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

Luca Pazzi * **Marco Pradelli** **

* *University of Modena and Reggio Emilia, Department of Engineering Sciences DII-UNIMORE, Via Vignolese 905, I-41100 Modena, Italy (e-mail: luca.pazzi@unimore.it)*
** *University of Modena and Reggio Emilia, Department of Engineering Sciences DII-UNIMORE, Via Vignolese 905, I-41100 Modena, Italy (e-mail: marco.pradelli@unimore.it)*

Abstract: The paper examines current trends in autonomic space software systems and proposes the adoption of a hierarchical state based formalism which allows the different paradigms employed in the field to meet seamlessly. It is not clear, for example, how goal-based on-board autonomy, originally conceived for deductive-flavored systems which may not be ground controlled for indefinite portions of time, be able to combine with model based engineering, which best fits current industrial design strategies. Other aspects, such as closed loop discrete control and fault tolerance do not easily lend themselves to modularity. The paper shows that, by employing the proposed formalism, goals can be decomposed and distributed in a very natural way among different modules. Each module is, at the same time, both a controller and a controllable part of the whole system, allowing to partition the closed loop control flow at different levels of complexity. Formal verification is also possible by employing goals as state based constraint in the implementation phase.

Keywords: Device degradation, dependable systems, Part-Whole Statecharts, goal-based approach, reusable subsystems, constraint-based control, model-based.

1. INTRODUCTION

Recent trends in space software engineering see a growing interest towards model-based software and, at the same time, towards methods which allow to specify the software starting from high-level objectives. The idea is to operate a system at the level of explicit intent [Chien et al. (2000)] and, at the same time, to have a model-based representation of low-level operational details needed in order to accomplish the high-level goals. Such a twofold autonomic approach, often referred to as goal-based (GB) control, seems particularly appealing in space missions, in order to improve the capabilities and robustness of on-board systems, simplifying their operations and allowing them to operate in absence of human operators. In place of traditional ground originated command sequencing, the spacecraft on-board control system should therefore take autonomously decisions on the activities to be performed depending on both unplanned events and overall mission goals [Truszkowski et al. (2006)], thus implementing varying levels of on-board autonomy, including Fault Detection, Isolation and Recovery (FDIR) capabilities.

By the GB approach, a goal is typically a "system state configuration" [Bennett et al. (2008)] and low-level control strategies have to be built "on the fly", i.e. dynamically, to track system state, diagnose faults, and perform reconfigurations according to the declarative model of the plant, through propositional inference [Williams et al. (2003)]. In other words, system *programming* is well distinguished from system *execution*, the former stating high level objectives, the latter "trying to transition the system toward a global state that satisfies the configuration goals" according to a model of the plant [Truszkowski et al. (2006)]. Figure 1 (a) depicts a simplified representation of the corresponding *control loop*.

Modularization concerns arise from the above drafted approach. It is not clear in fact how to build independent, reusable subsystems by which the monolithic schema of Figure 1 (a) can be decomposed. Although it is clear that the different constituents of the approach lend themselves, individually, to modular decomposition, the different nature of the employed paradigms, mainly goal/deductive and state-based, together with the closed-loop nature of control, do not seem to integrate well. For example, goals can be decomposed in a natural way, since their accomplishment can be subdivided, hierarchically, into a network of subgoals. Although such a goal decomposition has been clearly identified, it is not clear how to distribute it coherently within a modular decomposition. Goals are in fact also subject to strict timing and duration, as well as to mutual dependencies among them; activities performed in order to realize them, whether programmed explicitly or specified by a constraint-based control language like RMPL [Williams et al. (2003)], necessarily have to be consistent with subsystems and hence reside within related modules. Further difficulties in achieving modular-

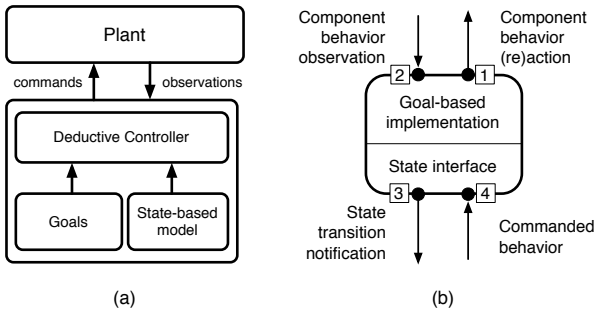

Fig. 1. Canonical goal based/deductive architecture and its control loop (a), Part-Whole Statecharts modular architecture (b).

ity requirements are given by the incorporation of FDIR techniques into hardware and software. Maintainable and extendable FDIR designs are indeed not available for reuse by different space projects designers mainly since they require tight hardware and software integration and since a coherent model of reactive behavioral decomposition poses still unresolved conceptual difficulties.

