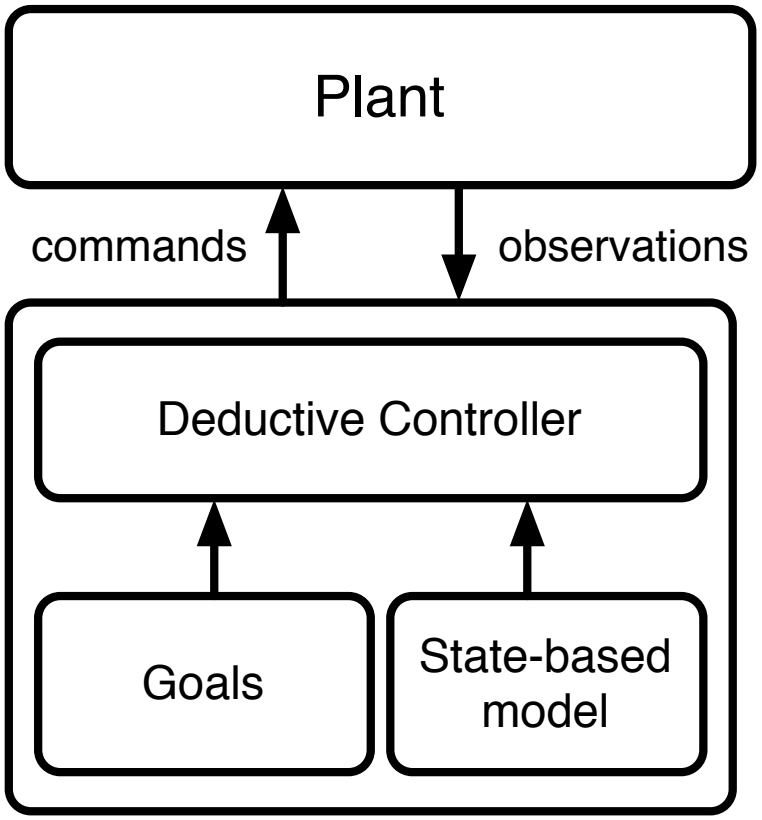
Part-Whole Hierarchical Modularization of Fault-Tolerant and Goal-Based Autonomic Systems

Luca Pazzi, Marco Pradelli
University of Modena and Reggio Emilia
Department of Information Engineering DII-UNIMORE
Via Vignolese 905, I-41125 Modena, Italy
{luca.pazzi,marco.pradelli}@unimore.it

Recent trends in space software engineering

- Model-based control;
- Goal based (GB) control;
  - Specification from high-level objectives:
    - Appealing in space missions, allows on-board systems to operate in absence of human operators;
    - Improve the overall robustness of on-board systems;
    - Implements varying levels of on-board autonomy, including Fault Detection, Isolation and Recovery (FDIR) capabilities;
  - The idea is to take autonomous decisions on the activities to be performed depending on both *unplanned events* and *overall mission goals*.

Monolithic architecture



- Goals allow to operate a system at the level of explicit intent;
- A state-based representation of the various plant devices is needed in order to accomplish the high-level goals;
- The “deductive controller” explores the state space and chooses the activities to be performed depending on both unplanned events and overall mission goals.

