KDM Analytics™

Model Driven Solutions
Where Business Meets Technology
A division of Data Access Technologies, Inc.

Tool Output Integration Framework (TOIF)

DHS SBIR Project briefing

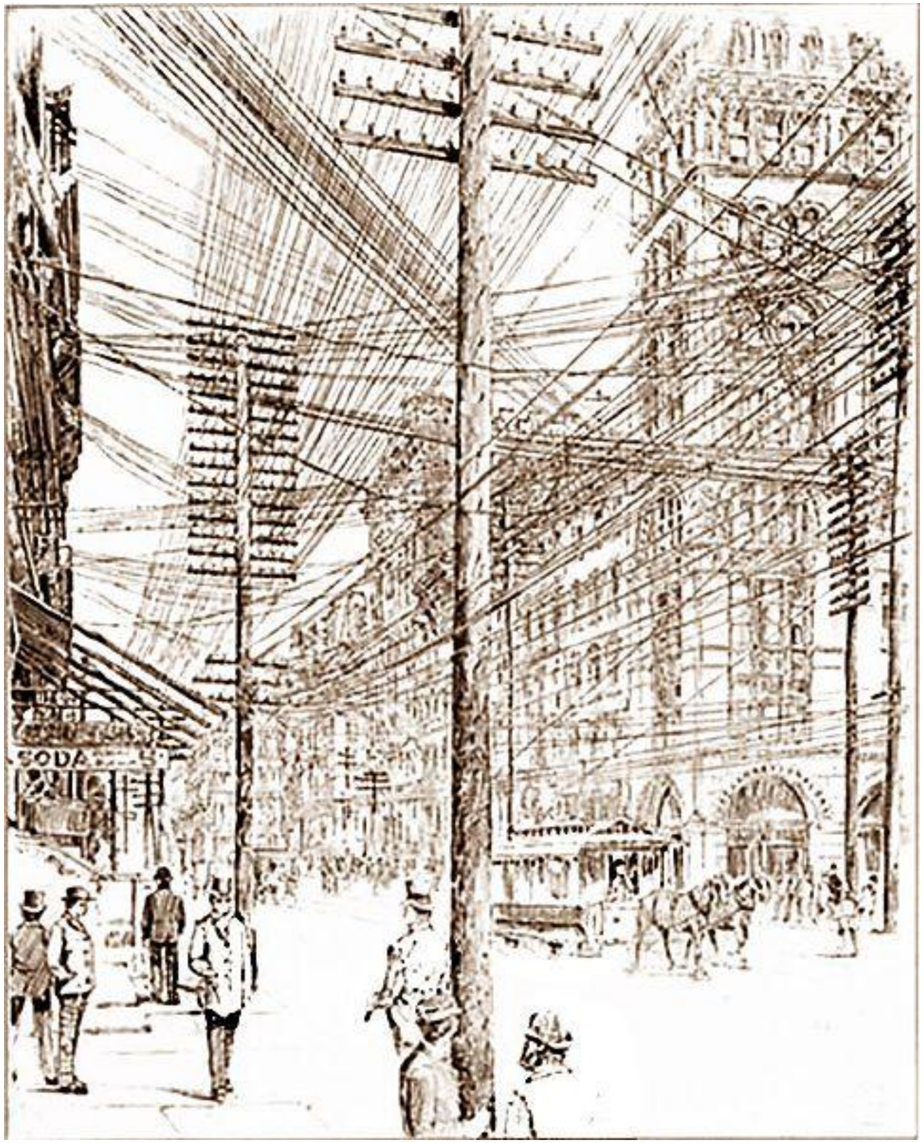
Djenana Campara, KDM Analytics Inc.

SwA WG meeting, December 15th 2010, Washington, DC

SBIR topic: Software Testing and Vulnerability Analysis

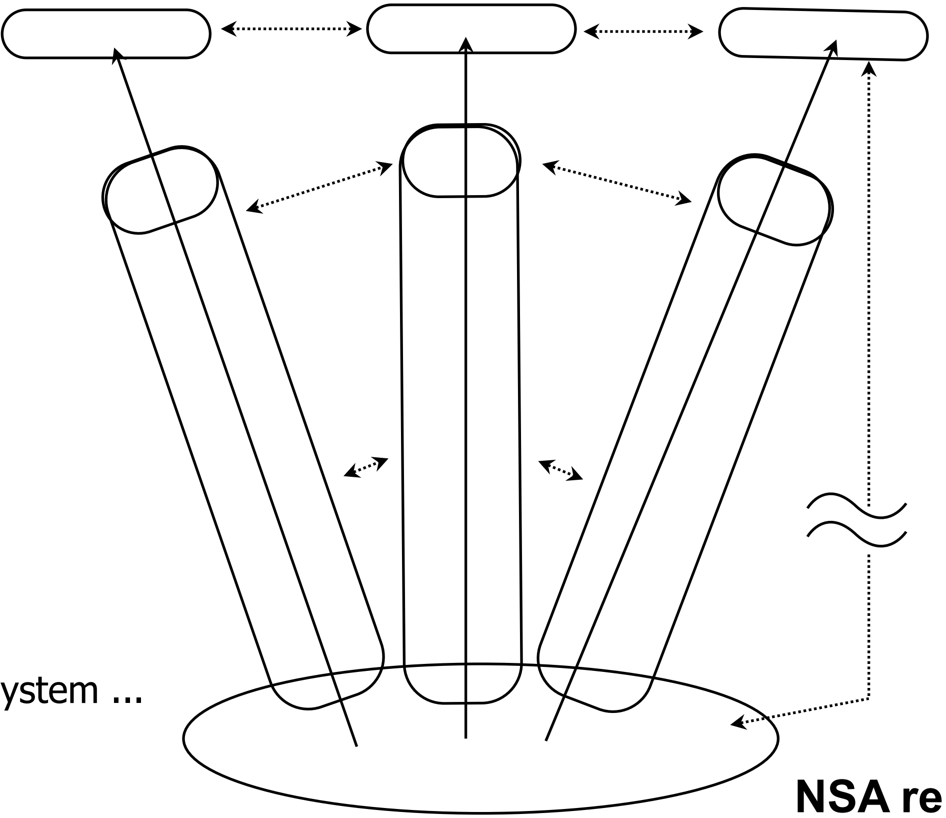
- Problem
  - Effective and systematic measurement of the risks posed by software vulnerabilities
- Challenge
  - One of the key challenges is that analysis solution consists of multiple tools, information sources and services that are currently fragmented lacking intuitive and efficient integration due to
    - Inconsistency in the nomenclature of reported vulnerabilities caused by ambiguity of vulnerability definitions (inconsistency in interpretation of CWE instances)
    - Lack of agreement on what are the parts of vulnerability to report – what constitutes vulnerability report
    - Lack of interoperability that is based on common definition of system artifacts