2. ARCHITECTURAL PROPOSAL

Part-Whole Statecharts (PWSs) [Pazzi (1997), Pazzi (2000)] are employed in the paper in order to develop autonomous, safety-critical systems in a way that allows an effective modular state-based system decomposition. A PWS system is assembled by composing, recursively, already specified PWS subsystems.

The basic composition mechanism is centered around an extended state diagram hosted within the PWS module, called *whole*, whose details are described in Pazzi (2008), which abides to specific state configuration constraints and accomplishes a twofold purpose: on one hand it coordinates the behavior of a set of PWSs, called *components*, by sending them appropriate logical signals, on the other hand it provides a state interface which is able to receive, on its turn, appropriate logical signals from other PWSs.

A PWS is therefore both a controller and a controlled entity: the two roles coexist seamlessy and can be visualized by considering an *internal* implementation as opposed to an *external* interface. The overall software architecture is built up by having the implementation part of one PWS connected to the interface of other PWSs through four kinds of *logical ports*, depicted in Figure 1 (b). The different system components operate asynchronously in a nonblocking discrete event communication environment, such as a field bus. The four kinds of logical ports shown above can be implemented through mutex or read-write blocks of memory shared among the different asynchronous execution and communication tasks.

The behavior of a PWS consists of state changes occurring within the modules' state machines induced by the mutual exchange of commands and notification signals exchanged through the ports. In the portion of the PWS which implements the behavior, logical signals named commands are issued towards component PWSs through port number 1 in order to request them to take state transitions according to the interface. The implementation portion

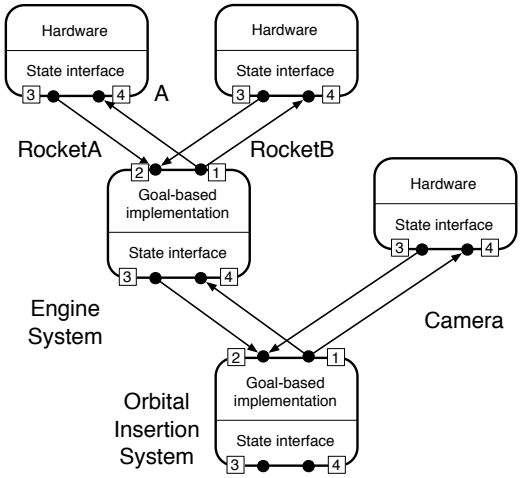

Fig. 2. Hierarchical part-whole arrangement of modules which realize the orbital insertion system.

is made aware of the current state and behavior of the components by receiving notification signals through port number 2. In a complementary way, the portion which acts as an interface sends notification signals to other PWSs trough port number 3 and receives commands through port number 4.

2.1 Example

A typical part-whole arrangement of communicating PWS modules is shown in Figure 2, where the orbital insertion of a spacecraft, taken from the example in Williams et al. (2003), requires to thrust a rocket engine after having retracted a science camera in order to avoid plume contamination.

Three special PWS modules (Camera, RocketA and RocketB) do not control further modules on their own. Such modules define only the interface portion of the PWS, since they may be thought of as being either some sort of "hardware driver" – as in case of the camera driver module – or to provide a state machine interface to non decomposable subsystems – such as time-driven ensembles. Rocket engines are for example constituted by further components which require tight timing integration and therefore are seen as an integrated, monolithic, time-driven block. The orbital insertion system module on the bottom of the hierarchy makes finally available its interface portion in order to be further composed in more complex control and coordination system modules.

