Synthesizing Client Load Models for Performance Engineering via Web Crawling

Rainbow Cai, John Grundy and John Hosking

Department of Computer Science, University of Auckland, New Zealand

November 2007, ASE, Atlanta, US

Outline

- Introduction
- Motivation and Related Work
- MaramaMTE+
- Example Usage
- Design and Implementation
- Conclusions and Discussions
- Summary

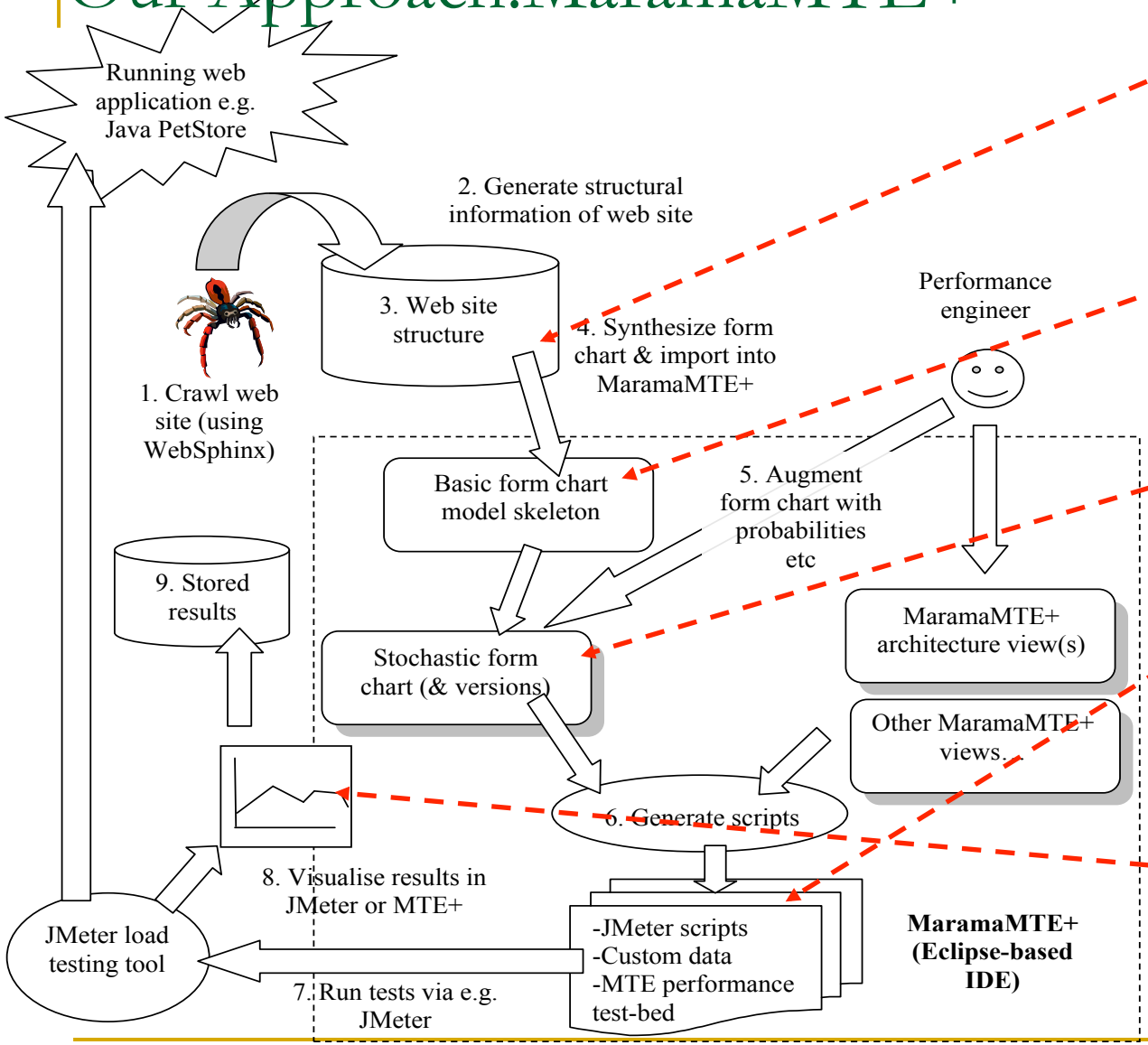
Motivation

- Web load testing needed to ensure performance requirements met
- Current load testing environments (e.g. JMeter, MS ACT) support complicated testing plans
- BUT - it's tedious and error-prone to manually script these

Motivation

- Ideally web load testing tool support must allow users to:
  - easily change client load model testing parameter values
  - generate multiple testing plans/scripts automatically
- Tool should be well-integrated within a generic performance engineering environment
  - allows realistic client behavior model to influence the design of other parts of software system e.g. the software architecture.

