VeriWeb

Automatically Testing Dynamic Web Sites

Michael Benedikt Juliana Freire Patrice Godefroid

Presented by

Krist Wongsuphasawat

Outline

1. Motivation
2. Previous Works
3. What is VeriWeb?
4. VeriWeb Architecture
5. Exploring Algorithm
6. Automatically Filling Form
7. Conclusion

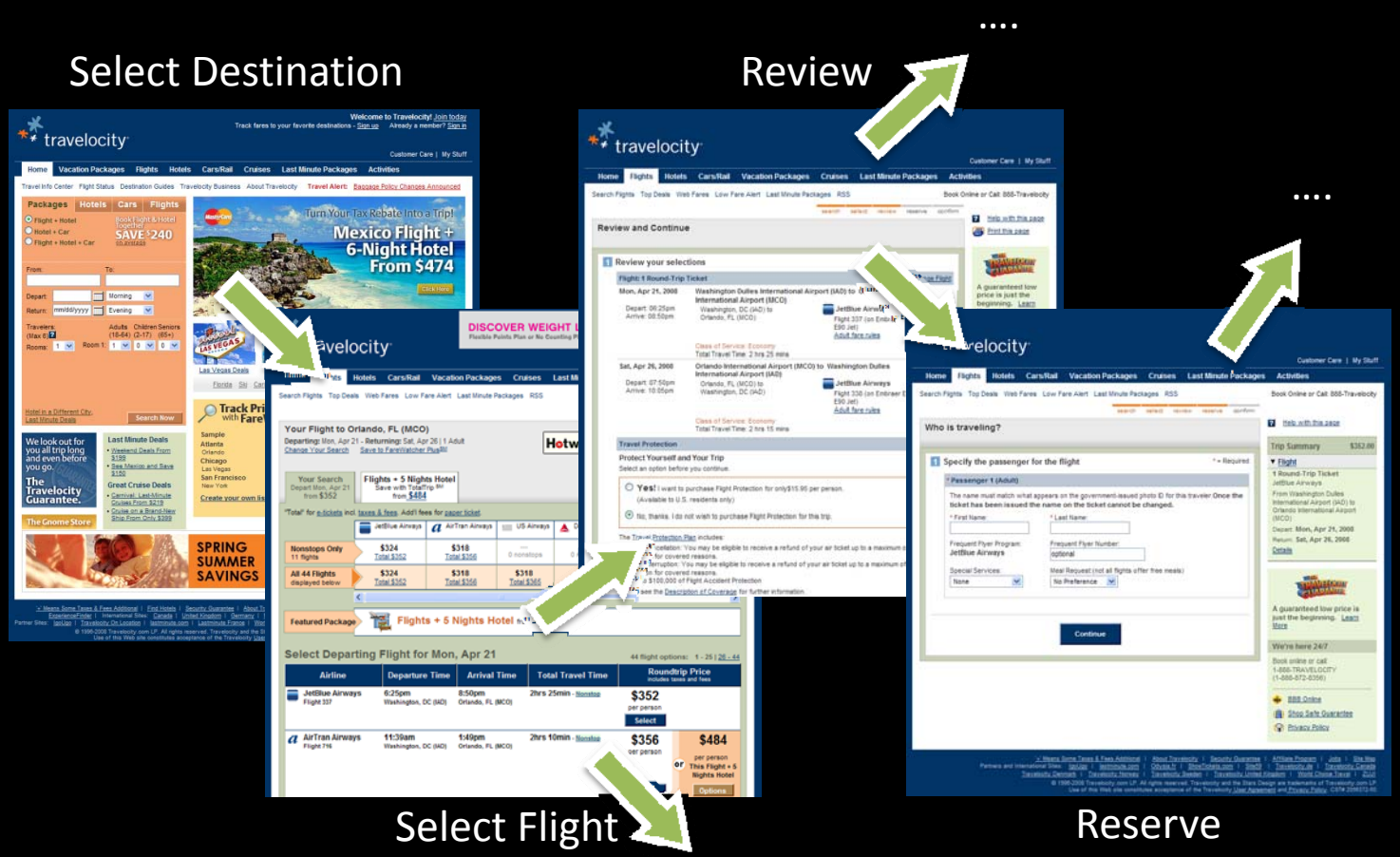
1.1 Motivation

Web pages became very complex.

- 154 Links
- 7 Forms
- >161 Actions?

1.1 Motivation

Web navigation became very complex.



1.1 Motivation

Many things can go wrong.

