Decentralized Supervisory Control of Reactive Discrete-Event Systems

Liang Du

A Thesis

in

The Department

of

Electrical and Computer Engineering

Presented in Partial Fulfillment of the Requirements

for the Degree of Master of Applied Science at

Concordia University

Montréal, Québec, Canada

October 2006

Your file Votre référence
ISBN: 978-0-494-28912-9
Our file Notre référence
ISBN: 978-0-494-28912-9

ABSTRACT

Decentralized Supervisory Control of Reactive Discrete-Event Systems

Liang Du

In this thesis we propose to apply Ramadge-Wonham supervisory control theory to Reactive Discrete-Event Systems (RDES). A reactive system continually interacts with its environment at the speed dictated by the latter. We will first present our decentralized RDES architecture, which is based on Input/Output (I/O) automata model. After introducing safety and progress, we define the corresponding centralized and decentralized supervisory control problems concerning both safety and progress. We explain through examples why the existing results in supervisory control theory cannot be directly applied. Substitute sufficient and necessary conditions for the existence of decentralized solutions are given. In the special case where only safety is considered, we also study centralized and decentralized supervisory control problems, and present more straightforward sufficient and necessary conditions for the existence of their solutions. An example is presented to illustrate how decentralized RDES are modeled, how decentralized components co-operate with each other, and how the revised decentralized supervisory control theory can be applied to the study of RDES.

ACKNOWLEDGEMENTS

Thanks are due first to my supervisors, Dr. Peyman Gohari and Dr. S. Laurie Ricker. Thanks so much for giving me this opporunity to doing research and for everything they have done for me during the past two years. I must say this thesis cannot be done without the motivation and benefit from their insight, guidance and financial support. Furthermore, I really appreciate their kind counsel on how to work with others, how to write an academic article, and how to pursue a successful career.

My special thanks should give to all my friends in Montreal. Thanks for Yue Yang for his generous help to get me through my first month abroad, and more important, for our great friendship. My dear friend, Wangnan Niu, I will miss you so much in future.

Most important things always come at last. I cannot find a word in any English or Chinese dictionary which can express my love to my dear parents and my beloved Jie. You give me love, hope, and motivity. I will use every minute in my life to love you.

LIANG DU

Montreal, Canada

October, 2006

TABLE OF CONTENTS

LIST OF FIGURES vi

LIST OF TABLES ix

1 Introduction **1**

1.1 Reactive Discrete-Event Systems 1

1.2 Ramadge-Wonham Supervisory Control of DES 3

1.3 Motivation 3

1.4 Related Work 6

1.5 Organization of the Thesis 8

2 Preliminaries **9**

2.1 Formal Languages 9

2.2 Automata Theory 10

2.3 Supervisory Control of DES 13

2.4 Decentralized Supervisory Control of DES 17

2.4.1 Background 17

2.4.2 An Observer Automaton 20

2.4.3 A Monitoring Automaton 22

2.5 I/O Automata 23

2.5.1 Overview 23

2.5.2 Internal Connection 25

2.5.3 Synchronous Product 27

2.6 Conclusion 30

3 Decentralized RDES: System Model and Specification **31**

3.1 Introduction 31

3.2 Decentralized RDES Architecture 32
3.3 Safety and Progress 34
3.4 Problem Definition 40
3.5 Conclusion 41

4 Decentralized Supervisory Control of RDES **42**
4.1 Introduction 42
4.2 R-W Supervisory Control Theory Revisited 43
4.3 Substitute Conditions 46
4.4 Substitute Conditions for Safety 50
4.5 Conclusion 54

5 Example: Pump System **56**
5.1 System Description 56
5.2 Analysis 64
5.2.1 A Service Specification E 64
5.2.2 Decentralized Supervisors that Implement E 65
5.3 Pump System: Revisited 67
5.4 Conclusion 73

6 Conclusion and Future work **75**
6.1 General Conclusion 75
6.2 Future Work 76

Bibliography **78**

LIST OF FIGURES

1.1 RDES in daily life: the digital watch. 2

2.1 Architecture of centralized supervisory control. 14

2.2 Architecture of decentralized supervisory control. 18

2.3 The observer automata of a DES: (a) the plant G; (b) $Obs_1(G)$; (c) $Obs_2(G)$ (adopted from [1]). 21

2.4 The monitoring automata for G, $Obs_1(G)$ and $Obs_2(G)$ in Figure 2.3. 23

2.5 I/O automaton: an explicit view. 25

2.6 Composition of compatible I/O automata. 29

3.1 Decentralized RDES architecture. 33

3.2 Interpretation of progress, where *cat* denotes the operation of catenation of strings. 36

3.3 K_1 and K_2 satisfy safety but $K_1 \cap K_2$ does not. 40

4.1 E is not controllable with respect to $P_{ex}(L(G))$, but a centralized solution to Problem 3.4.1 under full observation exists. Controllable events are graphically represented by ticks on their transition arrows. 44

4.2 E is not observable with respect to $P_{ex}(L(G))$ and $P_{ex,o}$, but a centralized solution to Problem 3.4.1 under partial observation exists. 45

4.3 Example 4.1 revisited to illustrate Proposition 4.3.1. 47

4.4 Example 4.2 revisited to illustrate Proposition 4.3.2. 48

4.5 Example 4.3: components, the plant, and the external plant behavior. 49

4.6 Example 4.3: the service specification and the decentralized solution. 49

5.1 Pump system: the system architecture. 57

5.2 Pump system: pipe 1. 58
5.3 Pump system: pipe 2. 59
5.4 Pump system: the pump. 61
5.5 Pump system: the entire plant behavior. Illegal events are shown by dashed lines. 63
5.6 Pump system: the service specification. 65
5.7 Pump system: $E' = \sup\mathscr{C}(P_{ex}^{-1}(E) \cap L(G))$. 66
5.8 Pump system: the plant under the supervision of S_{pp}. Events that are illegal but uncontrollable to S_{pp} (i.e., should be disabled by S_{pi1}) are shown by dashed lines. 68
5.9 Pump system: the plant under the supervision of S_{pi1}. Events that are illegal but uncontrollable to S_{pi1} (i.e., should be disabled by S_{pp}) are shown by dashed lines. 69
5.10 Observer and monitoring automata for a selected part of the plant. 72
5.11 Incorporating communication into RDES. 74

LIST OF TABLES

3.1 K_1 satisfies progress 39

3.2 K_2 satisfies progress 39

3.3 $K_1 \cap K_2$ does not satisfy progress 39

Chapter 1

Introduction

1.1 Reactive Discrete-Event Systems

Nuclear power plants, telecommunication networks, automobiles and e-commerce applications share a common characteristic: each functions in a mode of continuous interaction with an unpredictable environment. Such systems are called *reactive*, which continuously receive inputs from and react to their environments (by possibly sending outputs) at speeds determined by the latter. In fact, most embedded systems, many real-time systems, and most object-oriented software systems fall into the category of reactive systems.

In contrast with *transformational* systems that receive inputs, generate results and then terminate, reactive systems typically maintain in continuous interaction with their environment, and this process of interaction is usually non-terminating. If such a system terminates automatically, this is usually caused by an internal error. To summarize, a reactive system should have two key properties: (1) it interacts with the environment at the speed decided by the latter; (2) this interacting process terminates only if it is told to or an error occurs.

This thesis is concerned with event-driven reactive systems which can be modeled as *Discrete-Event Systems* (DES), which we will call *Reactive Discrete-Event Systems* (RDES).

Many e-commerce systems, such as on-line auctioning systems [2], are examples of RDES. An on-line auctioning system provides a distributed structure for consumers to meet one another, negotiate prices and trade goods. The electronic auction takes place according to the consumers' inputs, and consumers can react to this system as long as they want unless some failure happens such as the server going down.

Moreover, there are numerous other applications of RDES in our daily life, including consumer electronics (such as digital watches, digital cameras, and mp3 players), household appliances (such as microwave ovens and washing machines), and personal telecommunication systems (such as modems, mobile phones, and answering machines). Figure 1.1 shows the model of a digital watch [3].

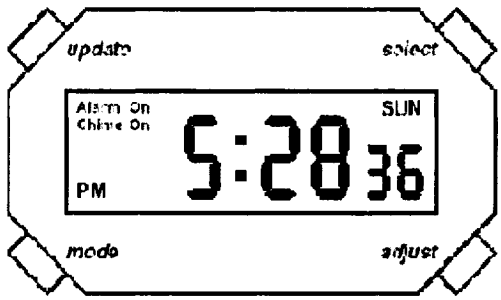


Figure 1.1: RDES in daily life: the digital watch.

The watch is off before the user turns it on. When it is on, the watch stays in the "working" state unless the user resets it, adjusts it, turns it off or some failure happens such as the battery is flat. In other words, the interaction follows a pace dictated by the user. Therefore, the digital watch is an example of a simple RDES.

1.2 Ramadge-Wonham Supervisory Control of DES

A Discrete-Event System is an event-driven dynamic system that typically possesses a discrete (in time domain) state space and a state-transition structure. Many physical systems fall into the category of DES, such as traffic systems, database management systems, many computer programs, telecommunication systems and industrial manufacturing systems [4].

This thesis proposes to apply the *Ramadge-Wonham supervisory control theory* to RDES. In the Ramadge-Wonham framework, a DES is modeled as the generator of a formal language and can be thought of as the set of trajectories (or the behavior) of a *plant.* A *string* is a sequence of events executable by the plant, and therefore it represents a run of the plant. To tune the system behavior, the whole alphabet is partitioned into the set of *controllable* and *uncontrollable* events. A *supervisor* is an external agent in charge of controlling the plant. It observes a sequence of events generated by the plant, and it can restrict its possible extensions by disabling a subset of controllable events. Thus, the behavior of the supervised plant either equals or is a restriction (or a sublanguage) of the specification of the desired behavior.

Since RDES involves interactions with the environment, it is no longer suitable to model RDES by standard automata. We will make several minor changes to the *Input/Output (I/O) automata* introduced by Lynch and Tuttle [5] to model decentralized RDES. With the alphabet partitioned into input events (inputs), output events (outputs) and internal events, an I/O automaton receives inputs and generates outputs and internal events autonomously and instantaneously.

1.3 Motivation

The development of formal methods that guarantee the correctness and validity of reactive systems is a fundamental objective of current strategies for the study of these systems. There is a significant distinction between formal methods that "check" system properties and those

methods that are based on control theory. Techniques based on checking such as static model-checking and theorem proving confirm that the model of the system satisfies certain fundamental safety and liveness properties and, if necessary, diagnoses any errors.

Formal methods based on control theory (such as supervisory control theory) do not check that the system is correct, but rather, prohibit certain behaviors that lie outside a predefined set of acceptable behavior. In the control theoretic framework, a key stage is the synthesis of the controllers. The operation of the resulting controllers guarantees that when the system operates in tandem with the controllers, the system will not perform any unacceptable behavior. Furthermore, such controllers must be sufficiently restrictive to guarantee certain properties and must be sufficiently permissive so as not to reduce the behavior of the system unnecessarily.

For RDES, we are interested in satisfying *service specifications* of RDES using supervisory control. Unlike specifications in supervisory control framework which are defined on the whole alphabet, service specifications are defined only on the *external* alphabet (namely the union of inputs and outputs). Generally speaking, a controlled plant is said to *implement* a service specification if certain properties called *safety* and *progress* hold.

Safety of reactive systems is of paramount importance in system design, since any violation of safety guidelines could result in loss of life or excessive economic damage. For instance, the consequence of a system error in an aircraft automatic pilot system or in a nuclear power plant controller could be disastrous. Informally, a safety specification stipulates that "bad things never happen" and must hold in every state of the system.

Furthermore, progress requires that the controlled system should not block the occurrence of an external event whenever it is allowed in the specification. For instance, if the specification requires sending a certain output in response to some input, but in *some* sequences leading from the input to the output an internal event is disabled by the controller, then the progress condition is violated.

Given an RDES and a service specification, it is interesting to know under what conditions a supervisor will implement the service specification. Since the plant behavior of RDES and the service specification are defined on different alphabets, some of existing results in supervisory control framework cannot be directly applied to RDES. For instance, we will show through examples that, unfortunately, the controllability condition in supervisory control framework is generally no longer necessary for the existence of such a supervisor under full observation, and the controllability and observability conditions are generally no longer necessary for the existence of such a supervisor under partial observation. These problems arise because the resulting control strategy does not take into account any information about the architecture of RDES that requires control, such as the role of internal events which are erased in the service specification. Therefore, we will investigate how they should be modified.

An essential feature of reactive systems is that they are usually distributed [6]. Most reactive systems contain distributed simultaneous components and modules that interact with one another through message passing of some sort. Thus decentralized RDES control problems arise naturally.

To summarize, the following challenges exist for the control-based analysis of RDES.

- Finding a compact but meaningful mathematical framework to model how distributed RDES co-operate with each other.
- Ensuring that the formal framework can specify decentralized architectures and synthesize the necessary controllers.
- Checking whether existing supervisory control theory can be directly applied to RDES. If not, import necessary modifications and extend the existing work into the formal framework of decentralized RDES.

1.4 Related Work

Reactive systems have been of interest to computer scientists for many years. The pioneer of verification of reactive systems is Pnueli, who started his work on reactive systems in 1985. Harel and Pnueli [7] singled out reactive systems as being a special problem for their importance. They compared reactive systems with transformational systems, and used the *State-Chart* method to describe their behavior.

In 1989, Lynch and Tuttle [5] used I/O automata to model DES. In this work, an I/O automaton is defined as a tool for modeling concurrent and distributed DES of the sorts arising in computer science. We will make some minor but necessary modifications to make it an appropriate framework for decentralized control of RDES.

Over the past 15 years, many languages and techniques have been used to specify programs and verify reactive systems in the computer science literature. These languages and techniques can be also used to simulate RDES. Typical approaches include graphical languages based on automata (such as *SyncCharts* [8] and *Argos* [9]), data-flow based languages (such as *Signal* [10] and *Lustre* [10]), and imperative languages (such as *Esterel* [11]).

Although so many languages and techniques are designated to simulate reactive systems, formal methods of reactive system based on supervisory control theory were not considered until recently. Jéron et al. [12] introduced supervisory control into RDES framework. They follow Lynch and Tuttle in using I/O automata for RDES and study the problem of controlling an implementation of reactive systems in order to make it conform with a given specification. The proposed solution is to control the implementation by means of an automatically computed supervisor, where ensuring a conformance relation between the implementation and its formal specification constitutes the control objective. The supervisors are seen as a "patch" that automatically fixes errors, which otherwise should have been discovered by testing and fixed by hand.

Other work that also proposes an input/output interpretation of supervisory control of DES includes the work of Balemi [13]. Compared with [12], which can be interpreted as the centralized supervisory control problem of RDES with zero tolerance (i.e., the controlled plant behavior equals the specification), Balemi's work presents the centralized supervisory control problem of RDES with tolerance (i.e., the controlled plant behavior is a subset of the specification). In other words, the *supervisor synthesis problem with local specifications* defined by Balemi requires finding a supervisor which is nonblocking and complete, and the *projection* of the closed-loop system behavior is nonempty and is a subset of the given local specification (which is defined only on a subset of the entire alphabet). Necessary and sufficient conditions for the existence of a solution are given.

Both [12] and [13] are concerned only with the centralized supervisory control problem. In contrast, we study the decentralized supervisory control problem of RDES with zero tolerance, and give corresponding necessary and sufficient conditions for the existence of a solution in [14].

The supervisory control problem of RDES is different from the standard supervisory control problem because the plant behavior and the service specification are defined on different alphabets. Several researchers have considered supervisory control problem of DES where the alphabet is partitioned into several subsets, for example, in the protocol conversion problem, i.e., the problem of reliable transmission of data over unreliable communication channels. The protocol conversion problem was first treated by Inan in the supervisory control framework as an important application example in [15], and the theoretical foundation on which this application is based was given in [16]. However, the work only addresses the *safety* constraint of the protocol conversion problem which requires that the projected language of the supervised system should equal the given specification language. The additional *progress* constraint, which requires that the supervised system never blocks an external event that is not blocked by the specification itself, was also considered as part of the protocol conversion

problem in the work of Calvert and Lam [17] and redefined in-depth by Kumar, Nelvagal and Marcus in [18].

1.5 Organization of the Thesis

The rest of this thesis is organized as follows: In Chapter 2 we study the mathematical preliminaries, including the syntax and semantics of automata theory, DES and supervisory control. We will also introduce modified I/O automata framework. In Chapter 3, we present the decentralized RDES architecture and introduce safety, progress and service specifications. In Chapter 4, we will define the corresponding supervisory control problems in two cases: when both safety and progress are considered, and when only safety is considered. Sufficient and necessary conditions for the existence of their solutions will be given. In the latter case where only safety is considered, the conditions are easily verifiable on the transition structure of the system. An example will be studied in Chapter 5 to illustrate how we model decentralized RDES, how decentralized components co-operate with each other, and how revised supervisory control theory and decentralized supervisory control theory can be applied. Chapter 6 summarizes this thesis and discusses future work.

Chapter 2

Preliminaries

In this chapter we proceed as follows. Section 2.1 provides a brief introduction to some basic concepts from formal languages. Section 2.2 reviews some standard concepts from automata theory. The fundamental elements of centralized and decentralized supervisory control theory are reviewed in Section 2.3 and Section 2.4, respectively. After presenting the Input/Output automata framework in Section 2.5, we summarize this chapter in Section 2.6.

2.1 Formal Languages

Let Σ be a finite set of distinct symbols, which will be referred to as an *alphabet* from now on. The set of all finite symbol *strings*, of the form $\sigma_1\sigma_2\ldots\sigma_k$ where $k \geq 1$ is arbitrary and $\sigma_i \in \Sigma$, is denoted by Σ^+. Let $\varepsilon \notin \Sigma$ denote the *empty string* (a sequence with no symbols), we have the set of all possible sequences $\Sigma^* = \{\varepsilon\} \cup \Sigma^+$, where an element of Σ^* is a string over the alphabet Σ.

Two strings can be joined with the operation of *catenation*, where $cat : \Sigma^* \times \Sigma^* \rightarrow \Sigma^*$ is

defined according to

$$\begin{aligned} cat(\varepsilon,s) &= cat(s,\varepsilon) = s, \quad s \in \Sigma^*; \\ cat(s,t) &= st, \quad s,t \in \Sigma^*. \end{aligned}$$

Any string in Σ^* can be generated by concatenating event symbols from Σ.

The *length* of a string $s \in \Sigma^*$ is denoted by $|s|$ and defined according to

$$\begin{aligned} |\varepsilon| &= 0; \\ |s| &= k, \ \textit{if } s = \sigma_1\sigma_2\ldots\sigma_k \in \Sigma^+. \end{aligned}$$

We can apply this operator to *cat* as follows: $|cat(s,t)| = |s| + |t|$, $s,t \in \Sigma^*$.

Let A be an arbitrary set and let $X = Pwr(A)$ be the set of all subsets of A (the *power set* of A). A *language* over A is any subset of Σ^*, i.e., an element of the power set $Pwr(\Sigma^*)$; thus the definition includes both the empty language $\emptyset$ (the language with no strings), and Σ^* itself.

For $s,t \in \Sigma^*$, t is called a *prefix* of s and denoted by $t \leq s$ if $s = tu$ for some $u \in \Sigma^*$. Let $L \subseteq \Sigma^*$ be an arbitrary language over Σ. Then the *prefix closure* $\overline{L}$ of L is the language consisting all prefixes of strings in L:

$$\overline{L} := \{t \in \Sigma^* | t \leq s \ \textit{for some} \ s \in L\}.$$

Note that $L \subseteq \overline{L} \subseteq \Sigma^*$. $\overline{L} = \emptyset$ if $L = \emptyset$, and $\varepsilon \in \overline{L}$ if $L \neq \emptyset$. For a string $s \in \Sigma^*$ we will denote its set of prefixes by $\overline{s}$ instead of $\overline{\{s\}}$. A language L is *closed* if $\overline{L} = L$. In this thesis, we assume that all languages are closed.

2.2 Automata Theory

An *automaton* is a mathematical model which is capable of representing a regular language. Formally, an automaton A over the alphabet Σ is a 4-tuple

$$A = (Q, \Sigma, \delta, q_0),$$

where Q is the nonempty set of *states*, Σ is the alphabet of *events*, $\delta : Q \times \Sigma \rightarrow Q$ is the (partial) *state transition function*, and $q_0 \in Q$ is the *initial state*.

