Verifying Security of Web Applications

Marius Minea

joint work with Petru Florin Mihancea

Politehnica University of Timișoara
Romania

Clarke Symposium, CMU
September 19, 2014

Verifying security is important

Model checking can be done for:

- hand-written models
- binaries of actual implementations
- *models extracted from applications* (source/byte-code)

This talk: extracting models from JSP web applications

Extracting Models from Web Applications

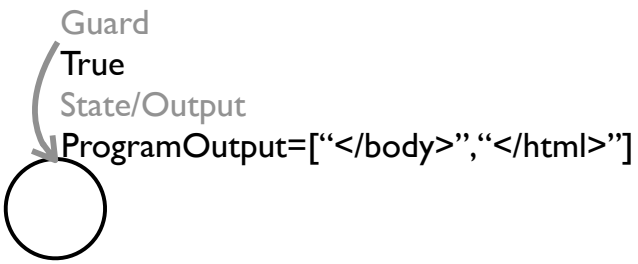
Tool: *jModex* [Mihancea & M., CSMR-WCRE'14]
tailored to applications written using JavaServer Pages

input: user request parameters, e.g., {(name, pepi), (age, 33)}
must handle *sets* of pairs (strings)
output: generated HTML
state: session attributes, often databases

$\Rightarrow$ translate JSP code into extended finite state machine (EFSM)

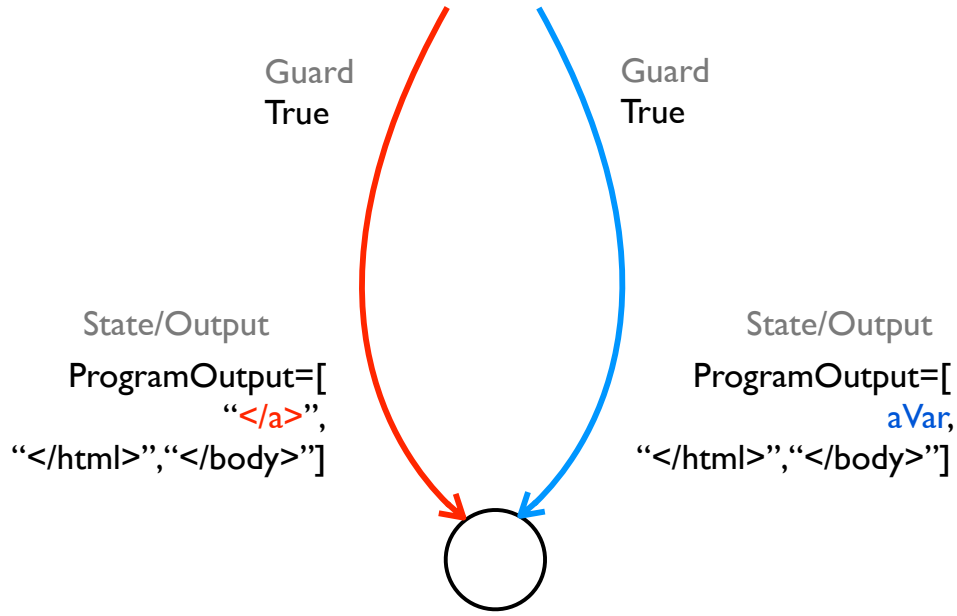
EFSM Building

```
void _jspService(...) { // A component entrypoint
  String aVar = “No Link!”;
  out.println("<html>");
  out.println("<body>");
  if(request.getParameter("update").equals("false")) {
      out.println(aVar);
   } else {
      request.getSession().setAttribute("seen", "true");
      out.println("<a href=\"B.jsp?name=exec\"");
      out.println("Click here!");
      out.println("</a>");
   }
  out.println("</body>");
  out.println("</html>");
}
```



