Scoped Types

A Statically Safe Memory Model for the Real-time Specification for Java

Jan Vitek
&
Tian Zhao & James Noble

PURDUE UNIVERSITY

[support: DARPA PCES and NSF/NASA HDCCSR]

(S³)

Scoped Types for Real-Time Java

Writing functionally correct programs in the RTSJ is harder than in Java, because of the implicit well-formedness relation on references imposed by the RTSJ memory model.

Contributions:

- formalization of the well-formedness relation in a *typed* object calculus
- proof: well-typed program ⇒ no dangling pointers & no memory leaks

Our results can be used as a *checked discipline for RTSJ programs on vanilla VM*

The Real-Time Specification for Java

The Real-Time Specification for Java

Extend Java and VM Specifications with an API that enables creation, verification, analysis, execution, and management of Java threads whose correctness conditions include timeliness constraints

Timeline:

1999 JSR-001 accepted w. 40 companies involved (IBM, Sun...)

2001 RTSJ v.1.0

2002 TimeSys reference implementation

2003 jRate, Ovm open source; jTime product
MacKinac project starts @ Sun Grenoble

2004 RTSJ v.1.0.1

Why Java as a Real-Time Platform?

Why switch to Java?

Software-intensive systems require high-level prog. langs.

C++ not ideal, Ada struggling

Java = lingua franca in education, well specified, ~ simple

combine real-time and plain Java in the same VM

What about the performance myth?

Folklore: *Java 2 times slower than C*; true for hand-tuned code, in practice < 2

Component-based apps easier to optimize in Java because code in a common IR

Dynamic compilers getting better than static compilers

Is Java too dynamic?

Classloading need not be used ⇒ off-line whole-system optimization

Garbage collection still a problem (if you allocate)

PRISMj

Mission critical avionics DRE

Boeing, Purdue, UCI, WUSTL

Route computation, Threat deconfliction algorithms

ScanEagle UAV

System	K LOCs
PRISMJ	109K
FACET EVENT CHANNEL	15K
ZEN CORBA ORB	179K
RTSJ LIBRARIES	60K
CLASSPATH LIBRARIES	500K
OVM VIRTUAL MACHINE	220K

Middleware stack is 1MLOC Java

⇒ 52KLOC w. Ovm optimizing compiler!

PrismJ avionics controller (app layer)

FACET event channel

ZEN Object Request Broker

Real-time Specification for Java (User level implementation)

kernel boundary

Ovm virtual machine kernel

3 rate groups (20, 5, 1Hz)
performance 2x jTime,
≈ Sun product VM

Embedded Planet PowerPC 8260

```
Core at 300 MHz
256 Mb SDRAM
32 Mb FLASH
PC/104 mechanical sized
Embedded Linux
```

The Real-Time Specification for Java

New language in Java clothing

no changes to syntax but idiomatic reinterpretation of existing constructs

- Thread Scheduling & Dispatching
- Synchronization
- Asynchronous Actions
- Memory management
- *Time, Clocks and Timers*

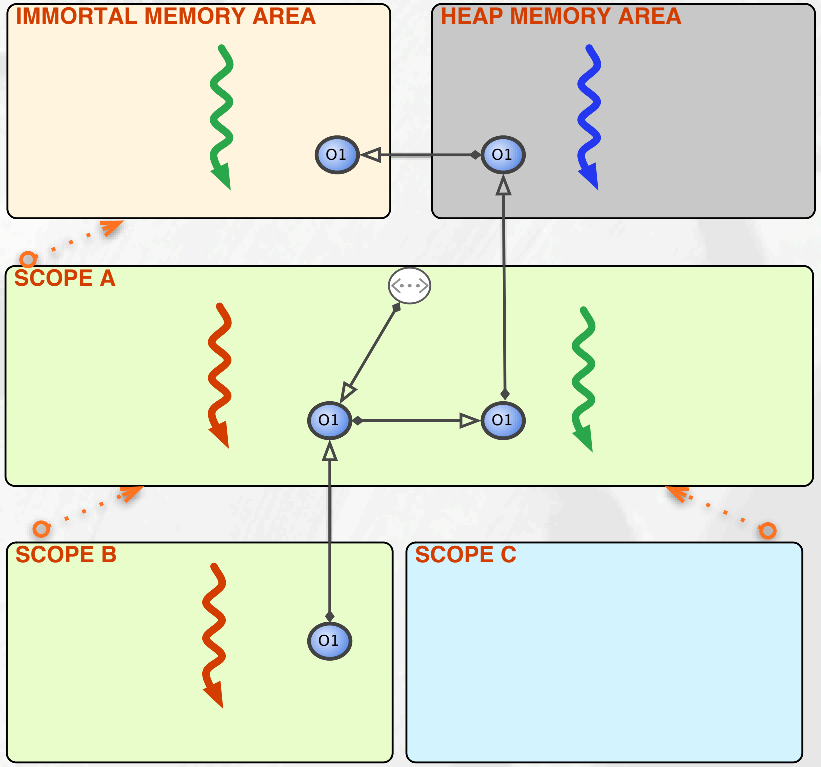
RTSJ Design Overview

Challenges

hard/soft/non RT codes must be allowed to operate in the same execution environment (90/9/1 rule)