Let $q, q' \in Q, \sigma \in \Sigma$; then $\delta(q,\sigma) = q'$ means that there is a state transition labeled by event σ from state q to state q'. We extend the domain of δ from events to strings with $\hat{\delta} : Q \times \Sigma^* \rightarrow Q$ and define $\hat{\delta}$ recursively as follows. Let $q \in Q, \sigma \in \Sigma$ and $s \in \Sigma^*$, then

1. $\hat{\delta}(q,\varepsilon) = q$;
2. $\hat{\delta}(q,\sigma) = \delta(q,\sigma)$;
3. $\hat{\delta}(q,s\sigma) = \hat{\delta}((\hat{\delta}(q,s)),\sigma)$.

In the rest of this thesis, we omit the $\hat{}$ and write δ in place of $\hat{\delta}$.

Given an automaton G, the language $L \subseteq \Sigma^*$ *recognized* by G is

$$L := \{s \in \Sigma^* | \delta(q_o,s) \in Q\}.$$

G is called a *recognizer* for L.

A state $q \in Q$ is *reachable* if $q = \delta(q_0,s)$ for some $s \in \Sigma^*$; and an automaton G is *reachable* if q is reachable for all $q \in Q$.

Let $L_1 \subseteq \Sigma_1^*$ and $L_2 \subseteq \Sigma_2^*$ be two languages, where, in general, $\Sigma_1 \cap \Sigma_2 \neq \emptyset$. Let $\Sigma = \Sigma_1 \cup \Sigma_2$. Define a mapping

$$P_i : \Sigma^* \rightarrow \Sigma_i^*, i \in \{1,2\}$$

according to

$$\begin{aligned} P_i(\varepsilon) &= \varepsilon; \\ P_i(\sigma) &= \begin{cases} \sigma, & \text{if } \sigma \in \Sigma_i; \\ \varepsilon, & \text{if } \sigma \notin \Sigma_i; \end{cases} \\ P_i(s\sigma) &= P_i(s)P_i(\sigma),\ s \in \Sigma^*, \sigma \in \Sigma. \end{aligned}$$

The map P_i is called the *natural projection* (or *canonical projection*) of Σ^* onto Σ_i^*. The action of P_i on a string s is to remove all occurrences of σ in s where $\sigma \notin \Sigma_i$. Note that the natural projection can be extended to the power set of Σ^* in the usual way.

Let the inverse image function of P_i be defined by

$$P_i^{-1} : Pwr(\Sigma_i^*) \rightarrow Pwr(\Sigma^*), i \in \{1,2\},$$

where for $K \subseteq \Sigma_i^*$,

$$P_i^{-1}(K) := \{s \in \Sigma^* | P_i(s) \in K\}.$$

For $L_1 \subseteq \Sigma_1^*$ and $L_2 \subseteq \Sigma_2^*$, the *synchronous product* $L_1 \| L_2 \subseteq \Sigma^*$ of L_1 and L_2 is defined according to

$$L_1 \| L_2 := P_1^{-1}(L_1) \cap P_2^{-1}(L_2).$$

Thus, $s \in L_1 \| L_2$ if and only if $P_1(s) \in L_1$ and $P_2(s) \in L_2$. If $G_1 = (Q_1, \Sigma_1, \delta_1, q_{10})$ and $G_2 = (Q_2, \Sigma_2, \delta_2, q_{20})$ are recognizers for L_1 and L_2, respectively, then the synchronous product $G_1 \| G_2$ of G_1 and G_2 is the automaton defined as

$$G_1 \| G_2 := (Q_1 \times Q_2, \Sigma_1 \cup \Sigma_2, \delta, (q_{10}, q_{20})),$$

where for $(q_1, q_2) \in Q_1 \times Q_2$ and $\sigma \in \Sigma_1 \cup \Sigma_2$,

$$\delta((q_1,q_2),\sigma) = \begin{cases} (\delta_1(q_1,\sigma), \delta_2(q_2,\sigma)), \\ \quad \text{if } \delta_1(q_1,\sigma) \text{ is defined} \wedge \delta_2(q_2,\sigma) \text{ is defined;} \\ (\delta_1(q_1,\sigma), q_2), \\ \quad \text{if } \delta_1(q_1,\sigma) \text{ is defined} \wedge \sigma \notin \Sigma_2 \text{ ;} \\ (q_1, \delta_2(q_2,\sigma)), \\ \quad \text{if } \sigma \notin \Sigma_1 \wedge \delta_2(q_2,\sigma) \text{ is defined;} \\ \text{undefined,} \quad \text{otherwise} \end{cases} .$$

The language $L_1 \| L_2$ recognized by the synchronous product $G_1 \| G_2$ of G_1 and G_2 can be viewed as generated cooperatively by agreeing to synchronize on those events which are defined in both G_1 and G_2.

The automaton $G_1 \| G_2$ satisfies the following equation:

$$L(G_1 \| G_2) = L(G_1) \| L(G_2).$$

2.3 Supervisory Control of DES

This thesis is built on the supervisory control framework for discrete-event systems developed by Ramadge and Wonham [19, 20], and decentralized supervisory control presented by Cieslak et. al. [21] and Rudie and Wonham [22]. A supervisory control approach requires the design of a supervisor (or at least two supervisors in the decentralized case) to oversee the system and issue control commands to keep the behavior of the controlled system within some *a priori* determined subset of desirable behavior. Such a supervisor, based on its observations of the plant behavior, either *enables* or *disables controllable* events to prevent the system from performing illegal moves.

In this section we review the basics of the Ramadge-Wonham supervisory control framework (where the overall plant is controlled by one supervisor) and the decentralized supervisory control theory (where at least two supervisors are synthesized to achieve an overall control objective). In the centralized problem, it may be the case that not all events can be seen, i.e., the supervisor has only a partial view of the plant behavior. In the decentralized case, the overall supervisory task is divided into two or more subtasks and each supervisor, likewise, has only a partial view of the plant behavior. Each subtask is solved using the results of centralized supervisory control under partial observation.

In the supervisory control framework, the discrete-event system requiring control, i.e., the plant in traditional control terminology, is the generator of a formal language, and we

model it by an automaton which is denoted by G. By designing a supervisor (a controller which monitors the plant and issues control commands), it is desired to keep the plant behavior within a pre-defined region. In other words, the language recognized by the supervised generator is contained in a specification language. The control task is considered fully accomplished if such a supervisor (or supervisors) can be found.

The one-plant-one-supervisor centralized architecture is shown in Figure 2.1(a), while Figure 2.1(b) shows the case when the controller supervises a composite plant, consisting of several parallel components whose alphabets are disjoint.

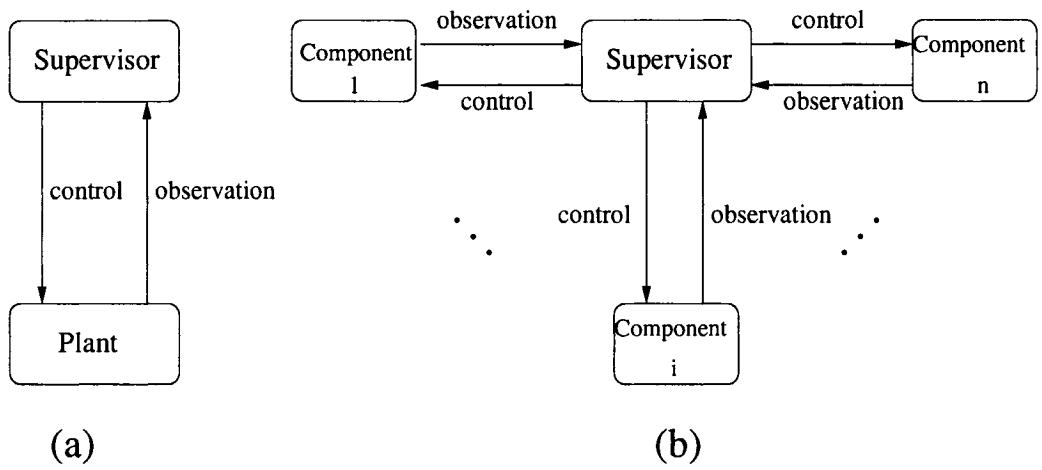


Figure 2.1: Architecture of centralized supervisory control.

Furthermore, an automaton G is characterized by a subset of Σ^* called the *closed behavior* of G, written as $L(G)$, and defined as

$$L(G) := \{s \in \Sigma^* | \delta(q_0, s) \in Q\}.$$

The closed behavior of an automaton can be interpreted as the set of all possible sequences of state transitions.

To impose supervision on the plant, we partition the alphabet of event labels $\Sigma = \Sigma_c \dot{\cup} \Sigma_{uc}$

into two sets called the *controllable* and *uncontrollable* event sets, respectively. Controllable events can be disabled by the supervisor, if necessary, while uncontrollable events are considered to be permanently enabled.

A language $K \subseteq \Sigma^*$ is *controllable* with respect to G (controllable, for short, where G is understood) if and only if

$$K\Sigma_{uc} \cap L(G) \subseteq K.$$

In other words, K is invariant under the occurrence of uncontrollable events in $L(G)$. K can be viewed as a specification of some legal behavior. Controllability requires that if s is legal, σ is uncontrollable, and $s\sigma$ is physically possible, then $s\sigma$ must be legal as well. That is, all events triggering state transitions that exit from the specification language K to the plant language $L(G)$ must be controllable.

To understand the way in which a control policy can be defined, a particular subset of events that should be enabled (others should be disabled) can be selected by specifying a subset of controllable events. It is convenient to adjoin with the set of enabled controllable events all the uncontrollable events, as they are always enabled. Each such subset of events is called a *control pattern*, and the set Γ of all control patterns is defined as follows.

$$\Gamma = \{\gamma \subseteq \Sigma | \gamma \supseteq \Sigma_{uc}\}$$

Given a plant G, a *supervisory control* for G is any map $V : L(G) \rightarrow \Gamma$. The pair (G,V) will be written as V/G, to suggest "G under the supervision of V". The closed behavior of V/G is defined to be the language $L(V/G) \subseteq L(G)$ defined as follows.

1. $\varepsilon \in L(V/G)$;

2. $s \in L(V/G) \wedge \sigma \in V(s) \wedge s\sigma \in L(G) \Rightarrow s\sigma \in L(V/G)$;

3. No other strings belong to $L(V/G)$.

Let S be any DES modeled by an automaton. Then S is said to *implement* V if

$$L(S\|G) := L(S) \cap L(G) = L(V/G)$$

S is a *supervisor* for G if $L(S)$ is controllable with respect to G.

To summarize, a supervisor is an agent that observes sequences of events generated by G and makes decisions to either enable or disable any of the controllable events at any point throughout the evolution of G. By performing such a manipulation on controllable events, the supervisor ensures that only the subset of $L(G) \cap L(S)$ is permitted to be generated as required.

The controllability condition is necessary and sufficient for the existence of a supervisor under full observation as shown in the following theorem.

Theorem 2.3.1 [19] *Given a plant G, $K \subseteq L(G)$ and $K \neq \emptyset$, there exists a supervisory control V for G such that $L(V/G) = K$ if and only if K is controllable with respect to G.*

So far, we have assumed that a supervisor can observe and record all events generated by the plant. In real applications, a more realistic problem of supervisory control is that a supervisor S may have only a partial observation of the system's behavior, i.e., only a subset of event labels generated by the plant can actually be observed by the supervisor.

Consequently, Σ can be partitioned into Σ_o of observable and Σ_{uo} of unobservable events. Note that S can potentially disable controllable events that are not observable. That is, the subset Σ_o need not in general have any particular relation to the subset of controllable events Σ_c. In this thesis, we assume that each event is either controllable or observable, or both.

We associate with Σ_o a natural projection $P_o : \Sigma^* \rightarrow \Sigma_o^*$ defined in Section 2.1, which can be interpreted as a supervisor's view of the plant behavior. Thus, the effect of P_o on a string s is to erase from s those events that are not observable. For $K \subseteq L(G), P_o(K)$ denotes the language $\{P_o(s) | s \in K\}$. Let S be a supervisor that implements a supervisory control V. S is said to be *feasible* if

$$\forall s, s' \in \Sigma^*, P_o(s) = P_o(s') \Rightarrow V(s) = V(s').$$

A language $K \subseteq \Sigma^*$ is *observable* [23] with respect to (G, P_o) (observable for short when G and P_o are understood) if for all $s, s' \in \Sigma^*$, and $\sigma \in \Sigma_c$, $P_o(s) = P_o(s')$ implies that

$$s\sigma \in K \wedge s' \in K \wedge s'\sigma \in L(G) \Rightarrow s'\sigma \in K.$$

In other words, observability requires that if two strings look identical (i.e., have the same projection) to a supervisor, then a control action which applies to one must be applied to the other as well. By contrast, if K is not observable, then an event (observable or not) may lead to different outcomes with respect to membership in K for look-alike strings; for example, if $P_o(s)\sigma = P_o(s')\sigma$, it could be the case that σ is disabled after s while enabled after s', and thus takes the system outside of K.

The observability condition and the controllability condition are necessary and sufficient for the existence of a supervisor under partial observation as shown in the following result.

Theorem 2.3.2 [23] *Given a plant G, $K \subseteq L(G)$ and $K \neq \emptyset$, there exists a feasible supervisory control V for G such that $L(V/G) = K$ if and only if*

1. *K is controllable with respect to G, and*
2. *K is observable with respect to (G, P_o).*

2.4 Decentralized Supervisory Control of DES

2.4.1 Background

RDES, include embedded systems and real-time systems, which are typically concurrent and distributed; therefore, control of RDES can naturally be formulated as a decentralized control problem. In this section we review the decentralized approach to the synthesis of supervisors for DES. In some cases, we may have a single plant that requires multiple supervisors. In other

cases, we need to find a set of local supervisors, each one designed to monitor and control a plant component. For example, in distributed systems, the plant components are geographically widely separated and a centralized supervisor satisfying the global control objectives may not be designed. In general, the overall supervisory task is divided into two or more subtasks, each accomplished by a local supervisor. The resulting local supervisors run concurrently to implement a given global specification, and we refer to the resultant supervisors as *decentralized supervisors*. The architecture is sketched below.

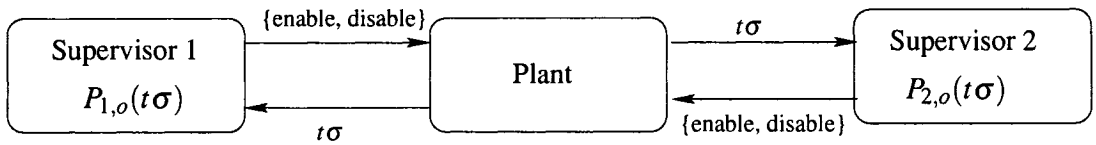


Figure 2.2: Architecture of decentralized supervisory control.

A local supervisor processes only a projection of the behavior of the DES to be controlled. Given an index set $I = \{1,2,\ldots n\}$, the corresponding subsets of events that local supervisor i can observe is denoted by $\Sigma_i \subseteq \Sigma$ (not necessarily disjoint), where $i \in I$ and $\bigcup_i \Sigma_i = \Sigma$. Assume that the overall legal specification on the behavior of G is expressed by the language

$$E = \bigcap_i P_i^{-1} E_i \in \Sigma^*$$

where $i \in I$ and $E_i \in \Sigma_i^*$. In other words, global legal behavior can be reduced to the simultaneous satisfaction of local legal specifications expressible as sublanguages Σ_i^*.

Let $\Sigma_c = \bigcup_{i \in I} \Sigma_{i,c}$ and $\Sigma_o = \bigcup_{i \in I} \Sigma_{i,o}$. Given a supervisor S_i controlling only events in $\Sigma_{i,c}$ while observing only events in $\Sigma_{i,o}$, S_i is called a *local* supervisor and its *global* extension $\widetilde{S_i}$ denotes the supervisor which takes the same control action as S_i on $\Sigma_{i,c}$, enables all events in $\Sigma - \Sigma_{i,c}$, makes the same transitions as S_i on $\Sigma_{i,o}$ and stays in the same state for events in $\Sigma - \Sigma_{i,o}$.

Two decentralized control problems, where local supervisors are sought to satisfy global specifications, called *GP* (Global Problem) and *GPZT* (Global Problem with Zero Tolerance), are described by Rudie and Wonham in [22], in which a global specification is to be satisfied by a set of local supervisors. For simplicity, we present the decentralized problem with two supervisors.

Problem 2.4.1 *(Global Problem* [22]*) Given a plant G over an alphabet* Σ*, a legal language* $E \subseteq L(G)$*, a minimally adequate language* $A \subseteq E$*, and sets* $\Sigma_{1,c}, \Sigma_{2,c}\Sigma_{1,o}, \Sigma_{2,o} \subseteq \Sigma$*, construct local supervisors* S_1 *and* S_2 *such that* $S = \widetilde{S}_1 \wedge \widetilde{S}_2$ *is a proper supervisor for G and such that*

$$A \subseteq L(\widetilde{S}_1 \wedge \widetilde{S}_2/G) \subseteq E.$$

Here, for $i = 1,2$, supervisor S_i can observe only events in $\Sigma_{i,o}$ and can control only events in $\Sigma_{i,c}$, and $\widetilde{S}_i$ is the global extension of S_i.

The solution to this general case can be derived from the solution to a more restricted case formulated as follows.

Problem 2.4.2 *(Global Problem with Zero Tolerance* [22]*) Given a plant G over an alphabet* Σ*, a language* $E \subseteq L(G)$ *and* $E \neq \emptyset$*, and sets* $\Sigma_{1,c}, \Sigma_{2,c}\Sigma_{1,o}, \Sigma_{2,o} \subseteq \Sigma$*, construct local supervisors* S_1 *and* S_2 *such that* $S = \widetilde{S}_1 \wedge \widetilde{S}_2$ *is a proper supervisor for G and such that*

$$L(\widetilde{S}_1 \wedge \widetilde{S}_2/G) = E.$$

Here again, for $i = 1,2$, supervisor S_i can observe only events in $\Sigma_{i,o}$ and can control only events in $\Sigma_{i,c}$, and $\widetilde{S}_i$ is the global extension of S_i.

It can be seen that the *GPZT* is a special case of the *GP*. The *GPZT* fits into the *GP* if we let the endpoints of the range of behavior in *GP* be equal, i.e., $A = E$.

The key to solving both the *GP* and *GPZT* is a property called co-observability [22]. A language $K \subseteq \Sigma^*$ is *co-observable* with respect to $(L(G), P_{1,o}, \Sigma_{1,c}, P_{2,o}, \Sigma_{2,c})$ if $\forall s \in K$ and

$\forall \sigma \in \Sigma_c$:

$$s\sigma \notin K \wedge s\sigma \in L(G) \Rightarrow (\exists i \in \{1,2\})\ P_{i,o}^{-1}[P_{i,o}(s)]\sigma \cap K = \emptyset\ \wedge\ \sigma \in \Sigma_c^i.$$

In other words, co-observability states that if an event generated by the plant leads the system to illegal behavior, then at least one of the two supervisors must know without ambiguity when it must disable this event. On the other hand, if the event does not lead the sequence generated so far into illegal behavior, then neither of the supervisors should disable it.

Controllability and co-observability properties are necessary and sufficient conditions for the existence of a solution to the *GPZT*, as indicated by the following result.

Theorem 2.4.1 [22] *There exist supervisors S_1 and S_2 for G that solve the GPZT problem if and only if*

1. *K is controllable with respect to G, and*
2. *K is co-observable with respect to $(L(G), P_{1,o}, \Sigma_{1,c}, P_{2,o}, \Sigma_{2,c})$.*

2.4.2 An Observer Automaton

In decentralized supervisory control problem, a local supervisor has only a partial observation of the plant behavior. Since such a local supervisor cannot observe the occurrence of every event, it may be unaware of the exact state the plant is in. For example, suppose that the plant is currently in state q, and $\delta(q,\sigma) = q'$. If the local supervisor cannot observe σ, then the supervisor will not know whether the plant is in state q or q'. In other words, the supervisor's view of the current state of the plant is $\{q, q'\}$.

