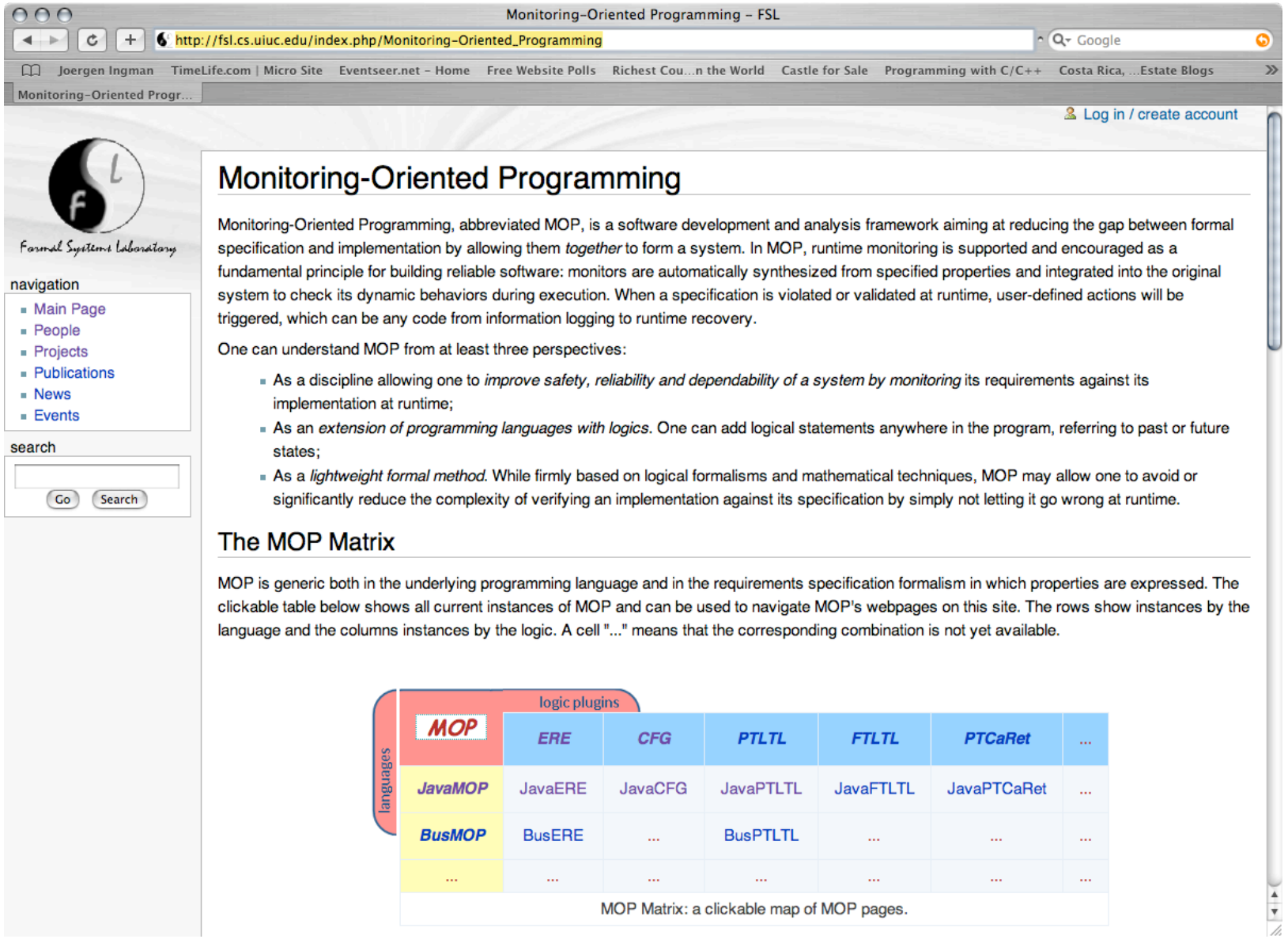
parameterized regular expressions in JavaMOP

CS 119

`“which file?”`

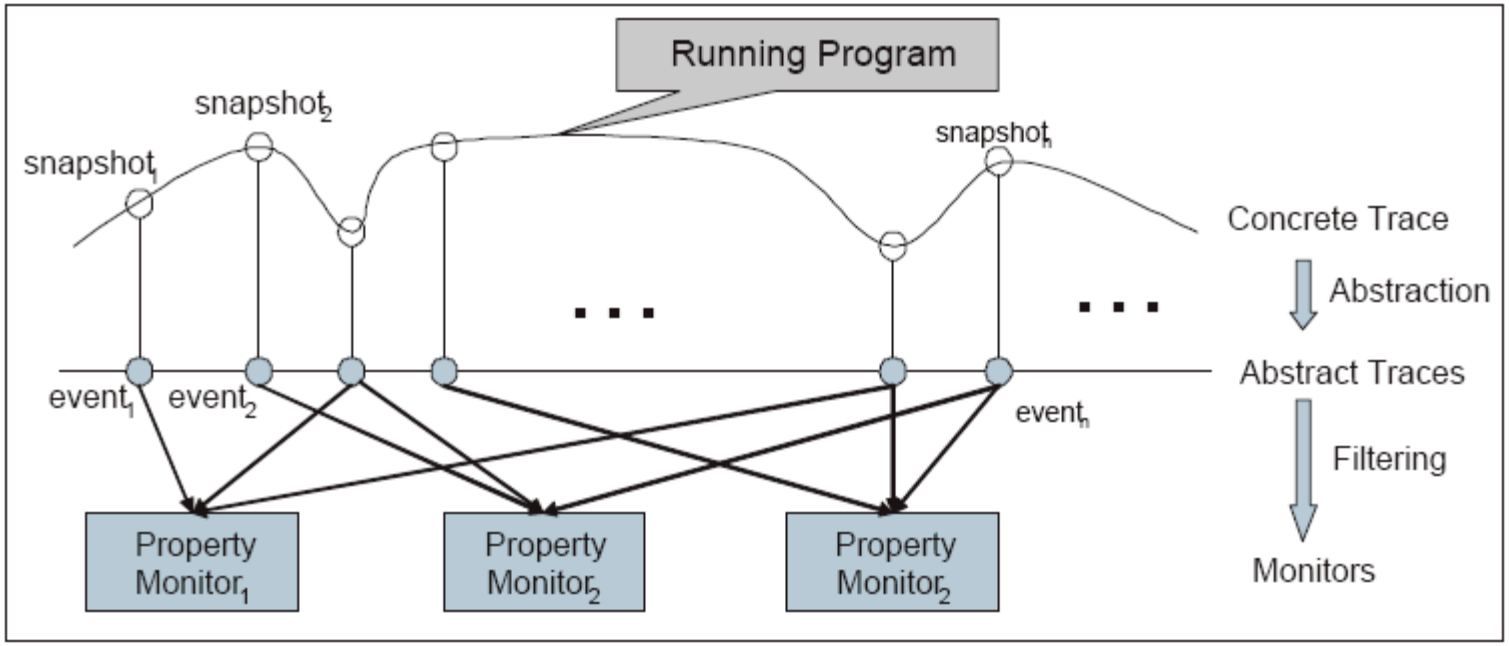
http://fsl.cs.uiuc.edu/index.php/Monitoring-Oriented_Programming



What is MOP? (their own words)

- Framework for reliable software development
  - Monitoring is basic design discipline
    - … rather than “add on” grafted onto existing code
  - Recovery allowed and encouraged
  - Provides to programmers and hides under the hood a large body of formal methods knowledge/techniques
    - Monitor synthesis algorithms
  - Generic for different languages and application domains

MOP Approach to Monitoring



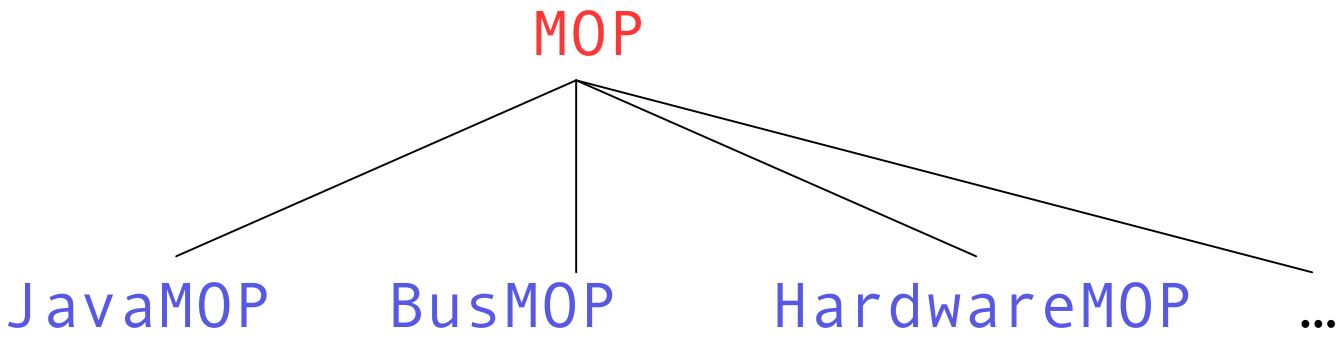
Keep the following distinct and generic:

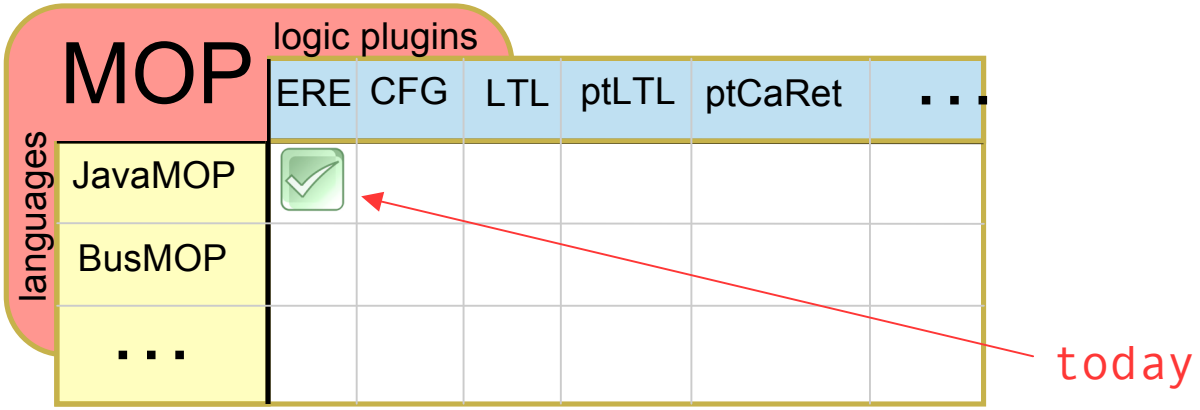
- specification formalisms
- event definitions
- validation handlers

MOP
Monitoring Oriented Programming

- Same idea as design by contract: specifications are written as comments in code. Monitors are generated from specs.
- Philosophy: no silver-bullet logic for specs
- MOP logic plugins (a subset):
  - ERE (extended regular expressions)
  - CFG (context free grammars)
  - PtLTL (Past-time LTL) and FtLTL (Future-time LTL)
  - JML (fragment of Java modeling language)
  - ATL (Allen temporal logic)
  - Jass (The CSP Process algebra)
- Generic wrt. parameters
  - Provide a plugin for a propositional logic, and MOP does the rest wrt. data parameterization.
  - Makes designing a new logic extremely easy compared to other frameworks.

Instances of MOP





JavaMOP Operation

```
1  package paramere.hasnext;
2
3  import java.util.*;
4
5  public class Test {
6      public static void main(String[] args) {
7          Vector<Integer> v1 = new Vector();
8          Vector<Integer> v2 = new Vector();
9          v1.add(1); v1.add(2);
10         Iterator it1 = v1.iterator();
11         Iterator it2 = v2.iterator();
12         while(it1.hasNext()) {
13             int sum = (Integer)it1.next() + (Integer)it1.next();
14             System.out.println("sum = " + sum);
15         }
16     }
17 }
18
```

```
package mop;

/*@
sync
scope = class(java.util.Iterator)
logic = ERE
HasNext
{
      event hasnext : end(exec(* hasNext()));
      event next : begin(exec(* next()));
      formula : (hasnext hasnext* next)*
}
Violation handler{
      @RESET;
}
@*/
```

under the hood

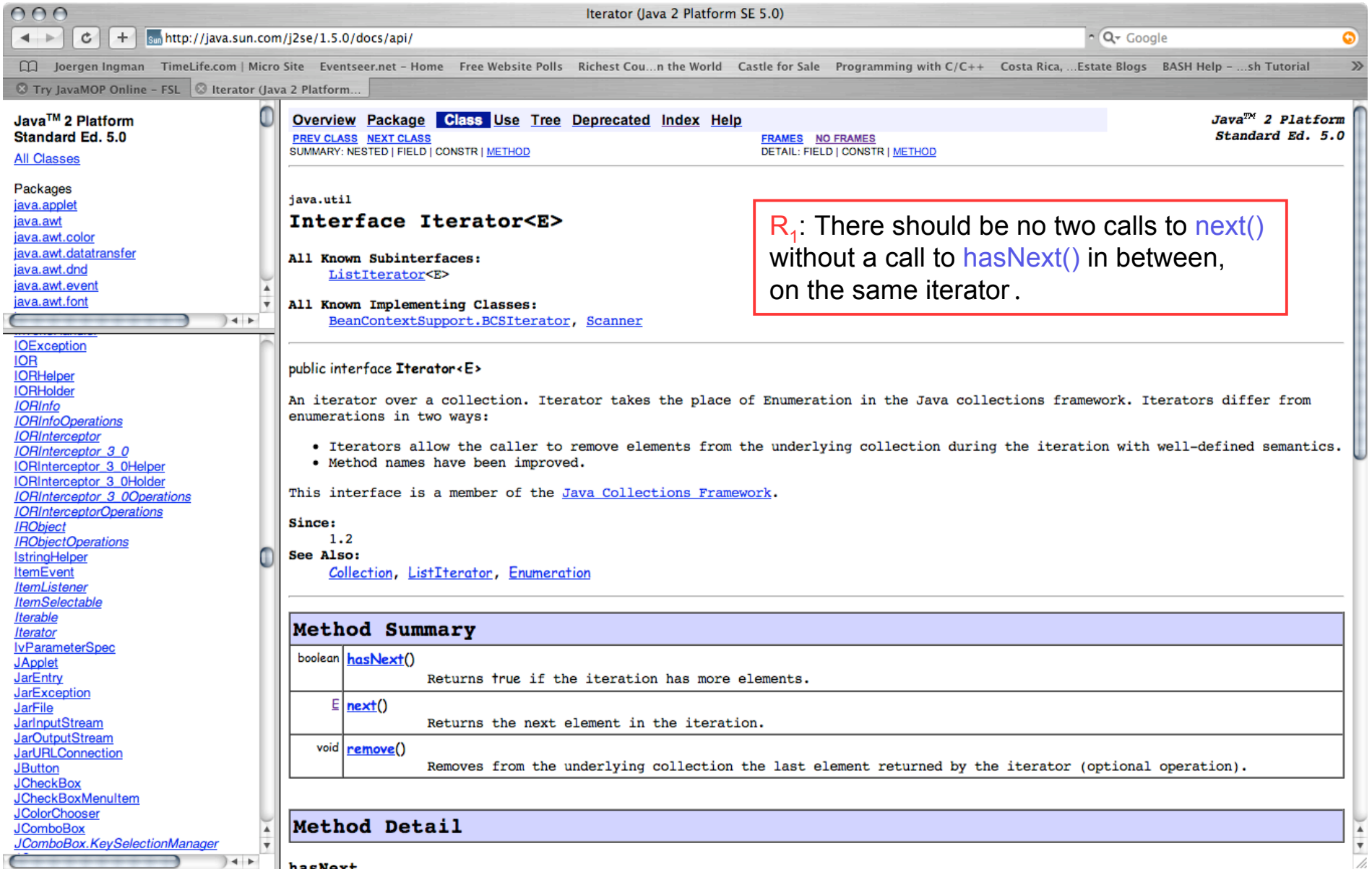
```
64 public aspect MonitorAspect {
65     Map makeMap(Object key){
66         if (key instanceof String)
67         return new HashMap();
68         else
69         return new ReferenceIdentityMap(AbstractReferenc
70     }
71     List makeList(){
72         return new ArrayList();
73     }
74     HasNextMonitor Iterator.HasNext_it_M = null;
75     pointcut HasNext_hasnext0(Iterator it) :  call(* Ite
76     after (Iterator it) : HasNext_hasnext0(it) {
77         System.out.println("--- hasNext");
```

AspectJ

JavaMOP

java VM

properties of Java library APIs



java.util

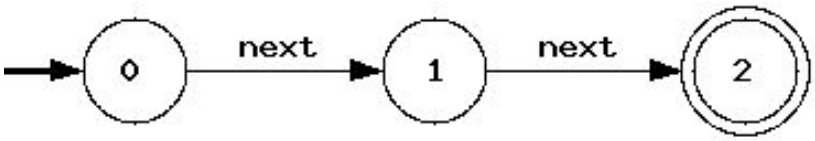
Interface Iterator<E>

All Known Subinterfaces:
ListIterator<E>

All Known Implementing Classes:
BeanContextSupport.BCSIterator, Scanner

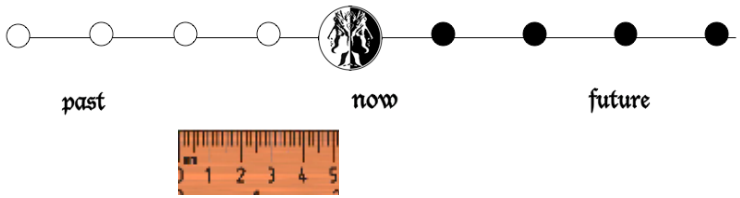
R_1: There should be no two calls to next() without a call to hasNext() in between, on the same iterator .

next next



i) suffix trace semantics

ii) looking for validation



Property in JavaMOP

PROPOSITIONAL!

```
/*@
partial
scope = global
logic = ERE

HasNext
{
   event hasnext : end(call(* Iterator.hasNext()));
   event next    : begin(call(* Iterator.next()));
   formula       : next next
}

validation handler{
   System.err.println("*** call hasNext() before next()");
}
@*/
```

check trace suffix

track all events

logic to be used

events

formula to be checked

code to be executed if formula becomes validated

The propositional property does not flag the following error

```
class Test {
  public static void main(String[] args) {
    Vector<Integer> v1 = new Vector();
    Vector<Integer> v2 = new Vector();
    v1.add(1); v1.add(3); v2.add(5); v2.add(7);
    Iterator it1 = v1.iterator();
    Iterator it2 = v2.iterator();
    int sum = 0;
    if(it1.hasNext())
      sum += (Integer)it2.next();
    if(it1.hasNext())
      sum += (Integer)it2.next();
    System.out.println("sum(v2) = " + sum);
  }
}
```

should have been:
if(it2.hasNext())

⇓

unguarded calls:
it2.next()

execution emits: hasNext() next() hasNext() next()

spec "next next" does not match and hence no error detected

Improved Property in JavaMOP

PARAMETERIZED!

```
/*@
partial centralized
scope = global
logic = ERE

HasNext(Iterator i)
{
   event hasnext<i> : end(call(* i.hasNext()));
   event next<i>    : begin(call(* i.next()));
   formula          : next next
}

validation handler{
   System.err.println("*** call hasNext() before next()");
}
@*/
```