RT threads should never wait for the garbage collector (GC)

prevent undesirable interferences, e.g. RT thread blocks while waiting for a plain thread to release a monitor



Scoped Memory

Object lifetime controlled by reachability, when last thread leaves an area, all objects allocated in it reclaimed

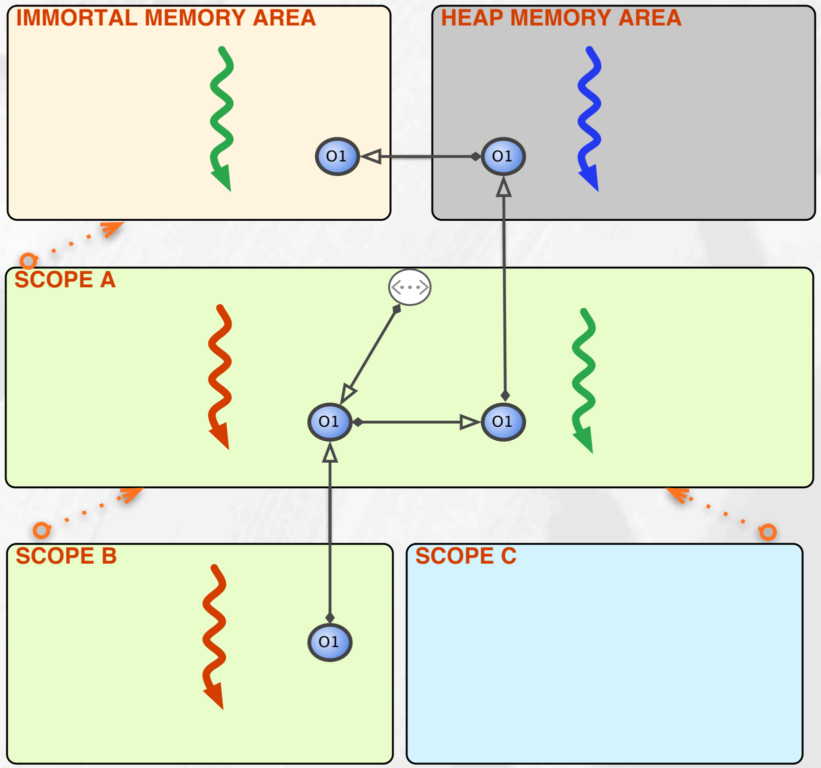
Eliminate GC latency; allow temporary objects

NHRT don't read from heap, thus protected from GC interference

Allocation time linear in size, deallocation O(1) (modulo finalizers)

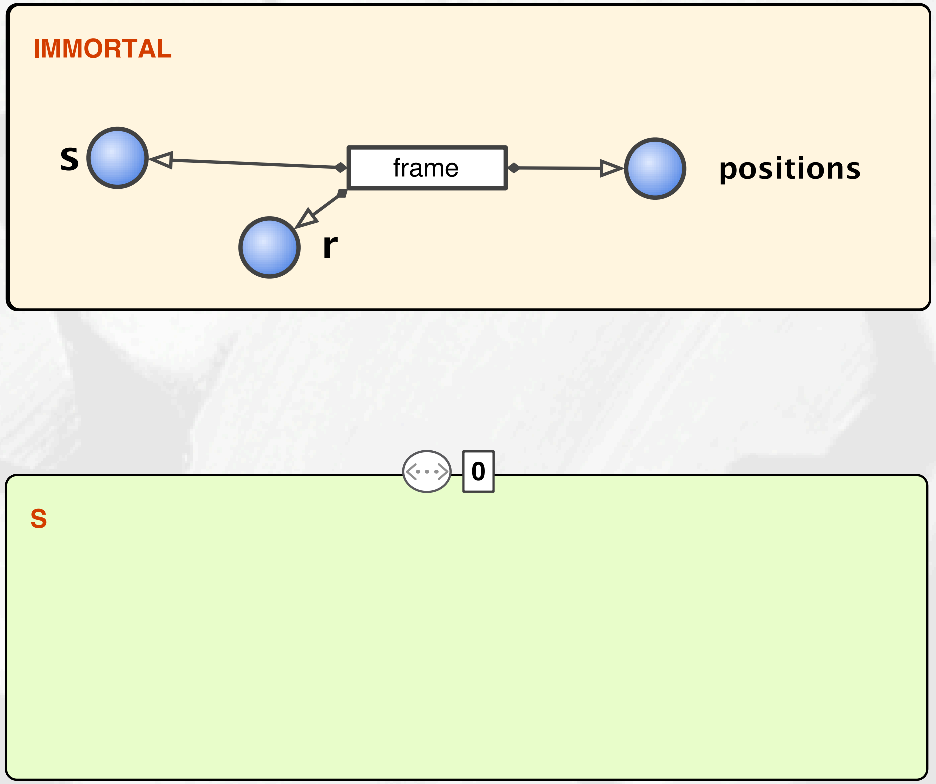
Multiple threads communicate through portal