Integration issues



The problem

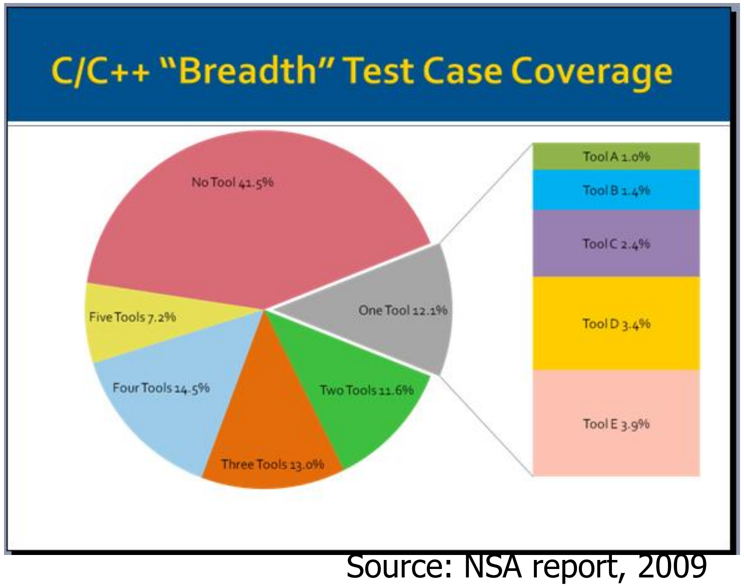
... incompatible vulnerability findings



same system ...

Source or machine code

Traditional vulnerability analysis and testing tools are built as "silos" making it difficult to correlate findings



