Compensations and Runtime Monitoring

Gordon J. Pace
Christian Colombo
University of Malta
November 2009
Motivation

- More widespread use of SOA, dynamic service composition, long-lived transactions, system-of-systems architectures lead to greater need for handling failure as part of a system’s normal behaviour.

- Catering for failure of components is becoming more important in various scenarios:
  - Systems built of separate components may not be able to trust the success of the constituent parts.
  - Components may be discovered, invoked at runtime, not knowing enough about them at compile time.
  - Invoking multiple services (for the same computation) and using the first result would require undoing the other partial transactions.
Compensation vs Reparation

- **Reparation**
  - Classical exception and error handling
  - What to do to make up for the fact that the current block has failed
  - Local by its very nature

- **Compensation**
  - What to do if something else fails later on and I have to ‘undo’ this block
  - Requires storing dynamic context
An Example of Compensations

- The bog-standard example: A customer buying books from an on-line bookshop
- The service proceeds as follows:
  - The customer commits an order
  - The bookshop gets payment from the customer
  - The bookshop orders a courier
  - The books are identified in the warehouse
  - The books are packed
  - The books are posted to the customer
- But errors may occur at various points in the process:
  - The bookshop realizes that one of the books is not in stock
  - The credit card payment may fail
  - The customer may cancel an order while still being processed
An Example of Compensations

1. Choose books
   - Give credit card details
   - Place order

   - Find book in warehouse
   - Find book in warehouse
   - Find book in warehouse
   - Confirm payment

   - Pack books
   - Order courier

   - Post order
The Question

- How can we use information about compensations in a system to support or strengthen runtime verification?
Further Motivation

- We are applying runtime verification techniques to an industrial case study system for financial transactions.
- Ixaris Systems Ltd are provide online payments solutions and virtual credit card provider.
- Their systems already include an implementation of compensable actions and rollbacks to handle long-lived transactions.
Challenges of Compensations

Actions performed before the failure occurs have to be “undone”: 
Challenges of Compensations

Actions performed before the failure occurs have to be “undone”: 
Challenges of Compensations

Actions performed before the failure occurs have to be “undone”:

\[ \frac{a}{b} n a^nb^n \]
Challenges of Compensations

Actions performed before the failure occurs have to be “undone”:

\[ a/(b/c) \]

\[ a^n b^n c^n \]
Challenges of Compensations

**Actions performed before the failure occurs have to be “undone”:**

- Undoing actions may involve doing something other than the inverse of the forward action:
  - chargeAcct == charge(€1) / refund(95c)

- Compensations may be nested:
  - ( ... ) / (destroyAcct / recreateBlacklistedAcct; ...)

- Compensations may be overridden:
  - ( createAcct; chargeAcct; makeOrder ) / destroyAcct; verifyClient

- Compensations may be scoped:
  - payment; { advertisment }; delivery

- Parallelizing parts of the process makes compensation handling more involved
So Many Logics, So Little Time

- There are various flavours of compensations:
  - Process calculus style (CSP, pi-calculus based)
  - Petri net based
  - Language-based approaches
  - Deontic logics
Remember the Question?

- How can we use information about compensations in a system to support or strengthen runtime verification?
Online Runtime Verification

- Instrumented code of the monitor $M$ is added within the running system $S$, sharing the same address space.
- Running synchronously with the system.
- As soon as a problem is identified, mitigation may occur without the system running further.
- But, we are effectively monitoring $S \parallel M$, not $S$.
- The overheads and interaction with the system are not always acceptable in an industrial setting.
Offline Runtime Verification

- The system produces a log to run the verification code on at a later stage.
- Running asynchronously with the system.
- Verification much more acceptable and faithful since the logging code is typically much more lightweight than the monitoring code.
- But by the time a problem is identified it may already been too late.
Quasi-Online Runtime Verification

- Monitoring is identical to offline monitoring:
  - The system produces a log at runtime.
  - The monitors run concurrently but on separate address spaces.
  - May not be in sync.
But when a problem is identified:

- The monitor may stop the system, and
- Use compensations to undo the actions performed by the system in the meantime.
In this example we would want to perform a compensation for:
- transfer; deposit; pay
Quasi-Online Runtime Verification

- Compensate for actions:
  - up to the point which the system reached,
  - regressing back to just before or after the error.
- Compensations may be specified by either:
  - the system, or
  - as part of the properties themselves.
- The major challenge is resuming the system from the point where it was ‘rewound’ to.
Runtime Verification and LARVA

GLOBAL {
  VARIABLES { Clock t; int c = 0; }
  EVENTS {
    interact() = {*.action()}
    t30() = {t@30*60}
    ...
  }
  PROPERTY users {
    STATES {
      BAD { inactive badlogins }
      NORMAL { loggedin }
      STARTING { loggedout }
    }
    TRANSITIONS {
      loggedout -> loggedin [goodlogin\t.reset();]
      loggedout -> loggedout [badlogin\c++;
      ...
    }
  }
}
Compensation Automata

- Extends automata with compensations using hierarchical automata with three structuring elements:
  - Compensation declaration to enact a compensation of an automaton.
  - Deviation to redirect compensations.
  - Scoping of compensations.
Compensation Automata

- pay(€1)
- rfnd(75c)
- finalise
- pack
- unpack
- addOrder
- notify
Compensations in Financial Transaction Systems

- We are exploring the use of LARVA with compensations to monitor transactions handled by the systems built at Ixaris.

- Using compensations as specifications of expected behaviour and to specify recovery actions can already be done by translating into base LARVA.

- We are exploring the quasi-online approach, in which we use the compensations already built in the system. Major challenges are:
  - *Resuming the system after recovery (easy on a transaction by transaction system, but tough otherwise).*
  - *Making sure that rollbacks induced by the monitors do not interfere with rollbacks induced by the system.*
Reactivity and Compensations

- How can compensation be encoded in a reactive setting?
- If the monitor lags behind the system by at most $n$ time units, compensation handling uses bounded space
- What if this is not the case?
  - Enable the monitor to pause the system
  - Skim over parts of the trace
  - Breaking loops and/or using additional properties (e.g., idempotency of certain compensations)
Conclusions and Future Directions

- The system is up and running at Ixaris – monitoring for properties and compensations but still working on the quasi-online approach.
- Naïve implementation of compensations may induce unbounded overhead.
- Investigating adding real-time.
Monitoring of Financial Transaction Systems

- **Life cycle**
  - Frozen or reclaimed credit cards cannot be used in financial transactions.
  - The states in the life-cycle of an entity (e.g., user, credit card) are correctly traversed, i.e., in the correct order.

- **Real-time**
  - After six months (but not before) of user inactivity, the user should be put in a dormant state.
  - After a year (but not before) of user inactivity, the user should be removed from the system.

- **Access rights**
  - A user must have the necessary right before loading money onto the credit card.
  - A user must have the necessary right before transferring money from a card to another.

- **Amounts**
  - The number of times a user loads money to a credit card should not exceed the stipulated amount for a day or a month.
  - The total sum of money loaded should not exceed the stipulated limit for a day or for a month.