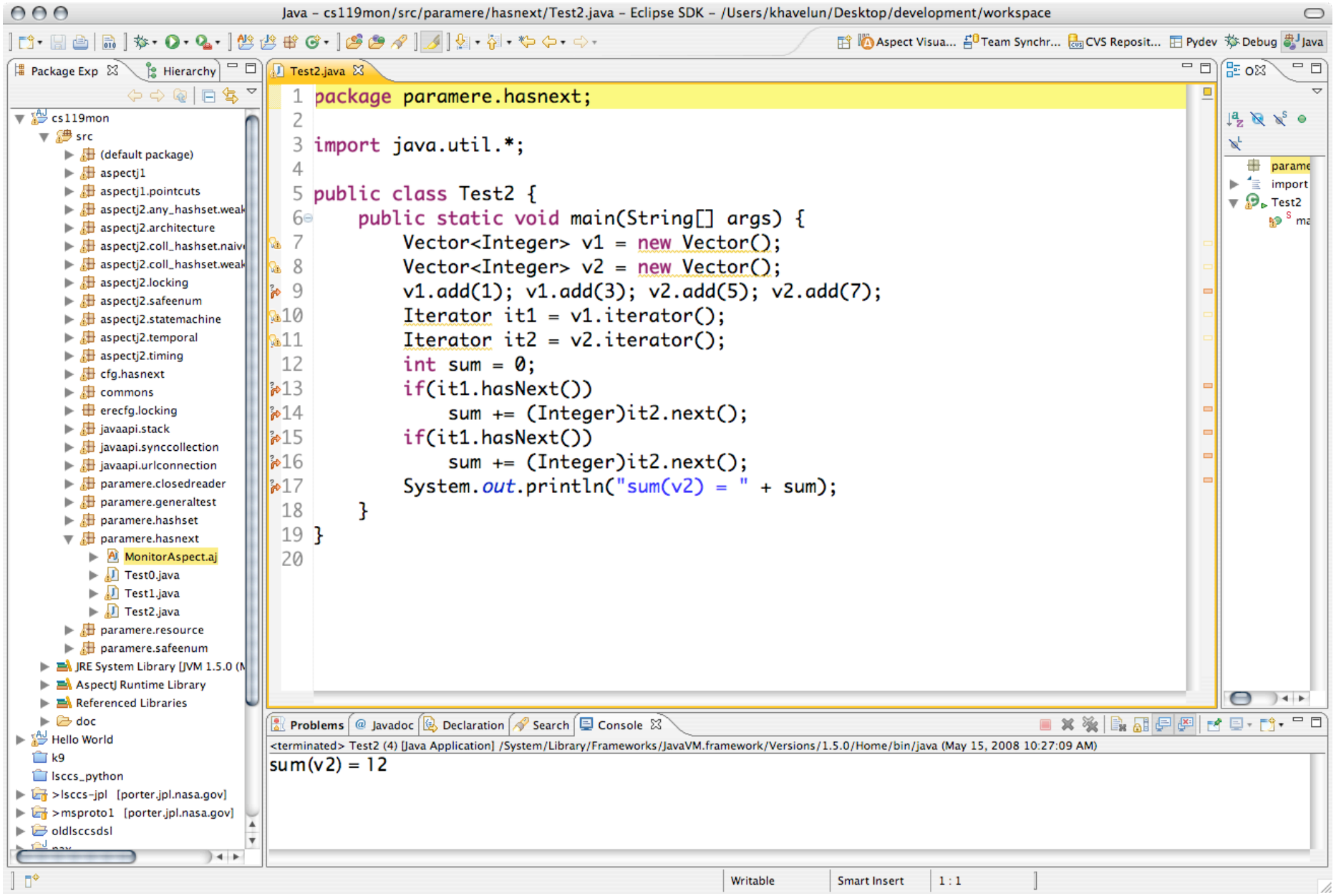
centralized tracking of values. We will get to this

parameterized with iterator

events refer to parameter

this causes the handler to be invoked

DEMO ON SLIDES


Java – cs119mon/src/paramere/hasnext/Test2.java – Eclipse SDK – /Users/khavelun/Desktop/development/workspace
Package Exp
Hierarchy
Test2.java
cs119mon
src
(default package)
aspectj1
aspectj1.pointcuts
aspectj2.architecture
aspectj2.locking
aspectj2.safeenum
aspectj2.statemachine
aspectj2.temporal
aspectj2.timing
cfg.hasnext
commons
javaapi.stack
javaapi.synccollection
javaapi.urlconnection
paramere.closedreader
paramere.generaltest
paramere.hashset
paramere.hasnext
MonitorAspect.aj
Test0.java
Test1.java
Test2.java
paramere.resource
paramere.safeenum
JRE System Library [JVM 1.5.0
AspectJ Runtime Library
Referenced Libraries
doc
Hello World
k9
lsccs_python
>lsccs-jpl [porter.jpl.nasa.gov]
>msproto1 [porter.jpl.nasa.gov]
oldlsccsdsl
1 package paramere.hasnext;
2
3 import java.util.*;
4
5 public class Test2 {
6 public static void main(String[] args) {
7 Vector<Integer> v1 = new Vector();
8 Vector<Integer> v2 = new Vector();
9 v1.add(1); v1.add(3); v2.add(5); v2.add(7);
10 Iterator it1 = v1.iterator();
11 Iterator it2 = v2.iterator();
12 int sum = 0;
13 if(it1.hasNext())
14 sum += (Integer)it2.next();
15 if(it1.hasNext())
16 sum += (Integer)it2.next();
17 System.out.println("sum(v2) = " + sum);
18 }
19 }
20
Problems
Javadoc
Declaration
Search
Console
<terminated> Test2 (4) [Java Application] /System/Library/Frameworks/JavaVM.framework/Versions/1.5.0/Home/bin/java (May 15, 2008 10:27:09 AM)
sum(v2) = 12
Writable
Smart Insert
1 : 1

Monitoring-Oriented Programming - FSL

http://fsl.cs.uiuc.edu/index.php/Monitoring-Oriented_Programming

Joergen Ingman | TimeLife.com | Micro Site | Eventseer.net - Home | Free Website Polls | Richest Cou...n the World | Castle for Sale | Programming with C/C++

Monitoring-Oriented Progr...

Log in / create account

Formal Systems Laboratory

navigation

- Main Page
- People
- Projects
- Publications
- News
- Events

search

Go Search

Monitoring-Oriented Programming

Monitoring-Oriented Programming, abbreviated MOP, is a software development and analysis framework aiming at reducing the gap between formal specification and implementation by allowing them *together* to form a system. In MOP, runtime monitoring is supported and encouraged as a fundamental principle for building reliable software: monitors are automatically synthesized from specified properties and integrated into the original system to check its dynamic behaviors during execution. When a specification is violated or validated at runtime, user-defined actions will be triggered, which can be any code from information logging to runtime recovery.

One can understand MOP from at least three perspectives:

- As a discipline allowing one to *improve safety, reliability and dependability of a system by monitoring* its requirements against its implementation at runtime;
- As an *extension of programming languages with logics*. One can add logical statements anywhere in the program, referring to past or future states;
- As a *lightweight formal method*. While firmly based on logical formalisms and mathematical techniques, MOP may allow one to avoid or significantly reduce the complexity of verifying an implementation against its specification by simply not letting it go wrong at runtime.

The MOP Matrix

MOP is generic both in the underlying programming language and in the requirements specification formalism in which properties are expressed. The clickable table below shows all current instances of MOP and can be used to navigate MOP's webpages on this site. The rows show instances by the language and the columns instances by the logic. A cell "..." means that the corresponding combination is not yet available.

languages \ logic plugins	ERE	CFG	PTLTL	FTLTL	PTCaRet	...
JavaMOP	JavaERE	JavaCFG	JavaPTLTL	JavaFTLTL	JavaPTCaRet	...
BusMOP	BusERE	...	BusPTLTL	...	...	...
...	...	...	...	...	...	...

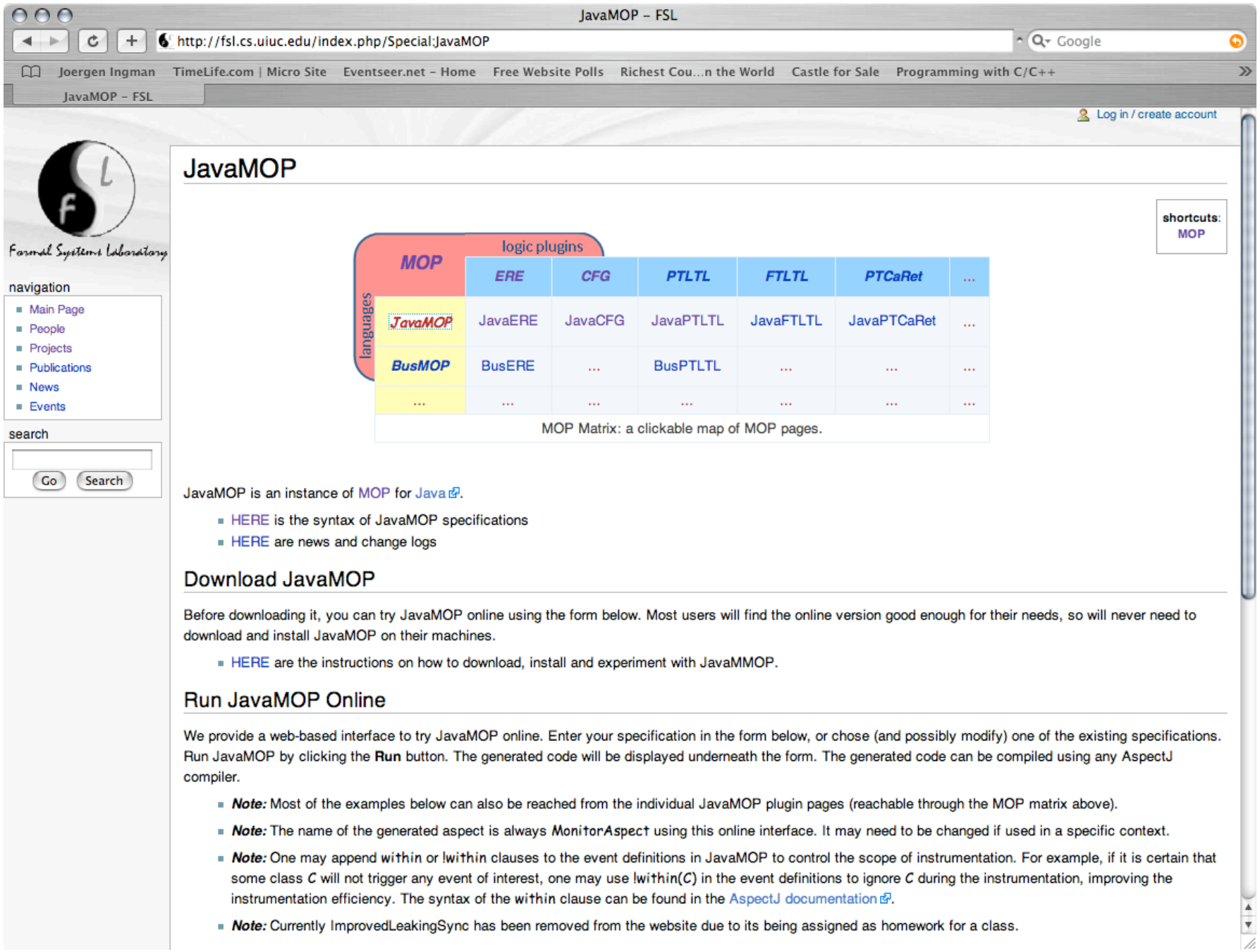
MOP Matrix: a clickable map of MOP pages.

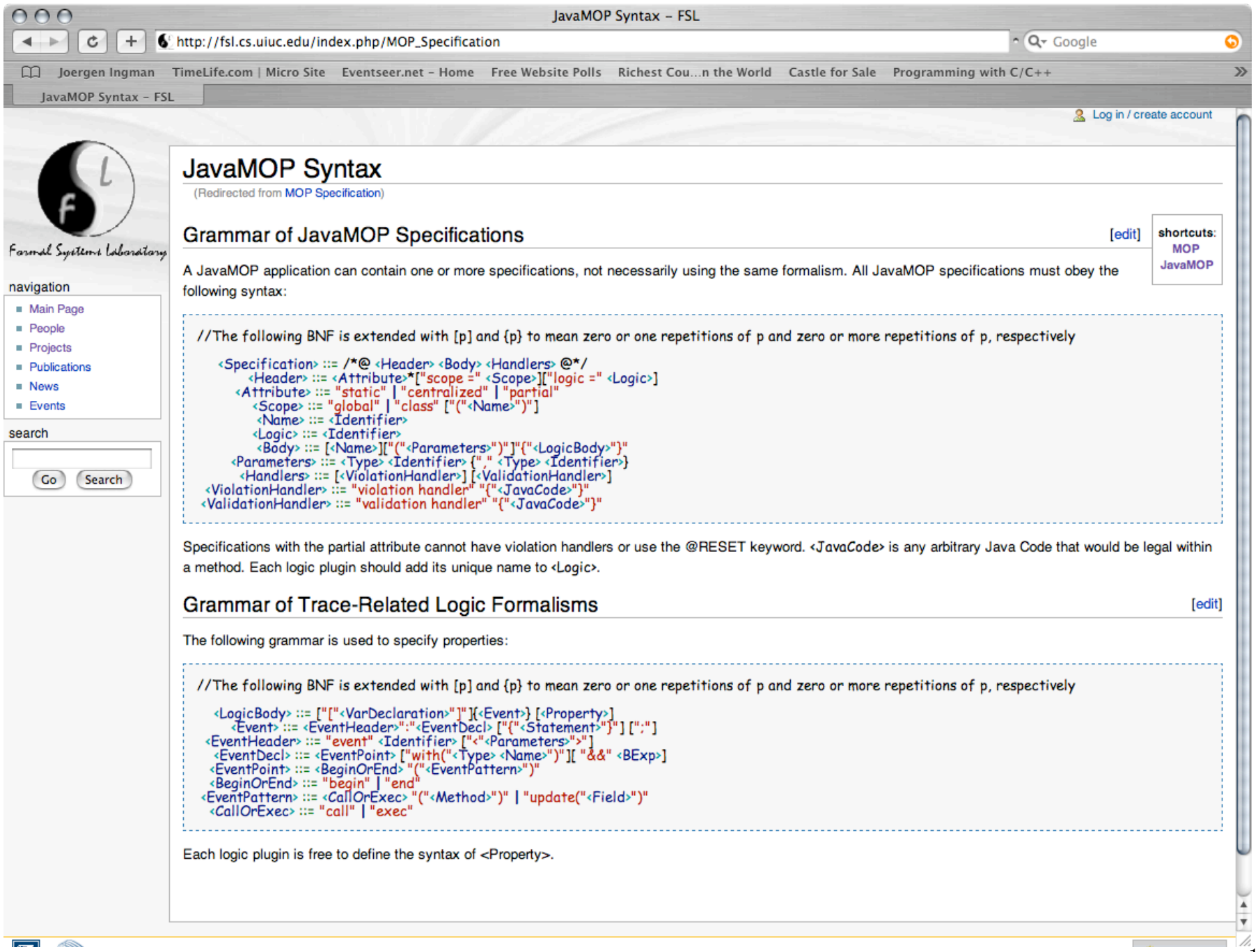
MOP Language Instances

- ***JavaMOP*** is an MOP tool for Java;
- ***BusMOP*** is an MOP tool for monitoring consumer off the shelf components over a bus;

MOP Logic Plugins

Logic plugins implement and encapsulate monitor synthesis algorithms for particular requirements specification formalisms. While they are a core feature of MOP,


JavaMOP – FSL
http://fsl.cs.uiuc.edu/index.php/Special:JavaMOP
Google
Joergen Ingman
TimeLife.com | Micro Site
Eventseer.net – Home
Free Website Polls
Richest Cou...n the World
Castle for Sale
Programming with C/C++
JavaMOP – FSL
Log in / create account
Formal Systems Laboratory
navigation
Main Page
People
Projects
Publications
News
Events
search
Go
Search
JavaMOP
shortcuts:
MOP
logic plugins
MOP
languages
ERE
CFG
PTLTL
FTLTL
PTCaRet
...
JavaMOP
JavaERE
JavaCFG
JavaPTLTL
JavaFTLTL
JavaPTCaRet
...
BusMOP
BusERE
...
BusPTLTL
...
...
...
...
...
...
...
...
...
...
MOP Matrix: a clickable map of MOP pages.
JavaMOP is an instance of MOP for Java.
HERE is the syntax of JavaMOP specifications
HERE are news and change logs
Download JavaMOP
Before downloading it, you can try JavaMOP online using the form below. Most users will find the online version good enough for their needs, so will never need to download and install JavaMOP on their machines.
HERE are the instructions on how to download, install and experiment with JavaMMOP.
Run JavaMOP Online
We provide a web-based interface to try JavaMOP online. Enter your specification in the form below, or chose (and possibly modify) one of the existing specifications. Run JavaMOP by clicking the Run button. The generated code will be displayed underneath the form. The generated code can be compiled using any AspectJ compiler.
Note: Most of the examples below can also be reached from the individual JavaMOP plugin pages (reachable through the MOP matrix above).
Note: The name of the generated aspect is always MonitorAspect using this online interface. It may need to be changed if used in a specific context.
Note: One may append within or !within clauses to the event definitions in JavaMOP to control the scope of instrumentation. For example, if it is certain that some class C will not trigger any event of interest, one may use !within(C) in the event definitions to ignore C during the instrumentation, improving the instrumentation efficiency. The syntax of the within clause can be found in the AspectJ documentation.
Note: Currently ImprovedLeakingSync has been removed from the website due to its being assigned as homework for a class.


JavaMOP Syntax - FSL
http://fsl.cs.uiuc.edu/index.php/MOP_Specification
Google
Joergen Ingman
TimeLife.com | Micro Site
Eventseer.net - Home
Free Website Polls
Richest Cou...n the World
Castle for Sale
Programming with C/C++
JavaMOP Syntax - FSL
Log in / create account
Formal System Laboratory
navigation
Main Page
People
Projects
Publications
News
Events
search
Go
Search
JavaMOP Syntax
(Redirected from MOP Specification)
Grammar of JavaMOP Specifications
[edit]
shortcuts:
MOP
JavaMOP
A JavaMOP application can contain one or more specifications, not necessarily using the same formalism. All JavaMOP specifications must obey the following syntax:
//The following BNF is extended with [p] and {p} to mean zero or one repetitions of p and zero or more repetitions of p, respectively
<Specification> ::= /*@ <Header> <Body> <Handlers> @*/
<Header> ::= <Attribute>*["scope =" <Scope>]["logic =" <Logic>]
<Attribute> ::= "static" | "centralized" | "partial"
<Scope> ::= "global" | "class" ["("<Name>")"]
<Name> ::= <Identifier>
<Logic> ::= <Identifier>
<Body> ::= [<Name>]["("<Parameters>")"]"{"<LogicBody>"}"
<Parameters> ::= <Type> <Identifier> {"," <Type> <Identifier>}
<Handlers> ::= [<ViolationHandler>] [<ValidationHandler>]
<ViolationHandler> ::= "violation handler" "{"<JavaCode>"}"
<ValidationHandler> ::= "validation handler" "{"<JavaCode>"}"
Specifications with the partial attribute cannot have violation handlers or use the @RESET keyword. <JavaCode> is any arbitrary Java Code that would be legal within a method. Each logic plugin should add its unique name to <Logic>.
Grammar of Trace-Related Logic Formalisms
[edit]
The following grammar is used to specify properties:
//The following BNF is extended with [p] and {p} to mean zero or one repetitions of p and zero or more repetitions of p, respectively
<LogicBody> ::= ["["<VarDeclaration>"]"]{<Event>} [<Property>]
<Event> ::= <EventHeader>":"<EventDecl> ["{"<Statement>"}"] [";"]
<EventHeader> ::= "event" <Identifier> ["<"<Parameters>">"]
<EventDecl> ::= <EventPoint> ["with("<Type> <Name>")"]["&&" <BExp>]
<EventPoint> ::= <BeginOrEnd> "("<EventPattern>")"
<BeginOrEnd> ::= "begin" | "end"
<EventPattern> ::= <CallOrExec> "("<Method>")" | "update("<Field>")"
<CallOrExec> ::= "call" | "exec"
Each logic plugin is free to define the syntax of <Property>.

MOP Extended Regular Expression (ERE) Plugin – FSL

http://fsl.cs.uiuc.edu/index.php/Special:EREPlugin

Google

Joergen Ingman | TimeLife.com | Micro Site | Eventseer.net – Home | Free Website Polls | Richest Cou...n the World | Castle for Sale | Programming with C/C++

MOP Extended Regular Exp...

Log in / create account

Formal Systems Laboratory

navigation

- Main Page
- People
- Projects
- Publications
- News
- Events

search

Go Search

MOP Extended Regular Expression (ERE) Plugin

shortcuts: MOP

MOP (languages \ logic plugins)	ERE	CFG	PTLTL	FTLTL	PTCaRet	...
JavaMOP	JavaERE	JavaCFG	JavaPTLTL	JavaFTLTL	JavaPTCaRet	...
BusMOP	BusERE	...	BusPTLTL	...	...	...
...	...	...	...	...	...	...

MOP Matrix: a clickable map of MOP pages.

This page discusses the MOP extended regular expression (ERE) plugin. It also allows to type in an ERE specification and then generate and visualize the state machine used to monitor the ERE pattern. Regular expressions can be easily understood by ordinary software engineers and programmers, as shown by the immense interest in and the success of scripting languages like Perl, based essentially on regular expression pattern matching. We believe that regular expressions provide an elegant and powerful specification language also for monitoring requirements, because an execution trace of a program is in fact a string of states. Extended regular expressions (EREs) add complementation to regular expressions, which brings additional benefits by allowing one to specify patterns that must not occur during an execution. Complementation gives one the power to express patterns on strings non-elementarily more compactly.

