Extensible Java Pre-processor

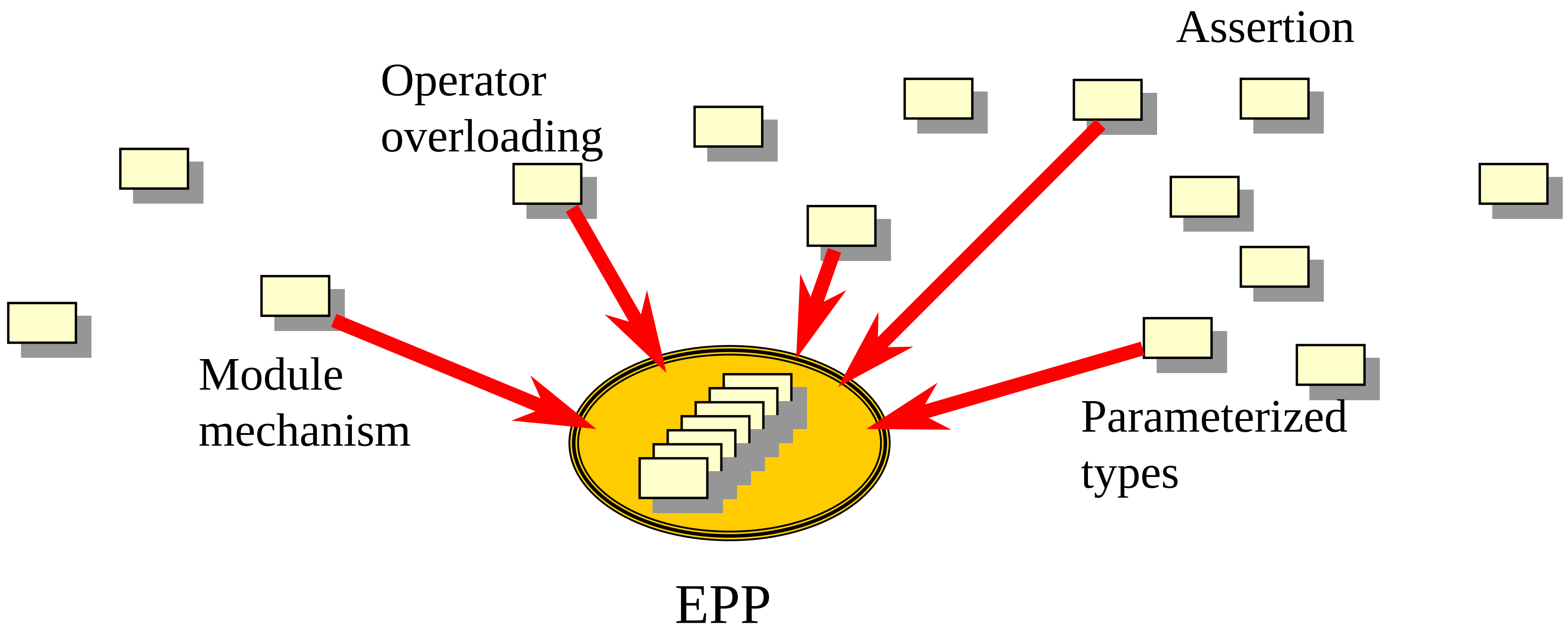
National Institute of Advanced Industrial Science and Technology(AIST)

Yuuji Ichisugi

http://staff.aist.go.jp/y-ichisugi/

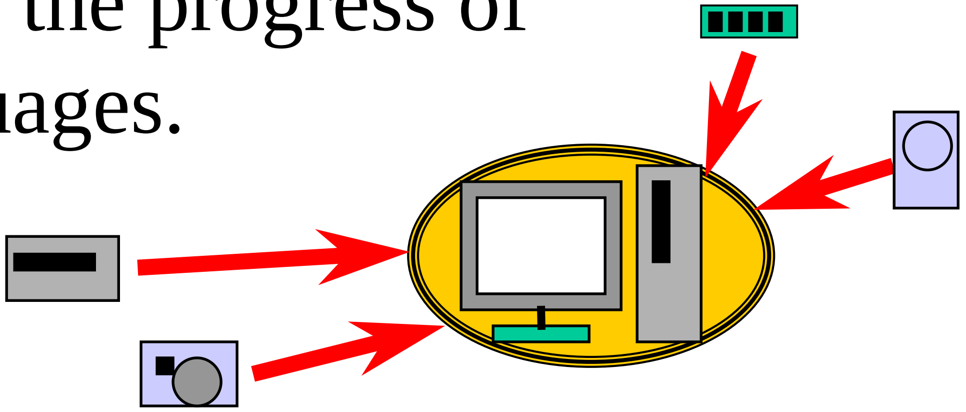
What is EPP ?

