Friedrich-Alexander-Universität Erlangen-Nürnberg
Lehrstuhl für Regelungstechnik
Prof. Dr.-Ing. G. Roppenecker Prof. Dr.-Ing. T. Moor

Diplomarbeit

Hierarchical Design of Discrete Event Controllers: An Automated Manufacturing System Case Study

Als Diplomarbeit
vorgelegt von

Sebastian Perk

Betreuer:	Betreuer:	Ausgabedatum: 21.06.04
Dipl.-Ing. Klaus Schmidt	Prof. Dr.-Ing. T. Moor	Abgabedatum: 21.12.04

Sebastian Perk

Hierarchical Design of Discrete Event Controllers: An Automated Manufacturing System Case Study

Aufgabenstellung:

In this thesis, a hierarchical method for the control of DES shall be applied to the Fischertechnik simulation model of the Institute of Control and Automation, University of Erlangen-Nuremberg.

As a first step, the components of the plant shall be modeled according to a modeling scheme which has already been applied to parts of the plant. This modeling scheme abstracts from the local control action and captures the plant progress. Also, rather than modeling the plant on the whole, it exploits the structural properties of the plant by generating models of decentralized subplants.

Based on the system model, the hierarchical control method from [Schmidt04] shall be applied. This involves the investigation of two system properties. The mutual controllability property, which has already been verified for parts of the system must be checked for the overall plant and the shared event marking condition does not hold for the subplants of the simulation model. Thus weaker conditions replacing the shared event marking condition but still guaranteeing nonblocking behavior shall be investigated.

Es wird ausdrücklich auf die „Richtlinien zur Anfertigung von Studien- und Diplomarbeiten“ hingewiesen.

(Prof. Dr.-Ing. T. Moor)

Erklärung

"When the only tool you own is a hammer,
every problem begins to resemble a nail."
Abraham Maslow (1908-1970)
American Psychologist

Contents

Chapter 1

Introduction

Over the past years several new methods for the control of discrete event systems based on the framework provided by P.J. Ramadge und W.M. Wonham [18] and the supervisory control theory [2] have been developed.

Up to now, most of the research efforts in this context have been spent on reducing the complexity of the supervisor synthesis, which becomes enormous when dealing with discrete event systems of relevant sizes.

Modular and decentralized approaches like [8, 9, 19, 6, 11, 10, 17] focus on describing the discrete event system as a compound of subsystems of manageable size with the aim to distribute the supervisor synthesis to decentralized supervisors or to avoid the composition of the subsystems. A second promising method is to postpone the supervisor synthesis to a superordinate level containing a less complex image of the detailed system model, see for example [7, 20, 3, 15, 5, 16].

In [13], a new approach for the control of a class of decentralized discrete event systems has been presented that combines the hierarchical and decentralized method. Certain system properties, local nonblocking and marked state acceptance, have been identified that guarantee consistency of the hierarchical architecture and nonblocking behavior of the controlled system.

This above-mentioned approach is extended and modified in this thesis. The restrictive local nonblocking condition is replaced by the so-called single-event controllability which covers praxis relevant cases. Also, a modeling procedure is developed which makes it possible to refine detailed low-level plant models by additional information for the abstraction to a higher level. Furthermore a particular supervisor implementation guarantees nonblocking behavior of the closed loop control system. The modeling and design method allows for automatic PLC code generation.

The outline of this work is as follows. An introduction to control of discrete event systems is given in Chapter 2, where the framework of the supervisory control theory is explained. Chapter 3 deals with the embedding of the supervisory control theory in a hierarchical control architecture and moreover the application to decentralized discrete event systems. In Chapter 4, an extension and modification of the hierarchical and decentralized control system presented in [13] is introduced in the form of a decentralized multi-level hierarchy. The notion of refinement automata is presented, which extend the decentralized system models by refinement events that for example abstract a complex task to be reported to the next level in the hierarchy. Furthermore, the property of local nonblocking is replaced by the practically relevant condition of single event controllability in combination with a modified supervisor implementation in the low-level. The modified and extended approach is applied to an automated manufacturing example, which is the Fischertechnik model of an industrial production process controlled by a standard programmable logic controller (PLC) in Chapter 5. The thesis concludes with a prospect to possible future research activities.

Chapter 2

Basics of Control of Discrete Event Systems

The following chapter will provide a brief introduction to discrete event systems (DES) and the supervisory control theory (SCT). More detailed explanations are given in [4].

2.1 Discrete Event Systems

In [18] W. M Wonham has given a framework for modeling the behavior of DES using *finite automata*. For this work, only *deterministic* finite automata are relevant.

Definition 2.1.1 (Deterministic Finite Automaton) *A deterministic finite automaton is a 5-Tupel $G := (X, \Sigma, \delta, x_0, X_m)$ consisting of*

- X*: the finite set of states*
- Σ*: the finite set of events (also referred to as alphabet) including the* empty event ε
- $\delta : X \times \Sigma \rightarrow X$ *the partial transition function* [1]
- x_0*: the initial state* [2]
- $X_m \subseteq X$*: the set of marked states*

[1]As G is deterministic, the function δ is unique.
[2]As G is deterministc, there exists only one initial state.

Any concatenation of events is called a *string*. The set of all strings over Σ is defined as Σ^*. The transition function δ is partial as it is defined for a subset of Σ only at any state $x \in X$. Given $x_1, x_2 \in X$ and $\sigma \in \Sigma$, $\delta(x_1, \sigma)!$ says that this transition is defined, and $\delta(x_1, \sigma) = x_2$ describes the transition from one state $x_1 \in X$ to its successor state $x_2 \in X$ driven by an event $\sigma \in \Sigma$.

δ can be extended to a partial transition function on $X \times \Sigma^*$ by the following recursive definition: $\delta(x, \varepsilon) := x$ and $\delta(x, s\sigma) := \delta(\delta(x, s), \sigma)$, whenever there exists $x' = \delta(x, s)$ and $\delta(x', \sigma)$ is defined.

The set of all events possible in a state x is called *active event set* $\Lambda(x) := \{\sigma \in \Sigma \mid \delta(x, \sigma)!\}$. Figure 2.1 shows the graph of an example automaton with $X = \{1,2,3,4\}$, $\Sigma = \{\sigma_1, \sigma_2, \sigma_3, \sigma_4\}$, $\delta(1, \sigma_1) = 2$, $\delta(2, \sigma_2) = 3$, $\delta(2, \sigma_3) = 4$, $\delta(4, \sigma_4) = 1$, $x_0 = 1$ and $X_m = \{3\}$.

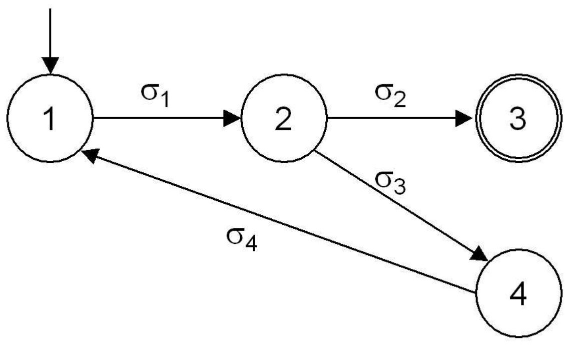


Figure 2.1: Example of a finite automaton

A finite automaton is the *Generator* of the language $L(G)$ and *marks* the language $L_m(G)$ as described in the subsequent definition.

Definition 2.1.2 (Generated and Marked Language) *For a given automaton* $G = (X, \Sigma, \delta, x_0, X_m)$ *the generated language is defined as*
$L(G) := \{s \in \Sigma^* \mid \delta(x_0, s)!\}$
and the marked language is
$L_m(G) := \{s \in \Sigma^* \mid \delta(x_0, s) \in X_m\}$.

