

## 5.2 ServiceOn Integrator

Im folgenden Kapitel sollen die Möglichkeiten verschiedener Testautomatisierungswerkzeuge in Bezug auf die Netzwerkintegrationssoftware ServiceOn Integrator untersucht werden.

Im Vordergrund der Tests steht die grafische Oberfläche des ServiceOn Integrators (SOI), das sogenannte Portal, welches als Javawebstartanwendung<sup>15</sup> sowohl unter Windows als auch unter Linux/Unix ausführbar ist. Der größte Teil der verfügbaren Funktionalitäten, die SOI bietet, ist über diese Oberfläche verfügbar. Der Schwerpunkt liegt daher auf dem Vergleich der beiden GUI-Automatisierungswerkzeuge QF-Test und Marathon.

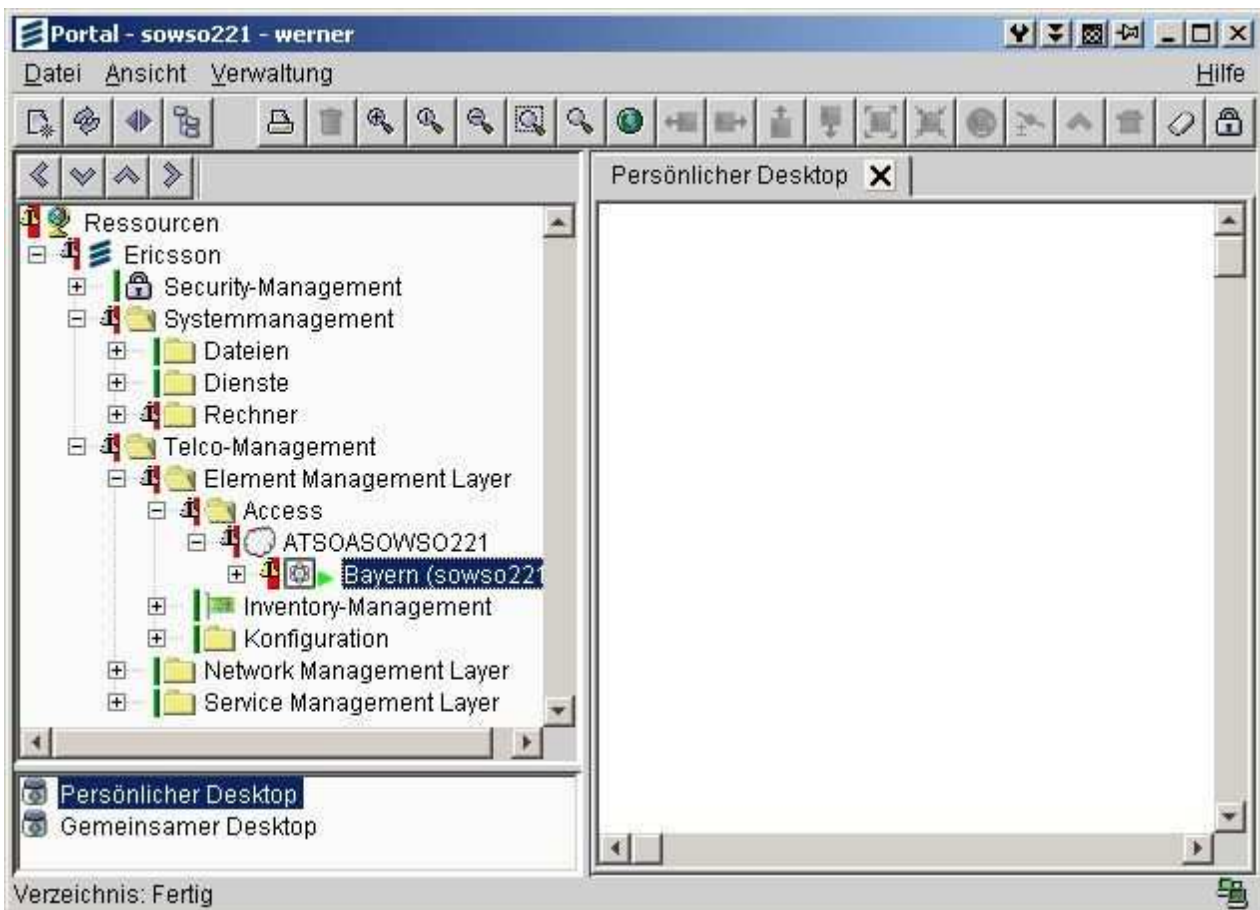


Abbildung 14: SOI Portal

15 <http://java.sun.com/products/javawebstart/index.jsp>

Abbildung 14 zeigt das SOI Portal ohne eine geöffnete Komponente. Der Ressourcenbaum links dient dazu, die einzelnen Komponenten (z.B. die Fehlerverwaltung) zu öffnen.

### 5.2.1 GUI-Automatisierung SOI-Portal

Nachfolgend soll untersucht werden, ob die GUI-Automatisierungswerkzeuge Marathon und QF-Test zur Automatisierung von Interaktionen mit dem SOI-Portal geeignet sind, und welches gegebenenfalls in der Situation des Projekts den größeren Mehrwert bietet.

#### 5.2.1.1 Marathon

Da Marathon keine speziellen Funktionen zum Starten von Webstartanwendungen mitbringt, müssen alle jar-Archive lokal auf den Testrechner gespeichert werden, bevor das Portal mit Marathon gestartet werden kann. Dies kann z.B. durch manuelles Starten des SOP-Loaders (der SOI-spezifischen Java Webstart-Implementierung) geschehen.