- Language extension framework which can be extended by adding “plug-ins”.



Aim: Making a programming language like a "**PC compatible machine**"

- EPP provides **the standard interface** for extension modules for language processors.
- Small venture companies or Universities can produce practical language parts.
- **Free competition**.
- EPP will accelerate the progress of programming languages.



Example : Symbol plug-in

Plug-in name used in this file

```
#epp jp.go.etl.epp.Symbol

import  jp.go.etl.epp.epp.Symbol;

public class TestSymbol {
  public static void main(String args[]){
    Symbol x = :foo;
    Symbol y = :"+";
    System.out.println(x == :foo); // true
    System.out.println(y == :foo); // false
  }
}
```

symbol literal

Translated program

```
/* Generated by EPP 1.0.2beta (by lisp-epp1.0.2beta) */
import jp.go.etl.epp.epp.Symbol;
public class TestSymbol {
  private static final Symbol _Sym0 = Symbol.intern("foo");
  private static final Symbol _Sym1 = Symbol.intern("+");
  private static final Symbol _Sym2 = Symbol.intern("foo");
  private static final Symbol _Sym3 = Symbol.intern("foo");
  public static void main(String args[]){
    Symbol x = _Sym0;
    Symbol y = _Sym1;
    System.out.println((x) == (_Sym2));
    System.out.println((y) == (_Sym3));
    }
  }
```

Actually, all line numbers are preserved by translation in order to support debugging.

Source code of Symbol plug-in

```
#epp jp.go.etl.epp.Symbol
#epp jp.go.etl.epp.SystemMixin
#epp jp.go.etl.epp.AutoSplitFiles
#epp jp.go.etl.epp.BackQuote
#epp jp.go.etl.epp.EppMacros

package jp.go.etl.epp.epp.Symbol;

defineNonTerminal(symbol, symbolOtherwise());
SystemMixin SymbolRep {
  class Epp {
    extend Tree primaryTop() {
      if (lookahead() == :":") {
        return symbol();
      } else {
        return original();
      }
    }
    extend Tree symbolTop() {
      if (lookahead() == :":") {
        matchAny();
        Token next = matchAny();
        if (next.isSymbol()){
          return new Tree(:symbol, new Identifier((Symbol)next));
        } else if (next.isLiteralToken()){
           String contents = next.literalContents();
          return new Tree(:symbol, new Identifier(Symbol.intern(contents)));
        } else {
          throw error("Symbol plug-in: "+ next+ "appeared after \":\" .");
        }
      } else {
        return original();
      }
    }
    define implement Tree symbolOtherwise(){
      throw error("symbol is required here.");
    }
    extend void initMacroTable() {
      original();
      defineMacro(:symbol, new SymbolMacro());
    }
  }
}
class SymbolMacro extends Macro {
  public Tree call(Tree tree){
    checkArgsLength(tree, 1);
    Tree[] args = tree.args();
    String str = args[0].idName().toString();
    Tree var = new Identifier(genTemp("_Sym"));

    addTree(:beginningOfClassBody,
             (decl (modifiers (id private) (id static)
                      (id final))
               (id Symbol)
               (vardecls
                (varInit ,var
                 (invokeLong (id Symbol) (id intern)
                  (argumentList ,(new LiteralTree(:string, str))))))));
    return var;
  }
}
```

Grammar extension

Macro expansion

Grammar extension part

```
#epp jp.go.etl.epp.Symbol
#epp jp.go.etl.epp.SystemMixin
#epp jp.go.etl.epp.AutoSplitFiles
#epp jp.go.etl.epp.BackQuote
#epp jp.go.etl.epp.EppMacros

...

   extend Tree primaryTop() {
      if (lookahead() == :":") {
        matchAny();
        Token next = matchAny();
        if (next.isSymbol()){
          return new Tree(:symbol, new Identifier((Symbol)next));
        } else if (next.isLiteralToken()){
          String contents = next.literalContents();
          return new Tree(:symbol,
                            new Identifier(Symbol.intern(contents)));
        } else {
          throw Epp.fatal("");
        }
      } else {
        return original();
      }
    }
```

The source code of EPP plug-ins also uses several plug-ins.