Our Approach:MaramaMTE+



1. Crawl target web site to extract structure into database
2. Synthesize a form chart model from the web structure database in MaramaMTE+
3. Engineer augments with probabilities = “stochastic form chart” formal model
4. Generate JMeter etc load test plans/scripts
5. Run & analyse the performance tests. Potentially change probabilities/ restructure form charts & compare results

Example Usage

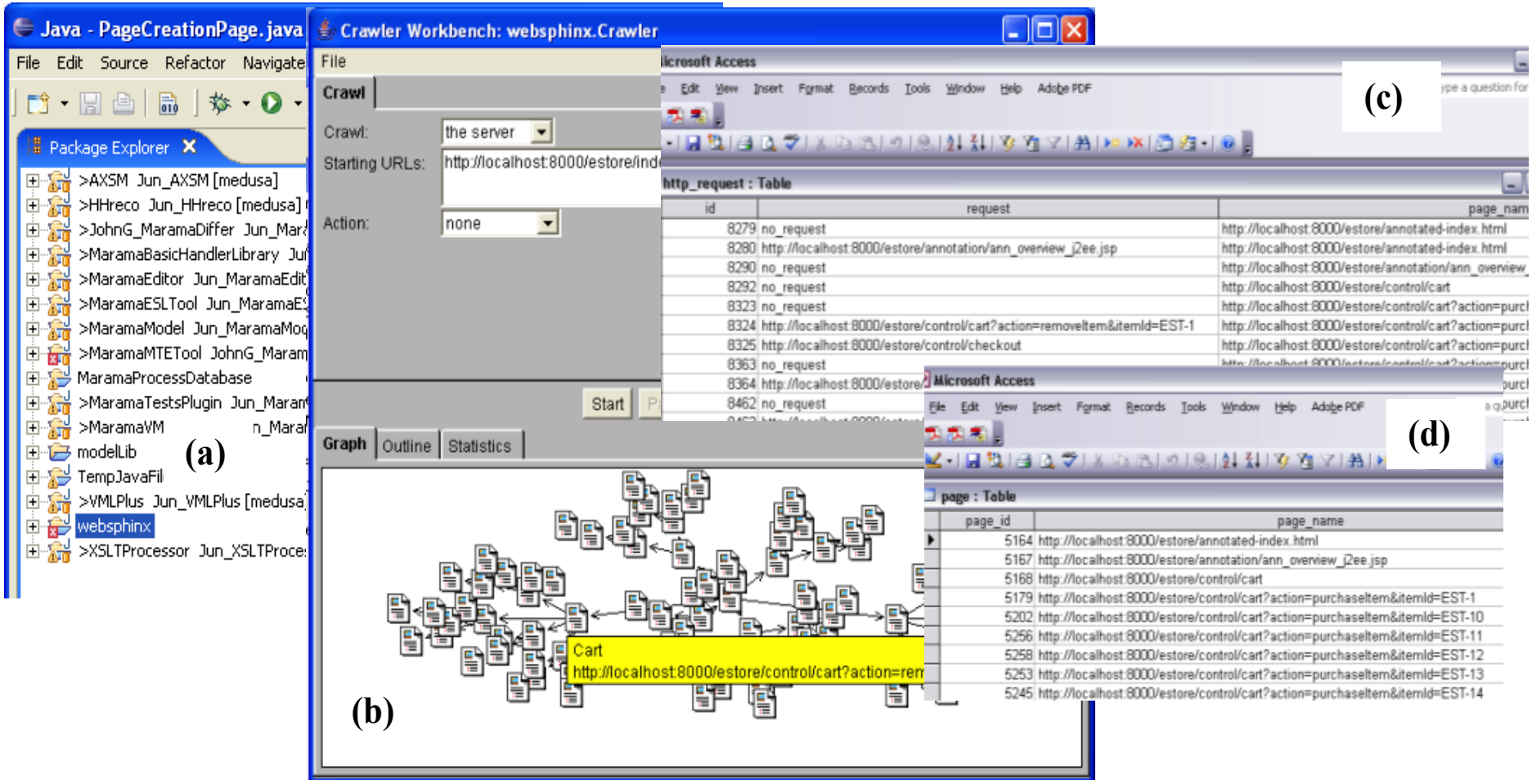
- Use Java Pet Store reference application to illustrate effectiveness of our MaramaMTE+ approach

  http://java.sun.com/developer/releases/petstore/

- Shows the main steps of load testing PetStore legacy system in MaramaMTE+:
  - HTTP request extraction from target web site
  - Form chart augmentation by engineer
  - Load test generation (target tool JMeter)
  - Running generated JMeter tests