So $L(G)$ contains all strings which can occur in G, and $L_m(G)$ contains the strings that lead to a marked state in G.

The description of languages by finite automata is not unique, i.e. the same language can be marked by a variety of different automata. The automaton that marks a given language

L_m with a minimum number of states is called the *canonical recognizer* of L_m.[3] For the rest of this work the notation "finite automaton" shall be restricted to the canonical recognizer of the corresponding marked language.
The example automaton of Figure 2.1 marks $L_m(G) = \{\sigma_1(\sigma_3\sigma_4\sigma_1)^*\sigma_2\}$.
The language H that contains all strings of H and all prefixes of these strings is called the *prefix-closure* of H:

$$\overline{H} := \{s \in \Sigma^* \mid \exists t \in (\Sigma)^* \text{ such that } st \in H\}$$

Every language generated by a finite automaton is prefix-closed, i. e. $L(G) = \overline{L(G)}$.
If there exist strings in $L(G)$ that are not prefix of a marked string, the generator of $L(G)$ is called *blocking* because there are states in G from which there is no path leading to a marked state. An automaton is *nonblocking* if it generates a language $L(G)$ that satisfies the following condition.

$$L(G) = \overline{L_m(G)}$$

This means that every string in $L(G)$ is prefix of a string in $L_m(G)$, i.e. a string that leads to a marked state.
An important tool for the manipulation of languages is the *natural projection*.

Definition 2.1.3 (Natural Projection) *Given $\Sigma_i \subseteq \Sigma$, the natural projection $p_i : \Sigma^* \rightarrow \Sigma_i^*$ is defined by the following recursion:*

1. $p_i(\varepsilon) := \varepsilon$
2. $p_i(s\sigma) := \begin{cases} p_i(s)\sigma \text{ if } \sigma \in \Sigma_i \\ p_i(s) \text{ otherwise.} \end{cases}$ *Where $\sigma \in \Sigma$ and $s \in \Sigma^*$.*

The projection can be applied to all strings of a language $K \in \Sigma^*$. Thus the above definition can be extended to languages as follows:

$$p_i(K) := \{t \in \Sigma_i^* : \exists s \in K \text{ with } p_i(s) = t\}$$

The projection $p_i(K)$ of a language $K \subseteq \Sigma^*$ reduces all its strings to strings contained in Σ_i^*. Accordingly the *inverse projection* p_i^{-1} of a string $t \in \Sigma_i^*$ returns all strings whose projection is given by the string t.

[3]Note that this automaton always exists.[18]

Definition 2.1.4 (Inverse Projection) *Given $\Sigma_i \subseteq \Sigma$, $t \in \Sigma_i^*$, the inverse projection $p_i^{-1} : \Sigma_i^* \longrightarrow 2^{\Sigma^*}$ is defined :*

$$p_i^{-1}(t) := \{s \in \Sigma^* \mid p_i(s) = t\}$$

So the inverse projection of a language $K_i \subseteq \Sigma_i^*$ is

$$p_i^{-1}(K_i) := \{s \in \Sigma^* : \exists t \in K_i \text{ with } p_i(s) = t\}$$

It is a useful and necessary procedure in the modelling process to regard systems as a structure composed of functional modules (subsystems) of manageable size and then describe the concurrent behavior of these modules by composing them correctly. To describe the concurrent behavior of two DES we use the *synchronous product*[4] of two finite automata.

Definition 2.1.5 (Synchronous Product of Finite Automata) *The synchronous product of two given finite automata $G_1 = (X_1, \Sigma_1, \delta_1, x_{0,1}, X_{m,1})$ and $G_2 = (X_2, \Sigma_2, \delta_2, x_{0,2}, X_{m,2})$ is*

$$G_1||G_2 := (X_1 \times X_2, \Sigma_1 \cup \Sigma_2, \delta, (x_{0,1}, x_{0,2}), X_{m,1} \times X_{m,2})$$

with

- $X_1 \times X_2 := \{(x_1 \in X_1, x_2 \in X_2) : \exists s \in (\Sigma_1 \cup \Sigma_2)^* \mid \delta((x_{01}, x_{02}), s) = (x_1, x_2)\}$
- $\delta((x_1, x_2), \sigma) := \begin{cases} (\delta_1(x_1, \sigma), \delta_2(x_2, \sigma)) & \textit{if} \quad \sigma \in \Lambda_1(x_1) \cap \Lambda_2(x_2) \\ (\delta_1(x_1), x_2) & \textit{if} \quad \sigma \in \Lambda_1(x_1) - \Sigma_2 \\ (x_1, \delta_2(x_2)) & \textit{if} \quad \sigma \in \Lambda_2(x_2) - \Sigma_1 \\ \textit{undefined} & \textit{else} \end{cases}$
- $X_{m1} \times X_{m2} := \{(x_1, x_2) \in X_1 \times X_2 \mid x_1 \in X_{1m} \wedge x_2 \in X_{2m}\}$

This means that a shared event can happen at a state of the resulting automaton only if it is in the active event set of both of the respective states of G_1 and G_2 (synchronization), while the rest of events can happen whenever they are generated by G_1 or G_2. A state of the resulting automaton is marked only if both respective states of G_1 and G_2 are marked. The language generated by the synchronous product of two automata $L(G_1||G_2)$ is the synchronous product of the generated languages $L(G_1)||L(G_2)$.

[4]also referred to as *parallel composition*

Definition 2.1.6 (Synchronous Product of Languages) *Given $\Sigma = \Sigma_1 \cup \Sigma_2,\ p_i : \Sigma^* \rightarrow \Sigma_1^*,\ p_2 : \Sigma^* \rightarrow \Sigma_2^*$ the synchronous product of two languages $L_1 \subseteq \Sigma_1^*$ and $L_2 \subseteq \Sigma_2^*$ is:*

$$L_1||L_2 := p_1^{-1}(L_1) \cap p_2^{-1}(L_2)$$

The intersection in Definition 2.1.6 means that two given automata are synchronized by *shared events* $\Sigma_{shared} = \Sigma_1 \cap \Sigma_2$, while the rest of events is generated asynchronously, which is caused by the inverse projections.
The following section will briefly describe how a DES described by a finite automaton can be controlled in accordance with a given specification.

2.2 Supervisory Control Theory

The idea of the supervisory control theory (SCT) is to reduce the uncontrolled behavior of a plant G so that the resulting closed-loop behavior satisfies a specification E given as a language or its generator. This is done by a supervisor S that observes events generated by the plant and prevents certain events from occuring.

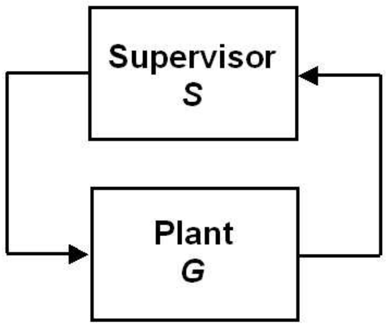


Figure 2.2: Plant G controlled by supervisor S

The event set Σ of the plant is therefore divided into two disjoint subsets: controllable events Σ_c, those that can be prevented from occuring (e.g. actuator signals), and uncontrollable events Σ_{uc} that cannot be prevented by the supervisor (e.g. sensor signals).

$$\Sigma = \Sigma_c \cup \Sigma_{uc}$$

The supervisor is a map of all strings that can happen in G (which is the language generated by G) to the events allowed by S after occurence of these strings:

$$S : L(G) \rightarrow \Gamma$$

where $\Gamma \subseteq 2^{\Sigma}$ is the set of all *control patterns* γ, $\Sigma_{uc} \subseteq \gamma \subseteq \Sigma$, and $S(s) \subseteq \Sigma$ denotes the set of events enabled by S after the occurence of the string s. A supervisor is *admissible* if it prevents only controllable events and allows uncontrollable events whenever they are possible in G. By supervisory control the language $L(G)$ is reduced to a sublanguage $L(S/G)$[5]. A sublanguage K of $L(G)$ that can be achieved by an admissible supervisor is said to be *controllable with respect to* $L(G)$.

