ORTHOGONAL TO THE JAVA IMPERATIVE

Suad Alagić, Jose Solorzano and David Gitchell

Department of Computer Science
Wichita State University

July 1998

Java limitations

- Parametric polymorphism
- Persistence
- Assertions

Point of view: *Persistent* and *database systems*

Constraint language

- *Declarative*
- Temporal *logic* based
- Based on an *advanced type system* (F-bounded polymorphism and self types)
- *Persistent*

Integrated into the *Java environment*

Assertions

- *Initial* constraints
- *Preconditions*
- *Postconditions*
- *Class invariants*
- *History properties*
- *Transition* constraints

Collection classes

Class Collection[T]
Observers
belongs(T)
Mutators
insert(T); delete(T)
Constraints
ForAll X:T;
$\Box$(self.belongs(X) $\leftarrow$ self.delete(X)),
$\Box$($\bigcirc$self.belongs(X) $\leftarrow$ self.insert(X))
End Collection.

Inheritance

Class Bag[T]
Inherits Collection
Observers
occurs(T, Natural)
Constructors
union(Bag[T]): Bag[T]
Constraints
ForAll B: Bag[T]; X:T; M, N: Natural;
new(Bag[T]).occurs(X,0) $\leftarrow$,
$\Box$(self.belongs(X) $\leftarrow$ self.occurs(X, N), N.gtr(0)),
$\Box$($\bigcirc$self.occurs(X, N.succ()) $\leftarrow$
self.occurs(X, N), self.insert(X)),
$\Box$($\bigcirc$self.occurs(X, N.pred()) $\leftarrow$
self.occurs(X, N), self.delete(X)),
$\Box$(self.union(B).occurs(X, M.max(N))
$\leftarrow$ self.occurs(X, M), B.occurs(X,N)),

Queries

Queries as *objects*

Class Query[T]
Observers
qualification(T)
Constructors
eval(Collection[T]): Collection[T]
Constraints
ForAll X: T; S: Collection[T];
$\Box$(self.eval(S).belongs(X)$\leftarrow$
S.belongs(X),
self.qualification(X))
End Query.

Queries

Class MyQuery[Employee]
Inherits Query[Employee]
Constraints
ForAll X: Employee; N: Real;
$\Box$(self.qualification(X)$\leftarrow$
    X.job("programmer"),
    X.salary(N),
    N.gtr(70,000.00))
End MyQuery.

Type-safe reflection

Bounded type quantification

Class OrderedCollection[T **Inherits** Ordered]
Inherits Collection[T]
...
End OrderedCollection.

Class Ordered
Observers
lessThan(Any)
...
End Ordered.

F-bounded polymorphism

Class Ordered[T]
Observers
lessThan(T)
. . .
End Ordered.

Class OrderedCollecton[T **Inherits** Ordered[T]]
Inherits Collection[T]
. . .
End OrderedCollection.

Typing *indices*

Persistence

- *Orthogonal*
- *Transparent*
- *Per-object* based
- *Message-passing* based
- *Transitive* (reachability)
- *Type safe*

Orthogonal persistence

Persistence in the root !

Class Any
Observers
persistent()
Mutators
persists()
Constraints
$\Box(\bigcirc \text{self.persistent}() \leftarrow \text{self.persists}())$
End Any.

