End-2-End Testing & Monitoring
Basics
Sakuli Use Cases

CI System

Monitoring System

UI-TEST

SYSTEM UNDER TEST

RICH-CLIENT

WEB-CLIENT

BACKEND

DB
Motivation

• Founded February 2014, Open Source (Apache)

• Objective:
  – Combine two open source automation tools (web + native UI)
  – Use the test results in CI and monitoring systems
  – Platform independence (Linux/Windows)

• Application tests from the perspective of the end users
  – Functionality
  – Performance (loading + rendering times)
Referenzen
Testing API
Component 1: **Sahi / Selenium**

**Web testing tool** (sahi.co.in, seleniumhq.org)
method based DOM access:

```javascript
assertContainsText("Logged in as: Sakuli", _div("user_field"));
click(_span("Loaded Run Tables"));
assertExists(_table("cross_tableFixed"));
assertExists(_cell("testing allowed", _rightOf(_span("Name")), _under(_cell("Action"))));
```

```html
<table>
  <tbody>
    <tr>
      <td>Alice</td>
      <td>Bob</td>
    </tr>
  </tbody>
</table>
```
**Visual automation tool** (sikuli.org)
image identification, mouse & keyboard interaction:

```javascript
screen.find("sap_ok").click();
screen.find("sap_ok").right(40).click().type("2223");

var bubble = new Region().waitForImage("bubble.png", 20);
bubble.dragAndDropTo(bubble.left(35)).highlight();
```
Comparison: Sahi/Selenium ➤ Sikuli

**Sikuli**
- ✔️ universal, complete screen
- ✗ (more) resource intensive
- ✗ needs a "unlocked" screen

**Sahi / Selenium**
- ✗ limited on web,
  (no Flash, Java applets...)
- ✔️ fast through DOM navigation
- ✔️ easy to write and stabilize
Common Challenges

• Current monitoring/testing strategy can’t:
  – Execute acceptance or smoke tests after a new release is rolled
  – No automatic check that basic functions of the main processes like e.g. “ticket creation”
  – Also no information available about the state of GUI side.
• Continuous monitoring from the end user perspective
• Selenium Accessor API didn’t work well with some UI components (random IDs)
• Test environments should be containerized and scalable (OpenShift)
Execution Types
Overview Execution Types

• **Native Execution**
  • Supports all end user platforms: Windows, Linux, Mac
  • Installable directly on the end user client
  • Easy JavaScript based API syntax
  • Direct execution of test scripts without compile process

• **Maven Execution**
  • Java Syntax, Maven Dependency
  • Easy integration in maven build cycle
  • Good writing and debug support through well known Java IDEs
  • Optional: Integrate Selenium as Web-Testing-Engine
Overview Execution Types

- **Containerized Execution**
  - Supported Container platforms: Docker Compose, Kubernetes, OpenShift
  - Ready to use E2E environment without installation process
  - Tests run in a real desktop and using a real browser or native client
  - Easy integration in server environments for running headless UI tests
  - Supports JavaScript and Java based tests
  - Scalable environment with all advantages of the container technology
- Comfortable writing and management of test suites
- Direct test execution with integrated live view and logs
- Extended reports for easy error detection
Integrations
OMD Monitoring Integration

- Ready-to-use setup in OpenShift
- Combinable with „traditional“ monitoring checks
- Performance graphs
- Cron scheduled check execution
- Error screenshots
- Mail / Chat Notification
- Live watching possible
CI Pipeline with Jenkins

- OpenShift integrated Jenkins
- Triggerable by Hand
- Triggerable by Webhooks
- Build & run E2E tests
- Error screenshots
- Mail / Chat Notifications
- Live watching possible
Links

- **Testautomation@ConSol**: [https://www.consol.de/it-services/testautomatisierung](https://www.consol.de/it-services/testautomatisierung)

- **Sakuli**:
  - Homepage: [www.sakuli.org](http://www.sakuli.org)
  - Github: [https://github.com/ConSol/sakuli](https://github.com/ConSol/sakuli)
  - Examples: [https://github.com/consol/sakuli-examples](https://github.com/consol/sakuli-examples)
Any questions?
Thank you!