Building reliable Web applications using atomic actions

Mark C. Little,
Department of Computing Science,
University of Newcastle

Background

- prior to Java, Web applications were concentrated at the server and accessed by a *thin-client*
- fault-tolerance techniques only at server-side
- Java-like technology turns Web into more traditional distributed system, allowing development of complex applications
  - word processors
  - spreadsheets
- fault-tolerance must now encompass browsers

End-to-end transactional guarantees

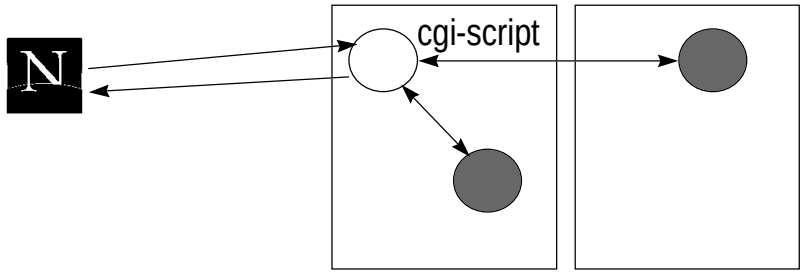
- new Web applications require “all-or-nothing” guarantees between browser and server(s)
  - electronic newspapers
  - credit cards purchases
- require that purchase (e.g., cookie) is delivered if user’s account is debited
  - failures at browser/server may prevent either from occurring
    - application and user’s account could become inconsistent
  - attempts at manual resolution may be difficult
- end-to-end transactional guarantees already used in traditional distributed systems

Atomic actions (atomic transactions)

- possess the following properties:
  - *Atomic*: if interrupted by failure, all effects are undone
  - *Consistent*: transaction effects preseve invariant properties
  - *Isolated*: intermediate states are not visible to other transactions
  - *Durable*: effects of completed transactions are persistent
- transaction manager controls protocol
  - use a two-phase commit protocol to commit/abort changes
  - transactional resources obey protocol
- “all-or-nothing” guarantee
  - build applications without considering failure scenarios

Transactional cgi-scripts

- no end-to-end transactional guarantees
  - suitable only for server-side transactions

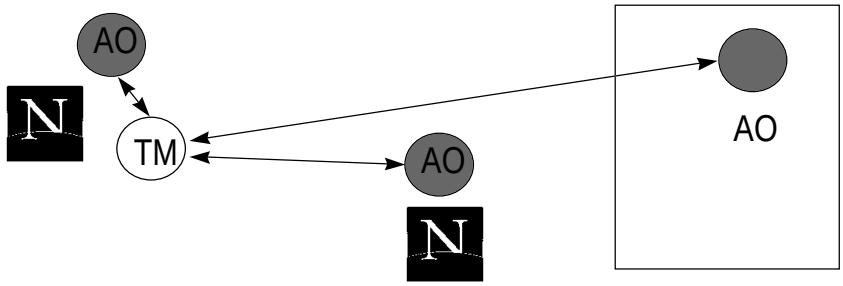


Transactional Web requirements

- support the construction of arbitrary transactional Web applications
  - browser-server
  - server only
  - browser-browser
- should not require changes to browsers
  - too slow
  - affects portability
- *lightweight transactional browsers*
  - *end-to-end guarantees without requiring full transactional infrastructure*

General transactional Web

- support transactional clients
  - enables wider range of applications
  - turns browser into another address space