We use the *observer automaton* to capture the view of the plant by supervisor i. Given an automaton $G = (Q, \Sigma, \delta, q_0)$, the observer automaton corresponding to supervisor S_i is defined to be $Obs_i(G) = (Q_i, \Sigma, \delta_i, q_{i,0})$, where $Q_i \subseteq Pwr(Q)$, $\delta_i : Q_i \times \Sigma \rightarrow Q_i$, $q_{i,0} = \{q | \delta(q_o, s) =$

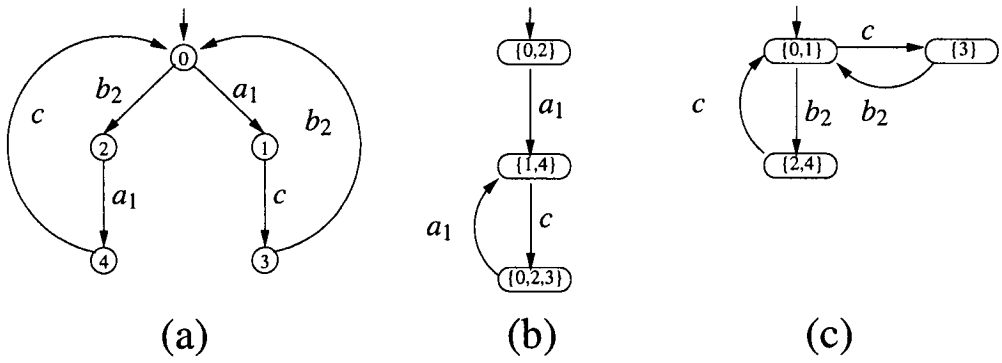


Figure 2.3: The observer automata of a DES: (a) the plant G; (b) $Obs_1(G)$; (c) $Obs_2(G)$ (adopted from [1]).

$q, s \in (\Sigma - \Sigma_{i,o})^*\}$. If $\sigma \in \Sigma_{i,o}$, then $\delta_i(\hat{q}, \sigma) = \{q | q \in Q \wedge q' \in \hat{q} \wedge \delta(q', \sigma u) = q \wedge u \in (\Sigma - \Sigma_{i,o})^*\}$.

Figure 2.3(a) shows a plant G, with $\Sigma_1 = \Sigma_{1,o} = \{a_1, c\}$ and $\Sigma_2 = \Sigma_{2,o} = \{b_2, c\}$. The observer automata for each supervisor are shown in Figure 2.3(b) and (c).

Take supervisor 1 for example. Since it cannot see b_2, it does not know whether the plant is in state 0 or state 2; therefore, the initial state of its observer automaton is {0,2}. Once a_1 occurs, supervisor 1 does not know whether the state transition is from state 0 to state 1 or from state 2 to state 4; therefore, the event a_1 leads the observer automaton to take a state transition from {0,2} to {1,4}. Similarly, when c occurs, supervisor 1 does not know whether the state transition is from state 4 to state 0 or from state 1 to state 3. Furthermore, it cannot observe b_2, i.e., it does not know whether b_2 has happened and took the plant from state 0 to state 2. Thus, the event c leads supervisor 1's observer automaton to take a state transition from {1,4} to {0,2,3}.

2.4.3 A Monitoring Automaton

A monitoring automaton [1] combines the evolution of the plant with a set of observer automata such that it tracks the current state of the plant as well as the current state of each observer. Formally, a monitoring automaton is defined as: $A = (Q^A, \Sigma, \delta^A, q_0^A)$, where $Q^A \subseteq Q \times Q_1 \times Q_2$ with Q denotes the current state of the plant and Q_1 (Q_2) denotes the current state of observer 1 (2). The initial state is $q_0^A = (q_0, q_{1,0}, q_{2,0})$. For $q \in Q, q_1 \in Q_1, q_2 \in Q_2$, the transition function δ^A is defined as:

$$\delta^A(\sigma,(q,q_1,q_2)) = \begin{cases} (\delta(q),q_1,q_2), & \text{if } \sigma \notin \Sigma_{1,o}, \sigma \notin \Sigma_{2,o}; \\ (\delta(q),\delta_1(q_1),q_2), & \text{if } \sigma \in \Sigma_{1,o}, \sigma \notin \Sigma_{2,o}; \\ (\delta(q),q_1,\delta_2(q_2)), & \text{if } \sigma \notin \Sigma_{1,o}, \sigma \in \Sigma_{2,o}; \\ (\delta(q),\delta_1(q_1),\delta_2(q_2)), & \text{if } \sigma \in \Sigma_{1,o}, \sigma \in \Sigma_{2,o}; \\ \text{undefined}, & \text{if } \delta(q) \text{ undefined.} \end{cases}$$

As an example, the monitoring automaton for G, $Obs_1(G)$ and $Obs_2(G)$ in Figure 2.3 are shown in Figure 2.4.

We start from the initial state. The initial state q_0^A is (0,{0,2},{0,1}) in this example. We interpret this to mean that initially the plant is in state 0, but observer 1 is not sure whether it is in state 0 or state 2 and observer 2 is not sure whether it is in state 0 or state 1. When a_1 occurs, the plant goes to state 1 and the observer automaton $Obs_1(G)$ goes to {1,4}. Since $a_1 \notin \Sigma_{2,o}$, $Obs_2(G)$ stays in {0,1}.

For the purpose of this thesis, we will consider the observers to be our decentralized supervisors.

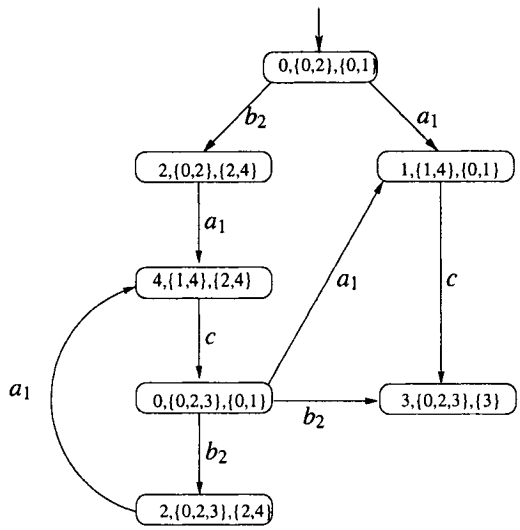


Figure 2.4: The monitoring automata for G, $Obs_1(G)$ and $Obs_2(G)$ in Figure 2.3.

2.5 I/O Automata

In this section we will provide the formal definition of I/O automaton, and describe how I/O automata can be composed, which will necessitate a special means of ensuring the automata are properly connected.

2.5.1 Overview

Generally speaking, instead of simply computing some functions of their inputs and then halting, RDES continuously receive inputs from and interact with an unpredictable environment. Therefore, following [12], in this thesis we use *Input/Output (I/O) automata* [5] to model RDES. Interactions between RDES and the environment are modeled by input events (inputs) and output events (outputs), and the internal behavior of RDES is modeled by internal events. An RDES generates outputs and internal events autonomously and instantaneously, but only transmits outputs instantaneously to its environment and other I/O automata. In contrast, the inputs for an RDES are generated by its environment and other I/O automata and transmitted

instantaneously to the RDES.

Furthermore, it is important to clarify the distinction between actions whose performance is under the control of the system and actions whose performance is controlled by the environment. That is, an I/O automaton can only establish restrictions on its own performance, namely output actions or internal actions, but it is unable to block the performance of the environment, namely its input actions.

For decentralized RDES, each system component, as well as each local supervisor, is modeled by an I/O automaton. Since a set of I/O automata can be composed to yield another I/O automaton, we can compose all system components to get the complete system behavior.

We define an I/O automaton as follows.

Definition 2.5.1 *An I/O automaton A is a quadruple* $(Q^A, \Sigma^A, \delta^A, q_0^A)$ *where*

- Q^A *is the set of states,*
- Σ^A *is the alphabet of events, which is partitioned into* $\Sigma_?^A$, $\Sigma_!^A$ *and* Σ_I^A *of inputs, outputs and internal events, respectively,*
- δ^A: $Q^A \times \Sigma^A \rightarrow Q^A$ *is the (partial) state transition function, and*
- $q_0^A \in Q^A$ *is the initial state.*

The primary difference between I/O automata and an automata defined in Section 2.2 is the partition of its alphabet. As Figure 2.5 shows, inputs and outputs model the interaction between the system and its environment. The set of *external* events is defined as $\Sigma_{ex} = \Sigma_! \cup \Sigma_?$, namely the union of inputs and outputs, and the set of *locally-controlled* events is defined as $\Sigma_{loc} \subseteq \Sigma_! \cup \Sigma_I$, which is a subset of the union of outputs and internal events.

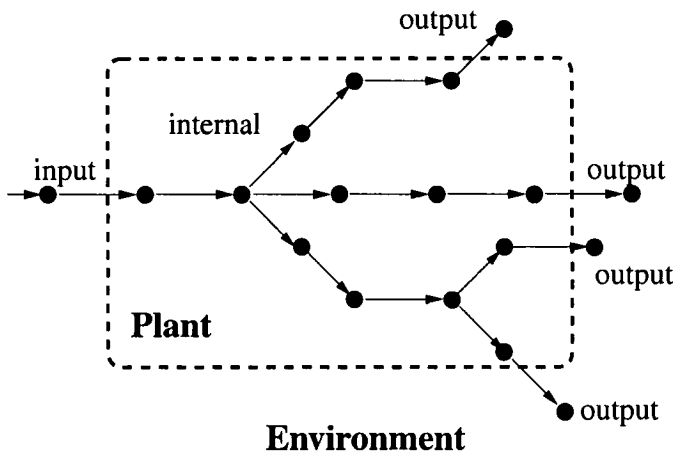


Figure 2.5: I/O automaton: an explicit view.

As defined in [5], one of the fundamental characteristics of an I/O automaton is based on who determines when the action is performed. An I/O automaton can only establish restrictions on when it will perform an output or an internal event, but it is unable to block the occurrence of an input event. These characteristics can be interpreted in the Ramadge-Wonham supervisory control framework as follows: inputs cannot be disabled, i.e., inputs are uncontrollable.

Aside from the uncontrollability of inputs, another fundamental assumption about an I/O automaton is that the performance of an action is controlled by at most one system component [5], namely, each event is controllable by at most one supervisor.

Finally, the difference between outputs and internal events is that outputs are sent to the environment by the plant, i.e., outputs are observable by the environment, while internal events are not.

2.5.2 Internal Connection

We can construct an I/O automaton modeling a complex system by composing a set of I/O automata modeling simpler system components. In particular, we impose certain restrictions

on the composition of I/O automata. Since internal events of an I/O automaton A are intended to be unobservable by another I/O automaton B, we can only allow A composed to B when the internal events of A are disjoint from the internal events of B. Otherwise, an internal event of A may trigger an unwanted state transition of B. Furthermore, in keeping with our philosophy that each event is controllable by at most one supervisor, we cannot allow A and B to be composed unless the outputs of A and B form disjoint sets, or else an output might be controlled by both the supervisor for A and the supervisor for B. Formally, we define this special category of composition of I/O automata as follows.

Definition 2.5.2 *A pair of I/O automata* $A^1 = (Q^1, \Sigma^1, \delta^1, q_0^1) and A^2 = (Q^2, \Sigma^2, \delta^2, q_0^2)$ *are* ***compatible*** *[5] if*

(1) $\Sigma_I^1 \cap \Sigma_I^2 = \emptyset$, *and*

(2) $\Sigma_!^1 \cap \Sigma_!^2 = \emptyset$.

In other words, a pair of I/O automata are compatible if both their output event sets and their internal event sets are disjoint. From this point on, we focus on the composition of two compatible I/O automata only. Note that this result can be extended to any countable collection of I/O automata when they are pairwise compatible.

For a pair of I/O automata $A^1 = (Q^1, \Sigma^1, \delta^1, q_0^1)$ and $A^2 = (Q^2, \Sigma^2, \delta^2, q_0^2)$, the essence of their composition is straightforward: when composing two compatible I/O automata, an output $\pi_!$ of one I/O automaton may be identified with the input $\pi_?$ of the other. For all $\pi_! \in \Sigma_!^1$, if there exists $\pi_? \in \Sigma_?^2$, we call the pair $(\pi_!, \pi_?)$ an *internal connection* from A^1 to A^2, denoted by π_{12}. Similarly, we can also define the internal connection π_{21} in the opposite direction. When the direction of the internal connection is unknown or unimportant, it is denoted simply by π with no subscript. Existence of internal connections provides channels for RDES to synchronize activities and co-operate together. We denote the set of all internal connections by Υ.

Suppose we have an internal connection π_{12}. We consider it to be an internal communication channel where A^1 is the sender and A^2 is the receiver. Then the sets of *connection outputs* and *connection inputs* are defined as

$$\Upsilon_! = \{\sigma_! \in \Sigma_!^1 \cup \Sigma_!^2 | \ \exists \sigma_? \in \Sigma_?^1 \cup \Sigma_?^2, \ (\sigma_!, \sigma_?) \in \Upsilon\}$$

and

$$\Upsilon_? = \{\sigma_? \in \Sigma_?^1 \cup \Sigma_?^2 | \exists \sigma_! \in \Sigma_!^1 \cup \Sigma_!^2, \ (\sigma_!, \sigma_?) \in \Upsilon\}.$$

2.5.3 Synchronous Product

Since the internal connections do not play a role in the interaction between the system and its environment, they are naturally defined to be internal events after composition.

The general principle of composing two compatible I/O automata is that internal events of the components remain internal events of the composition, input events remain input events, and output events remain output events. Internal connections become internal events of the composition and they should be removed from the set of inputs and the set of outputs of the composition.

The formal definition of synchronous product of two compatible I/O automata is given as follows.

Definition 2.5.3 *The **synchronous product** (or composition) $A^1 \| A^2$ of two compatible I/O automata $A^1 = (Q^1, \Sigma_?^1 \cup \Sigma_!^1 \cup \Sigma_I^1, \delta^1, q_0^1)$ and $A^2 = (Q^2, \Sigma_?^2 \cup \Sigma_!^2 \cup \Sigma_I^2, \delta^2, q_0^2)$ is an I/O automaton $(Q, \Sigma_? \cup \Sigma_! \cup \Sigma_I, \delta, q_0)$, where*

- $\Sigma_? = \Sigma_?^1 \cup \Sigma_?^2 - \Upsilon_?$,
- $\Sigma_! = \Sigma_!^1 \cup \Sigma_!^2 - \Upsilon_!$,
- $\Sigma_I = \Sigma_I^1 \cup \Sigma_I^2 \cup \Upsilon$,

- $Q = Q^1 \times Q^2$,

- $\forall q = (q^1, q^2) \in Q$ *and* $\sigma \in \Sigma$:

$$\delta((q^1,q^2),\sigma) = \begin{cases} (\delta^1(q^1,\sigma_!),\delta^2(q^2,\sigma_?)), \\ \quad \textit{if } \sigma := (\sigma_!,\sigma_?) \in \Upsilon \wedge \delta^1(q^1,\sigma_!) \textit{ and } \delta^2(q^2,\sigma_?) \textit{ are defined}; \\ (\delta^1(q^1,\sigma_?),\delta^2(q^2,\sigma_!)), \\ \quad \textit{if } \sigma := (\sigma_!,\sigma_?) \in \Upsilon \wedge \delta^1(q^1,\sigma_?) \textit{ and } \delta^2(q^2,\sigma_!) \textit{ are defined}; \\ (\delta^1(q^1,\sigma),\delta^2(q^2,\sigma)), \\ \quad \textit{if } \delta^1(q^1,\sigma) \textit{ is defined } \wedge \delta^2(q^2,\sigma) \textit{ is defined } \wedge \sigma \in \Sigma - \Upsilon; \\ (\delta^1(q^1,\sigma),q^2), \\ \quad \textit{if } \sigma \in \Sigma - \Upsilon \wedge \delta^1(q^1,\sigma) \textit{ is defined } \wedge \sigma \notin \Sigma^2; \\ (q^1,\delta^2(q^2,\sigma)), \\ \quad \textit{if } \sigma \in \Sigma - \Upsilon \wedge \sigma \notin \Sigma^1 \wedge \delta^2(q^2,\sigma) \textit{ is defined}; \\ \textit{undefined}, \quad \textit{otherwise} \end{cases}.$$

- $q_0 = (q_0^1, q_0^2)$

Next we present an example to illustrate how to compose compatible I/O automata.

Example 2.1: Synchronous product of compatible I/O automata

Let $\Sigma_!^A = \{c_!, d_!\}$, $\Sigma_?^A = \{a_?, b_?\}$, $\Sigma_I^A = \{e_I, f_I\}$, $\Sigma_!^B = \{b_!, \beta_!\}$, $\Sigma_?^B = \{c_?, \alpha_?\}$, and $\Sigma_I^B = \{\gamma_I, \sigma_I\}$. The pair $(c_!, c_?)$ forms an internal connection $c_{A,B}$ from A to B. Furthermore, the pair $(b_!, b_?)$ forms an internal connection $b_{B,A}$ from B to A.

As we can see, all internal connections are defined to be internal, so they are removed from the set of inputs and the set of outputs and added to the set of internal events. Therefore, the alphabets after composition are $\Sigma_? = \{a_?, \alpha_?\}$, $\Sigma_! = \{\beta_!, d_!\}$ and $\Sigma_I = \{c_{A,B}, b_{B,A}, e_I, f_I, \gamma_I, \sigma_I\}$.

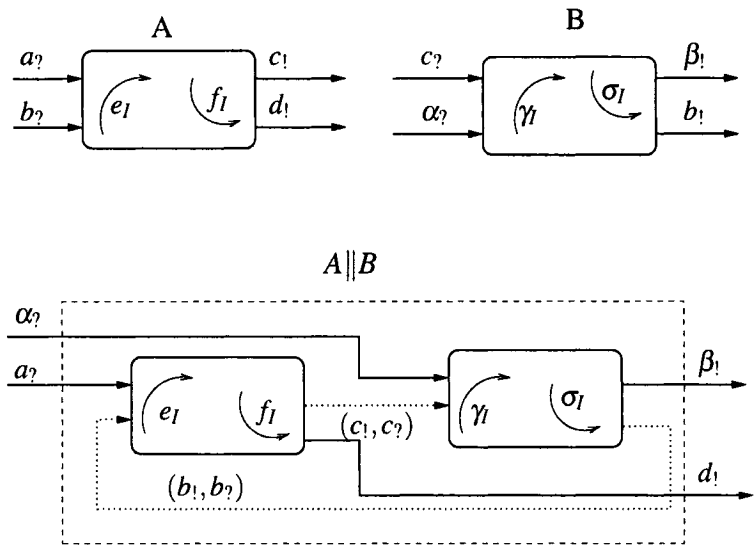


Figure 2.6: Composition of compatible I/O automata.

To summarize, we list several main differences between our model and the standard I/O automata framework introduced in [5]:

- In [5], the set of locally-controlled events, or simply local events, is defined as $\Sigma_{loc} = \Sigma_! \cup \Sigma_I$, i.e., all local outputs and internal events are assumed to be controllable, while we define $\Sigma_{loc} \subseteq \Sigma_! \cup \Sigma_I$, which indicates that some of the outputs and internal events may be uncontrollable.

- In [5], an I/O automaton A is defined with an equivalence relation $part(A)$ partitioning the set of local events into at most a countable number of equivalent classes, while we dropped this element because $part(A)$ is used in [5] only in the definition of *fair computation*, which is not our concern.

- In [5], for every state q and every input σ, there always exists a state $q' \in Q$ such that $q' = \delta(q, \sigma)$, while we define the state transition function δ to be a partial function, namely there exists a state $q' \in Q$ such that $q' = \delta(q, \sigma)$ only if σ is defined at q.

- In [5], when composing two I/O automata A and B, if an input $\pi_?$ of A is identified with an output $\pi_!$ of B, they simply remove this input $\pi_?$ from the set of inputs after composition with no other actions, i.e., $\Sigma_I^{A\|B} = \Sigma_I^A \cup \Sigma_I^A, \Sigma_!^{A\|B} = \Sigma_!^A \cup \Sigma_!^A, \Sigma_?^{A\|B} = \Sigma_?^A \cup \Sigma_?^A - \Sigma_!^{A\|B}$. In contrast, we define this identification to be an internal connection and treat all internal connections as internal events after composition, as described in Definition 2.5.3.

2.6 Conclusion

In this chapter, we first provided a brief introduction of some basic concepts from formal languages, automata theory, and supervisory control theory. The fundamental elements of centralized and decentralized supervisory control theory such, as sufficient and necessary conditions for the existence of solutions to those problems, were also reviewed. After formally defining I/O automata, we explained how I/O automata can model control of RDES after making a few minor modifications to the original model. We defined what internal connections are and showed how internal connections work by an example. Finally, we explained the general principle of composing compatible I/O automata and presented a formal definition.

Chapter 3

Decentralized RDES: System Model and Specification

In this chapter we proceed as follows. Section 3.1 provides a brief overview of this chapter. In Section 3.2, we present our decentralized RDES architecture. Safety, progress and service specifications are introduced in Section 3.3. We then define the corresponding supervisory control problem in Section 3.4. Section 3.5 summarizes this chapter.