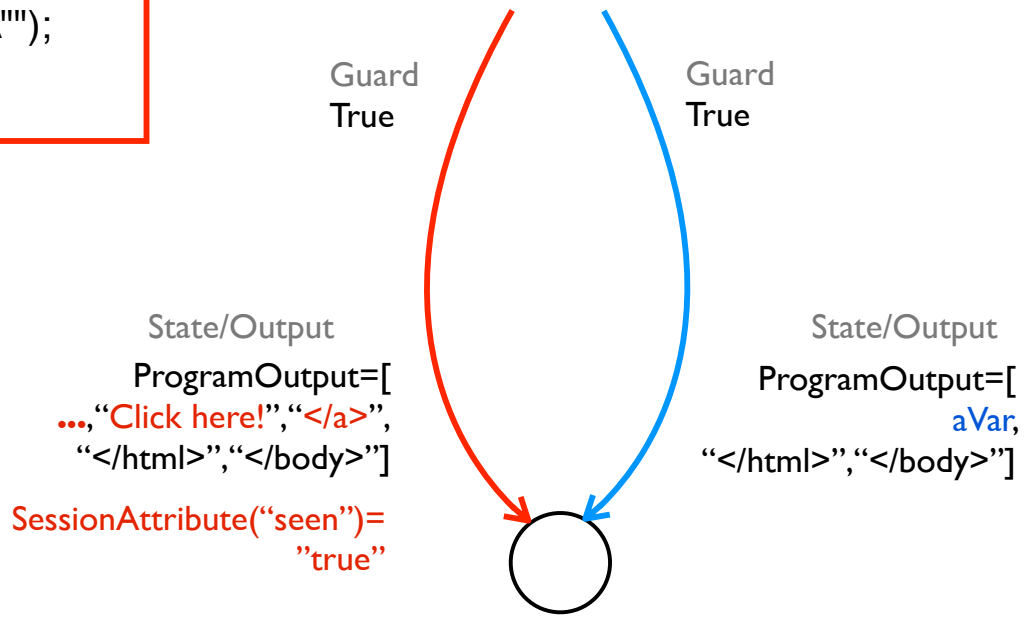
EFSM Building

```
void _jspService(...) { // A component entrypoint
  String aVar = “No Link!”;
  out.println("<html>");
  out.println("<body>");
  if(request.getParameter("update").equals("false")) {
     out.println(aVar);
   } else {
      request.getSession().setAttribute("seen", "true");
      out.println("<a href=\"B.jsp?name=exec\"");
      out.println("Click here!");
      out.println("</a>");
   }
   out.println("</body>");
   out.println("</html>");
}
```



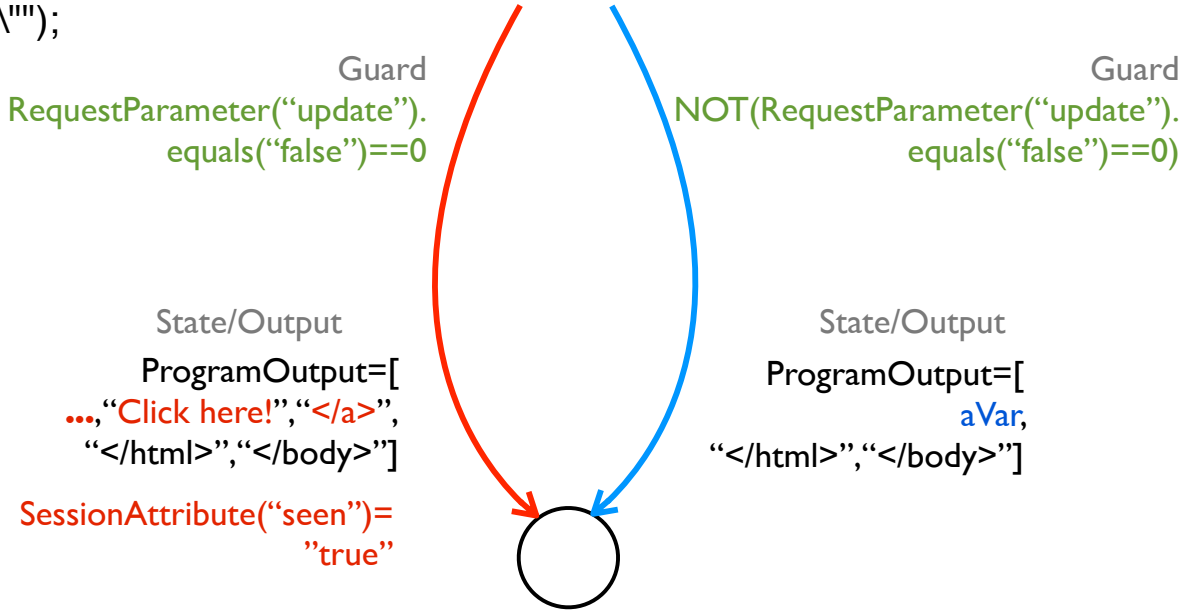
EFSM Building

```
void _jspService(...) { // A component entrypoint
  String aVar = “No Link!”;
  out.println("<html>");
  out.println("<body>");
  if(request.getParameter("update").equals("false")) {
     out.println(aVar);
   } else {
     request.getSession().setAttribute("seen", "true");
     out.println("<a href=\"B.jsp?name=exec\"");
     out.println("Click here!");
     out.println("</a>");
   }
   out.println("</body>");
   out.println("</html>");
}
```