- Interaction between HTML pages
- Application on pages
  - Java Applet, JavaScript, etc.
- Application on server side
  - Dynamic pages, Database Interfaces, CGI
- Wide variety of server and browsers
- Rapidly Changing Technology

1.2 Challenges

- Web sites are complex applications.
- Web sites are updated often.
  - Hard to keep test suite updated
- Web sites are highly intertwined with the environment
  - e.g. browser, OS, etc.
  - Impossible to test stand-alone
- Users are inexperienced
  - Unexpected behavior

2. Previous Works

- Crawlers

  check links and pages for common errors
  - e.g. LinkCheck, SiteInspector, Weblint, Webtrends
  - broken links, unreachable pages, HTML errors, etc.
  - Go through links
- Capture-replay tools

  explore particular scenario in a dynamic web site
  - e.g. MacroBot, Silktest, etester, etc.
  - Record interactions that can be played back unattended.
  - Must manually create interactions.

3.1 What is VeriWeb?

Exercises multiple scenarios in a web application that are automatically discovered

VeriWeb = Crawlers + Capture-replay tools + more

exploits scenarios that include dynamic portions of sites

Unlike a capture-replay tool, no recording is needed

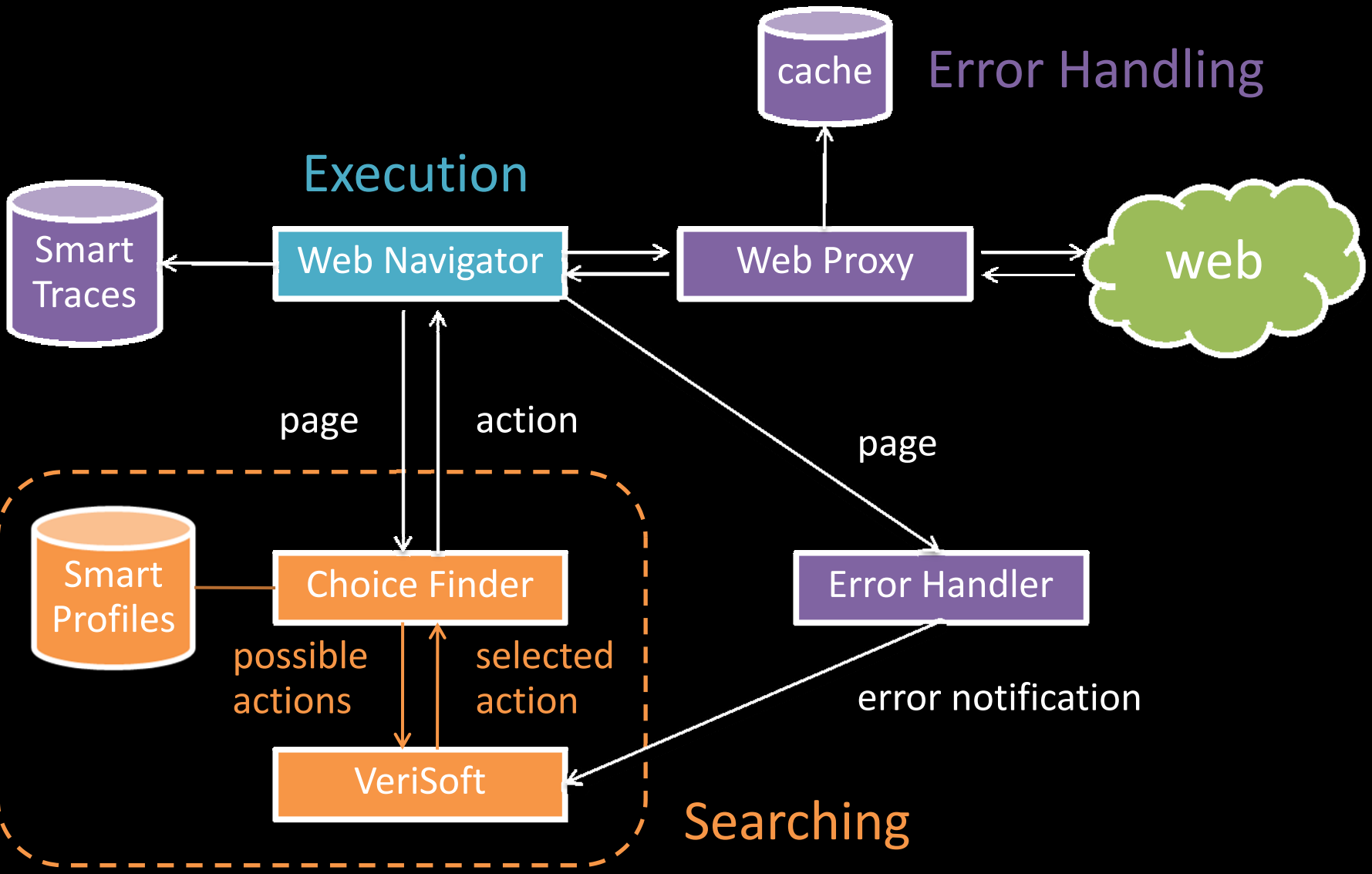
exhaustively searches the site for errors

