Formal Development of Cooperative Exception Handling for Mobile Agent Systems

L. Laibinis, E. Troubitsyna
A. Iliasov, A. Romanovsky

Åbo Akademi University, Turku, Finland
Newcastle University, Newcastle Upon Tyne, UK

Outline

- Motivation
- CAMA (Context Aware Mobile Agents) Framework
- Cooperative Recovery and Exception Handling
- Formal Development
- Conclusions

Mobile Agent Systems

- Mobile agents – decentralised and distributed entities, cooperating to achieve their individual goals
- Agents communicate asynchronously
- The characteristics of such systems:
  - have mobile elements (code, devices, data, services, users),
  - need to be *context-aware*,
  - are open (i.e., components can appear and disappear)

Need for Cooperative Recovery

- In most agent systems, agents recover from failures locally. Thus, any failure to recover can lead to agent termination
- In many situations cooperative and iterative recovery is beneficial for all involved agents
- However, an agent cannot be forced to participate in cooperative recovery
- The primary objective of an agent is recover itself, then to recover the environment, and, finally, to recover other agents

Our Approach

- Goal – formal development and analysis of cooperative recovery based on exception handling
- Our approach relies on combination of:
  - formal methods applied for rigorous development of the critical parts of the system, and
  - a set of design abstractions proposed specifically for the open context-aware applications (CAMA framework)
- Our formal framework – the B Method, supporting formal refinement-based development of systems
- A set of design abstractions is supported by special middleware

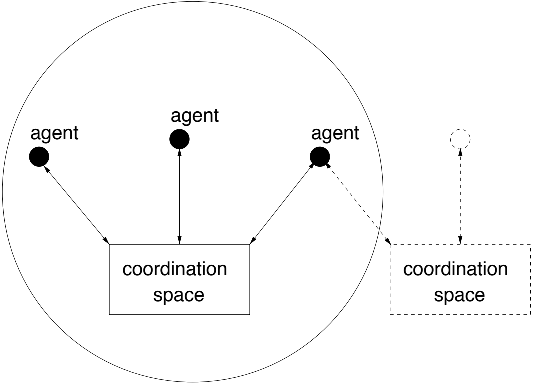
Outline

- Overview
- **CAMA Framework**
- Cooperative Recovery and Exception Handling
- Formal Development
- Conclusions

CAMA Framework

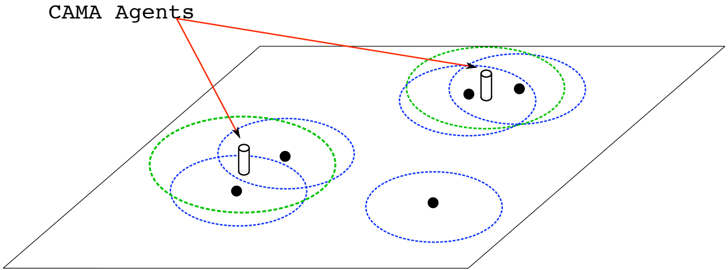
- CAMA – Context Aware Mobile Agents
- Framework for development and deployment of mobile agent applications
- Supports a set of abstractions for system structuring, openness and fault tolerance (exception handling)
- Methodology for formal design of open agent systems (based on the B Method)
- Verification by model checking CAMA process algebra
- CAMA middleware for mobile applications

CAMA Architecture

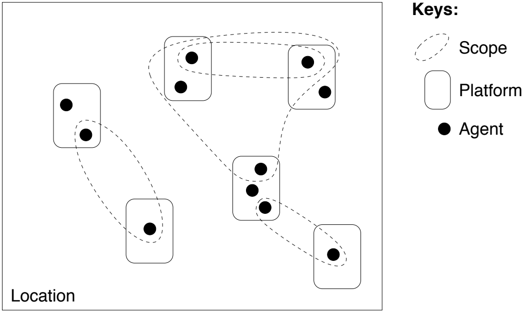


- Coordination via a number of independent coordination primitives (services)
- Agent communication is based on the Linda paradigm, providing a shared coordination space between the involved agents

Cama Agent System



CAMA Abstractions



Coordination	**Computation**
Location	Agent
Platform	Role
Scope	

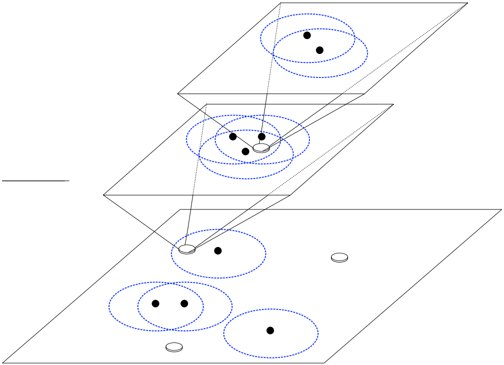
Abstractions - Location

- The core part of any CAMA system
- Acts as middleware
- Provides means for communication and coordination among agents
- Possible configuration: a server with wireless networking running CAMA daemon

Abstractions - Agent

- Basic structuring unit of the system
- Autonomous - decisions are made independently of other agents;
- Cooperative - achieves goals through communication with other agents;
- An agent is a piece of software that conforms to some formal *specification*;
- Each agent is associated with a *platform* (execution environment)
- Implements one or more *roles* (functionalities)