Input

The ERE formula follows the following syntax, which is defined in BNF:

```
<ERE Plugin>  ::= <Event Header> "pattern:" <ERE Pattern>
<ERE Pattern> ::= "empty" | "epsilon" | <Event Name>
                | "~" <ERE Pattern> | <ERE Pattern> "*"
                | <ERE Pattern> "+" <ERE Pattern>
                | <ERE Pattern> <ERE Pattern>
                | <ERE Pattern> "&" <ERE Pattern>
```

where epsilon corresponds to the language containing only the empty word, empty to the empty language, and <Event Name> to the language containing only the named event as a one-letter word. The different operators in increasing order of precedence are ~_ , _* , __, _&_, _|_ and have the following interpretation:

- ~ P contains all the words except for those accepted by the pattern P.
- P * contains zero or more occurrences of words accepted by the pattern P.
- P + P' is the union of words accepted by P and P'.
- P P' concatenates words accepted by P and those accepted by P'.

MOP Extended Regular Expression (ERE) Plugin - FSL

http://fsl.cs.uiuc.edu/index.php/Special:EREPlugin

Joergen Ingman | TimeLife.com | Micro Site | Eventseer.net - Home | Free Website Polls | Richest Cou...n the World | Castle for Sale | Programming with C/C++

MOP Extended Regular Exp...

Download the ERE plugin

- MOP_ERE_Plugin.zip

Run the EREPlugin Online

Choose an example:

- FileOperations
- HasNext
- HashSet
- closedReader
- closedWriter
- ex0
- ex1
- fun

HasNext

```
event hasnext
event next
pattern: (hasnext hasnext* next)*
```

Run Reset

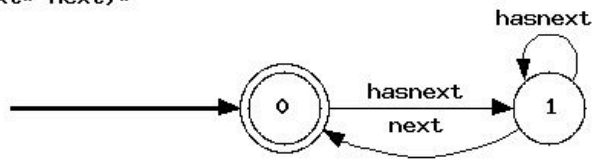
Done.

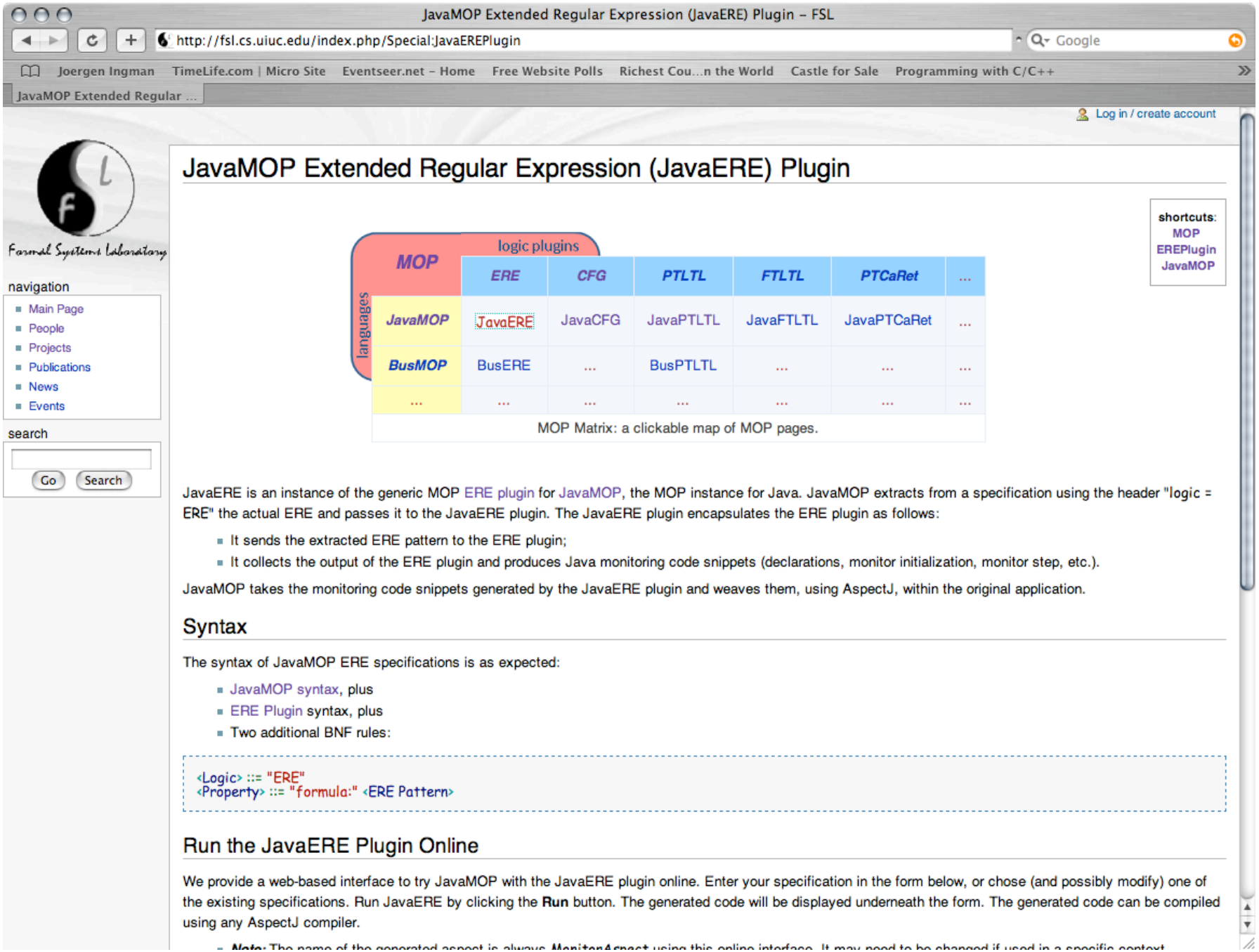
Text Output:

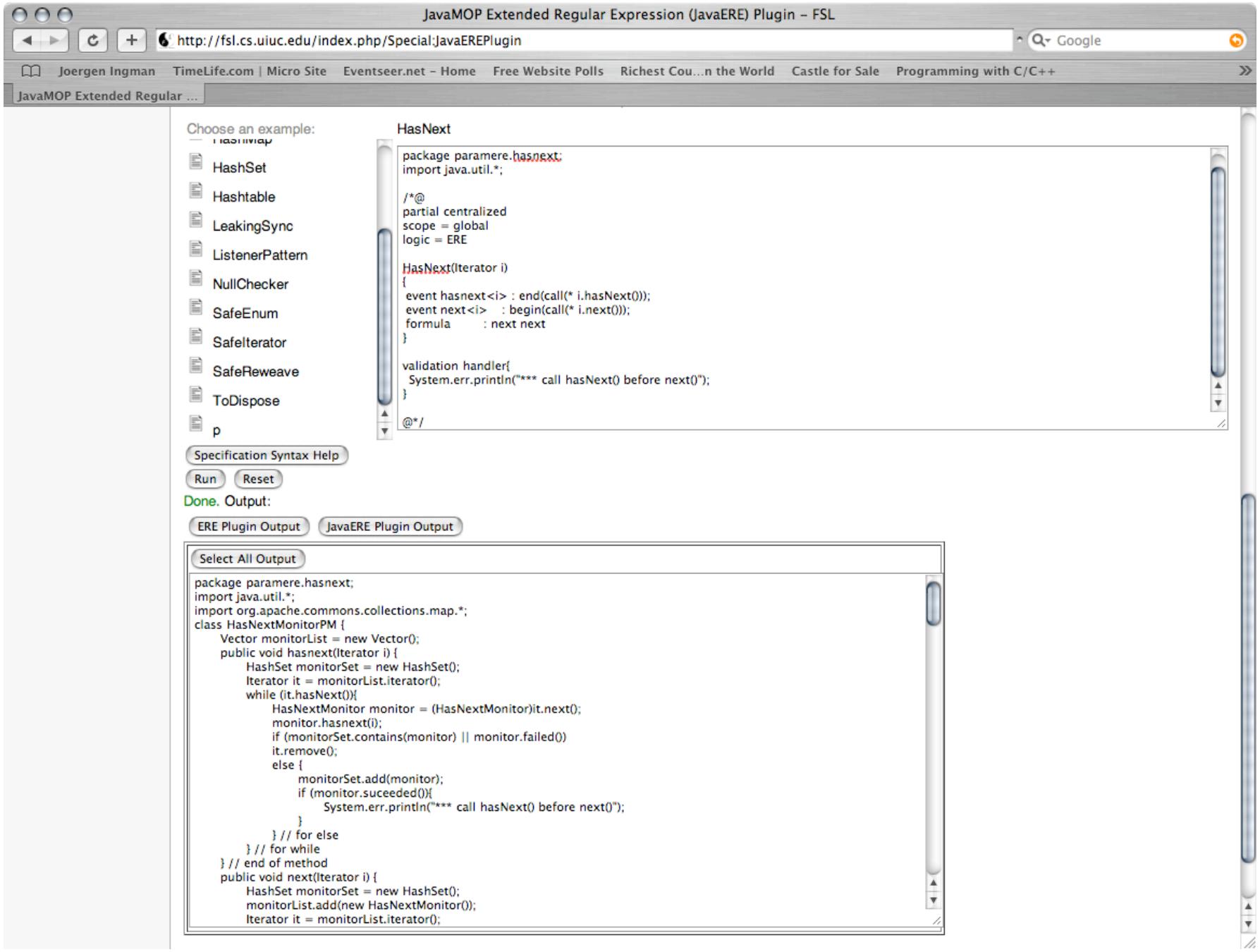
```
Starting state: 0
Accepting states: 0
Transitions:
        0 -> 1 [ label = "hasnext" ];
        1 -> 1 [ label = "hasnext" ];
        1 -> 0 [ label = "next" ];
```

Image representation:

(hasnext hasnext* next)*




JavaMOP Extended Regular Expression (JavaERE) Plugin - FSL
http://fsl.cs.uiuc.edu/index.php/Special:JavaEREPlugin
Google
Joergen Ingman
TimeLife.com | Micro Site
Eventseer.net - Home
Free Website Polls
Richest Cou...n the World
Castle for Sale
Programming with C/C++
JavaMOP Extended Regular ...
Log in / create account
Formal Systems Laboratory
navigation
Main Page
People
Projects
Publications
News
Events
search
Go
Search
JavaMOP Extended Regular Expression (JavaERE) Plugin
shortcuts:
MOP
EREPlugin
JavaMOP
MOP
logic plugins
languages
ERE
CFG
PTLTL
FTLTL
PTCaRet
...
JavaMOP
JavaERE
JavaCFG
JavaPTLTL
JavaFTLTL
JavaPTCaRet
BusMOP
BusERE
BusPTLTL
MOP Matrix: a clickable map of MOP pages.
JavaERE is an instance of the generic MOP ERE plugin for JavaMOP, the MOP instance for Java. JavaMOP extracts from a specification using the header "logic = ERE" the actual ERE and passes it to the JavaERE plugin. The JavaERE plugin encapsulates the ERE plugin as follows:
It sends the extracted ERE pattern to the ERE plugin;
It collects the output of the ERE plugin and produces Java monitoring code snippets (declarations, monitor initialization, monitor step, etc.).
JavaMOP takes the monitoring code snippets generated by the JavaERE plugin and weaves them, using AspectJ, within the original application.
Syntax
The syntax of JavaMOP ERE specifications is as expected:
JavaMOP syntax, plus
ERE Plugin syntax, plus
Two additional BNF rules:
<Logic> ::= "ERE"
<Property> ::= "formula:" <ERE Pattern>
Run the JavaERE Plugin Online
We provide a web-based interface to try JavaMOP with the JavaERE plugin online. Enter your specification in the form below, or chose (and possibly modify) one of the existing specifications. Run JavaERE by clicking the Run button. The generated code will be displayed underneath the form. The generated code can be compiled using any AspectJ compiler.


JavaMOP Extended Regular Expression (JavaERE) Plugin - FSL
http://fsl.cs.uiuc.edu/index.php/Special:JavaEREPlugin
Google
Joergen Ingman
TimeLife.com | Micro Site
Eventseer.net - Home
Free Website Polls
Richest Cou...n the World
Castle for Sale
Programming with C/C++
JavaMOP Extended Regular ...
Choose an example:
HashSet
Hashtable
LeakingSync
ListenerPattern
NullChecker
SafeEnum
SafeIterator
SafeReweave
ToDispose
p
HasNext
package paramere.hasnext;
import java.util.*;
/*@
partial centralized
scope = global
logic = ERE
HasNext(Iterator i)
{
event hasnext<i> : end(call(* i.hasNext()));
event next<i> : begin(call(* i.next()));
formula : next next
}
validation handler{
System.err.println("*** call hasNext() before next()");
}
@*/
Specification Syntax Help
Run
Reset
Done. Output:
ERE Plugin Output
JavaERE Plugin Output
Select All Output
package paramere.hasnext;
import java.util.*;
import org.apache.commons.collections.map.*;
class HasNextMonitorPM {
Vector monitorList = new Vector();
public void hasnext(Iterator i) {
HasSet monitorSet = new HashSet();
Iterator it = monitorList.iterator();
while (it.hasNext()){
HasNextMonitor monitor = (HasNextMonitor)it.next();
monitor.hasnext(i);
if (monitorSet.contains(monitor) || monitor.failed())
it.remove();
else {
monitorSet.add(monitor);
if (monitor.suceeded()){
System.err.println("*** call hasNext() before next()");
}
} // for else
} // for while
} // end of method
public void next(Iterator i) {
HashSet monitorSet = new HashSet();
monitorList.add(new HasNextMonitor());
Iterator it = monitorList.iterator();

http://fsl.cs.uiuc.edu/MOP/plugin.php

http://fsl.cs.uiuc.edu/MOP/plugin.php

Google

Joergen Ingman | TimeLife.com | Micro Site | Eventseer.net - Home | Free Website Polls | Richest Cou...n the World | Castle for Sale | Programming with C/C++ | Costa Rica, ...Estate Blogs

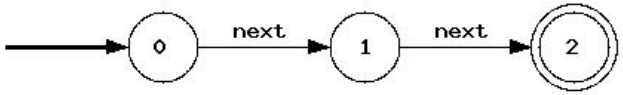
http://fsl.cs.uiuc.edu/MOP/...

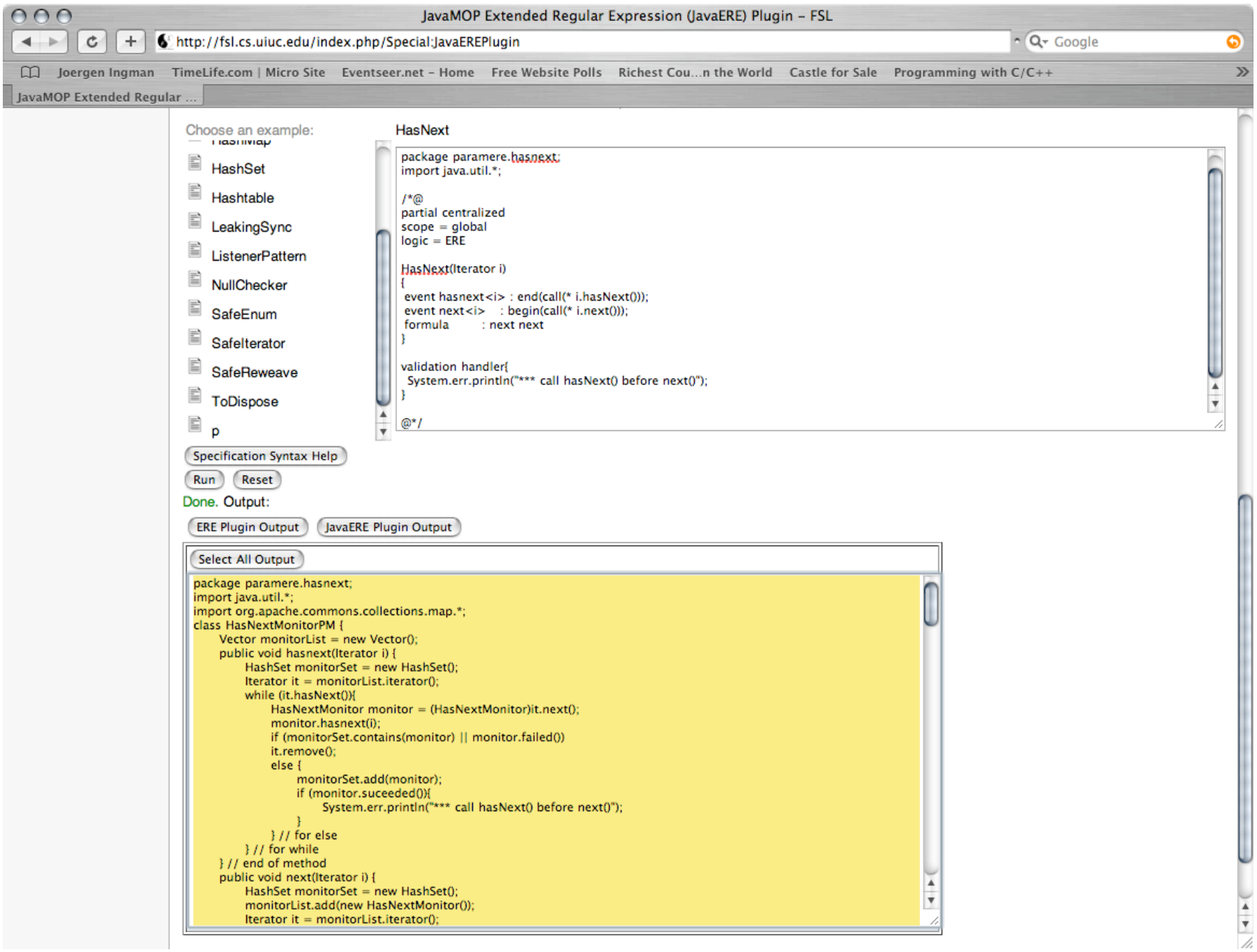
Text Output:

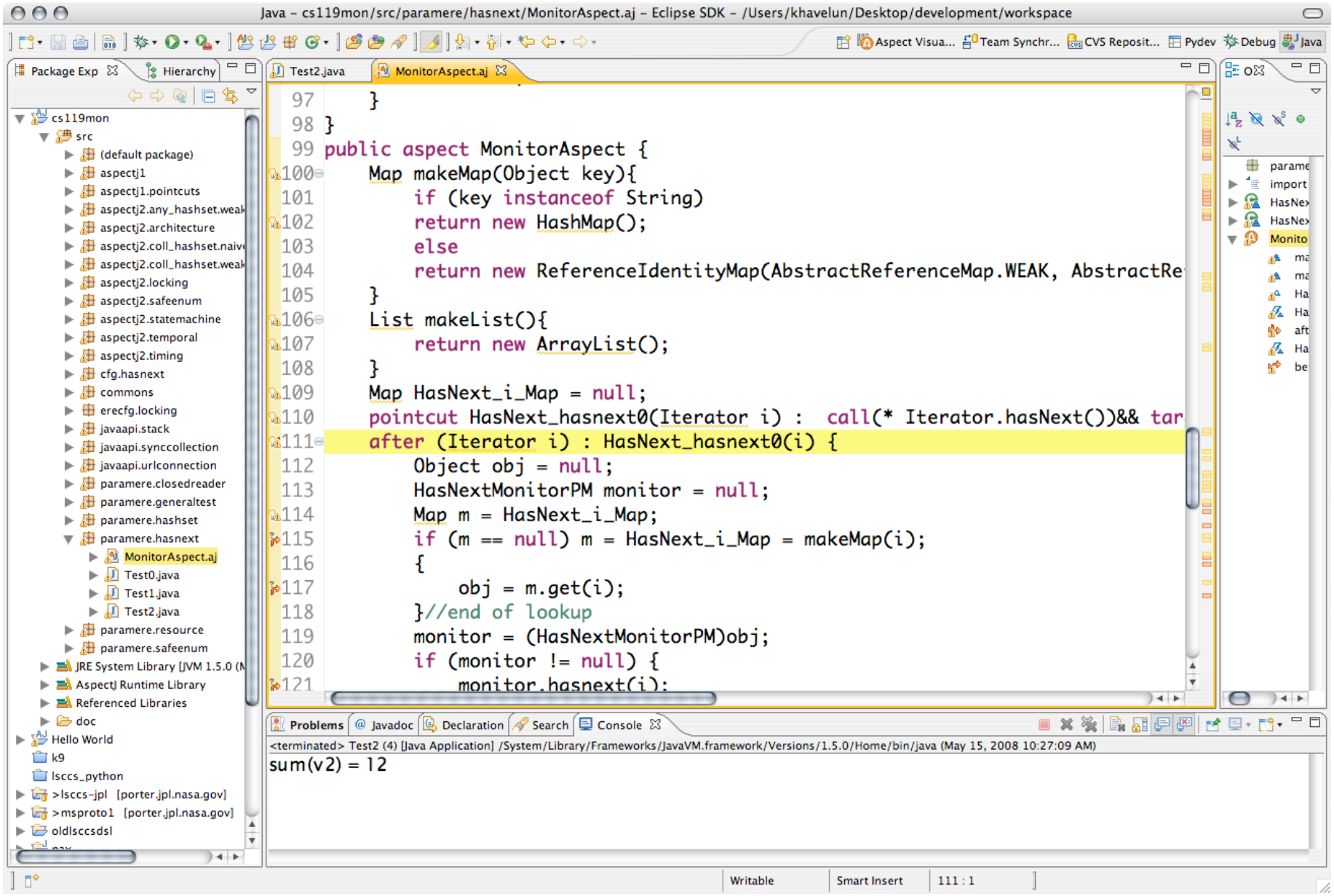
Starting state: 0
Accepting states: 2
Transitions:
0 -> 1 [label = "next"];
1 -> 2 [label = "next"];

