A Lisp Driver to be embedded in Java Applications

Taiichi Yuasa
Kyoto University

IPSJ SIG PRO workshop in October 18, 2002

Embedded Lisp Systems

- well-known examples
  - Emacs Lisp
  - AutoLisp in AutoCAD
- for compilers
  - intermediate languages and machine descriptions in S-expressions
  - can reduce development time and cost by embedding a Lisp system
  - COINS project (for compiler infrastructure)
- problems
  - application developers without Lisp implementation know-how
  - limited development time
  - difficult to customize existing systems

Lisp system for Java applications

- design and implementation principles
  - easy to add, delete, and modify functionalities even for Java programmers without implementation know-how
  - easy to add functions to handle Java components

    ⇒ implementation in Java
  - compact
  - minimum debugging facility
  - acceptable performance

Definition of predefined functions (case Bubu)

```
public static void Lcar(BCI bci) {
  Object x = bci.vs[bci.vsbase + 1];
  if (!(x instanceof List))
    throw SE.notList(x);
  bci.acc = ((List) x).car;
}
```

- need various knowledge about the implementation of Bubu
  - multi-threading
  - stack structure
  - how to pass arguments and return values
- duplicated type checking

Definition of predefined functions (our case)

```
public static Object car(List x) {
  return x.car;
}
```

```
public static Pair cons(Object x, Object y) {
  return new Pair(x, y);
}
```

```
public static Object setCar(Pair x, Object val) {
  return x.car = val;
}
```

Definition of special forms

```
(if c e1 e2)
```

```
public static
Object Lif(Object c, Object e1, Object e2, Env env) {
  if (eval(c, env) != F)
    return eval(e1, env);
  else if (e2 == null)
    return List.nil;
  else
    return eval(e2, env);
}
```

Interface to Java

- unnecessary in the Lisp level
- easier to write in Java

```
public static bitvec bitvec_and(bitvec x, bitvec y) {
  return x.bitvec_and(y);
}
```

Making system compact

- use Java mechanisms as much as possible
  - memory management and garbage collection
  - representing Lisp objects by standard Java classes
  - I/O
  - exception handling
  - reflection