3.1 Introduction

For RDES, we consider a special category of specifications which are partial specifications, i.e., they are defined on a subset of the entire alphabet consisting only of external events. In this chapter, we use the *supervisory control* approach introduced by Ramadge and Wonham [19], [24] to ensure that the *service specification* of decentralized RDES is met. Recall that the supervisory control approach requires the design of a supervisor to oversee the system and issue control commands to keep the behavior of the controlled system within a required region. Such a supervisor enables or disables events to prevent the system from performing illegal moves.

Given a plant G and its corresponding supervisor S, the closed-loop system, denoted by S/G, *implements* a service specification E if certain properties, called *safety* and *progress*, hold.

Safety property requires that "bad things never happen" in any state of the system. Safety of reactive systems is of paramount importance in system design, since any violation of safety guidelines could result in loss of life or excessive economic damage. For instance, an ineffective control in an on-line banking system or an unsupervised on-line financial transaction could result in the loss of large sums of money.

Progress requires that the occurrence of an external event should not be blocked as long as it is allowed by the specification. For instance, if the specification requires sending a certain output in response to some input, but in *some* sequences leading from the input to the output an internal event is disabled by the controller, then the progress property is violated.

Given a plant G, we impose the following assumptions on the service specification E throughout this thesis:

A.1 : E constrains only inputs and outputs (i.e., external events).

A.2 : E constrains only actions G can physically perform.

3.2 Decentralized RDES Architecture

When composing decentralized RDES, each system component is modeled by an I/O automaton. Without loss of generality, we assume that the plant G consists only of two components G_1 and G_2; it is not difficult to extend the results to any finite number of components.

We partition the input set $\Sigma_?^i$ of G_i $(i = 1, 2)$ into three disjoint alphabets:

- $\Sigma_{?,\updownarrow}^i$, events that can only be connected to outputs from the environment (thus unobservable to the other component),

- $\Sigma^i_{?,\leftrightarrow}$, events that can only be connected to outputs of the other component (thus unobservable to the environment), and

- $\Sigma^i_{?,\circlearrowleft}$, events that can be connected and thus are observable to both the environment and the other component.

Similarly, the output set $\Sigma^i_!$ can be partitioned into $\Sigma^i_{!,\updownarrow}$, $\Sigma^i_{!,\leftrightarrow}$ and $\Sigma^i_{!,\circlearrowleft}$, representing outputs that can be connected (and are thus observable) to the environment, the other component, and both the environment and the other component, respectively.

In accordance with many practical situations, we assume that $\Sigma^i_{?,\circlearrowleft} = \emptyset$, i.e., each input is exclusively used to either connect a component to its environment or to the other component but not both. Similarly, we assume in this chapter that $\Sigma^i_{!,\circlearrowleft} = \emptyset$. The decentralized RDES system architecture is shown in Figure 3.2.

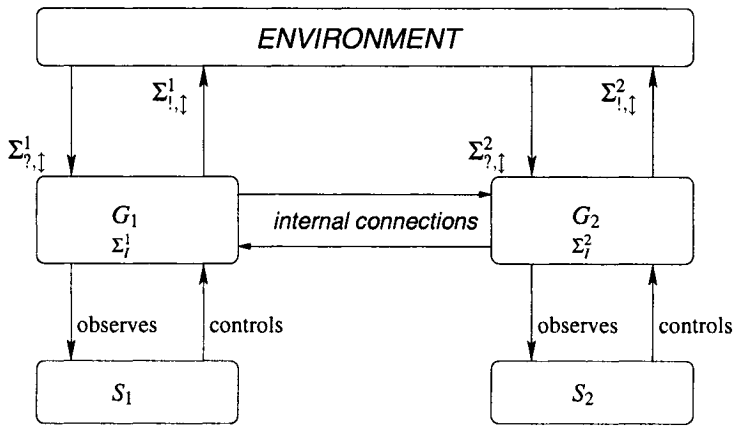


Figure 3.1: Decentralized RDES architecture.

Since each decentralized supervisor can completely observe its local alphabet, the controllable and observable event sets of G_1, G_2, and G are $\Sigma^1_c \subseteq \Sigma^1_! \cup \Sigma^1_l$ and $\Sigma^1_o = \Sigma^1 \cup \Sigma^2_{!,\leftrightarrow}$, $\Sigma^2_c \subseteq \Sigma^2_! \cup \Sigma^2_l$ and $\Sigma^2_o = \Sigma^2 \cup \Sigma^1_{!,\leftrightarrow}$, $\Sigma_c = \Sigma^1_c \cup \Sigma^2_c$ and $\Sigma_o = \Sigma^1_o \cup \Sigma^2_o$, respectively.

Furthermore, since the pair (G_1, G_2) is compatible, no event is controllable by both components, i.e., $\Sigma_c^1 \cap \Sigma_c^2 = \emptyset$. Thus, checking co-observability can be reduced to multiple checks of observability [25].

3.3 Safety and Progress

In this thesis we are interested in solving a slightly different supervisory control problem compared with the standard problems in R-W supervisory control theory, where the objective is to obtain a supervisor so that the closed-loop system implements a given service specification defined on the subset of external events. Therefore, in what follows, we will formally define what we mean by safety, progress, and implementation of a service specification.

Let $P_{ex} : \Sigma^* \rightarrow \Sigma_{ex}^*$ be a natural projection, which can be interpreted as the environment's view of the plant's behavior. We define service specification and safety as follows.

Definition 3.3.1 *Given a plant G with its generated language $L(G)$, a language $E \subseteq \Sigma_{ex}^*$, defined on the external alphabet, is called a* ***service specification*** *if $E \subseteq P_{ex}(L(G))$.*

Service specifications play an important role in communication systems; see for instance in [26] and [27]. For example, in the protocol verification problem, the desired functionality are specific to some external behavior.

Definition 3.3.2 *Given a language $K \subseteq L$, K satisfies* ***safety*** *with respect to a service specification $E \subseteq P_{ex}(L)$ if $P_{ex}(K) = E$.*

In other words, safety captures the notion that nothing illegal should happen, i.e., the event sequences generated by the system should correspond to those allowed by E.

Progress captures the notion that the supervised system should not block the occurrence of an external event whenever it is allowed by the specification E. The following definition of progress appears in [18]:

Definition 3.3.3 [18] *Given $K \subseteq L$, K is said to satisfy* ***progress*** *with respect to a service specification $E \subseteq P_{ex}(L)$ if*

$$\forall s \in K, \sigma \in \Sigma_{ex}, P_{ex}(s)\sigma \in E \Rightarrow \exists u \in (\Sigma - \Sigma_{ex})^*, su\sigma \in K. \tag{3.1}$$

Equation 3.1 states that if there is a sequence s and an external event σ such that $P_{ex}(s)\sigma$ is in the service specification E, there must also be (at least one) internal path u that connects s to σ.

For our purpose, this definition is not good enough because the plant behavior and the service specification are defined to be subsets of Σ^* and Σ_{ex}^*, respectively, and it is difficult to check the existence of a string in $(\Sigma - \Sigma_{ex})^*$. Therefore, we define progress as follows, where the focus is on straightforward checking of the existence of strings in Σ^*.

Definition 3.3.4 *Given a language $K \subseteq L$, K satisfies* ***progress*** *with respect to a service specification $E \subseteq P_{ex}(L)$ if*

$$\forall s \in K, t \in E, P_{ex}(s) \leq t \Rightarrow \exists u \in \Sigma^*, su \in K \wedge P_{ex}(su) = t. \tag{3.2}$$

Equation 3.2 interprets the requirement from a different point of view. In this case, if there is a sequence $s \in K$ whose projection $P_{ex}(s)$ is a prefix of a string t of E, then it should be possible to extend s by another sequence u such that their catenation equals t under the projection, i.e., $P_{ex}(su) = t$.

Assuming that safety is guaranteed, i.e., $P_{ex}(K) = E$, progress can be stated by the commutative diagram of Figure 3.2.

The diagram shows that for every $s \in K$, if there exists $t \in E$ such that $P_{ex}(s) \leq t$ (i.e., $t = cat(P_{ex}(s), t')$ for some $t' \in \Sigma_{ex}^*$), then it should be possible to concatenate s with *some* $u \in \Sigma^*$ such that $P_{ex}(cat(s,u)) = t$. In other words, concatenation and P_{ex} commute:

$$P_{ex}(cat(s,u)) = cat(P_{ex}(s), t')$$

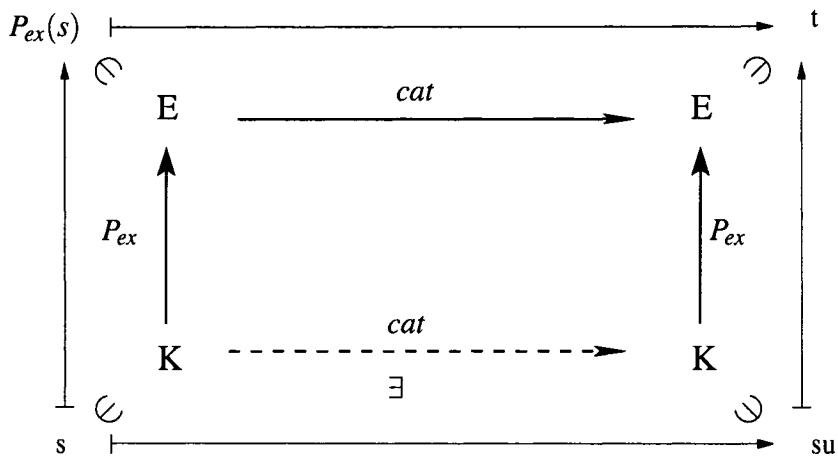


Figure 3.2: Interpretation of progress, where *cat* denotes the operation of catenation of strings.

The following proposition shows that Definitions 6 and 7 are logically equivalent.

Proposition 3.3.1 *Given a language $K \subseteq L$, K satisfies Equation 3.1 if and only if it satisfies Equation 3.2.*

Proof: (if) Suppose $K \subseteq L(G)$ satisfies Equation 3.2, then

$\forall s \in K, \sigma \in \Sigma_{ex}, P_{ex}(s)\sigma \in E$

$\Rightarrow s \in K, P_{ex}(s)\sigma \in E, P_{ex}(s) \leq P_{ex}(s)\sigma$

$\Rightarrow \exists u \in \Sigma^*, su \in K \wedge P_{ex}(su) = P_{ex}(s)\sigma$ (by Equation 3.2)

$\Rightarrow su \in K \wedge P_{ex}(u) = \sigma$

$\Rightarrow \exists u_1, u_2 \in (\Sigma - \Sigma_{ex})^*$ s.t. $u = u_1 \sigma u_2$

$\Rightarrow su_1 \sigma u_2 \in K$

$\Rightarrow su_1 \sigma \in K$ (since K is prefix-closed)

$\Rightarrow K$ satisfies Equation 3.1

(only if) Suppose $K \subseteq L(G)$ satisfies Equation 3.1, then

$\forall s \in K, t \in E, P_{ex}(s) \leq t$

$\Rightarrow \exists v = \sigma_1 \sigma_2 \ldots \sigma_n \in \Sigma_{ex}^*$, s.t. $P_{ex}(s)v = t$

$\Rightarrow s \in K, \sigma_1 \in \Sigma_{ex}, P_{ex}(s)\sigma_1 \in E$

$\Rightarrow \exists u_1 \in (\Sigma - \Sigma_{ex})^*, su_1\sigma_1 \in K$

$\Rightarrow su_1\sigma_1 \in K, \sigma_2 \in \Sigma_{ex}, P_{ex}(su_1\sigma_1)\sigma_2 \in E$

$\Rightarrow \exists u_2 \in (\Sigma - \Sigma_{ex})^*, su_1\sigma_1u_2\sigma_2 \in K$

$\Rightarrow \ldots$

$\Rightarrow su_1\sigma_1 \ldots u_{n-1}\sigma_{n-1} \in K, \sigma_n \in \Sigma_{ex}, P_{ex}(su_1\sigma_1 \ldots u_{n-1}\sigma_{n-1})\sigma_n \in E$

$\Rightarrow \exists u_n \in (\Sigma - \Sigma_{ex})^*, su_1\sigma_1 \ldots u_{n-1}\sigma_{n-1}u_n\sigma_n \in K$

$\Rightarrow \exists u = u_1\sigma_1 \ldots u_{n-1}\sigma_{n-1}u_n\sigma_n \in \Sigma^*$, s.t. $su \in K \wedge P_{ex}(su) = t$.

$\Rightarrow K$ satisfies Equation 3.2 ■

To summarize, safety requires that each generated sequence of a language K should correspond to a prefix of the specification E, i.e., no illegal sequences should occur in K. Since E is a partial specification, there may exist more than one sequence of K that corresponds to the same prefix of E. Progress requires that if an external event is possible after such a prefix of E, then it should also "eventually" be possible in *all* corresponding sequences of K, i.e., after the occurrence of zero or more internal events following each corresponding trace of K (which is, in fact, another interpretation of progress).

Remark: Note that safety only guarantees that such an external event is eventually possible following at *least one* of the corresponding traces of K.

A supervised system is said to implement a service specification if safety and progress hold. We formally define this statement in the next definition.

Definition 3.3.5 *Given a language $K \subseteq L$ and a service specification $E \subseteq P_{ex}(L)$, K is said to* ***implement*** *E if K satisfies both safety and progress with respect to E, that is:*

- $P_{ex}(K) = E$, *and*

- $\forall s \in K, t \in E, P_{ex}(s) \leq t \Rightarrow \exists u \in \Sigma^*, su \in K \wedge P_{ex}(su) = t$.

Furthermore, given a language K, we claim that the set of its sublanguages that satisfy both safety and progress is closed under arbitrary union.

Lemma 3.3.1 *Given a language $K \subseteq L$ and a service specification $E \subseteq P_{ex}(L)$, let $\mathscr{S} = \{K | K \subseteq L(G)$ and K implements $E\}$, then $\mathscr{S}$ is an upper semi-lattice (with respect to set union).*

Proof: Given an arbitrary index set I such that $K_i \in \mathscr{S}$ for $i \in I$, let $K := \bigcup_{i \in I} K_i$.

(1). $P_{ex}(K) = P_{ex}(\bigcup_{i \in I} K_i) = \bigcup_{i \in I} P_{ex}(K_i) = \bigcup_{i \in I} E = E$

$\Rightarrow K$ satisfies safety.

(2). Let $s \in K$ and $t \in E$ such that $P_{ex}(s) \leq t$.

$\Rightarrow \exists j \in I, s \in K_j$

$\Rightarrow \exists u \in \Sigma^*, su \in K_j \wedge P_{ex}(su) = t$ (since K_j satisfies progress)

$\Rightarrow \exists u \in \Sigma^*, su \in K \wedge P_{ex}(su) = t$ (since $K_j \subseteq K$)

$\Rightarrow K$ satisfies progress.

From (1) and (2) we can see that $K := \bigcup_{i \in I} K_i$ implements E if every $K_i, i \in I$, does, i.e., $K \in \mathscr{S}$. As we will see in the following counterexample, neither safety nor progress is closed under intersection. Therefore $\mathscr{S}$ is an upper semi-lattice and its supremal element exists in $\mathscr{S}$. ■

Example 3.1: Safety and progress are not closed under intersection.

Given K_1, K_2 and E as below, we can see that $P_{ex}(K_1) = P_{ex}(K_1) = E$, i.e., both K_1 and K_2 satisfy safety. However, $K_1 \cap K_2 = \overline{\{\alpha_!\}}$ and $P_{ex}(K_1 \cap K_2) \neq E$, i.e., $K_1 \cap K_2$ does not satisfy safety.

Tables 3.1 and 3.2 show that both K_1 and K_2 satisfy progress, but Table 3.3 shows that $K_1 \cap K_2$ does not. □

We summarize an existing result that will be important as we explore the role of controllability and observability properties in RDES.

Table 3.1: K_1 satisfies progress

$s \in K_1$	$t \in E$	$\exists u$	su	$su \in K_1$?	$P_{ex}(su) = t$?
ε	$\alpha_!$	$\alpha_!$	$\alpha_!$	*Yes*	*Yes*
ε	$\alpha_!\gamma_!$	$\alpha_!\sigma_I\gamma_!$	$\alpha_!\sigma_I\gamma_!$	*Yes*	*Yes*
$\alpha_!$	$\alpha_!\gamma_!$	$\sigma_I\gamma_!$	$\alpha_!\sigma_I\gamma_!$	*Yes*	*Yes*
$\alpha_!\sigma_I$	$\alpha_!\gamma_!$	$\gamma_!$	$\alpha_!\sigma_I\gamma_!$	*Yes*	*Yes*

Table 3.2: K_2 satisfies progress

$s \in K_2$	$t \in E$	$\exists u$	su	$su \in K_2$?	$P_{ex}(su) = t$?
ε	$\alpha_!$	$\alpha_!$	$\alpha_!$	*Yes*	*Yes*
ε	$\alpha_!\gamma_!$	$\alpha_!\pi_I I\gamma_!$	$\alpha_!\pi_I\gamma_!$	*Yes*	*Yes*
$\alpha_!$	$\alpha_!\gamma_!$	$\pi_I\gamma_!$	$\alpha_!\pi_I\gamma_!$	*Yes*	*Yes*
$\alpha_!\pi_I$	$\alpha_!\gamma_!$	$\gamma_!$	$\alpha_!\pi_I\gamma_!$	*Yes*	*Yes*

Table 3.3: $K_1 \cap K_2$ does not satisfy progress

$s \in K_1 \cap K_2$	$t \in E$	$P_{ex}(s) \leq t$?	$\exists u$	su	$su \in K_1 \cap K_2$?	$P_{ex}(su) = t$?
$\alpha_!$	$\alpha_!\gamma_!$	*Yes*	*No*	–	–	–

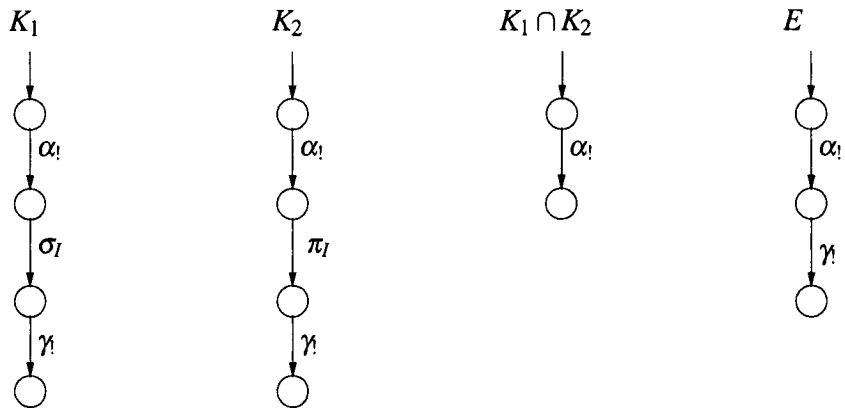


Figure 3.3: K_1 and K_2 satisfy safety but $K_1 \cap K_2$ does not.

Lemma 3.3.2 [18] *Given a language L and a service specification $E \subseteq P_{ex}(L)$, let $\mathscr{S} := \{K | K \subseteq L$, K is controllable and observable, and K implements $E\}$, then $\mathscr{S}$ is an upper semi-lattice (with respect to set union).*

In other words, we can find a supremal controllable sublanguage of $L(G)$ which implements the specification E (with full observation), or a supremal controllable and observable sublanguage of $L(G)$ which implements the specification E (under partial observation). This result will be important in the next section when we discuss sufficient and necessary conditions for the existence of a solution to the supervisory control problem for decentralized RDES.

3.4 Problem Definition

As noted earlier, in this thesis we are interested in solving a slightly different supervisory control problem from the standard supervisory control problems in the Ramadge-Wonham framework. The key difference here is that the control task is to design a supervisor so that the controlled plant implements a given service specification that is defined only on a subset of external events.

Formally, the decentralized supervisory control problem for RDES that we want to address is defined as follows. To simplify the problem, we restrict our analysis to a decentralized plant consisting of only two components.

Problem 3.4.1 *Given the plant G composed of G_1 and G_2, the service specification E, and the sets of observable and controllable events of G_1 and G_2 as Σ_o^1, Σ_c^1, Σ_o^2, $\Sigma_c^2 \subseteq \Sigma$, respectively, construct local supervisors S_1 and S_2 such that $S = \widetilde{S_1} \wedge \widetilde{S_2}$ is a supervisor for G and S/G implements E.*

We call such an S a *solution* to the decentralized supervisory control problem for RDES. Next we will present sufficient and necessary conditions for the existence of a solution to Problem 3.4.1.