2.2 Behavioral Patterns

As observed, one of the difficulties in modularizing monolithic closed loop control consists in decomposing the mechanism by which control and feedback are exchanged among the plant and the controlling system. It can be observed that two basic behavioral patterns are feasible from event flow across interconnected modules:

(1) Figure 3 (a) shows the basic *reactive* pattern, where a signal coming from one of the component PWSs triggers a transition in the internal state machine. Such a state transition in turn emits commands

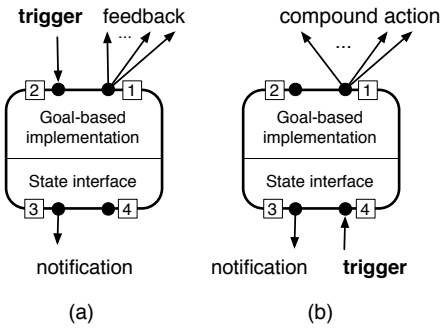

Fig. 3. Basic behavioral reactive (a) and propagation (b) pattern.

towards the components and emits a notification towards the PWS which has the current PWS as component;

(2) Figure 3 (b) shows instead the basic *propagation* pattern, where a signal coming from one of the PWSs which have the current PWS as component triggers a transition in the internal state machine. Such a state transition in turn emits commands towards the components and emits a notification back to the PWS which has the current PW as component.

The overall control flow may be therefore seen as distributed along multiple levels. For example, in Figure 2, logical signals may travel upward and downward the hierarchy amongst modules. 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 logical commands towards the rocket engines at the top. Notification of success or failure travel at the same time 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.

2.3 Goal-based implementation

Another difficulty in achieving full modularity consists in the appropriate distribution of goals amongst modules. Part-Whole Statecharts can be shown to lend themselves towards a natural interpretation of goals as state constraints which seamlessly integrate with state based implementation. Each goal to be achieved by the system can be in fact translated to a specific state configuration of one of its modules.

By considering the lowest PWS module in the hierarchy of Figure 2, which implements the coordinated operation of the camera and the engine system, moving the system from a configuration in which the camera is extended and the engine is stopped to one in which the camera has been retracted and the engine thrusts. An intermediate configuration must be traversed, in which the camera is already retracted and the engines are ready to thrust.

Figure 4 shows the Part-Whole Statechart describing the implementation of the orbital insertion system. The lower half of the PWS diagram depicts the implementation details of the system module, while the upper half of

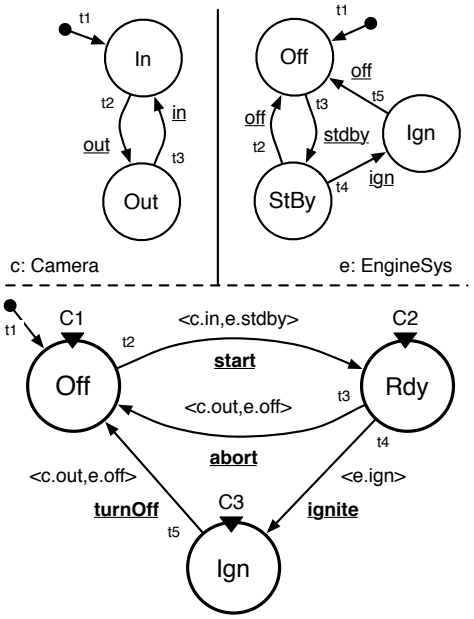

Fig. 4. The state machine describing the implementation of the orbital insertion system through the interfaces of its two immediate upper level components.

the diagram depicts the state machine interfaces of the component modules. The control is achieved through three states, namely Off, Rdy and Ign, each corresponding to a specific configuration of the states of the component systems. Three main goals associated to the orbital insertion module can be envisaged, namely:

(1) C_1: the engine is turned off and the camera arm is extended;
(2) C_2: the engine is ready to fire and the camera arm is retracted;
(3) C_3: the engine is thrusting and the camera arm is retracted;

Each goal is assigned to a state in the PWS state diagram, as shown graphically by a black triangle pointing to the state. State transitions may be triggered by actions coming from other PWSs and are labelled by actions to be directed towards the components of the PWS. The main rationale of the PWS approach is that, as the controller moves among its constituent states, components change their current state according to the commands associated within the state transitions in such a way that the corresponding state constraints are satisfied, in which case the PWS is said to correctly specified. A patent pending method for building correctly specified PW Statecharts is reported in Pazzi (2008).

2.4 Basic Fault Detection Mechanism

A fault can be defined as *the impossibility to achieve a goal by a controller module, in a given time, due to the unexpected behavior of some of its components.*

Consider a system trying to reach a goal C through a transition which commands actions to the system components. For example, by transition t_2 in Figure 4 the ignition system is required to retract the camera and to put the

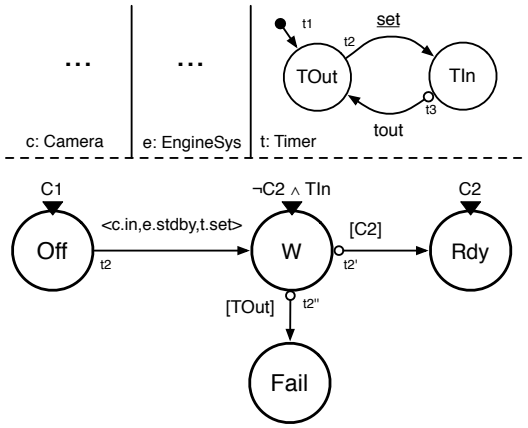

Fig. 5. Basic synchronization and failure detection schema through an additional wait state W and a Timer component.

engine system in standby mode, thus accomplishing the goal for such a transition.