Definition 2.2.1 (Controllability) *[18] A language K is controllable with respect to $L(G)$ and Σ_{uc} if*

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

This means that any prefix s of K extended with an uncontrollable event that is possible after s in $L(G)$ is still prefix of K, i.e. an uncontrollable event possible in $L(G)$ is never prevented in K. The *set of all controllable sublanguages* of $L(G)$ is denoted by $\mathcal{C}(L(G))$.

$$\mathcal{C}(L(G)) := \{H \subseteq L(G) \mid \overline{H}\Sigma_{uc} \cap L(G) \subseteq \overline{H}\}$$

The definition of controllability can be extended by the introduction of *forcible events*. If an uncontrollable event σ_{uc} can happen in a state of the plant (think of a sensor signal), but it is granted that a second event σ_f in the same state can generally happen *before* the respective uncontrollable event, then this second event is forcible. Thus a supervisor is still admissible if it prevents σ_{uc} by forcing σ_f. With the partition of the event set to forcible and unforcible events $\Sigma = \Sigma_f \cup \Sigma_{uf}$ (Σ_f and Σ_{uf} are disjoint), definition 2.2.1 can be extended as follows.

Definition 2.2.2 (Extended Controllability) *Given $\Sigma_{uc,f} := \Sigma_{uc} \cap \Sigma_f$ and $\Sigma_{uc,uf} := \Sigma_{uc} \cap \Sigma_{uf}$, a language K is extended controllable with respect to $L(G)$ and Σ_{uc} if:*

- $\overline{K}\Sigma_{uc,f} \cap L(G) \subseteq \overline{K}$
- $\forall s \in K, \forall \sigma_{uc,uf}:$ *if* $s\sigma_{uc,uf} \in L(G):$
  $(s\sigma_{uc,uf} \cap L(G) \in \overline{K}) \vee (\exists \sigma_f \in \Sigma_f$ *such that* $s\sigma_f \cap L(G) \in \overline{K})$

This definition of controllability shall be used for the rest of this work. As a consequence, the set of all controllable sublanguages of $L(G)$ is redefined as

$$\mathcal{C}(L(G)) = \{H \subseteq L(G) \mid H \; is \; extended \; controllable \; with \; respect \; to \; L(G)\}$$

[5] S/G can be read as "S controlling G"

For a given specification language E the *supremal controllable sublanguage* is the minimal restrictive behavior, which can be achieved by an admissible supervisor.

Definition 2.2.3 (Supremal Controllable Sublanguage) *The supremal controllable sublanguage of E with respect to $L(G)$ is*

$$\kappa_{L(G)}(E) := \cup\{K \in \mathcal{C}(L(G)) \mid K \subseteq E\}$$

So $\kappa_{L(G)}(E)$ is the union of all controllable sublanguages of $L(G)$ that do not violate the specification E.[6] A minimal restrictive supervisor can be derived from a recognizer of $\kappa_{L(G)}(E)$.
For nonblocking control, the following additional condition for the closed-loop behavior is necessary.

Definition 2.2.4 ($L_m(G)$-Closure) *A language K is $L_m(G)$-closed if*

$$K = \overline{K} \cap L_m(G)$$

This means that every string in K is prefix of a string in $L_m(G)$. The set of all $L_m(G)$-closed languages is denoted by $\mathcal{F}_{L_m(G)}$.
The conditions for the existence of an admissible and nonblocking supervisor are given in the following theorem [2].

Theorem 2.2.1 (Nonblocking controllability Theorem) *Let G be a DES with $\Sigma_{uc} \subseteq \Sigma$ and a sublanguage $K \subseteq L_m(G)$ and $\overline{K} \neq \emptyset$. A nonblocking supervisor with $L_m(S/G) = K$ and $L(S/G) = \overline{K}$ exists iff*

1. *K is controllable*
2. *K is $L_m(G)$-closed*

From the above conditions, algorithms have been derived to compute a minimal restrictive nonblocking supervisor for a plant and a given specification. While the plant can be described by modules of manageable size, the remaining problem is the explosion of the number of states of the entire plant, which can grow exponentially when composing the modules.
One way of dealing with that problem is to reduce the complexity of supervisor synthesis

[6] Also note that this supremum always exists as controllability is closed under union.

by distributing the control problem to several specifications and several respective supervisors that control the monolithic plant in parallel. In this case, called *modular control of DES*, the controlled plants have to be *nonconflicting* to ensure that the concurrent control by all supervisors does not cause a blocking.[4].
The computational complexity can also be reduced by avoiding the parallel composition of all modules and instead computing several supervisors, each controlling only a part of the system, which consists of only a small number of modules. This approach is called *decentralized control of DES*. Also in this case, conflicting behavior of the controlled parts of the plant has to be avoided. Additionally, decentralized supervisory control might not be minimal restrictive because of controllability conflicts.[8]
A further approach is to postpone the composition of subsystems to a higher level with abstracted and less complex models of the subsystems. We then speak of *hierarchical control of DES*. It has to be ensured that supervisory control of the high-level is transferred correctly to the low-level. One condition guaranteeing this requirement is *hierarchical consistency*.
This work considers a combination of the last two approaches, which will be described in the following chapter.

Chapter 3

Hierarchical Control of Decentralized Discrete Event Systems

In this chapter, a new approach for hierarchical control for structural dezentralized DES by Dipl.-Ing. Klaus Schmidt presented in [13] will be explained.
The first section describes the abstraction of a plant to an abstracted high-level, for which a high-level supervisor is developed along with the correct low-level implementation of the high-level supervisor.
An approach to control of structural decentralized DES is explained in section 3.2, followed by the combination of both, the hierarchical and the decentralized method in Section 3.3.

3.1 Hierarchical Control of DES

The basic idea of a hierarchical approach to control of discrete event systems is to introduce a control structure that consists of several hierarchical levels in order to reduce the overall complexity of control design.

3.1.1 Hierarchical Supervisor Implementation

The standard scheme for the hierarchical control of DES was introduced in [20] and is shown in Figure 3.1.

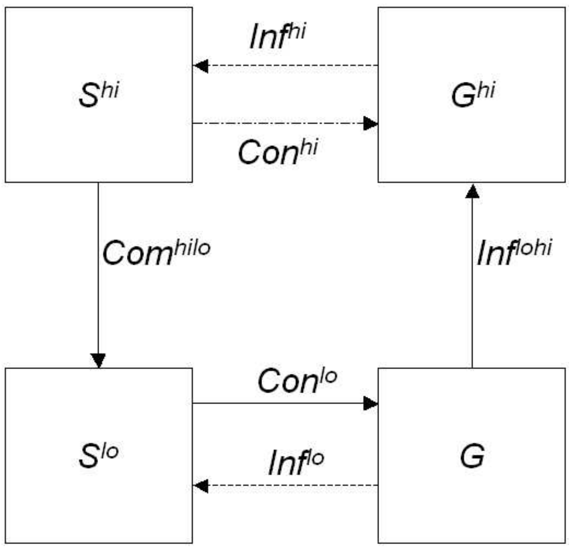


Figure 3.1: Scheme for hierarchical control of DES

G is abstracted to a less complex high-level model G^{hi} which is driven only by relevant events generated by G and reported via Inf^{lohi}. A high-level supervisor observes the signals Inf^{hi} in G^{hi}. As G^{hi} is just an image of the plant G, the control action of S^{hi} is not imposed directly on G^{hi} via the virtual signal path Con^{hi} but on G via S^{Lo}. The original detailed plant model G is controlled by a low-level supervisor S^{lo} that observes events Inf^{lo} generated by G and imposes control on G via Con^{lo}. Usually, the forward signals Com^{hilo} and Con^{lo} are designated as "command and control", while the feedback paths Inf^{lohi} and Inf^{hi} are referred to as "report and advise".
The abstraction of G to a high-level model G^{hi} is defined as follows.