Nesting is dynamic, established by entry order; can change for the same scope



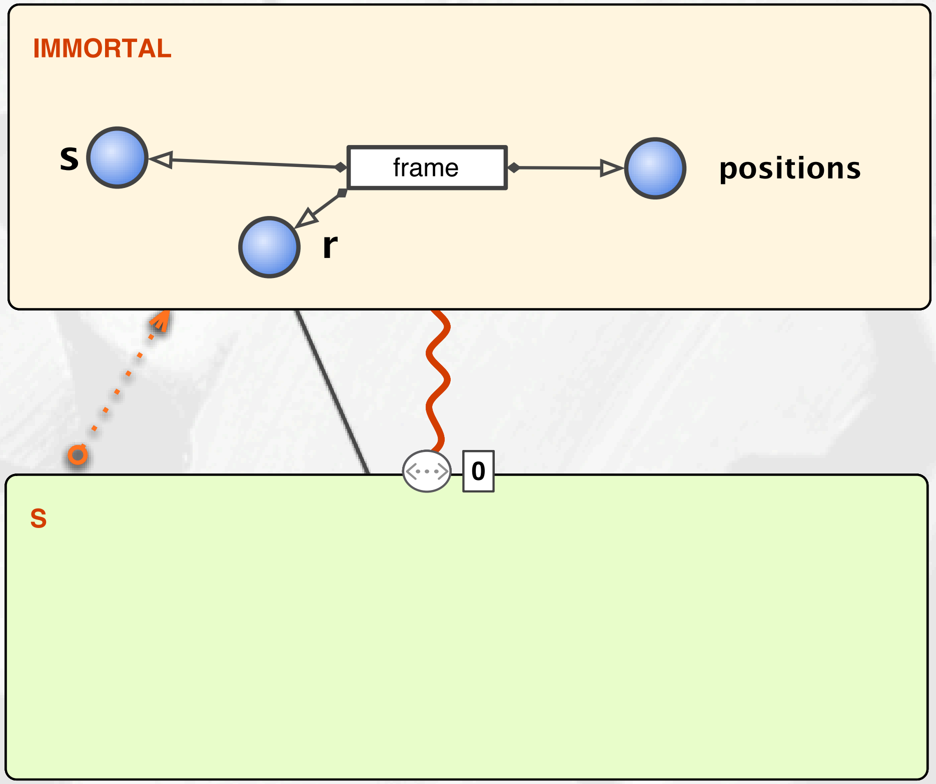
Scoped Memory Usage

```
positions = new float[size];
s= new ScopedMemoryArea(min,max);
r=new Runnable() {
    void run() {
        ...read sensor output...
        tmp = new float[4*size];
        for(i=0;i<size;i++)
            positions[i]=...tmp...
}};
s.enter(r);
done();
```



Scoped Memory Usage

```
positions = new float[size];
s= new ScopedMemoryArea(min,max);
r=new Runnable() {
    void run() {
        ...read sensor output...
        tmp = new float[4*size];
        for(i=0;i<size;i++)
            positions[i]=...tmp...
}};
s.enter(r);
done();
```



Scoped Types

Scoped Types

- A variant of ownership types.
- Related to region-types and separation logics.
- Formalized in a simple extension of Featherweight Java with threads & state.

