EOS: Exactly-Once E-Service Middleware

German Shegalov, Gerhard Weikum (University of Saarland, Germany)  
Roger Barga, David Lomet (Microsoft Research, U.S.A.)

PRESS THE SUBMIT BUTTON ONCE AND ONLY ONCE.

REGISTER

PLEASE BE PATIENT, SUBMISSION CAN TAKE UP TO 30 SECONDS

Image by Bruce W. Grant
Banal Examples

- **The same** item is ordered twice in an online-store. In some cases the order has to be cancelled by calling a toll hotline.

- Home banking (PIN/TAN procedure). Each TAN may be used only once.
  - The **first try** to place a money transfer order results in a browser **timeout**.
  - After the subsequent browser refresh the server says: "A TAN was used twice. All other TANs have been frozen. To reactivate, please, contact the nearest branch office".

- 8 health insurance cards for a family of 3 persons.
Why does it happen?

Exactly Once Guarantee is needed for correct execution
Sample n-Tier System

G. Shegalov et. al
EOS: Exactly-Once E-Service Middleware
VLDB’2002 Demo Session 1
Computational Model

§ Set of interacting components: clients, servers etc.
  piecewise deterministic (PWD)
  deterministic replay results in the same end state as originally
  asynchronous events are logged in the proper order
  (timers, interrupts, incoming messages etc.)
  relevant resources are mapped to logical ids

§ Failures are soft (no damage to stable storage)

§ Fail-stop so that only correct data is logged
Components Guarantees

§ **Testable State:** the component can determine if its state include some effects (e.g. receive or send of particular message).

§ **Testable Message:** the component can determine if it sent or received a particular message.

§ **Persistent State:** the component’s state will be available as of some specific time (e.g. when some message was sent).

§ **Persistent Message:** content of a particular message will be available either by periodic resend or explicitly requested.
Component Types

- eXternal Component (XCom) modelling human users
  - cannot provide any of the guarantees

- Persistent Component (PCom)
  - guarantees persistence and testability

- Transactional Component (TCom)
  - guarantees persistence and testability
  - only at transaction boundaries
Interaction Contracts

An **IC** between two components provides a mutually **committed state transition**. Both components guarantee that the state transition is *persistent*.

An IC can be set up *a priori* or *dynamically negotiated*. 
Committed IC (CIC)/Sender

**Sender Obligation 1 (S1):** Persistent Sender State

§ The sender state is as of the time of the message or more recent.

**Sender Obligation 2 (S2):** Persistent Message

§ **S2a:** periodically re-send the message until (perhaps implicitly) released.

§ **S2b:** re-send upon request until (explicitly) released.

**Sender Obligation 3 (S3):** Unique Message

§ Unique message content (headers, timestamps etc.)
Committed IC (CIC)/Receiver

**Receiver Obligation 1 (R1):** Message Dupl. Elimination
§ Dupl. messages detected & eliminated.

**Receiver Obligation 2 (R2):** Persistent Receiver State
§ **R2a:** the receiver makes its state persistent before releasing S2a *(stable interaction, message has been processed).*
§ **R2b:** the receiver makes its state persistent before releasing the sender from S2b *(installed interaction)*
Committed IC (CIC)

**Sender**
1. makePersistentState();
2. msn++;
3. makePersistentMessage(m);
4. periodicResend(m);
5. stopResend(m);

**Receiver**
1. if (notDuplicated(m)) {
2. process(m);
3. makePersistentState();
4. }
5. stableNotification();
6. doSomething();
7. ...
8. if(notInstalled(msn)
9. forceLog();
10. installedNotification();
11. forgetSender(msn);
6. forgetReceiver(msn);
Immediately CIC (ICIC)

$\S$ The receiver makes message and state persistent immediately.

$\S$ The sender is released from S2a and S2b immediately by notification about installed interaction skipping stability notification.
eXternal IC (XIC)

**Output Message Send (X1):**
- A Pcom (client) displays an output message to the external user after having logged it.

**Input Message Receive (X2):**
- The Xcom sends a message via keyboard, mouse etc. to a Pcom (client) and the message is logged **immediately**.
Transactional IC (TIC)/TCom

Atomic state transition (T1):
§ either commit or abort of the transaction and persistence

Faithful reply message (T2):
§ faithful final reply message regarding the outcome of TC1

Persistent commit reply message (T3):
§ persistence of the commit reply message
Transactional IC (TIC)/PCom

Persistent state and commit request message (P1):
- persistence of commit request message and the corresponding state

Unique messages (P2)

No periodic message re-send like in CIC
If ICs are applied then **all failures can be masked** with exception of failures during last external interaction.

**Proof**

*Induction on messages and state commits.*
Prototype System

Internet Explorer <-> Database Server

E-Service 1 <-> Database Server

E-Service 2 <-> Database Server

G. Shegalov et. al  EOS: Exactly-Once E-Service Middleware  VLDB’2002 Demo Session 1
Implementation Overview

Client (web browser) Recovery:
§ DHTML event handling
§ client persistence with XML store
§ JScript

Application Server Recovery:
§ enhanced PHP session management
§ enhanced PHP ODBC support

Database Server Recovery:
§ enhanced ODBC driver manager
§ stored procedures

Phoenix/ODBC style
How to log?

1. `<sdk:logger style='behavior:url(#default#userData);'
2. id='pagestate'
3. type='hidden'>
4. `</sdk:logger>`
5.
6. `<script language="JavaScript">
7. pagestate.load(installationPoint);
8. pagestate.save(installationPoint);
9. `</script>`
Web Browser Recovery (II)

What to log?

1. `var inputFields = document.all.tags('input');`
2. `var htmlForms = document.forms;`
3. `for (var i=0; i < inputFields.length; i++) {
   inputFields[i].attachEvent('onPropertyChange', updatePageState);
}`
4. `for (var i=0; i < htmlForms.length; i++) {
   htmlForms[i].attachEvent('onSubmit', logFormSubmission);
   htmlForms[i].attachEvent('onSubmit', periodicResend);
}`

Compromise 1: no way to install interaction on the browser side, i.e., browser recovery always depends on PHP.
Web AppServer Recovery

1. if( http_reply = lookUpReplyLog(sess_id, msn + 1) ) {
2.     sendToBrowser(http_reply);
3.     exit(0); // nothing to do anymore
4. } //end if
5. //there is no reply log entry yet
6. loadSessionData(session_id, msn);
17. //execute servlet,
18. //add new msn and JavaScript code for XIC
19. http_reply =
20.     executeScript(http_request, sess_id, ++msn, JSCode);
21. // create new installation point
22. saveSessionData(session_id, msn);
23. saveLastUrl(sess_id, http_request->url);
24. addToReplyLog(http_reply, msn);
25. sendToBrowser(http_reply);
Web AppServer Recovery (II)

Compromise 2: no way to request the original request message from the client after a crash on the web server

Compromise 3: no sense to periodically re-send the http reply. The PHP engine will have to wait for the browser to start recovery

Compromise 4: some users will never return after a failure. We need to define heuristic expiration period for garbage-collection
Stress Test Results

Overhead less than 0.1 sec even for 10 steps session

Accepted
Open Issues

- Web application considerations:
  - identifying and killing user-cancelled requests within PHP Engine

- Many application-specific optimization opportunities for application-independent recovery.
Summary

**EOS** provides a framework for *application-independent* application *recovery*

- (almost) perfect failure masking
- provably correct handling of failures
  - exactly-once semantics
- faster software development by designing new applications as *if there were no failures* at all.
- almost *no changes* in existing legacy software
- acceptable overhead