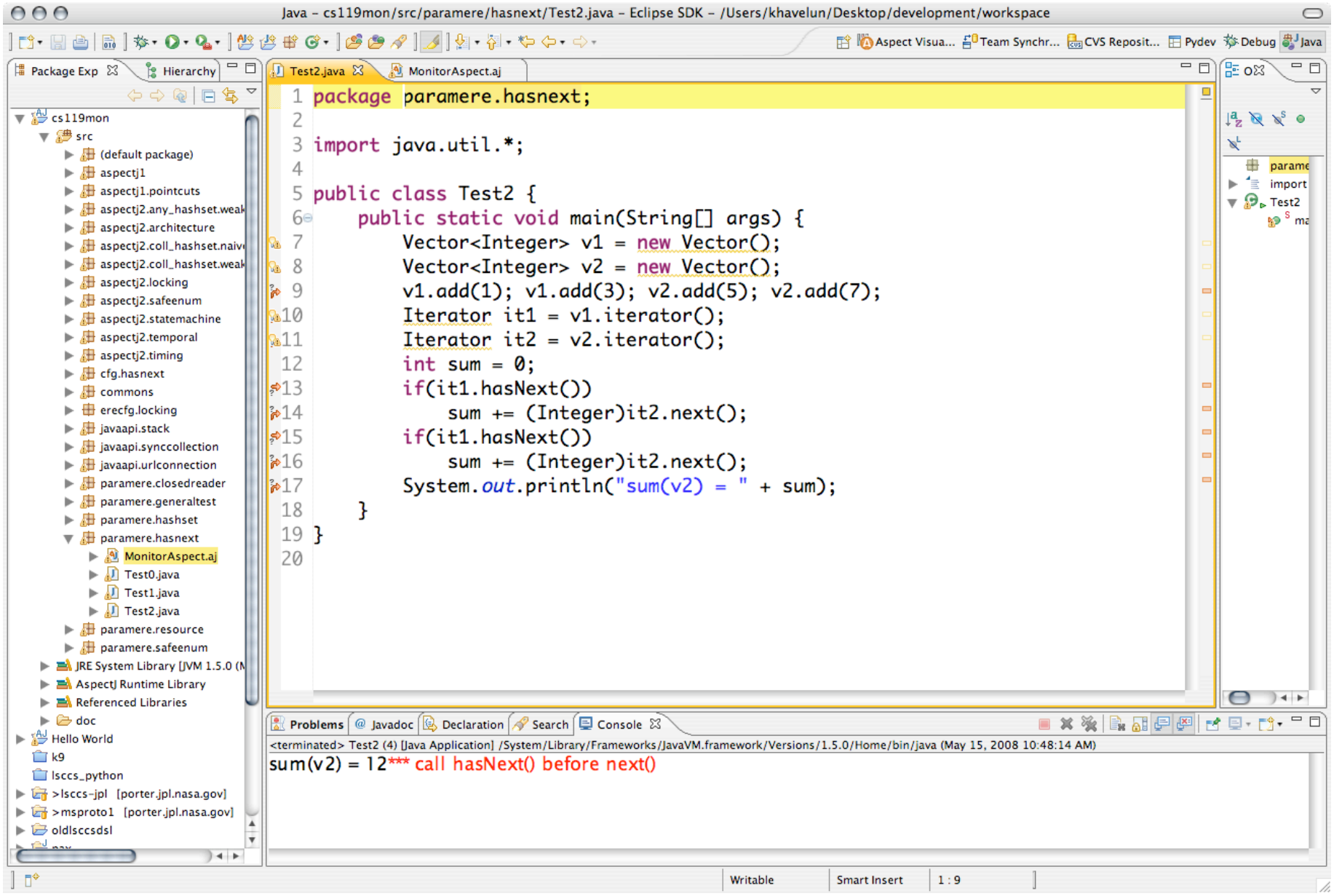
Image Representation:

next next




JavaMOP Extended Regular Expression (JavaERE) Plugin – FSL
http://fsl.cs.uiuc.edu/index.php/Special:JavaEREPlugin
Google
Joergen Ingman
TimeLife.com | Micro Site
Eventseer.net – Home
Free Website Polls
Richest Cou...n the World
Castle for Sale
Programming with C/C++
JavaMOP Extended Regular ...
Choose an example:
HashSet
Hashtable
LeakingSync
ListenerPattern
NullChecker
SafeEnum
SafeIterator
SafeReweave
ToDispose
p
HasNext
package paramere.hasnext;
import java.util.*;
/*@
partial centralized
scope = global
logic = ERE
HasNext(Iterator i)
{
event hasnext<i> : end(call(* i.hasNext()));
event next<i> : begin(call(* i.next()));
formula : next next
}
validation handler{
System.err.println("*** call hasNext() before next()");
}
@*/
Specification Syntax Help
Run
Reset
Done. Output:
ERE Plugin Output
JavaERE Plugin Output
Select All Output
package paramere.hasnext;
import java.util.*;
import org.apache.commons.collections.map.*;
class HasNextMonitorPM {
Vector monitorList = new Vector();
public void hasnext(Iterator i) {
HashSet monitorSet = new HashSet();
Iterator it = monitorList.iterator();
while (it.hasNext()){
HasNextMonitor monitor = (HasNextMonitor)it.next();
monitor.hasnext(i);
if (monitorSet.contains(monitor) || monitor.failed())
it.remove();
else {
monitorSet.add(monitor);
if (monitor.suceeded()){
System.err.println("*** call hasNext() before next()");
}
} // for else
} // for while
} // end of method
public void next(Iterator i) {
HashSet monitorSet = new HashSet();
monitorList.add(new HasNextMonitor());
Iterator it = monitorList.iterator();


Java – cs119mon/src/paramere/hasnext/MonitorAspect.aj – Eclipse SDK – /Users/khavelun/Desktop/development/workspace
Package Exp
Hierarchy
Test2.java
MonitorAspect.aj
cs119mon
src
(default package)
aspectj1
aspectj1.pointcuts
aspectj2.architecture
aspectj2.locking
aspectj2.safeenum
aspectj2.statemachine
aspectj2.temporal
aspectj2.timing
cfg.hasnext
commons
erecfg.locking
javaapi.stack
javaapi.synccollection
javaapi.urlconnection
paramere.closedreader
paramere.generaltest
paramere.hashset
paramere.hasnext
MonitorAspect.aj
Test0.java
Test1.java
Test2.java
paramere.resource
paramere.safeenum
JRE System Library [JVM 1.5.0
AspectJ Runtime Library
Referenced Libraries
doc
Hello World
k9
lsccs_python
>lsccs-jpl [porter.jpl.nasa.gov]
>msproto1 [porter.jpl.nasa.gov]
oldlsccsdsl
97 }
98 }
99 public aspect MonitorAspect {
100 Map makeMap(Object key){
101 if (key instanceof String)
102 return new HashMap();
103 else
104 return new ReferenceIdentityMap(AbstractReferenceMap.WEAK, AbstractRe
105 }
106 List makeList(){
107 return new ArrayList();
108 }
109 Map HasNext_i_Map = null;
110 pointcut HasNext_hasnext0(Iterator i) : call(* Iterator.hasNext())&& tar
111 after (Iterator i) : HasNext_hasnext0(i) {
112 Object obj = null;
113 HasNextMonitorPM monitor = null;
114 Map m = HasNext_i_Map;
115 if (m == null) m = HasNext_i_Map = makeMap(i);
116 {
117 obj = m.get(i);
118 }//end of lookup
119 monitor = (HasNextMonitorPM)obj;
120 if (monitor != null) {
121 monitor.hasnext(i);
Problems
Javadoc
Declaration
Search
Console
<terminated> Test2 (4) [Java Application] /System/Library/Frameworks/JavaVM.framework/Versions/1.5.0/Home/bin/java (May 15, 2008 10:27:09 AM)
sum(v2) = 12
Writable
Smart Insert
111 : 1


Java – cs119mon/src/paramere/hasnext/Test2.java – Eclipse SDK – /Users/khavelun/Desktop/development/workspace
Package Exp
Hierarchy
Test2.java
MonitorAspect.aj
cs119mon
src
(default package)
aspectj1
aspectj1.pointcuts
aspectj2.architecture
aspectj2.locking
aspectj2.safeenum
aspectj2.statemachine
aspectj2.temporal
aspectj2.timing
cfg.hasnext
commons
javaapi.stack
javaapi.synccollection
javaapi.urlconnection
paramere.closedreader
paramere.generaltest
paramere.hashset
paramere.hasnext
MonitorAspect.aj
Test0.java
Test1.java
Test2.java
paramere.resource
paramere.safeenum
JRE System Library [JVM 1.5.0
AspectJ Runtime Library
Referenced Libraries
doc
Hello World
k9
lsccs_python
>lsccs-jpl [porter.jpl.nasa.gov]
>msproto1 [porter.jpl.nasa.gov]
oldlsccsdsl
1 package paramere.hasnext;
2
3 import java.util.*;
4
5 public class Test2 {
6 public static void main(String[] args) {
7 Vector<Integer> v1 = new Vector();
8 Vector<Integer> v2 = new Vector();
9 v1.add(1); v1.add(3); v2.add(5); v2.add(7);
10 Iterator it1 = v1.iterator();
11 Iterator it2 = v2.iterator();
12 int sum = 0;
13 if(it1.hasNext())
14 sum += (Integer)it2.next();
15 if(it1.hasNext())
16 sum += (Integer)it2.next();
17 System.out.println("sum(v2) = " + sum);
18 }
19 }
20
Problems
Javadoc
Declaration
Search
Console
<terminated> Test2 (4) [Java Application] /System/Library/Frameworks/JavaVM.framework/Versions/1.5.0/Home/bin/java (May 15, 2008 10:48:14 AM)
sum(v2) = 12*** call hasNext() before next()
Writable
Smart Insert
1 : 9

END OF
DEMO ON SLIDES

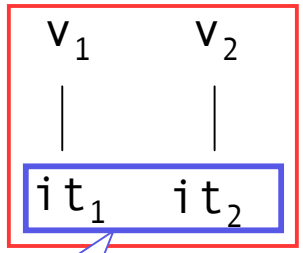
data values

- handling of data values efficiently in monitoring frameworks is one of the current research challenges.
- different frameworks approach it differently, achieving different expressiveness and efficiency.

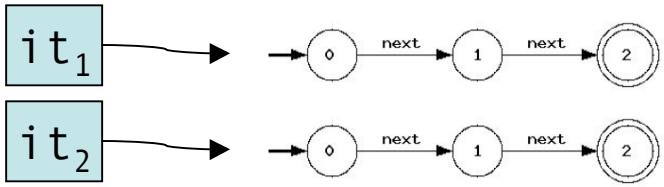
in this simple case: store monitor in iterator

our program again

```
class Test {
  public static void main(String[] args) {
    Vector<Integer> v1 = new Vector();
    Vector<Integer> v2 = new Vector();
    v1.add(1); v1.add(3); v2.add(5); v2.add(7);
    Iterator it1 = v1.iterator();
    Iterator it2 = v2.iterator();
    int sum = 0;
    if(it1.hasNext())
      sum += (Integer)it2.next();
    if(it1.hasNext())
      sum += (Integer)it2.next();
    System.out.println("sum(v2) = " + sum);
  }
}
```



only objects of interest

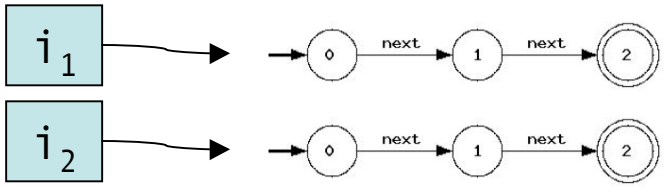


associate a monitor with each object of interest

Piggy-back (transportation), something that is riding on the back of something else

algorithm

- assume a property only mentions one kind of object (an iterator in this case).
- note, there could be many objects monitored (it_1,it_2).
- one can piggy-back a monitor on each object that is monitored.
- alternatively: keep a map from objects to their monitors.
- when an operation of interest occurs (hasNext() or next()) that refers to an object, the monitor of that object is “executed”. If null, a monitor is first created.



associate a monitor with each object of interest

algorithm pseudo-code

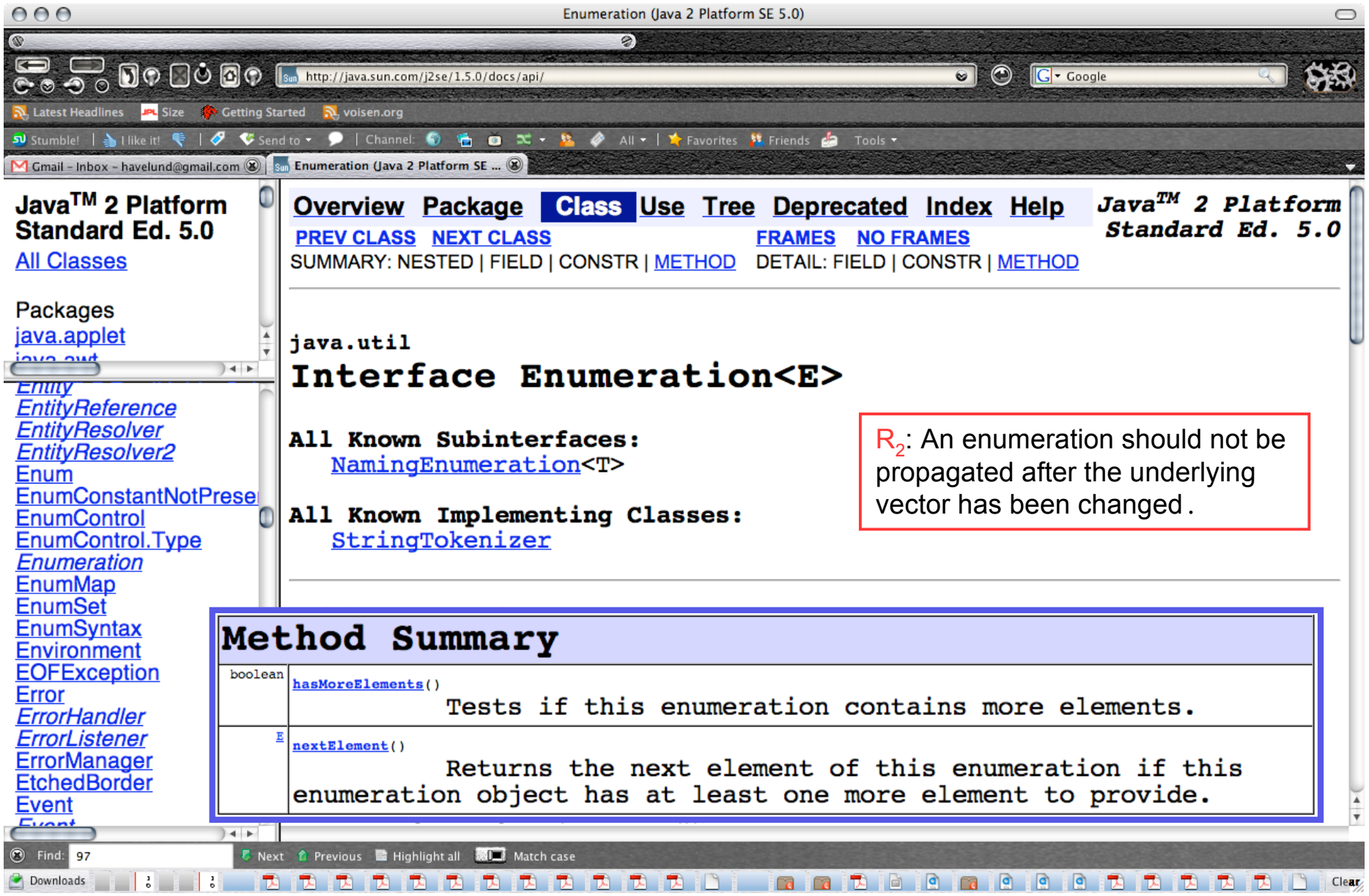
Consider a call of it.next() for example. This could lead to the following monitoring code:

```
if (it.monitor == null)
  it.monitor = new Monitor();
it.monitor.next();
```

however, this does not work when property refers to more than 1 object, where to store the monitor?

not to mention that the java library cannot be instrumented easily.

properties of Java library APIs



R_2: An enumeration should not be propagated after the underlying vector has been changed.

java.util

Interface Enumeration<E>

All Known Subinterfaces:
NamingEnumeration<T>

All Known Implementing Classes:
StringTokenizer

R_2: An enumeration should not be propagated after the underlying vector has been changed .

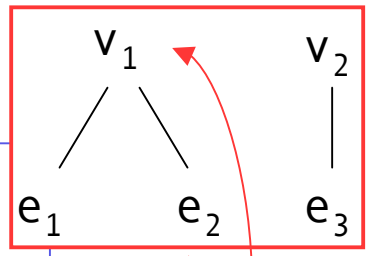
create next* updatesource updatesource* next

now the property must refer to more than one object: a vector and an enumerator

Piggy-back no longer works. There is not a 1-1 correspondence between objects and monitors

many vectors and enumerators

```
Vector v1 = new Vector();
Vector v2 = new Vector();
v1.add(1); v1.add(2); v2.add(4); v2.add(5);
Enumeration e1 = v1.elements();
Enumeration e2 = v1.elements();
Enumeration e3 = v2.elements();
while(e1.hasMoreElements())print(e1.nextElement());
v1.add(99);
while(e2.hasMoreElements())print(e2.nextElement());
while(e3.hasMoreElements())print(e3.nextElement());
```



add

next

events:

```
create(v,e)
next(e)
updatesource(v)
```

$\forall$ (v,e) $\in$ { (v_1,e_1),(v_1,e_2),(v_2,e_3) }

create(v,e)

next(e)*

updatesource(v)+

next(e)

catch this pattern

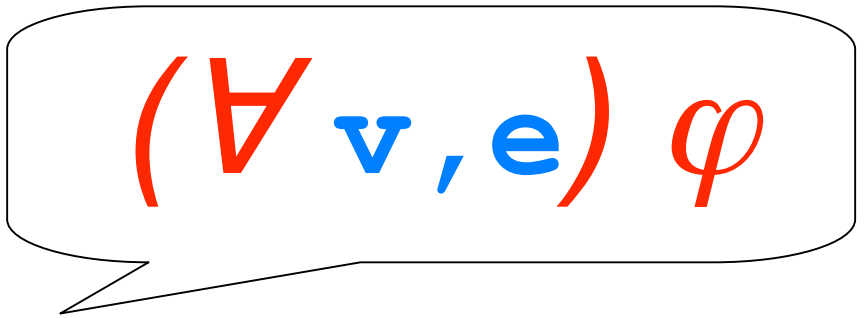
specification

```
/*@
partial centralized
scope = global
logic = ERE
SafeEnum (Vector v, Enumeration+ e) {
  event create<e, v> : end(call(Enumeration v.elements())) with (e);
  event updatesource<v> :
                        end(call(* v.add*(..)))
                    \/ end(call(* v.clear()))
                    \/ end(call(* v.insertElementAt(..))) &&
                    \/ end(call(* v.remove*(..)))
                    \/ end(call(* v.retainAll(..)))
                    \/ end(call(* v.set*(..)));
  event next<e> : begin(call(Object e.nextElement()));

  formula : create next* updatesource updatesource* next
}

validation handler {
   System.out.println("datasource changed during iteration!");
}
@*/
```

MOP's distinguished feature: separate handling of values

- *The problem*: how to monitor a universally quantified specification *efficiently*!

```
create<v,e>
udatesource<v>
next<e>
```




```
create next* updatesource+ next
```

two approaches to data

- Eagle, PQL, PTQL, Tracematches, … :
  - Choose a quantified logic
  - Device monitor synthesizer for it:

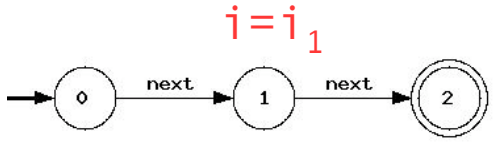
$$(\forall p)\ \varphi \quad \Longrightarrow \quad \mathcal{M}on_{(\forall p)\ \varphi}$$

- MOP:
  - only knows how to generate monitors for φ
  - The $(\forall p)$ is dealt with separately, and once-and-for-all, for all logics

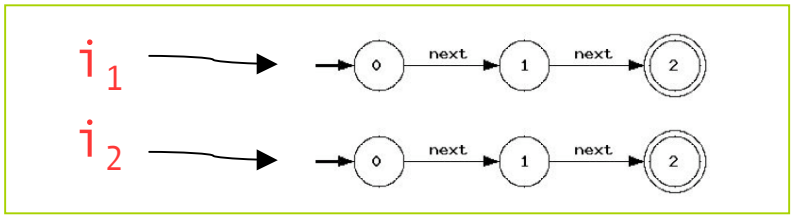
$$(\forall p)\ \varphi \quad \Longrightarrow \quad p \rightarrow \mathcal{M}on_{\varphi}$$

handling of data values

- value embedding scheme : the values become part of the monitors (state machines), for example by labeling transitions. This means modifying the state machine concept.