Unlike a crawler, it can go through forms

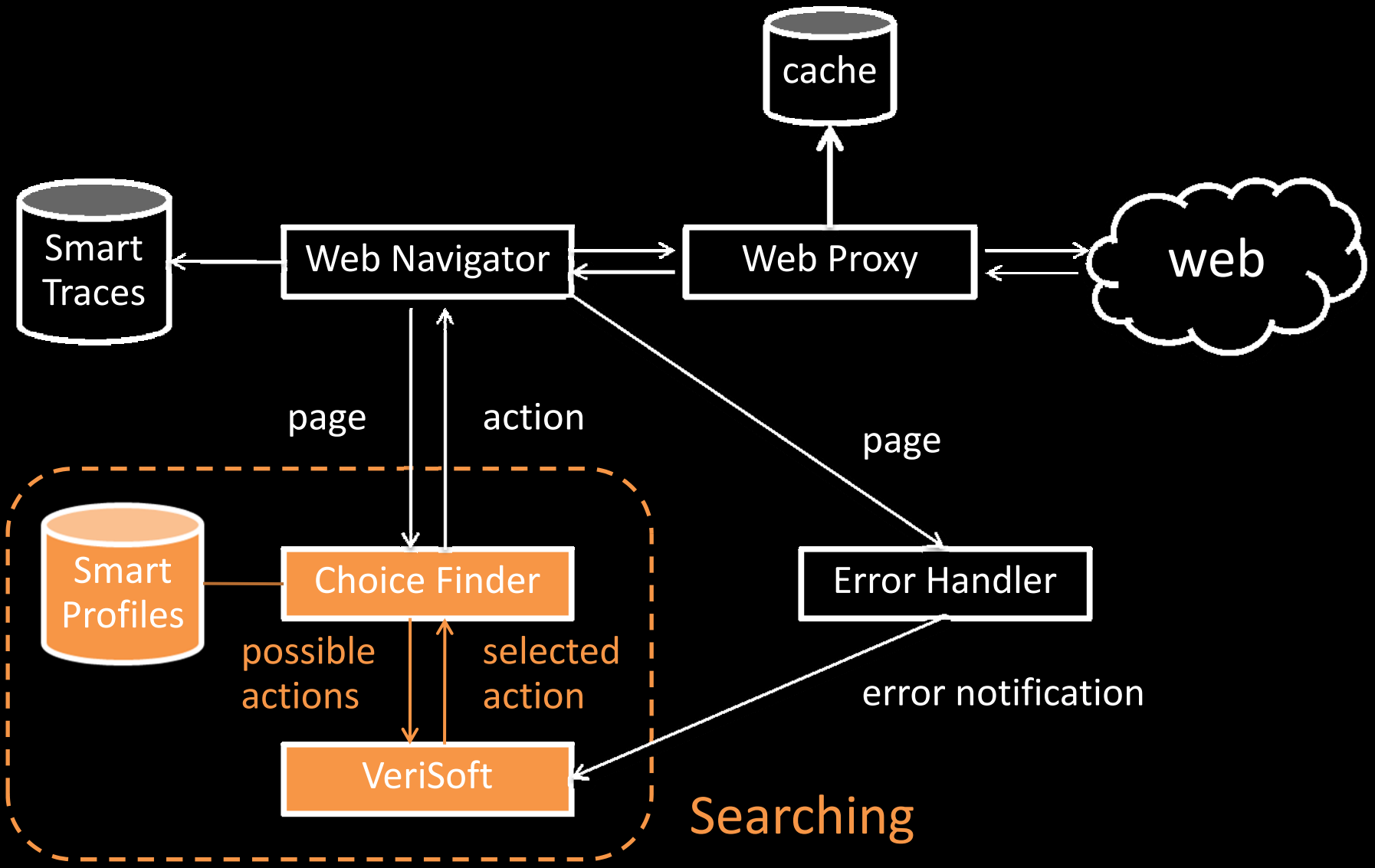
3.2 VeriWeb Features

- Functional testing
  - both static and dynamic elements of Web site
- Regression testing
- Standard correctness checks
  - e.g., broken links, malformed URLs
- Realistic testing
  - Tests are run through a Web browser and closely mimic users' actions

4. VeriWeb Architecture



4.1 Searching



4.1 Searching

Exercise all possible execution paths a user might follow in a web site

- 2 tasks
  1. Action Discovery – ChoiceFinder
  2. Exploration Control - VeriSoft