- *see also:*

  Flexible alias protection. Noble, Vitek, Potter [ECOOP'98]

  Scoped Types for Real-time Java. Zhao, Noble, Vitek [RTSS04]

  Ownership Types for Safe Region-Based Memory Management in Real-Time Java, Boyapati, Salcianu, Beebee, Rinard [PLDI03]

Scoped Types programming model

(set of) scopes	≡	Java package
nested scope	≡	nested package
`ScopedMemory` object	≡	`ScopedGate` object
`enter()`	≡	method invocation
`IllegalAssignment`	≡	compile-time error
scope cycle error	≡	compile-time error

Validity constraints for ST programs

$\mathcal{C}1$	A scoped type is visible only to classes in the same package or subpackages.
$\mathcal{C}2$	A scoped type can only be widened to other scoped types in the same package.
$\mathcal{C}3$	The methods invoked on a scoped type must be defined in the same package.
$\mathcal{S}1$	A gate type is only visible to the classes in the immediate super-package.
$\mathcal{S}2$	A gate type cannot be widened to other types.
$\mathcal{S}3$	The methods invoked on a gate type must be defined in the same class.

A Scoped Type Program

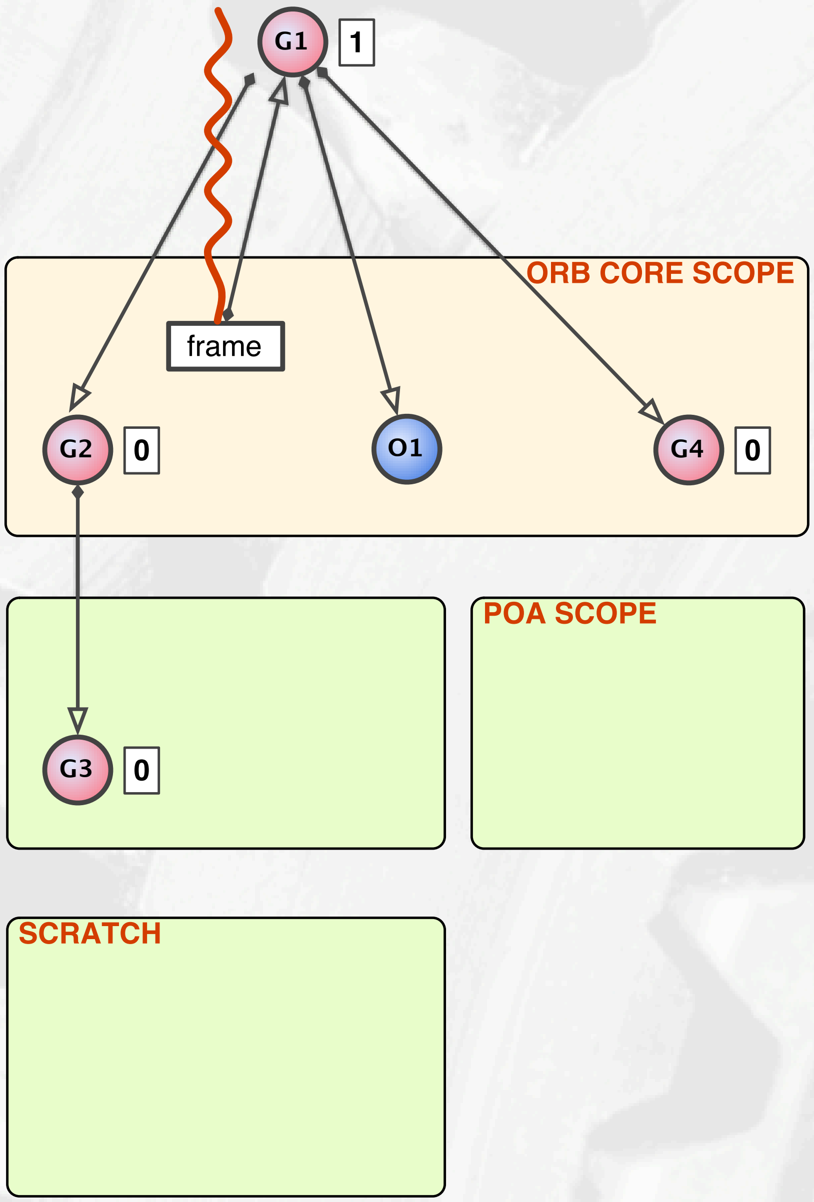
```
package corba.orb

public class ORB
        extends ScopedGate {
    POA[] poas = new POA[10];
    Message msg = new Message();

    public void handleRequest(Buffer b) {
        msg.init(b);
        POA poa = findPoa(msg);
        poa = (pos==null) ?
            addPoa(new Poa(msg)) : poa;
        poa.handleRequest(msg);
    }
    ...
}
public class Message {
    ...
    public Message duplicate() {
        return new Message(...); }
}
```

```
package corba.orb.poa

public class POA
        extends ScopeGate {
    Scratch scope = new Scratch();
    void handleRequest(Message msg) {
        Message message = msg;
        if (...)
            message = msg.clone();
        scratch.dispatch(message);
        scratch.reset();
    }
}
```

```
package corba.orb.poa.scratch

public class Scratch
        extends ScopeGate {
    void dispatch(Message m) {
        ... }
    ...
}
```

