Using Exception Handling for Fault-Tolerance in Mobile Coordination-Based Environments

Giovanna Di Marzo Serugendo *Alexander Romanovsky*

University of Geneva
Switzerland

University of Newcastle upon Tyne
UK

Content

Introduction:
Fault Tolerance by Exception Handling

- **Fault tolerance: error detection and error recovery**
- **Types of faults: process (agent) mistakes, environmental faults, mismatches, online upgrades/changes of the environment or in other mobile processes, application developers' mistakes, users' mistakes, malicious faults and all types of errors propagated from the underlying levels (OS, middleware, hardware) when they fail to deliver the required services**
- **This requires much more than software tolerance of hardware faults (ACID transactions, replications, atomic broadcasts, etc.)**

Introduction:
Fault Tolerance by Exception Handling

- We need software fault tolerance at the application level
- Forward error recovery (no rollback/backward error recovery)
- Exception handling as a means
- Separation of the normal and abnormal behaviour: separation of the code and of the flows of control

Introduction:
Exception Handling Techniques

- **Choice of exception handling techniques depends on many things such as the design paradigm, application types, computational models, types of faults, the environment, etc.**
- **The challenge is to develop novel exception handling techniques suitable for mobile systems based on the coordination paradigm**

- ***Definitely not Java or RMI Java exception handling***
- ***Definitely not conventional OO exception handling***

Motivations and Requirements: Specific Characteristics and their Effects

- **Processes are *mobile*. Very special exception handling techniques suitable for mobile systems: processes can leave the location and move to another location, the execution environment and the resources available can change on the fly**
- ***Asynchronous* communication. Decoupling producers and consumers, anonymous communication. Examples: event-based and data-driven systems, *a la* Linda (e.g. MARS, Lime, Lana)**
- **What if the process producing an erroneous data (event, tuple) moves? Or, if it becomes involved in other computations or completes its execution before an exception is signalled by the consumer of data?**

Motivations and Requirements: Specific Characteristics and their Effects

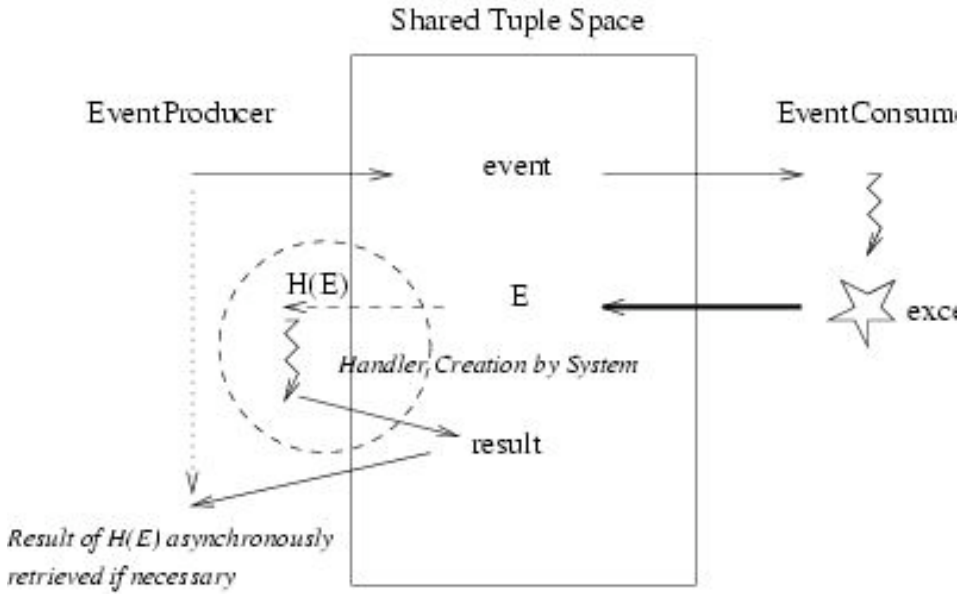
- **Exceptions cannot be treated as the normal events or tuples (Lana)**
  - **No separation of normal and abnormal behaviour**
  - **No guarantees that each exception is handled**
  - **No clear how to involve the producer cleanly**
  - **No special support for exception handling (error-prone)**
- **Similar problems with the standard notification mechanism**
  - **Its use is not compulsory**
  - **its use is intermixed with the normal code of the event producer (error prone)**
  - **…**

