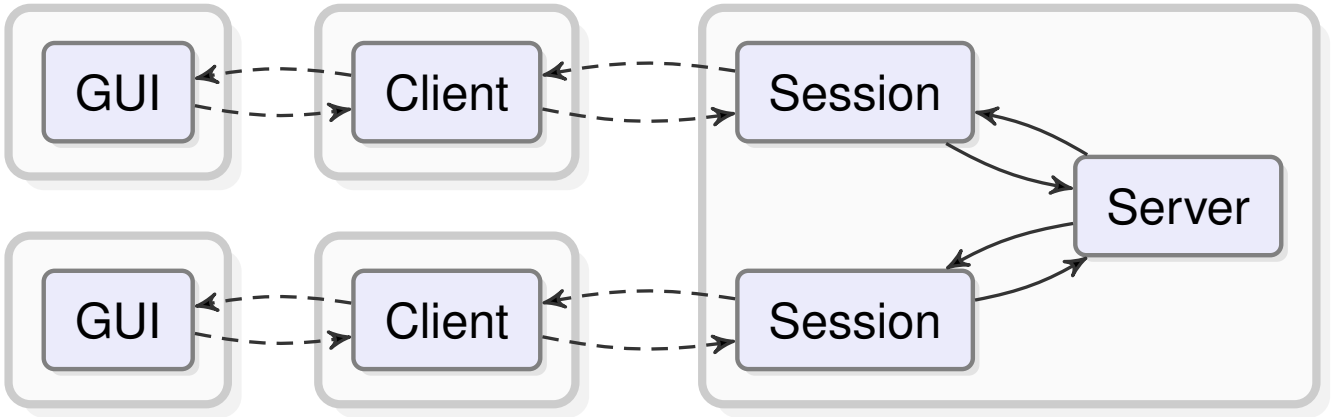
(T-TypeProm)
$$\frac{\vdash_{\mathsf{t}} \tau}{\vdash_{\mathsf{t}} \mathsf{P}\langle\tau\rangle}$$

Example: Chat Application

Example: Chat Application



Example: Chat Application




```
interface Client {
  void onChatMsg(Msg m);
}
interface Server {
  Session connect(Client c);
}
```

```
interface Session {
  void publish(Msg m);
  void keepAlive();
  void close();
}
```

JCoBox Implementation

```
@CoBox class AClient implements Client {
  Session session; GUI gui = new GUI();
  Client() {
    gui!register(this);
  }
  void connect(Server s) {
    Fut<Session> f = s!connect(this);
    session = f.await();
    session!sendMsg("Hello!");
    this!ensureAliveness();
  }
  void ensureAliveness() {
    while (!stopped) {
      session!keepAlive();
      yield(1, TimeUnit.SECONDS);
  } }
  void onChatMsg(Msg m) {
    gui!showMsg(m);
  }
  void msgEntered(Msg m) {
    session!publish(m);
  }
}
```

```
@CoBox class AServer implements Server {
  List<ASession> sessions = ...
  ...
  Session connect(Client c) {
     ASession s = new ASession(c);
     sessions.add(s);
     return s;
  }
  class ASession implements Session {
    Client client;
    long lastAliveSignal;
    ASession(Client c) {
      client = c;
    }
    void publish(Msg m) {
      for (ASession s : sessions) {
         s.client!onChatMsg(m);
    } }
    ...
  }
}
@Immutable class Msg { ... }
```