Parallel test components in web application testing

By Bernard Stepien

University of Ottawa

bernard@site.uottawa.ca

Use of parallel test components

- Stress testing: create a large number of test components and measure the rate of failure.
- Resource testing: create multiple web users and verify behavior when several users compete for the same products

Stress testing

```
testcase parallelSessions() runs on MTCType system SystemType {
        var integer num_of_ptcs := 10;
        var PTCType ptcArray[10];
        var integer i := 0;

        //create the PTCs
        for (i:=0; i<num_of_ptcs; i:=i+1) {
                ptcArray[i] := PTCType.create;
        }

        //map the PTCs to the system ports
        for (i:=0; i<num_of_ptcs; i:=i+1) {
                map (ptcArray[i]:web_port, system:system_web_port[i]);
        }

        // start test cases
        for (i:=0; i<num_of_ptcs; i:=i+1) {
                ptcArray[i].start(BaseCaseTest());
        }
        all component.done;
}
```

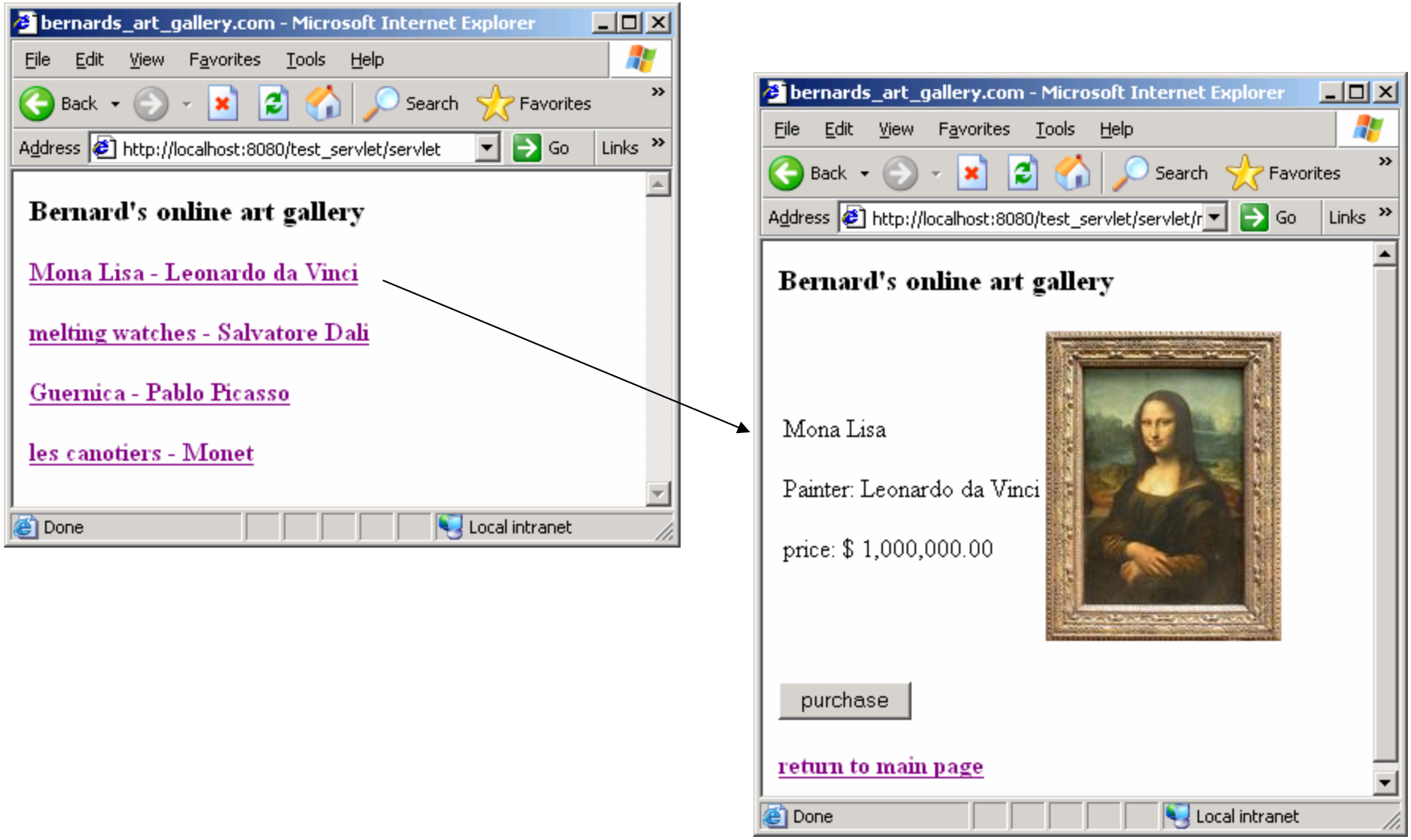
Problems with parallel components

- If the web application deals with inventories, sooner or later a product may become out of stock.
- This needs to be handled with an alternate receive statement:
  - One for the order confirmation
  - One for the out of stock situation
- However, the test needs to be refined to control the out of stock situation and be able to decide if a test really passed or failed and thus resolve non-determinism.

Choosing a parallel testing design

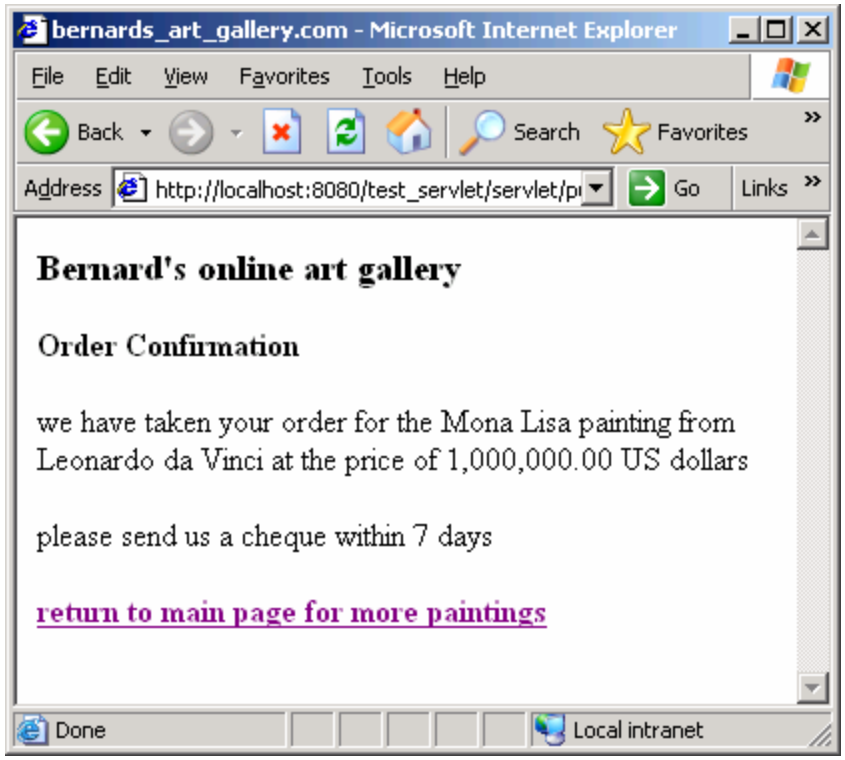
- The stress testing kind of parallel test components is inadequate for inventory based problems.
- The solution is a test configuration where the MTC coordinates the behavior of the PTCs to enable the selection of a appropriate verdict