Definition 3.1.1 (Hierarchical Abstraction) *Given a DES $G = (X, \Sigma, \delta, x_0, X_m)$ and the set of high-level events $\Sigma^{hi} \subseteq \Sigma$, a hierarchical abstraction is the tupel (G, θ^{hi}, G^{hi}).*
The reporter map $\theta : \Sigma^* \to (\Sigma^{hi})^*$ is defined as

1. $\theta(\varepsilon) = \varepsilon$
2. $\theta(s\sigma) = \begin{cases} \theta(s) \text{ or} \\ \theta(s)\sigma^{hi} \end{cases}$

where $s \in \Sigma^*, \sigma \in \Sigma$ and $\sigma^{hi} \in \Sigma^{hi}$.
The map of the low-level language is the high-level language denoted by $L^{hi} := \theta(L(G))$.
The high-level marking is given by a regular language $L^{hi}_m \subseteq L^{hi}$.
Accordingly, the high-level plant G^{hi} is the canonical recognizer that generates $L(G^{hi}) = L^{hi}$ and marks $L_m(G^{hi}) = L^{hi}_m$.

According to [14, 13], from now on the reporter map is implemented as the natural projection of Definition 2.1.3 to high-level strings.

$$\theta := p^{hi},\ p^{hi} : \Sigma^* \to (\Sigma^{hi})^*$$

The set of high-level events is partitioned into controllable and uncontrollable events $\Sigma^{hi} = \Sigma_c^{hi} \cup \Sigma_{uc}^{hi}$ which are disjoint $\Sigma_c^{hi} \cap \Sigma_{uc}^{hi} = \emptyset$.
With the above notations a *hierarchical control system* can be formally defined.

Definition 3.1.2 (Hierarchical Control System) *A hierarchical control system (HCS) is a structure $(G, p^{hi}, G^{hi}, S^{hi}, S^{lo})$ for which the following properties hold:*

- *High-level control patterns:* $\Gamma^{hi} := \{\gamma \mid \Sigma_u^{hi} \subseteq \gamma \subseteq \Sigma^{hi}\}$
- *High-level supervisor:* $S^{hi} : L^{hi} \to \Gamma^{hi}$
- *Low-level supervisor:* $S^{lo} : L(G) \to \Gamma$
- *Validity of S^{lo}:*

  $$p^{hi}(L(S^{lo}/G)) \subseteq L(S^{hi}/G^{hi})$$

  i.e. S^{lo} does not permit behavior that is not allowed by S^{hi}.

With the conventional procedures described in Chapter 2 and a given L_m-closed specification $E^{hi} \in \mathcal{F}_{L_m^{hi}}$, a nonblocking high-level supervisor can now be synthesized. $L(S^{hi}/G^{hi})$ represents the *desired* high-level closed-loop behavior that would result if S^{hi} could control G^{hi} directly via Con^{hi}.
The main step in hierarchical control of DES is to implement the control actions of S^{hi} imposed on S^{lo} such that S^{lo} is valid according to Definition 3.1.2 and nonblocking. This task is called the *hierarchical control problem*.

Definition 3.1.3 (Hierarchical Control Problem) *Given $(G, p^{hi}, G^{hi}, S^{hi})$, find a valid low-level supervisor S^{lo} such that the closed-loop behavior $L(S^{lo}/G)$ of the hierarchical control system is nonblocking.*

To find such a nonblocking low-level supervisor, the validity condition of Definition 3.1.2 is not sufficient, as it might end up with a strict subset relation $p^{hi}(L(S^{lo}/G)) \subset L(S^{hi}/G^{hi})$, which means e.g. that nonempty low-level controlled behavior can not be guaranteed. A way to avoid that problem is to restrict the validity condition to *hierarchical consistency*.

Definition 3.1.4 (Hierarchical Consistency) *A hierarchical control system (G, p^{hi}, G^{hi}, S^{hi}) is hierarchically consistent if it meets the following condition:*

$$p^{hi}(L(S^{lo}/G)) = L(S^{hi}/G^{hi})$$

This means that the control action on G^{hi} imposed indirectly via Com^{hilo}, Con^{lo} and Inf^{lohi} exactly complies with the virtual control action of S^{hi} via Con^{hi}. So hierarchical consistency is a powerful tool to show that the low-level controlled behavior is nonblocking.
A hierarchically consistent low-level implementation S^{lo} is the *standard supervisor implementation* presented in [14] and [13].[1]

Definition 3.1.5 (Standard Supervisor Implementation) *Given a hierarchical control system according to Definition 3.1.2, a standard supervisor implementation is*

$$S^{lo}(s) := S^{hi}(p^{hi}(s)) \cup (\Sigma - \Sigma^{hi}),\ \forall s \in L(G)$$

This implementation rule means that the low-level implementation of S^{hi} can disable high-level events only, while low-level events $\Sigma - \Sigma^{hi}$ are always enabled.
However, the low-level closed-loop behavior $L(S^{lo}/G)$ might still be blocking. This is because in $L(S^{lo}/G)$ there might exist low-level strings $s \in (\Sigma - \Sigma^{hi})^*$ that can neither be extended to a string ending with a high-level event nor to a string contained in $L_m(G)$, i.e. there exist blockings that are not "noticed" by G^{hi}. In the subsequent section, properties of the plant G are identified that guarantee nonblocking behavior under control of S^{lo}.[13]

3.1.2 Properties of the low-level plant

Before defining the properties of the low-level plant it is useful to introduce some definitions concerning the behavior of the low-level plant G.
A particular set of low-level strings that correspond to high-level strings is the set of *entry strings*.

Definition 3.1.6 (Entry Strings) *Given $s^{hi} \in L^{hi}$, the set of entry strings of s^{hi} is defined as*

$$L_{en,s^{hi}} := \{s \in L(G) \mid p^{hi}(s) = s^{hi} \wedge \nexists s' < s \text{ such that } p^{hi}(s') = s^{hi}\}$$

[1]Remember that the reporter map θ is implemented as the natural projection to high-level events p^{hi}.

Let $\sigma^{hi} \in \Sigma^{hi}$ be the last event of s^{hi}, then $L_{en,s^{hi}}$ consists of all low-level strings corresponding to s^{hi} that end with the event σ^{hi}, i.e. these low-level strings do not have a prefix with the same high-level projection s^{hi}.
The set of *exit strings* of s^{hi} contains all low-level strings corresponding to s^{hi} that can be extended with a next high-level event.

Definition 3.1.7 (Exit Strings) *Given $s^{hi} \in L^{hi}$, the set of exit strings of s^{hi} is defined as*

$$L_{s^{hi},ex} := \{s \in L(G) \mid p^{hi}(s) = s^{hi} \wedge \exists \sigma^{hi} \in \Sigma^{hi} \text{ such that } s\sigma^{hi} \in L(G)\}$$

The following language is introduced to collect all strings of low-level events starting from an exit string and possibly ending with a high-level event.