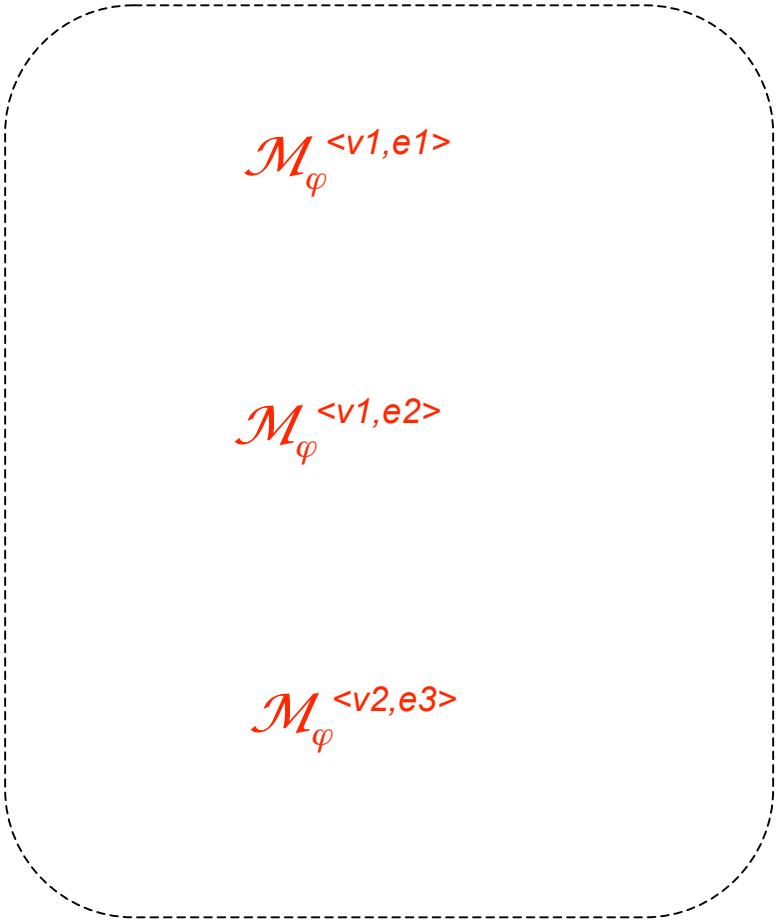


- value indexing scheme : Keep automata atomic and index them with data
  - If one object per property max:
    - Map objects to monitors
    - Or piggyback monitor on object
  - If more than one object per property:
    - Centralized index table from data to monitors
    - Perhaps combination of piggybacking and centralized index table



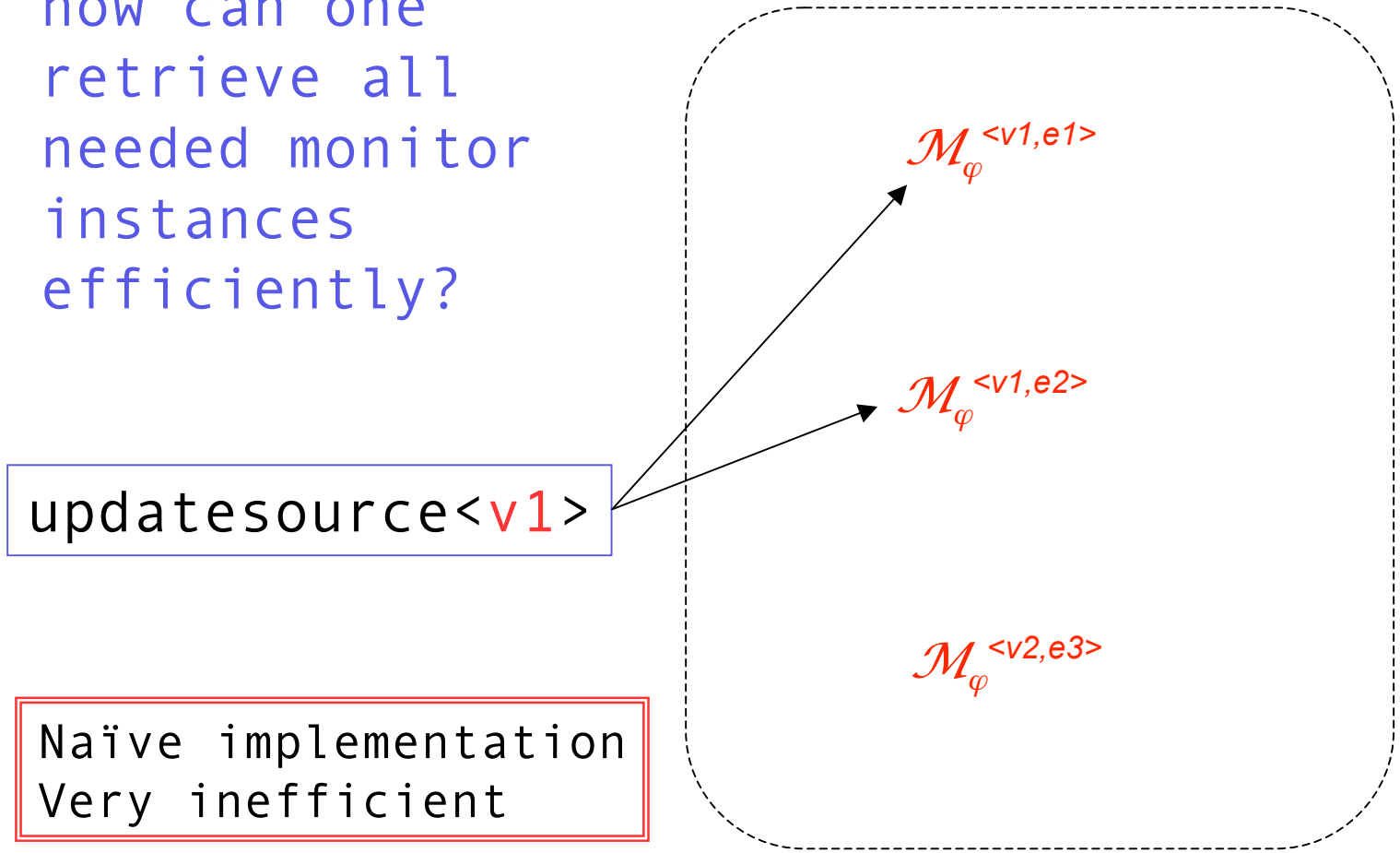
MOP's propositional monitors

one propositional monitor instance per parameter instance



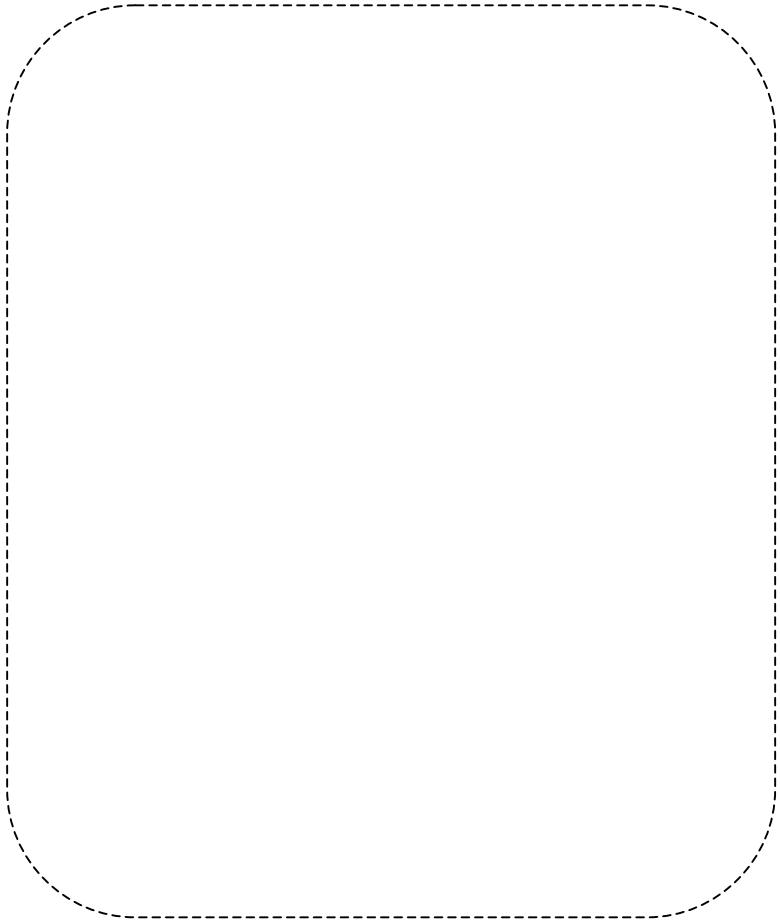
MOP's propositional monitors

The problem:
how can one retrieve all needed monitor instances efficiently?



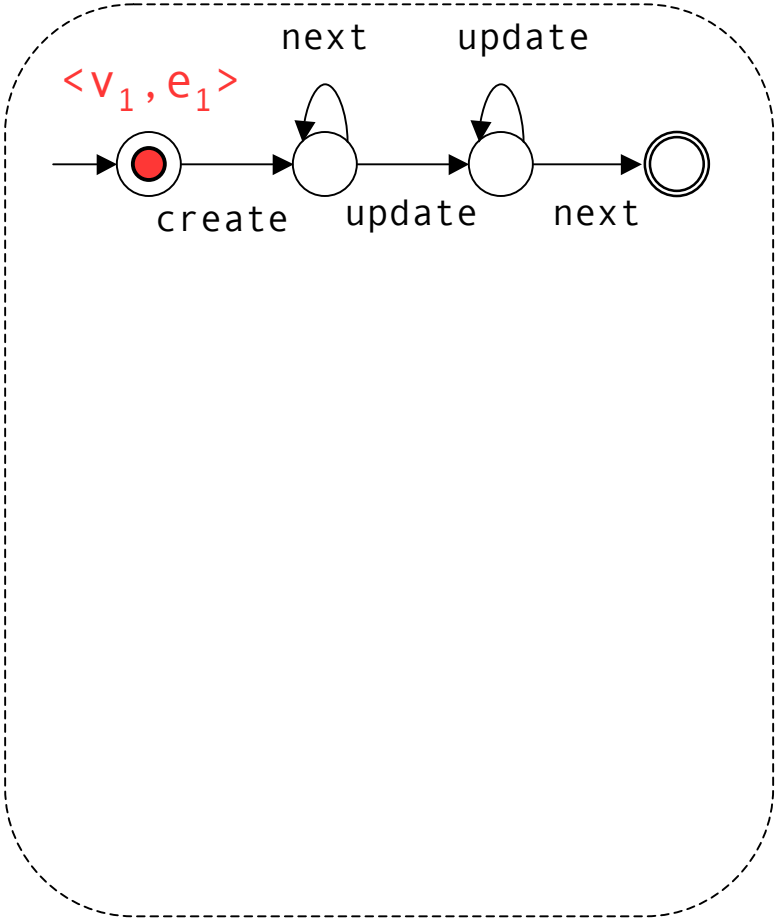
Naïve implementation

```
Vector v1 = new Vector();
Vector v2 = new Vector();
v1.add(1); v1.add(2); v2.add(4); v2.add(5);
Enumeration e1 = v1.elements();
Enumeration e2 = v1.elements();
Enumeration e3 = v2.elements();
while(e1.hasMoreElements())print(e1.nextElement());
v1.add(99);
while(e2.hasMoreElements())print(e2.nextElement());
while(e3.hasMoreElements())print(e3.nextElement());
```



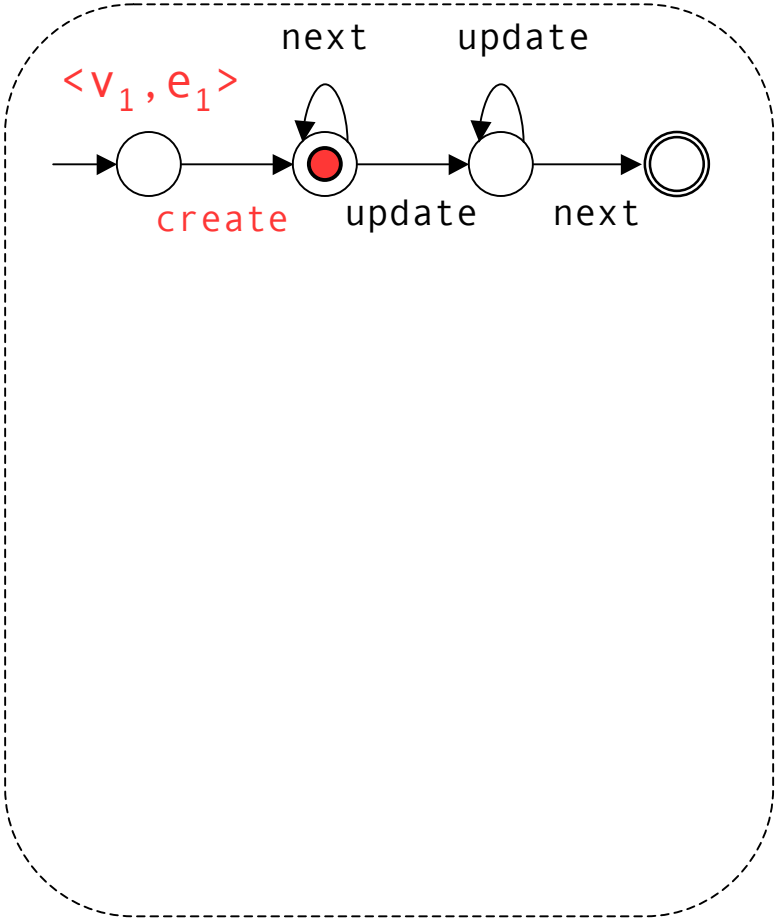
Naïve implementation

```
Vector v1 = new Vector();
Vector v2 = new Vector();
v1.add(1); v1.add(2); v2.add(4); v2.add(5);
Enumeration e1 = v1.elements();
Enumeration e2 = v1.elements();
Enumeration e3 = v2.elements();
while(e1.hasMoreElements())print(e1.nextElement());
v1.add(99);
while(e2.hasMoreElements())print(e2.nextElement());
while(e3.hasMoreElements())print(e3.nextElement());
```



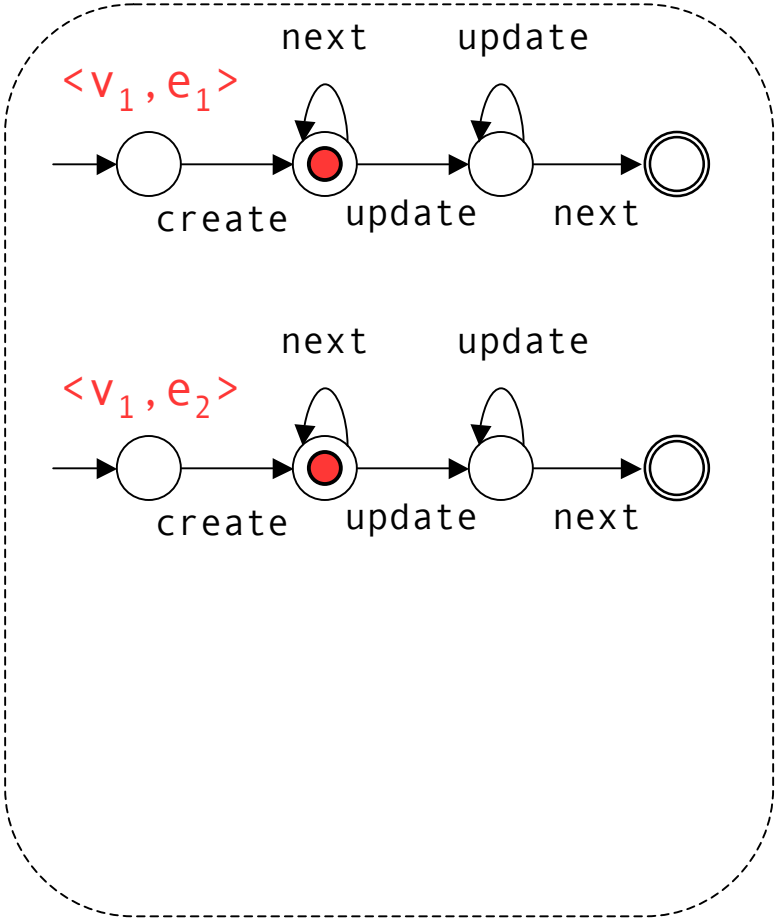
Naïve implementation

```
Vector v1 = new Vector();
Vector v2 = new Vector();
v1.add(1); v1.add(2); v2.add(4); v2.add(5);
Enumeration e1 = v1.elements();
Enumeration e2 = v1.elements();
Enumeration e3 = v2.elements();
while(e1.hasMoreElements())print(e1.nextElement());
v1.add(99);
while(e2.hasMoreElements())print(e2.nextElement());
while(e3.hasMoreElements())print(e3.nextElement());
```



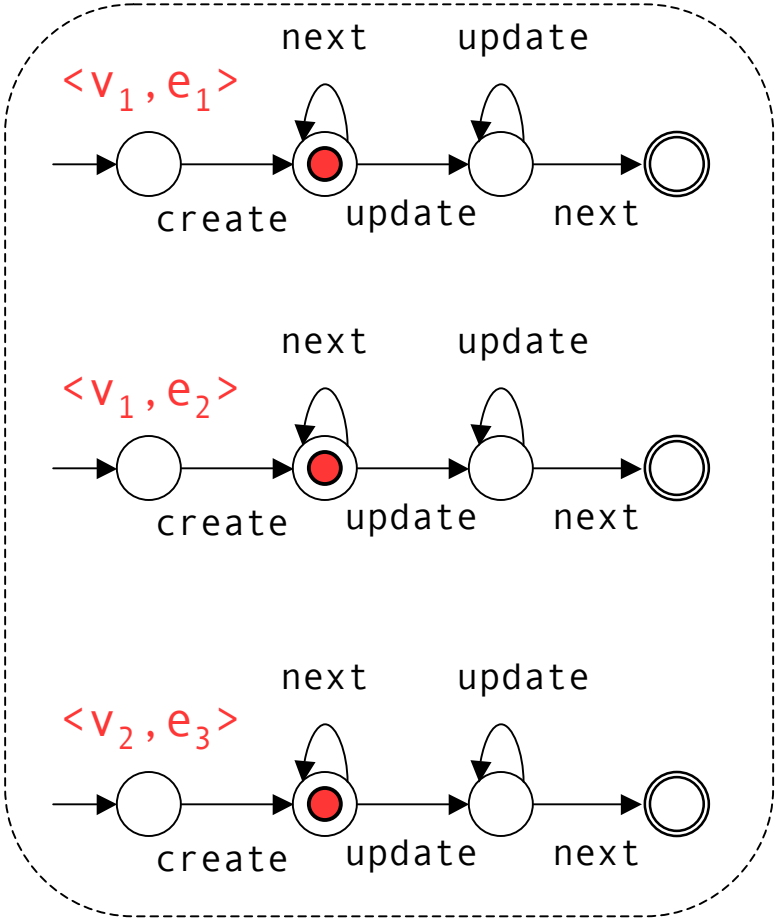
Naïve implementation

```
Vector v1 = new Vector();
Vector v2 = new Vector();
v1.add(1); v1.add(2); v2.add(4); v2.add(5);
Enumeration e1 = v1.elements();
Enumeration e2 = v1.elements();
Enumeration e3 = v2.elements();
while(e1.hasMoreElements())print(e1.nextElement());
v1.add(99);
while(e2.hasMoreElements())print(e2.nextElement());
while(e3.hasMoreElements())print(e3.nextElement());
```