Environments: Name space management

Environments

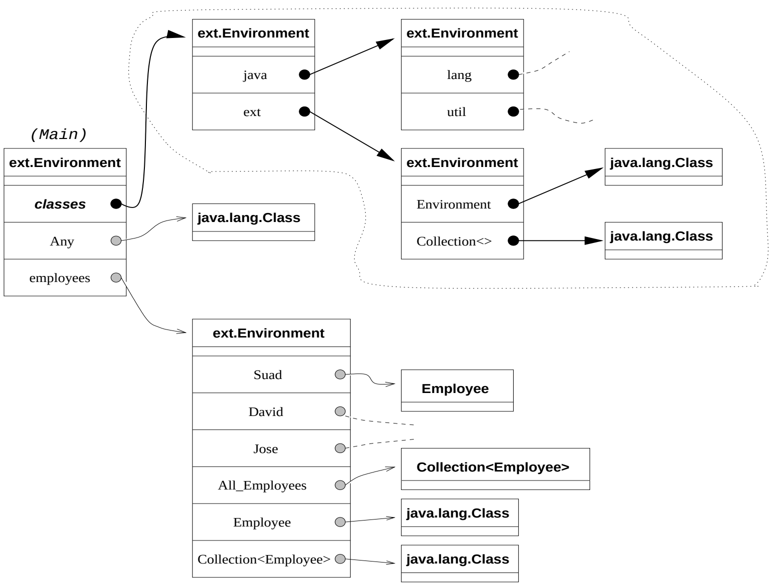
With Main
    Class ThreeDObject
    ...
    End ThreeDObject;
    Class MovingObject
    Inherits ThreeDObject
    ...
    End MovingObject;
    Environment MyEnvironment;
    With MyEnvironment
    Class Shuttle
    Inherits Moving Object
    ...
    End Shuttle;
    MyShuttle: Shuttle
    End MyEnvironment
End Main.

Persistence implementation architecture

- *PJama* - *orthogonal* and *transitive* persistence.
- Java is not extended
- *Java Virtual Machine* is extended

```
interface PJStore
{    public static PJStore getStore();
     public void newPRoot(String,Object);
     public void setPRoot(String,Object);
     public Object getPRoot(String);
}
```

Environments



Complex *name space management*

Parametric polymorphism

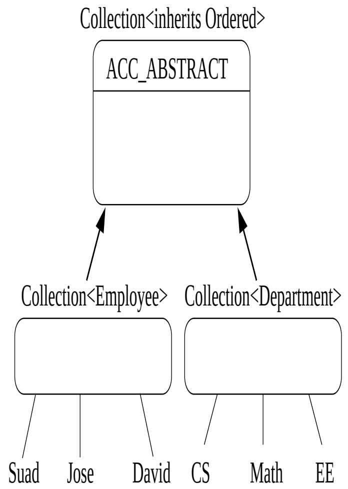
Implementation techniques

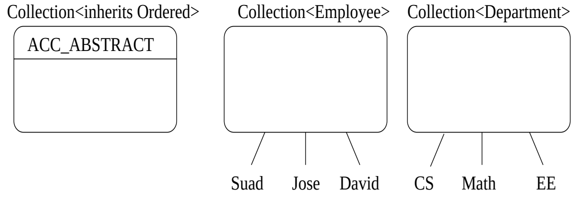
- Specialized code for each instantiated class (*heterogeneous translation*).
- Generic code shared by all instantiated classes (*homogeneous translation*).
- Generic code with tags (*load-time instantiation*).

Parametric Java classes

- Parametric *class files* ?
- *Bound types* as *type parameters*
- Our approach: *abstract classes*
- *Actual* type parameters
- *Persistence*: *Precise typing*
- Persistence: *class objects* and *class files*

