Fault-Tolerant Mobile Agents in Mozart

Iliès Alouini and Peter Van Roy*

Introduction

In any wide-area distributed system such as the Internet, fault tolerance is crucial for real-world applications. We describe a new practical fault-tolerant mobile agent platform. The agent platform is built on the top of a global store abstraction [1, 2] that provides a globally coherent and fault tolerant memory. The global store abstraction is implemented as a user library that runs on the Mozart platform. The implementation does not require recompiling the platform. Mozart is a general-purpose development platform for open, robust distributed applications that is based on the Oz language [3]. The global store implementation is based on Mozart's reflective fault model and takes full advantage of the platform's network-transparent properties.

The global store abstraction

A *global store* consists of a set of objects replicated on several processes. A *user* is any computation that is part of an OS process running Mozart. Users can connect to or disconnect from a store dynamically. This adds or removes processes from the global store. A user invokes objects by initiating a transaction, which calls the objects. It is possible to use the store so that a user's object updates are seen instantaneously by that user without waiting for the network. This implies a possible speculative execution, which is completely managed by the store. Users invoke objects without worrying about concurrency control or store failure. Both concerns are managed by the store. Because Mozart is network transparent, users can communicate and collaborate by *sharing* a global store. The global store tolerates any number of user failures, as long as it exists on at least one process. The store can migrate without dependencies, i.e., the migration depends on no fixed process.

The store is lightweight and requires no persistence to recover from failure. The store uses process redundancy; with n processes it tolerates up to $n-1$ fail-stop process failures.

Mobile agents using the global store abstraction

One of the challenges for mobile agent platforms is to provide robust and fault-tolerant mobile agents [4, 5]. We build an agent API on top of the global store that provides fault tolerance, agent mobility without site dependencies, and permits home communication without any dependencies. In general, an agent can create any number of global stores. A first global store is used for the agent's own state, so that it can migrate. A second global store is shared between agents, for communication. The second global store is used to implement Send & Receive operations in a few lines of code. Our agents have the following properties:

- An agent can move from one site (i.e., OS process) to another, e.g., to reduce network latency.
- The agent's internal state is maintained when moving.
- Agents live in Mozart's shared computation space and can therefore communicate any data including compound structures, procedures, classes, etc.

*Université catholique de Louvain, Département d'Ingénierie Informatique, B-1348 Louvain-la-Neuve, Belgium. E-mail: {ila,pvr}@info.ucl.ac.be

- An agent can continue to function despite network inactivity (disconnected operation).
- Agents are not affected by process or host failures, if at least one process survives somewhere.
- Agents do not reference the file system when they move (except possibly initially, when creating a global store). When an agent moves from one site to another, there is no need to load agent classes from a file system, e.g., like in IBM Aglets. The agent classes are transferred in a transparent way through interprocess communication.

Mobile agent API

We define a *mobile agent* to be any distributed computation that has the ability to move from one site to another. An *agent* is a set of concurrent tasks where a *task* is a computation that uses the resources of a single process and can communicate or collaborate with other agents. Here is an API that allows to program an agent:

- Agent creation: `MA={New Agent.agent init(NewObj AgentStore)}`, with arguments: `Agent.agent` (input agent class), `NewObj` (returned procedure to create new objects in the agent store), and `AgentStore` (returned reference to agent store).
- Execute the task `F` of agent `MA`: `{MA run(F)}`, with argument `F` (a task). In Mozart, the task is defined as a *functor*, which is a first-class data structure defining a component specification. A functor defines the process-specific resources the task needs. The functor `F` can be created on the fly during task execution.
- Move the agent `MA` to host `IPadd` and execute `F` remotely: `{MA move(IPadd F)}`, with arguments `IPadd` (IP host address) and `F` (a task). The original task is not moved, but because the global store contains the agent state and depends on no fixed process, the result is a move whose strength is intermediate between weak and strong mobility.
- Move to home site: `{MA movehome()}`.
- Communicate with other agents: `{MA send(M)}` (send asynchronously message `M` to agent `MA`) and `{MA receive(M)}` (receive message `M` at agent `MA`).

Conclusion

This extended abstract explains the highlights of our simple mobile agent platform. We have shown how a fault-tolerant agent platform can be built naturally using a general-purpose platform for open distributed computing augmented with the global store abstraction. Our current research includes building other high-level abstractions for fault tolerance, secure distributed programming, and the implementation of agent-based applications.

References

[1] Iliès Alouini. *Global Store Module.* Available at `http://www.mozart-oz.org/mogul/info/alouini/globalstore.html`, April 2000.

[2] Iliès Alouini and Peter Van Roy. *A Practical Fault-tolerant Store Abstraction for Multiple Application Domains.* To be submitted, July 2000.

[3] Mozart Consortium. *The Mozart Programming System (Oz 3).* Available at `http://www.mozart-oz.org`, January 1999.

[4] Holger Pals, Stefan Petri, and Claus Grewe. *FANTOMAS: Fault Tolerance for Mobile Agents in Clusters.* IPDS 2000 Workshops, LNCS 1800, pages 1236-1247, 2000.

[5] Detlef Schoder and Torsten Eymann. *The Real Challenges of Mobile Agents.* Communications of the ACM (43) 6, pages 111-112, June 2000.