A FRAMEWORK FOR USING FORMAL METHODS IN PROCESS CONTROL

Gábor KOVÁCS¹, Laurent PIETRAC²

¹ Department of Control Systems and Information Technology, Budapest University of Technology and Economics, Budapest, Hungary; gkovacs@hit.bme.hu
²Laboratoire Ampère, INSA Lyon, Lyon, France; laurent.pietrac@insa-lyon.fr

Abstract
This paper presents a novel methodology for the rapid prototyping of supervisory control systems based on Supervisory Control Theory, providing component-based design with the aid of a library allowing the reuse of components. Passing from functional design to technical details is supported by multi-face modelling, which also helps avoiding the problem of state explosion. Implementation of controllers, including distributed architectures, is also presented.

Key words
Discrete event systems, Supervisory control, Distributed control

1 Introduction

In the last decades increasing complexity of industrial systems has proposed new challenges of designing efficient, safe and reliable control systems, still based on human expertise. An important aspect is safety and reliability, which traditionally needs time-consuming test and validation, increasing the on-market time. Supervisory Control Theory (SCT) provides a formal methodology assuring that the closed loop meets the specifications. However, it uses finite state machine models of the process and the specifications making modelling of complex systems and supervisor synthesis a cumbersome task. The approach proposed in this paper remains in the formal framework of SCT, but provides a component library-based methodology for control design, accelerating the modelling phase. Implementation methods for the designed controller, including solutions for distributed deployment, are also presented.

2 Supervisory Control Theory

SCT [1] deals with discrete event reactive systems modelled by finite state machines. Events corresponding to transitions, generated by the process itself, can be controllable or uncontrollable, and the occurrence of the former can be disabled by the supervisor. Models of complex systems can be obtained as the parallel composition of subsystems, where the state set of the composed system is the Cartesian product of the state sets of the subsystems, so the cardinalities are multiplied.

The goal of supervisory control is to limit the operation of the closed loop such that it meets the specifications, which are also modelled as a discrete event system. The supervisor is a function which gives the events to be enabled after the occurrence of a given event sequence. If the specification is not controllable, i.e., closed loop behaviour identical to it cannot be obtained, the closed loop operation has to be restricted in order to respect the specification. A maximal permissive (least restrictive) sublanguage can be found, which impedes everything forbidden by the specification, but limits the operation of the closed loop the less possible. The supervisor restricts the operation of the process such that it generates the maximal permissive sublanguage in closed loop.

3 Framework for rapid prototyping

Fundamentals of the framework will be illustrated on a simple running example of a manufacturing cell composed of a pick and place robotic arm and an assembly station (see Fig. 1).

Fig. 1 Architecture of the manufacturing cell
The operation of the cell is as follows. Palettes equipped with a vertical shaft are loaded onto the turntable of the assembly station by a conveyor, and then forwarded to the assembly zone. The robot takes a pinion from lot #1 and places it onto the palette at the assembly zone, followed by a cap from lot #2. Then the table is turned, so a new palette arrives to the assembly zone.

The robotic arm is composed of three linear joints realized by electric-drive linear axes equipped with end switches, and a vacuum gripper. The assembly station consists of a loading conveyor and a turntable.

3.1 A component-based approach with multi-face modelling

The proposed framework uses a component-based paradigm for modelling complex systems, based on the ideas of modular and hierarchical supervisory control [2],[3]. A component is defined as an object, which can be distinguished from other parts of the system by the mean of its signals and its functionality. Components are divided into two classes. Ones which are built up from other components (e.g. a robotic arm) are referred to as composed components, while others, which can not or needed not to be decomposed, are classified as atomic components. In order to speed up control design, commonly used components are collected to a model library.

Generally, a component can be regarded from different aspects. Let us focus on an axis: one major point of view is the technological one, which needs a detailed model describing the evolution of IOs (motor, end switches). Passing to the abstract, functional point of view of the robot controller, only the position of the axis (extended or retracted) is significant, not the state of the actual IOs. These aspects are represented by discrete event models of different faces.