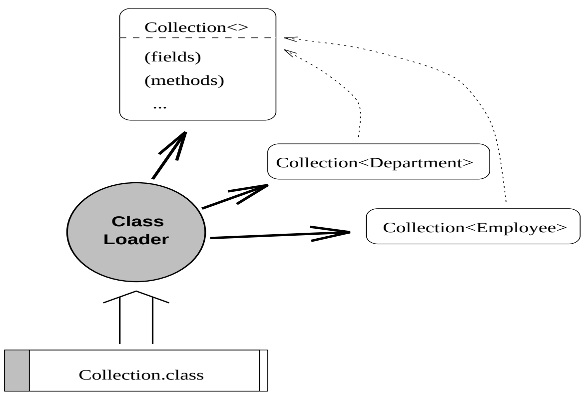
Implementing parametric polymorphism



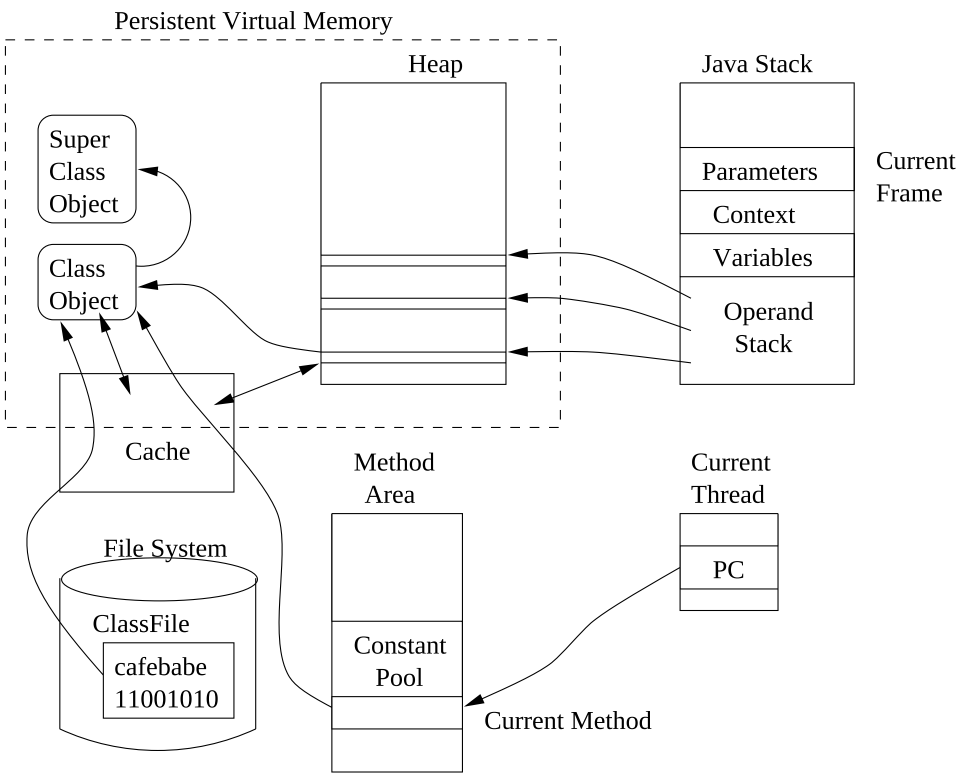


Class loader

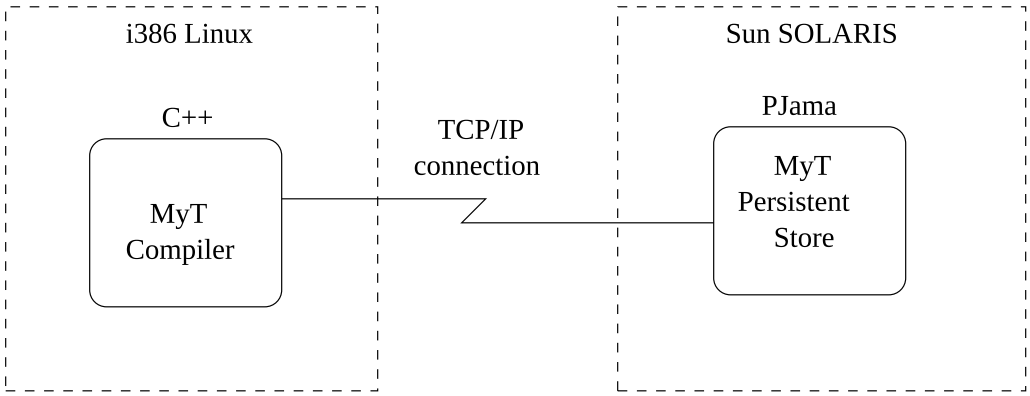
Java technology: *Extensible* class loader



System architecture



System communication



Conclusions

- *Java Virtual Machine* - *general implementation platform*
- Java technology - allows *integration* of *non-traditional languages*
- *Parametric polymorphism* - a *required extension*
- *Persistence* – an *extension* of the *Java Virtual Machine*
- *Name space* management