Navigation example

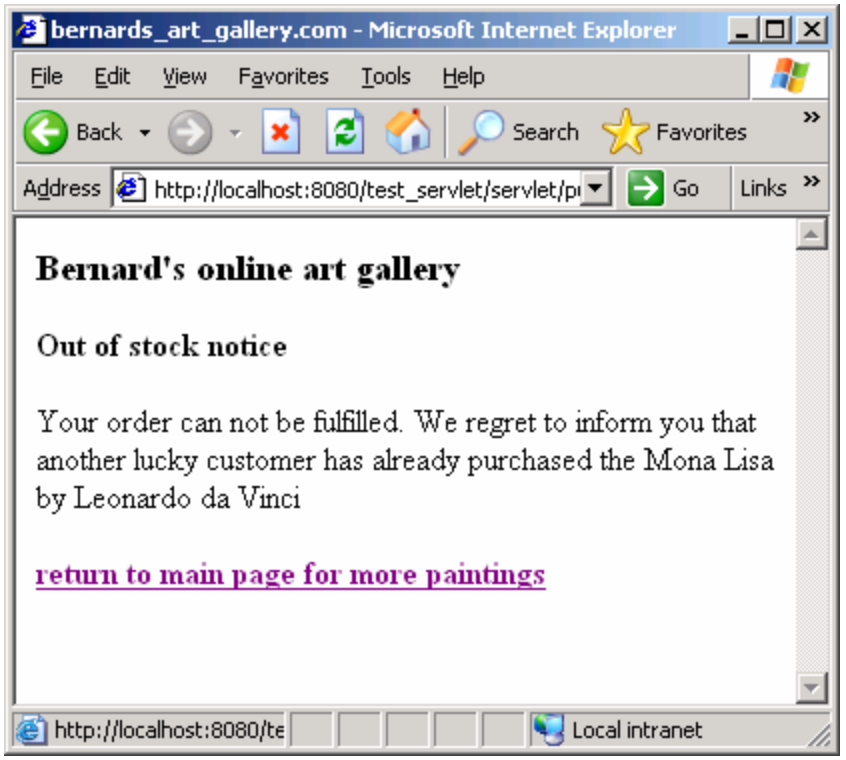


Clicking purchase outcome

Product is available alternative



Product is out of stock alternative



MTC design

```
testcase testcoordinator() runs on MTCType system SystemType {
        …
        var PTCType ptcArray[2];
        …
        for (i:=0; i<num_of_ptcs; i:=i+1) {
                ptcArray[i] := PTCType.create;
        }
        for (i:=0; i<num_of_ptcs; i:=i+1) {
                map (ptcArray[i]:web_port, system:system_web_port[i]);
        }
        //start the PTC's behaviour
        ptcArray[0].start(singleUserTest("user_A"));
        ptcArray[1].start(singleUserTest("user_B"));

        connect(ptcArray[0].coord_port, mtc:coord_port[0]);
        connect(ptcArray[1].coord_port, mtc:coord_port[1]);

        coord_port[0].send(“purchase mona lisa");
        coord_port[0].receive("purchased");

        coord_port[1].send(“purchase mona lisa");
        coord_port[1].receive("soldout");

        setverdict(pass);

        all component.done;
}
```

PTC design

```
function singleUserTest(charstring userID) runs on PTCType {
        var charstring paintingToBuy;

        coord_port.receive(charstring:?) -> value paintingToBuy;

        web_port.send("http://localhost:8080/gallery/servlet");
        web_port.receive(mainPageTemplate) -> value theBrowsePageResult;

        if(paintingToBuy == "mona lisa") {
                clickOnLink("Mona Lisa – Leonardo da Vinci");

                web_port.receive(monalisaTemplate);
                web_port.send(orderFormTemplate);
                alt {
                        [] web_port.receive(orderConfirmationTemplate) {
                                coord_port.send("purchased")
                        }
                        [] web_port.receive(soldoutTemplate) {
                                coord_port.send("soldout")
                        }
                }
        }
}
```

Conclusions

- TTCN-3 is efficient for testing E-Commerce applications
- TTCN-3 has simple but powerful features to coordinate multiple users testing