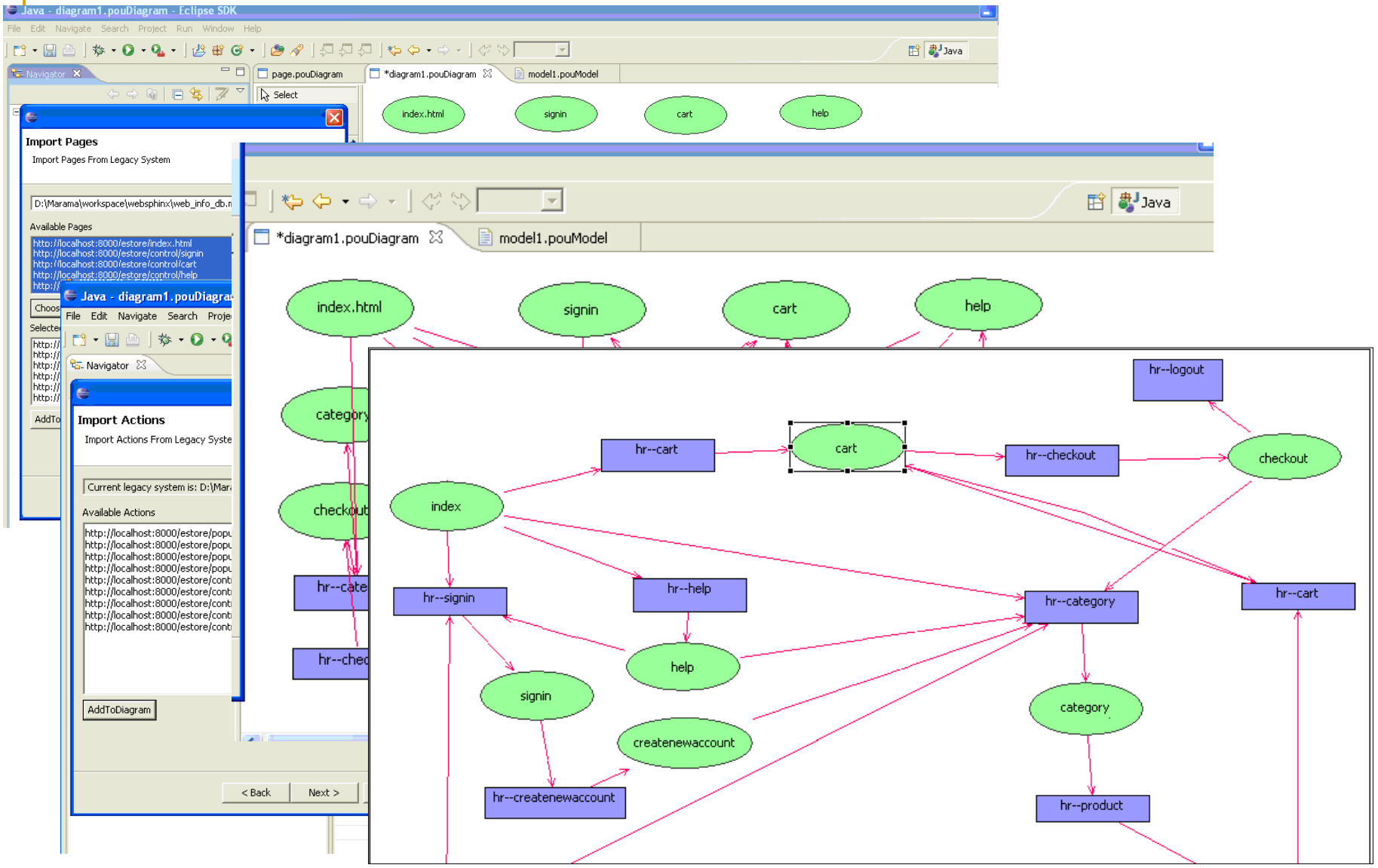
Example Usage - HTTP Request Extraction

- WebSphinx used to extract Pet Store structural information
  - User supplies target web site information to the crawler
  - Crawler explores the main screens, hyper links, and http requests, parameters and values for target web site
- MaramaMTE+ collects data into a purpose-built crawler/result/http request database
- E.g.:
  - "http_request" table holds http requests and associated pages
  - "page" table holds information about page ids and names