4.1.1 ChoiceFinder

Find the set of possible actions from that page

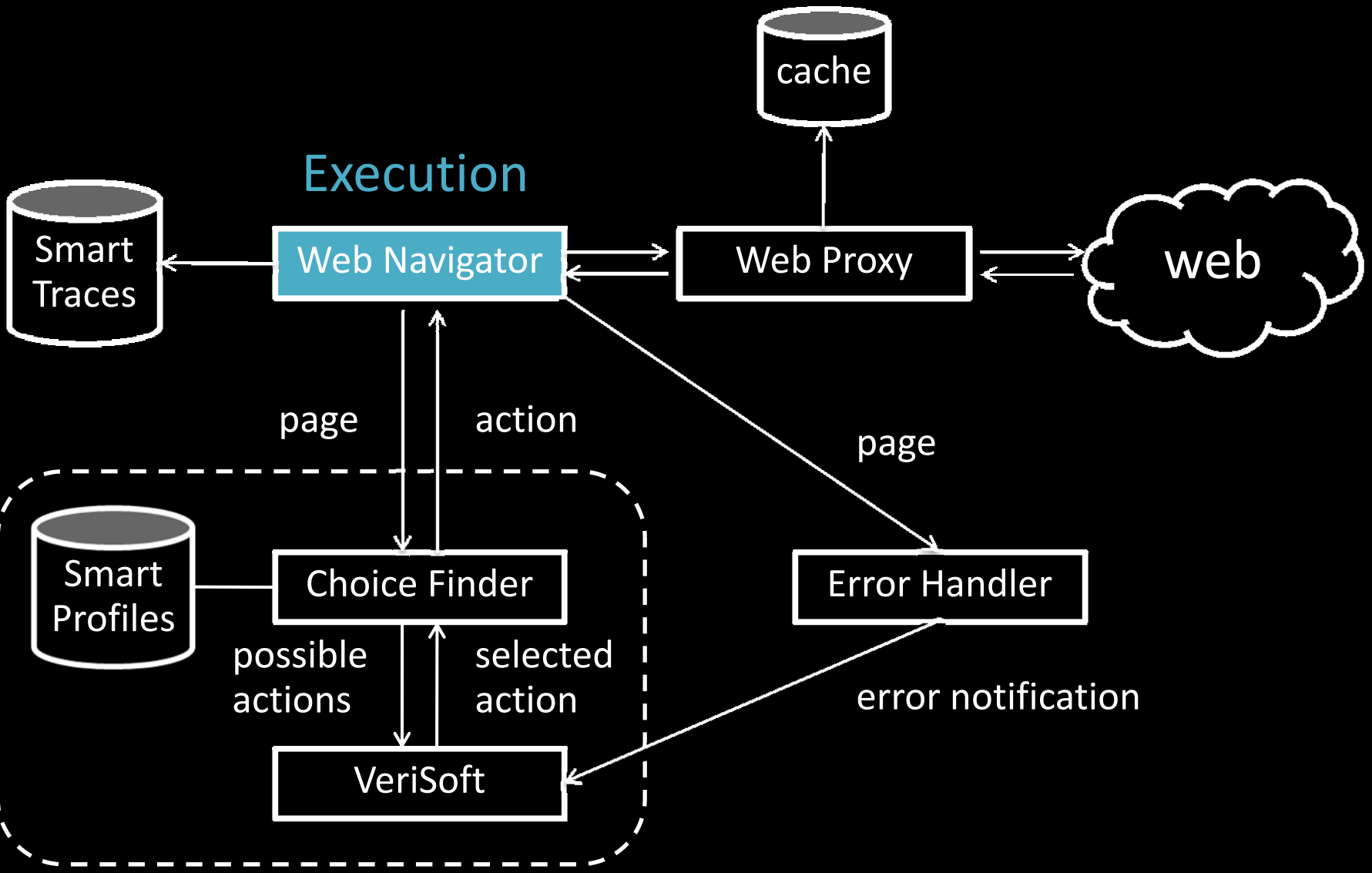
- Analyze HTML to find links, forms, buttons, etc.
- 2 challenges
  1. How to fill in forms?
  2. How to prune the search spaces?

4.1.2 Verisoft

Explore the state spaces

- Guarantee complete coverage of the state space up to some depth
- When error is detected, a scenario leading to error state is saved.
- Saved scenarios can be replayed.