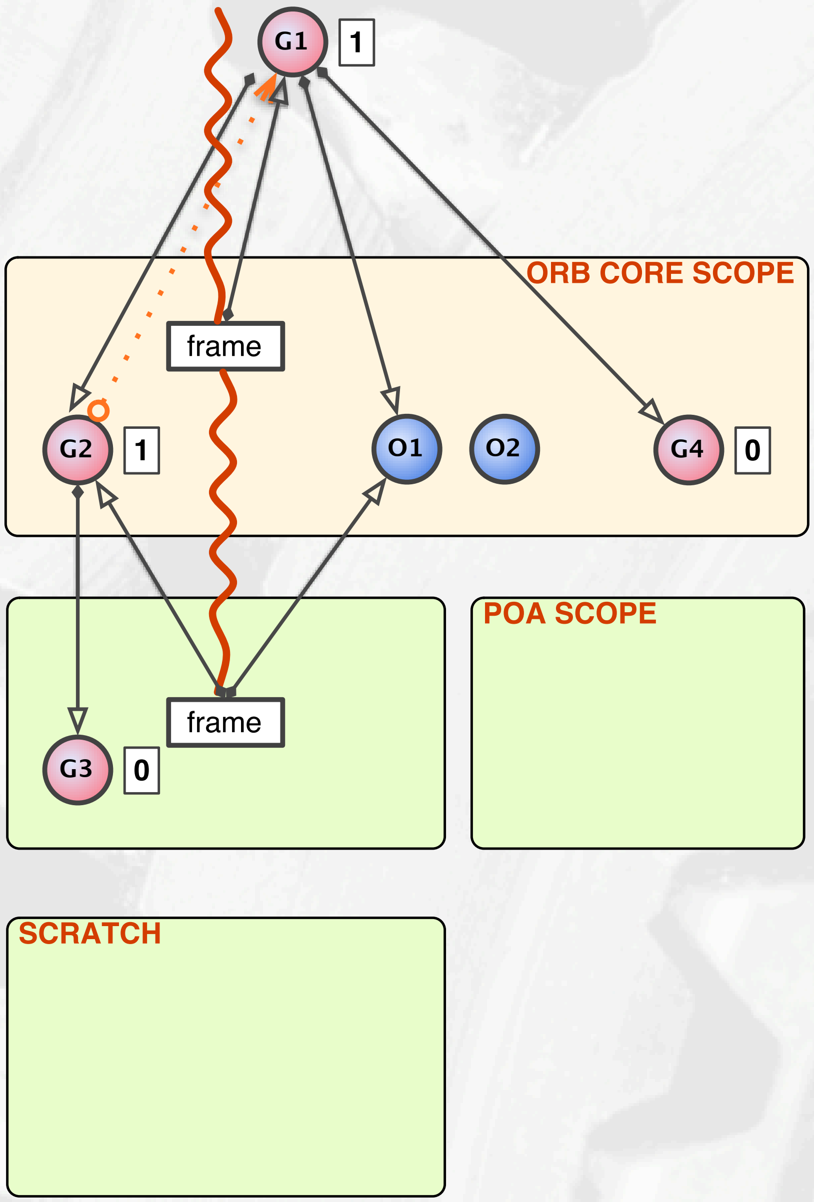
Scoped Program

- Every dynamic scope is
  - implemented by a package in the source
  - represented by a gate at runtime
- Several scope of the same kind can be instantiated
- Gates are norma Java objects with fields pointing into the scope