Java - PageCreationPage.java
File Edit Source Refactor Navigate
Package Explorer
>AXSM Jun_AXSM [medusa]
>HHreco Jun_HHreco [medusa]
>JohnG_MaramaDiffer Jun_Mar
>MaramaBasicHandlerLibrary Ju
>MaramaEditor Jun_MaramaEdit
>MaramaESLTool Jun_MaramaE
>MaramaModel Jun_MaramaMo
>MaramaMTETool JohnG_Maram
MaramaProcessDatabase
>MaramaTestsPlugin Jun_Maram
>MaramaVM
modelLib
TempJavaFil
>VMLPlus Jun_VMLPlus [medusa
websphinx
>XSLTProcessor Jun_XSLTProce
(a)
Crawler Workbench: websphinx.Crawler
File
Crawl
Crawl: the server
Starting URLs: http://localhost:8000/estore/ind
Action: none
Start
Graph Outline Statistics
Cart
http://localhost:8000/estore/control/cart?action=rem
(b)
Microsoft Access
File Edit View Insert Format Records Tools Window Help Adobe PDF
(c)
http_request : Table
id request page_nam
8279 no_request http://localhost:8000/estore/annotated-index.html
8280 http://localhost:8000/estore/annotation/ann_overview_j2ee.jsp http://localhost:8000/estore/annotated-index.html
8290 no_request http://localhost:8000/estore/annotation/ann_overview
8292 no_request http://localhost:8000/estore/control/cart
8323 no_request http://localhost:8000/estore/control/cart?action=purcl
8324 http://localhost:8000/estore/control/cart?action=removeItem&itemId=EST-1 http://localhost:8000/estore/control/cart?action=purcl
8325 http://localhost:8000/estore/control/checkout http://localhost:8000/estore/control/cart?action=purcl
8363 no_request
8364 http://localhost:8000/estore/
8462 no_request
Microsoft Access
File Edit View Insert Format Records Tools Window Help Adobe PDF
(d)
page : Table
page_id page_name
5164 http://localhost:8000/estore/annotated-index.html
5167 http://localhost:8000/estore/annotation/ann_overview_j2ee.jsp
5168 http://localhost:8000/estore/control/cart
5179 http://localhost:8000/estore/control/cart?action=purchaseItem&itemId=EST-1
5202 http://localhost:8000/estore/control/cart?action=purchaseItem&itemId=EST-10
5256 http://localhost:8000/estore/control/cart?action=purchaseItem&itemId=EST-11
5258 http://localhost:8000/estore/control/cart?action=purchaseItem&itemId=EST-12
5253 http://localhost:8000/estore/control/cart?action=purchaseItem&itemId=EST-13
5245 http://localhost:8000/estore/control/cart?action=purchaseItem&itemId=EST-14