4.2 Execution

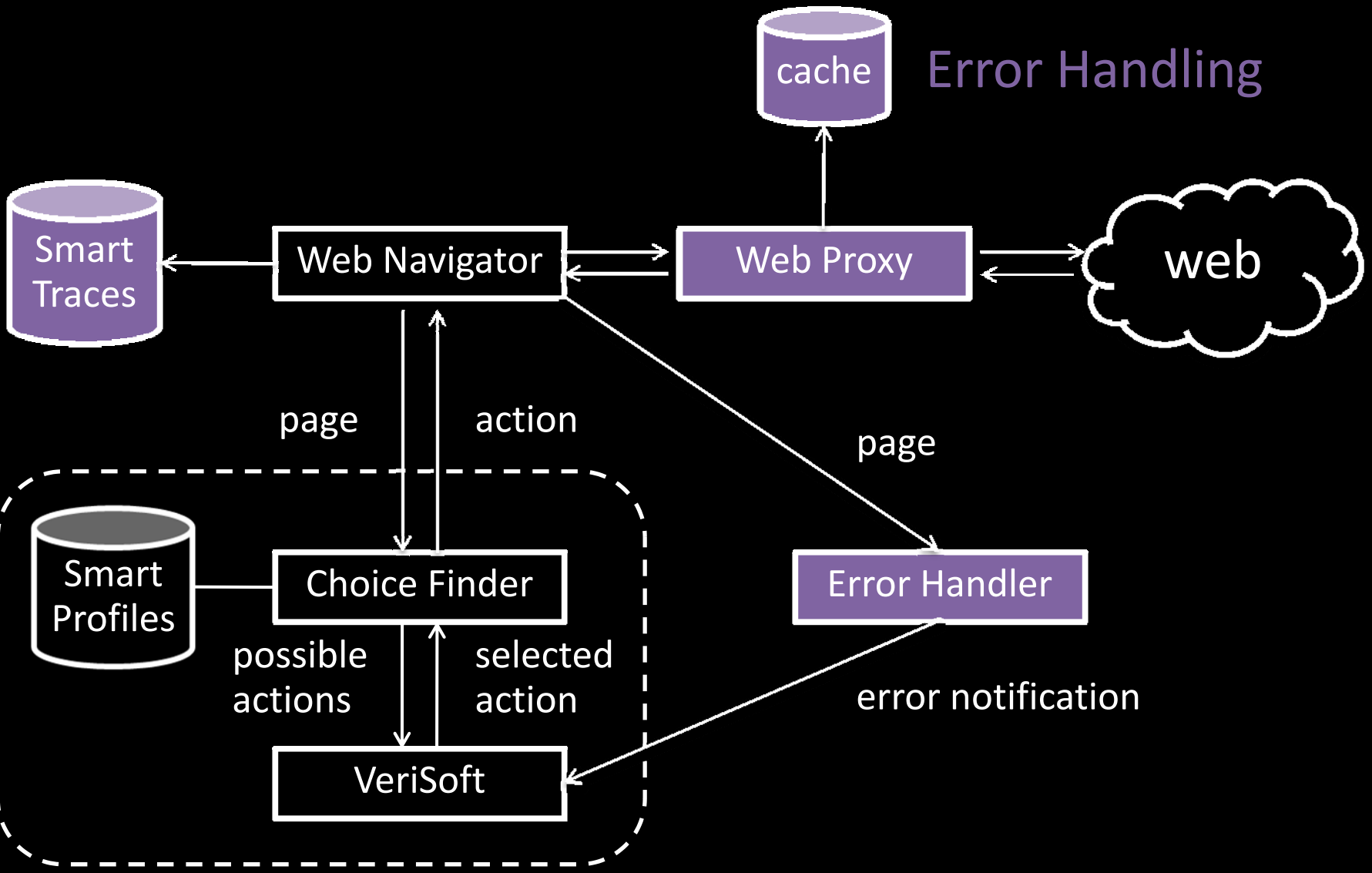


4.2.1 Web Navigator

Mimics what a user would do while interacting with Browser

- Execute actions for selected choice
  - Link: follow link
  - Form: fill and submit form
  - etc.

