

웹소프트웨어의 신뢰성

- ✓ **Instructor:** Gregg Rothermel
- ✓ **Institution:** 한국과학기술원
- ✓ **Dictated:** 김윤정,장보윤,이유진,이해솔,이정연

🔊 [0:00]

Hello everyone

My name is Kyu-chul

Today I'm going to talk about this paper, IESE 09, name is "Invariant-based automatic testing of AJAX user interfaces"

Before start explaining main approaches of this paper, I'll briefly explain about some background knowledge of AJAX and DOM and XPath.

So, what is Ajax?

AJAX stands for asynchronous Java script and XML.

AJAX is not a single technology, it is a bit of grewed up technology.

So, XHTML and CSS is used standard-based presentation and the Dom is accessed with java script to dynamically display and allow the users to interact with information display.

Java script and XML Http Request object provide a method for exchanging data asynchronously between server and client to avoid full-page rewrote..

What is the big difference traditional web application and AJAX based application is the interaction styles.

The traditional web application, interaction between client and server, is asynchronously, so when user request data to the server, tester or user should wait for the response from the server.

But in a AJAX based application, communication between client and server and client and AJAX engine, is asynchronously, so, user should not wait for the result because AJAX engine can modify some UI, using modify DOM.

So, it is a big difference between traditional application and application.

🔊 [03:10]



The document object model, DOM, is application program interface API for valid HTML and XML document.

So, it defines logical structure of document and the way of excess and manipulate. As you can see, the DOM consists of this DOM tree.

The last one is X Path.

X Path is the XML Path Language.

It is a query language selecting nodes from an XHTML document.

In addition, X Path maybe is used to compute values from the content of an XTM document.

X Path can express the position of the state.

So, this one means the project and project, editions and editions.

And these below are the Ajax-based web application characteristic.


Details of these characteristic is shown by the next following slide.

So when testing web applications, traditional web application's testing is request and response from the server and analyze the resulting HTML page.

It is based on the page-sequence paradigm.

But, AJAX- based application's testing has difficulties to apply the traditional web applications testing approach, because AJAX-based application have complex time behavior.

So it is more [?] and more errors than traditional web application.

 **[06:01]**

So, in a related work, current Ajax testing approaches is the server side of current Ajax testing approaches can be tested any conventional testing technique, but on the client-side in a functional level, unit testing tool for JavaScript is Jsunit.

And DOM based testing, there is some testing tools are Seluniem, Webking and Sahi.

It is a capture event fired by usual interaction.

But, it has a problem, because a substantial amount of manual effort on the part of tester, because these tools based on capture and replay, so testers have a lot of things to do in testing.



So, in this paper, the authors crawl the Ajax application, simulating real user events on the user interface and infer the abstract model automatically.

This figure is an example of state-flow graph.

State-flow graph consists of node and edges.


In the first time, there is an initial state, when event is occurred, New Node is added on the state-flow graph on the state machine and edge is connected between before and after the event occurred.

So, from this slide, I'm going to talk about Ajax crawling technique and in this paper to crawl Crawljax.

They named a Crawljax.

Crawljax can exercise client side code.

It identifies clickable elements and from these state changes, crawljax infers a state-flow-graph.


 **[09:00]**

In this paper, they define an Ajax UI state changes on the DOM tree caused either by server-side state changes propagated to the client or client-side events handled by the Ajax engine.

So, in order to make a state-flow-graphs, these steps are needed deriving Ajax states, inferring the state machine, detecting clickables, creating states, processing document tree deltas and navigating the states.

So, I will explain this each steps respectively.

This figure is processing view of [?]

 **[10:20]**

Used embedded browser into base supporting technologies required by AJAX.

And a robot is used to simulate user input on the embedded browser.

And the finite state machine is a data component maintaining the state flow graph as well as a pointer to the current state.

And the controller has access to the browser DOM and detects the state challenges, changes.

It also controls the robot's action and it is also responsible to responsible for updating the state machine when update occurs on the DOM.

And the algorithm used by this component to actually infer the state machine.

The AJAX application as a ...

I don't know the details of the state but in the result of the test web applications, they shows the one state there is some... yeah.

Maybe the one state has a information about this... when event occurs this state information about the usual interface

🔊 **[14:30]**

Maybe I agree with 경욱 because it's a paper about testing usual interface so maybe usual interface application of AJAX related to DOM tree structure so I think they didn't have the details of what is the state.

