WAI-ARIA

Accessibility APIs and a concrete UI taxonomy

Jaroslav Pullmann
FIT Fraunhofer

Fraunhofer Institut
Angewandte
Informationstechnik

WAI-ARIA Motivation

- generic presentational markup
- missing "semantic cues"
  - data tables vs. formatting tables vs. css
  - opaque "div"
- no means to attach element meta-data
  - identify (interactive) custom widgets by AT
  - know how to handle them according to role
- keyboard focus only on forms and anchors
  - hard to navigate by keyboard

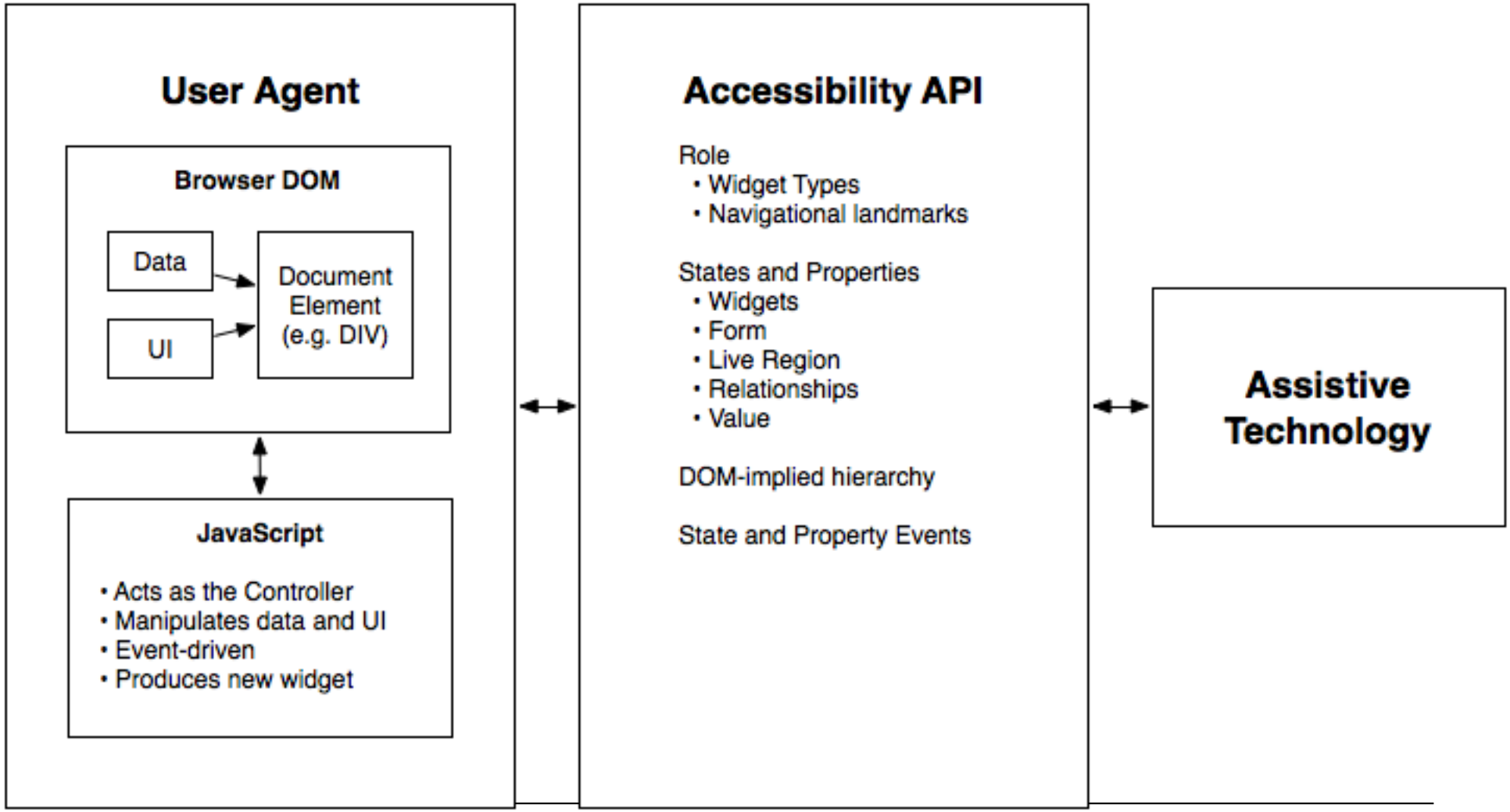
WAI-ARIA Motivation

- Controller
  - agent's default document interpretation
- JavaScript "controller"
  - overrides default user agent behavior on DOM
  - invalidates default accessibility contract

