

A generic modular architecture for the control of an autonomous spacecraft

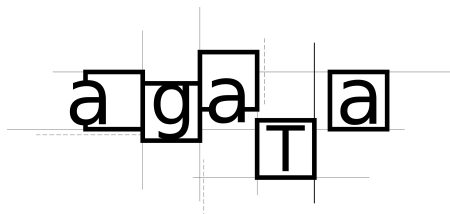
Gérard Verfaillie & Marie-Claire Charmeau
ONERA & CNES, Toulouse, France



The AGATA project

CNES-ONERA-LAAS **AGATA** project :
Autonomy Generic Architecture: Tests and Applications.

Objective : to build a **ground demonstrator** of an **autonomous spacecraft**,
in order to explore the **technical challenges**
and to convince project teams that **autonomy** is **possible** and **useful**.



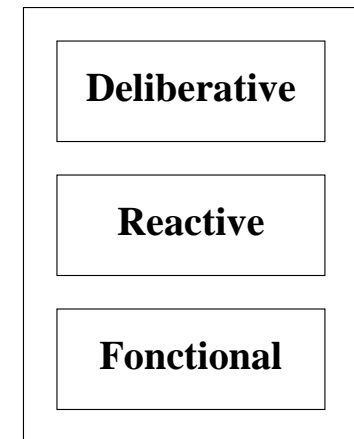
First objective

To define an **autonomous control architecture**.

Existing architectures

The most **usual** in the robotics and space domain: based on **three levels**:

1. **deliberative**: mission tracking, generation and maintenance of activity plans;
2. **reactive**: execution triggering and tracking;
3. **functional**: encapsulation of all the basic system functionalities.



Main identified **drawbacks**:

1. no explicit place for **situation tracking** (system and environment state tracking);
2. potential lack of **reactivity**: no action outside **planning**, which is the **core** of the control system;
3. lack of **modularity**, except for the functional level.

Expected qualities of an autonomous control architecture

1. **genericity**;
2. adequacy for **closed-loop control**:
 - (a) state tracking;
 - (b) objective tracking;
 - (c) decision making;
 - (d) decision execution;
 - (e) control supervision;
3. possible guarantees in terms of control **quality** and **reactivity**;
4. control **modularity**;
5. control and data **encapsulation**.

Main features of the proposed architecture (1)

Inspired from **GENOM** [LAAS-CNRS] and **IDEA** [NASA-Ames].

1. **decomposition** of the control into **control modules**;
2. **hierarchical organization** of the modules;
3. **encapsulation** of **control** and **data** inside each module;
4. **communications** between modules via :
 - (a) system **control requests**;
 - (b) **information** about **request** and **system states**;