The basic model of a component is the technological one with events corresponding to the evolution of IOs following from the physical manifestation. The physically possible behaviour described by the technological model is restricted by the safety specifications, which have to be respected during the operation of the component. In case of the axis, safety specifications forbid to retract the axis when it is in fully retracted position or to extend it when it is fully extended. According to the principles of SCT, the restricted behaviour is the supremal controllable part of the technological model respecting the safety specifications and will be referred to as the nominal model. Nominal model of the axis is shown by Fig. 2. Events m0, m- and m+ correspond to the stop and the start of the motor to the two directions, respectively. Events labelled with n or p and arrow symbols correspond to the rising and falling edges of the end switches located at the two extremities.

Based on the nominal model, tasks of the component can be defined. Tasks are such more or less complex activities carried out by the component, which are composed of several events in a given order. The definition of tasks is done by collecting appropriate event sequences of the nominal model to task models, adding the controllable start and confirmation events, which are used to start the given task and to report its successful termination (see again Fig. 2). These events, referred to as the task events, allow to describe complex activities on higher levels of abstraction by only two events. The tasks defined for the axis component are related to extension (e) and retraction (r), as shown by Fig. 2. Start and confirmation events are denoted by s and c for the reason of brevity.

In order to pass to a higher level of abstraction, task models of a component are composed with the nominal model, resulting in the integral model, containing both IO and task events. The integral model is an intermediary one, serving as a bridge to pass on from the nominal to the functional model. The latter is obtained by the natural projection of the integral model to the alphabet of the task events. Therefore, the functional model contains only task start and confirmation events, in the order they are present in the integral model. As illustrated by Fig. 2, passing from the nominal model to the functional one, the size of the state space of the model representing the axis component can be reduced by more than 50%.

Composed components are built up as the synchronous products of functional models of their subsystems. It follows that the models of composed components contain only task start and confirmation events of their sub-components, e.g. the model of the robotic arm contains only start and confirmation events of retract and extend tasks of the three axes and grip and release events of the gripper. Based on these models, using the safety specifications, nominal
model of the composed component can be obtained as in case of the
atomic components. After the definition of task models, the
functional model of the composed component can be obtained as in
case of atomic components.

Remember that the state space of parallel composition of two
subsystems is the Cartesian product of the state spaces of the
subsystems, so by using the functional models, the ensemble of the
three axes can be modelled by $4^3 = 64$ states instead of
$9^2 = 729$, which means that the state space is significantly
reduced.

![Diagram]

*Fig. 2 Nominal model, Task models and Functional model of the axis component XXX*

The procedure of passing from Nominal to Functional models
described afore can be formalized, and it can be proved that the
parallel composition of the Functional and the Task models, playing
an important role in supervisor synthesis, does not violate the safety
specifications [3].

### 3.2 Incremental design of supervisory control

The design of the supervisory control architecture starts with the
successive decomposition of the system, which is continued until the
resulting subsystems can be associated with models in the
component library or cannot be further decomposed. In the
followings we assume that the component library is empty.

At first glance, the manufacturing cell might be decomposed to
two main components, namely the robotic arm and the assembly
station. These two components are independent as there are no
sensor or actuator signals they share and there are no restrictions on
their co-operation resulting from the architecture of the cell (i.e. no
physical constraints are present). However, as it will be seen in the
sequel, functional dependency of these two components will be
established by the cell controller.

The model of the robotic arm can be further decomposed to the
components of three axes and a gripper. Note that, due to the built-up
of the arm, there are no physical constraints on the movement of the
axes relative to each other. Therefore, they can be treated as
individual components, like the vacuum gripper. The model of the
assembly station can be decomposed to the atomic components of
the loading conveyor and the turntable in a similar way.

Following the steps of the decomposition, a component tree can
be assembled, illustrating the relationship of the components (see
Fig. 3). Leaves of the tree represent the basic building blocks of the
system, while its root node stands for the global system to be
controlled.
Control design starts from the leaves of the component tree. In the case of the manufacturing cell, all leaves correspond to atomic components. However, this is not generally true, since if a suitable model for a complex component is found in the library, further decomposition can be omitted. In case of atomic components, if a suitable component model is not found in the library, models defined in the previous section should be obtained, and their ensemble should be added to the library.

