EATester - software pro automatizovanou simulaci testů

Vedoucí: MFF UK(Nečaský), Integrators (Toman)
Řešitelé: studenti z MFF UK
Datum vypsání: 26.10.2012

1. PŘEHLEDEM

Jakékoli řešení, které má být nainstalováno do produkčního prostředí, musí být nejprve řádně otestováno. Existuje řada různých druhů testů od unit testů až po integrační testy.

Softwarová řešení jsou většinou instalována do produkčního prostředí po fázích, takže s každým releasem (oficiálním uvolněním software do produkčního prostředí) je mnohokrát třeba opakovaně provádět a vyhodnocovat ty samé testovací scénáře.

Automatizovaný software, který umožňuje definovat testovací scénáře a tyto scénáře automatizovaně provádět a vyhodnocovat, velmi usnadňuje práci testerům i celému vývojovému týmu.

V dnešní době existuje řada testovacích nástrojů, kde každý z nich nabízí specifické portfolio funkčností. Jedním za takovýchto software by mohl být i opensource EATester (Emulated Automatic Tester), jehož výsledná podoba by byla zastřešována MFF UK za stranu univerzity a Integrators (www.integrators.cz) za stranu obchodu.

2. SPECIFIKACE

Navrhněte a naprogramujte aplikaci, která bude umět definovat testovací scénáře a tyto scénáře simulovat a vyhodnocovat, a to podle následujících funkčních a nefunkčních požadavků.

2.1. Funkční požadavky

Následuje popis funkčních požadavků:

- testovací scénáře jsou definovány s využitím XML konfigurace
- v rámci jedné definice testů je třeba umožnit definovat libovolně mnoho testovacích scénářů
- testovací scénáře lze spouštět na více strojích (distribuovaná aplikace), při čemž výsledky testů jsou sbírány na jednom stroji. *Poznámka: zde lze hovořit o návrhu server – agenti, kde server sbírá data od agentů, při čemž agenti vykonávají a vyhodnocují testovací scénáře*
- aplikace umí definovat testovací scénáře pomocí grafického rozhraní
- aplikace umí definovat závislosti mezi testovacími scénáři, tedy v jakém pořadí mají být testovací scénáře (např. na základě návratového kódu z předešlých testovacích scénářů)
- aplikace umí sbírat informace o výsledku vykonaných testovacích scénářů a tato výsledná data graficky zobrazovat

- aplikace umí sbírat informace o zátěži (spotřeba CPU, spotřeba paměti nebo disku) během vykonávaných testů a tato výsledná data graficky zobrazovat. *Poznámka: spotřebu paměti, CPU a disku by mělo být možné měřit s využitím služeb operačního systému (tedy i bez spuštění agenta na daném cílovém stroji)*
- aplikace umí definovat reporty (např. v PDF) o vykonaných testovacích scénářích a jejich výsledcích. Reporty by měly sumarizovat: např. počet provedených scénářů, jejich dílčí výsledky a celkový stav testů (počet OK, počet chyb, … atd.)
- aplikace by měla umožňovat sledovat výsledky testů průběžně i během jejich vykonávání

Bonusem:

- nepovinně může být do aplikace integrován jednoduchý skriptovací jazyk, který bude umožňovat spouštět testy z příkazové řádky bez grafického rozhraní a který bude umět vyhodnocovat výsledky testů na základě jednoduchých podmínek

2.2. Nefunkční požadavky

Následuje popis nefunkčních požadavků:

- Aplikace je napsána v C nebo Java. GUI k aplikaci může být napsáno v Java nebo .NET
- Aplikaci je možné používat k funkčnímu testování rozhraní frontendů, a to že např. zavolám URL adresu (např. webovou službu) s definovanými parametry a dostanu návratový kód volání služby
- Aplikaci je možné používat k funkčnímu testování rozhraní backendů, a to že např. zavolám definovaný skript na systémové úrovni (např. python), který mně vrátí návratový kód volání rozhraní
- Aplikaci je možné používat k zátěžovým testům (a to právě např. měření spotřeby CPU, paměti a disku)
- Aplikace by měla umět spouštět testovací scénáře ve více vláknech s jasně definovanými prodlevami vykonávání testů, aby tak bylo možné např. simulovat skutečnou zátěž klientů
- Aplikace by měla být napsána jako standalone aplikace, kterou je možné nainstalovat a spustit v prostředí zákazníka
- Velký důraz je kladen na distribuovanou povahu aplikace (aby bylo testy možné spouštět snadno na více strojících a výsledky testů sbírat na jeden k tomu určený stroj)

3. PŘÍKLADEM

Klasická implementace napojení aplikací (resp. systémů) je v mnoha případech realizována přes jasně definované rozhraní. Toto rozhraní lze specifikovat jako sadu interfaců.