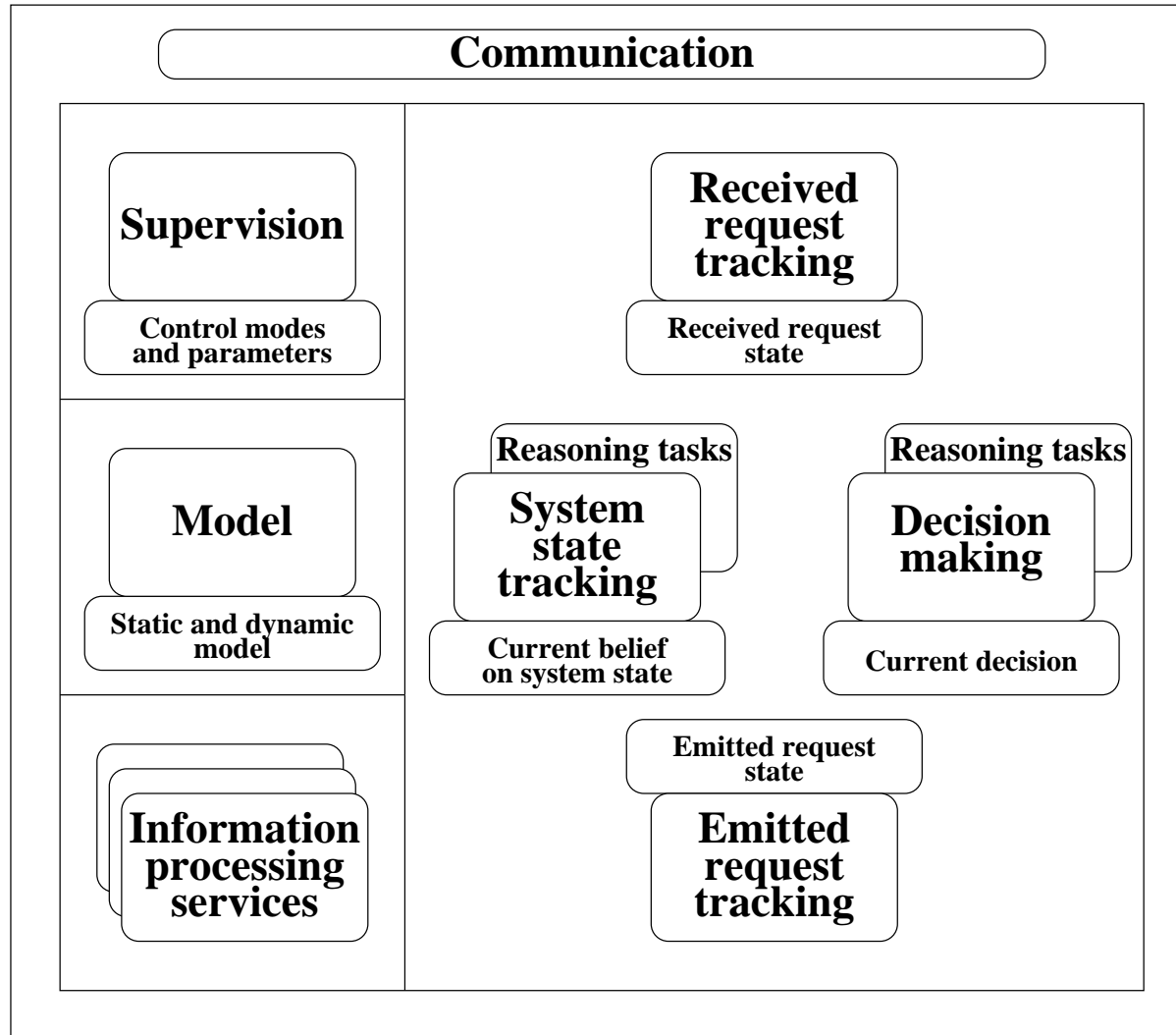
Main features of the proposed architecture (2)

5. **organization common** to all modules:
 - (a) **system control** part:
 - i. system state tracking;
 - ii. received request tracking;
 - iii. decision making;
 - iv. emitted request tracking;
 - (b) **model part**: static and dynamic models;
 - (c) **information processing** part;
 - (d) **module supervision** part;
 - (e) standard **communication** part.