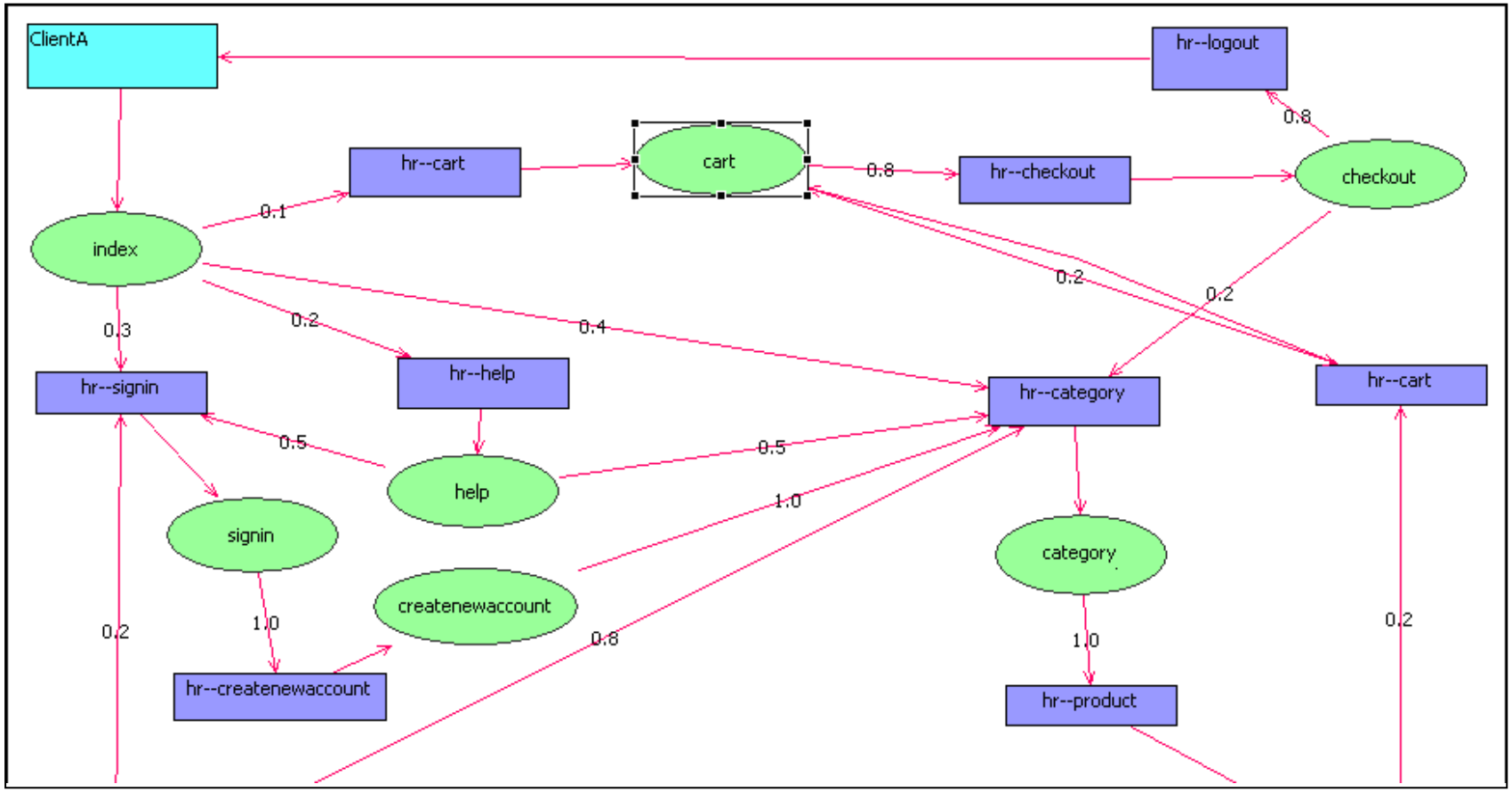
Example Usage - Form Chart Extraction

- MaramaMTE+ uses crawler data to generate an initial form chart model structure
  - Includes forms, actions, transitions, links to underlying web site structure (URLs)
- For large web sites engineers can view and edit partial form charts in multiple diagrams that share a single model
- Diagram and model versioning mechanism allows alternate versions of the form charts to be created, then compared, differentiated and merged.


Java - diagram1.pouDiagram - Eclipse SDK
File Edit Navigate Search Project Run Window Help
Navigator
page.pouDiagram
*diagram1.pouDiagram
model1.pouModel
Select
index.html
signin
cart
help
Import Pages
Import Pages From Legacy System
D:\Marama\workspace\websphinx\web_info_db.n
Available Pages
http://localhost:8000/estore/index.html
http://localhost:8000/estore/control/signin
http://localhost:8000/estore/control/cart
http://localhost:8000/estore/control/help
Java - diagram1.pouDiagra
File Edit Navigate Search Proje
Navigator
Import Actions
Import Actions From Legacy Syste
Current legacy system is: D:\Mar
Available Actions
AddToDiagram
< Back
Next >
Java
*diagram1.pouDiagram
model1.pouModel
index.html
signin
cart
help
category
checkout
hr--logout
hr--cart
cart
hr--checkout
checkout
index
hr--signin
hr--help
hr--category
hr--cart
help
signin
createnewaccount
category
hr--createnewaccount
hr--product

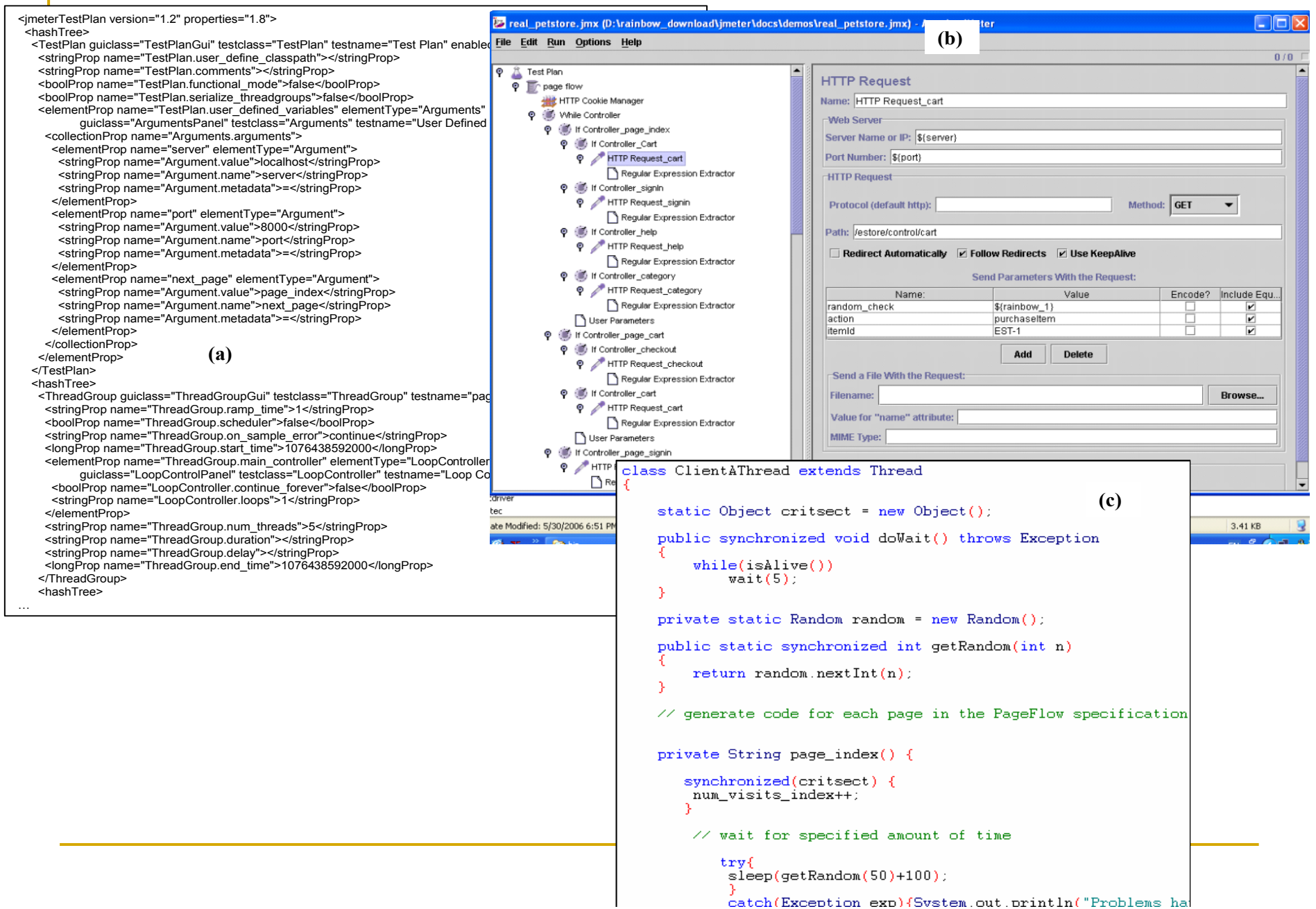
Example Usage ------ Form Chart Augmentation