Anschließend müssen alle heruntergeladenen jar-Dateien im Klassenpfad des Marathontestprojektes verfügbar gemacht werden, was sich auch mit Hilfe eines Skriptes automatisieren lässt. Schwieriger wird es, dann sämtliche in den jnlp-Dateien<sup>16</sup> definierten Java-Properties zu setzen; allerdings ist auch dies aufgrund der XML-Struktur der jnlp-Dateien mit Hilfe eines Parsers möglich.

Eine weitere Schwierigkeit, die sich im Laufe der Tests der Software herausgestellt hat ist die Art mit der Marathon die Komponentenerkennung handhabt, da gerade die Titel der einzelnen Programmfenster dynamisch erzeugt werden und es mitunter auch vorkommt, dass Texte innerhalb eines Fensters mehrfach verwendet werden.

Die Handhabung des Marathon GUI Testing Tools ist sehr stark entwicklerorientiert, insbesondere durch das komplett in Jython gehaltene Aufnahmeformat (vgl. Listing 8). Dieses erfordert ausführliche Kenntnisse der Skriptsprache, insbesondere wenn die Aufnahmen modularisiert und wiederverwendet werden sollen.

---

<sup>16</sup> XML-Datei entsprechend dem Java Network Launching Protocol (JNLP)

### **5.2.1.2 QF-Test**

Das Starten des SOI-Portals kann bei QF-Test direkt mit Hilfe des SOPLoaders erfolgen, vorausgesetzt das im SOPLoader verwendete JDK ist instrumentiert (vgl. Kap 4.3.4).

Auch die Komponentenerkennung in QF-Test ist stark abhängig von der verwendeten Sprachversion, da im Quelltext des Projekts in der Regel keine Namen für die Komponenten vergeben. Durch die Möglichkeiten, manuell in die Komponentenerkennung einzugreifen, konnten in ersten Tests bereits gute Ergebnisse erzielt werden. Allerdings wurde auch sehr schnell absehbar, dass sich durch die Pflege der Komponentenerkennung ein großer Entwicklungsaufwand in die Tests verlagert, der dort klassisch eher direkt in die Entwicklung des Systems einfließen sollte.