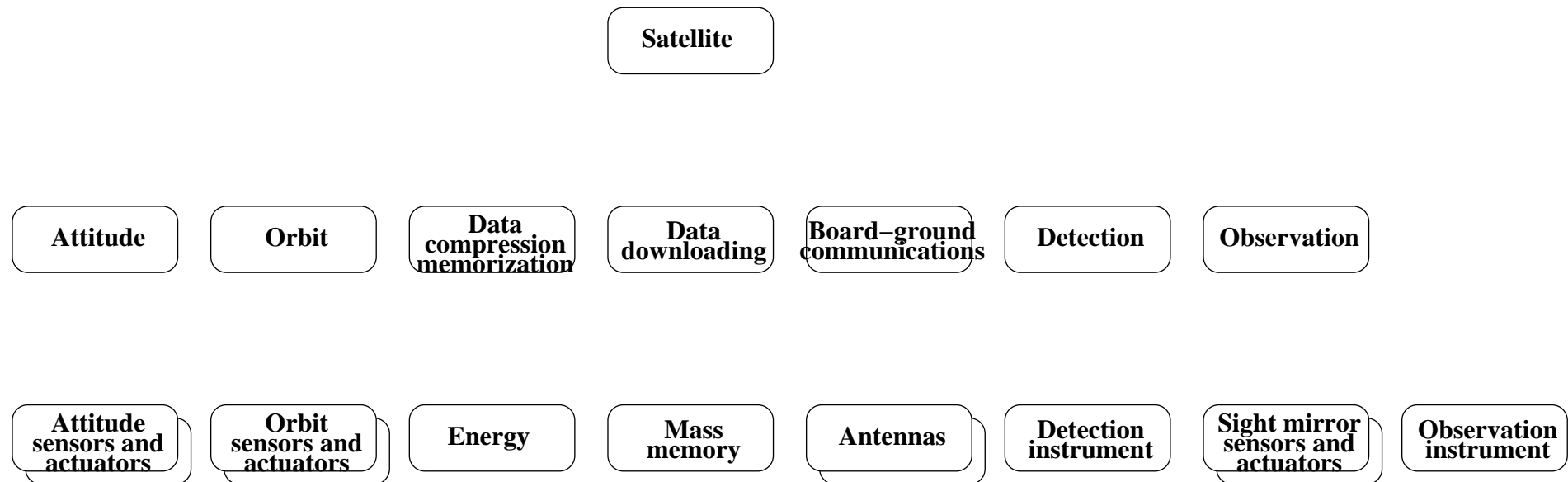
Close to works from **Brian Williams & al.** [MIT].