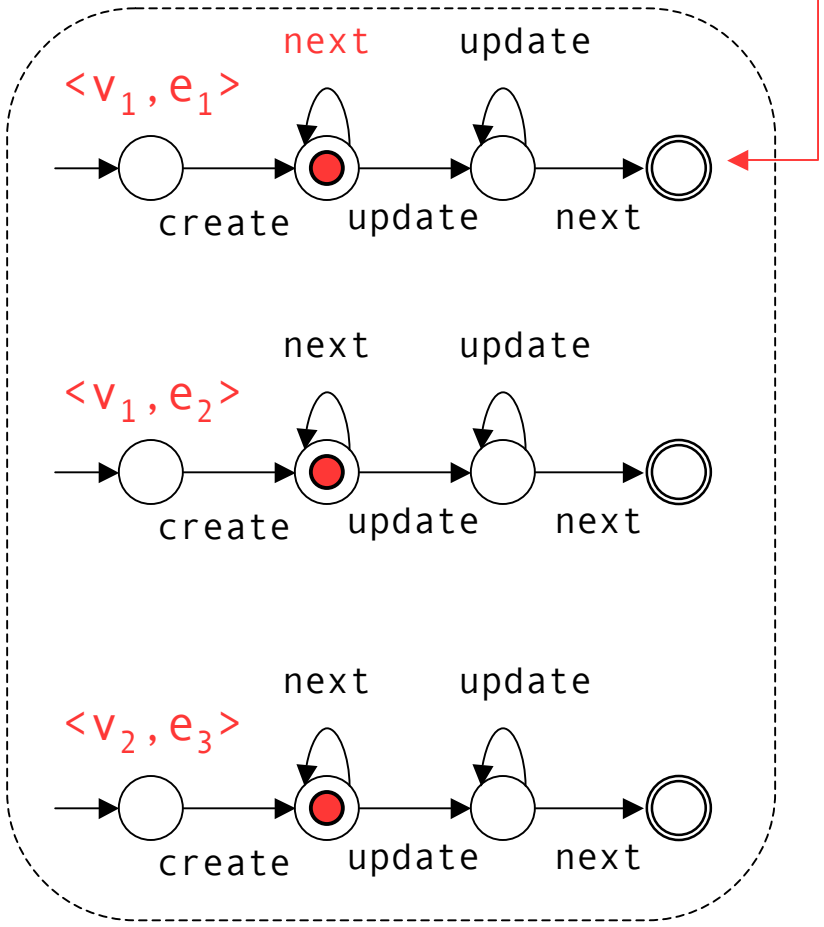
Naïve implementation

```
Vector v1 = new Vector();
Vector v2 = new Vector();
v1.add(1); v1.add(2); v2.add(4); v2.add(5);
Enumeration e1 = v1.elements();
Enumeration e2 = v1.elements();
Enumeration e3 = v2.elements();
while(e1.hasMoreElements())print(e1.nextElement());
v1.add(99);
while(e2.hasMoreElements())print(e2.nextElement());
while(e3.hasMoreElements())print(e3.nextElement());
```



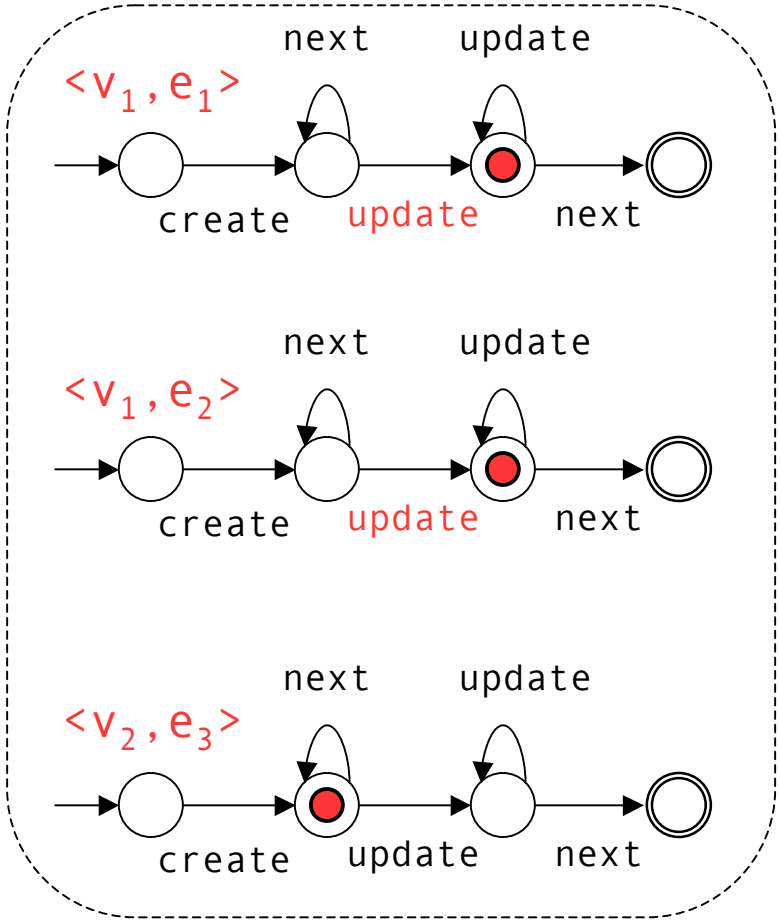
Naïve implementation

```
Vector v1 = new Vector();
Vector v2 = new Vector();
v1.add(1); v1.add(2); v2.add(4); v2.add(5);
Enumeration e1 = v1.elements();
Enumeration e2 = v1.elements();
Enumeration e3 = v2.elements();
while(e1.hasMoreElements())print(e1.nextElement());
v1.add(99);
while(e2.hasMoreElements())print(e2.nextElement());
while(e3.hasMoreElements())print(e3.nextElement());
```



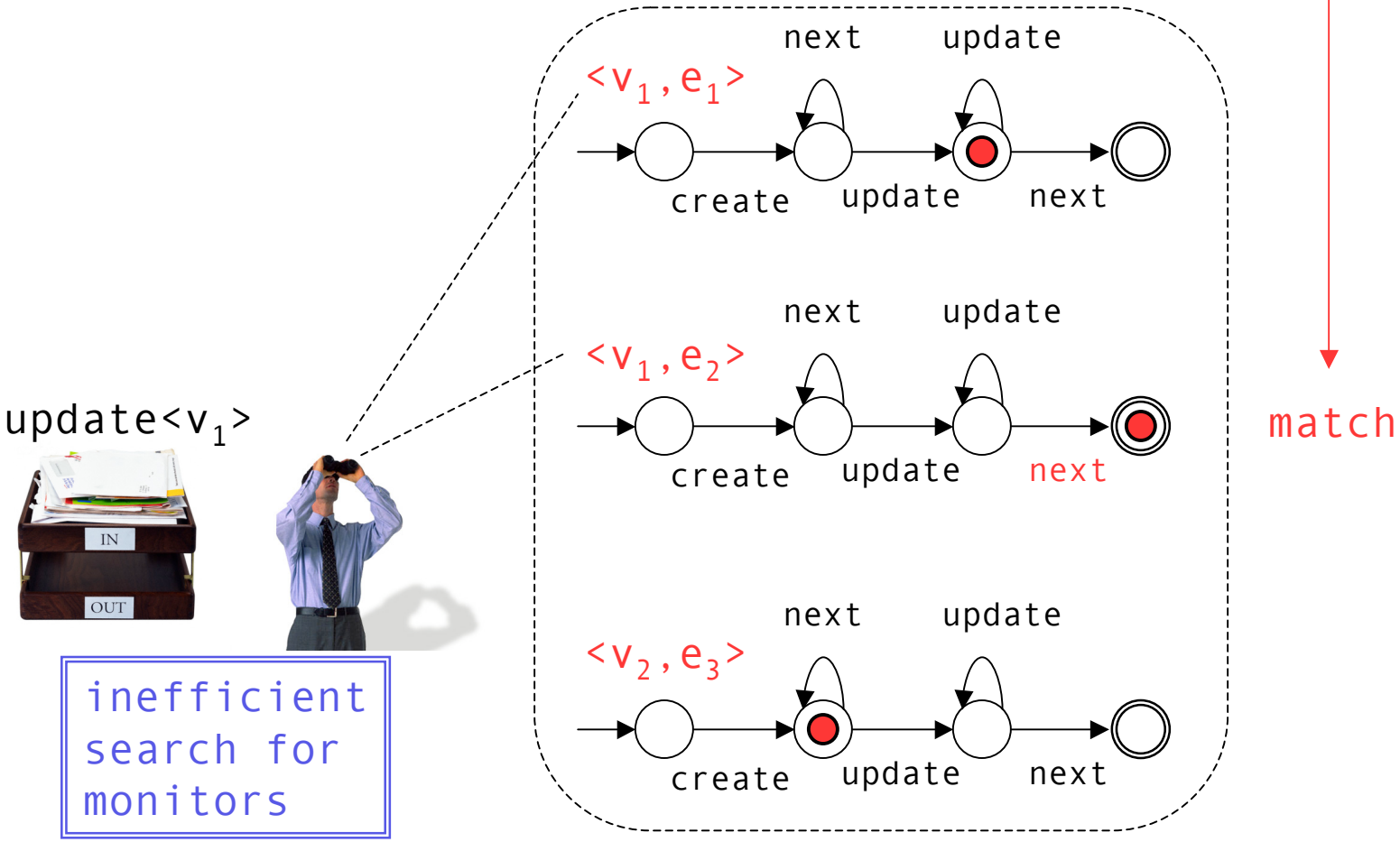
Naïve implementation

```
Vector v1 = new Vector();
Vector v2 = new Vector();
v1.add(1); v1.add(2); v2.add(4); v2.add(5);
Enumeration e1 = v1.elements();
Enumeration e2 = v1.elements();
Enumeration e3 = v2.elements();
while(e1.hasMoreElements())print(e1.nextElement());
v1.add(99);
while(e2.hasMoreElements())print(e2.nextElement());
while(e3.hasMoreElements())print(e3.nextElement());
```



Naïve implementation

```
Vector v1 = new Vector();
Vector v2 = new Vector();
v1.add(1); v1.add(2); v2.add(4); v2.add(5);
Enumeration e1 = v1.elements();
Enumeration e2 = v1.elements();
Enumeration e3 = v2.elements();
while(e1.hasMoreElements())print(e1.nextElement());
v1.add(99);
while(e2.hasMoreElements())print(e2.nextElement());
while(e3.hasMoreElements())print(e3.nextElement());
```

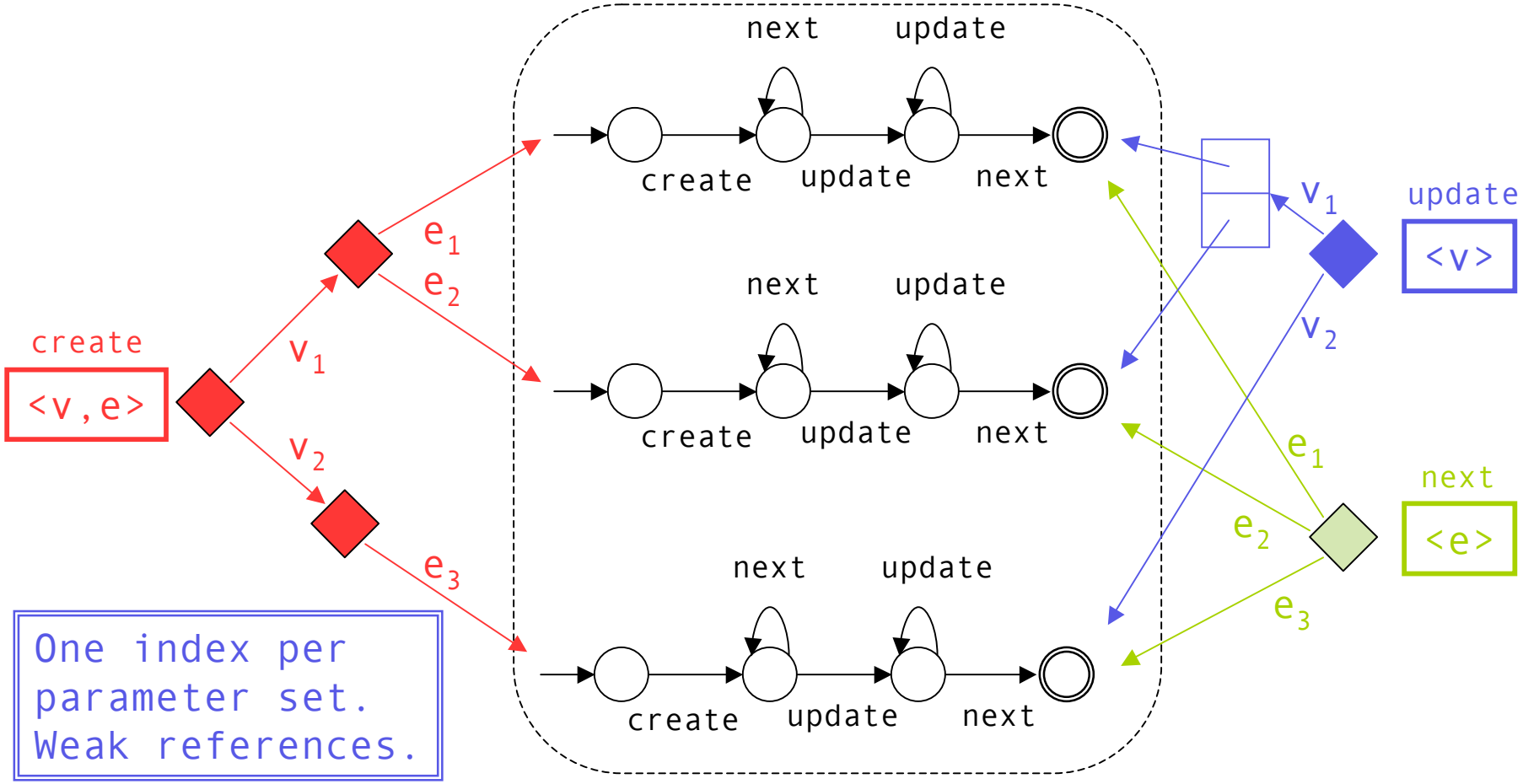


Indexed implementation

```
Vector v1 = new Vector();
Vector v2 = new Vector();
v1.add(1); v1.add(2); v2.add(4); v2.add(5);
Enumeration e1 = v1.elements();
Enumeration e2 = v1.elements();
Enumeration e3 = v2.elements();
while(e1.hasMoreElements())print(e1.nextElement());
v1.add(99);
while(e2.hasMoreElements())print(e2.nextElement());
while(e3.hasMoreElements())print(e3.nextElement());
```

events:
create<v,e>
update<v>
next<e>

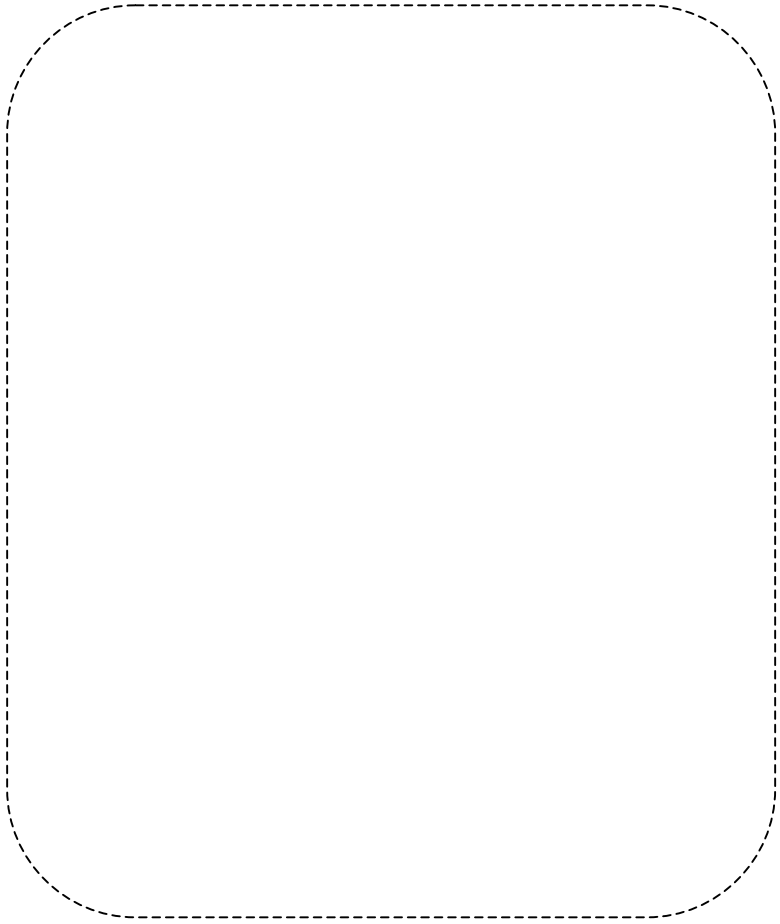
monitor creation event



One index per parameter set. Weak references.

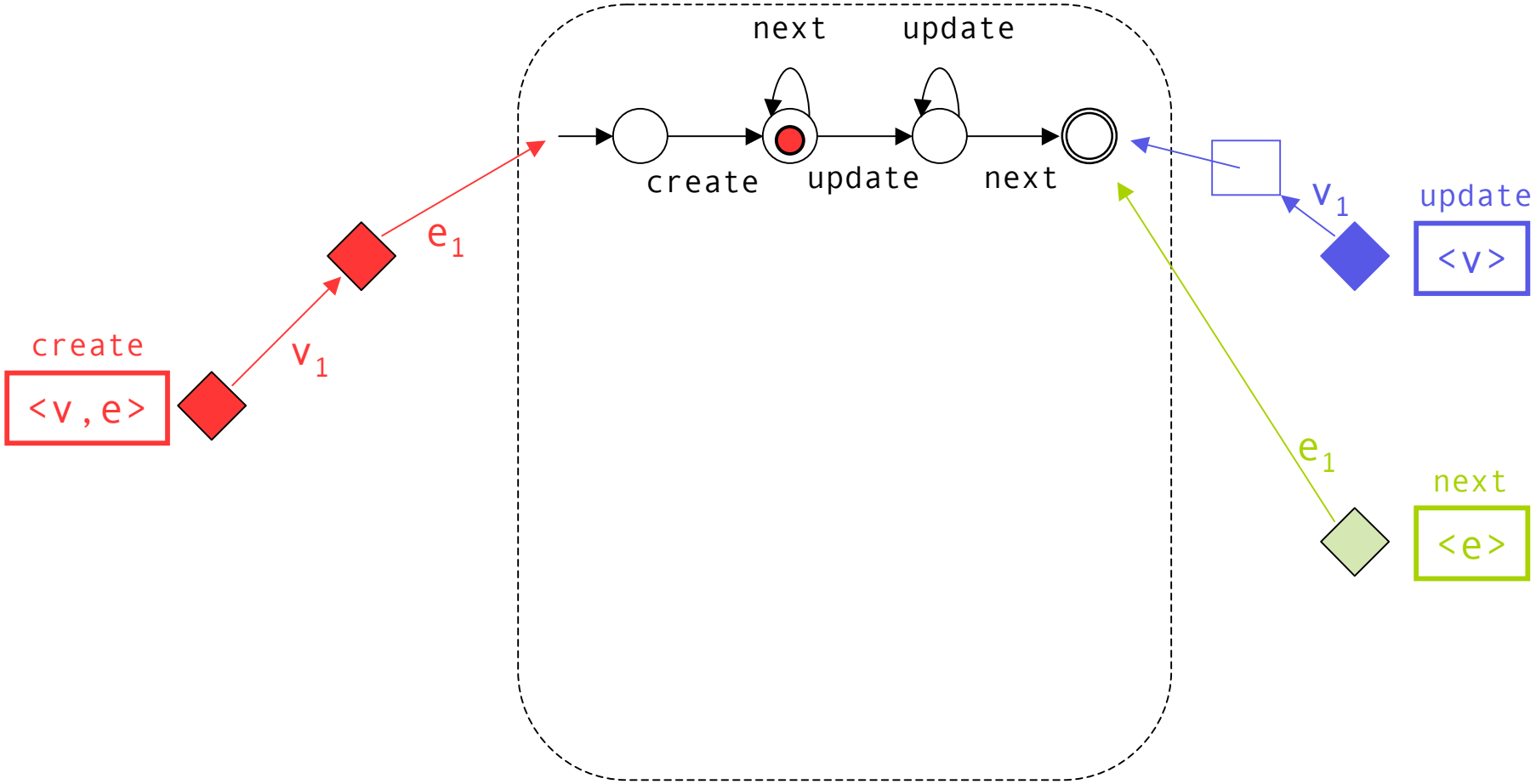
Indexed implementation

```
Vector v1 = new Vector();
Vector v2 = new Vector();
v1.add(1); v1.add(2); v2.add(4); v2.add(5);
Enumeration e1 = v1.elements();
Enumeration e2 = v1.elements();
Enumeration e3 = v2.elements();
while(e1.hasMoreElements())print(e1.nextElement());
v1.add(99);
while(e2.hasMoreElements())print(e2.nextElement());
while(e3.hasMoreElements())print(e3.nextElement());
```