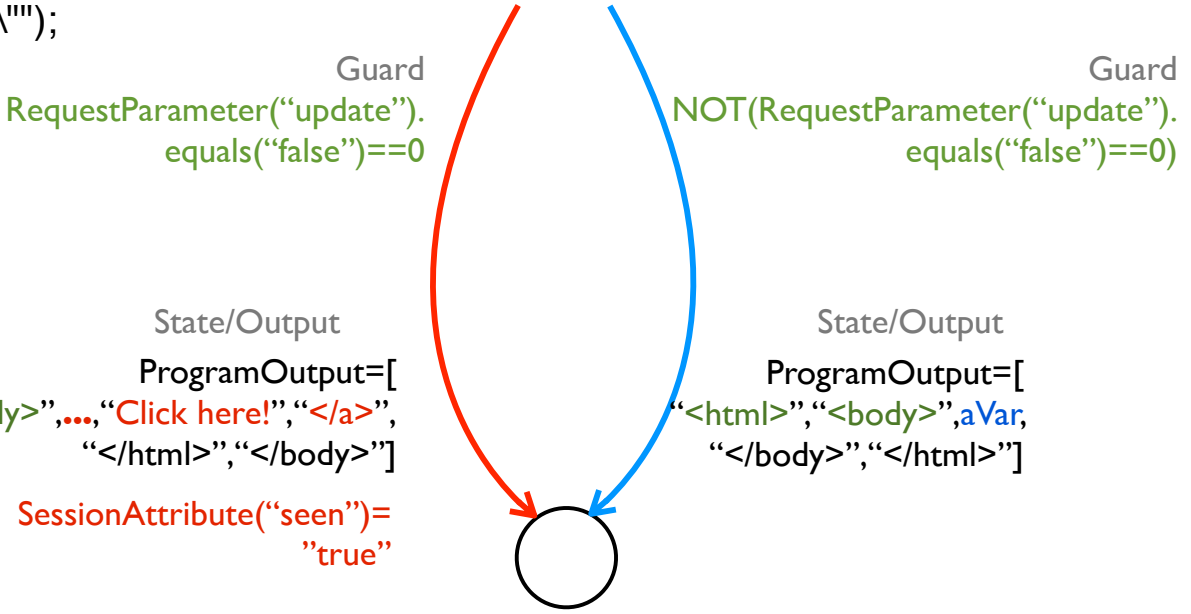
EFSM Building

```
void _jspService(...) { // A component entrypoint
  String aVar = “No Link!”;
  out.println("<html>");
  out.println("<body>");
  if(request.getParameter("update").equals("false")) {
      out.println(aVar);
   } else {
      request.getSession().setAttribute("seen", "true");
      out.println("<a href=\"B.jsp?name=exec\"");
      out.println("Click here!");
      out.println("</a>");
   }
   out.println("</body>");
   out.println("</html>");
}
```