Schema of a generic control module

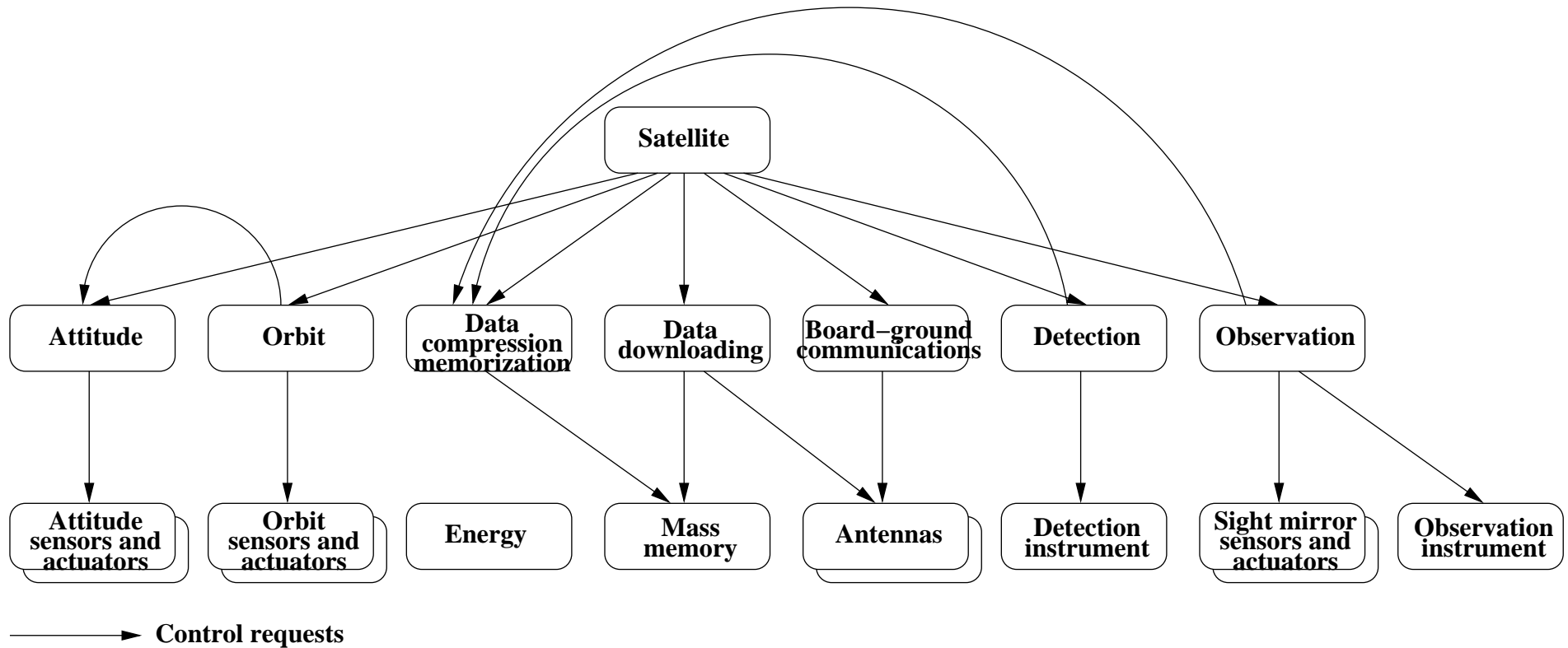
Control module



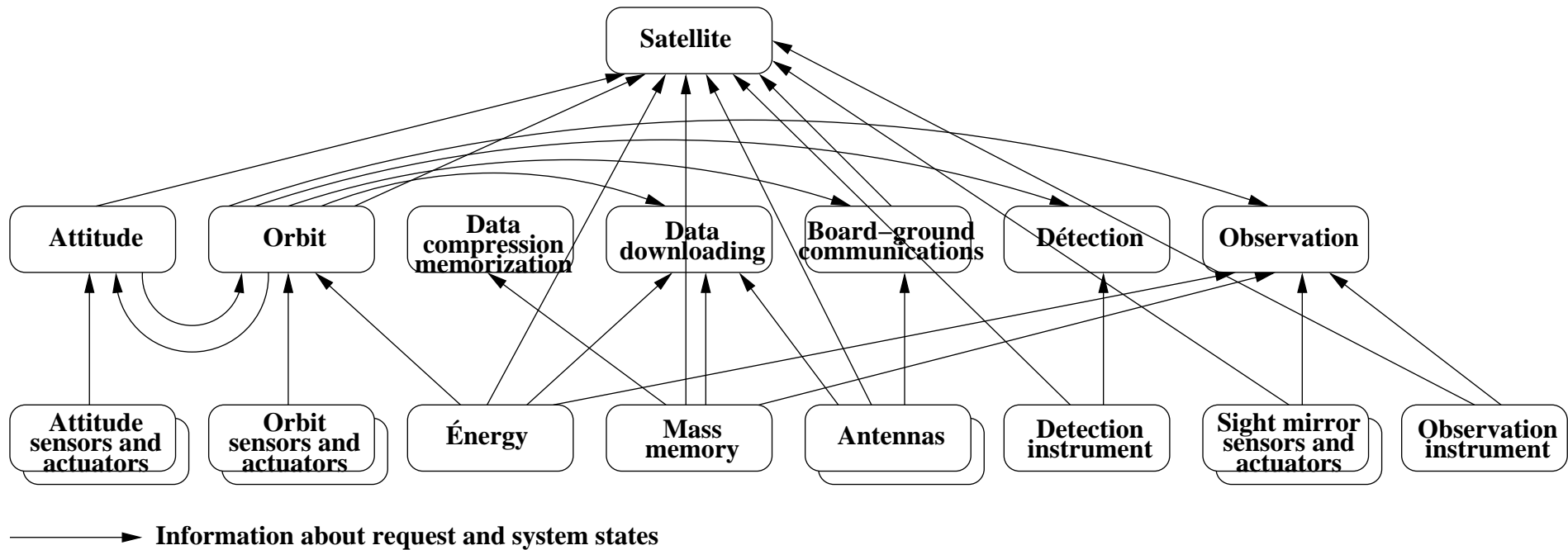
Example of decomposition of the control of an Earth surveillance and observation satellite