4.3 Error Handling



4.3.1 Type of Errors

2 Types of errors

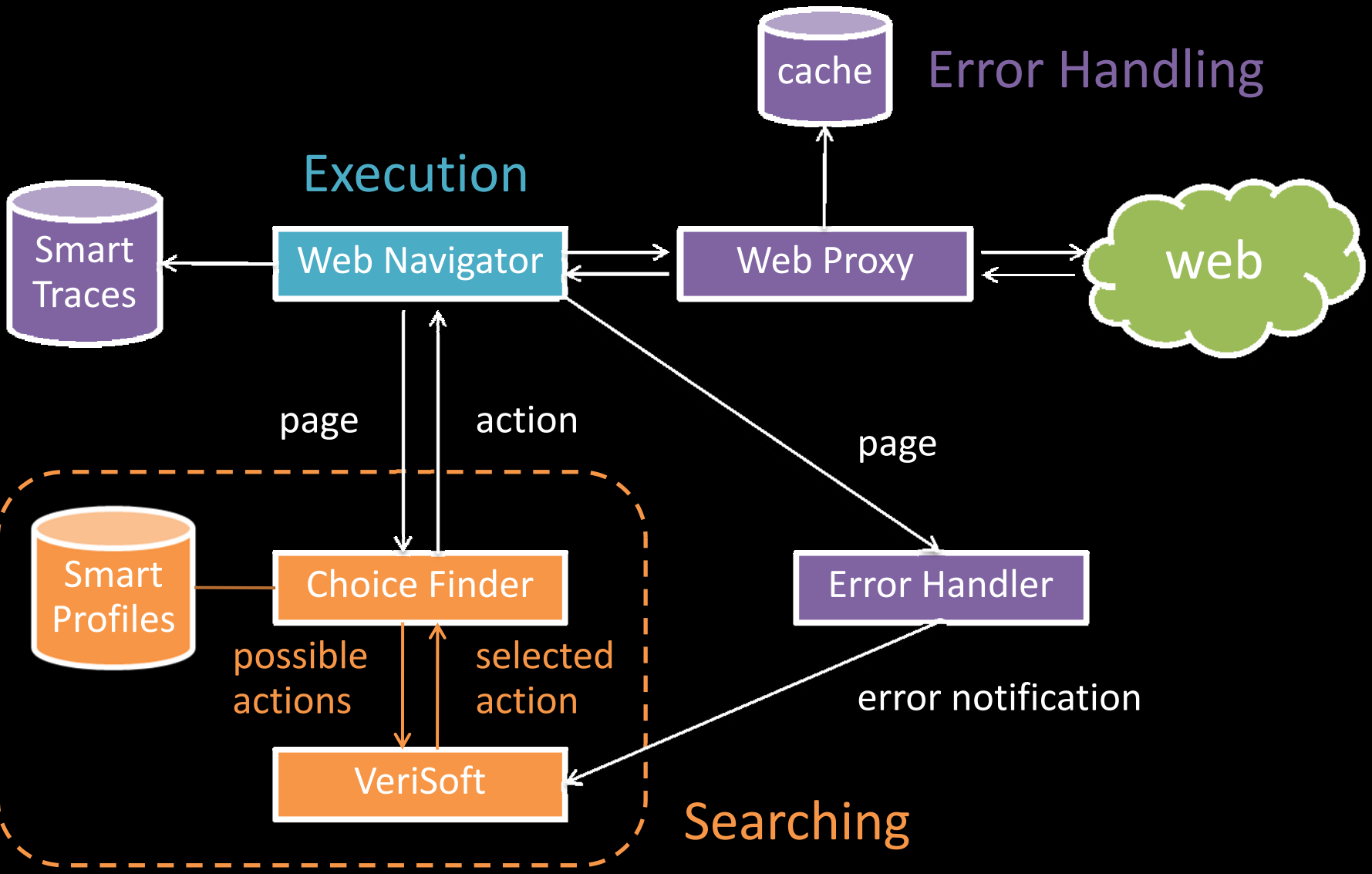
1. Navigation Errors
   - e.g. Failure in retrieving a page
   - Detected by Web Navigator
2. Page Errors
   - Your constraints
   - Detected by Error Handler
   - Various checking modules can be invoked from ErrorHandler

4.3.2 Error Logging

3 Different levels

1. VeriSoft
   - error traces
   - scenario (i.e. state-space path from the starting page)
2. Web Navigator
   - smart traces (smart bookmarks)
   - richer representation of scenario
   - can be replayed by WebVCR
   - robust, can be replayed even web page structural changes
3. Web Proxy
   - cached pages
   - all visited pages or just pages in scenarios that lead to errors
   - offline error detection