Definition 3.1.8 (Local exit string extensions) *For $s \in L_{s^{hi},ex}$, the set of local exit string extensions is*

$$L_{s,s^{hi}} := \{u\sigma \mid su\sigma \in L(G) \ \wedge \ p^{hi}(su) = s \ \wedge \ \sigma \in \Sigma\} \subseteq \Sigma^*$$

So this language contains all low-level strings starting from an exit string s with $p^{hi}(s) = s^{hi}$ which can be extended with an event that is possibly a high-level event.
The system property defined next deals with *local blocking*. For the explanation of local blocking consider the example of a low-level plant G in Figure 3.2 with $\Sigma = \{a, b, x, y\}$ and $\Sigma^{hi} = \{x, y\}$.

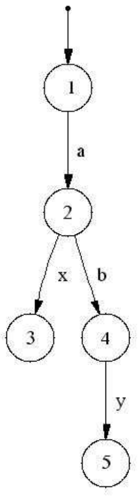


Figure 3.2: locally blocking low-level plant G

The language generated by G is $L(G) = \{a, ab, ax, aby\}$. The abstraction of G to the high-level shown in Figure 3.3 generates the language $L^{hi}(G^{hi}) = p^{hi}(L(G)) = \{x, y\}$.

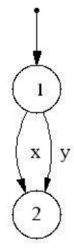


Figure 3.3: high-level abstraction of G

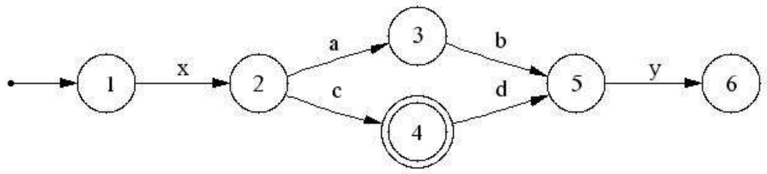
The set of exit strings for $s^{hi} = \varepsilon$ is $L_{\varepsilon,ex} = \{a, ab\}$. If y is disabled by a high-level supervisor, then x can happen instead in G^{hi}. But as S^{lo} has to enable all low-level events, string ab is allowed in G which can not be extended by x. This is called a local blocking.
This problem never occurs, if all exit strings belonging to a high-level string s^{hi} can always be extended with any event that is possible after s^{hi} in the high-level.
The following definition provides a system property that deals with the problem described above.

Definition 3.1.9 (Locally Nonblocking Hierarchical Abstraction)
Let (G, p^{hi}, G^{hi}) be a a hierarchical abstraction. The high-level string $s^{hi} \in \overline{L_m^{hi}}$ is said to be locally nonblocking if
$\forall s \in L(G)$ *with* $p^{hi}(s) = s^{hi}$ *and* $\forall \sigma \in \Sigma^{hi}(s^{hi})$ *with* $p^{hi}(s)\sigma \in \overline{L_m^{hi}}$*:*

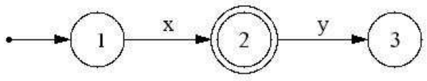
$$\exists u_\sigma \in (\Sigma - \Sigma^{hi})^* \text{ such that } su_\sigma\sigma \in L(G)$$

(G, p^{hi}, G^{hi}) *is locally nonblocking if the above condition holds* $\forall s^{hi} \in \overline{L_m^{hi}}$*.*

So all strings s in the low-level which project to s^{hi} can always be extended with local strings u leading to *any* high-level event possible in the high-level after s^{hi}. While in the antecendent example all exit strings have to be considered, the above definition generally refers to all strings $s \in L(G)$.
The subsequent condition is introduced to preserve a consistent marking when abstracting G to the high-level. Therefore consider the example plant G of Figure 3.4 with the set of high-level events $\Sigma^{hi} = \{x, y\}$.



(a) G



(b) G^{hi}

Figure 3.4: Non marked state accepting abstraction

While in the high-level $G^{hi} = p^{hi}(G)$ the event x leads to a marked state, in G after the occurence of string xa a marked state is never reached. This problem is avoided, if all exit strings of $s^{hi} = x$ have a marked predecessor string which projects to the same high-level string.[13]

Definition 3.1.10 (Marked State Acceptance)
Let (G, p^{hi}, G^{hi}) be a hierarchical abstraction. (G, p^{hi}, G^{hi}) is marked state accepting if

$$\forall s_m^{hi} \in L_m^{hi}, \forall s \in L_{s_m^{hi},ex} : \exists s' \leq s \text{ with } p^{hi}(s') = s_m^{hi} \text{ and } s' \in L_m$$

This means that every exit string corresponding to a marked high-level string has a marked prefix with the same projection to the high-level, i.e. it has passed a marked state.
Given a hierarchical abstraction that is locally nonblocking and marked state accepting, it can be shown that the following lemma is valid.[13]

Lemma 3.1.1 *Let (G, p^{hi}, G^{hi}) be a hierarchical abstraction with $s \in L$ and $s^{hi} \in L^{hi}$. If (G, p^{hi}, G^{hi}) is locally nonblocking and marked state accepting, then it meets the following property:*

$$\forall t \in (\Sigma^{hi})^* \text{ with } s^{hi}t \in L^{hi} : \exists u \in \Sigma^* \text{ such that } su \in L(G) \wedge p^{hi}(u) = t$$

This means if a successor string t of s^{hi} is possible in the high-level G^{hi}, then it always can be implemented in the low-level G after the respective string s.
With that result one can verify the following theorem.

Theorem 3.1.1 *Let $(G, p^{hi}, G^{hi}, S^{hi}, S^{lo})$ be a hierarchical control system with a hierarchical abstraction (G, p^{hi}, G^{hi}) that is locally nonblocking and marked state accepting and a standard supervisor implementation S^{lo}.*
Then the hierarchical control system is hierarchically consistent and solves the hierarchical control problem, and the controlled low-level behavior is nonblocking.[13]

The hierarchical abstraction of a low-level plant can reduce the complexity of supervisor synthesis as the complexity of plant G is reduced to a less complex high-level plant. But it does not take into account the state explosion when the modules of a system of praxis relevant size are composed to a monolithic plant G at the low-level. We will see that if the hierarchical approach is applied to a decentralized control system, the composition can be postponed to the high-level and moreover the complexity of high-level supervisor synthesis can be reduced by additional local supervisors.
In the following section, the control of decentralized DES will be explained, before the hierarchical method and the decentralized method are combined in Section 3.3.

3.2 Control of Structural Decentralized DES

The structure of subsystems that is object of decentralized control is called a *decentralized control system*, which is defined as follows.

Definition 3.2.1 (Decentralized control system) *A decentralized control system (DCS) is composed of subsystems $G_i,\ i = 1, 2, \ldots, n$ with the respective event sets Σ_i. Each pair of subsystems G_i and G_j is synchronized by shared events, if $\Sigma_i \cap \Sigma_j \neq \emptyset$. The composition to the entire plant is given by $G = ||_{i=1}^{n} G_i$. The event set of G results in $\Sigma := \bigcup_{i=1}^{n} \Sigma_i$ and is partitioned into controllable and uncontrollable events $\Sigma = \Sigma_c \dot{\cup} \Sigma_{uc}$. Consequently, the controllable and uncontrollable individual event sets are $\Sigma_{i,c} := \Sigma_i \cap \Sigma_c$ and $\Sigma_i \cap \Sigma_{uc}$. For brevity and convenience, the following notation is introduced:*
$L(G_i) := L_i,\ L_m(G_i) := L_{i,m}$

Given local specifications $E_i \in \mathcal{F}_{L_{i,m}} \subseteq \Sigma_i^*, i = 1, \ldots, n$ each subsystem G_i is controlled by a local supervisor S_i, as shown in Figure 3.5.[14]