Source: NSA report, 2009

NSA reported 84% non overlapping results

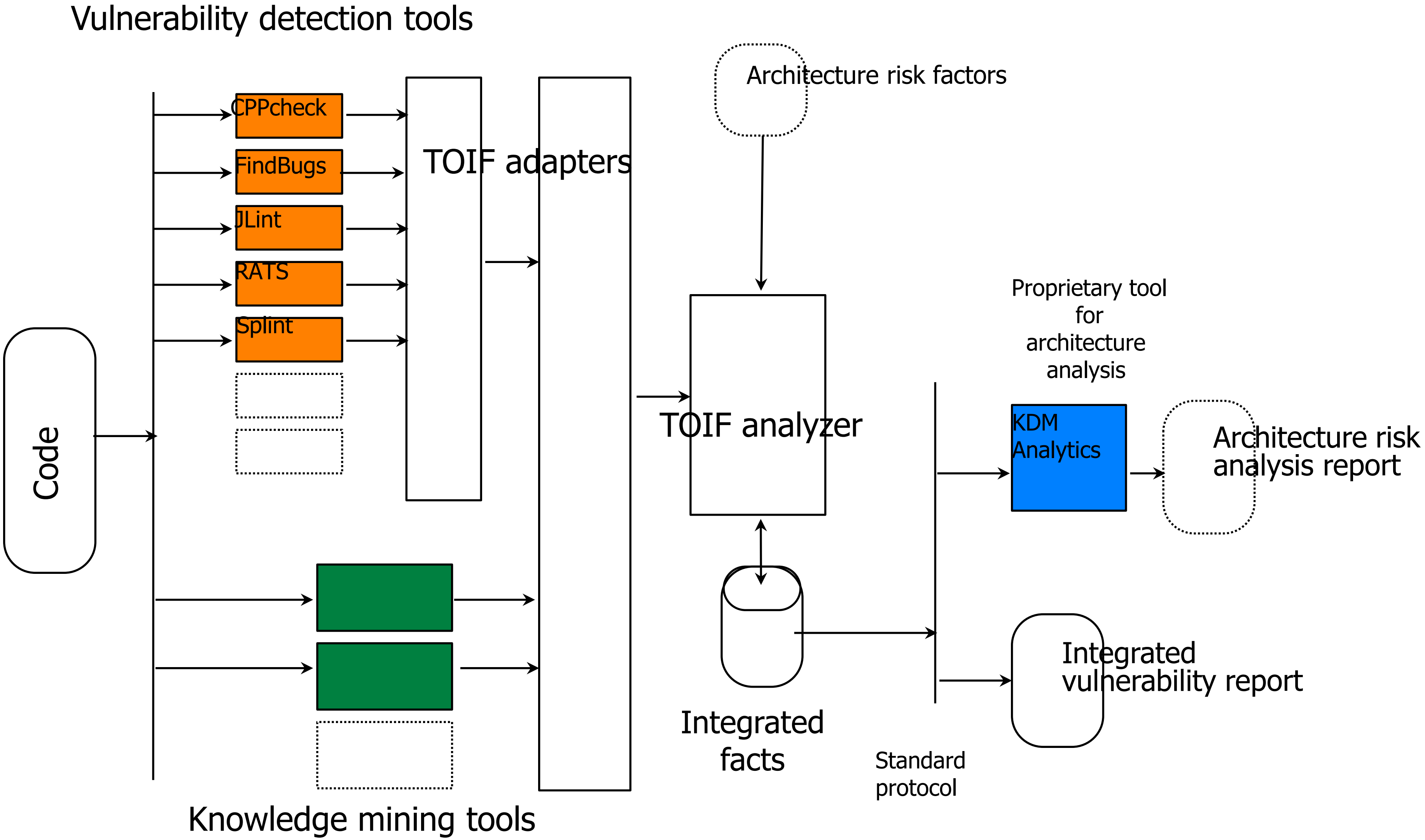
Technological Achievements

- Creating next-generation composite vulnerability analysis tool on top of existing off-the-shelf vulnerability detection tools
- Improving the breadth and accuracy of vulnerability analysis
- Improving the rigor of assessments by bringing vulnerability detection into architecture context
- Normalizing vulnerability reporting protocols
- Leveraging OMG Software Assurance Ecosystem standards and formalizations of CWE content

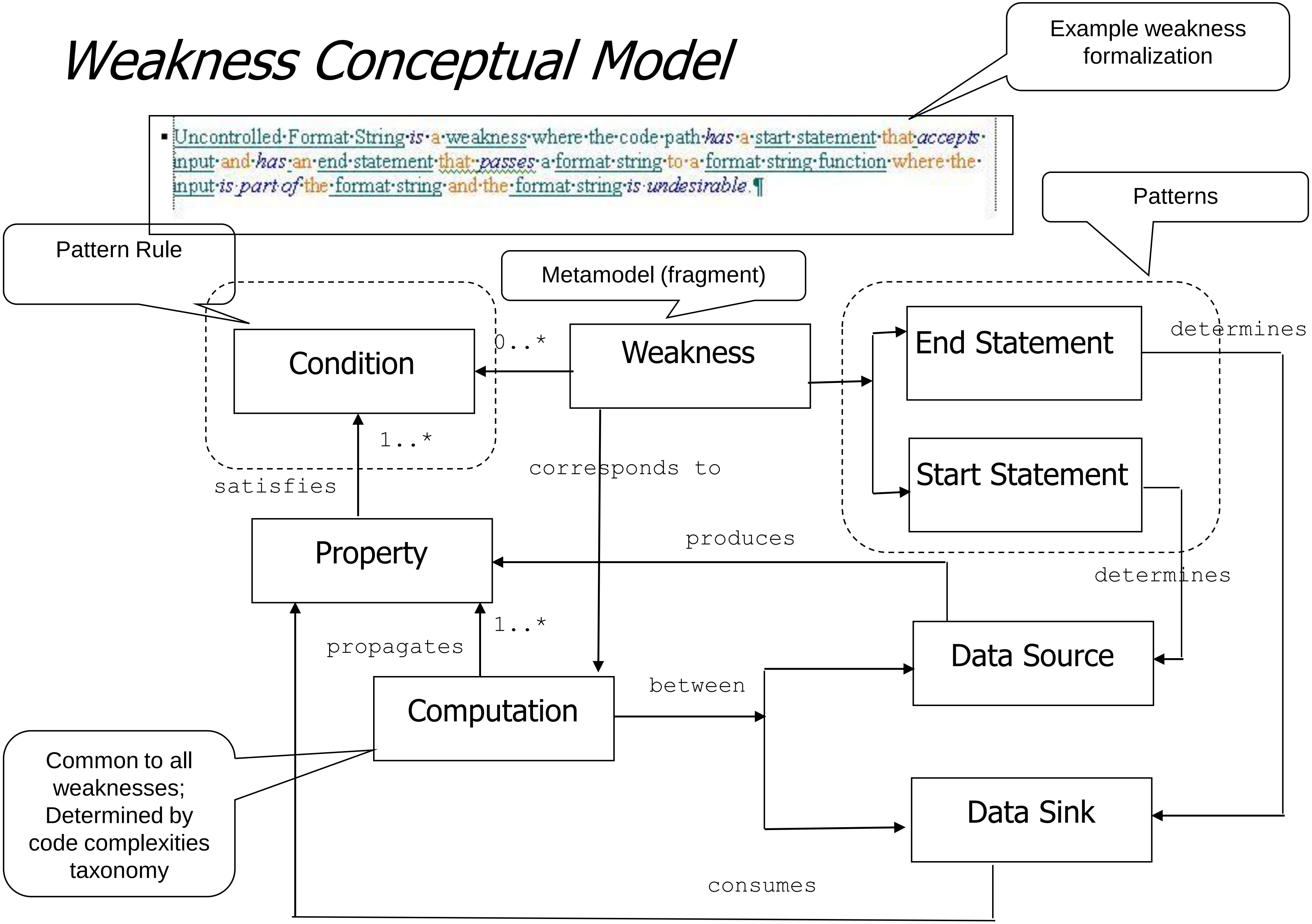
Delivering open source product: analyzer and run time framework for integrating findings of vulnerability detection tools including integration of 5 existing open source vulnerability analyzers.