4. VeriWeb Architecture



5.1 Exploration Algorithm

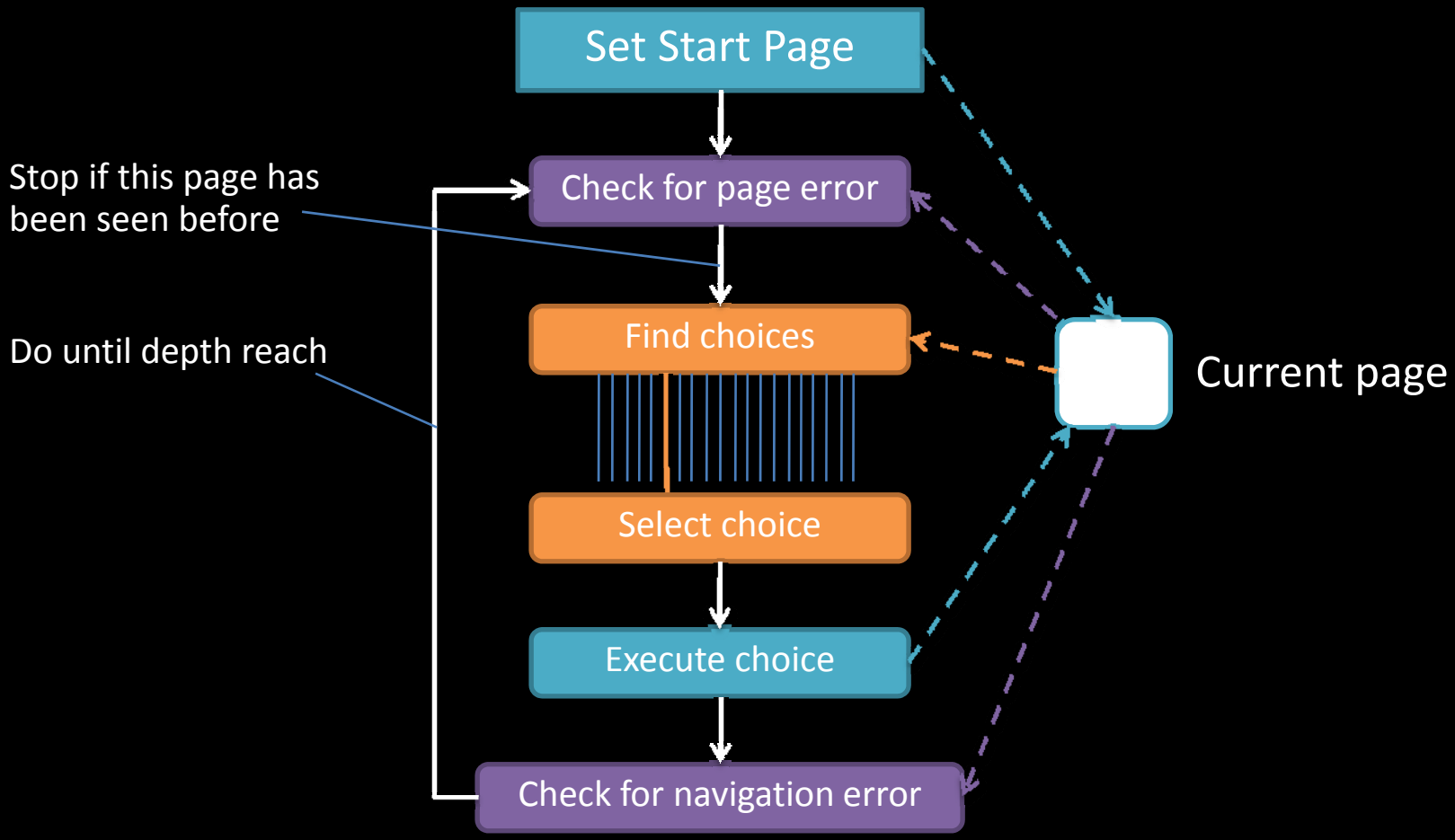
ExploreSite(startingURL,constraints) ⟸ *One run for one Path

```
currentPage = Navigator.load(startingURL);
while (true) {
    error = ErrorHandler(currentPage,constraints);
    if (error.status==true)
        VeriSoft.assert(currentPage,error);
    if (this page has been seen before)
        VeriSoft.abort(currentPage,``cycle'');
    else {
        choices = ChoiceFinder(currentPage);
        selectedChoice = VeriSoft.toss(choices);
        currentPage = Navigator.execute(selectedChoice,choices);
        if (currentPage.error != null)
            VeriSoft.assert(currentPage,error);
    }
  }
}
```

} Page Error

} Navigation Error

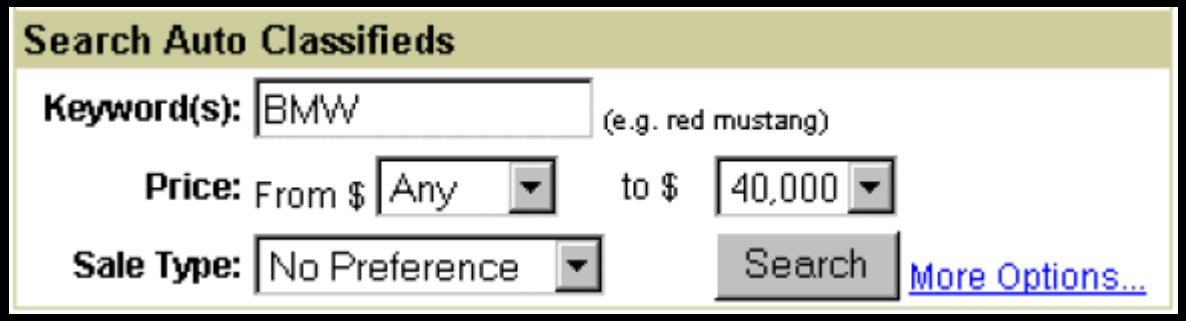
5.1 Exploration Algorithm



5.2 Controlling the search

- Filtering
  - ignore “unimportant” actions
  - e.g. links to PDF , Word documents, external sites, mailto links, etc.
- Cycle detection
  - State vs. stateless: page that look identical may represent different states.
  - e.g., the initial page of Amazon with and without cookies
  - Configurable cycle-detection

6. Automatically filling forms



How to fill this form automatically?

Tester needs to provide some information!

