ARIA-AT Automation Design Doc

2020-12-18

Summary

Prototype automation of ARIA AT tests with NVDA. Then, develop a cross-AT automation protocol and fully automate ARIA AT testing and results reporting.

Please provide feedback on this document in aria-at issue 349.

Goals

Near term (2020):

- Proof of concept: At least one ARIA-AT test is automated with NVDA/Chrome/Windows 10.
- Start a discussion with Browser Testing and Tools WG and web-platform-tests project about standardizing a WebDriver-like protocol for screen reader automation.
- Stretch: ARIA-AT Automation is running in CI for at least one of w3c/aria-at and nvaccess/nvda projects.
- Stretch: Automated test results integration with aria-at-app.

Longer term (2021 onwards):

- Automate JAWS, VoiceOver (macOS & iOS), Narrator, ChromeVox, TalkBack, Orca.
- Develop a WebDriver-like protocol for screen reader automation.
- ARIA AT tests are automated using the new protocol.
- ARIA AT tests are run daily in NVDA, JAWS, VoiceOver (and others), and test results available in a report with no manual intervention.

Use cases for a WebDriver-like protocol

- Ease of maintenance for the ARIA AT tests
- Web developers can write automated AT tests for their web sites/apps
- Integrate with devtools
- Use for “remote desktop” with AT?

Bocoup
PO Box 961436
Boston, MA 02196
617 379 2752

ARIA-AT Automation Design Doc

2020-12-18

- Integrate with AT CI
- Integrate with browser CI (fuzzing, thread sanitizer, compat)
- Make available to WPT
- Simulators and design tools for web designers/developers

Design

Test format

Each test in aria-at has one JSON file per AT, with a sequence of commands and assertions needed to run the test in that AT. The automated tests are in the same folder as the manual tests, and include the name of the AT in the filename.

Example:
`tests/checkbox/automated/test-01-navigate-to-unchecked-checkbox-reading.nvda.json`

The JSON structure is as follows:

- At the top level is an array
- Each item of the array is an object that represents commands.
- The command object has a single key that represents the name of the command, and a value that is an array containing the arguments to the command. Unless otherwise stated, commands accept one argument.
- Each argument is a string, boolean, number, or null. Unless otherwise stated, the expected type is a string.

Strings are case-sensitive unless otherwise stated.

Example:

```
[
  {
    "nav": [
      "tests/checkbox/reference/two-state-checkbox.html"
    ]
  },
```

```
  {
    "press": [
      "x"
    ]
  },
  {
    "assert_role": [
      "checkbox"
    ]
  }
]
```

The commands are:

- `nav`
    - Navigate the browser to the given URL. The URL may be relative, in which case the base is the root of the git repository.
    - This command must be the first command.
    - Example:
      ```
      {
        "nav": [
          "tests/checkbox/reference/two-state-checkbox.html"
        ]
      }
      ```
- `press`
    - Simulate a key press or a key combination. The output of the AT in response to the key press will be appended to *lastSpeech* as a string to the following assertions. *lastSpeech* is normalized by replacing sequences of whitespace with a single space.
    - Example:
      ```
      {
        "press": [
          "Shift+x"
      ```

```
  ]
}
```

- `press_until_contains`
  - Takes 2 arguments (strings). Simulate a key press or a key combination repeatedly until the AT output contains the second argument. The output of the AT in response to the key presses will be appended to *lastSpeech* as a string to the following assertions. *lastSpeech* is normalized by replacing sequences of whitespace with a single space.
  - Example:

```
{
  "press_until_contains": [
    "h",
    "Checkbox Example (Two State)"
  ]
}
```

- `press_until_role`
  - Takes 2 arguments (strings). Simulate a key press or a key combination repeatedly until the AT output contains the AT-specific representation for the ARIA role given in the second argument. The output of the AT in response to the key presses will be appended to *lastSpeech* as a string to the following assertions. *lastSpeech* is normalized by replacing sequences of whitespace with a single space.
  - Example:

```
{
  "press_until_role": [
    "x",
    "checkbox"
  ]
}
```

- `clear_output`
  - No arguments. Set *lastSpeech* to the empty string.
  - Example:

```
{
```

```
  "clear_output": []
}
```

- `assert_contains`
  - Takes one or two arguments: *expected* (a string), *count* (an integer). Assert that *lastSpeech* contains the *expected* exactly *count* times, or at least once if *count* is not given.
  - Example:

```
{
  "assert_contains": [
    "Checkbox Example (Two State)",
    1
  ]
}
```

- `assert_role`
  - Assert that *lastSpeech* contains the AT-specific representation for the given ARIA role exactly once. For example, NVDA says “check box” (as two words) for the ARIA role `checkbox`. May be used when checking the role.
  - Example:

```
{
  "assert_role": [
    "checkbox"
  ]
}
```

- `assert_state_or_property`
  - Takes 2 arguments: *state_or_property* (a string), *value* (a string). Assert that *lastSpeech* contains the AT-specific representation for the ARIA state or property *state_or_property* with *value*, and does not contain other possible AT-specific representations for that state or property with other possible values (ignoring any that are substrings of the expected value).
  - Example:

```
{
```

```
  "assert_state_or_property": [
    "aria-checked",
    "false"
  ]
}
```

        - With NVDA, this would assert that *lastSpeech* contains “not checked” and does not contain “half checked”. It can’t look for the `true` state, because “checked” is a substring of “not checked”.

- `assert_equals`
    - Assert that *lastSpeech* is equal to the given string. May be used to test the full output.
    - Example:

```
{
  "assert_equals": [
    "Sandwich Condiments grouping list with 4 items
Lettuce check box not checked"
  ]
}
```

Authoring tests

The test format outlined in the previous section is expected to be generated from the same test source material that generates the manual tests. This might need changes to the test source material to make this possible, and as a consequence could result in some changes to the generated manual tests. In particular, for automation, a precise sequence of key presses and assertions are necessary.

Issue: How to solve this needs discussion. #349, #358

ARIA-AT Automation Design Doc

2020-12-18

Running tests

Each AT is expected to implement an AT driver and ARIA-AT test runner.

An **AT driver** is a piece of software that can simulate keypresses to control the AT and record output from the AT in response to those keypresses, as a string.

Issue: The AT driver may also need to be able to change settings and maybe read internal state.

An AT driver could support communication with HTTP or WebSocket, similar to WebDriver for browsers. A protocol for this idea is not yet defined.

Issue: It's unclear if it's possible to have a single protocol that works for all ATs.

Issue: Discuss security considerations.

An **ARIA-AT test runner** is a piece of software that can parse the test format outlined above and forwards commands either to a browser (`nav`) or to the AT driver (`press`), and collect the output from the AT driver (for assertions).

Issue: Expand this section.

Prototype implementation

A prototype implementation of the test format and test runner described in this document for the NVDA screen reader is available for review.

- Test runner implementation for NVDA
- Test file for aria-at