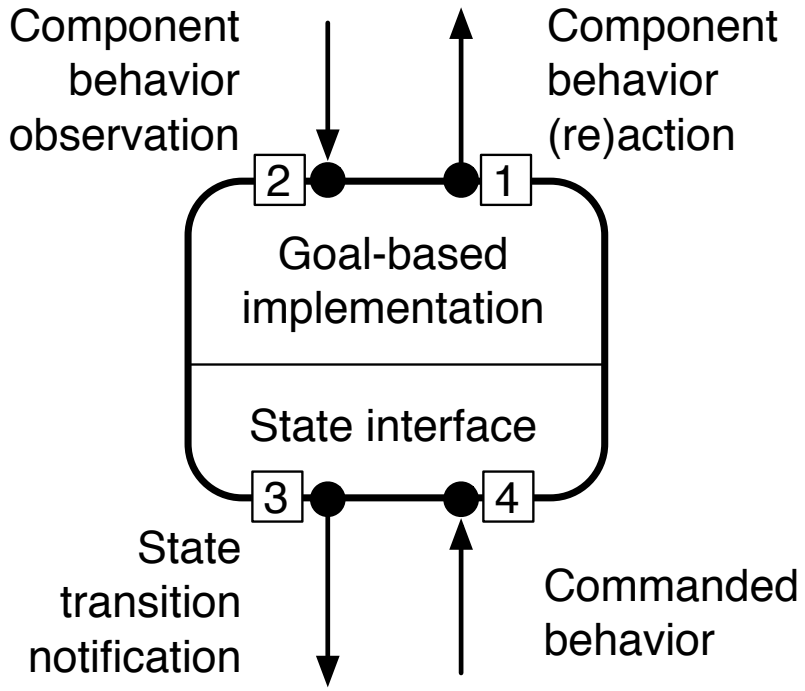
Open issues

- Are goals coherent one with each other? Can they formally be validated against system behavior?
- Can low-level sequences be calculated at design time?
- Can we plan “unplanned events”?
- How can the system architecture be decomposed in order to build independent and reusable subsystems?
  - Although both state-based and goal-based paradigm are decomposable, it is not clear how to have the two notions agree;
  - Can FDIR strategies be modularized?
  - Finally, how to decompose the main closed loop control?

Basic Approach

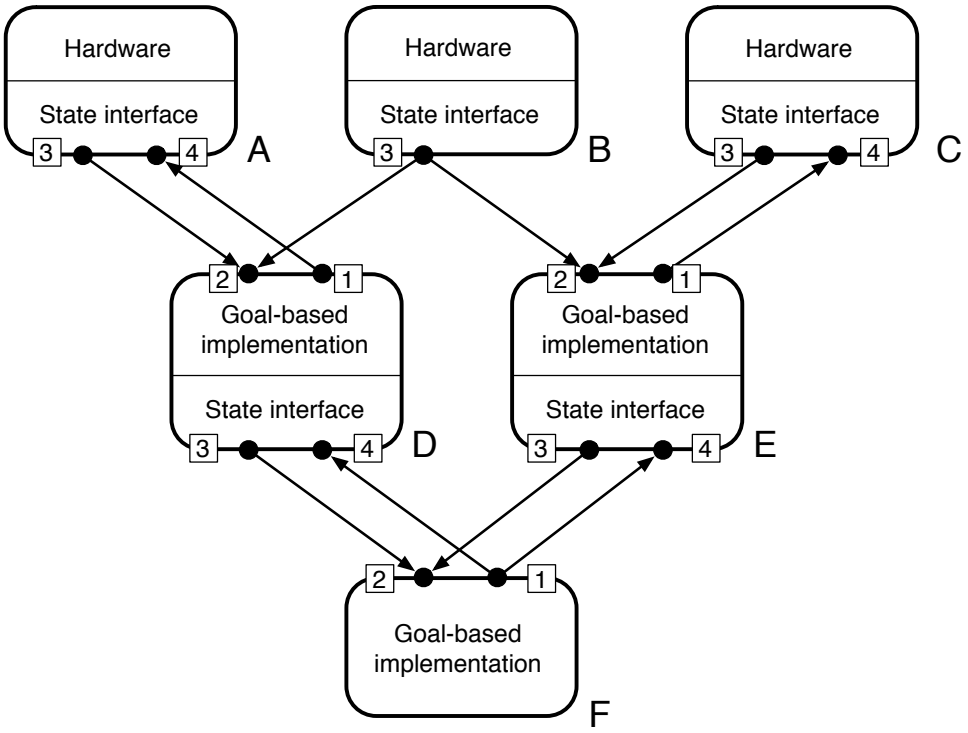
- We express goals as state constraints:
  - Goals can be seen indeed as “system state configurations” which have to reached by the system through sequences of “low level system interactions”;
  - It becomes therefore possible to calculate such sequences at design time instead that by “on the fly” computation which are potentially energy and time consuming;
- We employ a modular state-based paradigm: Part Whole Statecharts
  - allow to specify both a implementation and interface by a state based formalism;
  - allow an effective modular state-based *system decomposition*: a PWS system is assembled by composing, recursively, already specified PWS subsystems.
  - Each state module reacts to events coming both from the system to which it belongs and propagates further events to its subsystems.

