Modeling and Reasoning about DOM Events

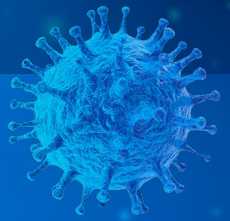
Presentation by Zachary Maizes & Nicole Gerzon

Authors: **Benjamin S. Lerner**, Matthew J. Carroll, Dan P. Kimmel, Hannah Quay-de la Vallee, and Shriram Krishnamurthi, *Brown University*

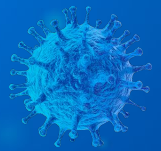
Presented at USENIX Conference on Web App Development

Year of Publication: 2012

Purpose Statement



Current implementations of the document object model are very lackluster and lead to inconsistencies and are implemented in different ways among different browsers. This leads to issues in testing apps and their interactions.



In order to solve this issue, a more specific and modular DOM was modeled in this paper.

Paper Outline

List of Topics

Document Object Model (DOM)

Research Goals

DOM Events

Improvements & Challenges

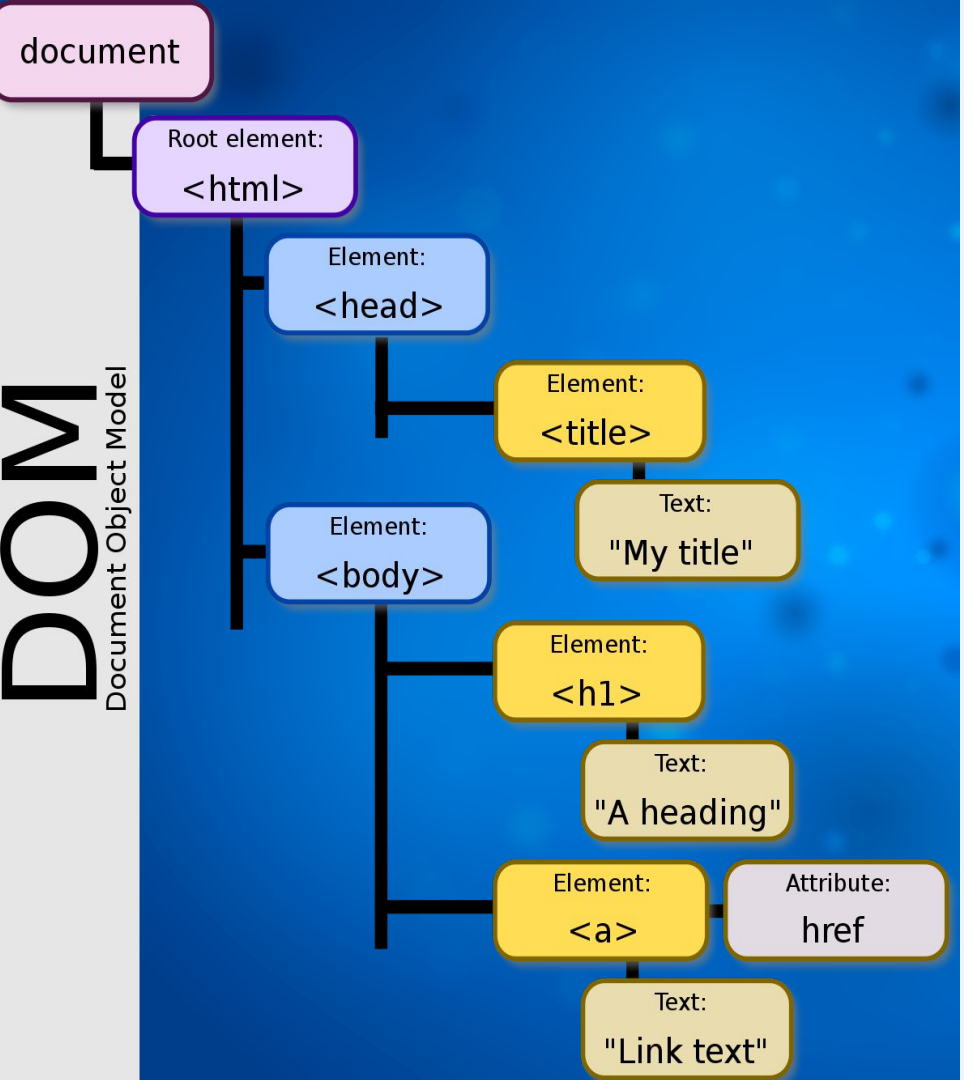
Model Specifics

Event Dispatching

Event Listener Specifics

Properties & Inconsistencies

Event Propagation



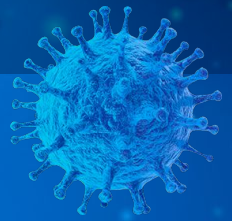
Specifies Behaviours of Triggered Events in Web Pages

“HTML tells events how to propagate, and events tell HTML how to evolve”

Poor Support for Modularity

Extensions may not work together

Research Goals



This research was conducted on various existing implementations of the Document Object Model, focusing on specific web browsers.