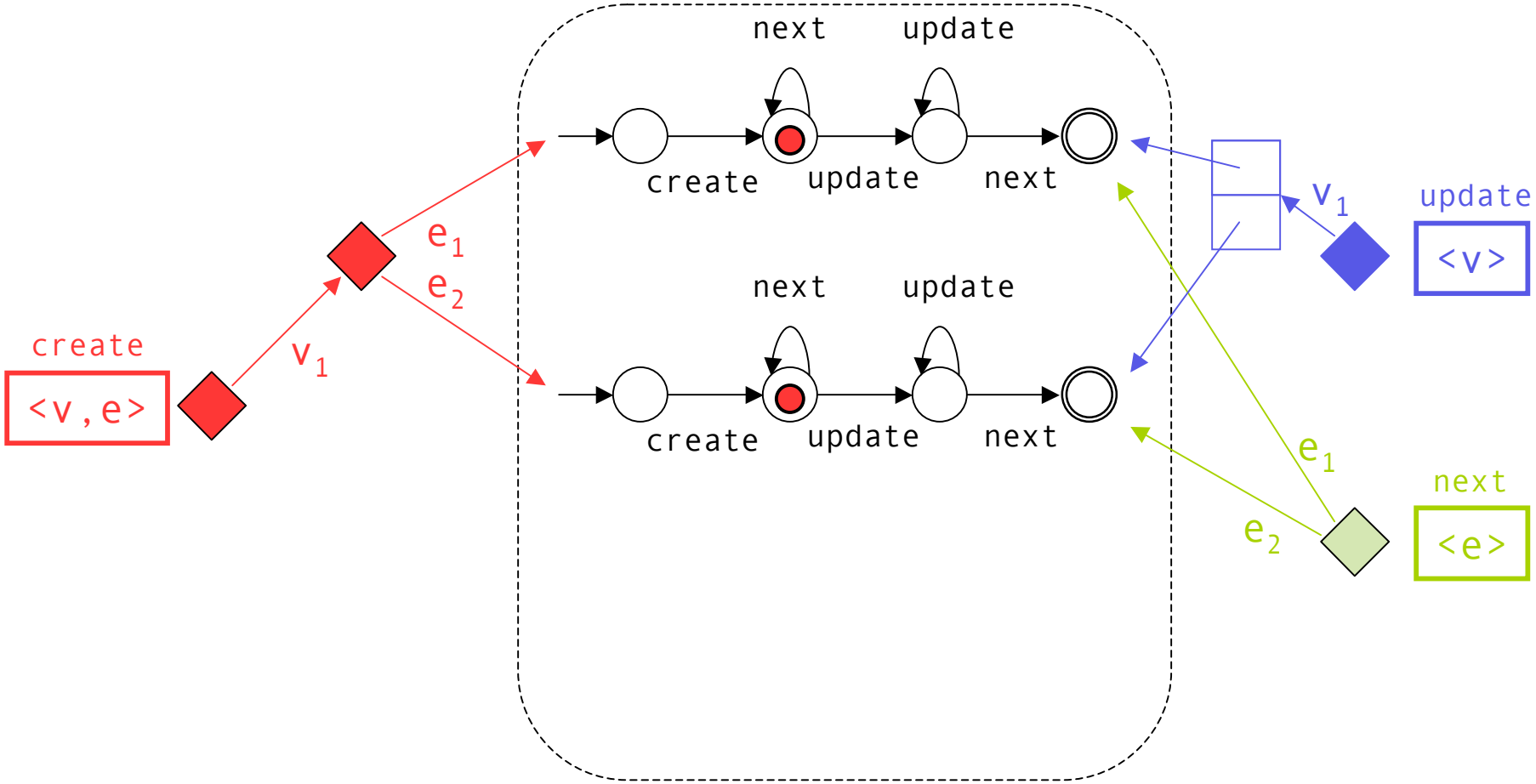
Indexed implementation

```
Vector v1 = new Vector();
Vector v2 = new Vector();
v1.add(1); v1.add(2); v2.add(4); v2.add(5);
Enumeration e1 = v1.elements();
Enumeration e2 = v1.elements();
Enumeration e3 = v2.elements();
while(e1.hasMoreElements())print(e1.nextElement());
v1.add(99);
while(e2.hasMoreElements())print(e2.nextElement());
while(e3.hasMoreElements())print(e3.nextElement());
```



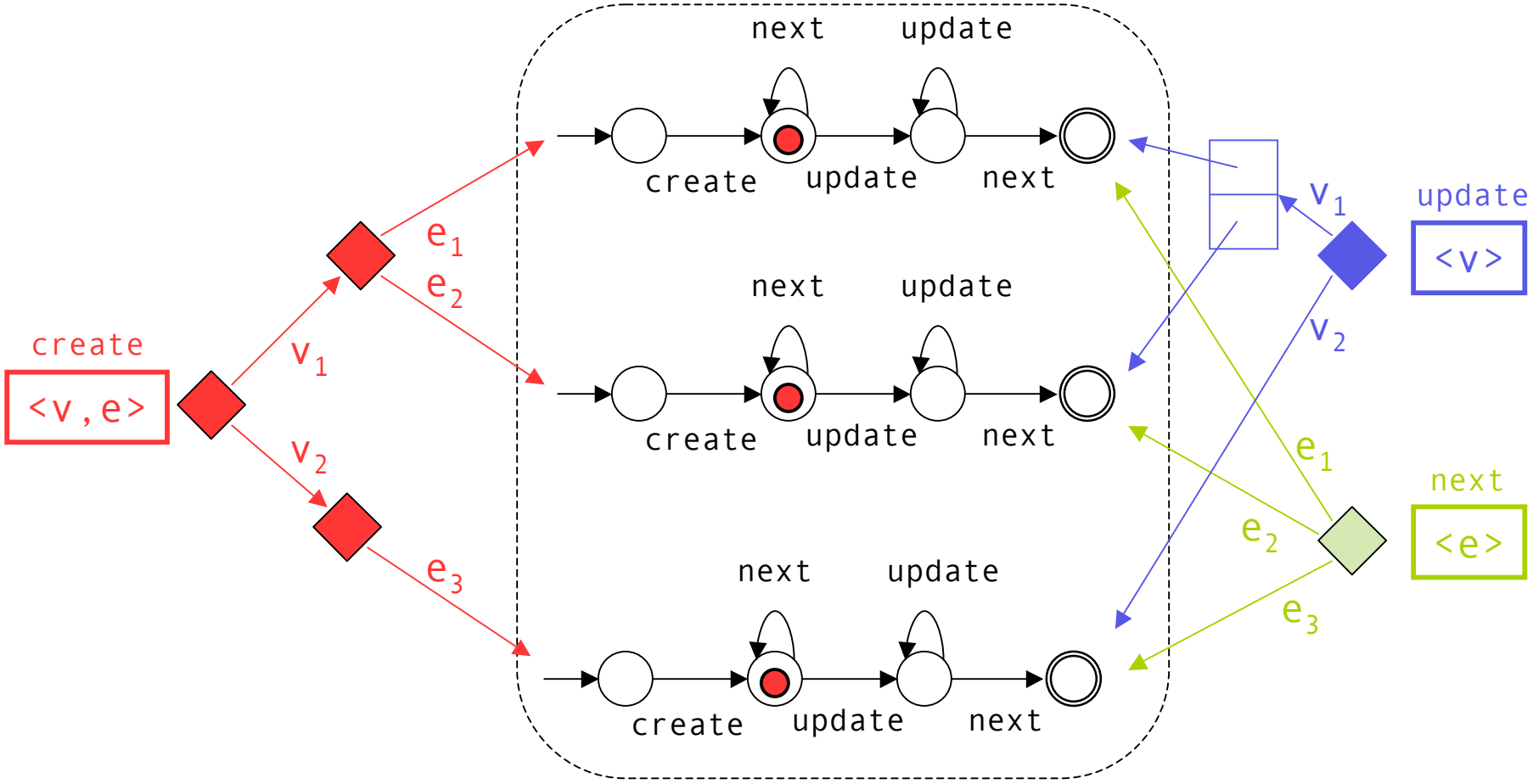
Indexed implementation

```
Vector v1 = new Vector();
Vector v2 = new Vector();
v1.add(1); v1.add(2); v2.add(4); v2.add(5);
Enumeration e1 = v1.elements();
Enumeration e2 = v1.elements();
Enumeration e3 = v2.elements();
while(e1.hasMoreElements())print(e1.nextElement());
v1.add(99);
while(e2.hasMoreElements())print(e2.nextElement());
while(e3.hasMoreElements())print(e3.nextElement());
```



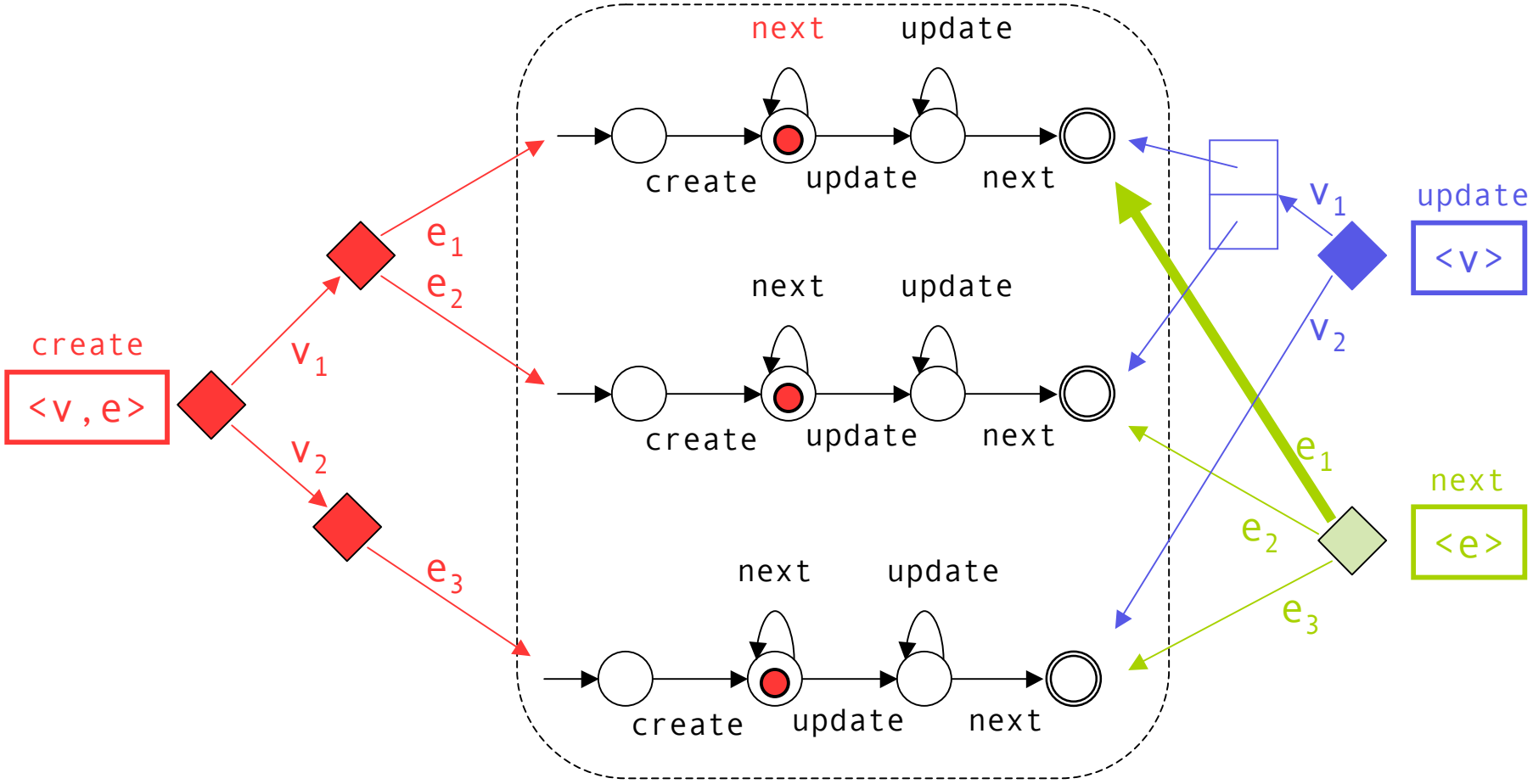
Indexed implementation

```
Vector v1 = new Vector();
Vector v2 = new Vector();
v1.add(1); v1.add(2); v2.add(4); v2.add(5);
Enumeration e1 = v1.elements();
Enumeration e2 = v1.elements();
Enumeration e3 = v2.elements();
while(e1.hasMoreElements())print(e1.nextElement());
v1.add(99);
while(e2.hasMoreElements())print(e2.nextElement());
while(e3.hasMoreElements())print(e3.nextElement());
```



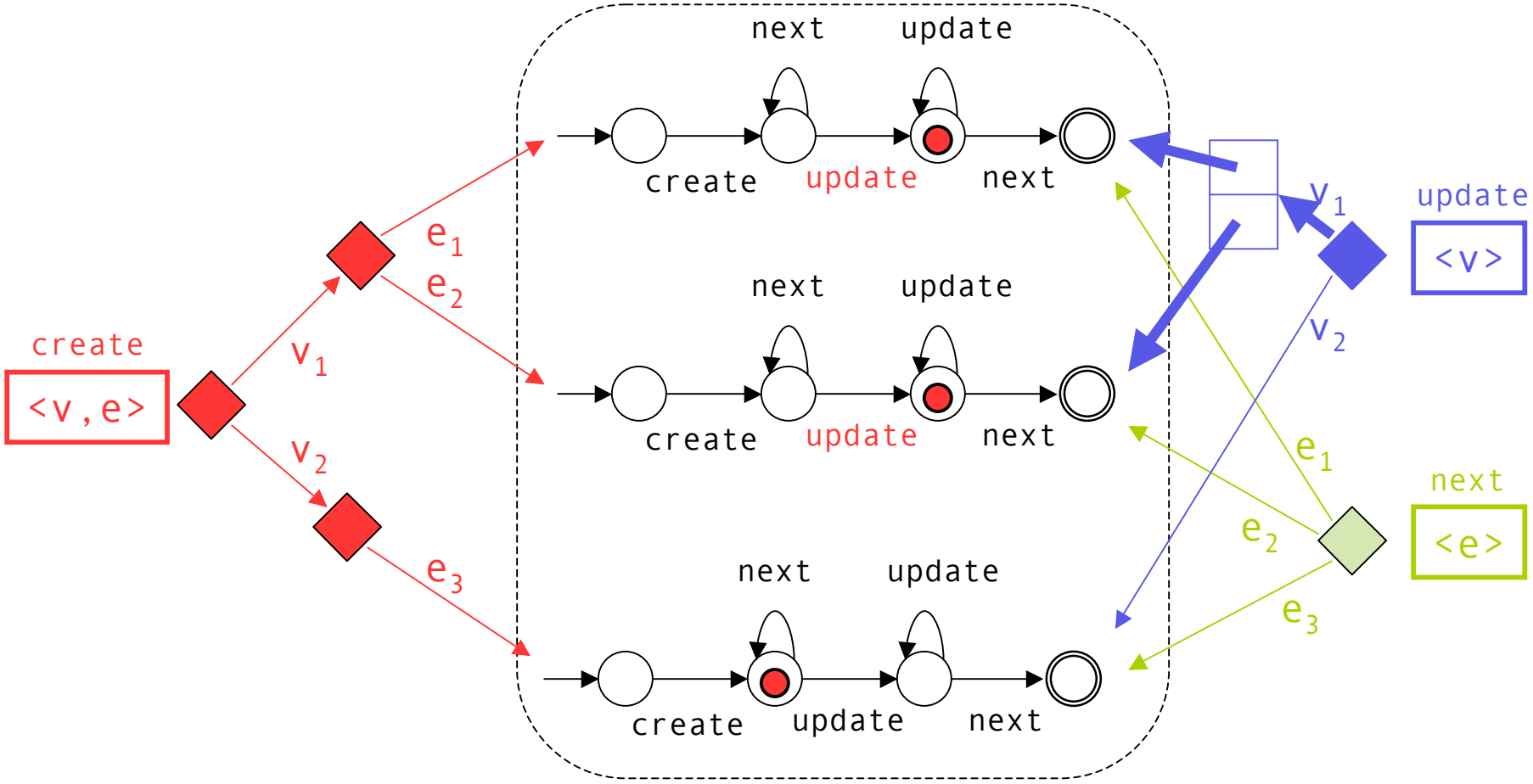
Indexed implementation

```
Vector v1 = new Vector();
Vector v2 = new Vector();
v1.add(1); v1.add(2); v2.add(4); v2.add(5);
Enumeration e1 = v1.elements();
Enumeration e2 = v1.elements();
Enumeration e3 = v2.elements();
while(e1.hasMoreElements())print(e1.nextElement());
v1.add(99);
while(e2.hasMoreElements())print(e2.nextElement());
while(e3.hasMoreElements())print(e3.nextElement());
```



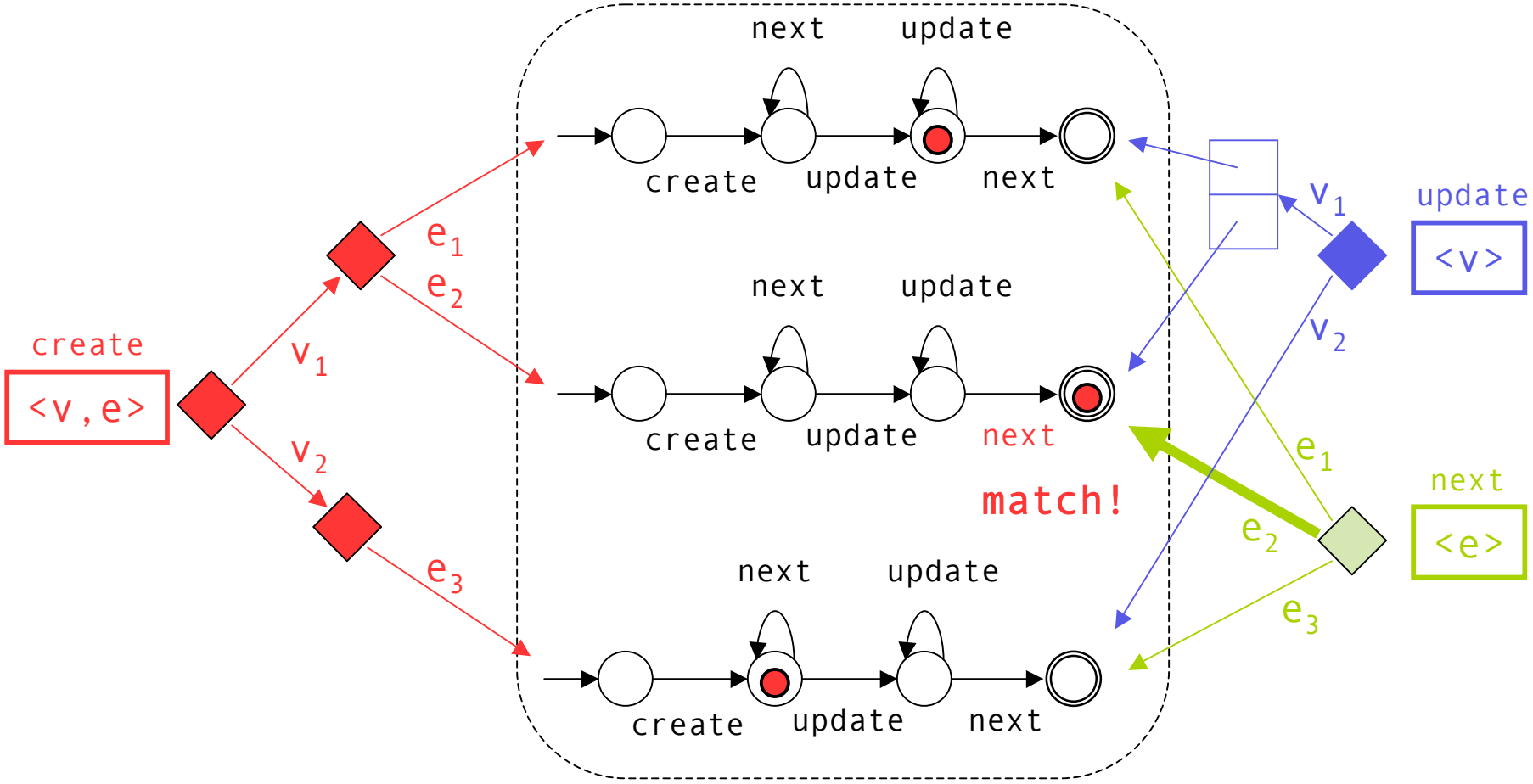
Indexed implementation

```
Vector v1 = new Vector();
Vector v2 = new Vector();
v1.add(1); v1.add(2); v2.add(4); v2.add(5);
Enumeration e1 = v1.elements();
Enumeration e2 = v1.elements();
Enumeration e3 = v2.elements();
while(e1.hasMoreElements())print(e1.nextElement());
v1.add(99);
while(e2.hasMoreElements())print(e2.nextElement());
while(e3.hasMoreElements())print(e3.nextElement());
```