Basic modular construct



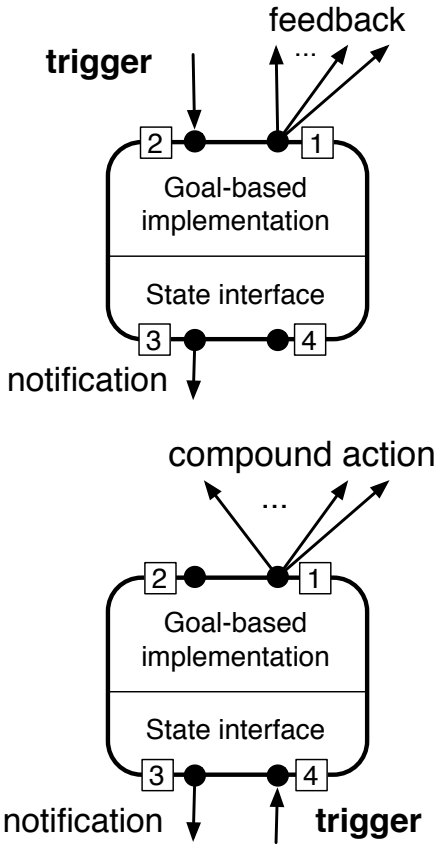
- Each module implements a specific system control according to a goal-based implementation;
- Each module has four ports:
  - the two ports on the upper side allow the module to coordinate subsystem behavior;
  - the two ports on the lower side allow the module to be coordinated on its turn by other system of which it is part.

Modular part-whole composition



- Leaf modules are nothing but hardware drivers;
- Intermediate modules describe subsystem control;
- Root modules describe global control.

Basic control patterns



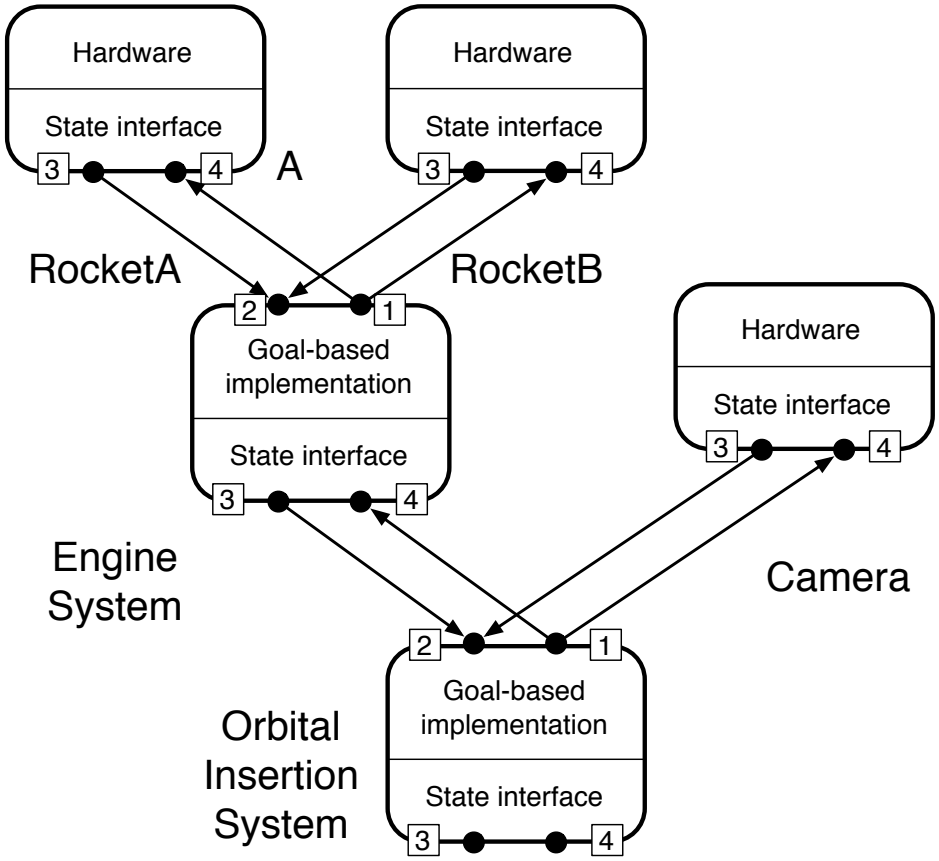
- A triggering event comes to the module from one of its subsystems:
  - feedback is directed back to the module subsystems;