Extension of the recursive descent parser

Macro expansion part

```
class SymbolMacro extends Macro {
  public Tree call(Tree tree){
    checkArgsLength(tree, 1);
    Tree[] args = tree.args();
    String str = args[0].idName().toString();
    Tree var = new Identifier(genTemp("_Sym"));

    addTree(:beginningOfClassBody,
            `(decl (modifiers (id private) (id static)
                    (id final))
               (id Symbol)
               (vardecls
                (varInit ,var
                 (invokeLong (id Symbol) (id intern)
                  (argumentList ,(new LiteralTree(:string, str))))))));
    return var;
  }
}
```

Macro expansion using back quote macro.

Another example: Collection plug-in

```
#epp jp.go.etl.epp.Collection

public class Test {
  void test(){
    Vec<String> vec = {“aaa”, “bbb”, “ccc”};
    Table<String, int> table = {};
    foreach (index i, String s in vec){
      table.put(s, i);
    }
    int x = table.get(“bbb”) ifNull {
      throw new Error();
    };
  }
}
```

Collection types with type parameters

Type safe `foreach` statement

Table lookup with enforced null checking

Plug-ins currently implemented

- C, C++, g++ features
  - enum, defmacro, assert macro, operator overloading, optional parameters, typeof
- Lisp features
  - Symbol, back quote macro, mixin, progn
- ML features
  - parameterized types
- Perl features
  - association array
- And more…

EPP can be used as framework of ...

- Java extensions.
- Source-code analyzing tools.
  - Metrics, cross reference, ...
- Source-code translation tools.
  - refactoring, source level optimization, obfuscation, ...
- **Aim: A standard platform of Java related tools.**

Why EPP is useful as framework?

- Parser
- **Type checking**
- Abstract syntax tree manipulation libraries
- **Useful language extensions:**
  - Symbol, backquote macro, mixins, parameterized types, …
- Fully documented, with source code.

For language researchers

- Before EPP : so-called PhD languages
  - More than half a year to implement an experimental programming language system.
  - Developing tools(e.g. debuggers) are future work.
  - Only a few users.
- After EPP
  - 3 hours 〜 3 month to implement.
  - Existing developing tools can be reused.
  - 100〜10000 users can try the new language.
  - The idea becomes practical system immediately.

For programmers

- Before EPP
  - Language features **selected by the designer** to suit his own tastes are imposed.
    (No free competition !)
  - The latest research results are not adopted to the user's language processor.
- After EPP
  - Programmers can select their **favorite** features.
  - The latest research results can be used immediately.

Technical characteristics

- **Mixin based design**
  - The behavior of EPP can be extended as if "patch file" have been applied.
- **Extensible architecture**
  - extensible lexical analyzer
  - extensible recursive descent parser
  - extensible type checking mechanism
- **Simple and straightforward architechture**

Recursive Descent Parser(without mixins)

```
Tree exp() {
  Tree tree = expTop();
  while (true){
    Tree newTree = expLeft(tree);
    if (newTree == null) break;
    tree = newTree;
  }
  return tree;
}

Tree expTop() {
  if (lookahead() == :"++"){
    matchAny();
    return new Tree(:"preInc", exp());
  } else if (lookahead() == :"("){
    matchAny();
    Tree e = exp();
    match(:")");
    return new Tree(:"paren", e);
  } else {
    return expRight(exp1());
  }
}
```

```
Tree expRight(Tree tree) {
  if (lookahead() == :"+="){
    matchAny();
    return new Tree(:"+=", tree, exp());
  } else {
    return tree;
  }
}

Tree expLeft(Tree tree) {
  if (lookahead() == :"+"){
    matchAny();
    return new Tree(:"+", tree, exp1());
  } else if (lookahead() == :"++"){
    matchAny();
    return new Tree(:"postInc", tree);
  } else {
    return null;
  }
}

Tree exp1() { return term(); }
```

Extensible parser skeleton.

```
SystemMixin Exp {
  class Parser {
    define Tree exp(){
      Tree tree = expTop();
      Tree newTree;
      while (true){
        newTree = expLeft(tree);
        if (newTree == null) break;
        tree = newTree;
      }
      return tree;
    }
    define Tree expTop(){ return expRight(exp1()); }
    define Tree expRight(Tree tree) { return tree; }
    define Tree expLeft(Tree tree){ return null; }
    define Tree exp1() { return term(); }
  }
}
```