Große Vorteile bringt die Möglichkeit, auf die Komponenten komplexer Strukturelemente wie beispielsweise Baumstrukturen oder Tabellen je nach Situation entweder durch einen absoluten Pfad oder die relative Position zuzugreifen. Dadurch ist es möglich, sowohl gezielt ein Element auszuwählen als auch das Vorhandensein eines Elements mit bestimmten Eigenschaften, allerdings ohne genau bekannten Namen zu überprüfen.

Die Handhabung der QF-Testsuite unterscheidet sich durch die Knotenstruktur radikal von anderen Java-GUI-Automatisierungswerkzeugen. Dieser Aufbau ermöglicht einen einfachen Überblick über die aufgenommenen oder manuell konfigurierten Abläufe. Trotz dieser einfachen Struktur erfordern manche Testszenarien einen Eingriff in Form von Skripten.

Ein Beispiel hierfür ist die Überprüfung von Tabelleninhalten, die nur aus Bildern bestehen. QF-Test bietet hierfür die Möglichkeit, das Abbild einer einzelnen Zelle zu testen, doch dieser Test ist dann von der Breite der Zelle abhängig. Listing 12 liest mit Hilfe des von QF-Test zur Verfügung gestellten Runcontextes `rc` die Breite einer Zelle aus, um diese dann in einer nachgestellten „Drag-And-Dropsequenz“ auf eine definierte Breite zu setzen.

Zusätzlich benötigen auch viele Tests mehr als nur eine einfache Textabfrage und den Vergleich mit einem regulären Ausdruck oder einem Textabschnitt. In diesem Fall muss ebenso auf die Funktionalitäten eines Jythonskriptes zurückgegriffen werden.

```
tablecom=rc.getComponent("Alarme:_Bayern_(sowso221).table")
Cell_0_Width=tablecom.getCellRect(0,0, false).getWidth()
rc.setGlobal("C0W",int(Cell_0_Width))
```

Listing 12: Jythonskript zum Extrahieren der Breite einer Tabellenzelle

Eine weitere für die Automatisierung von SOI interessante Möglichkeit bietet der Knoten zum Ausführen eines Shellkommandos. Dadurch können mit der in Kapitel 5.2 beschriebenen Methode auch auf einem nicht lokal verfügbaren Kernsystem Funktionen aufgerufen werden.

### **5.2.1.3 Fazit**

Die beiden Werkzeuge verfolgen einen sehr unterschiedlichen Ansatz bei der Verwaltung der Tests. Marathon richtet sich mit seiner reinen Quelltextform an Entwickler und Tester mit Entwicklungserfahrung. QF-Test dagegen versucht, Tester nur dann mit Quelltext in Berührung kommen zu lassen, wenn dies zwingend notwendig ist. Durch diese Entkopplung sollte es auch möglich sein, dass Tester mit wenig Programmiererfahrung Tests aufnehmen und modularisieren können. Für die trotzdem anfallenden Skriptanteile würde es dann genügen einzelne Tester mit Jythonkenntnissen zu haben.

Die Komponentenerkennung wird in der aktuellen Situation von keinem der beiden Tools ausreichend standardmäßig gelöst. QF-Test liefert mit der Möglichkeit der eigenen Nameresolver eine Möglichkeit zu einem schnellen Workaround, der punktuell in der Lage ist gute Ergebnisse zu liefern. Langfristig wird man auch hier aber nicht an einer Änderung der Anwendung vorbeikommen. Ich denke, es ist akzeptabel, während der Entwicklung Aufwand zu betreiben, um die Anwendung einfacher testen zu können, also das Design der Anwendung im Hinblick auf eine bessere Testbarkeit zu erweitern.

Insgesamt lässt sich zusammenfassen, dass sich die QF-Testsuite den erfolgversprechenderen Ansatz zum Testen des ServiceOn Integrators bietet.