An Asynchronous Reflection Model for Object-Oriented Distributed Reactive Systems

© Jacques Malenfant, 2006

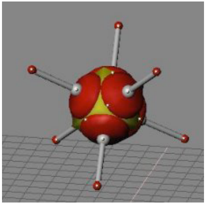
Laboratoire d'informatique de Paris 6 (LIP6)
Université Pierre et Marie Curie et CNRS (UMR 7606)
Jacques.Malenfant@lip6.fr

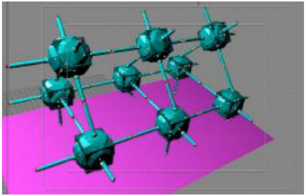
$Revision: 1.5 $ — $Date: 2006/03/25 09:18:14 $

1. Context & problem
2. Our MAAM software architecture
3. Extensions and future work
4. Conclusion

The MAAM Project

- Molecule := Atom | Atom + Molecule
- An atom : an autonomous module
  - 6 motorized legs
  - End connectors
- A molecule : modular robot
  - 1, 2 or 3D set of atoms
  - shape $\Leftrightarrow$ function
  - reconfiguring (changing shape) $\Rightarrow$ new capabilities, new task





Challenges for the software architecture

- Strategic level
  - Reconfiguring is needed to tackle different tasks (walking, climbing, transporting, ...).
  - Shows when the robot goes from one task to another.
- Tactical level
  - Connecting new atoms $\Rightarrow$ new capabilities, but also different ways for doing the same things.
  - New functions appear in the robot interface.
  - Old functions take a new implementation

Need to adapt the control software.
How? When? Efficiency?

General problem and proposed solution

- Applications in ambiant intelligence, pervasive systems, and the like impose a similar challenge to the software architecture.
- The high dynamicity of such applications as well as the variabilility in the available resources require run-time adaptation.
- Reflective systems have been studied for two decades to provide means to adapt software systems by giving the programmer introspection and intercession API.
- However, the traditional way reflective systems have been constructed does not scale to distributed and reactive systems.
- We propose $\mathcal{ARM}$, a new model that we applied to the MAAM project, but which also generalizes to the case of distributed reactive systems.

Hybrid deliberative/reactive architectures

- Reactive subsumption achieves good reflex but doesn't cope well with long term goals.
- Deliberative approaches use symbolic reasoning to control actuators but tend to be too heavy to sustain the real-time.
- The $\mathcal{ARM}$/MAAM achitecture adopts a hybrid deliberative/reactive approach, as many current robot control architectures.
- Achieves good performance by mixing short-term reactive capabilities and long-term deliberative ones.

How to implement this conceptual architecture?

- programming model mixing synchronous reactive objects and concurrent asynchronous objects.

The reactive framework

Implementing the subsumption reactive layer.

- Objectives:
  - Giving a framework (set of abstract classes) for a subsumption approach to program reactive modules in a high level programming language (Java).
  - Offering the runtime to execute the modules (perception, action).
  - Enabling the coordination among atoms in molecules.
- User perspective:
  - The user designs his schema, programs the modules and registers them.
  - Modules are programmed by inheriting from abstract framework classes and then defining:
    - putting conditions on signals
    - implementing the `handle` abstract method.

Implementation : synchronous programming

- Synchronous model: notion of cycle
  - at each cycle, the system evaluates the received signals and starts the corresponding activities.
  - interests: determinism, formal verification.
- Solutions:
  - Using REJO, a reactive extension of Java: rejected
  - Defining our own minimal synchronous runtime: adopted
    Idea: implementing a harmonious intergration of concurrent asynchronous objects (for deliberation, see later) and synchronous reactive objects for the control.

Minimal synchronous runtime

- Package ActiveObjects (active and reactive objects)
    - minimal support for executing active objects with asynchronous communication capability
    - standard implementation: asynchronous execution model (immediate reaction to events, messages).
    - extension: synchronous model (react to events at each cycle).
- Interest of the asynchronous communication
    - heterogeneous but transparent runtime: an asynchronous object can communicate with a synchronous one, and vice versa.

Overall GALS runtime, synchronization

- GALS: globally asynchronous, locally synchronous
  - keep a synchronous approach for the reactive part while mastering the inherent global asynchrony.
  - possible, thanks to the asynchronous communication.
- Synchronization in a GALS distributed system:
  - ensure the synchronization of synchnous objects without imparing their respectives execution constraints.
  - our solution: busy wait on future values, by looking for the value at each cycle.
  - advantages: transparent use of future variables between reactive synchronous objects and concurrent asynchronous objects.