Indexed implementation

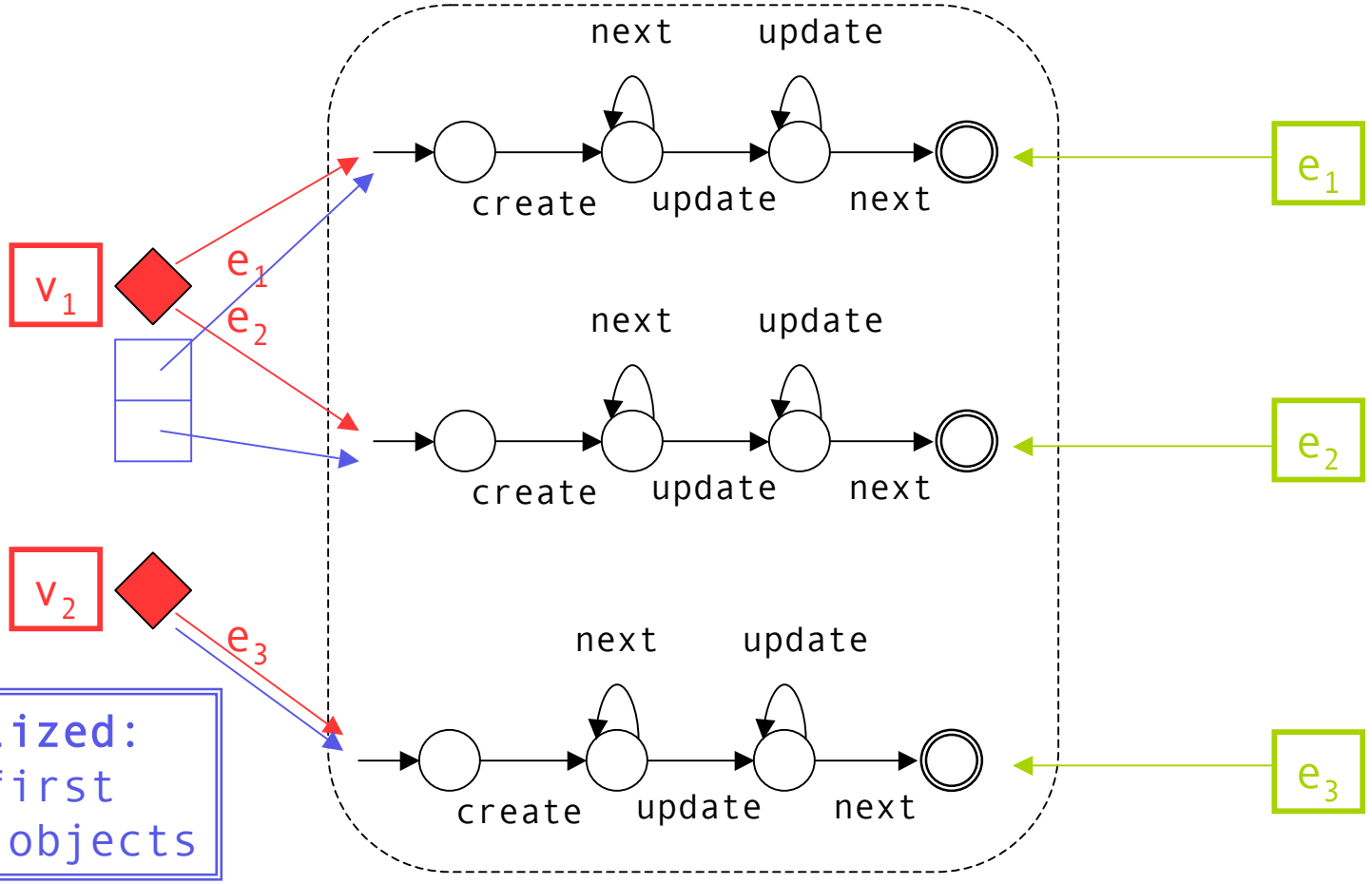
```
Vector v1 = new Vector();
Vector v2 = new Vector();
v1.add(1); v1.add(2); v2.add(4); v2.add(5);
Enumeration e1 = v1.elements();
Enumeration e2 = v1.elements();
Enumeration e3 = v2.elements();
while(e1.hasMoreElements())print(e1.nextElement());
v1.add(99);
while(e2.hasMoreElements())print(e2.nextElement());
while(e3.hasMoreElements())print(e3.nextElement());
```



```
Vector v1 = new Vector();
Vector v2 = new Vector();
v1.add(1); v1.add(2); v2.add(4); v2.add(5);
Enumeration e1 = v1.elements();
Enumeration e2 = v1.elements();
Enumeration e3 = v2.elements();
while(e1.hasMoreElements())print(e1.nextElement());
v1.add(99);
while(e2.hasMoreElements())print(e2.nextElement());
while(e3.hasMoreElements())print(e3.nextElement());
```

events:
create<v,e>
update<v>
next<e>

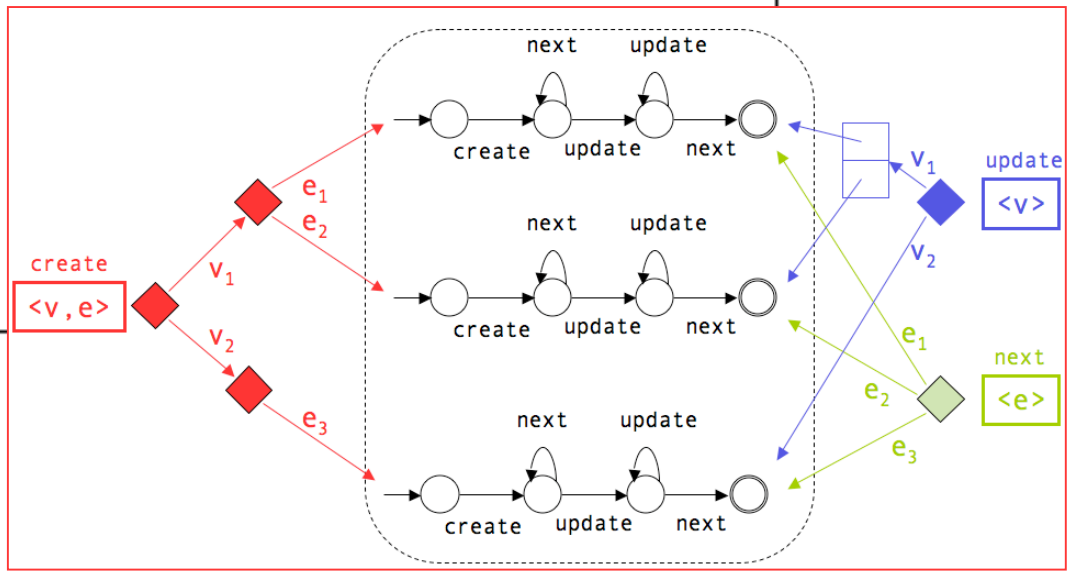
monitor creation event



decentralized:
storing first
index in objects

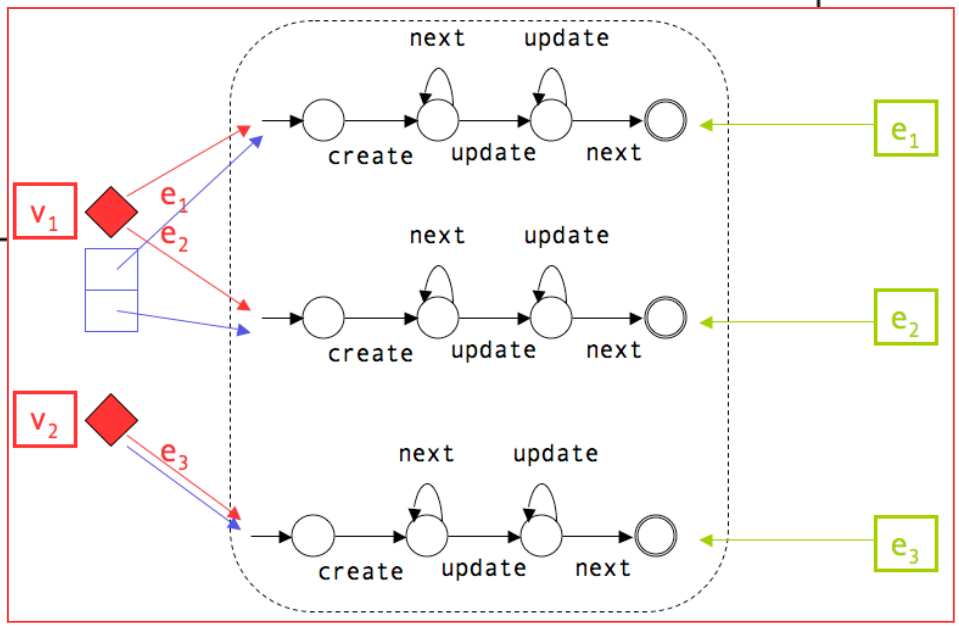
aspect for centralized solution

```
Map SafeEnum_v_map = makeMap();
pointcut SafeEnum_updatesource0(Vector v) :
                call(* Vector.add*(..))&& target(v);
after (Vector v) : SafeEnum_updatesource0(v) {
  Map m = SafeEnum_v_map;
  Object obj = null;
  obj = m.get(v);
  if (obj != null){
    Iterator monitors = ((List)obj).iterator();
    while (monitors.hasNext()) {
     SafeEnumMonitor monitor=(SafeEnumMonitor)monitors.next();
     monitor.updatesource(v);
     if (monitor.suceeded()) {
       //validation handler
     }
    }//end of while
  }//end of if
}
```



aspect for decentralized solution

```
List Vector.SafeEnum_v_List = null;
pointcut SafeEnum_updatesource0(Vector v) :
                call(* Vector.add*(..))&& target(v);
after (Vector v) : SafeEnum_updatesource0(v) {
  if (v.SafeEnum_v_List != null) {
    Iterator monitors = (v.SafeEnum_v_List).iterator();
    while (monitors.hasNext()) {
      SafeEnumMonitor monitor=(SafeEnumMonitor)monitors.next();
      monitor.updatesource(v);
      if (monitor.suceeded()) {
        //validation handler
      }
    }//end of while
  }
}
```



properties of Java library APIs

R_3: An collection should not be modified while it is a member of a hashset (don't change the hashcode).

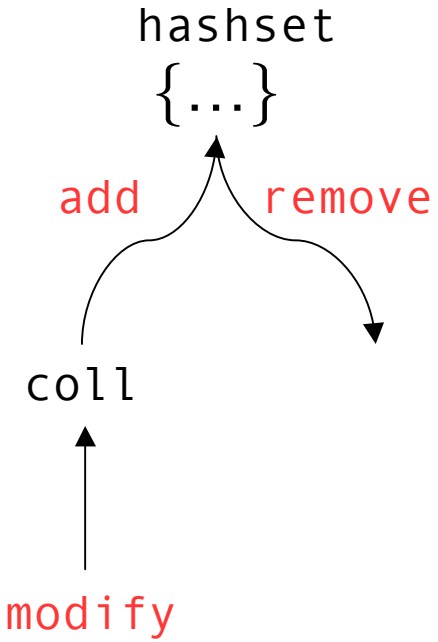
java.util
Class HashSet<E>

of

java.lang.Object
└ java.util.AbstractCollection<E>
└ java.util.AbstractSet<E>
└ java.util.HashSet<E>

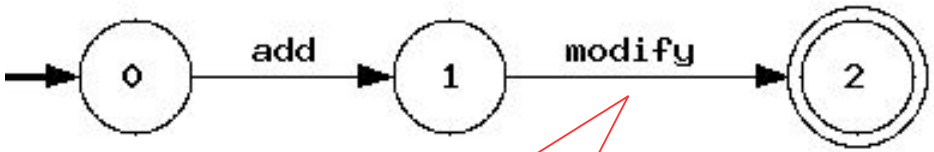
java.util
Interface Collection<E>

R_3: An collection should not be modified while it is a member of a hashset (don't change the hashcode).



add	collection to hashset
remove	collection from hashset
modify	collection

add modify



note: without having seen a remove

adding collection to hash set and then updating it

```
HashSet s = new HashSet();
Collection c = new ArrayList();
c.add("this is ok");
s.add(c);
System.out.println(s.contains(c));
c.add("this is not ok");
System.out.println(s.contains(c));
```

bad update changing hashcode of collection c

subsequently s.contains(c) yields false

specification 1 : hashset of collections

```
/*@
partial centralized
scope = global
logic = ERE
HashSet (HashSet t, Collection+ c){
  event put <t, c> : end(call(* t.add(c)));
  event remove <t, c>: end(call(* t.remove(c)));
  event modify <c> :
     begin(call(* c.add(..)))    \/
     begin(call(* c.addAll(..))) \/
     begin(call(* c.remove(..))) \/
     begin(call(* c.removeAll(..)));
  formula : put modify
}

validation handler {
  System.out.println("collection updated while in hashset");
}
@*/
```

watch out.
should be:
… add(Object)
… remove(Object)

will not work

specification 2 : hashset of any objects

```
/*@
partial centralized
scope = global
logic = ERE
HashSet (HashSet t, Object o){
  [int hashcode=0;]
  event put <t, o> : end(call(* t.add(o))) {hashcode = o.hashCode();};
  event remove <t, o>: end(call(* t.remove(o)));
  event contains <t, o> : begin(call(* t.contains(o)));
  formula : put contains contains*
}

validation handler {
  if (@MONITOR.hashcode != o.hashCode())
    System.out.println("HashCode changed");
  }
@*/
```

local variable

updated on put events

tested in handler

Reader (Java 2 Platform SE 5.0)
Method Summary
close()
Close the stream.
mark(int readAheadLimit)
Mark the present position in the stream.
markSupported()
Tell whether this stream supports the mark() operation.
read()
Read a single character.
java.io
Class Reader
java.lang.Object
R4: If a Reader is created on top of an Inputstream, none of the two may be read from if one (even the other) is closed .
InputStream (Java 2 Platform SE 5.0)
Overview Package Class Use Tree Deprecated Index Help
PREV CLASS NEXT CLASS
FRAMES NO FRAMES
SUMMARY: NESTED | FIELD | CONSTR | METHOD
DETAIL: FIELD | CONSTR | METHOD
java.io
Class InputStream
java.lang.Object
java.io.InputStream
All Implemented Interfaces:
Closeable
Direct Known Subclasses:
AudioInputStream, ByteArrayInputStream, FilterInputStream, InputStream, ObjectInputStream, PipedInputStream, SequenceInputStream, StringBufferInputStream
public abstract class InputStream
extends Object
implements Closeable
This abstract class is the superclass of all classes representing an input stream of bytes.
Method Summary
available()
Returns the number of bytes that can be read (or skipped over) from this input stream without blocking by the next caller of a method for this input stream.
close()
Closes this input stream and releases any system resources associated with the stream.
mark(int readlimit)
Marks the current position in this input stream.
markSupported()
Tests if this input stream supports the mark and reset methods.
read()
Reads the next byte of data from the input stream.

something one should not do:

- create reader from input stream,
- close down reader,
- continue reading from inputstream

```
try{
  InputStream i = new StringBufferInputStream("AB");
  InputStreamReader r = new InputStreamReader(i);
  System.out.println((char)r.read());
  r.close();
  System.out.println((char)i.read());
} catch(IOException e) {
  System.out.println("*** io error");
}
```

closing down reader

reading inputstream

output:

A

?

specification

```
/*@
partial centralized
scope = global
logic = ERE

ClosedReaderI(Reader+ r, InputStream+ s)
{
  event create<r, s> : end(call(Reader+.new(s, ..))) with(r);
  event closeS<s> : end(call(* s.close()));
  event closeR<r> : end(call(* r.close()));
  event readS<s> : begin(call(* s.read*(..)));
  event readR<r> : begin(call(* r.read*(..)));

  formula : create (readS + readR)* (closeS + closeR) (readS + readR)
}

validation handler{
    System.out.println("one input source has been closed");
}
@*/
```

output:
A
one input source has been closed
?

specification on total trace + profiling

```
/*@
centralized
scope = global
logic = ERE
ClosedReaderI(Reader+ r, InputStream+ s)
{
  [static int createcount=0; int readcount = 0;]
  event create<r, s> : end(call(Reader+.new(s, ..))) with(r) {createcount++;};
  event closeS<s> : end(call(* s.close()));
  event closeR<r> : end(call(* r.close()));
  event readS<s> : begin(call(* s.read*(..)));
  event readR<r> : begin(call(* r.read*(..))) {readcount++;};

  formula : (create readR* ((closeS closeR)+(closeR closeS)))*
}

violation handler{
    System.out.println("pattern " + @MONITOR.createcount +
        " violated after reading " + @MONITOR.readcount);
}
@*/
```

no longer partial

counters: one static, common for all monitors and one for each monitor

property over total trace. we require only read by Reader.

reference to monitor contents

<Specification> ::= /*@ <Header> <Body> <Handlers> @*/

<Header> ::= <Attribute>*[**scope** =<Scope>][**logic** =<Logic>]
<Attribute> ::= **static** | **outline** | **offline** | **centralized** | **partial**
<Scope> ::= **global** | **class** [(<Name>)] | **interface** | **method**
<Logic> ::= **ERE** | **CFG** | **PTLTL** | **FTLTL** | **JML** | **RAW** | …

<Body> ::= [<Name>][(<Parameters>)]{<LogicBody>}
<Parameters> ::= (<Type> <Identifier>)+
<LogicBody> ::= [[<VarDeclaration>]]<Event>* [<Formula>]
<Event> ::= <EventHeader>:<EventDecl> [{<Statement>}];
<EventHeader> ::= **event**<Identifier>[<<Parameters>>]
<EventDecl> ::= <EventPoint>[**with**(<Type> <Name>)][&& <BExp>]
<EventPoint> ::= (**begin** | **end**)(<EventPattern>)
<EventPattern> ::= (**call**|**exec**)(<Method>) | **update**(<Field>)
<Formula> ::= **formula** : <Formula>

<Handlers> ::= [<ViolationHandler>] [<ValidationHandler>]
<ViolationHandler> ::= **validation handler** {<Statements>}
<ValidationHandler> ::= **validation handler** {<Statements>}

generic syntax

ERE syntax

<Formula> ::= **empty** | **epsilon** | a | ~ <Formula> | <Formula> *
| <Formula> <Formula> | <Formula> + <Formula> | <Formula> & <Formula>

MOP: Evaluation

- More than 100 program-property pairs
  - Dacapo benchmark, Tracematches benchmark, Eclipse, …
- Overhead < 8% in most cases; close to hand-optimized

	SafeEnum		SafeIterator		HashMap		HasNext		LeakingSync		ClosedReader	
antlr	0.0	1.5	0.0	0.0	0.0	1.1	0.4	0.0	0.0	0.0	5.8	0.0
chart	0.0	0.0	0.0	0.0	3.6	4.8	0.0	0.0	0.5	0.0	0.0	0.0
eclipse	4.1	2.8	0.0	1.4	3.7	0.5	3.8	1.5	3.0	3.1	2.2	2.4
fop	1.2	0.6	1.5	0.0	0.0	0.0	0.8	1.5	0.5	1.0	0.0	0.0
hsqldb	3.3	0.0	0.9	1.2	0.0	2.1	0.8	0.0	1.4	1.4	0.0	0.0
jython	0.6	0.0	0.8	0.5	0.2	0.3	0.0	0.6	0.0	2.3	0.4	0.2
luindex	1.6	0.2	1.9	0.5	1.2	1.8	0.3	0.0	3.2	2.2	1.7	1.1
lusearch	0.5	0.0	0.0	0.0	0.0	0.0	0.3	0.0	1.1	0.6	0.0	0.1
pmd	0.0	0.0	44.8	11.3	0.0	0.0	25.4	13.7	5.4	8.0	0.0	0.0
xalan	3.5	4.4	6.7	5.4	4.7	6.5	0.0	2.8	1.5	1.7	2.2	4.5

Overhead in % MOP monitors VS. hand-optimized monitors

MOP: Evaluation (cont.)

- Even significantly faster than logic specific solutions

Property	Program	LOC	Hand Optimized	MOP	Tracematches	PQL
Listener	ajHotDraw	21.1K	0	6.6	354	2193
SafeEnum	jHotDraw	9.5K	0.1	136	1509	7084
NullTrack	CerRevSim	1.4K	210	232	452	N/A
Hashtable	Weka	9.9K	3.3	3.3	15.2	N/A
HashSet	Aprove	438.7K	21.2	23.9	124.3	N/A
Reweave	ABC	51.2K	11.1	20.2	63.5	N/A

Results for Tracematches benchmarks, Overhead in %

Limitations of the Current Implementation of JavaMOP

- No nested or existential parameters
- Monitor creating events must contain all the parameters of the specification
- No support for real-time

end