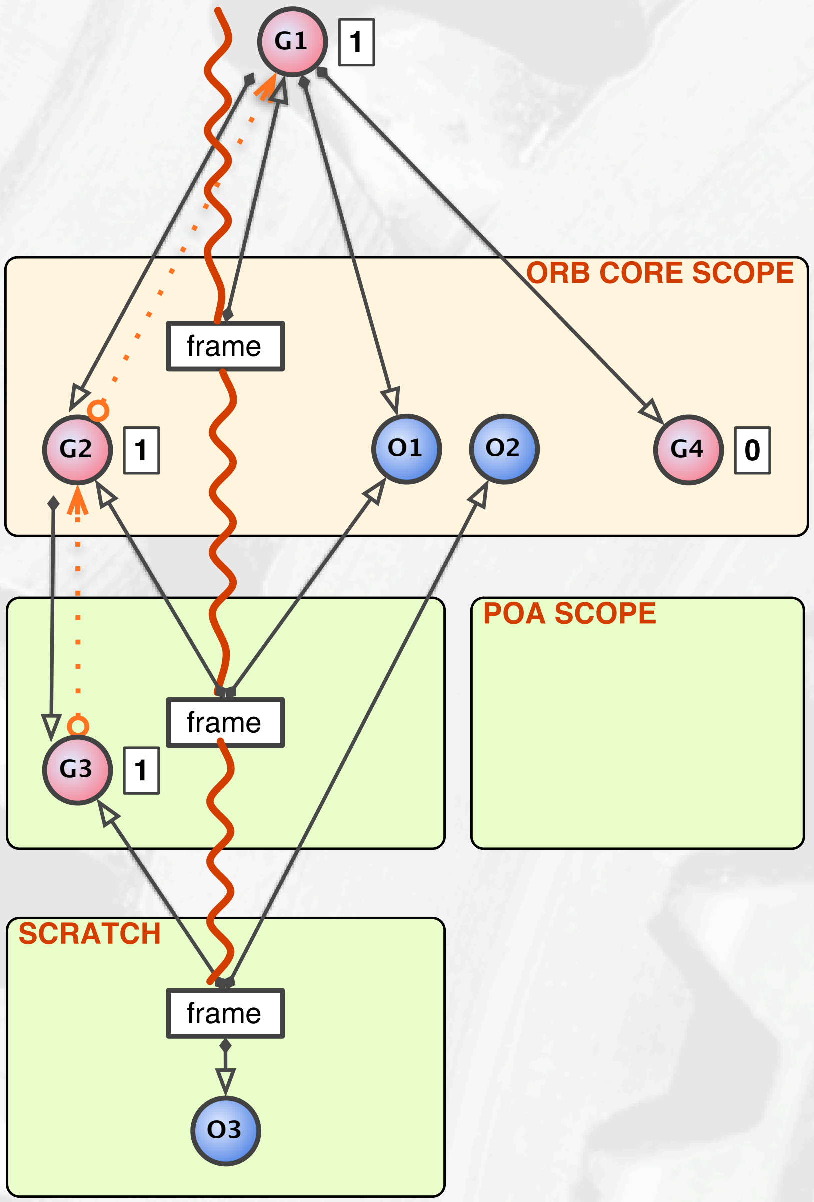
Scoped Program

- Every call to a gate switches the allocation context
- scoped classes can refer to objects in the parent package
- Gates have an associated reference count



Scoped Program

- Scopes are cleared by calling reset() on a gate with RC=0.
- Code duplication may arise if the same class must be used in different scopes.



Scoped Java Calculus

$$
\begin{array}{lcl}
\mathtt{L} & ::= & \circ\ \mathtt{class}\ \mathtt{P.C} \lhd \mathtt{D}\ \{\,\bar{\mathtt{C}}\ \bar{\mathtt{f}};\mathtt{K}\ \bar{\mathtt{M}}\,\} \\
\mathtt{K} & ::= & \mathtt{C()}\ \{\mathtt{super}();\ \mathtt{this}.\bar{\mathtt{f}} := \overline{\mathtt{new}\ \mathtt{D}()};\ \} \\
\mathtt{M} & ::= & \mathtt{C}\ \mathtt{m}(\bar{\mathtt{C}}\ \bar{\mathtt{x}})\ \{\,\mathtt{return}\,\mathtt{e};\ \} \\
\mathtt{e} & ::= & \mathtt{x} \mid \mathtt{e.f} \mid \mathtt{e.m}(\bar{\mathtt{e}}) \mid \mathtt{new}\ \mathtt{C}() \mid \mathtt{e.f} := \mathtt{e} \\
 & & \mid \mathtt{spawn}\,\mathtt{e} \mid \mathtt{reset}\ \mathtt{e} \mid \mathtt{v} \\
\circ & ::= & \mathtt{gate} \mid \mathtt{scoped} \qquad \mathtt{v} ::= \ell \qquad \mathtt{P} ::= \mathtt{p} \mid \mathtt{p.P}
\end{array}
$$

Fig. 7. Syntax of the SJ calculus.