Minimum debugging facility

- appropriate error messages
- backtrace

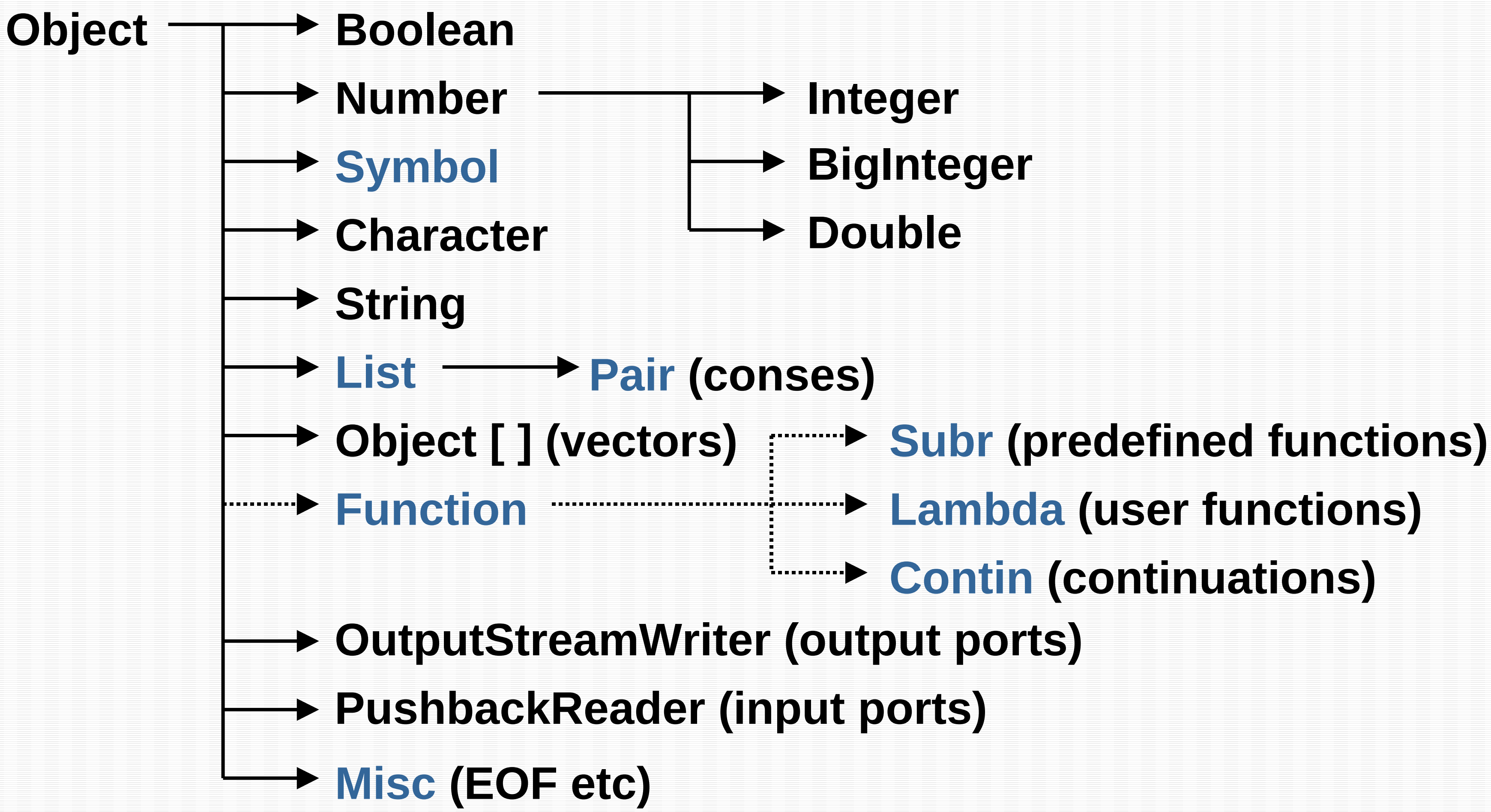
```
>(define (fact x)
   (if (zero? x)
       (/ 1 0)
       (* x (fact (- x 1)))))
>(fact 3)
ArithmeticException: / by zero
Backtrace: / < if < fact < if < fact < if < fact < if
< fact < top-level
>
```

- function trace

Language specification

- not essential, but ...
- almost full set of IEEE Scheme
  - continuations are escape procedures (i.e., non-local exits)
  - no tail-recursion optimization
  - immutable strings

Data representation



Object

Boolean

Number

Integer

BigInteger

Double

Symbol

Character

String

List

Pair (conses)

Object [ ] (vectors)

Function

Subr (predefined functions)

Lambda (user functions)

Contin (continuations)

OutputStreamWriter (output ports)

PushbackReader (input ports)

Misc (EOF etc)

Interpreter

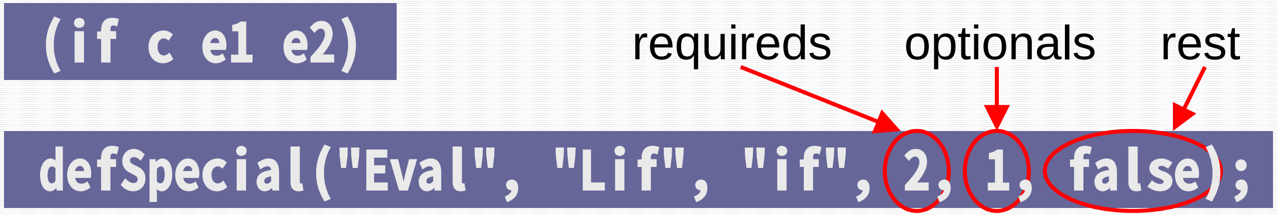
```
static Object eval(Object expr, Env env)
```