- MaramaMTE+ uses the generated form chart in two ways:
  - model of user interaction behaviour from which to generate testing tool scripts e.g. for JMeter (system developed and under stress test)
  - with other MaramaMTE+ models (e.g. the architecture design models, business process models and service composition models) to generate a performance evaluation test bed for server-side (system under design)
- For either, engineer needs to augment model with properties that specify user interaction behaviours and code generation information
- Can version form chart and add different probabilities etc to compare performance under different user behaviours (i.e. loading conditions)


ClientA
hr--logout
0.8
hr--cart
cart
0.8
hr--checkout
checkout
0.1
index
0.2
0.2
0.2
0.3
0.2
0.4
hr--help
hr--signin
hr--category
hr--cart
0.5
0.5
help
1.0
signin
category
createnewaccount
0.2
1.0
0.8
1.0
0.2
hr--createnewaccount
hr--product

Example Usage ------ Loading Test Generation

- MaramaMTE+ uses stochastic form chart to generate e.g. JMeter testing plan and associated scripts:
  - MaramaMTE+ form chart "Page" = state of the website;
  - "Action" = JMeter http requests to obtain certain web pages;
  - "Transition" = transitions between web pages via "Actions";
  - "Probability" and "URL" properties used to generate the logic controllers of the JMeter testing plan
- MaramaMTE+ may also generate a Java load testing programme for client and/or server-side (if system under design):
  - uses a state machine implementation
    - web pages = states
    - linked by form chart actions = http requests


<jmeterTestPlan version="1.2" properties="1.8">
<hashTree>
<TestPlan guiclass="TestPlanGui" testclass="TestPlan" testname="Test Plan" enable
<stringProp name="TestPlan.user_define_classpath"></stringProp>
<stringProp name="TestPlan.comments"></stringProp>
<boolProp name="TestPlan.functional_mode">false</boolProp>
<boolProp name="TestPlan.serialize_threadgroups">false</boolProp>
<elementProp name="TestPlan.user_defined_variables" elementType="Arguments"
guiclass="ArgumentsPanel" testclass="Arguments" testname="User Defined
<collectionProp name="Arguments.arguments">
<elementProp name="server" elementType="Argument">
<stringProp name="Argument.value">localhost</stringProp>
<stringProp name="Argument.name">server</stringProp>
<stringProp name="Argument.metadata">=</stringProp>
</elementProp>
<elementProp name="port" elementType="Argument">
<stringProp name="Argument.value">8000</stringProp>
<stringProp name="Argument.name">port</stringProp>
<stringProp name="Argument.metadata">=</stringProp>
</elementProp>
<elementProp name="next_page" elementType="Argument">
<stringProp name="Argument.value">page_index</stringProp>
<stringProp name="Argument.name">next_page</stringProp>
<stringProp name="Argument.metadata">=</stringProp>
</elementProp>
</collectionProp>
</elementProp>
</TestPlan>
<hashTree>
<ThreadGroup guiclass="ThreadGroupGui" testclass="ThreadGroup" testname="pag
<stringProp name="ThreadGroup.ramp_time">1</stringProp>
<boolProp name="ThreadGroup.scheduler">false</boolProp>
<stringProp name="ThreadGroup.on_sample_error">continue</stringProp>
<longProp name="ThreadGroup.start_time">1076438592000</longProp>
<elementProp name="ThreadGroup.main_controller" elementType="LoopController
guiclass="LoopControlPanel" testclass="LoopController" testname="Loop Co
<boolProp name="LoopController.continue_forever">false</boolProp>
<stringProp name="LoopController.loops">1</stringProp>
</elementProp>
<stringProp name="ThreadGroup.num_threads">5</stringProp>
<stringProp name="ThreadGroup.duration"></stringProp>
<stringProp name="ThreadGroup.delay"></stringProp>
<longProp name="ThreadGroup.end_time">1076438592000</longProp>
</ThreadGroup>
<hashTree>
...
(a)
real_petstore.jmx (D:\rainbow_download\jmeter\docs\demos\real_petstore.jmx)
(b)
File Edit Run Options Help
Test Plan
page flow
HTTP Cookie Manager
While Controller
If Controller_page_index
If Controller_Cart
HTTP Request_cart
Regular Expression Extractor
If Controller_signin
HTTP Request_signin
Regular Expression Extractor
If Controller_help
HTTP Request_help
Regular Expression Extractor
If Controller_category
HTTP Request_category
Regular Expression Extractor
User Parameters
If Controller_page_cart
If Controller_checkout
HTTP Request_checkout
Regular Expression Extractor
If Controller_cart
HTTP Request_cart
Regular Expression Extractor
User Parameters
If Controller_page_signin
HTTP Request
Name: HTTP Request_cart
Web Server
Server Name or IP: ${server}
Port Number: ${port}
HTTP Request
Protocol (default http):
Method: GET
Path: /estore/control/cart
Redirect Automatically
Follow Redirects
Use KeepAlive
Send Parameters With the Request:
Name: Value: Encode? Include Equ...
random_check ${rainbow_1}
action purchaseItem
itemId EST-1
Add Delete
Send a File With the Request:
Filename: Browse...
Value for "name" attribute:
MIME Type:
3.41 KB
class ClientAThread extends Thread
{
(c)
static Object critsect = new Object();
public synchronized void doWait() throws Exception
{
while(isAlive())
wait(5);
}
private static Random random = new Random();
public static synchronized int getRandom(int n)
{
return random.nextInt(n);
}
// generate code for each page in the PageFlow specification
private String page_index() {
synchronized(critsect) {
num_visits_index++;
}
// wait for specified amount of time
try{
sleep(getRandom(50)+100);
}
catch(Exception exp){System.out.println("Problems ha