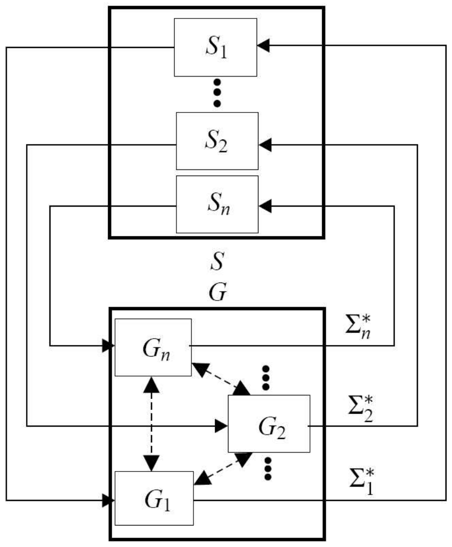


Figure 3.5: Decentralized control structure

With the local behavior of each controlled subsystem $L(S_i/G_i) = \overline{\kappa_{L_i}(E_i)}$, the *controlled behavior of the global system* results in

$$||_{i=1}^{n} L(S_i/G_i) = \bigcap_{i=1}^{n} (p_i)^{-1}(\overline{\kappa_{L_i}(E_i)}) \cap L$$

The main task of decentralized control is to guarantee that this behavior is nonblocking and complies with the minimally restrictive behavior that could be achieved by a monolithic approach of Chapter 2. The global formulation of the local specifications E_i is given by $E = \bigcap_{i=1}^{n} p^{-1}(E_i) \cap L$, with the natural projection $p_i : \Sigma \to \Sigma_i$. So the behavior that would result from control by a monolithic supervisor is $L(S/G) = \overline{\kappa_L(E)}$.
The compliance of the decentralized approach with the monolithic approach can be denoted by the following conditions [9]:

(1) $\bigcap_{i=1}^{n} (p_i)^{-1}(\overline{\kappa_{L_i}(E_i)}) = \overline{\kappa_L(E)}$
(2) $p_i(\overline{\kappa_L(E)})$ is nonblocking with respect to $L_{i,m}$.

The first condition says that there is no loss of optimality compared to the global supervisor synthesis, while condition number two means that nonblocking control of each subsystem is also nonblocking with respect to all remaining subsystems.
One way to fulfill these conditions is to guarantee the following properties concerning the structure of the subsystems.
The first property is required to avoid controllability conflicts when applying decentralized

control. This is the case, when a local supervisor disables uncontrollable behavior of the respective subsystem that can never happen because of the synchronization with the remaining subsystems, which means the loss of minimal restriction.

Definition 3.2.2 (Mutual Controllability) *Given a decentralized control system of definition 3.2.1 and the natural projection $p_i^{ij} : (\Sigma_i \cup \Sigma_j)^* \rightarrow \Sigma_i^*$, two local languages L_i and L_j are said to be mutually controllable if*

$$\overline{L_i}(\Sigma_{j,uc} \cap \Sigma_i) \cap p_i^{ij}((p_j^{ij})^{-1}(\overline{L_j})) \subseteq \overline{L_i}$$

The decentralized control system is mutually controllable, if the above property holds $\forall i, j = 1, 2, \ldots, n\, ; i \neq j$.

For understanding the meaning of this property, first we concentrate on the expression $p_i^{ij}((p_j^{ij})^{-1}(\overline{L_j}))$. The inverse projection takes into account that events that are in $\Sigma_i - \Sigma_j$ are generated in G_i asynchronously from G_j. The projection to events of Σ_i returns the behavior of G_j that is relevant for G_i. So mutual controllability says that L_i is controllable with respect to the behavior of G_j as seen from G_i.
The next property guarantees that that one subsystem never causes blocking in any of the remaining subsystems.

Definition 3.2.3 (Shared Event Marking) *Given a decentralized control system of definition 3.2.1, the property of shared event marking is defined as follows.*

$$\Sigma_i^*(\Sigma_i \cap \Sigma_j) \cap \overline{L_{i,m}} \subseteq L_{i,m}(\Sigma_i \cap \Sigma_j), \;\; \forall i, j = 1, 2, \ldots, n\, ; i \neq j$$

This means that in all subsystems G_i, shared events are generated in marked states only, which means that blocking can never be caused by the concurrent behavior of all subsystems.
In [9] it is shown that a decentralized control system with the above properties fulfills conditions (1) and (2) and therefore decentralized control can be applied without loss of optimality and without blocking.
The cost of this approach is that shared event marking requires that every task of each subsystem can be completed independently from the remaining subsystems, which means a considerable restriction to the models of the subsystems.
In the following section we will see that guaranteeing shared event marking is not necessary, if this approach is combined with hierarchical control, because then blocking caused by the concurrent behavior of the controlled subsystems is avoided by a nonblocking high-level supervisor.

3.3 Hierarchical Control of Structural Decentralized DES

The combination of the hierarchical control structure described in Section 3.1 and the decentralized approach explained in Section 3.2 was presented in [13] and is shown in the subsequent figure.

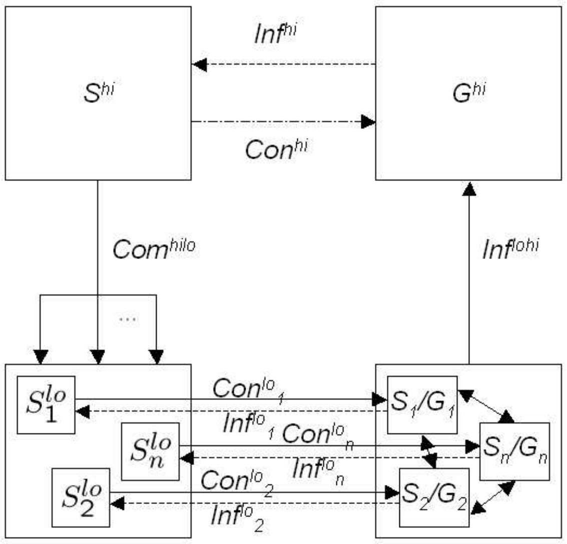


Figure 3.6: Hierarchical and decentralized control scheme

Each subsystem of the decentralized plant G is controlled by a local supervisor $S_i,\ i = 1,2,\dots,n$. The locally controlled concurrent behaviour of the subsystems is then abstracted to a high-level plant G^{hi}, for which a high-level supervisor S^{hi} is designed. The control action of S^{hi} is imposed on each locally controlled subsystem by the low-level supervisors S_i^{lo}.
This hierarchical and decentralized control architecture is defined as follows [13].

Definition 3.3.1 *A Hierarchical and Decentralized Control System (HDCS) is a structure that consists of the following components.*

- *The detailed low-level plant model G is given as a decentralized control system of definition 3.2.1.*
- *Low-level controllers $S_i\colon L_i \to \Gamma_i$, where Γ_i are the respective control patterns that achieve locally nonblocking low-level closed-loop languages denoted by*[2]
  $L_i^c := L(G_i^c) := L(S_i/G_i)$, $L_{i,m}^c := L_i^c \cap L_{i,m}$, $L^c := ||_{i=1}^n L_i^c$, $L_m^c := ||_{i=1}^n L_{i,m}^c = L^c \cap L_m$.

[2]the index "c" is to be read "controlled".

$G^c = ||_{i=1}^n G_i^c$ is the canonical recognizer of the globally resulting locally controlled behavior L^c such that $L^c = L(G^c)$, $L_m^c = L_m(G^c)$.