- Java programmers are not expected to modify Lisp compilers
  - implementation of the if form in Bubu

```
(define (c2if fmla form1 form2)
   (if (and (eq? (car form2) c2constant)
            (eq? *value-to-go* 'trash)
            (member *exit* '(next escape)))
       (let ((tlabel (next-label)) (flabel *exit-label*))
         (dlet ((*exit* 'next) (*exit-label* tlabel)) (cjf fmla flabel))
         (wt-label tlabel)
         (c2expr form1))
       (let ((tlabel (next-label)) (flabel (next-label)))
         (dlet ((*exit* 'next) (*exit-label* tlabel)) (cjf fmla flabel))
         (wt-label tlabel)
         (case *exit*
           ((next) (dlet ((*exit* 'escape)) (c2expr form1)))
           ((unwind-next) (dlet ((*exit* 'unwind-escape)) (c2expr form1)))
           (else (c2expr form1)))
         (wt-label flabel)
         (c2expr form2))))
```

Definition of special forms

```
(if c e1 e2)
```



requireds optionals rest

```
defSpecial("Eval", "Lif", "if", 2, 1, false);
```

```
public static
Object Lif(Object c, Object e1, Object e2, Env env) {
  if (eval(c, env) != F)
    return eval(e1, env);
  else if (e2 == null)
    return List.nil;
  else
    return eval(e2, env);
}
```

Definition of special forms

```
(begin e e1 ... en )
```

```
defSpecial("Eval", "begin", "begin", 1, 0, true);
```

```
public static Object begin(Object e, List es, Env env) {
  Object val = eval(e, env);
  while (es != List.nil) {
    val = eval(es.car, env);
    es = (List) es.cdr;
  }
  return val;
}
```

Definition of predefined functions

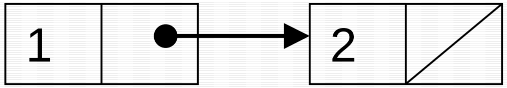
```
def("List", "car", "car", 1, 0, false);
```

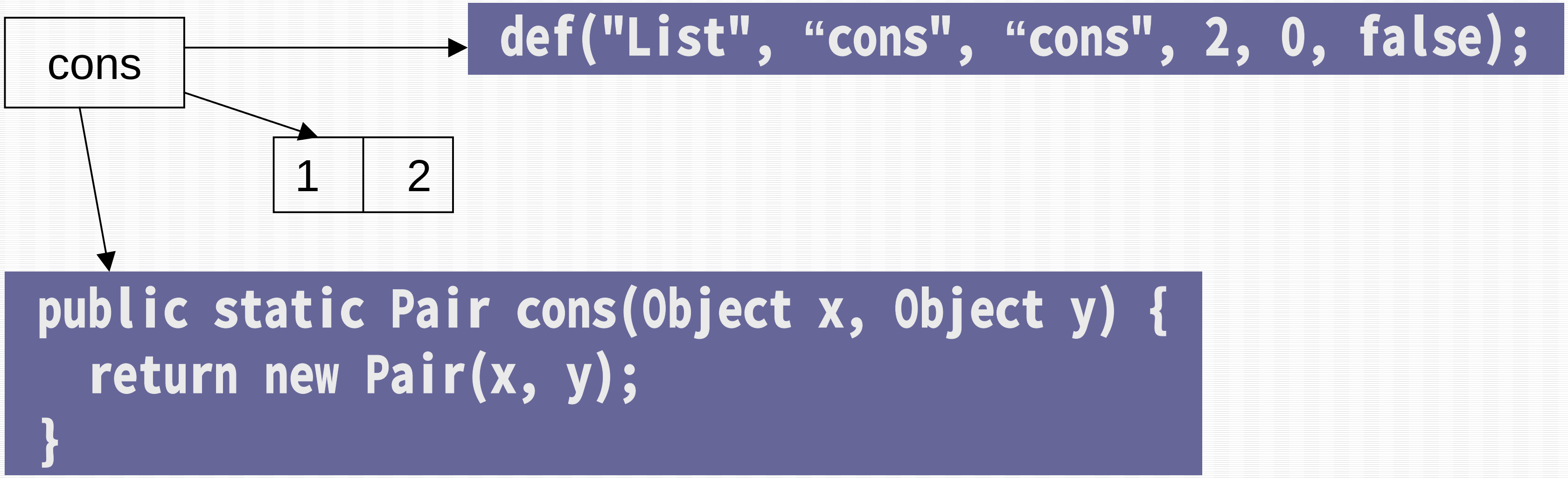
```
public static Object car(List x) {
  return x.car;
}
```