The goal of this research was to better understand the DOM and make an improved model that would allow for various enhancements when using it.

Event Flow

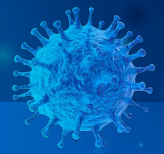
Event listeners have triggers and actions.

These actions go down the DOM until they find their target and then go back up, as illustrated here by the triggered phases when targeting <**span**/> in this model:

<**div**><**p**><**span**/></**p**></**div**>

1. On <**div**/> for phase capture, then
2. On <**p**/> for phase capture, then
3. On <**span**/> for phase target, then
4. On <**p**/> for phase bubble, then
5. On <**div**/> for phase bubble.

Suggested Improvements



Multiple Listeners & Propagation Path:

Adds listeners to a queue instead of overriding

Aborting Event Propagation:

Ability to tell it to stop propagating while currently propagating

Dynamic Listeners:

Remove listeners at any time

Check listeners at each step to accommodate

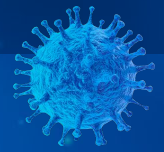
Challenges

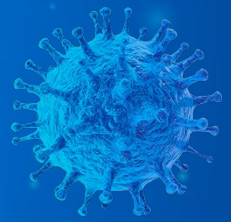
Invited 3rd-Party Code:

Issues arise if ads are given free reign, and if not there are issues with DOM events

Uninvited 3rd-Party Code:

Can’t defend without a model of what to defend against



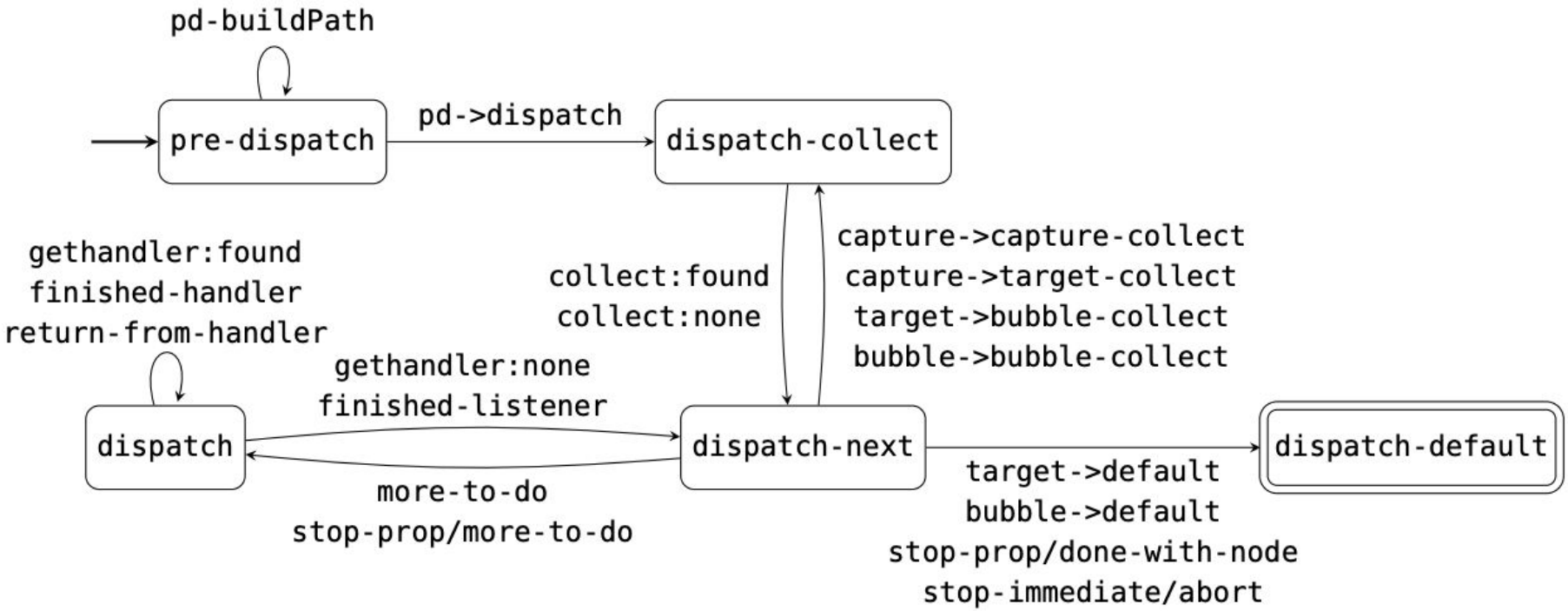


Model Highlights

Listener Execution and Dispatch

- 1200 lines of code
- Implemented using PLT Redex modeling language
- 5 states, 8 transitions, 18 transition conditions
- Felleisen-Heib Style:
  - A particular style of operational semantics