- A triggering event comes to the module from one of the systems which have the module as subsystem:
  - a compound action is forwarded to the module subsystems;

Example: orbital insertion system

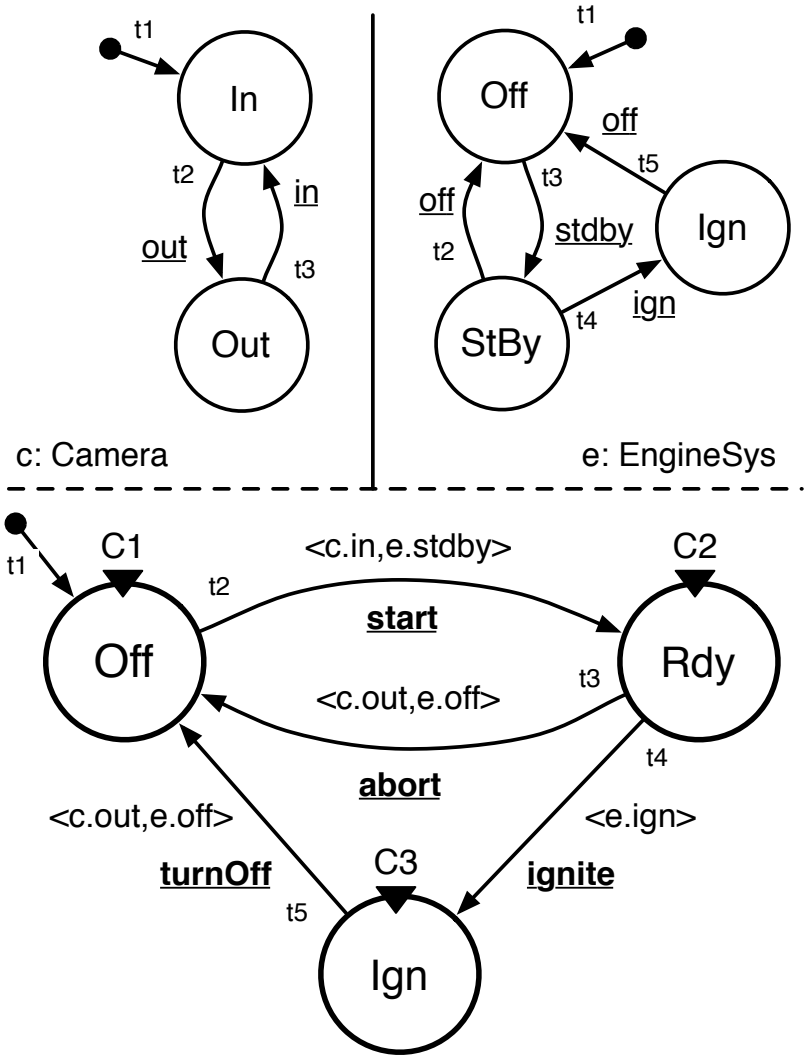
- the orbital insertion of a spacecraft, requires to thrust a rocket engine after having retracted a science camera in order to avoid plume contamination;
- the rocket engine consists of a main engine and of a backup engine, in order to improve reliability;