Function calls

- using evlis, avoiding use of stacks
  - ⇒ avoiding global control structures
- using Java reflections

```
(cons a b)
```





```
def("List", “cons", “cons", 2, 0, false);
```

```
public static Pair cons(Object x, Object y) {
  return new Pair(x, y);
}
```

Multi-purpose implementation methods

```
def("List", "list2vec", "list->vector", 1, 0, false)
```

```
def("List", "list2vec", "vector", 0, 0, true)
```

```
public static Object[] list2vec(List x) {
  ...
}
```

- also used internally
  - for backquote macros
  - for reading vector literals

```
>(list->vector '(1 2 3))
#(1 2 3)
>(vector 1 2 3)
#(1 2 3)
```

Error message conversion: Java→Lisp

- Errors detected by Java reflections

  IllegalArgumentException: argument type mismatch

  → RuntimeException: the first argument is not a Pair

- Errors detected by casts

  ClassCastException: List

  → RuntimeException: unexpected List object

```
public static Object nth(List x, int n) {
  while (--n >= 0)
    x = (List) x.cdr;
  return x.car;
}
```

Error message conversion: Java→Lisp

- Calling process of Object Subr.invoke(List args)

```
Store the arg list elements into an arg vector.
Throw an exception, if wrong number of args.
try {
  return method.invoke(null, argV);
} catch (InvocationTargetException e) {
  Convert the error message in e.getTargetException()
  and thow again.
} catch (IllegalArgumentException e) {
  Check arg types and throw an exception.
}
```

Backtrace

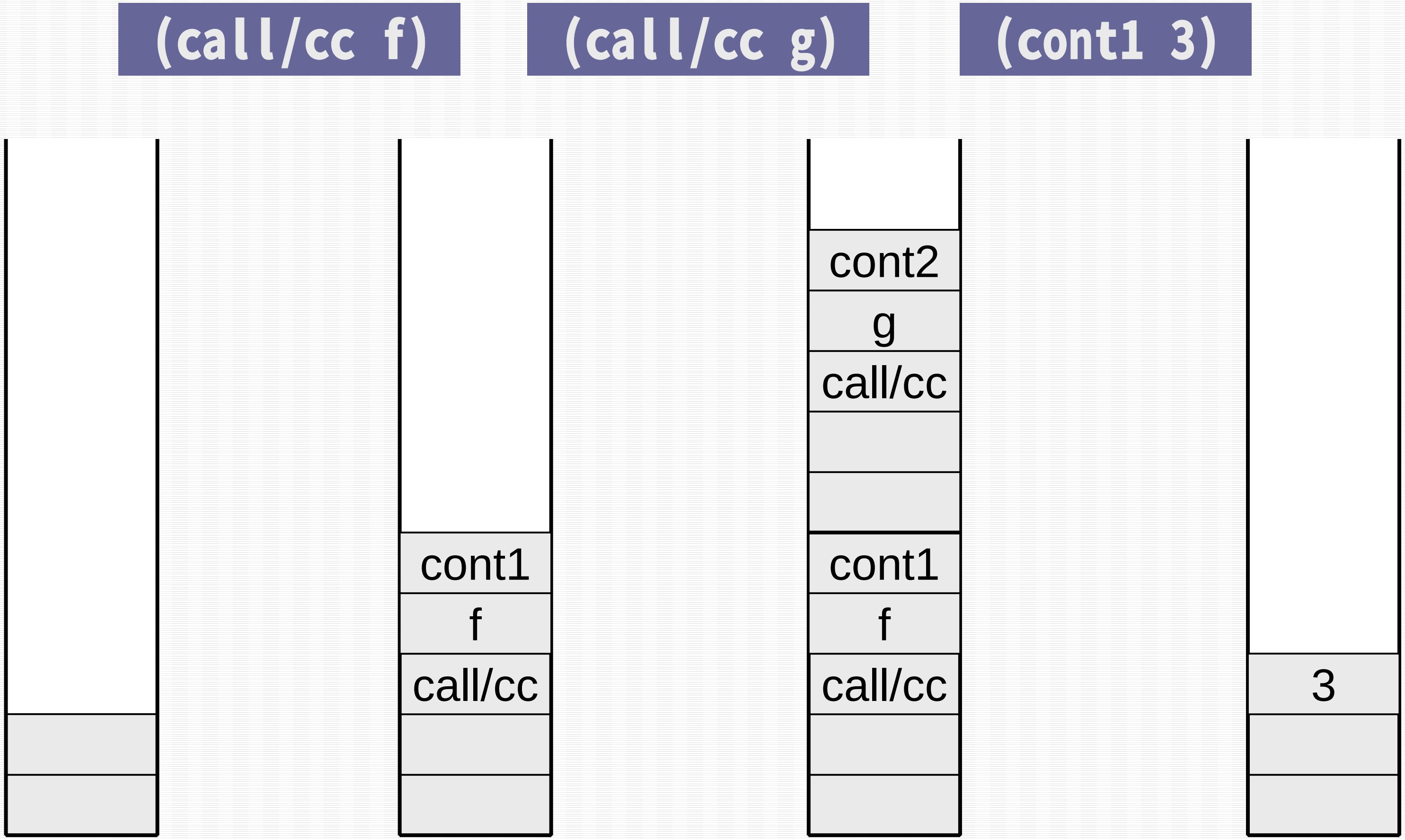
```
>(define (fact x)
   (if (zero? x)
       (/ 1 0)
       (* x (fact (- x 1)))))
>(fact 3)
ArithmeticException: / by zero
Backtrace: / < if < fact < if < fact < if < fact < if
< fact < top-level
```