The Scoped Type System

$$\Gamma, \Sigma \vdash \mathtt{x} : \Gamma(\mathtt{x})$$

$$\Gamma, \Sigma \vdash \ell : \Sigma(\ell)$$

$$\frac{\Gamma, \Sigma \vdash \mathtt{e}_0 : \mathtt{C} \quad \mathit{fields}(\mathtt{C}) = (\overline{\mathtt{C}}\ \overline{\mathtt{f}})}{\Gamma, \Sigma \vdash \mathtt{e}_0.\mathtt{f}_\mathtt{i} : \mathtt{C}_\mathtt{i}}$$

$$\frac{\Gamma, \Sigma \vdash \mathtt{e}_0 : \mathtt{C}_0 \quad \mathit{fields}(\mathtt{C}_0) = (\overline{\mathtt{C}}\ \overline{\mathtt{f}}) \qquad \Gamma, \Sigma \vdash \mathtt{e} : \mathtt{C} \quad \boxed{\mathtt{C} \preceq \mathtt{C}_\mathtt{i}}}{\Gamma, \Sigma \vdash \mathtt{e}_0.\mathtt{f}_\mathtt{i} = \mathtt{e} : \mathtt{C}_\mathtt{i}}$$

$$\frac{\Gamma, \Sigma \vdash \mathtt{e}_0 : \mathtt{C}_0 \quad \mathit{mdef}(\mathtt{m},\ \mathtt{C}_0) = \mathtt{C}'_0 \qquad \mathit{mtype}(\mathtt{m},\ \mathtt{C}'_0) = \overline{\mathtt{C}} \rightarrow \mathtt{C} \qquad \Gamma, \Sigma \vdash \overline{\mathtt{e}} : \overline{\mathtt{D}} \quad \boxed{\overline{\mathtt{D}} \preceq \overline{\mathtt{C}} \quad \mathtt{C}_0 \preceq \mathtt{C}'_0}}{\Gamma, \Sigma \vdash \mathtt{e}_0.\mathtt{m}(\overline{\mathtt{e}}) : \mathtt{C}}$$

$$\frac{\Gamma, \Sigma \vdash \mathtt{e} : \mathtt{Thread}}{\Gamma, \Sigma \vdash \mathtt{spawn}\ \mathtt{e} : \mathtt{Thread}}$$

$$\frac{\Gamma, \Sigma \vdash \mathtt{e} : \mathtt{C} \quad \mathtt{C} \text{ is a gate}}{\Gamma, \Sigma \vdash \mathtt{reset}\ \mathtt{e} : \mathtt{C}}$$

$$\Gamma, \Sigma \vdash \mathtt{new}\ \mathtt{C}() : \mathtt{C}$$

Correctness

- If it is the case that

  $P \equiv P'' \mid t[\ \ldots\ .l\ E\ [\ \textbf{reset}\ l_0\ ]\ ]$

  $\sigma\ P$ *is well typed*

  $\sigma\ P \Rightarrow \sigma'\ P'$ *where* $P' \equiv P'' \mid t[\ \ldots\ l\ E[\ l_0\ ]]$

- then

  objects allocated in the scope represented by gate $\sigma\ l_0$ are not reachable in σ' P′

  (i.e. no dangling pointers)

[Zhao, Noble, Vitek. RTSS'04]