- *A hierarchical abstraction (G, p^{hi}, G^{hi}) according to definition 3.1.1, with the reporter map to the high-level $\theta := p^{hi}$, where $p^{hi} : \Sigma^* \to (\Sigma^{hi})^*$ is the natural projection to high-level events.*
  The set of high-level events contains the set of all shared events
  $\bigcup_{i,j,i \neq j}^{n} (\Sigma_i \cap \Sigma_j) \subseteq \Sigma^{hi} \subseteq \Sigma$.
  The high-level marked language is defined as
  $L_m^{hi} := p^{hi}(L_m^c)$, *where L_m^{hi} is regular by construction.*

- *The high-level supervisor is denoted $S^{hi} : L^{hi} \to \Gamma^{hi}$ with the high-level closed-loop language $L(S^{hi}/G^{hi})$ and a valid low-level supervisor implementation $S^{lo} : L^c \to \Gamma$ that guarantees $p^{hi}(L(S^{lo}/G^c)) \subseteq L(S^{hi}/G^{hi})$.*

- *The decentralized implementation of S^{lo} is given by supervisors S_i^{lo} with $L(S_i^{lo}/G_i^c) = p_i(L(S^{lo}/G^c))$.*

Since the locally controlled concurrent behavior of all decentralized modules $G^c = ||_{i=1}^n G_i^c$ is abstracted to one monolithic high-level plant G^{hi}, the following lemma presented in [14] is the important step concerning reduction of complexity:

Lemma 3.3.1 (Commutativity of Composition and Abstraction) *Given a hierarchical and decentralized control system with the notation $L_i^{hi} := p^{hi}(L_i^c)$, the high-level closed and marked languages are*

$$L^{hi} = p^{hi}(||_{i=1}^n L_i^c) = ||_{i=1}^n L_i^{hi} \text{ and } L_m^{hi} = p^{hi}(||_{i=1}^n L_{i,m}^c) = ||_{i=1}^n L_{i,m}^{hi}$$

This lemma says, that the abstraction of the computationally expensive synchronous product of the detailed local plants is identical to the less complex synchronous product of the abstraction of subplants. With that result it is now possible to compute the high-level plant G^{hi} and a high-level supervisor S^{hi} as described in Section 3.1 with the standard supervisor implementation S^{lo}.
The main task remaining is to find a correct *decentralized* implementaion of the low-level supervisor, such that hierarchical consistency and nonblocking low-level behavior are guaranteed.
A given controllable high-level behavior can be implemented by admissible decentralized supervisors S_i^{lo}, if the languages of the abstracted modules are *mutually controllable*.

Definition 3.3.2 (Mutual Controllability, High-Level) *Given a hierarchical and decentralized control system, two high-level languages L_i^{hi} and L_j^{hi} are said to be mutually controllable if*

$$L_j^{hi}(\Sigma_{uc}^{hi} \cap \Sigma_i \cap \Sigma_j) \cap p_j((p_i^{hi})^{-1}(L_i^{hi})) \subseteq L_j^{hi}$$

Given mutually controllable high-level languages, the decentralized supervisors can be implemented as follows.

Definition 3.3.3 (Decentralized Standard Supervisor Implementation) *Let H be a hierarchical and decentralized control system with a standard supervisor implementation of definition 3.1.5.*
The decentralized standard supervisor implementation is defined as

$$S_i^{lo}(p_i(s)) := p_i(S^{lo}(s)), \;\; i = 1, 2, \ldots, n \;, \; \forall s \in L^{hi}$$

This means that the control action of S^{lo} is projected to the local event set Σ_i of each module G_i.
The mutual controllability condition guarantees that the projections of a high-level supervisor to the subsystems are controllable with respect to the respective subsystem.[13]

Lemma 3.3.2 (Projected High-Level Supervisors)
Let H be a hierarchical and decentralized control system as in Definition 3.3.1.
If L_i^{hi} and L_j^{hi} are mutually controllable for $i, j = 1, \ldots, n$, then

$$\exists S_i^{hi} : (\Sigma_i^{hi})^* \rightarrow \Gamma_i^{hi} \text{ such that } L(S_i^{hi}/G_i^{hi}) = p_i(L(S^{hi}/G^{hi})) \; \forall i = 1, \ldots, n$$

One way to show that a hierarchical and decentralized control system with the above decentralized supervisor implementation fulfills the properties of hierarchical consistency and nonblocking behavior, is to make use of a property of many application examples, which is that a system should always be able to return to a marked state after a task has been completed. This property results in the circularity of the respective controlled language.

Definition 3.3.4 (Circular Language)
A regular language $L \in \Sigma^*$ is said to be circular if

$$\forall s \in L, \; \exists \sigma \in \Sigma \text{ such that } s\sigma \in L$$

For the application example of this work, this property is given by the circular structure of the system models. However, there are different conditions with which the following theorem can be proven.

Theorem 3.3.1 ([13]) *Let H be a hierarchical and decentralized control system as in Definition 3.3.1 that meets the following conditions:*

- *All hierarchical abstractions $(G_i^c, p^{hi}, G_i^{hi}),\ i = 1, 2, \dots, n$ are marked state accepting and locally nonblocking.*
- *The high-level languages $L_i^{hi},\ i = 1, 2, \dots, n$ are mutually controllable.*
- *All languages $p_i(L(S^{hi}/G^{hi}))$ are circular.*
- *For S^{lo}, the standard supervisor implementation of Definition 3.1.5 is chosen, with the corresponding standard decentralized supervisor implementations S_i^{lo}.*

Then H is hierarchically consistent, and the low-level control is nonblocking.

Note that although this approach contains decentralized supervisory control, the property of shared event marking is no longer required, as nonblocking control of the concurrent behavior is guaranteed by the high-level supervisor in combination with the aforementioned structural properties of the system.
However, for larger systems, the composition of all abstracted subsystems to one high-level plant might still be too expensive. In the following section we will see how this approach can be extended to a multi-level hierarchy.

Chapter 4

Modifi cation of the Hierarchical and Decentralized Approach

In this chapter the approach presented in [13] will be extended to a multi-evel hierarchy and modified such that a property less restrictive than local nonblocking guarantees hierarchical consistency and nonblocking behavior of the controlled system.

4.1 Multi-Level Hierarchy

An example of this decentralized multi-level structure with 4 subsystems and 3 levels in the hierarchy is shown in Figure 4.1. Generally, there is no restriction on the number of levels and the number of subsystems at each level.