Abstractions - Scope



- Structures activity of agents in a specific location
- Provides isolation of several communicating agents, thus structuring the cooordination space
- Agents can cooperate only when they are participating in the same scope
- Supports error confinement and localised error recovery

Abstractions - Role

- Structuring unit of agent functionality
- Each agent has one or more roles associated with it
- Composition of agent roles forms its specification
- Scope definition determine the roles that can participate in it (ensuring compatibility and inter-operability of cooperating agents)
- A role is implemented as a number of reactions (events), which can be triggered by matching tuples in the coordination space

Outline

- Overview
- CAMA Framework
- **Cooperative Recovery and Exception Handling**
- Formal Development
- Conclusions

Mechanism of Cooperative Recovery

- Based on exception handling
- Cooperative recovery is attempted within a scope
- While an agent is unable to recover from a failure itself, it raises an external exception. This starts cooperative recovery involving all agents of the scope
- All the agents attempt to recover independently

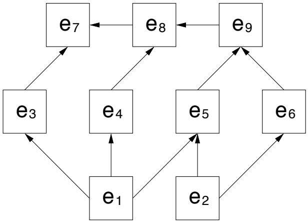
Mechanism of Cooperative Recovery (cont.)

- If recovery succesful (for all agents), the scope proceeds with normal activity. Otherwise, a new exception can be raised and broadcasted to the involved agents
- Participation in coordinated recovery is voluntary and the whole process is asynchronous
- Implemented by extending a role with additional reactions defining recovery actions

Exception Handling

- Several exceptions can happen at the same time, so exception resolution should be supported
- New exceptions can arrive when an agent is in recovery
- Exceptions can come in different order for different agents
- Exception resolution should be context-specific
- Agents can get disconnected or disappear during recovery

Termination of Coordinated Recovery



- We introduce a partial order (lattice) on the set of external exceptions. It is based on exception criticality
- The exception resolution function is based on the lattice structure
- A lattice has the top element, which guarantees termination of recovery process

Resolution of Concurrent Exceptions

- When an agent starts its recovery, it could happen that there are several exceptions waiting. We use the resolution function to map a set of exceptions into a single one.
- Moreover, resolution should be context-aware as context of an agent and its internal state can provide hints on choosing a recovery path
- Mathematically, a new exception is chosen from a set of common parents of the pending exceptions in the exception lattice

Outline

- Overview
- CAMA Framework
- Cooperative Recovery and Exception Handling
- **Formal Development**
- Conclusions

The B Method

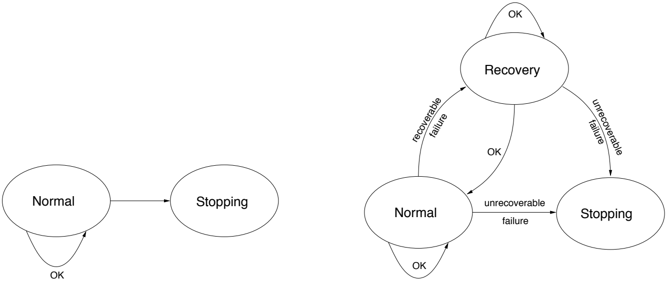
B supports top-down system development by correctness preserving steps – refinements. Each refinement step is validated by proofs. Refinements allows us to incorporate missing implementation details, at the same time preserving previously stated properties.

MACHINE *AM*
SETS *TYPES*
VARIABLES *v*
INVARIANT *I*
INITIALISATION *INIT*
EVENTS
$E_1 \;=\; \ldots$
$\ldots$
$E_N \;=\; \ldots$
END

Formal Development Process

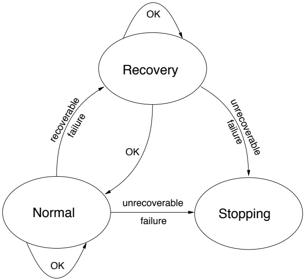
- Application development starts with an abstract specification of a scope
- It describes the required behaviour of multi-agent application
- In the refinement process, we incorporate implementation details concerning concrete functionality, communication, and fault tolerance

Initial Specification and the First Refinement



- Abstract model specifies global, high-level view of the system behaviour
- In the first refinement we introduce representation of various failure modes, i.e., distinguish between recoverable and unrecoverable errors

Initial Specification and the First Refinement (cont.)



- Upon detecting a recoverable error, the system enters the *Recovery* state
- Termination of iterative recovery is proved abstractly, by using decreasing variant expression

Further Refinement Steps

- System functionality and state is distributed over the set of active agents
- Concrete data structures modelling exceptions and their communication mechanisms are introduced
- Partial order between concrete exceptions is defined
- The exception resolution mechanism (based on the exception lattice) is introduced
- Termination is proved by associating abstract variant(s) with (inverse) criticality of the last generated exception

Conclusions

- We proposed a mechanism for cooperative recovery (based on exception handling) in multi-agent systems
- An agent has freedom to decide what role it should play in recovery process
- We formally verified a mechanism focusing on context-aware resolution of concurrent exceptions and termination of coordinated recovery
- The experience from using prototype tool for the CAMA system shows that the mechanism smoothly integrates with coordination paradigm and performs well in real applications

Thank You!

Questions?