Každý interface je jasně definovaný, do úrovně vstupních a výstupních parametrů. Na základě výběru služeb na frontendu (většinou klikáním na grafické GUI) jsou sestavovány požadavky pro backend a přijímány odpovědi od backendu.

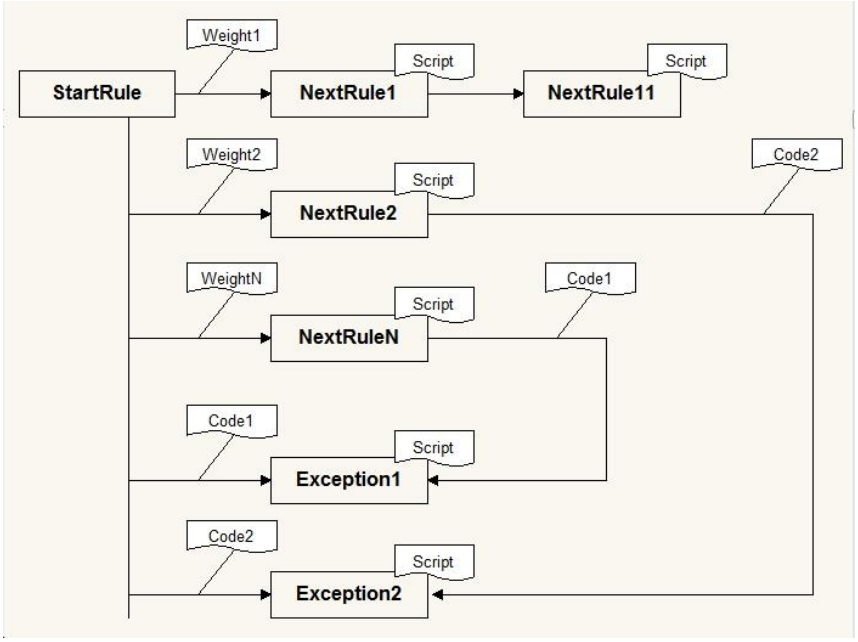
Je jasné, že pomocí aplikace lze toto klikání a výběr služeb simulovat. Na základě pravděpodobnosti lze simulovat výběr služeb a více vlákny lze simulovat více klientů.

Simulace testů je realizována:

- definicí stromové strkutury pravidel volání skriptů nebo funkcí DLL
- definicí vah pro výběr následujícího pravidla volání
- definicí výjimek na základě vrácených kódů z volání skriptu nebo funkce DLL

V rámci volání skriptů nebo volání funkce DLL jsou simulovány funkčnosti jednotlivých služeb.

Volání stromové struktury pravidel a výjimek se periodicky opakuje v definovaných intervalech.



Obrázek 1 Schematický tok testování v EATesteru

Konfigurace jedné instance EATesteru může vypadat např., jak následuje.

```
<Tester>
        <Name>Tester1</Name>
        <CntOfTesterThreads>200</CntOfTesterThreads>
        <CntOfAdminThreads>1</CntOfAdminThreads>
        <PerformTestsST>5000000</PerformTestsST>
        <WhenErrorST>5000000</WhenErrorST>
        <WaitForTCPRequest>1000</WaitForTCPRequest>
        <WaitForTCPReply>10000</WaitForTCPReply>
        <Administration>
                <Port>8053</Port>
                <Hostname>localhost</Hostname>
        </Administration>
        <StartRuleName>Name1</StartRuleName>
</Tester>
```

```
<Rule>
  <Name>NameR2</Name>
  <Weight>0</Weight>
  <Type>1</Type>
  <Path>.\scripts\handle_ruleR2.cmd</Path>
  <Rule>
    <Name>R2_NameCR2</Name>
    <Weight>2</Weight>
    <Type>1</Type>
    <Path>.\scripts\handle_ruleCR2.cmd</Path>
  </Rule>
  <Rule>
    <Name>R2_NameCR3</Name>
    <Weight>1</Weight>
    <Type>1</Type>
    <Path>.\scripts\handle_ruleCR3.cmd</Path>
  </Rule>
  <Rule>
    <Name>NameR3</Name>
  </Rule>
  <Exception>
    <Name>Exception1</Name>
  </Exception>
  <Exception>
    <Name>R2_E2</Name>
    <Type>1</Type>
    <Path>.\scripts\handle_exception_R2E2.cmd</Path>
  </Exception>
</Rule>

<Rule>
  <Name>NameR3</Name>
  <Weight>0</Weight>
  <Type>1</Type>
  <Path>.\scripts\handle_ruleR3.cmd</Path>
</Rule>

<Exception>
  <Name>Exception1</Name>
  <Code>22</Code>
  <Type>1</Type>
  <Path>.\scripts\handle_exception1.cmd</Path>
</Exception>
```