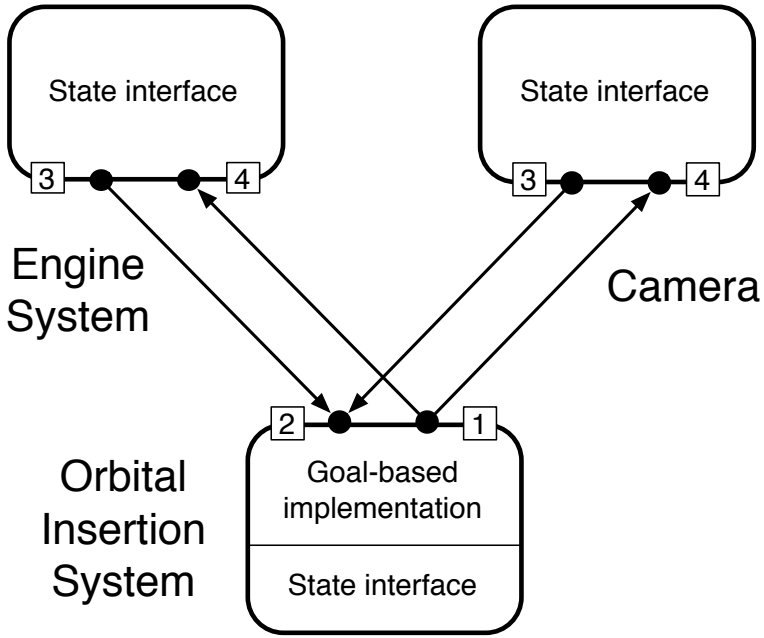
Example: orbital insertion system



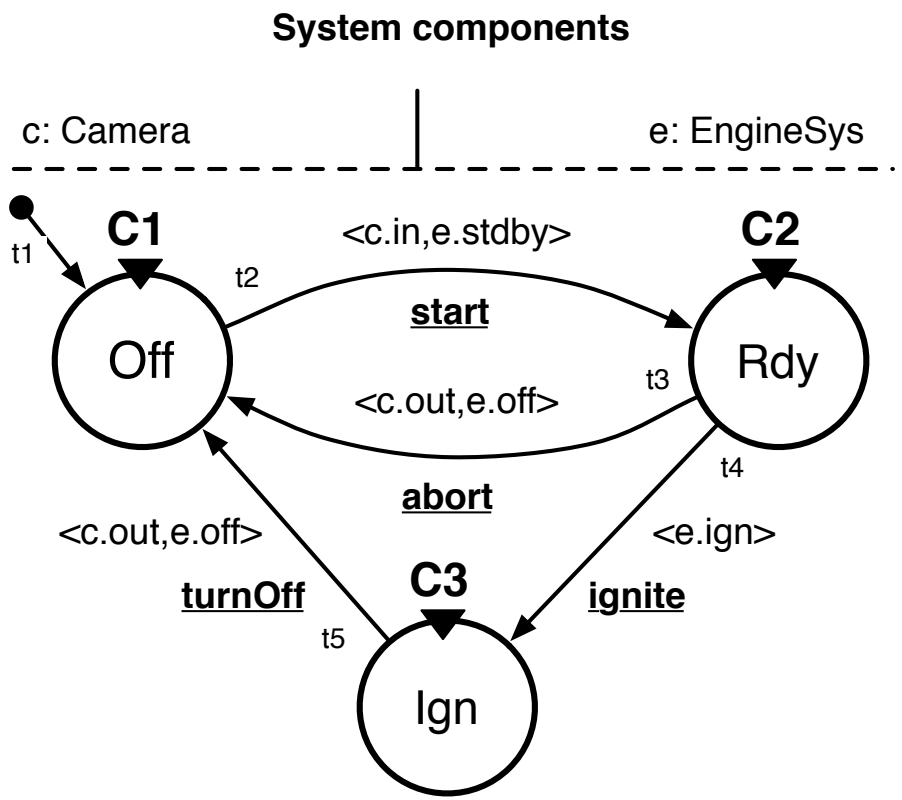
- Different control loops may be established, at the same time, among modules at each level
- For example commands may be exchanged among the orbital insertion system and the engine system, which in turn propagates further commands towards the rocket engines at the top.
- Notification of success or failure travel in the same way downwards, from the rocket engine to the engine system, which may send back upward commands to the rocket engines and, at the same time, downward, failure or success notification to the orbital insertion system.