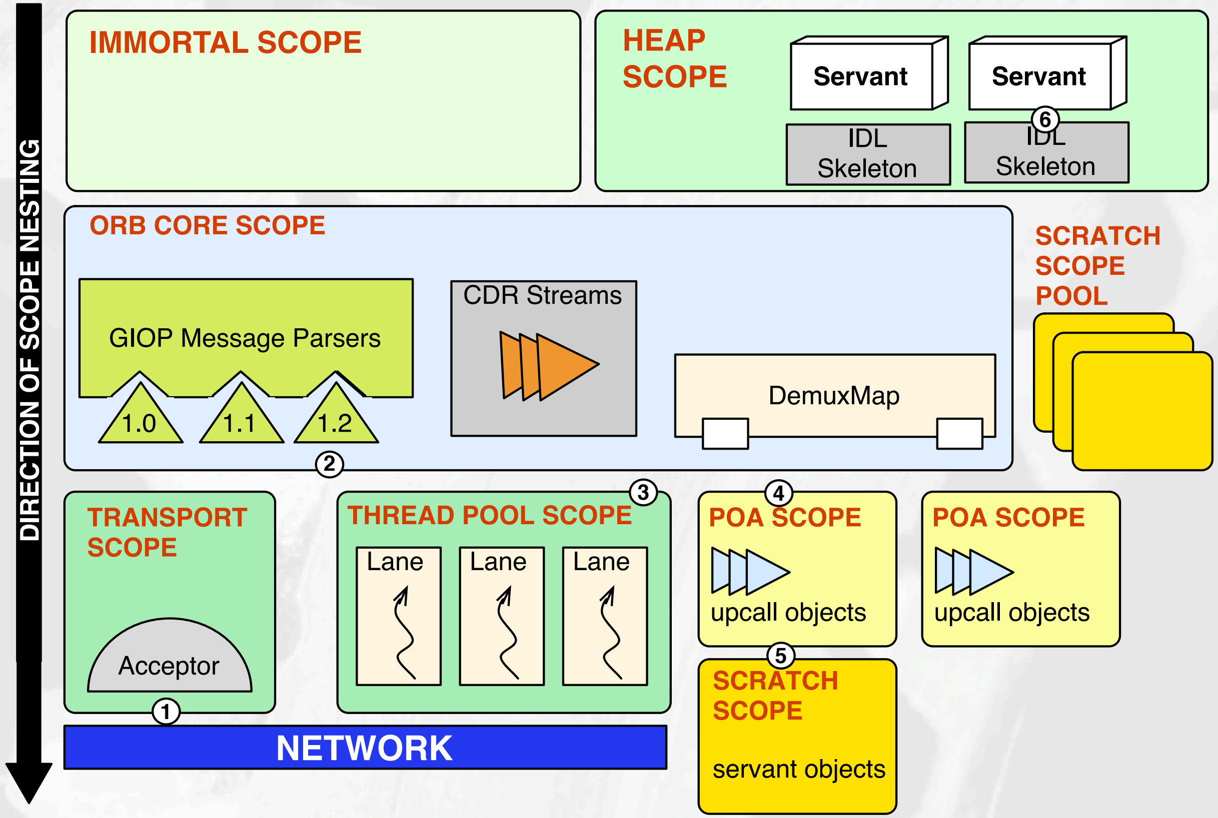
Empiric Validation

- Zen is a RT-CORBA object request broker
- ~ 100 KLoc written in RTSJ at UCI
- with a rich (ie. complex) memory scope structure
- scopes protect the ORB core from interference
- use RTSJ design patterns

  Bridge, Wedge Thread,

  ScopedRunLoop, EIR

  Real-Time Java scoped memory: design patterns and semantics. Pizlo, Fox, Holmes, Vitek. [ISORC04]



Refactoring ZEN

Successfully refactored Zen

Eliminated ~30 classes out of 186, little code duplication

Software structure became easier to understand

Several bugs were discovered

Faster execution times

zen.orb	38	zen.orb	16
zen.orb.any	2	–	
zen.orb.any.monolithic	1	–	
zen.orb.dynany	11	–	
zen.orb.giop	6	zen.org.giop	4
zen.orb.giop.IOP	3	zen.orb.giop.IOP	3
zen.orb.giop.type	5	zen.orb.giop.type	5
zen.orb.giop.v1_0	9	zen.orb.giop.v1_0	9
zen.orb.giop.v1_1	5	zen.orb.giop.v1_1	5
zen.orb.giop.v1_2	4	zen.orb.giop.v1_2	4
zen.orb.policies	13	zen.orb.policies	9
zen.orb.resolvers	2	–	
zen.orb.transport	11	zen.orb.transport	3
zen.orb.transport.iiop	4	zen.orb.transport.iiop	1
zen.poa	16	zen.poa	3
zen.poa.mechanism	27	zen.poa.mechanism	19
zen.poa.policy	7	zen.poa.policy	7
zen.util	21	zen.util	11
		scope.orb	45
		scope.orb.connection	7
		scope.orb.requestprocessor	10
		scope.requestwaiter	3

... 2nd round of refactoring in progress

Conclusions

- Java is safer than C++ because
  - VM guarantees memory errors do not occur ⇒ increased productivity
  - comes at cost in performance/predictability
  - GC = a system-managed memory leak
- RTSJ is memory-safe but
  - harder to use because of extra dimension (locality)
  - errors are reported at run-time ⇒ decreased reliability
  - memory leaks are reported eagerly
- Scoped Types
  - prevent dynamic access violation and
  - structure code so as to reflect a program's memory layout

Credits

- Ovm Core:

  SUN: *Krzysztof Palacz*

  DLTech (Brisbane): *David Holmes*

  Purdue: *Jason Baker, Chap Flack, Filip Pizlo, Hiroshi Yamauchi*

- ... and also

  Christian Grothoff, Marek Prochazka, Andrei Madan (Medtronics), Jacques Thomas, James Liang (JPL), Antonio Cunei

- *Scoped Types*

  Tian Zhao (UWisc) and
  James Noble,
  Alex Pontanin (U Vic, NZ)


STOP