Compensations and Runtime Verification

Gordon J. Pace
Christian Colombo
University of Malta
September 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.
  - Sometimes it is simpler to describe a system in terms of what to do, and how to undo in case of failure.

Motivation

- The questions we are addressing are:
  - How can runtime verification support compensations in a system (without compensations)?
  - 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.

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 (eg. 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.

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

Challenges of Compensations

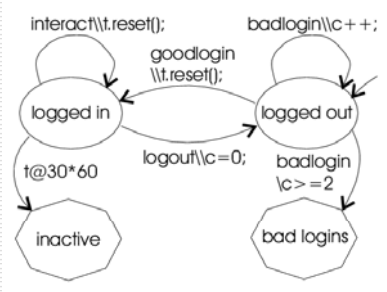
Actions performed before the failure occurs have to be "undone":

- Undoing actions may involve doing something other than the inverse of the forward action:
  - chargeAcct == charge(€1) 🕑 refund(95c)
- Compensations may be overridden:
  - (createAcct; chargeAcct; makeOrder) 🕑 destroyAcct; verifyClient
- Compensations may be nested:
  - (…) / (destroyAcct 🕑 recreateBlacklistedAcct; …)
- Compensations may be scoped:
  - payment; { advertisment }; delivery
- Parallelizing parts of the process makes the compensation more involved

So Many Logics, So Little Time

- There are various flavours of compensations:
  - Process calculus style (CSP, pi-calculus based)
  - BPEL (and BPEL-like)
  - Petri net like
  - Language-based approaches
  - Deontic logics