3.5 Conclusion

In this chapter, we first presented our decentralized RDES model. We then studied safety, progress and service specifications in the decentralized RDES framework. In general, a service specification is a partial specification (i.e., defined only on the subset of the external events) where safety and progress hold. At last, we defined the corresponding decentralized supervisory control problem.

Chapter 4

Decentralized Supervisory Control of RDES

In this chapter we proceed as follows. Section 4.1 provides a brief overview of this chapter. Section 4.2 shows by examples that Ramadge-Wonham supervisory control theory cannot be directly applied to the problem of controlling RDES, and substitute sufficient and necessary conditions for the existence of a solution to this control problem are discussed in Section 4.3. In some research domains, only safety is of concern; we will define the corresponding decentralized supervisory control problem, and present easier-to-check sufficient and necessary conditions for the existence of its solution in Section 4.4. Section 4.5 summarizes this chapter.

4.1 Introduction

Decentralized supervisory control of DES in the Ramadge-Wonham framework has been well-studied in [22] and [21]. Unfortunately, these results cannot be directly applied to RDES. Some of the system properties that arise from partitioning events into inputs, outputs and internal events does not satisfy the standard conditions for decentralized control. Therefore, we present appropriate extensions in this chapter.

4.2 R-W Supervisory Control Theory Revisited

Generally speaking, since $L(G)$ and E are defined over different alphabets, the standard definitions of controllability and observability in the Ramadge-Wonham framework (where the plant and the specification are usually defined over the same alphabet) need minor adjustments before they can be applied to our framework.

From the external behavior's point of view, the plant acts as a black box, with only inputs and outputs concerned. Under the standard definitions, controllability and observability of the specification provide sufficient (but no longer necessary) conditions for the existence of a solution to the supervisory control problem (under full or partial observation), as suggested by the following examples.

Example 4.1: Controllability of E with respect to $P_{ex}(L(G))$ is not necessary for the existence of a solution under full observation.

In this example we assume that the plant consists of a single component, and the supervisor has full observation of the plant events.

As Figure 4.1 shows, let $\Sigma = \{\alpha_!, \pi_I, \gamma_!, \beta_?\}$ and $\Sigma_c = \{\alpha_!, \pi_I, \gamma_!\}$, E is not controllable with respect to $P_{ex}(L(G))$ since $\alpha_! \in E$ and $\alpha_!\beta_? \in P_{ex}(L(G)) - E$, i.e., it is possible to exit the specification at $\alpha_! \in E$ through a feasible but uncontrollable transition labeled with $\beta_?$.

However, a supervisor that disables both π_I implements the specification. As we can see in Figure 4.1, disabling both π_I prevents $\beta_?$ and $\gamma_!$ from happening. The only possible nonempty string then is $\alpha_!$, which coincides with E. Furthermore, it is not difficult to check that the closed-loop language $\{\varepsilon, \alpha_!\}$ also satisfies progress with respect to E. □

Next we show that the standard definition of observability does not provide a necessary condition for the existence of a supervisor under partial observation for RDES.

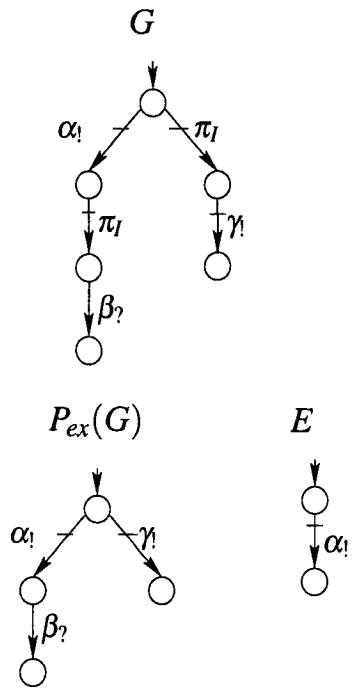


Figure 4.1: E is not controllable with respect to $P_{ex}(L(G))$, but a centralized solution to Problem 3.4.1 under full observation exists. Controllable events are graphically represented by ticks on their transition arrows.

Example 4.2: Controllability of E with respect to $P_{ex}(L(G))$ and observability of E with respect to $P_{ex}(L(G))$ and $P_{ex,o}$ is not necessary for the existence of a solution under partial observation.

In this example we assume that the supervisor has only a partial observation of the plant behavior, namely, it can observe events in Σ_o. As Figure 4.2 shows, let $\Sigma_o = \Sigma_c = \{c_!, b_I\}$ and $\Sigma_{uo} = \Sigma_{uc} = \{a_?\}$.

Note that E is not observable with respect to $L(G)$ and $P_{ex,o}$, where $P_{ex,o} : \Sigma_{ex}^* \rightarrow (\Sigma_o \cap \Sigma_{ex})^*$ is a natural projection. We can see that $P_{ex,o}(\varepsilon) = P_{ex,o}(a_?)$, $\varepsilon c_! \in E$, $a_? \in E$, $a_? c_! \in P_{ex}(L(G))$, but $a_? c_! \notin E$. This is a violation of observability.

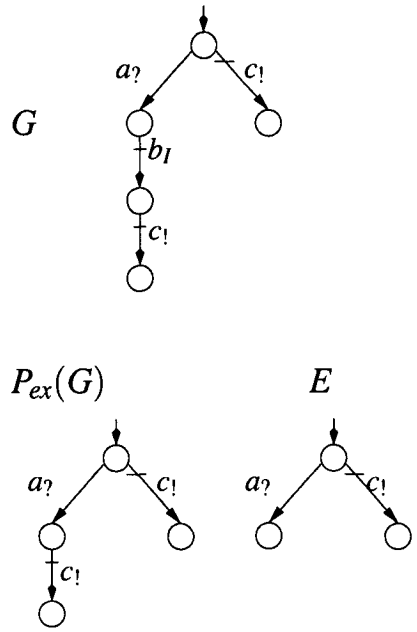


Figure 4.2: E is not observable with respect to $P_{ex}(L(G))$ and $P_{ex,o}$, but a centralized solution to Problem 3.4.1 under partial observation exists.

However, a supervisor S under partial observation implements the service specification E by disabling the event $c_!$ after observing b_I, or disabling the event b_I at all times. □

To understand why the controllability and observability of E with respect to $P_{ex}(L(G))$ is not necessary for the existence of a supervisor for an RDES, we first investigate the problem with Example 4.1. The problem there is that an uncontrollable exit ($\beta_?$) from E to $P_{ex}(L(G))$ could be prevented by disabling a leading controllable internal event (π_I). However, since the service specification E is defined on external behavior only, all internal events are removed by the projection P_{ex} and their role in making E controllable is not considered.

We can see that a similar difficulty occurs in Example 4.2, where the supervisor S under partial observation has to make the control decision correctly. From the external behavior's point of view, S cannot distinguish $a_?$ from ε (since $a_?$ is unobservable). Since the control action is to disable $c_!$ after $a_?$ and enable $c_!$ after ε, S cannot accomplish this control task

without ambiguity. However, if we go back to the complete plant behavior, i.e., both external and internal behavior, the occurrence of internal event b_I can help S to tell apart $a_?$ from ε since $P_o(a_?b_I) = b_I$ and $P_o(\varepsilon) = \varepsilon$ and there is no ambiguity any more.

4.3 Substitute Conditions

We can conclude from the above discussion that controllability and observability in the Ramadge-Wonham framework are no longer necessary for the existence of a supervisor in RDES because the role of internal events is ignored. We turn our attention to "recovering" legal internal behavior possible in $L(G)$. That is to say, we extend E from a language restricting external behavior to one restricting the entire plant behavior to get a new specification E' which is a controllable subset of $P_{ex}^{-1}(E) \cap L(G)$, namely $K \in \mathscr{C}(P_{ex}^{-1}(E) \cap L(G))$ where $\mathscr{C}(K)$ denotes the set of controllable sublanguages of K.

First we present a sufficient and necessary condition for the existence of a supervisor under full observation.

Proposition 4.3.1 *Given a plant G, a service specification $E \subseteq \Sigma_{ex}^*$, there exists a supervisor S for G such that S/G implements E if and only if there exists $K \in \mathscr{C}(P_{ex}^{-1}(E) \cap L(G))$ such that K implements E.*

Proof: (if) Suppose there exists K such that $K \in \mathscr{C}(P_{ex}^{-1}(E) \cap L(G))$ and K implements E, i.e., K is controllable and K implements E, then it follows from Theorem 2.3.1 that there exists a supervisory control S such that $L(S/G) = K$. Therefore S/G implements E.

(only if) Suppose there exists a supervisor S for G such that S/G implements E. S is a supervisor for G implies that $L(S/G)$ is controllable. Since $L(S/G) \subseteq L(G)$ and $P_{ex}(L(S/G)) = E$, it follows that $L(S/G) \in \mathscr{C}(P_{ex}^{-1}(E) \cap L(G))$. Thus there exists $K = L(S/G)$ such that $K \in \mathscr{C}(P_{ex}^{-1}(E) \cap L(G))$ and K implements E. ■

As an example, we already showed in Example 4.1 that the supervisory control problem under full observation has a solution. By Proposition 4.3.1 there should exist $K \in \mathscr{C}(P_{ex}^{-1}(E) \cap L(G))$ which implements E. In fact, as shown in Figure 4.3, we can see that $\mathscr{C}(P_{ex}^{-1}(E) \cap L(G))$ has two elements K_1 and K_2, out of which only K_2 implements E since K_1 fails to satisfy progress, and therefore it can be the language of the supervised system in the example.

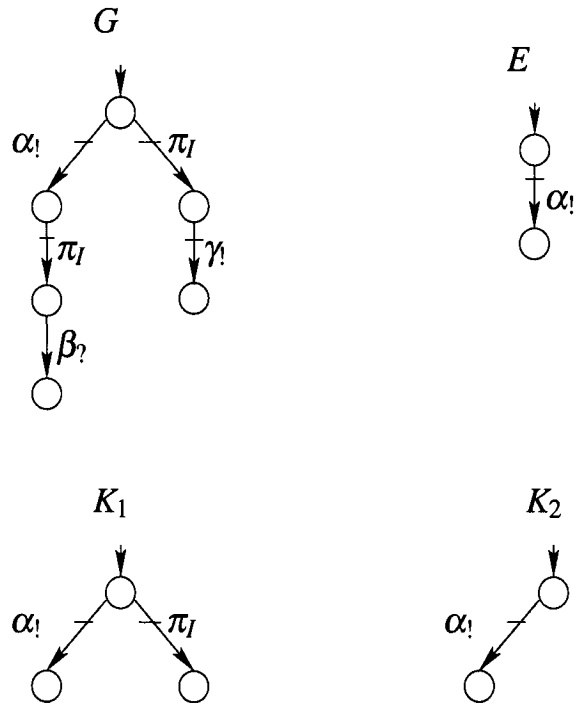


Figure 4.3: Example 4.1 revisited to illustrate Proposition 4.3.1.

Furthermore, the following result states a necessary and sufficient condition for the existence of a supervisor under partial observation.

Proposition 4.3.2 *Given a plant G, a service specification $E \subseteq \Sigma_{ex}^*$, there exists a feasible supervisor S for G such that $\widetilde{S}/G$ implements E if and only if there exists $K \in \mathscr{C}(P_{ex}^{-1}(E) \cap L(G))$ such that K implements E and K is observable with respect to (G, P_o).*

The proof here directly follows from Theorem 2.3.2 and is analogous to the proof of Proposition 4.3.1.

For instance, we already showed in Example 4.2 that the supervisory control problem under partial observation has a solution. By Proposition 4.3.1 there should exist $K \in \mathscr{C}(P_{ex}^{-1}(E) \cap L(G))$ that implements E and is observable. In fact, we can see that there exist two such languages as shown in Figure 4.4.

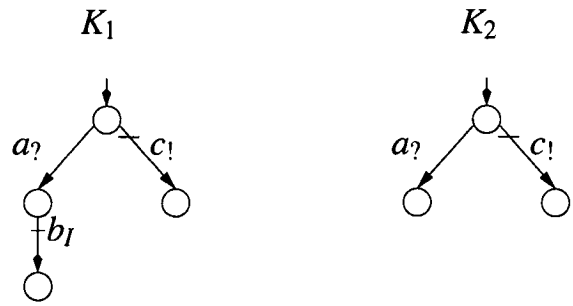


Figure 4.4: Example 4.2 revisited to illustrate Proposition 4.3.2.

We can see that both K_1 and K_2 are controllable with respect to $L(G)$ and both of them implement E. For K_1, the control action is to enable $c_!$ after ε and to disable $c_!$ after $a_? b_I$. Since $P_o(\varepsilon) \neq P_o(a_? b_I)$, the supervisory can perform its control task without ambiguity and therefore K_1 is observable with respect to (G, P_o). Similarly, we can verify that K_2 is also observable with respect to (G, P_o).

Remark: As the above example illustrates, since the specification does not impose any restriction on the internal events other than the progress condition, there may be several $K \in \mathscr{C}(P_{ex}^{-1}(E) \cap L(G))$ that are observable and implement E. Any such K can be selected in designing a supervisory control.

These results immediately lead to the following theorem.

Theorem 4.3.1 *The decentralized supervisory control problem of RDES has a solution if and only if there exists* $K \in \mathscr{C}(P_{ex}^{-1}(E) \cap L(G))$ *such that K implements E and K is co-observable with respect to* $(G, P_{1,o}, \Sigma_{1,c}, P_{2,o}, \Sigma_{2,c})$.

The proof of Theorem 4.3.1 follows directly from Proposition 4.3.1, Proposition 4.3.2 and Theorem 2.3.2.

Example 4.3: An example to illustrate Theorem 4.3.1.

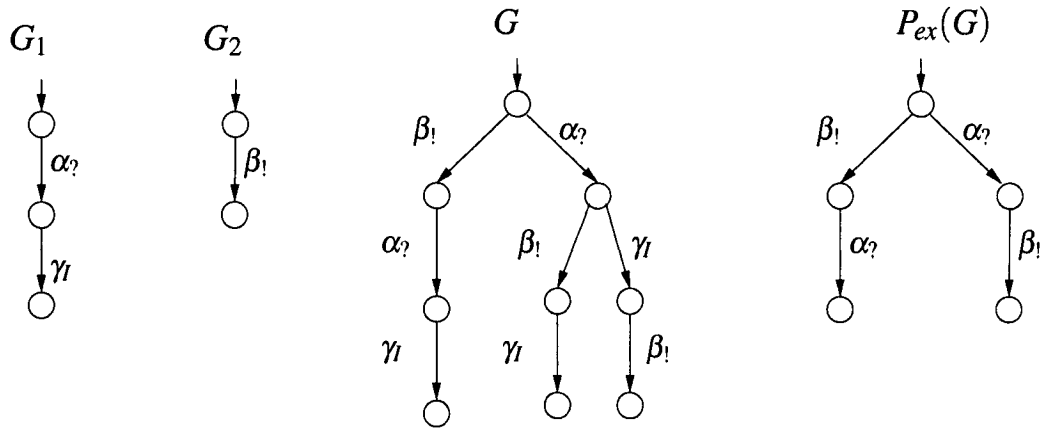


Figure 4.5: Example 4.3: components, the plant, and the external plant behavior.

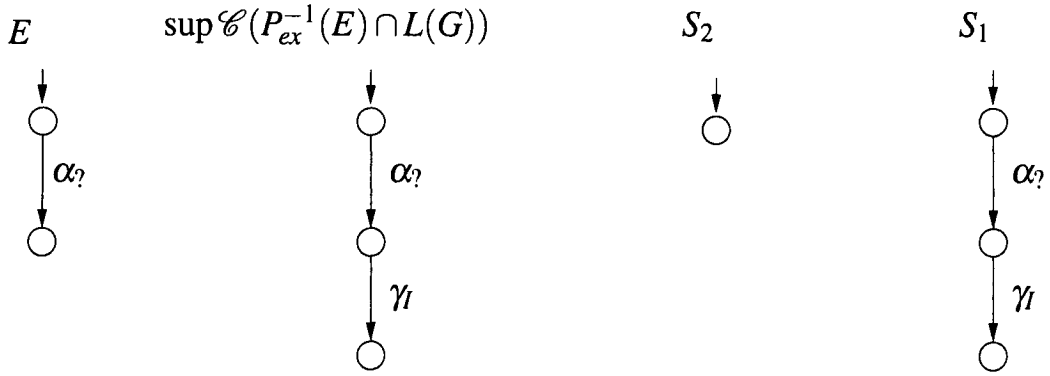


Figure 4.6: Example 4.3: the service specification and the decentralized solution.

As Figure 4.5 shows, in this example we have two components G_1 and G_2 with $\Sigma_{1,o} = \{\alpha_?, \gamma_I\}$, $\Sigma_{1,c} = \{\gamma_I\}$ and $\Sigma_{2,o} = \Sigma_{2,c} = \{\beta_!\}$. The entire plant behavior and its external view are shown in Figure 4.5.

The service specification E is shown in Figure 4.6. In this example, note that if supervisor S_1 takes no control action and supervisor S_2 disables $\beta_!$ at all times, then $L(S_1 \wedge S_2/G)$ will be equal to $\sup\mathscr{C}(P_{ex}^{-1}(E) \cap L(G))$ in Figure 4.6. Since $P_{ex}(\sup\mathscr{C}(P_{ex}^{-1}(E) \cap L(G))) = E$ and no violation of progress exists, the above $S_1 \wedge S_2/G$ implements E and is, therefore, a solution.

By Theorem 4.3.1, there should exist $K \in \mathscr{C}(P_{ex}^{-1}(E) \cap L(G))$ such that K implements E and K is co-observable with respect to $(G, P_{1,o}, \Sigma_{1,c}, P_{2,o}, \Sigma_{2,c})$. In fact, $K = \sup\mathscr{C}(P_{ex}^{-1}(E) \cap L(G))$ (as shown in Figure 4.6) implements E and is co-observable with respect to $(G, P_{1,o}, \Sigma_{1,c}, P_{2,o}, \Sigma_{2,c})$. Since supervisor S_1 takes no control action and supervisor S_2 disables $\beta_!$ at all times, either supervisor can make the correct control decision unambiguously.

On the other hand, there exists $K = \sup\mathscr{C}(P_{ex}^{-1}(E) \cap L(G)) \in \mathscr{C}(P_{ex}^{-1}(E) \cap L(G))$ such that K implements E and K is co-observable with respect to $(G, P_{1,o}, \Sigma_{1,c}, P_{2,o}, \Sigma_{2,c})$. By Theorem 4.3.1 there should exist local supervisors S_1 and S_2 such that $L(S_1 \wedge S_2/G)$ implements E and $S_1 \wedge S_2/G$ is co-observable with respect to $(G, P_{1,o}, \Sigma_{1,c}, P_{2,o}, \Sigma_{2,c})$. The required S_1 and S_2 are shown in Figure 4.6. □

Remark: When a service specification E cannot be implemented by decentralized controllers, it may be of interest in some cases to find its sublanguages and superlanguages that can be implemented. As shown in [28], unfortunately, neither the supremal sublanguage nor the infimal superlanguage, which can be implemented, exists in general.

4.4 Substitute Conditions for Safety

In some research domains, only safety is considered, see for example [12] and [13]. Therefore, it is of interest to study the case with safety alone, and investigate whether the necessary and sufficient conditions for the existence of decentralized supervisors implementing a service specification can be simplified. Formally, we have the following definition.

Definition 4.4.1 *Given a language $K \subseteq L(G)$ and a service specification $E \subseteq P_{ex}(L(G))$, K is*

said to ***conform to*** *E if K satisfies safety with respect to E, i.e.,* $P_{ex}(L(S/G)) = E$.

Then the decentralized supervisory control problem for RDES concerning only safety is defined as follows.

Problem 4.4.1 *Given the plant G composed of* G_1 *and* G_2*, the service specification E, and the sets of observable and controllable events of* G_1 *and* G_2 *as* $\Sigma_{1,o}$, $\Sigma_{1,c}$, $\Sigma_{2,o}$, $\Sigma_{2,c} \subseteq \Sigma$, *respectively, construct local supervisors* S_1 *and* S_2 *such that* $S = \widetilde{S_1} \wedge \widetilde{S_2}$ *is a supervisor for G and* S/G *conforms to E.*

We want to find the necessary and sufficient conditions for the existence of a solution to this problem. But first, we must address several technical difficulties. We modify the definitions of controllability, observability and co-observability as follows. Given a plant G, its generated language $L(G)$ and a service specification $E \subseteq P_{ex}(L(G))$,

Definition 4.4.2 *E is* P_{ex}*-controllable with respect to G if*

$$E = P_{ex}(\sup\mathscr{C}(P_{ex}^{-1}(E) \cap L(G))).$$

E is P_{ex}-controllable with respect to G if the supremal controllable sublanguage of $P_{ex}^{-1}(E) \cap L(G)$ exists and its projection is equal to E under P_{ex}. P_{ex}-controllability differs from standard controllability as any illegal exits from the specification can be disabled either directly or via one of the controllable internal events that lead to this exit. For convenience, we denote $E' := \sup\mathscr{C}(P_{ex}^{-1}(E) \cap L(G))$.