Example Usage ------ Running Load Tests

- Run the generated JMeter testing plan against the web application server (Java PetStore).
- Analyse important performance measurements, e.g. :
  - visited web pages
  - sample numbers,
  - average response time,
  - min and max response time,
  - throughput, etc.
- Results presented using generated JMeter tabular result viewer
  - Can also use other JMeter-supplied result analysis tools and visualisations
  - Supports exploratory performance engineering for web applications in early-phase design or reengineering.

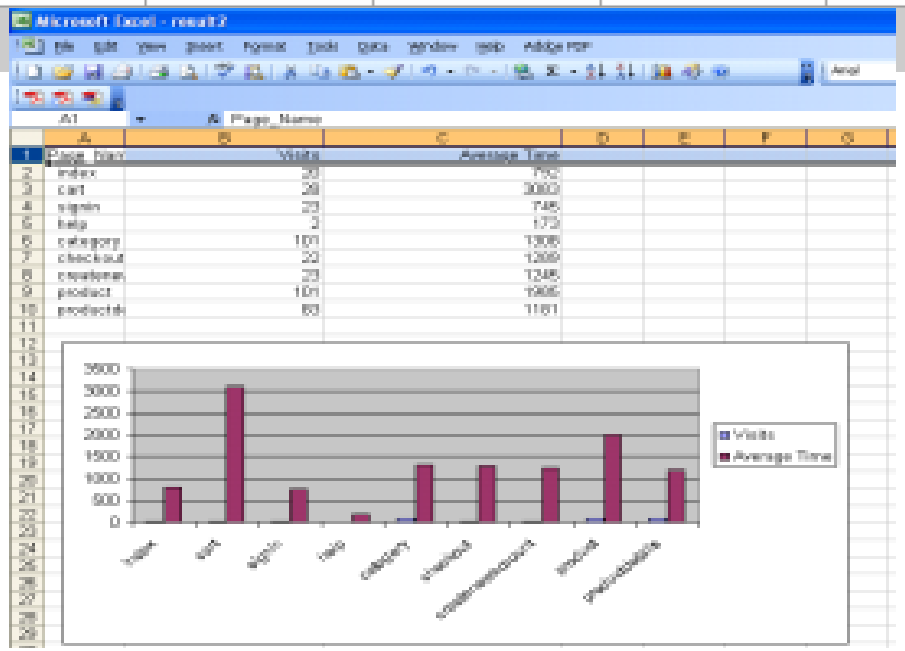
Summary Report

Name: Summary Report

Write All Data to a File

Filename | Browse... | Log Errors Only | Configure

Label	# Samples	Average	Min	Max	E
HTTP Request_help	2	200	50	351	0.00%
HTTP Request_signin	9	189	20	461	0.00%
HTTP Request_createnewaccount	9	252	20	601	0.00%
HTTP Request_category	45	377	40	711	0.00%
HTTP Request_getProduct	45	506	171	982	0.00%
HTTP Request_productdetails	38	766	241	1422	0.00%
HTTP Request_cart	8	1009	401	1612	0.00%
HTTP Request_checkout	7	472	210	761	0.00%
HTTP Request_logout	5	198	40	361	0.00%
TOTAL	168	509	20	1612	0.00%