$\boxed{\text{Exp} \rightarrow \text{Term}}$

Adding a left associative operator.

$$\boxed{Exp \rightarrow Exp + Term}$$

```
SystemMixin Plus {
  class Parser {
    Tree expLeft(Tree tree) {
      if (lookahead() == :"+") {
        matchAny();
        return new Tree(:"+", tree, exp1());
      } else {
        return original(tree);
      }
    }
  }
}
```

Adding a right associative operator.

$$\text{Exp} \rightarrow \text{Term} \mathbin{\textbf{+=}} \text{Exp}$$

```
SystemMixin Assign {
  class Parser {
    Tree expRight(Tree tree) {
      if (lookahead() == :"+=") {
        matchAny();
        return new Tree(:"+=", tree, exp());
      } else {
        return original(tree);
      }
    }
  }
}
```

Application 1. Mobile agent system

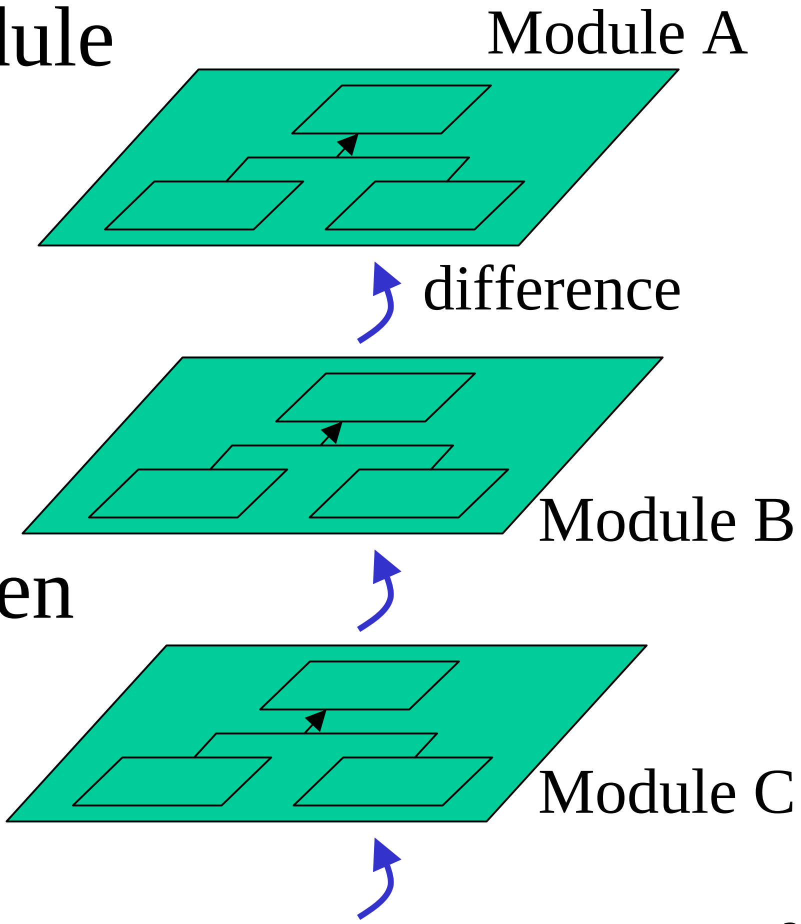
- PLANET project at Tsukuba University [Abe, Kato, Ichisugi 2000]
- Pure Java implementation of thread migration.
  - **No JavaVM modification**.
  - **Less than 1% overhead** have been achieved.
- Context saving/restoring code is added by source-code translation.

Application 2. Java metrics tool

- Enhance reliability of Java source code.
  - Code audit.
  - Code metrics.
  - Instrumentation for test coverage.
- Have become commercial product in Japan.
  - http://www.toyo.co.jp/ss/jtaster/

Application 3. Difference-based modules

- Simple and powerful module mechanism for Java. [Ichisugi ECOOP2002]
- Supports separation of crosscutting concerns.
- EPP itself will be re-written using this module mechanism.



Conclusion

- EPP is a powerful framework for Java source code processing.
- Distributed with source-code and examples.
  - http://staff.aist.go.jp/y-ichisugi/epp/