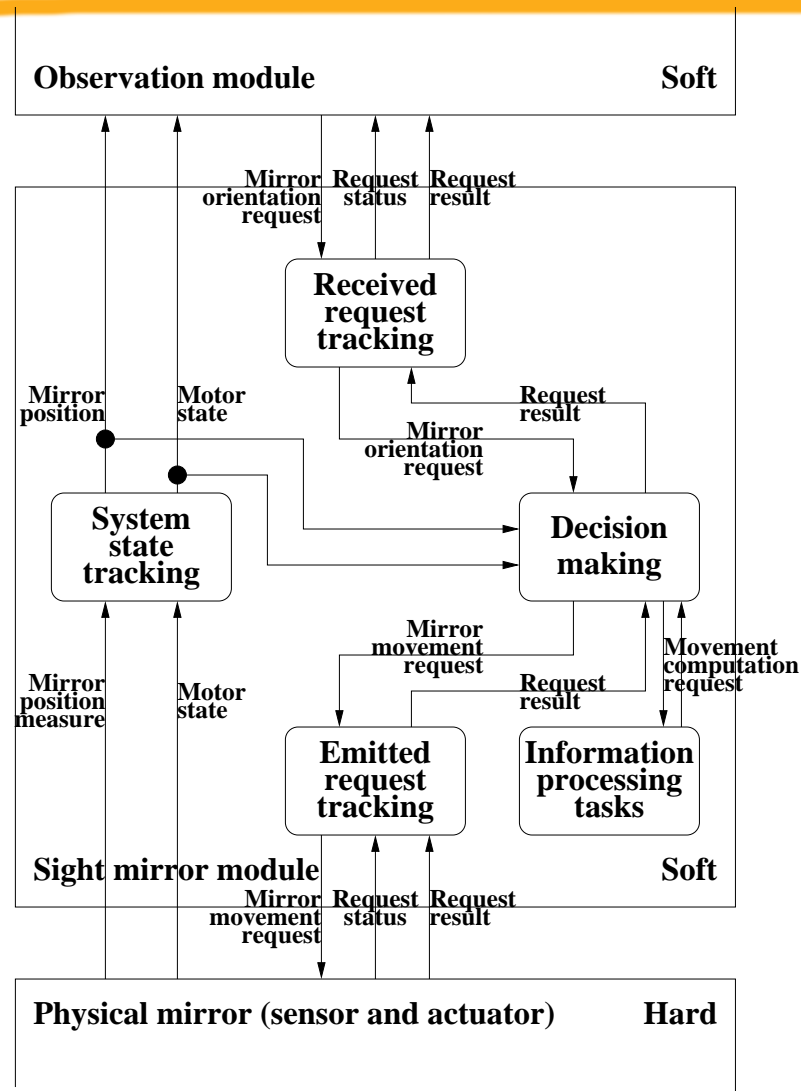
Exchanges of control requests between modules