In addition proprietary architecture analysis tool will be integrated to show greater value when viewing CWEs within the architecture content.

TOIF Architecture


Vulnerability detection tools
Architecture risk factors
CPPcheck
FindBugs
JLint
RATS
Splint
TOIF adapters
Code
Proprietary tool for architecture analysis
TOIF analyzer
KDM Analytics
Architecture risk analysis report
Integrated facts
Standard protocol
Integrated vulnerability report
Knowledge mining tools

Weakness Conceptual Model



Example weakness formalization

- Uncontrolled Format String *is* a weakness where the code path *has* a start statement that *accepts* input and *has* an end statement that *passes* a format string to a format string function where the input *is part of* the format string and the format string *is undesirable*.¶

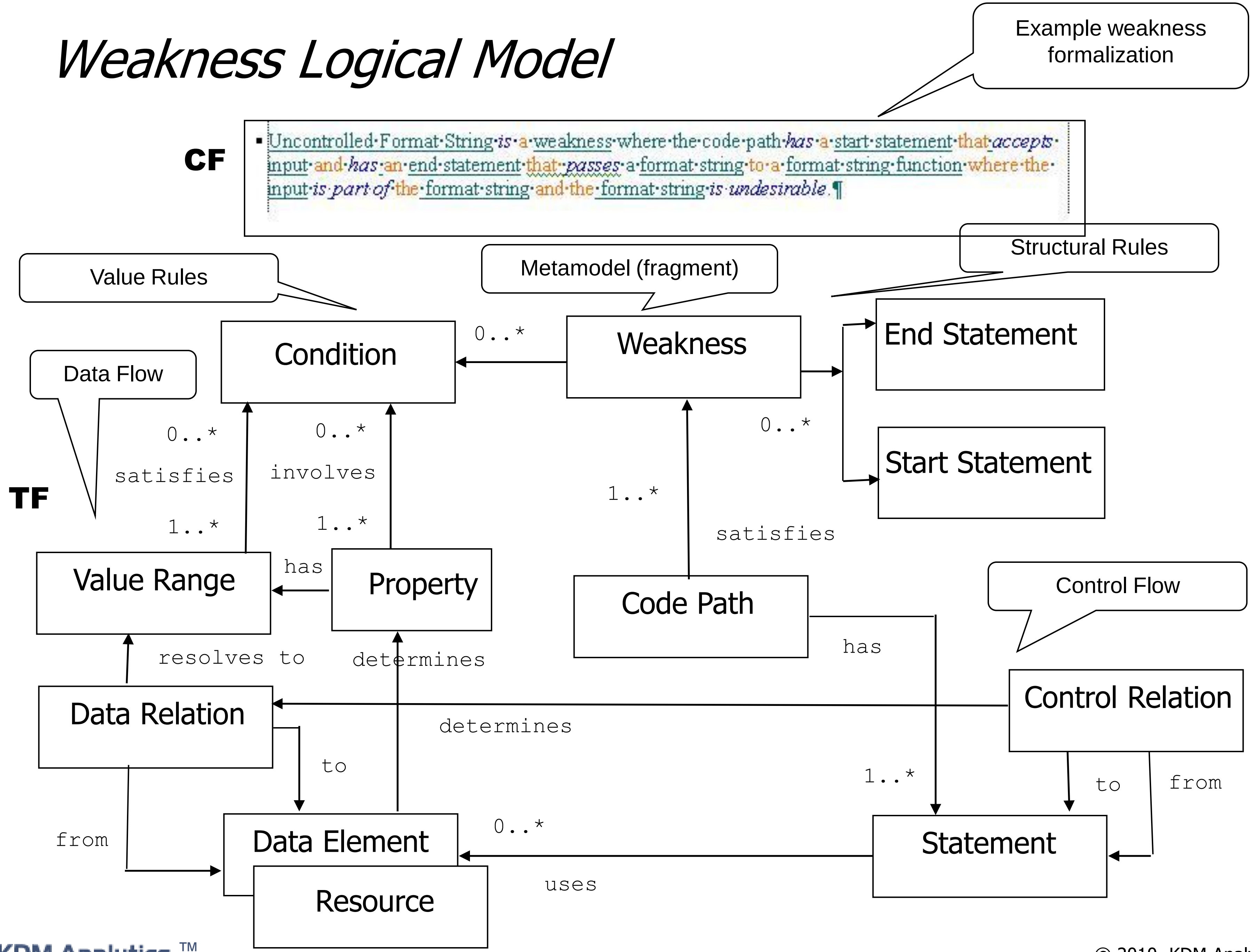
Patterns

Pattern Rule

Metamodel (fragment)

Common to all weaknesses; Determined by code complexities taxonomy

Weakness Logical Model



- Uncontrolled Format String *is* a weakness where the code path *has* a start statement that *accepts* input and *has* an end statement that *passes* a format string to a format string function where the input *is part of* the format string and the format string *is undesirable*.¶

The Tools' Output Integration Framework Fact-oriented integration

- Capability to integrate multiple vulnerability detection tools as "data feeds" into the repository
  - Based on a common protocol for exchanging vulnerability findings
  - Achieved through normalizing vocabularies across multiple tools
- Capability to collate findings from several tools
- Capability to put vulnerability findings into the context of other facts about the system (such as metrics, architecture, design patterns, etc.)
  - Based on existing standard protocol for exchanging system facts, the OMG Knowledge Discovery Metamodel (KDM),
  - now ISO/IEC 19506

- As the result: single integrated repository of high-fidelity facts about a software system