Backtrace

```
try {
  Calling process
} catch (Throwable e) {
  if (e != backtraceToken) {
    Display the error message, and start backtrace output.
  } else {
    Output the caller name as part of the backtrace.
  }
  throw backtraceToken;
}
```

```
ArithmeticException: / by zero
Backtrace: /
```

Continuations (escape procedures)



(call/cc f)

(call/cc g)

(cont1 3)

cont1
f
call/cc

cont2
g
call/cc
cont1
f
call/cc

3

Continuations (escape procedures)

```
(call/cc f)
```

```
public static Object callcc(Function f) {
  Contin cont = new Contin();
  try {
    return f.invoke1(cont);
  } catch (Contin c) {
    if (c == cont)
      return c.value;
    else
      throw c;
  } finally {
    cont.canCall = false;
  }
}
```

```
try {
  Calling process
} catch (Contin c) {
  throw c;
} catch (Throwable e) {
  ...
}
```

I/O

- output port: OutputStreamWriter
  - wrapping OutputStream
  - can output Japanese characters using the specified encoding
- input port: PushbackReader
  - wrapping InputStreamReader
    - wrapping InputStream
    - can input Japanese characters using the specified encoding
  - enables character-wise unreading

I/O

- the Reader is implemented by using *character properties* and *reader macros* as in Common Lisp
  - though they cannot be altered nor added in the Lisp level
  - compact and clear code for lexical analysis
  - enables drastic changes to the Reader

I/O

- uses Java facilities for parsing numbers
  - lexical structure shared with Java
- the Printer is made compatible with the Reader
  - also uses parsing facilities of Java
  - saves printed representations of symbols and reuses them

```
try {
  return makeInt(Integer.parseInt(s));
} catch (NumberFormatException e) { }
try {
  return new BigInteger(s);
} catch (NumberFormatException e) { }
try {
  return new Double(s);
} catch (NumberFormatException e) {
  return Symbol.intern(s);
}
```

Comparison with other systems

- systems written in Java and with open sources
  - HotScheme: Scheme-based language, S-expression interpreter
  - Skij: almost IEEE, S-expression interpreter
  - Kawa: almost IEEE, compiles to JVM
  - Jscheme: almost IEEE, compiles to intermediate code
  - SISC: full R5RS, compiles to intermediate code