For each action sent, there are two possible failure outcomes: the commanded component either do not move to the desired state in a 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. The regular behavior consists therefore, by negation, in letting some limited time pass in order for the components to complete the requested operations, before moving the controller to the final state of the transition.

Special intermediate wait states are therefore introduced, like the state W of Figure 5. Let C be the original goal proposition of the ending state of transition t_2. An intermediate derived proposition C' can be associated to state W, meaning "goal C 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 same picture, it becomes possible to translate timing issues into state machine behavior, thus having to deal with a unique paradigm. A timer is indeed a state machine which can be moved, synchronously, to a *time in* state by a *set* command issued to it as part of the action list associated to the transition. When a specific time interval has passed (for simplicity it is supposed that a specific timer is defined for a stated fixed time interval), the timer returns to its initial *time out* state.

2.5 Implementing Compound FDIR

The nature of the part-whole paradigm allows to have FDIR strategies distributed and partitioned along the hierarchy by modular, self-contained component FDIR strategies. Component off-the-shelf FDIR strategies may be assembled, as part of the overall component behavior, in order to achieve higher-level FDIR strategies.

Two feasible approaches in dealing with recovery strategies are envisaged:

(1) Recovery strategies may be attempted autonomously by the system and, in case such attempts fail additionally, the system reaches an explicit fail state;
(2) Recovery strategies are made available by the system starting from the explicit fail state; such strategies must then be commanded as part of the external driven behavior.

An example of the former approach is presented by applying the fault detection mechanism shown in Figure 5 to the engine system of Figure 4. The resulting self-healing FDIR behavior is hidden and totally self-contained within the implementation.

Figure 6 shows how the transition t_4 from the stand-by (StdBy) to the ignited (Ign) state of the engine system may be extended by inserting intermediate fault detection states, in such a way that the transition either ends explicitly in a success or in a fail state. The idea is to attempt an autonomous fault recovery after the first rocket ignition attempt fails. The system starts in the StdBy state satisfying the state constraint proposition "engine A is in state standby *and* engine B is in state standby and the timer is in state timeout". A conjunction of basic state propositions can be written as a triple: in this way, the initial configuration is given by (StdBy,StdBy,Tout). After the ignition command is sent to the main rocket, the system moves to the wait state W1 and waits until either it successfully ignites or a timeout signal is received from the timer. In the former case, the system moves to the Ign final state, in the latter it moves to the fail state FailA, meaning that the first rocket failed. The fail recovery strategy then starts automatically by the transition $t_4^{(3)}$ which send an ignition command to the backup rocket and moves to wait state W2. The system again waits until either success or failure are signalled by the component or by the timer In the former case it moves to the final states Ign or FailB. It may be observed that state Ign is, coherently with incoming transitions, labeled by a constraint which states exactly the goal for the self-healing system: in such a way a goal becomes part of a state based design, allowing it to be reached as part of the module execution.

3. CONCLUSIONS

The paper showed, mainly through examples, how layered system modularization by Part-Whole Statecharts achieve a tight integration of the different modeling paradigms employed for fault tolerant autonomic systems. The main point in employing the PWS state-based paradigm consists in the fact that each module is, at the same time, both a controller system and a system under control, thus allowing a multi-level decomposed control loop. Each PWS module is moreover implemented through state constraints which, on their turn, represent the goals to be achieved by the autonomic behavior. The paper showed finally how information hiding features within PW Statecharts allow to hide fault detection and recovery strategies within the modules, thus allowing them to be further assembled into more complex recovery strategies as the modules are assembled into more complex systems.