Accessibility API

- contract between
  - Native applications (UA) -> OS's accessibility API
  - Java GUI (Swing) and -> JAPI
- set of interfaces to
  - expose object properties and behavior
  - transmit events

Accessibility API



Accessibility API

– examples:
  - Microsoft Active Accessibility [MSAA]
  - Mac OS X Accessibility Protocol [AXAPI]
  - Gnome Accessibility Toolkit (ATK) [ATK]
  - Java Accessibility API [JAPI]
    - javax.accessibility.*

Java Accessibility API

- Interface Accessible
  - AccessibleContext getAccessibleContext()
- Class AccessibleContext
  - general properties and access methods
  - name, description
  - **role**/generic function, e.g. AccessibleRole.MENU
  - combined component **state**: AccessibleStateSet
    - particular state, e.g. AccessibleState.COLLAPSED
  - overall **relation** set: AccessibilityRelationSet
    - AccessibleRelation.CHILD_NODE_OF, MEMBER_OF

Java Accessibility API Interfaces

- AccessibleComponent
  - GUI surface element (visibility, color, focus)
- AccessibleAction (CLICK, TOGGLE_EXPAND)
  - enumeration and invocation of object's behavior
- AccessibleSelection
  - retrieve and modify currently selected items
- AccessibleText
  - editable (hyper)text value
- AccessibleValue - numeric (min,max)

WAI-ARIA Taxonomy

- describes roles, their (dynamic) properties and states commonly found in accessibility APIs
- role
  - machine readable purpose/function of an element
  - multiple, invariant types
    - <li role="menuitem">Open file…
- augments/overrides the default interpretation
- from XHTML-Role Attribute Module

WAI-ARIA Base Classes

- *abstract classes*
  - top level, organization
  - not part of the API
- base classes
  - 2 aspects - 3 classes
    1) (page) **structure**
    2) functionality: **widgets**
    3) window
       - » unbalanced ?
         single subrole: dialog

roletype
- *structure*
  - *section*
    - region
    - *landmark*
  - *sectionhead*
- *widget*
  - *composite*
  - *input*
    - *range*
    - *select*
- *window*

Some WAI-ARIA Roles

- feedback: alert, alertdialog, status
- segmentation:
  - window, application, dialog
  - document, article, section, group
  - main, presentation, complementary
- meta data:
  - contentinfo - note, defintion,
  - navigation – directory
- hard to distinguish – disjointness criteria ?

WAI-ARIA States

- "managed states"
  - controlled by user agent (focus, keyboard)
  - often have CSS-pseudo classes (:focus)
- "unmanaged states"
  - controlled by the author (through scripting)
  - UI dynamics through CSS attribute selector
    - **[aria-checked="true"]**:before {
      background-image: url(checked.gif);
      }
  - transient, compared to more properties

WAI-ARIA Properties

- intrinsic to roles, do not change often
- attributes
  - aria-label, aria-atomic
- relationships
  - aria-live: designates dynamic content area
  - aria-ownes: constructs containment graphs of external elements to augment DOM
  - aria-readonly, -relevant, -required

Remarks

- Same naming style for states and properties
  - aria-* attributes
- Neither states nor properties defined
  - missing domain / range definition
- Classes are annotated as individuals, not described by property restrictions

```
<owl:Class rdf:ID="grid">
 <role:mustContain rdf:resource="#row"/>
 <role:supportedState rdf:resource="&aria;#aria-level"/>
</owl:Class>
```