Comparison with other systems

- changes to special forms
  - quite hard for compiler-based Kawa, Jscheme, and SISC
  - special forms in HotScheme and Skij receive lists and check the syntax by themselves ⇒ sometimes forget checking

```
HotScheme >> (if)
error: Arg not a list:  in car of: #f
```

```
skij> (if)
SchemeException: Can't eval null
```

```
>(if)
RuntimeException: too few arguments to if
```

Comparison with other systems

- implementation of predefined functions
  - HotScheme, Skij, Kawa
    - one class for each predefined function
    - huge number of classes, huge system size
  - Jscheme, SISC
    - unique id number for each predefined function
    - dispatches at run time in a switch statement
    - discourages adding new predefined functions
- backtrace
  - HotScheme, SISC: no backtrace
  - Kawa: Java level only
  - Skij, Jscheme: more complicated than ours, but are they effective?

Comparison with other systems

- continuations
  - HotScheme: no continuations
  - Skij, Jscheme： escape procedures
    - no checking at invocation ⇒ system malfunctioning
  - Kawa: escape procedures
    - represented as a continuation object that contains an exception object, though they can be a single object
  - SISC: real continuations
- I/O
  - Kawa: based on character properties, parsing from scratch
  - Jshcme: the Reader uses parsing facilities of Java, though the Printer is not compatible with the Reader
  - HotScheme, Skij, SISC: too complicated (I gave up analyzing them)

Implementation classes and their sizes

class name	Java sources		class files
	# of lines	# of bytes	# of bytes
Char	256	7,786	6,618
Contin	61	1,307	1,516
Env	66	1,655	1,888
Eval	576	18,377	12,006
Function	10	172	257
IO	767	23,128	15,446
Lambda	61	1,479	2,259
List	449	11,666	8,353
Misc	14	192	342
Num	728	22,462	12,267
Pair	10	168	287
Subr	219	6,714	6,692
Symbol	235	6,541	6,009
total	**3,452**	**101,647**	**73,940**