6.1 Smart Profiles

Information used to fill in forms

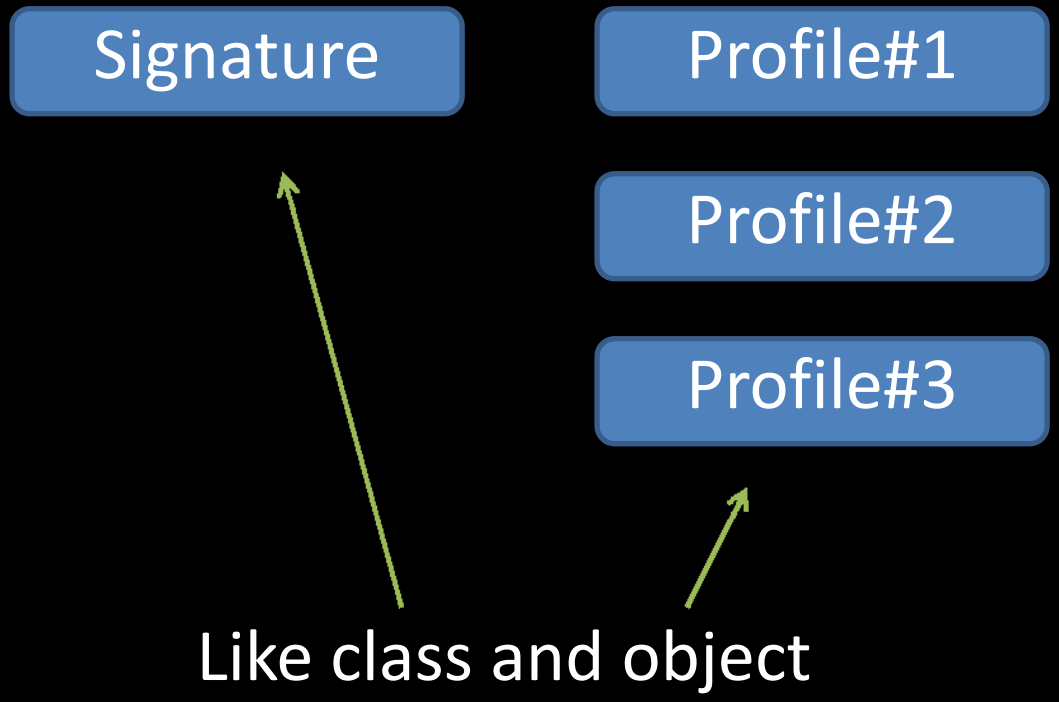
1. Data model level
   - Not form-by-form approach
   - e.g. car sale site
     - Specify a dataset that describe
       - Customer profiles
       - Car models or properties
2. As light-weight as possible
3. independent of the structure of the Web site
4. reusable as structure of Web site changes

6.1 Smart Profiles

XML Schema

```
<!DOCTYPE smartprofile [
<!ELEMENT smartprofile (signature* |
profile*)>
<!ELEMENT signature (name, field+)>
<!ELEMENT field (name,synonym)>
<!ATTLIST field key CDATA>
<!ELEMENT profile (name, signature,
fieldvalue+)>
<!ELEMENT fieldvalue (name,regexp)>
<!ELEMENT regexp (#PCDATA) >
<!ELEMENT name (#PCDATA)>
<!ELEMENT synonym (#PCDATA)>
]>
```

6.1 Smart Profiles



6.1 Smart Profiles

Signature

```
<signature>
    <name> CarType </name>
    <field key=true>
        <name> Make </name>
        <synonym> *carmake* </synonym>
    </field>
    <field key=false>
        <name> Model </name>
    </field>
    <field key=false>
        <name> Year </name>
    </field>
    <field key=false>
        <name> Interior </name>
    </field>
    <field key=false>
        <name> Type </name>
    </fiield>
</signature>
```

Profile

```
<profile>
    <name> Rolls </name>
    <signature> CarType </signature>
    <fieldvalue>
        <name> Make </name>
        <regexp> Rolls-Royce </regexp>
    </fieldvalue>
    <fieldvalue>
        <name> Model </name>
        <regexp> *Corniche* </regexp>
    </fieldvalue>
    <fieldvalue>
        <name> Year </name>
        <regexp> 2002 </regexp>
    </fieldvalue>
    <fieldvalue>
        <name> Type</name>
        <regexp> Convertible* </regexp>
    </fieldvalue>
</profile>
```

6.2 Filling out forms

1. Form Analysis
   - Generate field name for every field
   - Output = Form schema
2. Form Matching
   - Match form schema to smart profiles
   - Output = Candidate profiles
3. Form Completion
   - Match value in candidate profile with field

6.3 More

- Creating profiles is an iterative process
  - System prompts tester for missing information
    - Unknown field

7. Conclusion

As web sites became more complex and change rapidly, we need some automatic and robust way to test them

Contributions of this paper are

- Infrastructure to automate testing of dynamic Web sites
- Search algorithms for automatically exploring all the paths a user might follow in a Web application
- SmartProfiles as a high-level specification of test data to populate forms
- Automatically filling forms strategy.

References

VeriWeb: Automatically testing dynamic web sites

by Michael Benedikt, Juliana Freire, Patrice Godefroid

In: Proceedings of the 11th international world wide web conference (WWW2002)

http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.4.5474

VeriWeb: Presentation Slides

by Juliana Freire

www.cs.utah.edu/~juliana/pub/**veriweb**-www2002-talk.pdf