Motivations and Requirements: General Requirements

- **In spite of asynchronous communication *all exceptions have to be caught and handled.* Two possible solutions:**
  - **Chase the producer of the event causing the exception**
  - **Create a local handler process but only when an process signals an exception**
- **Choose dynamically whom to inform (flexibility)**
- **Inform several processes about exceptions (flexibility)**
- **Scopes (i.e. the exception handling contexts) should include all the processes to be involved in handling, for example, all processes (possibly) contaminated by the error:**
  - **Defined as all processes in a particular location**
  - **Dynamically defined by mutual agreements among a number of cooperating processes**
  - **Use knowledge-management to define it (cf M.Klein's work)**

Solutions

Specialised process H(E) is locally created when an exception E is signalled by the EventConsumer



EventProducer is not involved, handling is decoupled from it

H(E) always exists - handling is guaranteed

The handler process is created only when an exception is signalled outside EventConsumer

Solutions

H(E) is usually designed by the developer of EventProducer

Alternatively EventConsumer can provide a handler Hc(E), in which case it overrides the initial one

Each tuple T has a number of exceptions declared in its signature in addition to a set of parameters

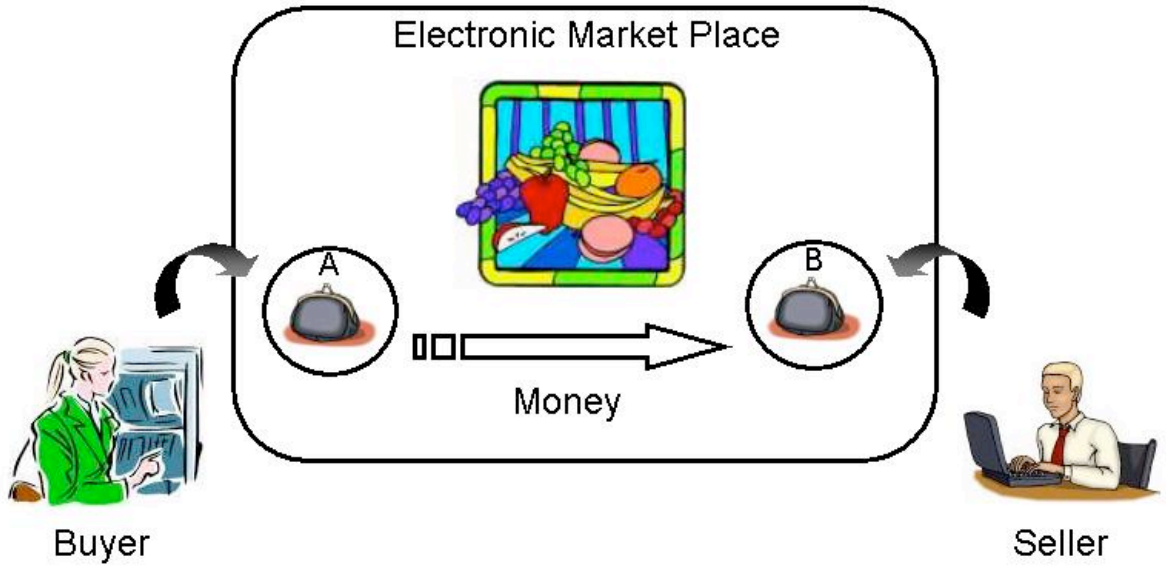
We are working on adding flexibility by:

- allowing several handlers
- allowing dynamic association of handlers with the tuple
- allowing any existing process to join handling
- allowing EventConsumer to move

Working Example

Market place. The buyer and the seller processes

The seller inserts the selling request into the local tuple space. The buyer finds it and proposes a contract, which can be accepted by the seller. After that the payment starts, it is executed by transferring money from the buyer-purse ot the seller e-purse