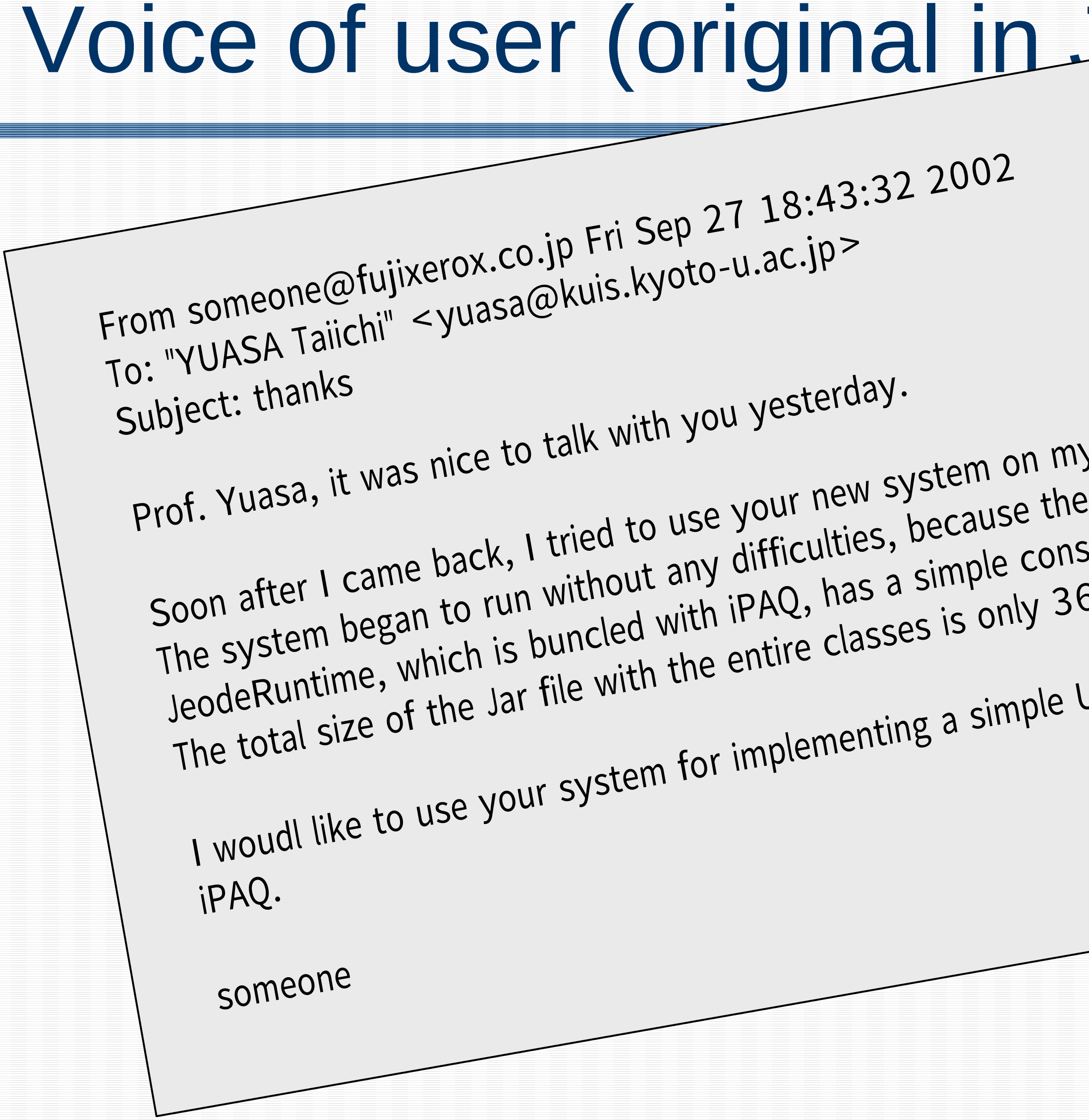
Comparison of system sizes

system	Java sources		class files		Scheme sources	
	# of files	# of bytes	# of files	# of bytes	# of files	# of bytes
our system	13	101,647	13	73,940	0	0
HotScheme	114	139,363	121	134,811	0	0
Skij	41	151,841	173	280,427	53	122,439
Kawa	135	338,927	160	429,886	18	48,582
Jscheme	51	266,338	55	228,150	42	262,642
SISC	90	408,201	96	223,001	38	381,559
Bubu	55	470,255	69	309,675	8	170.232

Benchmark results

test	our system	KCL interpreter	Kawa	Skij
ctak	15.754	1.700	8.329	16.826
deriv	8.569	1.660	2.550	6.425
div (ite)	6.505	0.890	0.829	10.587
div (rec)	8.445	1.720	0.909	8.535
fprint	0.106	0.010	0.118	0.981
fft	27.310	7.600	11.192	(error)
fread	0.851	0.030	0.506	0.059
puzzle	76.258	13.410	20.547	96.751
tak	4.399	1.486	1.021	4.172
takr	4.663	1.790	1.561	5.017
tprint	0.050	0.010	0.096	1.138
traverse (init)	69.476	15.870	20.598	132.938
traverse (run)	363.169	101.520	56.675	373.436
triangle	1067.892	230.320	219.743	1000.534

Voice of user (original in Japanese)



From someone@fujixerox.co.jp Fri Sep 27 18:43:32 2002
To: "YUASA Taiichi" <yuasa@kuis.kyoto-u.ac.jp>
Subject: thanks

Prof. Yuasa, it was nice to talk with you yesterday.

Soon after I came back, I tried to use your new system on my iPAQ.
The system began to run without any difficulties, because the JVM
JeodeRuntime, which is buncled with iPAQ, has a simple console.
The total size of the Jar file with the entire classes is only 36 KB.

I woudl like to use your system for implementing a simple UI on
iPAQ.

someone

Future plans

- Give a name to the system
- Free download from my home page
- Evaluation based on real experiences
- Another version with Common Lisp like language spec
- Use for education