After each leaf is associated with a library element, composed components can be formed from leaves with the same parent node. In our example, functional models of the three axes and the gripper are composed, resulting in the technological model of the robotic arm. Safety specifications comprise that the arm can move only with retracted axis Z and that only one piece can be gripped at a time. Based on the resulting nominal model, tasks are defined for picking a piece from lot #1 (see Fig. 4 with task events ending in corresponding to axis n), picking a piece from lot #2, and placing a piece onto the pallet at the assembly zone. Then the functional model, shown by Fig. 5 is obtained, comprising of 5 states. For the assembly station, tasks corresponding to the 90 degrees turn of the turntable and the loading operation carried out by the conveyor are defined in a similar manner. Sub-trees of composed components are then replaced by single leaves representing the robotic arm and the assembly station. Note that these ones are still independent components as no physical constraints are present between them.

![Component tree of the manufacturing cell](image)

**Fig. 3 Component tree of the manufacturing cell**

![Model of the task for picking a component from lot #1](image)

**Fig. 4 Model of the task for picking a component from lot #1**

![Functional model of the robotic arm](image)

**Fig. 5 Functional model of the robotic arm**

To obtain the technological model of the cell, the functional models of the robot and the assembly zone are composed. The moderate size-model of 20 states allows easy definition of safety specifications, compared to the model resulting from the ordinary approach, consisting of 528 states. Safety specifications include forbidding the placement task of the robotic arm during the movement of the turntable in order to avoid the movement of the pallet by the table with the arm over the assembly zone. Since the component of the manufacturing cell is the root of the tree, there are no other components to compose it with, and therefore there is no need to define tasks. On the other hand, functionality of the cell has to be defined. Functional specifications include the sequence of robot tasks (pick from lot #1, place, then pick from lot #2 and place) and the coordination of the operation of the arm and the assembly station. Latter means that the table can be moved only after the second place operation and that each turn must be preceded a loading operation. Note that these specifications are given for the current assembly task (e.g. if the lots containing pinions and caps are replaced by each other, sequence of picking tasks should be changed), in contrast with the safety specifications, which have to be respected regardless the actual functionality. By finding the supremal controllable sublanguage of the model of the cell with respect to the functional
specifications, the model of the closed loop, consisting of 30 states and 48 transitions, can be obtained.

4 Distributed control design

From the functional model of the closed-loop system, an appropriate controller should be obtained. For discrete-event systems, controller is implemented as a state machine, which corresponds to the model of the desired closed loop operation (i.e. the system respecting the specifications), and with control actions defined for its states. For each state, every controllable event should be disabled, expect the one corresponding to a transition leaving the given state (if such transition exists). However, in the everyday practice, control means effective actions carried out by different actuators and not enabling/disabling events generated by the process, as assumed by the theory. Therefore these control actions have to be included to the state machine and a suitable code (e.g. ladder diagram) should be generated. For details, see [5].

Another issue for code generation is how to obtain a controller dealing with IO events from a closed loop based on the top level functional model, containing only functional events. A natural idea to implement the controller is to flatten the closed loop model, replacing task events by the IO event sequences in the corresponding task models. However, a more flexible way is the implementation of task controllers as subroutines, invoked when the appropriate task is started. In our example, at first subroutines of the atomic components manipulating the IOs according to the task models are obtained. The robot and the assembly station controllers are then generated by replacing task events by calls to the aforementioned subroutines. Finally the cell controller, handling subroutines of the robot and the assembly station controller, is obtained.

The framework supports naturally the implementation of distributed control systems as the component tree can be divided into sub-trees and supervisors can be implemented on distinct devices. Obviously, this approach needs a communication channel between the controllers in order to propagate task events of remote subsystems. In our example, the component tree is divided into two pieces at the root node, i.e. resulting sub-trees represent the robotic arm and the assembly station. Let us consider the controller of the robot, on which the cell controller should be implemented such that events of the pick and place tasks start the appropriate task subroutines and send messages to the controller of the assembly station. On the other hand, start and confirmation events of load and turn tasks should be implemented as waiting for the appropriate message from the controller of the assembly station (see Fig. 6). This approach allows a flexible way to react the changes in the physical system. If the robot is replaced by another with a different configuration, there is no need to redesign the whole controller as the functional model remains the same. Only the task models (and controllers) have to be changed in the controller of the robot and the controller of the assembly station can be left unchanged.

Fig. 6 Distributed control architecture

5 Acknowledgement

This research was partially funded by the Hungarian Scientific Research Fund under grant OTKA K 71762.

6 References