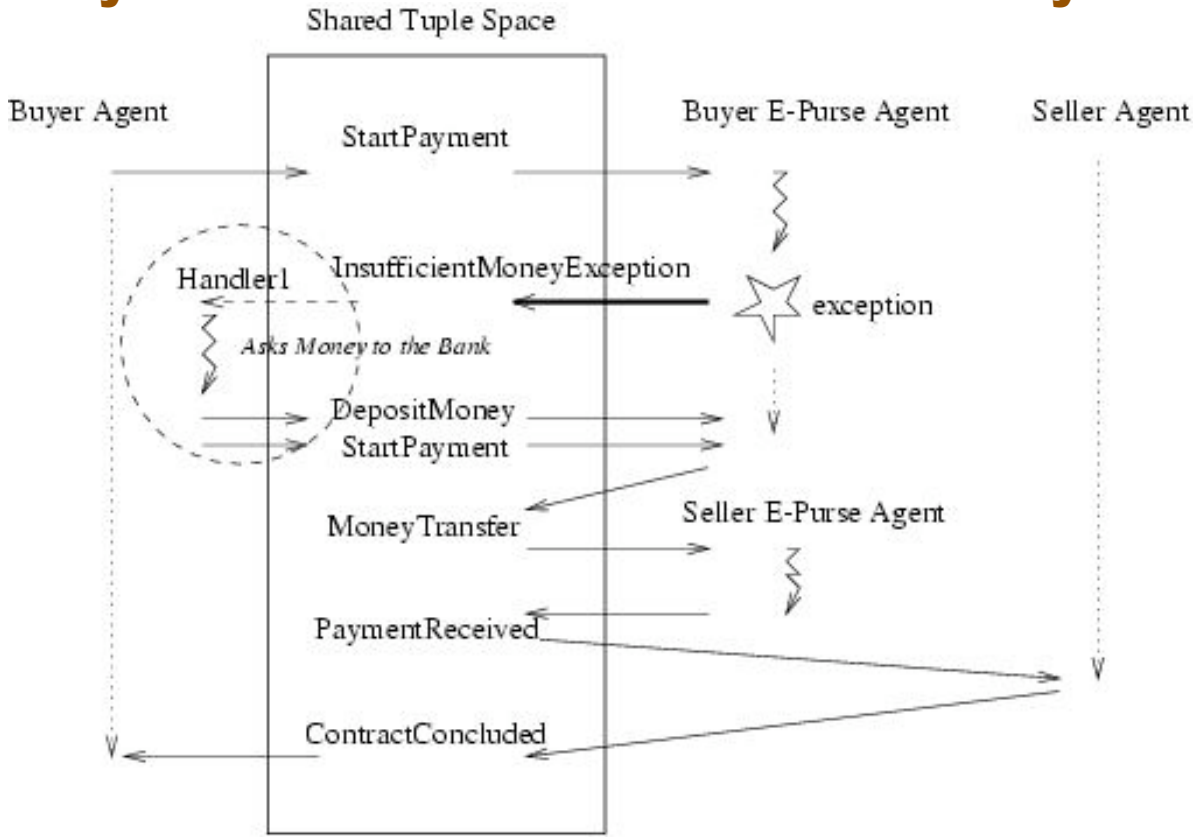
Working Example

The Buyer asks its E-Purse to release the money

The Buyer provides a name of the handler (Handler1)

The E-Purse signals an exception because there is not enough money

Handler1 accesses the buyer's bank account to transfer money to the e-purse



Working Example

In Lana:

- **the interacting processes have to wait until all notifications have been received (this essentially undermines the asynchrony which the model promotes)**
- **Processes can freely move without waiting for all notifications**
- **All exception handling code is intermixed with the normal behaviour (which error prone, not flexible - we can use different handlers for the same exceptions - e.g. in different locations)**

Future work

Implementation and experiments

Cooperative handling by several mobile processes

Exception handling context

Nesting of contexts