When a creating state, they lift a change detected they had a new state for the state flow graph using a compare DOM tree so I think state is kind of DOM tree using a compare ...

And next step is detecting clickables.

So find a set of candidate elements they expose a event type click or event type mouse over or something like that in a automatic mode the candidate clickables are based on their Html type level name such as (?17:36) in the html and attribute to constraint such as class equal main [?17:53]

🔊 **[17:54]**

So for each candidate element the crawler fires click on the element embedded browser.

And after firing an event on a click candidate click cover, they creating states, using compare DOM tree, before event and after the event check whether the event result in a state change.

If a state change is detected, new state is created on add to the state flow graph on a state machine.

And a new edge is created on the graph between states.

They use in order to compare DOM tree, they use a Levenshton edit distance during the method.

I haven't investigated this comparison method so I skipped this in this presentation.



And AJAX application has a characteristic of avoiding full pages below so there is some delta update on a DOM tree, so current document tree differs from the previous document tree and the current one they use an enhanced document tree to compare the delta update.

And delta update may be due to a server request call that injects new element into the DOM

And the crawling procedure to recursively call to find new possible states in the partial change made to DOM tree.

So crawls using a depth first search so when adding a new state called a find a new possible state in a that starts a new adding element.

Ah new adding states.

🔊[20:40]

But using a that first three, in order to back to the previous state they using a back button but uh...there is a problem because browser doesn't record the history for these crawling.

So they made solutions save information about elements and the order.

And reload the application and follow and execute from initial state to the desired state.

In this time they use X Path expressions because X Path expressions can present the position of state.

But this X Path expression is not 100 percent guaranteed because of side effect of element execution server side states.

I don't have an example of this problem but maybe it's obviously because there are some side effects of element execution.

So when because of the when the server side state changed because of the network problem the client side doesn't change same as server side.

In that time maybe this expression is not 100 percent guaranteed.

(Students questioning)

Like this? Maybe you can see this details of X Path from the Wikipedia.

I forgot the details so...



🔊 [24:30]

(Students questioning)

Yes, a crawler crawls the client side of application.

Maybe some forms submit from the client side to the server side, then server side execute the compute the through the database or other services and they response to results to the client side then that result can modify the X Path because the result from the server side can affect the user interface of the client side.

So maybe that is the reason of the side effects of server side state.

Yes, because not fully supported because crawlers just related to event so in the UI, there is some forms there is submit buttons so in that time they generate random values and click the submit button that time is crawlers can relate with server side result.

🔊 [27:50]

(Professor questioning)

I will investigate that questions later and I will answer the question next time.

And the next step is data entry points.

It means in order to submit data to the server we need input values so the crawlers detect all DOM forms and for each new state extract all form elements in a state.

And for each form calculates a hash code based on the properties.

🔊 [31:00]

And for each new state extract all form elements in a state.

And for each form calculate a hash code based on the form properties.

But they didn't write to the details so this part.

But maybe that's...

And check it form is available in the database because when the on each state they have these steps respectively.

So, in the database there is some already extracted forms in the database.

So check if forms is available in database.



So if forms is available, so if not store the forms, a form completely in the database with the hash code as its ID.

And if available, uses the corresponding custom values in the database to fill the form in the browser and submit.

Upon submission, the resulting state is analyzed recursively.

So these are the full scan process.

For each state, retrieve all candidate clickables.

And fire event on each candidate element and compare DOM tree using edit distance.

And if DOM is changed, make a new state and add to the state-flow graph.

And find the X Path expression of the clickable element.

And add the clickable as an edge to the state-flow graph between state before and after the event.

And analyze forms to make input values and do all this recursively.

This is the approach to full scan process.

It's the comparison method name is levenshtein edit distance, I don't know this technique.

So, it's a...maybe So, I...

🔊 **[34:43]**

Exquisiting applying levenshtein edit distance result is a some...

What is the result of this?

What's the result of the applying this...ah, real number.

How much...same or not.

It's a good question.

Edit distance?

In section 4 and creating states?

Page two hundred twelve.

They just a...they didn't write why they use this...

Yeah, okay.

They test AJAX states through invariants, using invariants.

So, with access to different dynamic DOM states they can check the user interface against different constraints.

So, they use invariants as oracle.

So, they categorize invariant to generic DOM invariants and state machine invariant and application specific invariants.

So generic DOM invariant is a validated DOM and no error messages in DOM and other in consist of these invariants.