Integration points

- Nomenclature of the vulnerability (CWE)
- Location of the vulnerability
  - basic: file, linenumber, position
  - advanced: system facts
    - procedure, method, statement, call, read, etc
    - scenario
- Pattern
  - sink
  - source

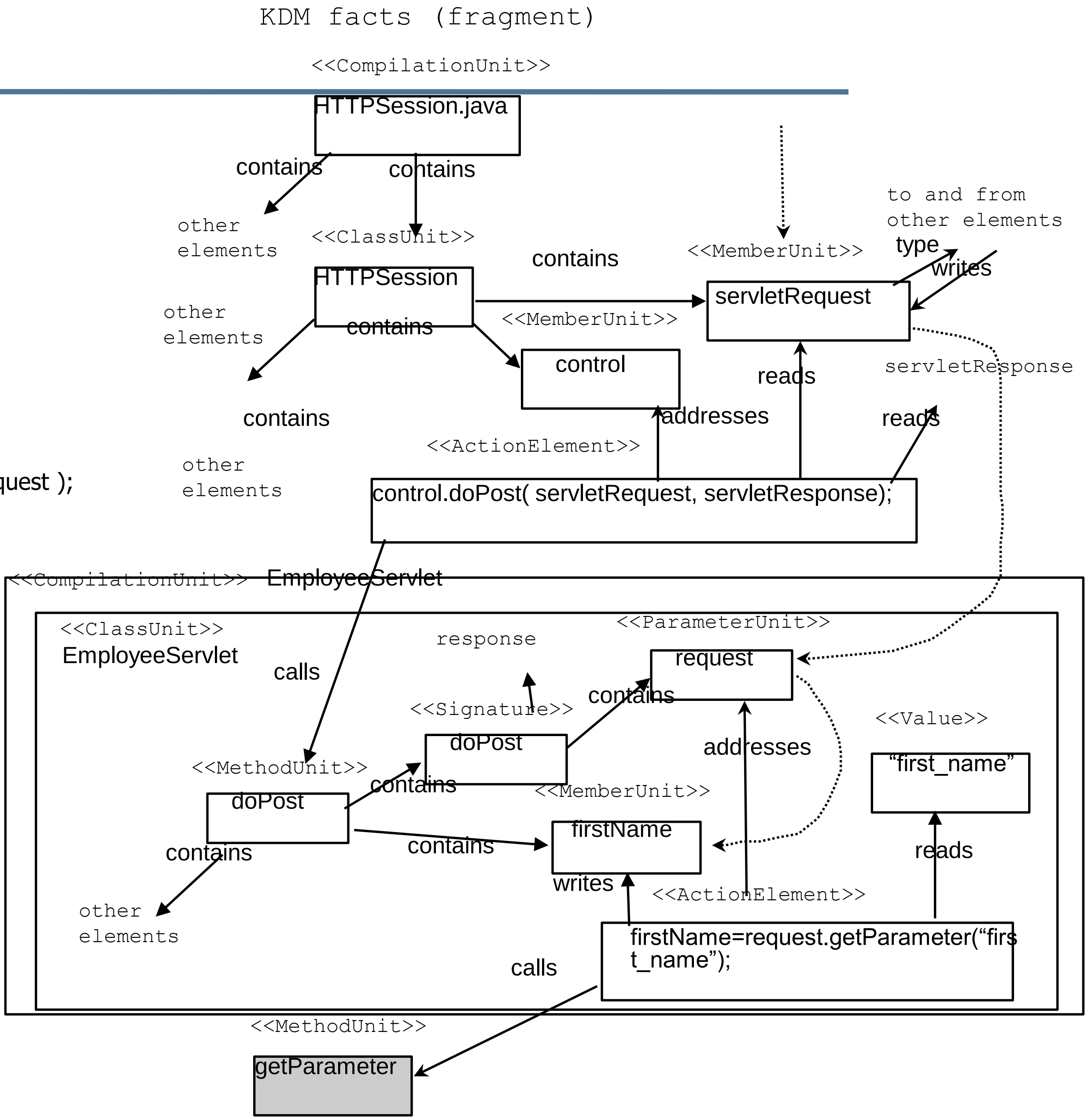
KDM code facts

HTTPSession.java(fragment)