Motivation: adaptation scenarios

- Changing task
  - reconfiguration of the robot, end of a task
  - changing part or all of the modules
- Fault-tolerance
  - detecting a fault on a sensor or motor
  - replacing impaired modules by others bypassing the faulty equipments.
- Optimizing parameters
  - enhancing the behavior of a module

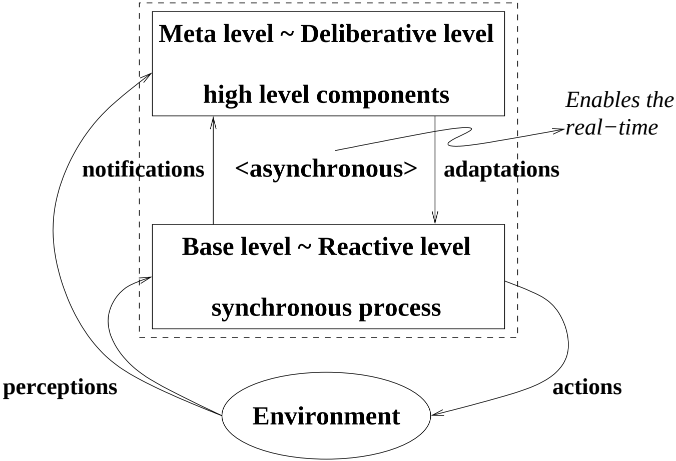
Adaptation framework

- Objective: ease the adaptation.
- Provide mechanisms for the dynamic adaptation of the system.
- Cope with different granularities:
  - fault-tolerance: granularity = reactive module.
  - changing task: granularity = set of modules = reactive “schema”.

Our reflective model: $\mathcal{ARM}$

- Reflection: capability for a program to know itself and to modify itself at runtime.
  - The base level does the “standard” processing of the application.
  - A metalevel does processing on the base level on order to adapt it to new execution conditions.
- $\mathcal{ARM}$ (Asynchronous Reflection Model)
  - An object kernel for distributed reactive systems.
  - Separate the execution of the base and meta levels using an asynchronous communication between the two levels.

Hybrid Reflective Architecture



Adaptation protocol

- Notifications from the base level to the meta level
  - at each cycle, data is emitted towards on the state of the atom (sensors, actuators)
- Processing of the notifications at the meta level
  - integration in the meta level model of the atom
  - delibaration on the current state
- Adaptation requests sent to the base level (if needed)
  - serialized method call to the base level, applied at the next cycle
  - before or after the reactive activity of the cycle?

Inside the $\mathcal{ARM}$ model

The $\mathcal{ARM}$ model is:

- independent of the precise base level objet model (unit of concurrency, communication, ...)
- parameterized by the kind of meta level representation of the base level
- integrates a control theory (or decision theory) approach to decide when and how to adapt the base level.

Middleware level

- In the MAAM project, adaptation of the base level is currently limited to the exchange of reactive modules.
- In general, both the base level application and the underlying middleware may need to be adapted.
- Much work has been done on reflective middleware, including our own LIP6 PolyOrb reconfigurable ORB.
- For ambiant intelligence and pervasive systems, but also for less constrained robotic systems, such a middleware layer should be deployed on individual nodes.
- A communication between the meta level and such middleware has to be established to guide the latter in applying the best policies to the current application.

Virtual machine level

- Many systems today uses a virtual machine abstraction at the base level.
- As a specific middleware, the virtual machine can also be the target for adaptation at run-time.
- Work has been done to render virtual machines reflective and easily reconfigurable, including our LIP6 Virtual Virtual Machine (VVM).
- A strong connection between the VVM and the meta level is also sought to control the adaptation of the virtual machine from an application perspective.

Model-based reasoning

- As software systems grow in complexity, reasoning about their behavior and its adaptation can cope with all the details.
- Reasoning at a model level, thus hiding unnecessary details, appears as a natural evolution to engineer such systems.
- Work has been done to provide run-time tools to manipulate models, namely UML models, such as our LIP6 ModFact system.
- Connecting the meta level with model-based reasoning tools promised to seamlessly integrate run-time adaptations and software updating done in the development and maintenance processes.

Conclusion I

On $\mathcal{ARM}$/MAAM:

- Objectives reached:
  - minimal runtime of the GALS type
  - high level reactive control framework
  - integration in a reflective platform
  - identification of run-time adaptation scenarios
- Experiments:
  - experiments on a typical complet schema
  - at the time of these experiments, the atom programming model was too premature to enable “real” experiments on atoms

Conclusion II

On $\mathcal{ARM}$ in general:

- Coupling with underlying middleware layers (system, communication, virtual machine, model-based reasoning tools).
- Coordination among meta levels of different entities in a distributed application: problem of decentralized adaptation decisions.
- Implementing a well-defined distributed adaptation protocol: domain-specific language with locking and transaction capabilities.