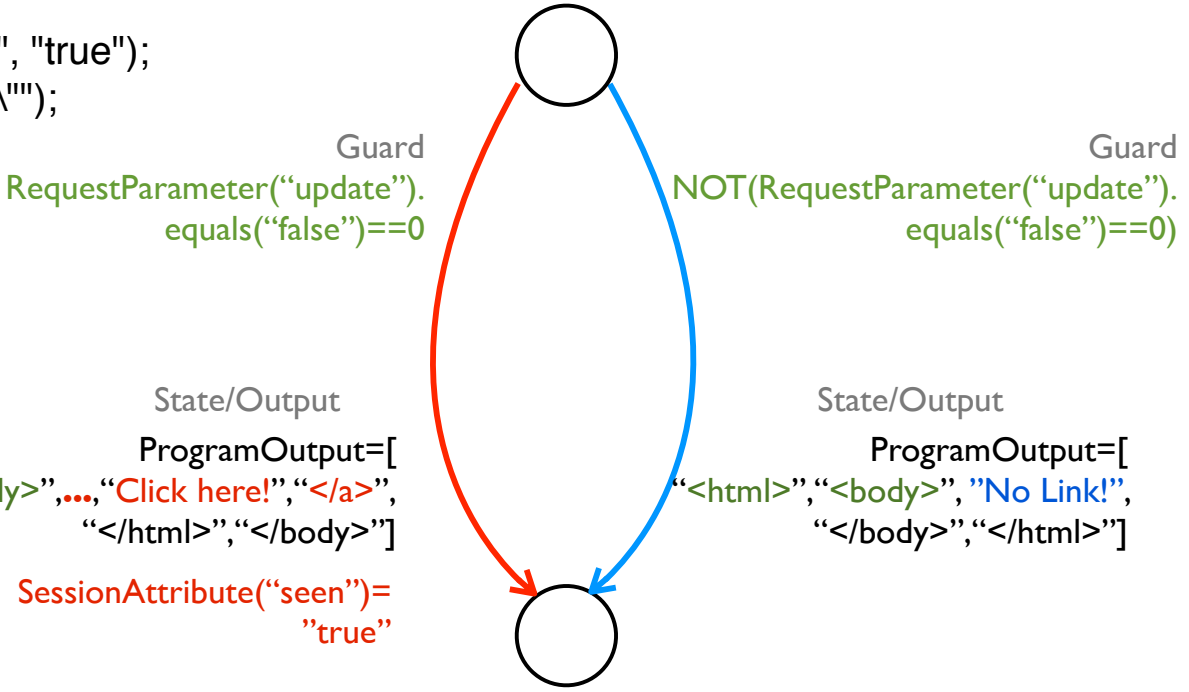
EFSM Building

```
void _jspService(...) { // A component entrypoint
  String aVar = “No Link!”;
  out.println("<html>");
  out.println("<body>");
  if(request.getParameter("update").equals("false")) {
      out.println(aVar);
   } else {
      request.getSession().setAttribute("seen", "true");
      out.println("<a href=\"B.jsp?name=exec\"");
      out.println("Click here!");
      out.println("</a>");
   }
   out.println("</body>");
   out.println("</html>");
}
```