Definition 4.4.3 *E is* P_{ex}*-observable with respect to* $(G, P_o, \Sigma_{ex,c})$ *if*

$$(\forall s, s' \in E', \sigma \in \Sigma_{ex,c}) P_o(s) = P_o(s') \Rightarrow$$
$$s\sigma \in E' \wedge s' \in E' \wedge s'\sigma \in L(G) \Rightarrow s'\sigma \in E'.$$

When we say that E is P_{ex}-observable with respect to $(G, P_o, \Sigma_{ex,c})$, this can be interpreted as $\sup\mathscr{C}(P_{ex}^{-1}(E) \cap L(G))$ is observable with respect to (G, P_o, Σ_c).

Definition 4.4.4 *E is P_{ex}-co-observable with respect to $(G, P_{1,o}, \Sigma^1_{ex,c}, P_{2,o}, \Sigma^2_{ex,c})$ if* $\forall s \in E', \sigma \in \Sigma_{ex,c} = \Sigma^1_{ex,c} \cup \Sigma^2_{ex,c}$

$$s\sigma \notin E' \wedge s\sigma \in L(G) \Rightarrow$$
$$(\exists i \in \{1,2\}) P^{-1}_{i,o}[P_{i,o}(s)]\sigma \cap E' = \emptyset \wedge \sigma \in \Sigma^i_{ex,c}.$$

Similar to P_{ex}-observability, E being P_{ex}-co-observable can be interpreted as $\sup\mathscr{C}(P^{-1}_{ex}(E) \cap L(G))$ being co-observable.

We can see that in Example 4.1, E is P_{ex}-controllable with respect to G, and that in Example 4.2, E is P_{ex}-observable with respect to $(G, P_o, \Sigma_{ex,c})$.

Since $\Sigma^1_c \cap \Sigma^2_c = \emptyset$, we can conclude from [25] that checking P_{ex}-co-observability can be reduced to multiple checks of P_{ex}-observability , i.e., E is P_{ex}-co-observable with respect to $(G, P_{1,o}, \Sigma^1_{ex,c}, P_{2,o}, \Sigma^2_{ex,c})$ if and only if E is P_{ex}-observable with respect to $(G, P_{1,o}, \Sigma^1_{ex,c})$ and E is P_{ex}-observable with respect to $(G, P_{2,o}, \Sigma^2_{ex,c})$.

First we show that P_{ex}-controllability is necessary and sufficient for the existence of a supervisor under full observation such that the supervised plant conforms to the given service specification.

Proposition 4.4.1 *Given a plant G and a service specification E, there exists a supervisor S for G such that S/G conforms to E if and only if E is P_{ex}-controllable with respect to G.*

Proof: (only if) Suppose a supervisor S exists such that S/G conforms to E. Let V be the supervisory control map induced by S. We have

$s \in L(S/G) \wedge \sigma \in \Sigma_{uc} \wedge s\sigma \in L(G)$

$\Rightarrow s \in L(V/G) \wedge \sigma \in V(s) \wedge s\sigma \in L(G)$

$\Rightarrow s\sigma \in L(V/G)$ (by definition of $L(V/G)$)

$\Rightarrow s\sigma \in L(S/G)$

$\Rightarrow L(S/G)$ is controllable with respect to G.

Furthermore

$$P_{ex}(L(S/G)) = E$$

$$\Rightarrow L(S/G) \subseteq P_{ex}^{-1}(E)$$

$$\Rightarrow L(S/G) \subseteq P_{ex}^{-1}(E) \cap L(G)$$

$$\Rightarrow L(S/G) \in \mathscr{C}(P_{ex}^{-1}(E) \cap L(G))$$

$$\Rightarrow L(S/G) \subseteq \sup\mathscr{C}(P_{ex}^{-1}(E) \cap L(G))$$

$$\Rightarrow E = P_{ex}(L(S/G)) \subseteq P_{ex}(\sup\mathscr{C}(P_{ex}^{-1}(E) \cap L(G)))$$

Next we show $E \supseteq P_{ex}(\sup\mathscr{C}(P_{ex}^{-1}(E) \cap L(G)))$. We have:

$$\sup\mathscr{C}(P_{ex}^{-1}(E) \cap L(G)) \subseteq P_{ex}^{-1}(E)$$

$$\Rightarrow P_{ex}(\sup\mathscr{C}(P_{ex}^{-1}(E) \cap L(G))) \subseteq P_{ex}(P_{ex}^{-1}(E)) = E.$$

Together we get $E = P_{ex}(\sup\mathscr{C}(P_{ex}^{-1}(E) \cap L(G))$.

(if) Suppose we have $E = P_{ex}(\sup\mathscr{C}(P_{ex}^{-1}(E) \cap L(G)))$. We define $V(s)$ according to

$$V(s) = \begin{cases} \Sigma_{uc} \\ \quad \text{if } s \in L(G) - \sup\mathscr{C}(P_{ex}^{-1}(E) \cap L(G)) \\ \Sigma_{uc} \cup \{\sigma \in \Sigma_c | s\sigma \in \sup\mathscr{C}(P_{ex}^{-1}(E) \cap L(G)) \\ \vee s\sigma \notin L(G)\} \\ \quad \text{if } s \in \sup\mathscr{C}(P_{ex}^{-1}(E) \cap L(G)) \end{cases}$$

We show that $L(V/G) = \sup\mathscr{C}(P_{ex}^{-1}(E) \cap L(G))$. First note that by definition $\varepsilon \in L(V/G)$ and $\varepsilon \in \sup\mathscr{C}(P_{ex}^{-1}(E) \cap L(G))$. Assume inductively that $s \in L(V/G) \Rightarrow s \in \sup\mathscr{C}(P_{ex}^{-1}(E) \cap L(G)))$. Let $s\sigma \in L(V/G)$, i.e., $s \in L(V/G)$, $\sigma \in V(s)$, and $s\sigma \in L(G)$ by definition of $L(V/G)$. If $\sigma \in \Sigma_{uc}$ then $s\sigma \in \sup\mathscr{C}(P_{ex}^{-1}(E) \cap L(G))\Sigma_{uc} \cap L(G)$, which implies $s\sigma \in \sup\mathscr{C}(P_{ex}^{-1}(E) \cap L(G))$; whereas if $\sigma \in \Sigma_c$ then $s\sigma \in \sup\mathscr{C}(P_{ex}^{-1}(E) \cap L(G)))$ by the definition of V. Therefore $L(V/G) \subseteq \sup\mathscr{C}(P_{ex}^{-1}(E) \cap L(G))$.

Conversely, as $\varepsilon \in L(V/G)$ and $\varepsilon \in \sup\mathscr{C}(P_{ex}^{-1}(E) \cap L(G))$, assume inductively that $s \in \sup\mathscr{C}(P_{ex}^{-1}(E) \cap L(G)) \Rightarrow s \in L(V/G)$. Let $s\sigma \in \sup\mathscr{C}(P_{ex}^{-1}(E) \cap L(G)))$. If $\sigma \in \Sigma_{uc}$ then

$s\sigma \in L(V/G)\Sigma_{uc} \cap L(G)$, which implies $s\sigma \in L(V/G)$ by controllability; whereas if $\sigma \in \Sigma_c$, i.e., $\sigma \in V(s)$, then $s\sigma \in L(V/G)$ by definition of $L(V/G)$. Therefore $\sup\mathscr{C}(P_{ex}^{-1}(E) \cap L(G)) \subseteq L(V/G)$.

We conclude that $P_{ex}(L(S/G)) = P_{ex}(L(V/G)) = P_{ex}(\sup\mathscr{C}(P_{ex}^{-1}(E) \cap L(G))) = E$.

■

The following result states a necessary and sufficient condition for the existence of a solution to Problem 4.4.1 under the condition of partial observation.

Proposition 4.4.2 *Given a plant G, a service specification $E \subseteq \Sigma_{ex}^*$, there exists a feasible supervisor S for G such that S/G conforms to E iff*

1. *E is P_{ex}-controllable w.r.t G, and*
2. *E is P_{ex}-observable w.r.t $(G, P_o, \Sigma_{ex,c})$.*

The proof of Proposition 4.4.2 follows directly from Proposition 4.4.1 and Theorem 2.3.2.

These results immediately lead to the following theorem.

Theorem 4.4.1 *Problem 4.4.1 has a solution if and only if*

1. *E is P_{ex}-controllable w.r.t G, and*
2. *E is P_{ex}-observable w.r.t $(G, P_{1,o}, \Sigma_{ex,c}^1)$ and E is P_{ex}-observable with respect to $(G, P_{2,o}, \Sigma_{ex,c}^2)$.*

The proof of Theorem 4.4.1 follows directly from Proposition 4.4.1 and Theorem 2.4.1.

4.5 Conclusion

When introducing a service specification into the decentralized RDES framework, several technical difficulties arise. In this chapter, we first showed by examples that the standard conditions for controllability and co-observability are no longer necessary for the existence of

decentralized supervisors to implement a given service specification, and proposed alternative necessary and sufficient conditions for the existence of decentralized supervisors for RDES. In some cases, where only safety is considered, we also studied the corresponding decentralized supervisory control problem, and presented simpler necessary and sufficient conditions for the existence of decentralized solutions.

Chapter 5

Example: Pump System

In this section, we illustrate the concepts introduced in Chapters 3 and 4 on a simple reactive system: a pump/pipe configuration for pumping water. Throughout this chapter, we assume that all outputs are controllable.

5.1 System Description

Our pump system consists of one pump and two pipes. The pump delivers water while both (or only one) of the two pipes carry it to other components (which are not modeled in our system). We assume that there is an operator who is in charge of the entire system, for example giving orders to start or stop a mission, so it is natural to treat the operator as part of the environment.

We further assume that pipe 1 and pipe 2 are of different types, for example, they are owned by two different companies. Suppose that pipe 1 and the pump are owned by the same company. In contrast, let pipe 2 belong to another company, and any other company who uses pipe 2 will be charged for the time that pipe 2 is in use. Furthermore, we assume that the running expense of pipe 2 is higher than that of pipe 1. Therefore, when a command is issued to the pump to start operating, pipe 1 will be the default pipe to use.

The system architecture is shown in Figure 5.1.

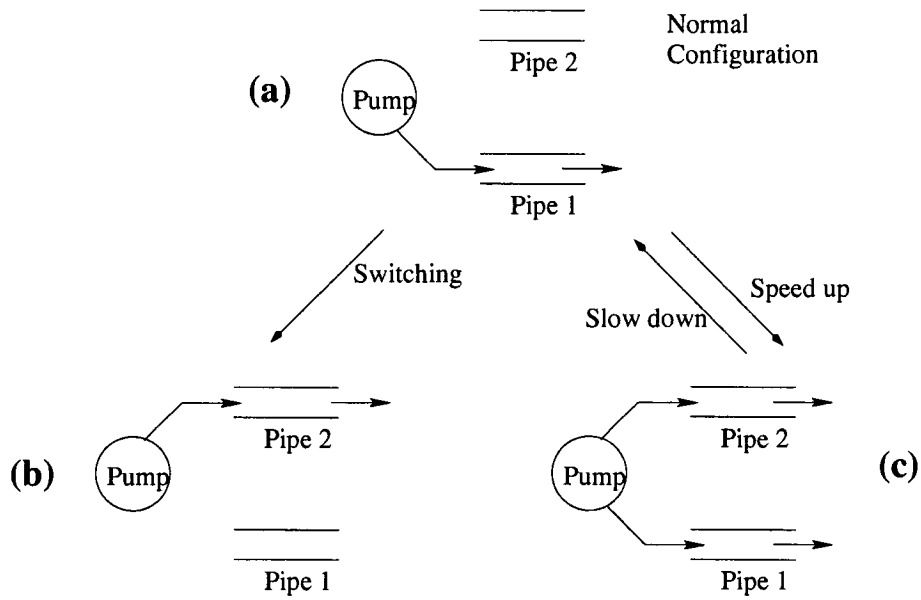


Figure 5.1: Pump system: the system architecture.

When the operator issues a command to start a mission, the pump will use the default pipe (pipe 1) (Figure 5.1(a)). If this pipe is found to be leaking, the pump will switch to another pipe (Figure 5.1(b)). If the operator finds that the pumping speed is not enough to finish the mission on time, a command to speed up the pump will be executed, i.e., both pipes are used at the same time (Figure 5.1(c)). When both pipes are in use and if the operator finds that the pumping speed is faster than necessary, a command to ask the pump to slow down will be issued to reduce the total expense, i.e., the usage of pipe 2 is discontinued while pipe 1 remains in use.

We assume that there is a valve for each pipe that governs whether or not the pipe is in use. In general, pipe 1 has three modes: the "On" mode when the valve is open, the "Faulty" mode when the valve is open but pipe 1 is leaking, and the "Off" mode when the valve is closed. Pipe 1 is initially in the "Off" mode waiting for the operator's command to start a mission. When the "start" command is issued, the valve will be opened and pipe 1 will start

to deliver water. Consequently, pipe 1 goes to the "On" mode. If an input is issued ordering pipe 1 to stop delivering water, the valve will be closed and the mode of pipe 1 goes back to "Off". Finally, if a sensor detects that pipe 1 is leaking, pipe 1 goes into the "Faulty" mode. In this mode, the valve will automatically be closed. Pipe 1 is modeled as an RDES shown in Figure 5.2.

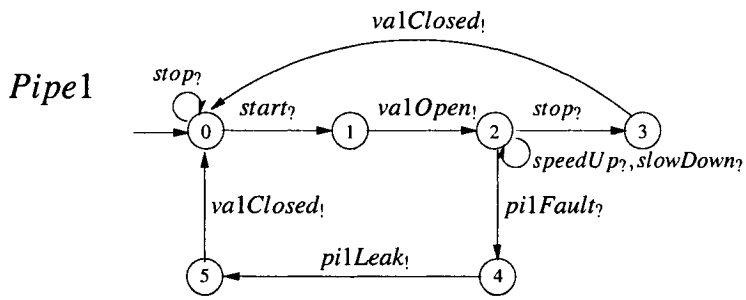


Figure 5.2: Pump system: pipe 1.

Note that state 0 and state 1 correspond to the "Off" mode, state 2 and state 3 correspond to the "On" mode, and state 4 and state 5 correspond to the "Faulty" mode.

The meaning of event labels in Figure 5.2 is listed below.

- $start_?$: the input pipe 1 receives from the operator to open the valve.
- $valOpen_!$: the output pipe 1 sends to its environment indicating that the valve has just been opened.
- $stop_?$: the input pipe 1 receives from the operator to close the valve.
- $valClosed_!$: the output pipe 1 sends to its environment indicating that the valve has just been closed.
- $pilFault_?$: the input from some sensors which informs pipe 1 that a leak has just been detected.

- $pi1Leak_!$: the output pipe 1 sends to the pump indicating that pipe 1 is leaking and will be turned off so that the pump should disconnect from pipe 1.

Note that $speedUp_?$ (which leads to transition from Figure 5.1(a) to Figure 5.1(c)) and $slowDown_?$ (which leads to transition from Figure 5.1(c) to Figure 5.1(a)) are only possible when pipe 1 is in state 2 (as indicated by selfloops at state 2). This can be interpreted as the operator will only issue the "speed up" and the "slow down" orders (Figure 5.1(a) and Figure 5.1(c)) when the valve of pipe 1 is open and no input has been issued to close the valve. In other words, the "speed up" and the "slow down" orders are not defined when the system is still in the start-up phase (states 0 and 1), in the faulty mode (states 4 and 5) or in the termination phase (state 3).

We assume that pipe 2 is leakproof and is always ready to be used. Since pipe 2 and the pump belong to different companies, we assume that the valve of pipe 2 will be automatically opened when it is connected to the pump. In other words, in this example, we only design a control policy for the company that owns the pump and pipe 1. The RDES that models pipe 2 is shown in Figure 5.3.

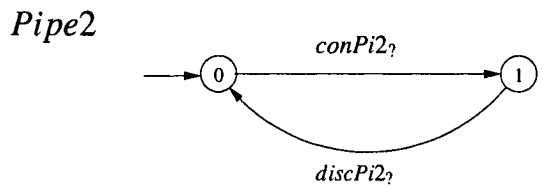


Figure 5.3: Pump system: pipe 2.

The meaning of event labels in Figure 5.3 is listed below.

- $conPi2_?$: input from the pump indicating that the pump has just connected to pipe 2.
- $discPi2_?$: input from the pump indicating that the pump has just disconnected from pipe 2.

With these two inputs, the owner of pipe 2 will be able to record the starting time and the ending time of a given usage period and calculate the usage fee.

The behavior of the pump is described as follows.

- When the *start*? event is received to start a delivering mission, pipe 1 will be the default pipe used by the pump.
- When only pipe 1 is in use and the *stop*? event is received to stop the mission, the pump will disconnect from pipe 1, halt and wait for the next *start*? command.
- When only pipe 1 is in use and the *speedUp*? event is received to accelerate, the pump will connect to pipe 2 and both pipes are in use.
- When only pipe 1 is in use and pipe 1 is found to be leaking, the pump will switch the mission to pipe 2 by connecting to pipe 2 first and then disconnecting from pipe 1.
- When both pipes are in use and the *stop*? event is received, the pump will stop delivering water by disconnecting from pipe 2 first and then disconnecting from to pipe 1, since pipe 2 is always more expensive to use.
- When both pipes are in use and the *slowDown*? event is received, the pump will disconnect from pipe 2.
- When both pipes are in use and pipe 1 is found to be leaking, the pump will disconnect from pipe 1 and the valve of pipe 1 will be closed. In this case, the system is running in a faulty mode, and hence the *speedUp*? order and the *slowDown*? order are not defined. When the *stop*? event is received, the pump will disconnect from pipe 2.

The RDES that models the pump is shown in Figure 5.4.

The meaning of event labels in Figure 5.4 is listed below.

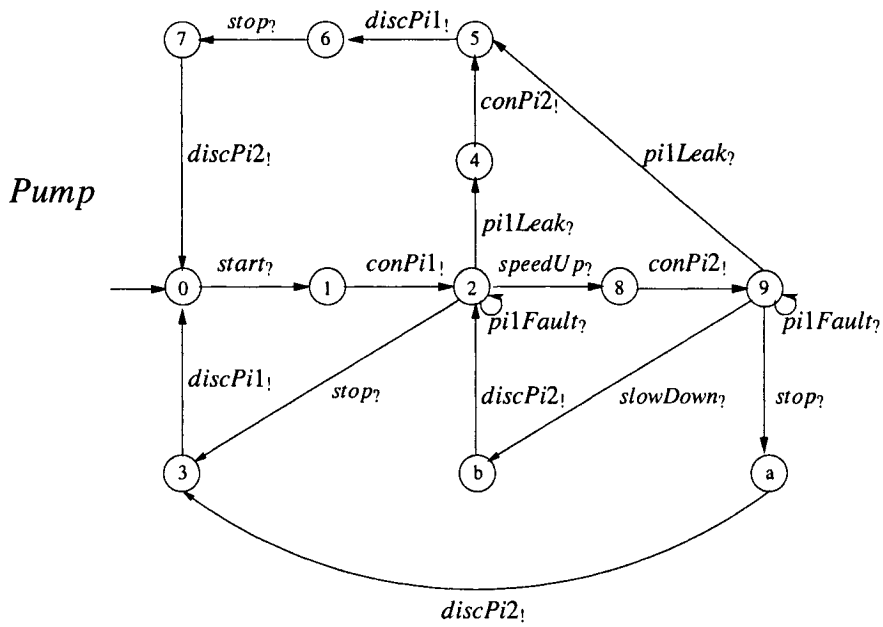


Figure 5.4: Pump system: the pump.