A validated DOM is the use the DOM tree obtained after each state change and transform it to the corresponding HTML instance.

A W3C HTML validator serves as oracle.

So they use a diff algorithm to prevent duplicate occurrences of failures.

And the other invariant is a accessibility or link discoverability or security constraint's on the DOM.

But they just wrote about this invariant's name, they didn't explain about the details.

So, it's other invariants.

And the state machine invariants is a identify requirement on the state machine and its transitions.

So there are two invariants, no dead clickables and consistent back-button.

So no dead clickables is the occurrence of dead links.

So error messages from the server are mostly swallowed by the Ajax engines.

🔊 **[41:20]**

So, this is the problem.

So, by listening to the client server request and response traffic after event, dead clickables can be detected.

And the consistent back-button...as I mentioned before, a triggering back-button is ...makes the browser completely leaves the web application.



So, register each state changes with the browser history and they made a framework, so frameworks are appearing with handle this issues.

And final...last invariant is application-specific invariants.

So this is a ...these are the example invariants expressed using X Path in Java.

They ...this is from study two in this paper, they said that it is easy to write, the...just one sentence.

In this application-specific invariant is means the list tag.

Below the list tag there is a DIV tag.

It can be collapsible, so they checked about collapsible DIVs within expandable items.

And second application-specific invariant is a warn about the collapsible items within expandable items...is a list tag and below the UL tag and below and list tag.

They set a...this application-specific invariants to the ...to their study and check to find these invariants in the result.

To test Ajax paths, while running the crawler to derive the state machine can be considered as a first full test pass, the state machine itself can be further used for testing purposes.

To generate test cases from the state machine, they use K shortest paths algorithm from initial to sinks and in this tool, the loops are included once, all transitions coverage.

And finally they transform each path found into a J Unittest case.

And they execute test-case, using a framework they provide a framework to run all the generated tests automatically, using a real web browser and generate the report.

🔊 **[45:00]**

They made a tool, name Crawljax and ATUSA.

ATUSA is based on the Crawljax, Crawljax are crawling through dynamic states and ATUSA architecture can be divided into three phases.

Pre-Crawling and in-Crawling and post-Crawling.

Pre-Crawling occurs after the application has fully been loaded into the browser and in-Crawling occurs after each detected state change different types of invariant can be checked through plug-ins.



And post-Crawling occurs after the crawling process is done and the state-flow graph is inferred fully.

So ATUSA gave some plug-ins to each phase and for each phase, ATUSA provides the tester with specific APIs to implement plug ins for validation and fault detection.

This DOM validator and Test case generator is kind of plug-ins for ATUSA...a plug ins of ATUSA.

This is the architecture of ATUSA.

You can see there is a plug in DOM validator, in-Crawling plug in and test-case generator, post-Crawling plug ins.

In this paper, their goal is to evaluate the fault revealing capabilities and scalability and required manual effort and level of automation of the approach of their tools.

So, research questions is research question 1 is a what is the fault revealing capability of ATUSA?

And research question 2 is a effectiveness of ATUSA.

And research question 3 is the ...what is the automation level and how much manual effort is involved.

🔊 [48:00]

So, study1, the result is the coverage...the tool can detect for 80% of detecting faults and RQ2 effectiveness they cover they coverage server side and client side is 73% and 75%.

And the manual effort is around 30minutes they say that it is a the good result, for the study1.

And study2 is a to find real-life bugs, so this case study involves the development of an...they...authors to developers are more participated in new development of Ajax application in a small commercial project, so ATUSA was used to test the new Ajax interface.

So, they made a specific...application specific invariant and finally they...the tool founder generate DOM invariant and specific, application specific invariant.

In this paper, there is some threat for validity.

In external validity, they conduct 2 case studies, so it's a possible to...in a generalization...in the general, it is not sufficient for ...support their approach, but they conducted another 2 studies.



So, total 4 case studies conducted.

And internal validity is ATUSA encourages to may have errors because they ...that tools use some libraries of ...

So but ...they said that bugs do limit the current applicability of approach but bugs do not affect the validity of results.

🔊 **[51:00]**

So ... this is a last slide.

They have proposed a method for testing Ajax applications automatically through a crawler that can detect clickables and forms and analyze state changes.

And testing framework they made a testing framework that can check the application behavior through generic invariants and application-specific invariants.

And a test suite generation from the inferred state machine.

So, this is the end of my presentation.