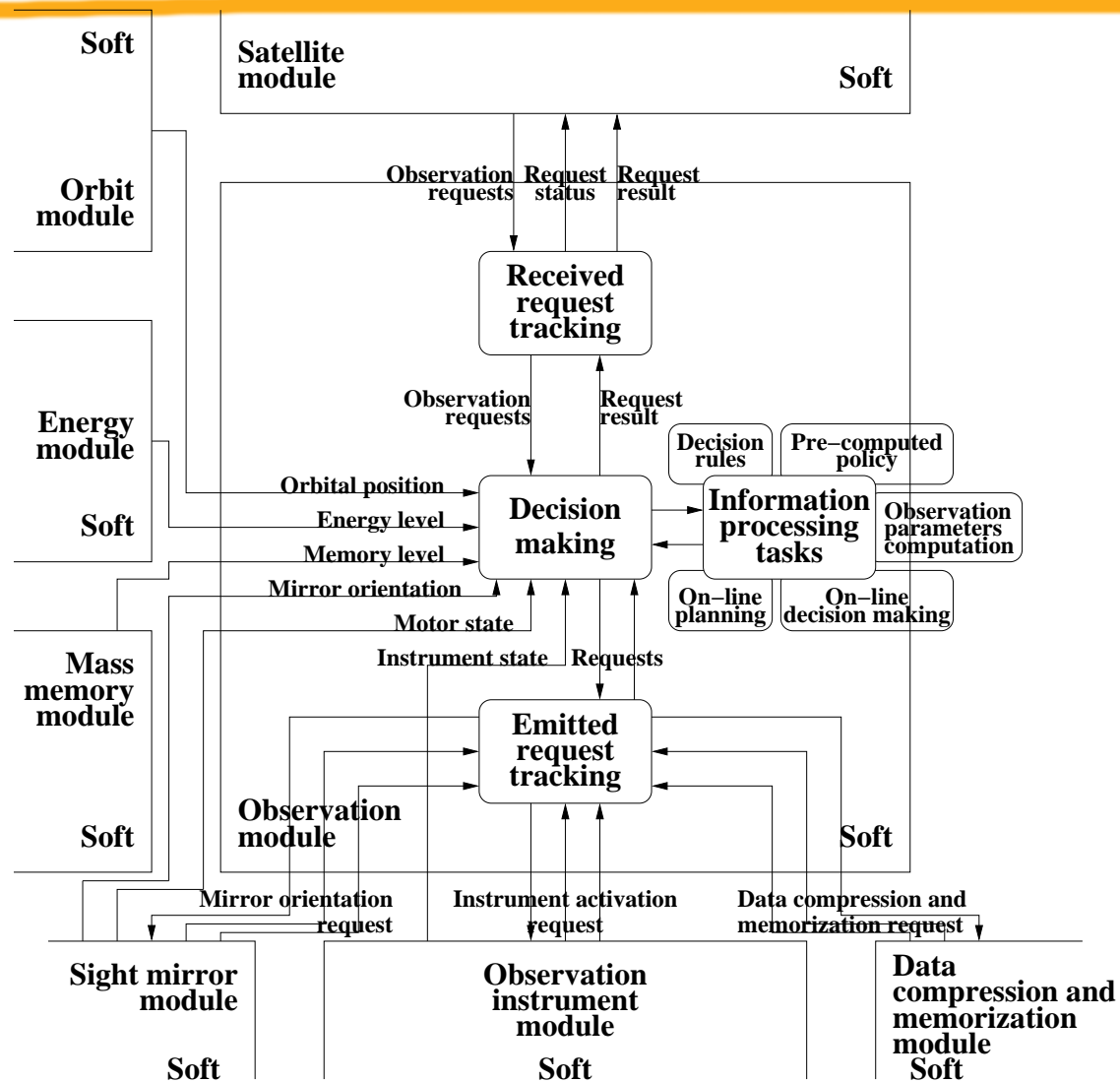
Exchanges of information about request and system states between modules



Example of module at the lowest functional level: the sight mirror control module (sensor + actuator)



Example of module at a higher level: the observation control module



Dynamic behavior of each module (1)

Each component inside a module (system state tracking, received request tracking, decision making, emitted request tracking ...) is **reactive** *i.e.*, able to react **at the rhythm of its environment** (requests, information ...).

It can be defined using **synchronous languages**, like for example **Esterel**.

Some of them may need the support of **deliberative** tasks (a **planning** task for the decision making component, a **diagnosis** task for the system state tracking component ...).