- *start*? and *stop*? have the same meaning as in Figure 5.2.
- *conPi*1!: output to the operator indicating that the pump has connected to pipe 1.
- *discPi*1!: output to the operator indicating that the pump has disconnected from pipe 1.
- *conPi*2!: output to pipe 2 indicating that the pump has connected to pipe 2.
- *discPi*2!: output to pipe 2 indicating that the pump has disconnected from pipe 2.
- *piLeak*?: input received from pipe 1 indicating that pipe 1 is leaking.
- *speedUp*?: input received from the operator ordering the pump to use both pipes.
- *slowDown*?: input received from the operator ordering the pump to use only pipe 1.

Note that *pi*1*Fault*? is defined only at states 2 and 9 (indicated by selfloops). This can be interpreted as the sensors of pipe 1 only function when pipe 1 is in use, i.e., pipe 1 is connected to the pump and is delivering (then sensors can work by comparing incoming and outgoing

water flow speeds). In other words, it is assumed that $pi1Fault_?$ does not occur when the pump is in some transitional stage (since a transitional stage is very short in time), such as when the pump is switching from the "Off" mode to the "On" mode triggered by $start_?$ (state 1) and when the pump is switching from using only pipe 1 to using both pipes triggered by $speedUp_?$ (state 8).

The entire plant behavior is shown in Figure 5.5.

Note that every state of the plant in Figure 5.5 is assigned a three-digit hexadecimal number of the form *ijk* such as 000, which indicates that the pump is now in state *i*, pipe 1 is in state *j*, and pipe 2 is in state *k*. The I/O automata that model the pump, pipe 1 and pipe 2 are pairwise compatible. When they are composed, there are three internal connections:

- $pi1Leak_{pi1ToPump}$ passes messages from pipe 1 to the pump,
- $conPi2_{pumpTopi2}$ passes messages from the pump to pipe 2, and
- $discPi2_{pumpTopi2}$ passes messages from the pump to pipe 2.

The sets of controllable and observable events of the pump, pipe 1 and pipe 2 are noted below.

The pump: $\Sigma_{pp,o} = \{start_?, stop_?, conPi1_!, discPi1_!, conPi2_!, discPi2_!, pi1Leak_?,$
$slowDown_?, speedUp_?, val1Open_!, val1Closed_!\}$,

$\Sigma_{pp,c} = \{conPi1_!, discPi1_!, conPi2_!, discPi2_!\}$.

Pipe 1: $\Sigma_{pi1,o} = \{start_?, stop_?, conPi1_!, discPi1_!, pi1Fault_?, pi1Leak_!, val1Open_!, val1Closed_!\}$,

$\Sigma_{pi1,c} = \{val1open_!, pi1Leak_!, pi1Off_!\}$.

Pipe 2: $\Sigma_{pi1,o} = \{conPi2_?, discPi2_?\}$,

$\Sigma_{pi1,c} = \emptyset$.

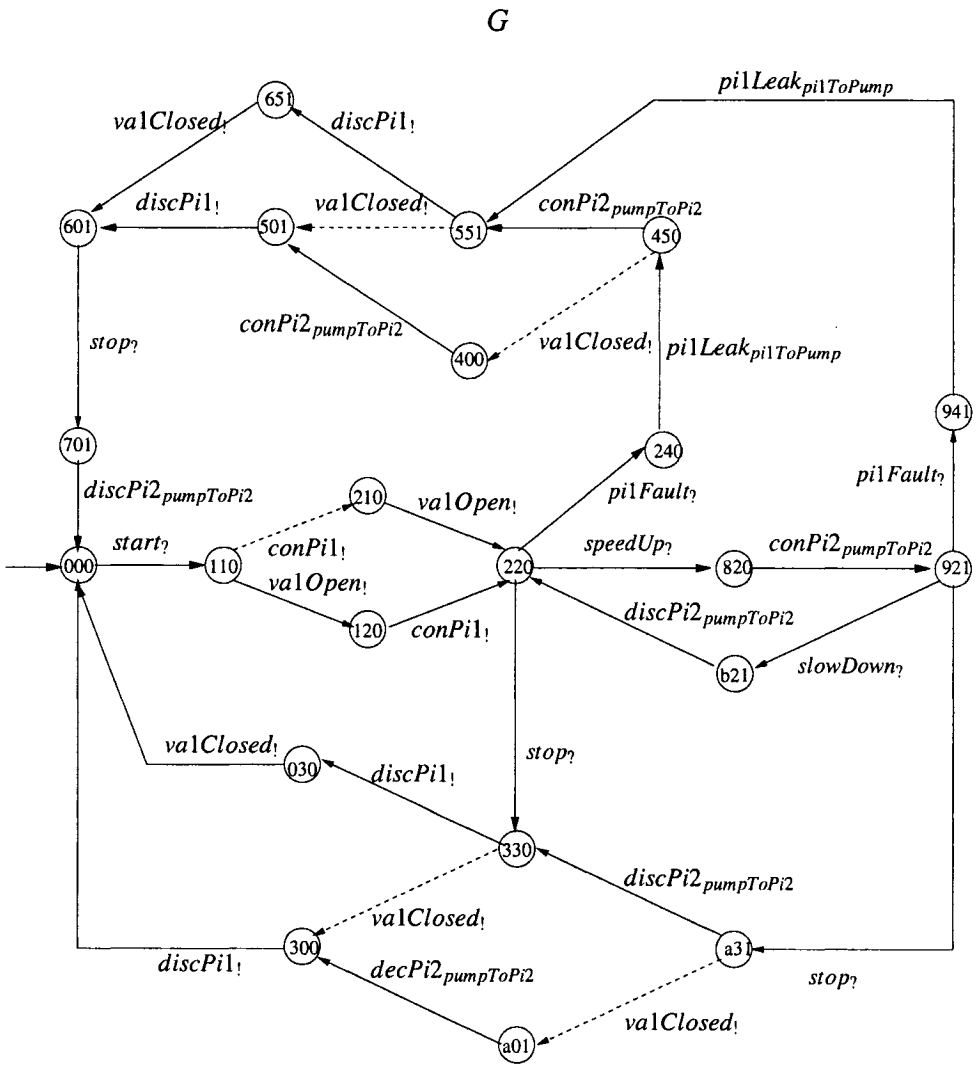


Figure 5.5: Pump system: the entire plant behavior. Illegal events are shown by dashed lines.

5.2 Analysis

In this section we show how our supervisory control theory for RDES can be applied to this pump system.

5.2.1 A Service Specification E

We define the following service specification E:

1. The valve of Pipe 1 should be already opened before the pump connects to pipe 1.

2. The pump should disconnect from pipe 1 before the valve of Pipe 1 is closed (either caused by the $stop_?$ event order or by the leaking of pipe 1).

The service specification for 1 and 2 is shown in Figure 5.6.

The complete service specification E in Figure 5.6 is determined by $(E_1 \cap E_2) \cap L(G)$. Note that in Figure 5.6, Σ denotes the union of all events in the pump system, i.e., $\Sigma = \Sigma_{pp,o} \cup \Sigma_{pi1,o} \cup \Sigma_{pi2,o}$. The joint behavior of the entire plant and the service specification equals the entire plant behavior in Figure 5.5 with illegal events (i.e., those events that must be disabled) shown by dashed lines.

We denote the supervisors for the pump and pipe 1 by S_{pp} and S_{pi1}, respectively. There are 5 cases where an event should be disabled:

- The event $conPi1_!$ that leads state transition from state 110 to state 210 should be disabled by S_{pp}.

- The event $val Closed_!$ that leads state transition from state a31 to state a01 should be disabled by S_{pi1}.

- The event $val Closed_!$ that leads state transition from state 330 to state 300 should be disabled by S_{pi1}.

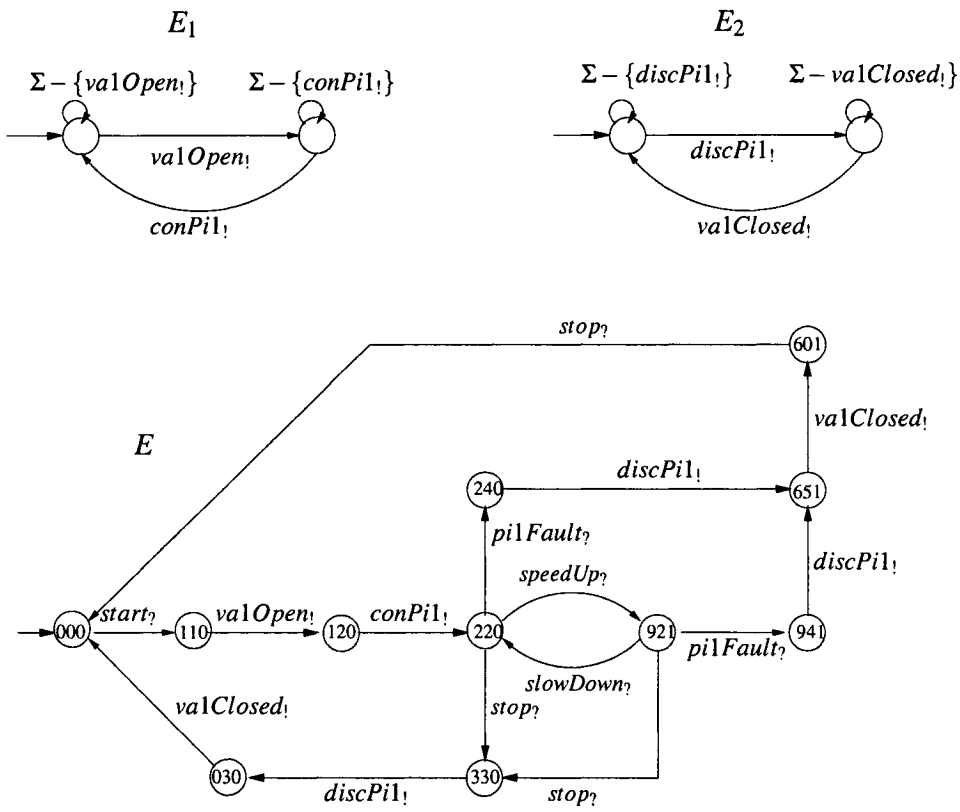


Figure 5.6: Pump system: the service specification.

- The event $valClosed_!$ that leads state transition from state 551 to state 501 should be disabled by S_{pil}.
- The event $valClosed_!$ that leads state transition from state 450 to state 400 should be disabled by S_{pil}.

5.2.2 Decentralized Supervisors that Implement E

To check whether there exist decentralized supervisors that implement E, using Theorem 4.3.1, we need to check whether or not there exists $K \in \mathscr{C}(P_{ex}^{-1}(E) \cap L(G))$ such that K implements E and K is co-observable with respect to $(G, P_{pp,o}, \Sigma_{pp,c}, P_{pil,o}, \Sigma_{pil,c})$, where $P_{pp,o} : \Sigma^* \rightarrow \Sigma_{pp,o}^*$ and $P_{pil,o} : \Sigma^* \rightarrow \Sigma_{pil,o}^*$ are natural projections. If such a K exists, it should be a subset of

$E' = \sup\mathscr{C}(P_{ex}^{-1}(E) \cap L(G))$. Therefore we start by checking E', which is given in Figure 5.7.

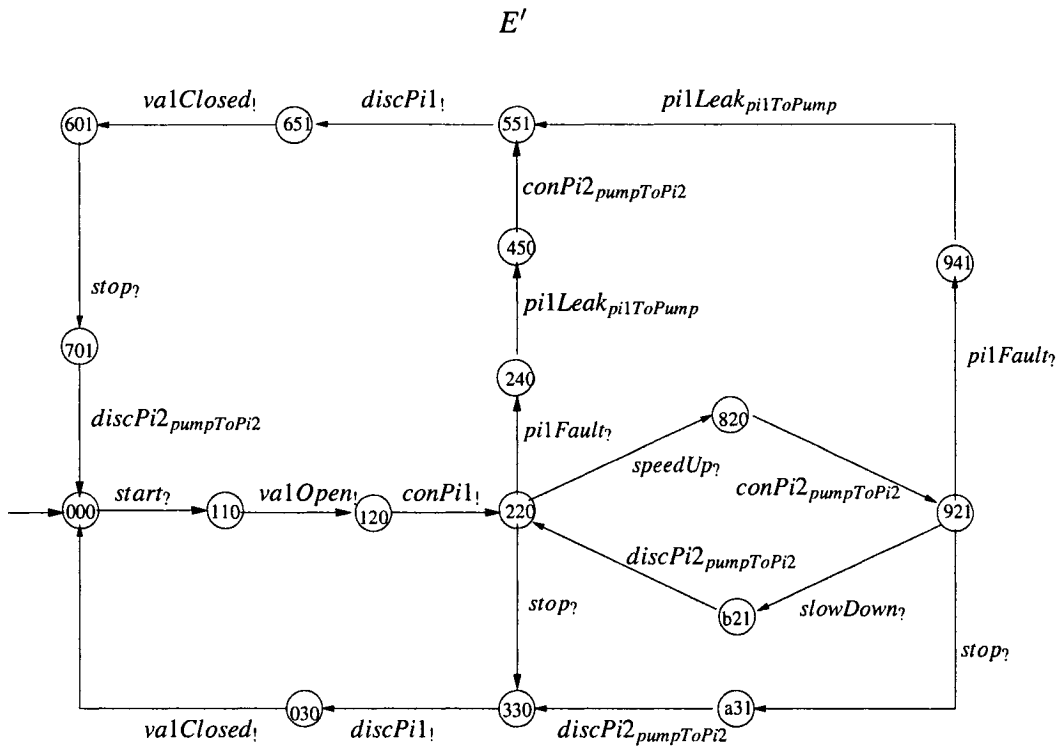


Figure 5.7: Pump system: $E' = \sup\mathscr{C}(P_{ex}^{-1}(E) \cap L(G))$.

We can see that $P_{ex}(E') = E$, i.e., E' satisfies safety. E' also satisfies progress, since for every $s \in E'$ whose projection $P_{ex}(s)$ is a prefix of another sequence $t \in E$, it is possible to extend s by another sequence u such that their catenation equals t under P_{ex}, i.e., $P_{ex}(su) = t$. For instance, consider the string $s = start_?valOpen_!conPil_!\ speedUp_? \in E'$, its projection is a prefix of $t = start_?valOpen_!conPil_!speedUp_?slowDown_? \in E$. There exists $u = conPi2_{pumpToPi2}slowDown_? \in \Sigma^*$ such that $P_{ex}(su) = t$. Furthermore, E' is co-observable since all control decisions can be made without ambiguity. For example, $conPil_!$ should be disabled after $start_?$ and enabled after $start_?valOpen_!$. As $start_?$, $valOpen_! \in \Sigma_{pp,o}$, S_{pp} can tell $start_?$ and $start_?valOpen_!$ apart and make the correct control decision.

Therefore, E' is the right K we are looking for. The behavior of E' can be interpreted in

our scenario as follows, which coincides with all our assumptions and requirements.

- When $start_?$ is issued, S_{pi1} enables $val Open$, and then S_{pp} allows the pump to connect to pipe 1 and turn itself on. The pump system starts to work. When $stop_?$ is issued, S_{pp} enables the pump to disconnect from pipe 1 and turn itself off, and then S_{pi1} enables $val Closed$ to close the valve. The pump system terminates.

- When the pump and pipe 1 are working and pipe 1 is leaking, S_{pi1} sends a message to S_{pp} ($pi1Leak_{pi1ToPump}$). S_{pp} enables the pump to connect to pipe 2 and then disconnect from pipe 1. Finally S_{pi1} enables $val Closed$ and the valve is closed.

- When the pump and pipe 1 are working and $speedUp_?$ is received, S_{pp} enables the pump to connect to pipe 2 and both pipes are in use. If later $slowDown_?$ is received, S_{pp} allows the pump to disconnect from pipe 2 and only pipe 1 is in use.

- When the pump and both pipes are working and pipe 1 is leaking, S_{pi1} sends a message to S_{pp} ($pi1Leak_{pi1ToPump}$). S_{pp} enables the pump to disconnect from pipe 1 and then S_{pi1} allows the valve to be closed. In this case only pipe 2 is use.

  S_{pp} and S_{pi1} are shown in Figure 5.8 and Figure 5.9, respectively.

5.3 Pump System: Revisited

In this example, one can also model this pump system in a slightly different way: the environment of the pump (or pipe 1) is only the operator instead of the operator and pipe 1 (or the pump). In other words, some of the output events of pipe 1, such as $val Open_!$ and $val Closed_!$, are no longer observable to the supervisor for the pump. Similarly, the output events of the pump, such as $conPi1_!$ and $discPi1_!$, are no longer observable to the supervisor for the pipe, which is more difficult to handle but more realistic. Thus, the sets of controllable and observable events of the pump, pipe 1 and pipe 2 are updated as follows.

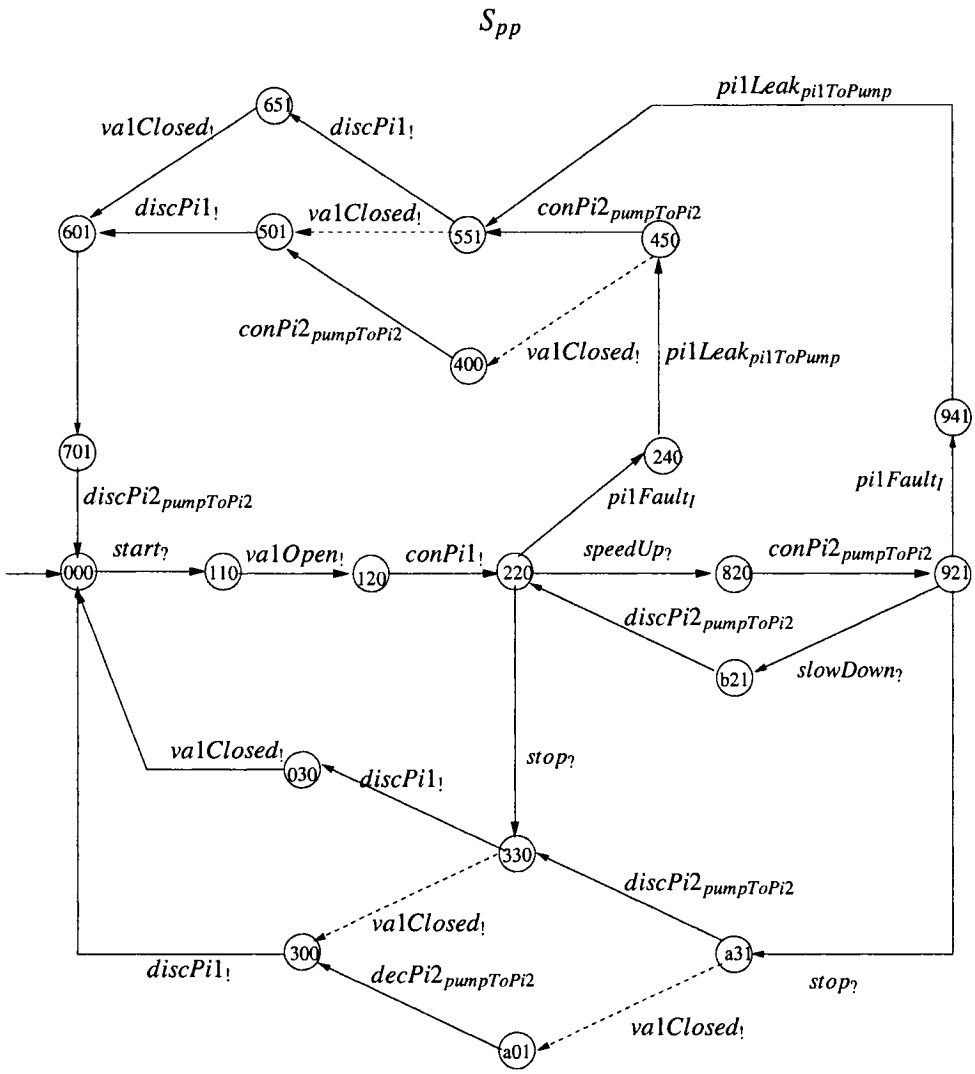


Figure 5.8: Pump system: the plant under the supervision of S_{pp}. Events that are illegal but uncontrollable to S_{pp} (i.e., should be disabled by S_{pi1}) are shown by dashed lines.