Example: orbital insertion system





Implementation by PW Statecharts



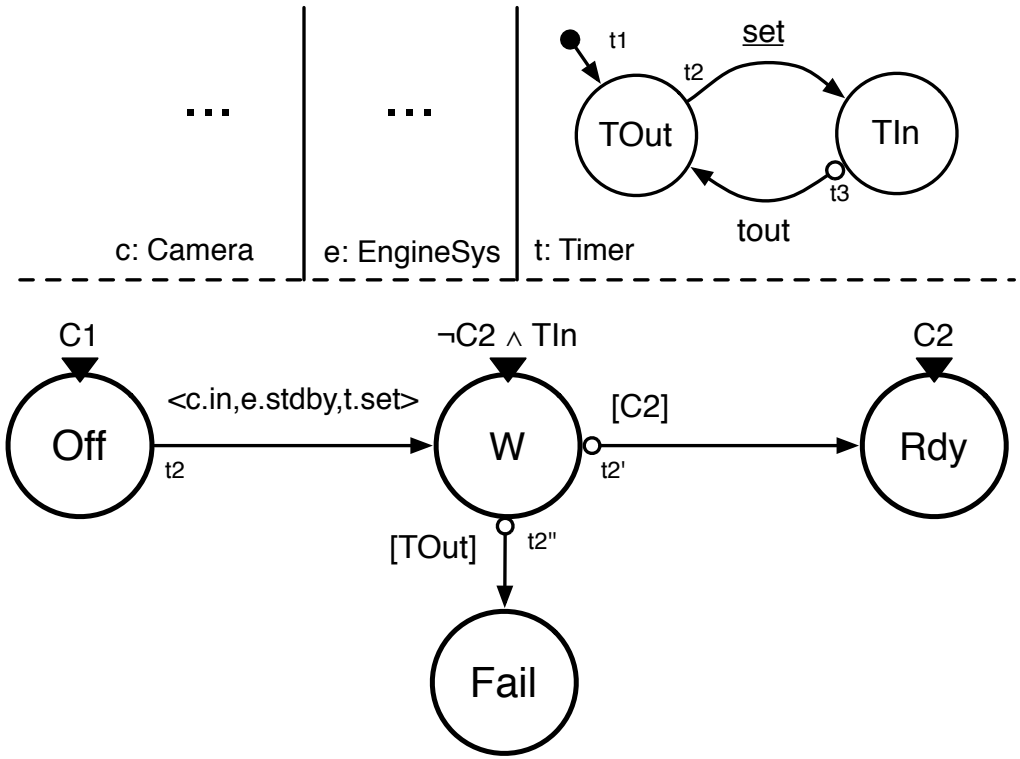
- System behavior implemented through extended state machines;
  - state transitions labelled by:
    - triggering events:
      - **start**, **abort**, **turnOff**, **ignite;**
    - lists of commands directed towards subsystems;
      - <c.in, e.stdby>
  - states constrained by state propositions expressing system goals:
    - **C1**: the engine is turned off and the camera arm is extended;
    - **C2**: the engine is ready to fire and the camera arm is retracted;
    - **C3**: the engine is thrusting and the camera arm is retracted;

Basic Fault Detection Mechanism

- A **fault** is the impossibility to achieve a goal by a controller module, in a given time, due to the unexpected behavior of some of its components;
- For each action sent, there are two possible failure outcomes:
  - the commanded component either do not move to the desired state in a “reasonable” given time or
  - it fails silently, which again means that it takes an indefinitely long time to complete.

  On the other hand, since both the components and the controller operate and communicate asynchronously, it becomes necessary to achieve, at the programming level, some form of nonblocking synchronization amongst them.

Wait States



- Special intermediate wait states are therefore introduced,
- Let C2 be the original goal proposition of the ending state of transition t2;
- An intermediate derived proposition C2' can be associated to state W, meaning
  - goal C2 has not been achieved (yet) *and* less that T time units have been elapsed, where T is the maximum expected time required to accomplish both camera retraction and engine standby.
- By adding a special "Timer" synchronous state machine, like the one depicted in the picture, it becomes possible to translate timing issues into state machine behavior, thus having to deal with a unique paradigm.

Example: self-healing rocket engine