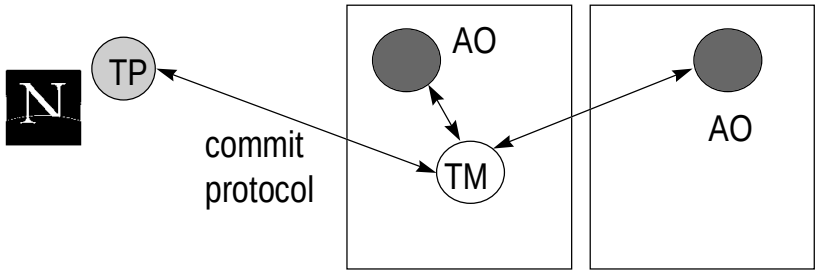


Transactional proxies

- transactional resources must obey transaction protocol
  - provide operations required to participate within commit protocol
- transactional proxies “wrap” legacy code/resources
  - proxy registered with transaction instead of actual resource
  - performs implementation specific work to make resource transactional
  - proxy participates within commit/abort protocol
  - work guaranteed to be completed or undone despite failures
    - simply another resource to transaction manager
  - open commit, reflection

Lightweight transactional applications

- support “thin” clients
  - require minimal client resources
  - concentrate sensitive resources at servers



Browser proxies

- used when browser-side requires end-to-end guarantees
  - e.g., purchase *and* receive cookie
- no transactional resources within browser
  - lightweight, Java support
- main transactional application executes at servers
  - need not be written in Java
- proxy code at server and browser
  - makes browser transactional
    - e.g., write encyrpted cookie during phase 1 and decrypt or remove during phase 2

Typical browser proxy

- browser downloads application
  - contains browser-side proxy code
- server-side registers browser proxy
- applet presents user with operations, e.g., subscribe
  - may require other information, e.g., credit card details
  - results transmitted by proxy
- operations performed at server within atomic action
  - may use transactional resources at other sites, e.g., bank
- transaction manager at server co-ordinates commit
- maintain results until commit completes

W3OTrans toolkit

- supports the construction of fault-tolerant Web applications using objects and actions
  - fully transactional browsers
  - thin, transaction aware browsers
- provides end-to-end transactional guarantees between browser(s) and server(s)
  - top-level, nested, nested-top-level transactions
- built in Java and C++
  - runs on *any* Java-aware browser
- standards compliant (OTS and JTS)

W3OTrans implementation

- object-oriented, based upon Arjuna
  - State management, lock management and AtomicAction classes
  - user classes inherit desired properties, e.g., replication
- stub generation tool
  - C++ /Java client and server stubs
  - can use ORB for distribution
- flexible implementations
  - persistence
  - concurrency control

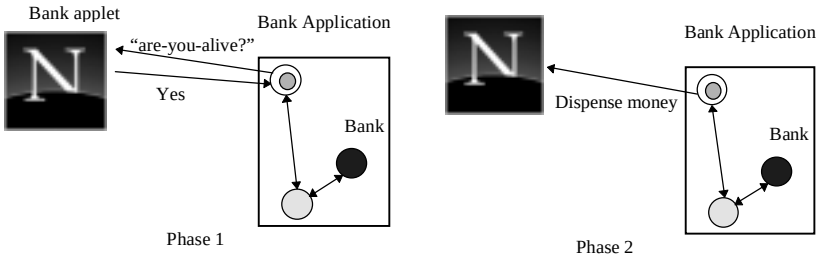
Building W3OTrans applications

- build server-side transactional application
- specify application operations in high-level language
  - tools generate Java object (stub) for application
- automatically generate browser proxy code
  - simple “are-you-alive” protocol
  - applet stub hides browser-side proxy
- server-side application registers browser proxy in each transaction
- proxy automatically invoked during commit/abort protocol
  - transmit result or error response
- crash recovery completes transactions

Bank account example

- insert, inspect and withdraw operations
- require end-to-end guarantees for

Bank account



Standards compliance

- industry standard for transactions is OTS
  - C++ version of W3OTrans is OTS compliant
    - supports nested transactions
    - runs on various Orbs
    - no OMG Recovery Service!
- JTS recently announced
  - not a new standard, but OTS in Java
- W3OTrans is JTS compliant
  - interoperates with OTS objects/applications

Future work

- replication and caching techniques
  - disconnected operation
- groupware
- additional flavours of transactions
  - weaker forms of consistency
- failure detection
- security
  - SET