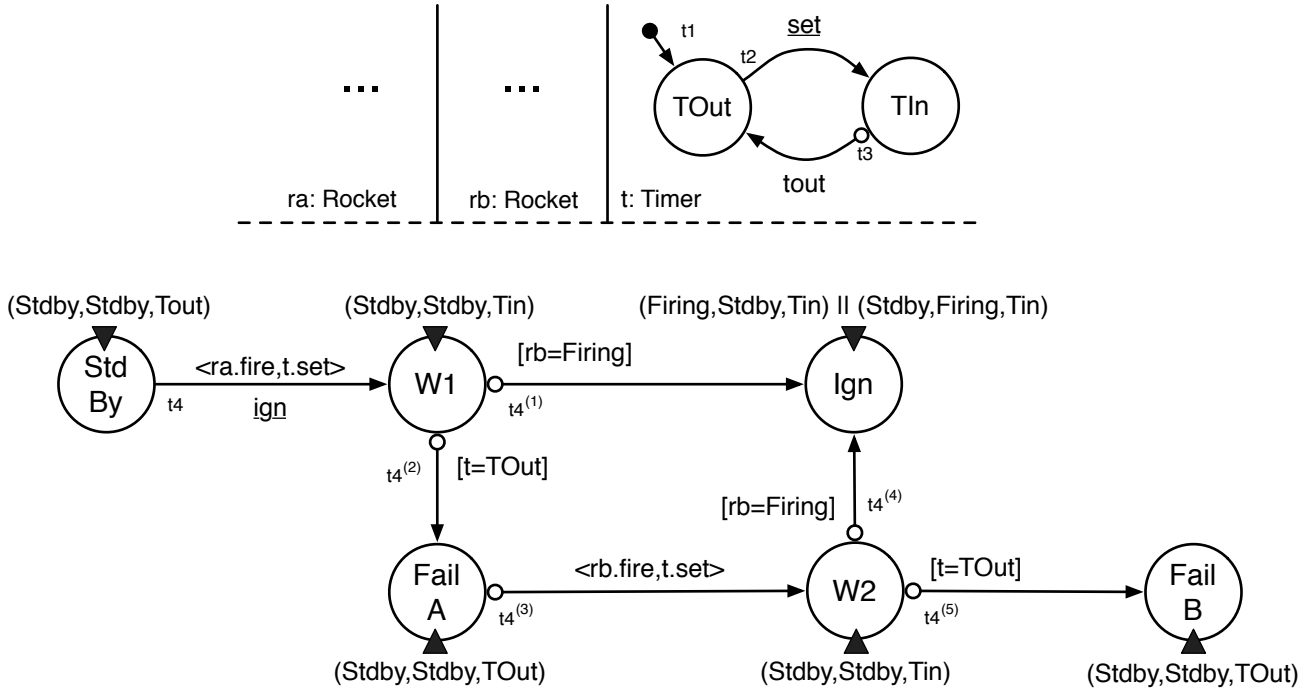

Fig. 6. Ignition of a fault tolerant engine system employing a backup rocket engine.

REFERENCES

Bennett, M., Dvorak, D., Hutcherson, J., Ingham, M., Rasmussen, R., and Wagner, D. (2008). An architectural pattern for goal-based control. *Aerospace Conference, 2008 IEEE*, 1–17. doi:10.1109/AERO.2008.4526594.

Chien, S., Rabideau, G., Knight, R., Sherwood, R., Engelhardt, B., Mutz, D., Estlin, T., Smith, B., Fisher, F., Barrett, T., Stebbins, G., and Tran, D. (2000). Automated planning and scheduling for space mission operations. In *Proceedings of SpaceOps 2000, Toulouse, France*.

Pazzi, L. (1997). Extending statecharts for representing parts and wholes. In *Proceedings of the EuroMicro-97 Conference, Budapest, Hungary*.

Pazzi, L. (2000). Part-whole statecharts for the explicit representation of compound behaviors. In *Proceedings of the UML 2000 Conference, York (UK)*, volume 1939 of *LNCS*, 541–555. Springer.

Pazzi, L. (2008). A method for ensuring safety and liveness rules in a state-based design, http://cris.unimore.it/cris/files/2008-02-01.pdf. Technical Report CRIS-2008-02-01. Patent pending PCT/EP2008/051300.

Truszkowski, W., Hinchey, M., Rash, J., and Rouff, C. (2006). Autonomous and autonomic systems: a paradigm for future space exploration missions. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, 36(3), 279–291. doi: 10.1109/TSMCC.2006.871600.

Williams, B., Ingham, M., Chung, S., and Elliott, P. (2003). Model-based programming of intelligent embedded systems and robotic space explorers. *Proceedings of the IEEE*, 91(1), 212–237. doi: 10.1109/JPROC.2002.805828.