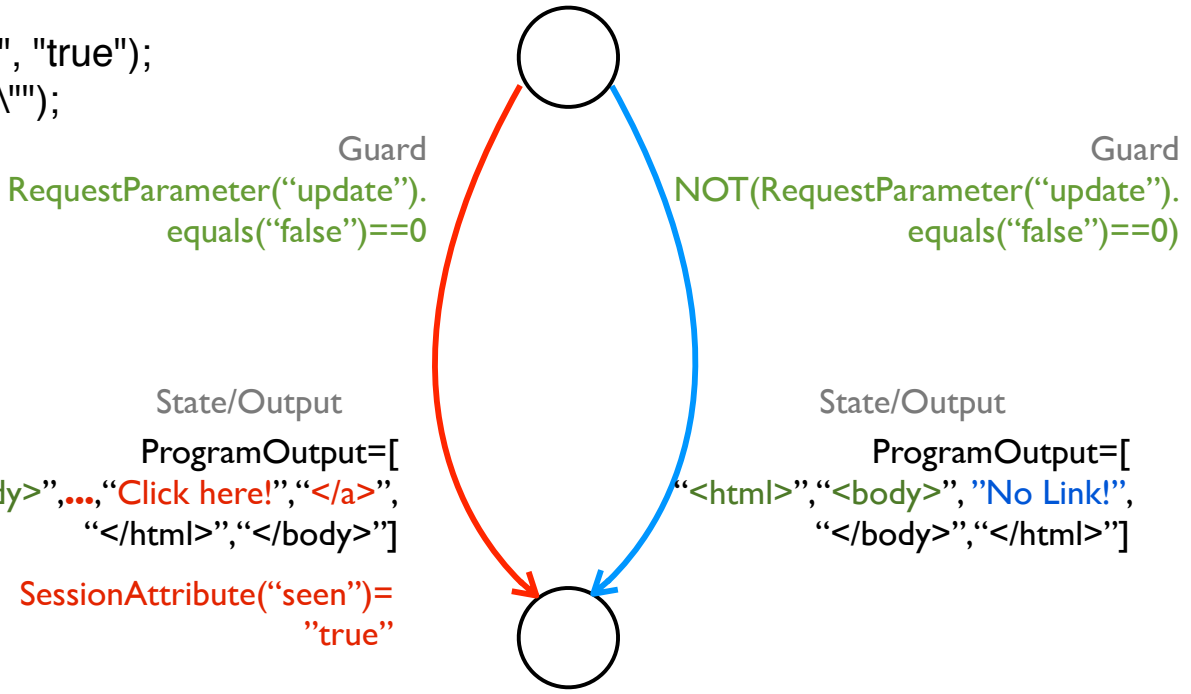
EFSM Building

```
void _jspService(...) { // A component entrypoint
  String aVar = “No Link!”;
  out.println("<html>");
  out.println("<body>");
  if(request.getParameter("update").equals("false")) {
      out.println(aVar);
   } else {
      request.getSession().setAttribute("seen", "true");
      out.println("<a href=\"B.jsp?name=exec\"");
      out.println("Click here!");
      out.println("</a>");
   }
   out.println("</body>");
   out.println("</html>");
}
```



EFSM Building

```
void _jspService(...) { // A component entrypoint
  String aVar = “No Link!”;
  out.println("<html>");
  out.println("<body>");
  if(request.getParameter("update").equals("false")) {
      out.println(aVar);
   } else {
      request.getSession().setAttribute("seen", "true");
      out.println("<a href=\"B.jsp?name=exec\"");
      out.println("Click here!");
      out.println("</a>");
   }
   out.println("</body>");
   out.println("</html>");
}
```