Design and Implementation of MaramaMTE+

- Implemented as a set of Eclipse IDE plug-ins using Marama meta-tool
- Eclipse Java Emitter Templates (JET) scripts used to generate form chart-based test plans and scripts
  - JET template generates root JMeter test plan from client form chart component and its properties (name, host(s), threads etc)
  - JMeter initialisation components/scripts generated e.g. to set up timing monitors (WhileController).
  - First form chart page transformed to an initial http request on target web application (IfController).
  - Transitions to Actions in the form chart generate decision logic in the JMeter test script implementing a state machine model
    - Via JMeter's RegexExtractor, UserParameters, etc
  - Probabilities in form chart model may be simple random, fixed times, or complex stochastic probability models
    - Implemented as JMeter's BeanShellTimer, Gaussian Random Timer, etc

Discussion - Key advantages to MaramaMTE+

- The use of a formal stochastic form chart model for client load behavior modeling
  - Models user behavior in terms of probabilistic interactions
  - Model can be both reasoned about and used to generate test plans and scripts.
- The ability to extract basic form chart model structure from a web application via web crawling
  - Much less costly than hand-crafting a user model

Discussion - Key advantages to MaramaMTE+

- Can version stochastic form charts to compare and contrast performance under different client behavior models
  - i.e. under different loadings on the server
- Model-based generation of 3rd party stress testing tool test plans and scripts –
  - means MaramaMTE+ can leverage 3rd party load testing tools' advanced features e.g. JMeter's sophisticated measurement, reporting, distributed test execution and test scheduling support features
- Can run and compare web application performance under numerous different loading models accurately and efficiently

Discussion - Limitations

- Engineer needs to manually augment generated form chart with probabilities
- Must live with 3rd party tool limitations:
  - Most web application stress testing tools have less rich client behavioural models than form charts.
  - Thus we need to simplify the model when test scripts are generated or extend the testing tool (if possible)
  - Sometimes implementing form chart-specified behaviour is quite complex in the 3rd party testing tool
    - e.g. implementing a probabilistic state machine in JMeter is challenging
  - It is not always easy to control such tools in the way we are able to when generating our own client load test implementation

Summary: MaramaMTE+

- Automates retrieval of website structural data from a web user's perspective
- Generates a formal model of user interaction behaviour and load testing plans
- Effectiveness demonstrated through a case study, Java Pet Store:
  - site crawled, structural data extracted, a form chart model automatically generated and manually augmented, JMeter testing plans generated and executed, and load testing results collected
- Future plans:
  - will combine the generated form chart with a generated design level model of a legacy system
  - will make it possible for ordinary tool users to make rigorous comparisons between different products (e.g. Java PetStore and .NET PetShop)

Thank you!

Questions?

References

- Cai, Y., Grundy, J.C. and Hosking, J.G. Synthesizing Client Load Models for Performance Engineering via Web Crawling, In Proceedings of the 2007 IEEE/ACM International Conference on Automated Software Engineering, Atlanta, Nov 5-9 2007, IEEE CS Press.
- Grundy, J.C., Cai, Y. and Liu, A. SoftArch/MTE: Generating Distributed System Test-beds from High-level Software Architecture Descriptions, Automated Software Engineering, Kluwer Academic Publishers, vol. 12, no. 1, January 2005, pp. 5-39.
- Cai, Y., Grundy, J.C. and Hosking, J.G. Experiences Integrating and Scaling a Performance Test Bed Generator with an Open Source CASE Tool, In Proceedings of the 2004 IEEE International Conference on Automated Software Engineering, Linz, Austria, September 20-24, IEEE CS Press, pp. 36-45.
- Cai, Y., Grundy, J.C., Hosking, J.G., Dai, X. Software Architecture Modelling and Performance Analysis with Argo/MTE, In Proceedings of the 2004 Conference on Software Engineering and Knowledge Engineering, Baniff, Canada, June 20-24 2004.
- Grundy, J.C., Wei, Z., Nicolescu, R. and Cai, Y. An Environment for Automated Performance Evaluation of J2EE and ASP.NET Thin-client Architectures, In Proceedings of the 2004 Australian Software Engineering Conference, Melbourne, Australia, April 14-17 2004, IEEE CS Press.
- Grundy, J.C., Cai, Y. and Liu, A. Generation of Distributed System Test-beds from High-level Software Architecture Descriptions, In Proceedings of the 16th International Conference on Automated Software Engineering, San Diego, 26-29 Nov 2001, IEEE CS Press, pp. 193-200.