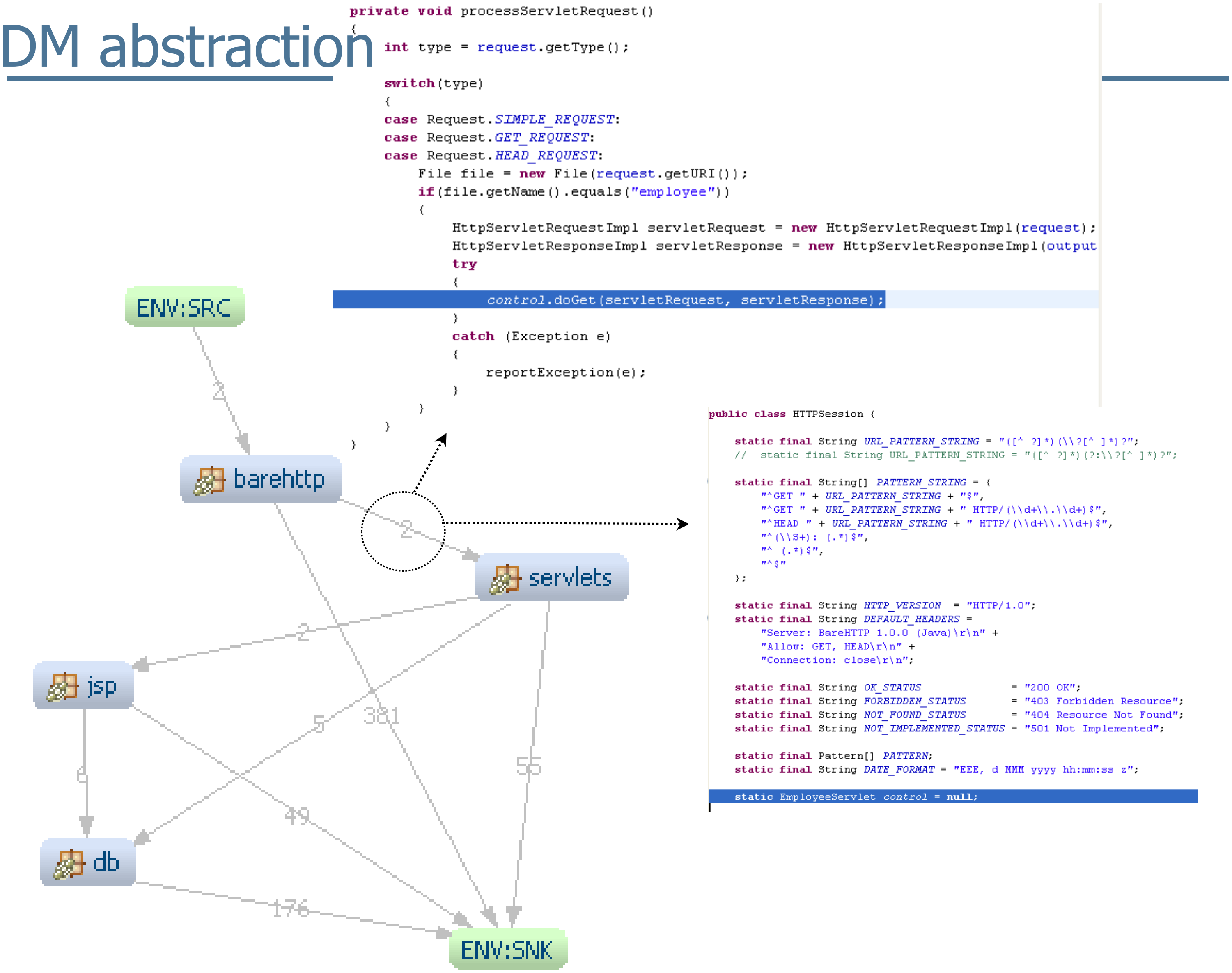
```
public class HTTPSession {
EmployeeServlet control;
Request request;
…
control = new EmployeeServlet();
…
          void processServletRequest() {…
          HTTPServletRequest servletRequest=
                new HttplServletRequestImpl( request );
          …
          control.doPost( servletRequest,
                          servletResponse );
          …
          }
}
```

EmployeeServlet.java(fragment)

```
public class EmployeeServlet {
…
 void doPost( HttpServletRequest request,
HttpServletResponse response) {
  …
  String firstName=
       request.getParameter("first_name");
  …
  }
 …
}
```