Stages of Dispatch:



Propagation, Execution, & Default

Propagation:

- Determined in pre-dispatch
- Cannot be changed regardless of any mutation on the page

Execution:

- Either in dispatch-next or dispatch
- Listeners may invoke synchronous event dispatches - or cancel the current one
- Allowed to modify the DOM

Default Actions:

- Meta-function when path ends

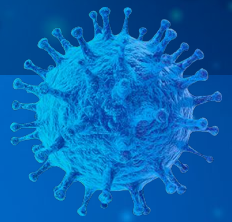
Representing Listeners

- Model separates specification and representation
- Event listeners are called in the order they were installed
  - Must be for either <target> & <capture> or <target> and <bubble>

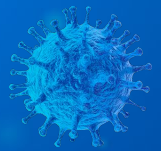
The target node changes depending on the stage of the propagation path

- Lists are updated w/ a meta function

Provable Properties (the highlights)



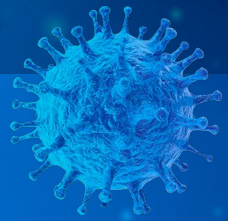
1. No pointers to null
2. The nodes in the heap are tree structured
3. Every heap location is used as a node or listener
4. If dispatch is not stopped, each node will be visited exactly twice

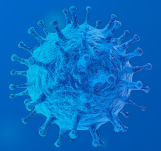


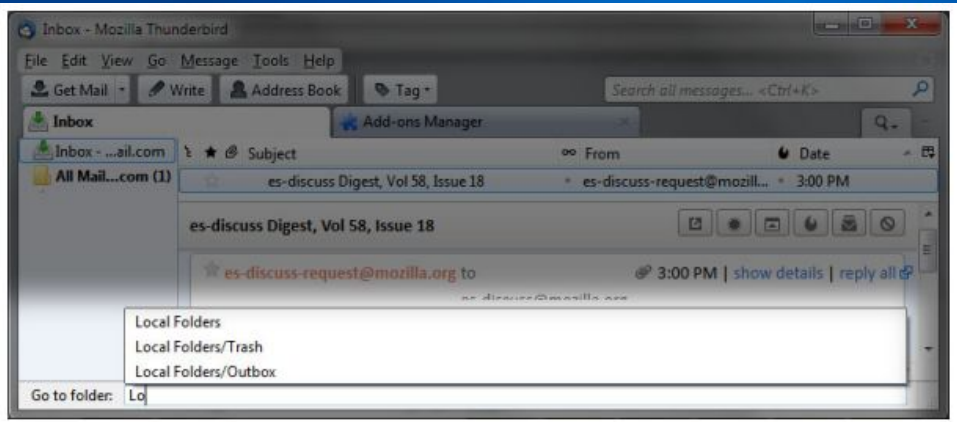
Event Dispatch is **Deterministic**

Finding & Handling Inconsistencies

- The model is the ideal - not what is currently implemented in widely used browsers
  - ex. Legacy handlers
- Handling extension conflicts
  - ex. Thunderbird - Nostalgy







(a) Nostalgy's main interface: a folder selector in the status bar

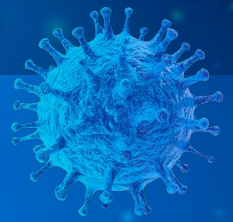
(b) Thunderbird Conversation's main interface: text boxes in the conversation view for quick replies

The extension and original browser offer two conflicting UI's for conversation views

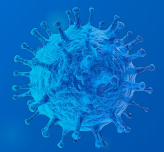
Hot keys in Nostalgy change the UI even more by creating a separate dispatch chain

Possible bugs in Thunderbird which the model helps identify

Sandboxes and Event Propagation

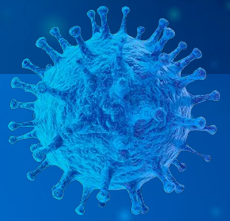
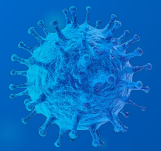


- Sandboxes protect webapps from 3rd party code
- These widgets can prevent DOM events or even invoke code on their own
- No model exists to prove that widgets are sandboxed out of bubble phase interference
- 2 possible solutions!

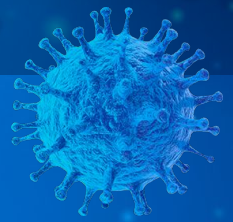


Related + Future Work

- Browser testing
  - Firefox, Chrome, etc...
- Extension implementation
- Future Improvement:
  - Keeping up with changing browsers
  - Fully incorporating Javascript
  - Non-simplified event modeling

Conclusion



This paper describes a model for reasoning about and testing various DOM events in browsers

Such a model is useful in bug testing, security research, and general browser design.