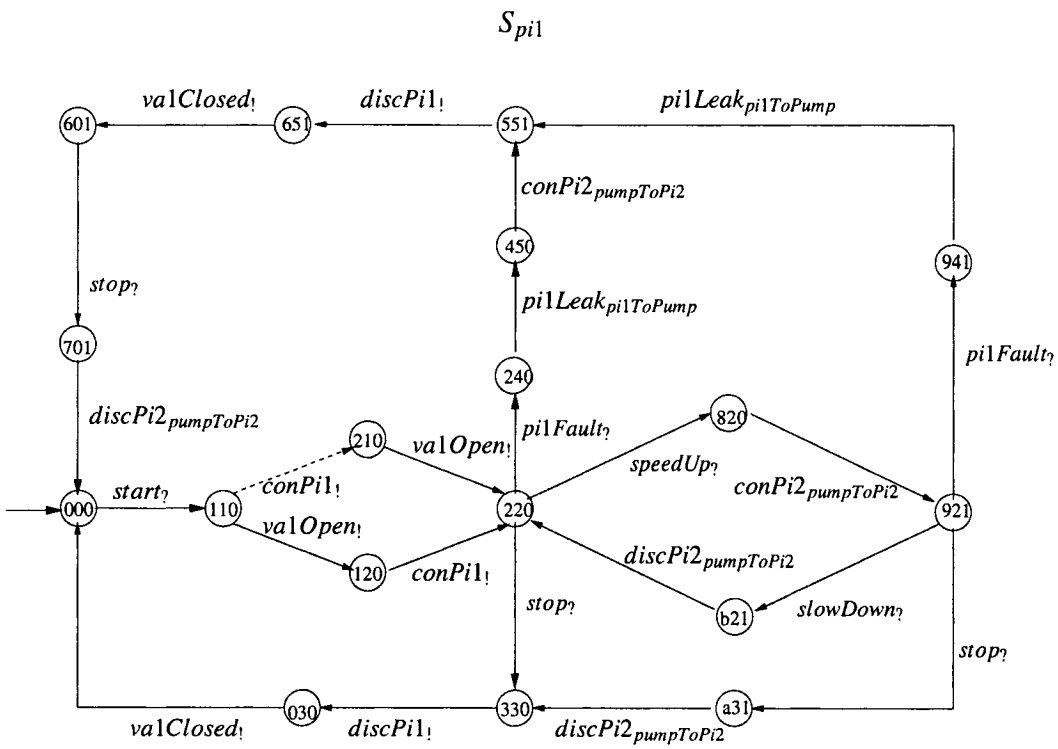


Figure 5.9: Pump system: the plant under the supervision of S_{pi1}. Events that are illegal but uncontrollable to S_{pi1} (i.e., should be disabled by S_{pp}) are shown by dashed lines.

The pump: $\Sigma_{pp,o} = \{start_?, stop_?, conPi1_!, discPi1_!, conPi2_!, discPi2_!, pi1Leak_?,$

$slowDown_?, speedUp_?\}$,

$\Sigma_{pp,c} = \{conPi1_!, discPi1_!, conPi2_!, discPi2_!\}$.

Pipe 1: $\Sigma_{pi1,o} = \{start_?, stop_?, pi1Fault_?, pi1Leak_!, va1Open_!, va1Closed_!\}$,

$\Sigma_{pi1,c} = \{va1open_!, pi1Leak_!, va1Closed_!\}$.

Pipe 2: $\Sigma_{pi1,o} = \{conPi2_?, discPi2_?\}$,

$\Sigma_{pi1,c} = \emptyset$.

In this case, E' still satisfies safety and progress, i.e., a centralized supervisor exists. However, E is no longer P_{ex}-co-observable. For example two sequences $start_?$ and $start_?va1Open_!$ looks the same to the supervisor of the pump, and the event $conPi1_!$ must be disabled after the former and enabled after the latter sequence, but $P_{pp,o}(start_?va1Open_!) = P_{pp,o}(start_?) = start_?$. In other words, S_{pp} cannot make the correct control decision without ambiguity. Thus, a decentralized solution in our framework does not exist.

When E is P_{ex}-controllable but not P_{ex}-co-observable, i.e., when at least one event must be disabled but none of the local supervisors knows that, then communication between local supervisors may help to solve the problem. In this section we show that existing results on incorporating communication into decentralized discrete-event control, such as [29], can be applied directly to RDES.

The "as early as possible" communication strategy introduced in [29] is concerned with analysis of legal and illegal sequences that cannot be distinguished by supervisors (i.e., sequences that violate co-observability). For example, $start_?va1Open_!conPi1_!$ is a legal sequence and $start_?conPi1_!$ is an illegal sequence, and the supervisor S_{pp} making the control decision cannot tell them apart, i.e., $P_{pp,o}(start_?va1Open_!conPi1_!) = P_{pp,o}(start_?conPi1_!)$. The communication strategy in [29] finds an "earliest possible" event $\pi \notin \Sigma_{pp,o}$ along those indistinguishable sequences. Since S_{pp} must make the control decision for $conPi1_!$, there must exist at least one event π that S_{pi1} sees that will allow S_{pp} to distinguish the two sequences.

The property of observability guarantees that such an event π must exist. If S_{pi1} communicates the occurrence of π to S_{pp}, S_{pp} will be able to make the correct control decision about $conPi1_!$.

Our procedure for incorporating communication as adapted for decentralized RDES proceeds as follows:

1. Isolate all legal and illegal sequences, i.e., families of sequences that violate P_{ex}-co-observability [25]. Find the first place the illegal/legal sequences differ, i.e., an "earliest possible" event that the sender observes but the receiver does not.

2. Translate the RDES plant into a "monitoring automata" (Section 2.4.2) as the new automaton structure into which we incorporate communication.

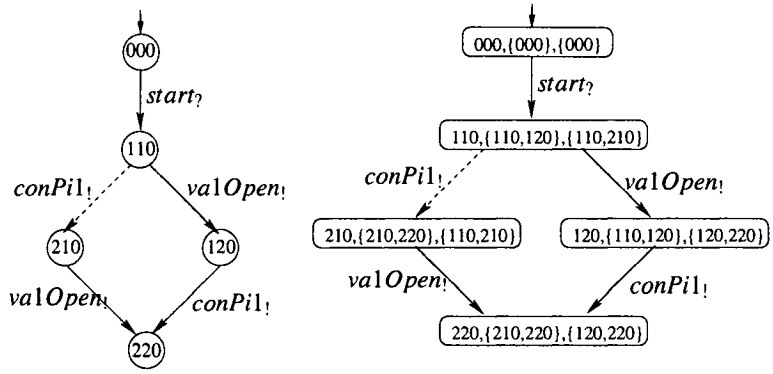
3. Add a communication event $com_{senderToReceiver} : i$ right after each selected event in Step 1, where $i \in \mathbb{Z}^+$ and *sender* (or *receiver*) denotes the supervisor who sends (or receives) this communication event. Note that such a communication event, once sent out, is observable to both the sender and the receiver.

We assume that the communication event occurs instantaneously. The communication event represents the sending of the "sender" supervisor's local state estimates; however, it could also represent the communication of the last observed event of the sender.

We show how this procedure can be applied to our pump system. As stated in Section 5.2.1, there are five cases where an illegal event must be disabled. Unfortunately, all these five control decision cannot be made without ambiguity. For example, consider the pair of strings $start_?conPi1_!$ and $start_?val1Open_!conPi1_!$. We show how to find the "earliest possible" communication event for this pair. For the purpose of this thesis, only the observer automata and monitoring automaton for a selected part of the plant, called G', is shown in Figure 5.10.

The observer automaton in Figure 5.10(c) shows that the event $conPi1_!$ should be disabled but S_{pp} does not exactly know whether this state transition will lead to state 210 (in

Without communication



(a) Part of the plant G' **(b) Monitoring automaton for G'**

(c) $Obs_1(G')$ for the pump **(d) $Obs_2(G')$ for pipe 1**

Figure 5.10: Observer and monitoring automata for a selected part of the plant.

which case, it has to disable $conPil_!$), or 220 (in which case, it has to enable $conPil_!$). Therefore, this is a violation of P_{ex}-co-observability. This pair of illegal /legal sequences are already isolated. Let the legal sequence $s = start_?conPil_!$ and the illegal sequence $s' = start_?val Open_!conPil_!$, since $conPil_!$ is the event S_{pp} must control, we consider their prefixes $start_?val Open_!$ and $start_?$ where $P_{pp}(start_?val Open_!) = P_{pp}(start_?)$. When we remove the common prefixes (i.e., $start_?$), we are left with ε and $val Open_!$. Therefore, we select $val Open_!$ to be our "earliest possible" non-empty event to communicate.

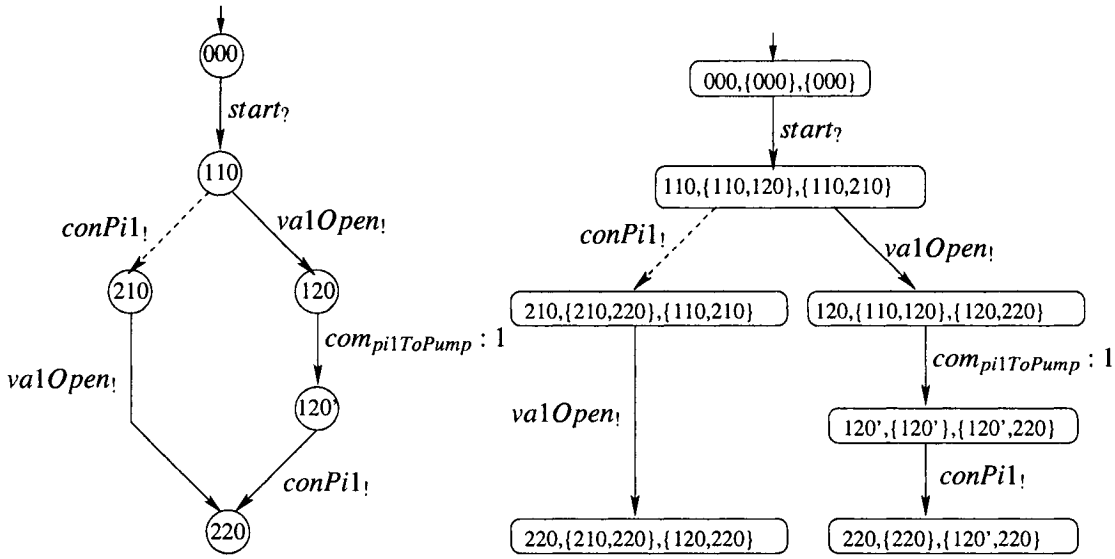
The observer automata and monitoring automaton for G' after incorporating communication is shown in Figure 5.11.

We can see that after incorporating communication into RDES, s and s' no longer look the same to S_{pp} since $com_{pilToPump} : 1$ is observable to S_{pp} and therefore $P_{pp,o}(start_?$ $val Open_!com_{pilToPump} : 1) = start_?com_{pilToPump} : 1$ and $P_{pp,o}(start_?) = start_?$ are different. That is to say, the control decision can be made without ambiguity. After incorporating communication into RDES, we can design a decentralized solution even if the service specification is not P_{ex}-co-observable (as long as the service specification is P_{ex}-observable).

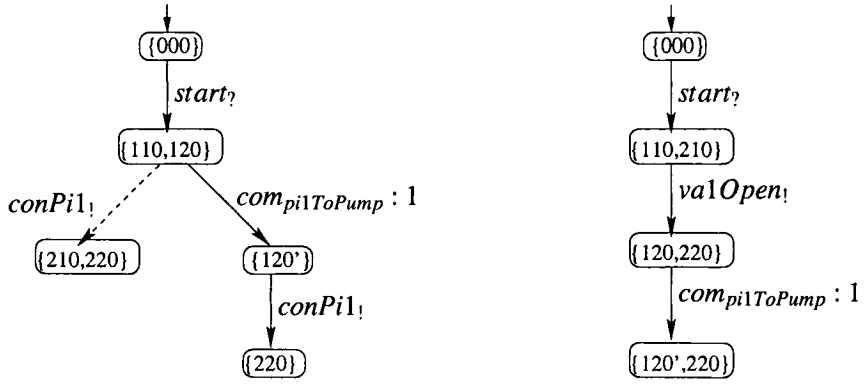
5.4 Conclusion

In this chapter, we presented a pump system to illustrate the concepts developed in Chapters 3 and 4. We showed that the decentralized supervisory control theory in Chapter 3 can be applied to this pump system when both safety and progress are considered. Finally we illustrated that an existing strategy for incorporating communication into the decentralized supervisory control of DES can be directly applied to our decentralized RDES framework.

With communication



(a) Part of the plant G' **(b) Monitoring automaton for G'**



(c) $Obs_1(G')$ for the pump **(d) $Obs_2(G')$ for pipe 1**

Figure 5.11: Incorporating communication into RDES.

Chapter 6

Conclusion and Future work

6.1 General Conclusion

In this thesis, we studied the supervisory control problem of Reactive Discrete-Event Systems (RDES). The study of RDES is a research area of current vitality and plays an important role in a wide range of application domains. The motivation of this thesis is based on the observation that although many computer languages and software have been developed to simulate reactive systems, the modeling and analysis problem from a control perspective is in an early stage. Understanding how reactive systems can be modeled from a control perspective and how existing results in supervisory control framework can be applied will allow us to combine analysis tools and simulation tools to handle RDES better.

We made some necessary but minor modification to the Input/Output (I/O) automata introduced by Lynch and Tuttle [5] to model RDES. We presented a compact but meaningful decentralized RDES architecture to study how RDES co-operate with one another. Within this framework, we defined the supervisory control problem and decentralized supervisory control problem of RDES which concern with two important properties: safety and progress.

In addition, we illustrated how the standard definitions of controllability, observability and co-observability cannot be directly applied to our framework. Substitute sufficient and necessary conditions for the existence of decentralized solutions were given.

Furthermore, in many applications, only safety is considered. We also considered this case, provided the definition of the corresponding supervisory control problem and decentralized supervisory control problem, and gave simpler sufficient and necessary conditions for the existence of their solutions.

Finally we illustrated our decentralized RDES architecture and the revised supervisory control theory by an example. The example was revisited to illustrate that existing results on introducing communication between supervisors can be directly applied to solve the problem when the service specification is not P_{ex}-co-observable (as long as it is P_{ex}-controllable and P_{ex}-observable).

6.2 Future Work

Many interesting applications of RDES are real-time systems. Many real-time systems are *safety-critical* [30]. A safety-critical system is a system whose functional failure could lead to loss of life or property. For example, safety-critical systems that produce outputs too late may cause harm to their environment, such as a heart-monitoring system. Therefore, it may be interesting to extend the existing work to timed I/O automata.

The sufficient and necessary conditions for the existence of solutions to the supervisory control problem and decentralized supervisory control problem of RDES concerning both safety and progress given in Chapter 4 are in general difficult to check. We would like to find a more suitable version of these conditions that lend themselves to a procedure for constructing local supervisors.

We plan to further explore the modeling effects of introducing communication into

RDES. The decentralized RDES architecture may possibly be more powerful with communication between local supervisors.

Bibliography

[1] S. L. Ricker, *Knowledge and communication in decentralized discrete-event control.* PhD thesis, Department of Computing and Information Science, Queen's University, 1999.

[2] K. Q. Zhu, W. Y. Tan, and A. E. Santosa, "Reactive web agents with open constraint programming," *Proceedings of 5th International Symposium on Autonomous Decentralized Systems*, pp. 251–254, March 2001.

[3] B. Lee, *Specification and design of reactive systems.* PhD thesis, University of California, Berkeley, 2000.

[4] W. M. Wonham, "Supervisory control of discrete-event systems," *http://www.control.toronto.edu/people/profs/wonham/wonham.html.*

[5] N. A. Lynch and M. R. Tuttle, "An introduction to input/output automata," *CWI Quarterly*, vol. 2, no. 3, pp. 219–246, 1989.

[6] Z. Manna and A. Pnueli, *The Temporal Logic of Reactive and Concurrent Systems: Specification.* Springer-Verlag, 1991.

[7] D. Harel and A. Pnueli, "On the development of reactive systems," in *Logics and Models of Concurrent Systems* (K. R. Apt, ed.), NATO ASI series F-13, pp. 477–498, Springer-Verlag, 1985.

[8] N. Leveson, M. Heimdahl, H. Hildreth, and J. D. Reese, "Requirements specification for process control systems," *IEEE Transactions on Software Engineering*, vol. 20, no. 9, pp. 1270–1282, 1994.

[9] F. Maraninchi and Y. Remond, "Argos: an automaton-based synchronous language," *Computer languages*, vol. 27, pp. 61–92, 2001.

[10] P. LeGernic, T. Gautier, M. LeBorgne, and C. LeMarie, "Programming real-time applications with signal," *Proceedings of the IEEE*, vol. 79, no. 9, pp. 1283–1304, 1991.

[11] G. Berry and G. Gonthier, "The esterel synchronous programming language: Design, semantics, implementation," *Science of Computing Programming*, vol. 19, no. 2, pp. 83–152, 1992.

[12] T. Jeron, H. Marchand, V. Rusu, and V. Tschaen, "Ensuring the conformance of reactive discrete-event systems using supervisory control," *Proceedings of the 42th IEEE Conference on Decision and Control*, pp. 2692–2697, 2003.

[13] S. Balemi, "Input/output processes and communication delays," *Discrete-Event Dynamic Systems: Theory and Application*, vol. 4, pp. 41–85, 1992.

[14] L. Du, S. L. Ricker, and P. Gohari, "Decentralized supervisory control and communication for reactive discrete-event systems," *Proceedings of the 2006 American Control Conference*, pp. 6045–6050, 2006.

[15] K. Inan, "Supervisory control and formal methods for distributed systems," *Proceedings of International Workshop on Discrete-Event Systems (WODES'92)*, pp. 29–41, 1992.

[16] K. Inan, "Nondeterministic supervision under partial observations," *Guy Cohen and Jean-Pierre Quadrat Ed., Lecture Notes in Control and Information Sciences*, vol. 199, pp. 39–48, 1994.

[17] K. Calvert and S. S. Lam, "Formal methods of protocol conversion," *IEEE Journal on Selected Areas in Communication*, vol. 8, no. 1, pp. 127–142, 1990.

[18] R. Kumar, S. Nelvagal, and S. I. Marcus, "A discrete event systems approach for protocol conversion," *Discrete-Event Dynamic Systems: Theory and Application*, vol. 7, no. 3, pp. 295–315, 1997.

[19] P. J. Ramadge and W. M. Wonham, "Supervisory control of a class of discrete event processes," *SIAM Journal on Control and Optimization*, vol. 25, no. 1, pp. 206–230, 1987.

[20] W. M. Wonham and P. J. Ramadge, "On the supremal controllable sun;anguage of a given language," *SIAM Journal on Control and Optimization*, vol. 25, no. 3, pp. 637–659, 1987.

[21] R. Cieslak, C. Desclaux, A. Fawaz, and P. Varaiya, "Supervisory control of discrete-event processes with partial observations," *IEEE Transactions on Automatic Control*, vol. 33, pp. 249–260, 1988.

[22] K. Rudie and W. M. Wonham, "Think globally, act locally: Decentralized supervisory control," *IEEE Transactions on Automatic Control*, vol. 37, no. 11, pp. 1692–1708, 1992.

[23] F. Lin and W. Wonham, "On observability of discrete-event systems," *Information Sciences*, vol. 44, no. 3, pp. 173–198, 1988.

[24] P. J. Ramadge and W. M. Wonham, "The control of discrete event systems.," *Proceedings of IEEE; Special issue on Dynamics of Discrete Event Systems.*, vol. 77, no. 1, pp. 81–98, 1989.

[25] K. Rudie and J. C. Willems, "The computational complexity of decentralized discrete-event control problems," *IEEE Transactions on Automatic Control*, vol. 40, no. 7, pp. 1313–1319, 1995.

[26] K. Rudie and W. Wonham, "Supervisory control of communicating processes," *L. Logrippo, R. L. Probert and H. Ural, Ed., Protocol Specification, Testing and Verification*, vol. X, pp. 243–257,, 1990.

[27] K. Rudie and W. Wonham, "Protocol verification using discrete-event systems," *Proceedings of the 31th IEEE Conference on Decision and Control*, vol. 3, pp. 3770–3777, 1992.

[28] S. Takai, A. Takae, and S. Kodama, "The extremal languages in supervisory control of discrete event systems with service specifications," *Proceedings of the 35th IEEE Conference on Decision and Control*, vol. 2, pp. 2231–2236, 1996.

[29] S. L. Ricker and K. Rudie, "Incorporating communication and knowledge into decentralized discrete-event systems," *Proceedings of the 38th IEEE Conference on Decision and Control*, pp. 1326–1332, 1999.

[30] R. J. Wieringa, *Design methods for reactive systems: Yourdon, Statemate, and the UML*. The Morgan Kaufmann Series in Software Engineering and Programming, Morgan Kaufmann Publishers Inc, 2006.