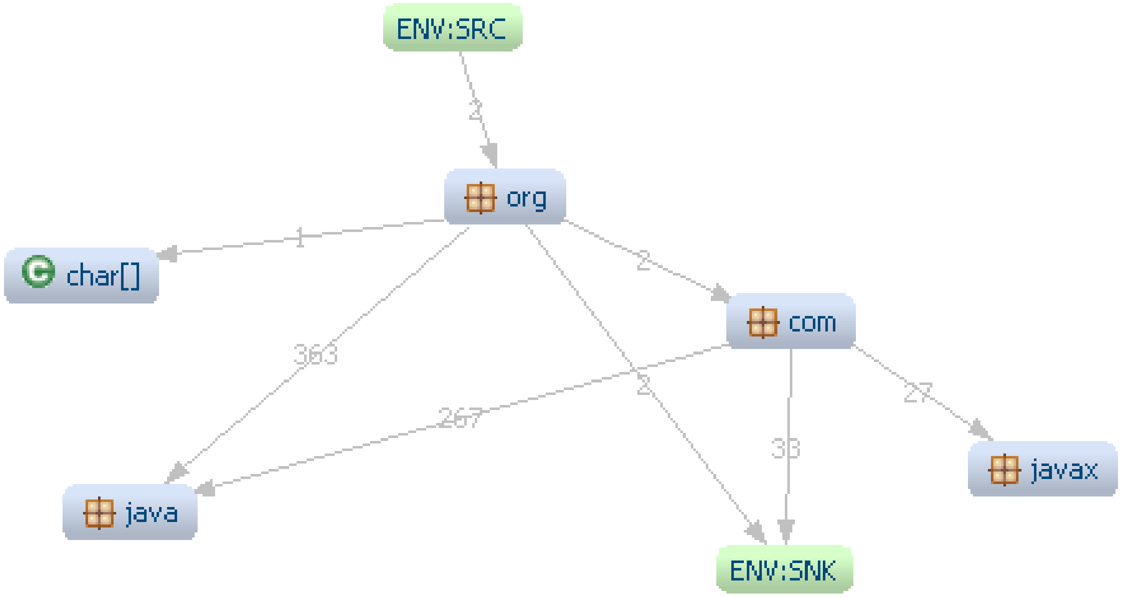
KDM abstraction



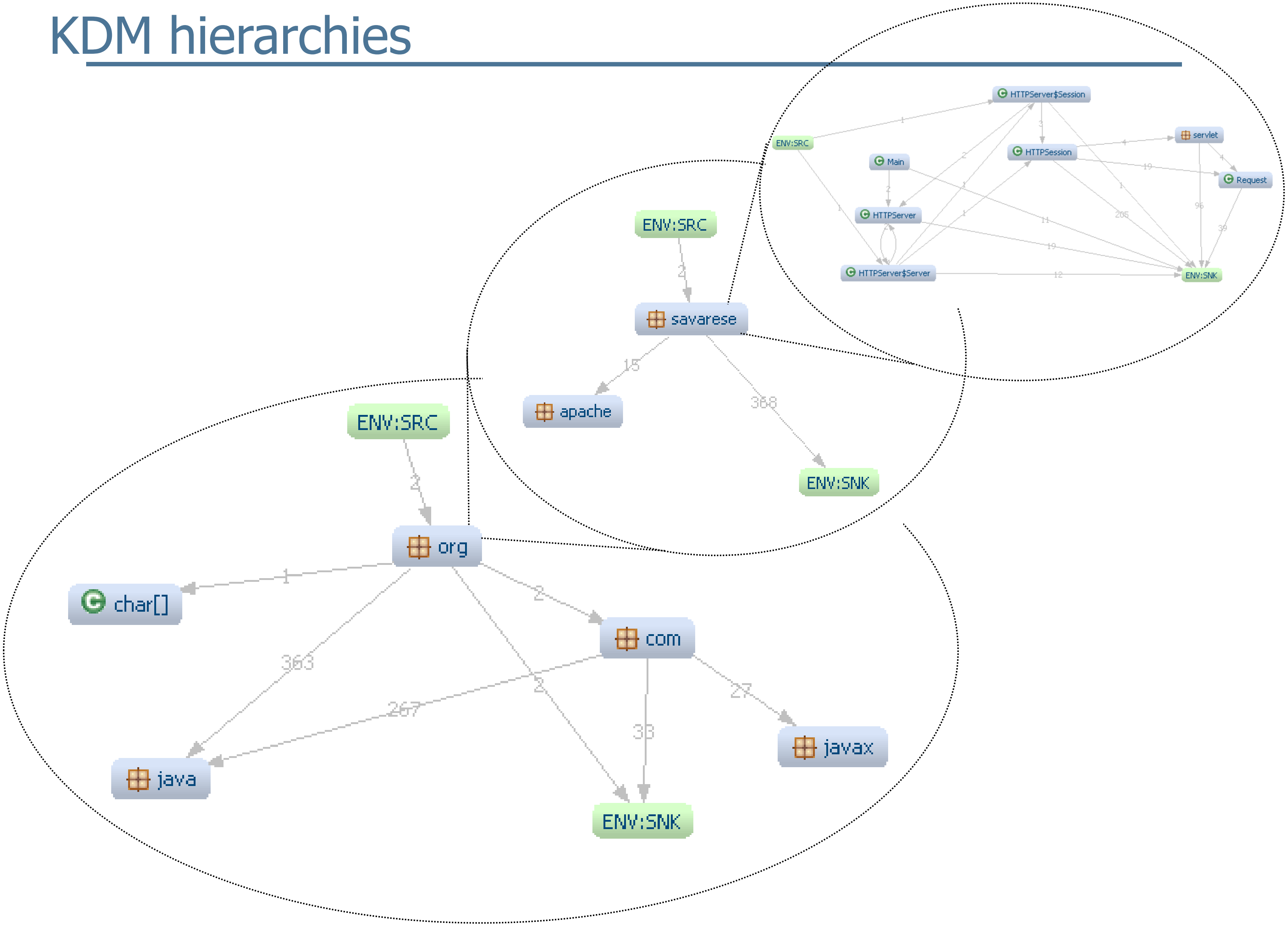
```
private void processServletRequest()
{
    int type = request.getType();

    switch(type)
    {
    case Request.SIMPLE_REQUEST:
    case Request.GET_REQUEST:
    case Request.HEAD_REQUEST:
        File file = new File(request.getURI());
        if(file.getName().equals("employee"))
        {
            HttpServletRequestImpl servletRequest = new HttpServletRequestImpl(request);
            HttpServletResponseImpl servletResponse = new HttpServletResponseImpl(output
            try
            {
                control.doGet(servletRequest, servletResponse);
            }
            catch (Exception e)
            {
                reportException(e);
            }
        }
    }
}
```

```
public class HTTPSession {

    static final String URL_PATTERN_STRING = "([^ ?]*)(\\?[^ ]*)?";
    //  static final String URL_PATTERN_STRING = "([^ ?]*)(?:\\?[^ ]*)?";

    static final String[] PATTERN_STRING = {
        "^GET " + URL_PATTERN_STRING + "$",
        "^GET " + URL_PATTERN_STRING + " HTTP/(\\d+\\.\\d+)$",
        "^HEAD " + URL_PATTERN_STRING + " HTTP/(\\d+\\.\\d+)$",
        "^(\\S+): (.*)$",
        "^ (.*)$",
        "^$"
    };

    static final String HTTP_VERSION  = "HTTP/1.0";
    static final String DEFAULT_HEADERS =
        "Server: BareHTTP 1.0.0 (Java)\r\n" +
        "Allow: GET, HEAD\r\n" +
        "Connection: close\r\n";

    static final String OK_STATUS                 = "200 OK";
    static final String FORBIDDEN_STATUS          = "403 Forbidden Resource";
    static final String NOT_FOUND_STATUS          = "404 Resource Not Found";
    static final String NOT_IMPLEMENTED_STATUS = "501 Not Implemented";

    static final Pattern[] PATTERN;
    static final String DATE_FORMAT = "EEE, d MMM yyyy hh:mm:ss z";

    static EmployeeServlet control = null;
```