Dynamic behavior of each module (2)

When a **reactive** component needs the support of a **deliberative** task, it must be able at any time:

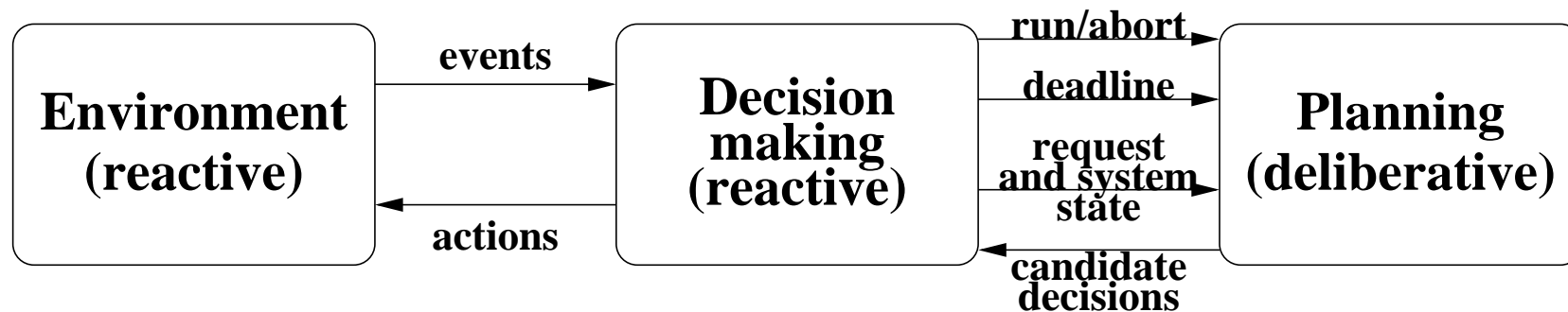
1. to **activate** the deliberative task;
2. to give it a **deadline** for an answer;
3. to **check** an answer against the current state of the system and of the requests;
4. to produce itself a **default answer** when the deadline occurs in case of **no answer** or **incorrect answer**.

Dynamic behavior of each module (3)

In the other direction, the **deliberative** task must produce an (as good as possible) answer by the deadline, for example by using an **anytime** algorithm able to produce better and better solutions as computing time increases.

Example for a **planning** task: to reason on **longer and longer horizons** or on **more and more precise data**, in order to produce an (as good as possible) decision when time is up to decide, for example when the satellite enters a ground area visibility window in case of an observation decision.

Interaction between a reactive decision making component and a deliberative planning task



Controversial message to the IWPSS community

In the setting of the autonomous control of a spacecraft in a **dynamic** and **uncertain environment** (physical system and environment state, user requests . . .), with **reactivity requirements** and **limited computing resources**, the core of the control system is not **planning**, but **decision making**.

Planning is only a **support** for decision making. By anticipating the possible future, it can help decision making to make a good decision at a given time.

But decision making must be possible without planning, for example by using **default decision rules**.

Anyway, what is important is the **decision**, not the **plan**.

Conflicts between modules (1)

A **decision making** component inside each module
→ **possible conflicts** between decisions,
for example for the access to common resources (energy, memory ...).

Two main ways of managing such conflicts:

1. at a **low level**, by the modules that manage these resources
→ high reactivity, weak anticipation ability;
2. at a **high level**, by a module M higher in the hierarchy than the potentially conflicting modules M_1 and M_2
→ lower reactivity, stronger anticipation ability.

Conflicts between modules (2)

A **system state tracking** component inside each module
→ **possible conflicts** between system state estimations,
for example by two sensors of the same feature (orbital position, attitude ...).

Need for a module M higher in the hierarchy than the potentially conflicting modules M_1 and M_2 , able to **merge local beliefs** produced by M_1 and M_2 and to **produce a global belief** about the state of this feature F .

→ Any other module in the architecture that would need to know the current state of F (in fact, the current belief about the state of F) must ask M for that, and not M_1 or M_2 .

What about FDIR?

FDIR = **Fault Detection, Identification, and Recovery.**

Detection and identification → **System state tracking.**

Recovery → **Decision making.**

→ **FDIR inside each module**, with the same rules to manage conflicts in terms of system state tracking and decision making (see above).

→ No strict frontier between **normal** and **anormal behavior.**