Inter-procedural

recursion not supported

Loops

state node for each loop header

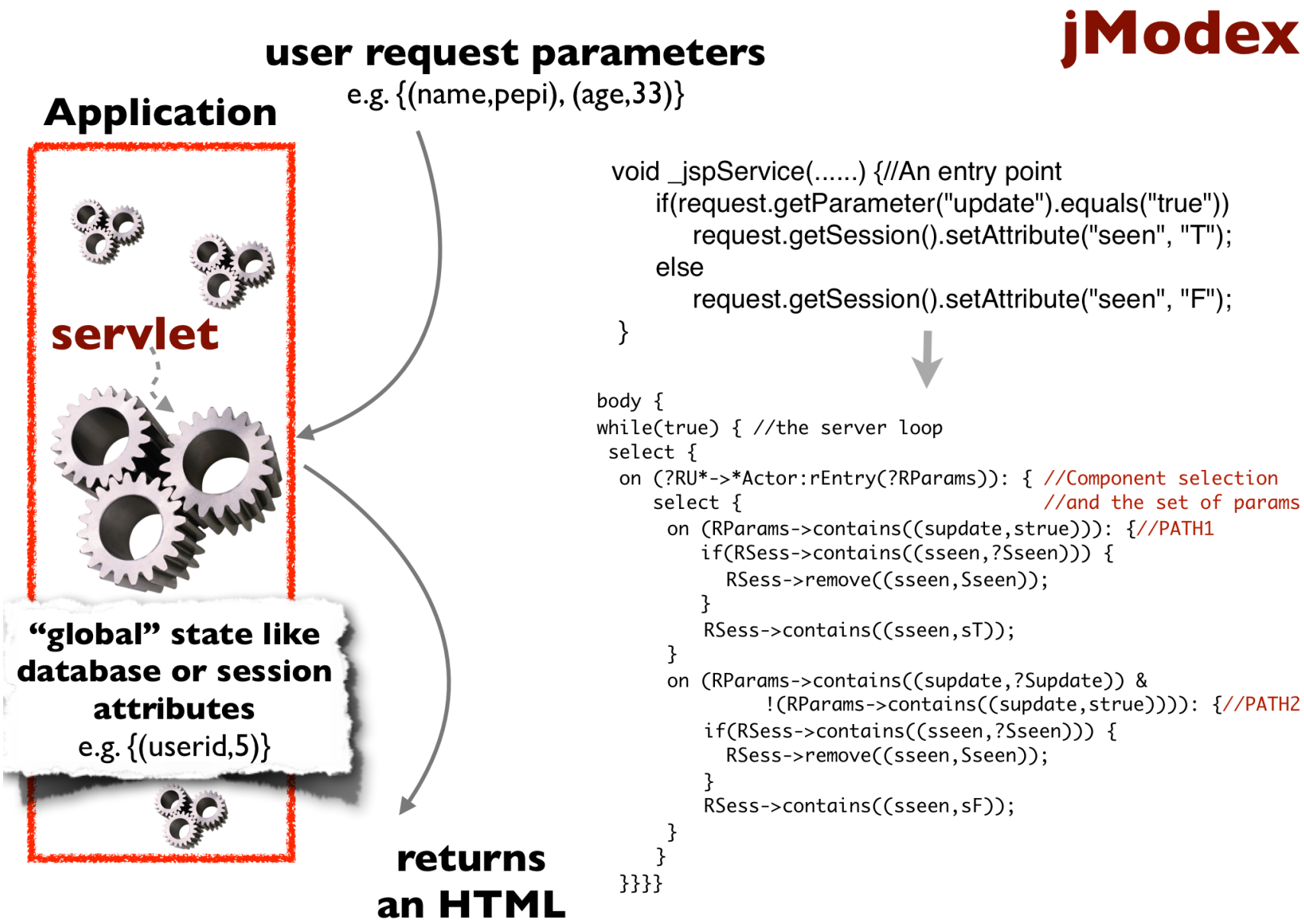
From EFSM to ASLan++ and Model Checking

jModex designed for specific language and model checkers
part of the SPaCIoS FP7 project `www.spacios.eu`
(Secure Provision and Consumption in the Internet of Services)

ASLan: low-level language with semantics based on term-rewriting
incorporating Doley-Yao intruder model
ASLan++: higher-level language, closer to usual imperative syntax

Model checkers:
CL-AtSe (Loria-INRIA), constraint logic
OFMC (IBM/ETHZ), on-the-fly
SATMC (U. Genova), sat-based

Model extraction in a nutshell



Special jModex features

Model *database operations*

needed since many applications use SQL

supports essential subset (select, update, delete, etc.)

User-specified *abstractions*

e.g. functions that can be ignored (identity)

critical to reduce model size, dependent on analysis goal

Extensible

not hard-coded for JSP, adaptable to other technologies

Case Study: GotoCode Bookstore

open-source, 28 JSPs, 22 kLOC Java code, several security checks

... and one flaw: can change any profile (incl. admin password)

Experimental Results

Simplifications used:
- ignored HTML output (irrelevant for control flow)
- ignored some sanitizing functions, assumed all parameters exist

Full bookstore code (22 kLOC)
- 90 seconds for model construction
- 600 lines ASLan++ model, 171 transition rules

For targeted flaw (change admin password):
- 2 components only, 140 lines ASLan++, 22 transition rules
- CL-AtSe model checker *finds flaw in 8 sec.*
- after code correction, model checker shows absence of flaw

Conclusions

Building analyzable models of web applications is possible
by *adapting translation to specific technologies* (e.g., JSP)

Flaws in actual applications can be found/confirmed.

Abstraction is key for obtaining models that can be handled

Next steps:

extend language support (e.g. exceptions)
more abstraction (selecting components, predicate abstraction)
optimize representation (working with sets, strings, etc.)