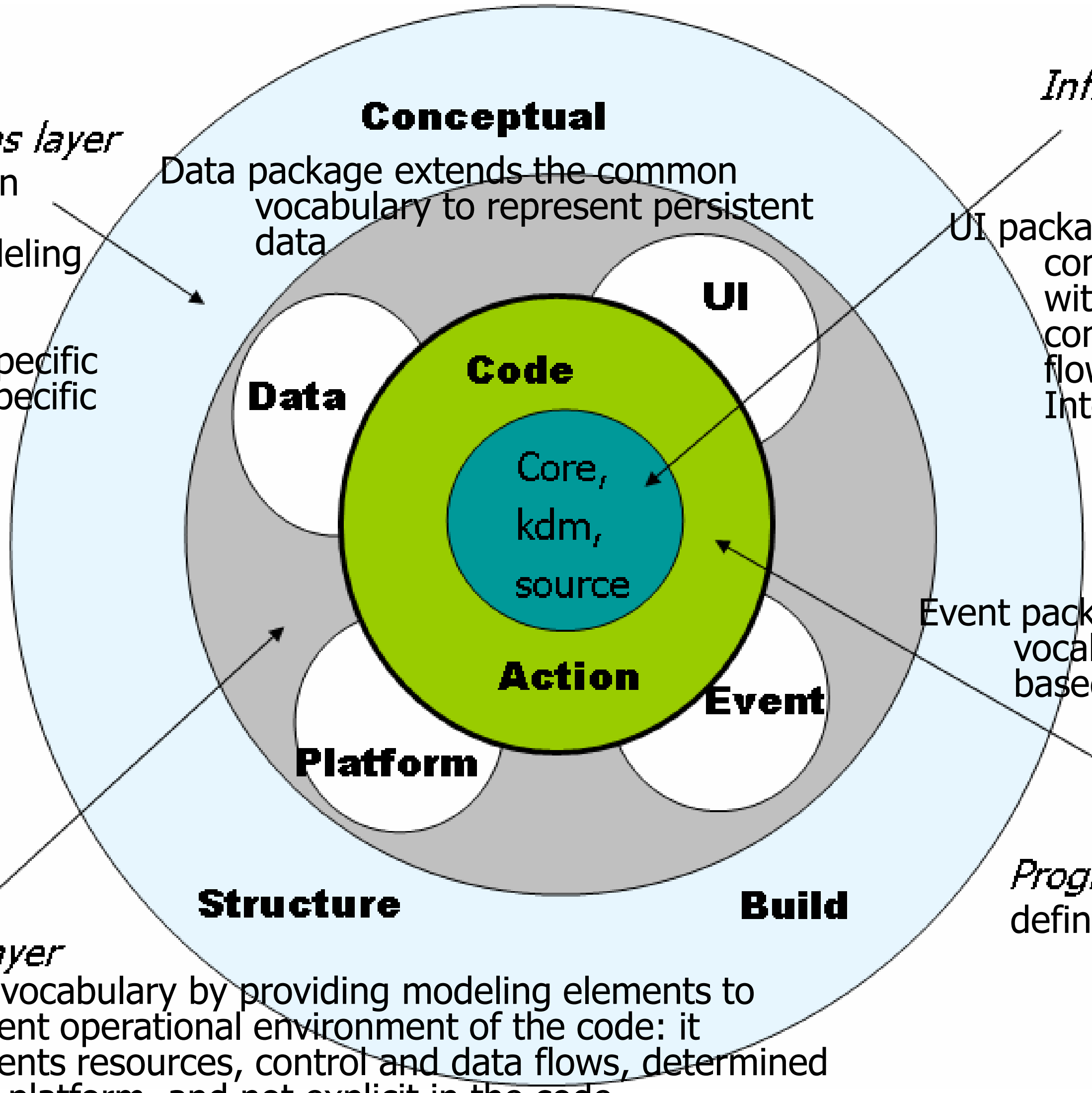
KDM Top model



KDM hierarchies



Knowledge Discovery Metamodel:



Abstractions layer

Extends the common vocabulary by providing modeling elements to represent architecture-specific and domain-specific abstractions

Data package extends the common vocabulary to represent persistent data

Infrastructure layer

UI package extends the common vocabulary with resources, control and data flow related to User Interfaces

Event package extends the common vocabulary to represent state-based behavior

Program Elements layer

defines the common vocabulary for representing code

Resource layer

extends the vocabulary by providing modeling elements to represent operational environment of the code: it represents resources, control and data flows, determined by the platform, and not explicit in the code

Going forward

- Integration of existing vulnerability detection tools and cross-correlation of their findings with architectural analysis is important for software assurance
- Commercialization through open source
  - Integrate selected open source vulnerability detection tools
  - Open source KDM extraction tools
- TOI Framework protocol is easy to adopt by tool vendors
- Phase II will involve a practical case study
  - Assessment of DNS Bind and Wireshark
- Deliverables:
  - a ready-to-use open source composite vulnerability analyzer integrating 5 existing open source vulnerability detection tools
  - integrating proprietary architecture analysis tool
  - a protocol for exchanging vulnerability findings
  - blueprints for adaptors of the protocol
  - practical usability and accuracy data based on the case study

Potential Benefits

- Powerful open source vulnerability detection platform
- Reference implementation for standard-based adaptors
  - Blue print how to integrate additional analyzers
- Further CWE normalization of vulnerability reports based on the Software Fault Patterns; adoption of SFPs
- Adoption of standard-based reporting of vulnerabilities
- Utilization of open source development to advance the SwA space