Runtime Verification and LARVA

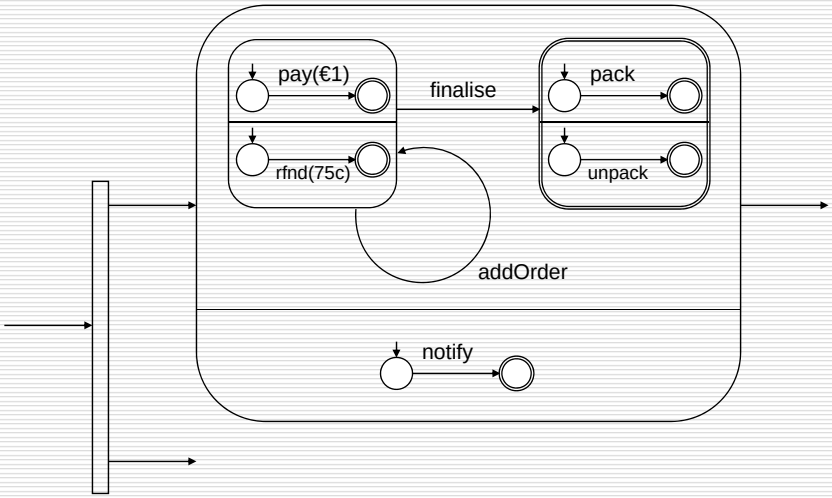



```
GLOBAL {
  VARIABLES { Clock t; int c = 0; }
  EVENTS {
    interact() = {*.action()}
    t30() = {t@30*60}
    ...
  }
  PROPERTY users {
    STATES {
      BAD { inactive badlogins }
      NORMAL { loggedin }
      STARTING { loggedout }
    }
    TRANSITIONS {
      loggedout -> loggedin  [goodlogin\\t.reset();]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



Compensations in LARVA

- Compensation automata extended to specify properties in LARVA.
- There are different modes of application of the compensations, which we are investigating, and will be discussing.
- As yet, there still is no proper real-time compensation management.

Using Compensations I:

Compensations as Specifications

- Complex contracts (properties) regularly have statements which should hold, but with provisos if they do not:
  - "A client is obliged to keep his/her account balance positive, and will be charged a fee of €10 if he/she does not"
- We expect the monitored system to implement such recoveries.
- Compensation logics and automata can be seen as a specification of what the system should do – it is simply a more appropriate notation to specify certain systems.

Using Compensations II:

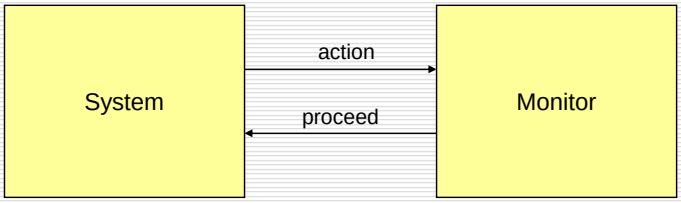
Compensations for Instrumentation

- On the other hand, the compensations can be seen as a specification of the recovery actions to recover from a particular problem.
- Essentially a *monitor-oriented programming* style very cleanly, avoiding keeping complex histories.
- This can be combined with the previous approach by providing two types of compensations:
  - exceptions handled by the system and;
  - exceptions to be executed by the monitor.

Using Compensations III:

Almost Online Runtime Verification

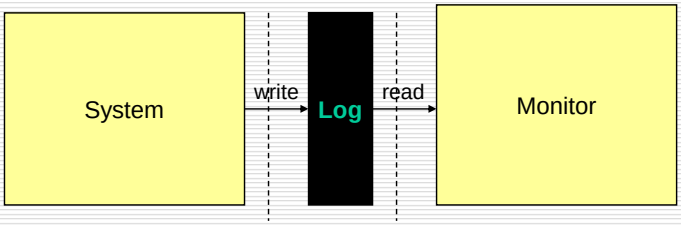
Online monitoring



- 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 *M* || *S*, not *S*.
- The overheads and interaction with the system are not always acceptable in an industrial setting.

Almost Online Runtime Verification

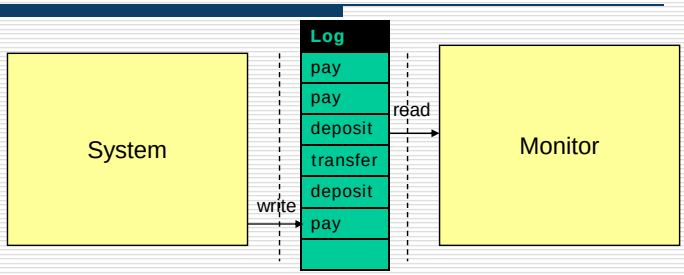
Offline monitoring



- 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.

Almost Online Runtime Verification

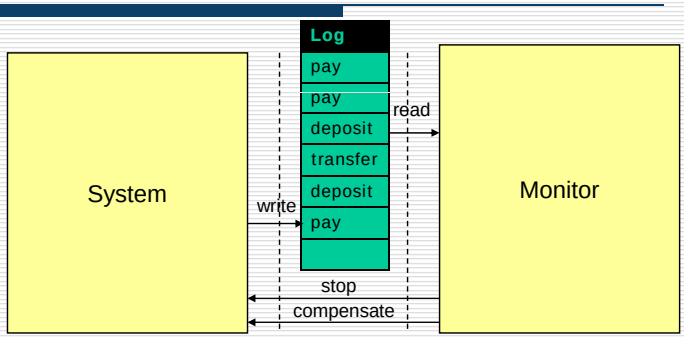
Almost online monitoring



- 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.

Almost Online Runtime Verification

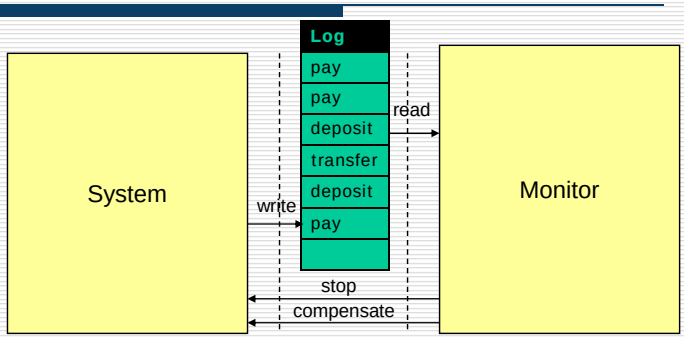
Almost online monitoring



- 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.

Almost Online Runtime Verification

Almost online monitoring



- In this example we would want to perform a compensation for:
  - transfer; deposit; pay

Almost Online Runtime Verification

- Compensate for actions:
  - up to the point which the system reached,
  - regressing back to a point before 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.

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 *almost-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.*

Conclusions and Future Directions

- The system is up and running at Ixaris – monitoring for properties and compensations but still working on the almost-online approach.
- Compensations may induce unbounded overhead.
- Investigating adding real-time.
- Relationship to deontic contracts – conflicts, analysis, etc.