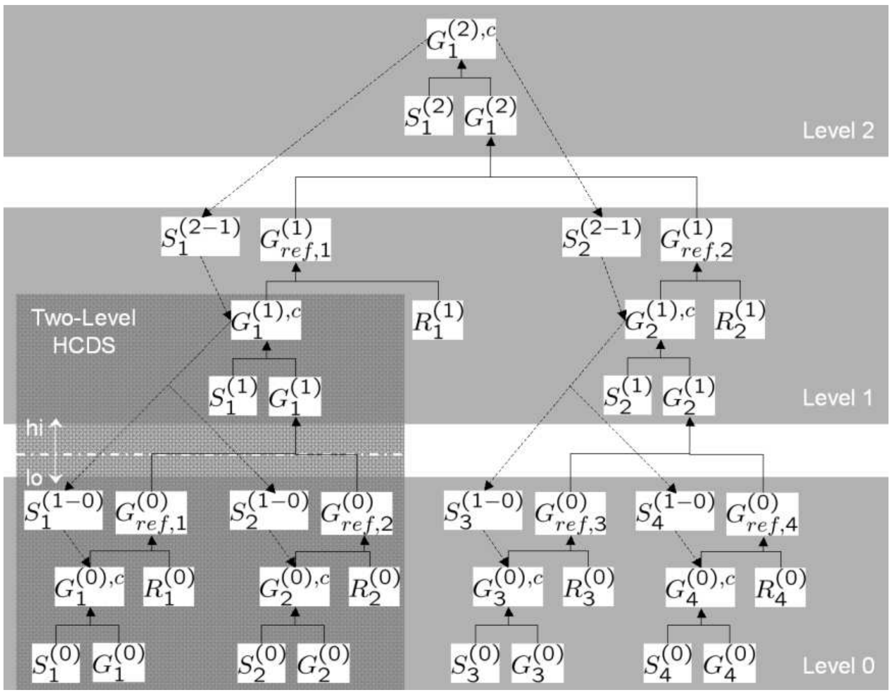

$G_1^{(2),c}$
$S_1^{(2)}$ $G_1^{(2)}$
Level 2
$S_1^{(2-1)}$ $G_{ref,1}^{(1)}$
$S_2^{(2-1)}$ $G_{ref,2}^{(1)}$
Two-Level HCDS
$G_1^{(1),c}$ $R_1^{(1)}$
$G_2^{(1),c}$ $R_2^{(1)}$
$S_1^{(1)}$ $G_1^{(1)}$
$S_2^{(1)}$ $G_2^{(1)}$
Level 1
hi
lo
$S_1^{(1-0)}$ $G_{ref,1}^{(0)}$ $S_2^{(1-0)}$ $G_{ref,2}^{(0)}$ $S_3^{(1-0)}$ $G_{ref,3}^{(0)}$ $S_4^{(1-0)}$ $G_{ref,4}^{(0)}$
$G_1^{(0),c}$ $R_1^{(0)}$ $G_2^{(0),c}$ $R_2^{(0)}$ $G_3^{(0),c}$ $R_3^{(0)}$ $G_4^{(0),c}$ $R_4^{(0)}$
$S_1^{(0)}$ $G_1^{(0)}$ $S_2^{(0)}$ $G_2^{(0)}$ $S_3^{(0)}$ $G_3^{(0)}$ $S_4^{(0)}$ $G_4^{(0)}$
Level 0

Figure 4.1: Example of a decentralized 3-level hierarchy

At the lowest level (0), the plant is described by detailed models of subsystems $G_i^{(0)}$ ($i = 1,2,3,4$) that are locally controlled by decentralized supervisors $S_i^{(0)}$.

The locally controlled systems are then refined, i.e. additional events, defined in a *refinement alphabet*, are added by composition of the locally controlled plant $S_i^{(0)}/G_i^{(0)}$ with a refinement automaton $R_i^{(0)}$. One possible function of a refinement automaton is identifying unique low-level strings representing a certain task.

The refined system models $G_{ref,i}^{(0)} = (S_i^{(0)}/G_i^{(0)})||R_i^{(0)}$ are abstracted to level 1 by the natural projection to the set of high-level events $\Sigma_i^{(1)}$, which consists of the refinement alphabet as well as those events out of the respective local event set $\Sigma_i^{(0)}$, that are defined to be high-level events. Arbitrary groups of the abstracted subsystems are then composed to several modules $G_j^{(1)}$ of level 1, where in this example $j = 1,2$. For each level-1 subsystem, level-1 supervisors $S_i^{(1)}$ are synthesized. For this abstraction, level 0 is viewed as the low level, and level 1 represents the high level of the hierarchy.

In a next step, the level-1 system can be considered as a low-level system which shall be abstracted to a higher level 2. Thus the locally controlled level-1 subsystems are refined by refinement events and abstracted to level 2, where they are composed to (in the example case) one level-2 system $G^{(2)}$. The behavior of $G^{(2)}$ is controlled by $S^{(2)}$, whose decentralized low-level implementations are $S_1^{(2-1)}$ and $S_2^{(2-1)}$. These low-level implementations are a translation of the abstract control actions of level 2 to detailed

level-1 control actions for each locally controlled subplant in level 1. So, $S_1^{(2-1)}$ and $S_2^{(2-1)}$ restrict the locally controlled behaviors $S_1^{(1)}/G_1^{(1)}$ and $S_2^{(1)}/G_2^{(1)}$ on level 1.
Analogously, the control action resulting on level 1 is implemented on level 0 by $S_i^{(1-0)}$, which additionally restrict the controlled behavior of each local subsystem $S_i^{(0)}/G_i^{(0)}$ $(i = 1, \dots, 4)$ on level 0.
Note that each pair $S_i^{((j+1)-j)}$ and $G_{ref,i}^{(j)}$ can be interpreted as an *interface* that translates detailed strings generated in the lower level to high-level events as well as high-level control actions to more complex tasks in the lower level. The notion of an interface connecting two levels is found in several approaches to hierarchical control, see for example [7].

For convenience and to be able to easily apply the notations of 3.3, we will focus on a two-level hierarchy with a lower level defined over the event set Σ^{lo} and a higher level defined over Σ^{hi}, as for example the part of the system marked by the dark box in Figure 4.1. For the general structure of a two-level hierarchy see Figure 4.2.

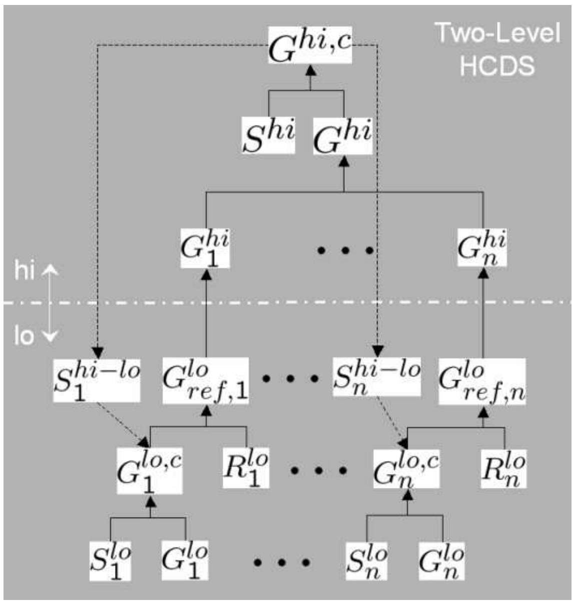


Figure 4.2: Two-level hierarchy

This part of two connected levels can be seen as a elementary hierarchical and decentralized control system, the results of which can be transferred to an arbitrary multi-level hierarchy.[1]

[1]For clarity reasons the local high-level subsystems G_i^{hi} are not shown in Figure 4.1

At first, we define the resulting hierarchical and decentralized control system, when refinement languages are applied.

Definition 4.1.1 (Extended Hierarchical and Decentralized Control System)
An extended hierarchical and decentralized control system (EHDCS) is a structure that consists of the following components.

- *The detailed low-level plant model G^{lo} over the event set $\Sigma^{lo} := \bigcup_{i=1}^{n} \Sigma_i^{lo}$ is given as a decentralized control system according to Definition 3.2.1 with a number of n subsystems G_i^{lo} over the respective event sets Σ_i^{lo}, generating the respective languages $L_i^{lo} := L(G_i^{lo})$ and marking $L_{i,m}^{lo} := L_m(G_i^{lo})$. The overall system is defined as $G^{lo} := ||_{i=1}^{n} G_i^{lo}$ over the alphabet $\Sigma^{lo} := \bigcup_{i=1}^{n} \Sigma_i^{lo}$. The controllable and uncontrollable events are $\Sigma_{i,c}^{lo} := \Sigma_i^{lo} \cap \Sigma_c^{lo}$ and $\Sigma_{i,u}^{lo} := \Sigma_i^{lo} \cap \Sigma_u^{lo}$, respectively, where $\Sigma_c^{lo} \cup \Sigma_u^{lo} = \Sigma^{lo}$ and $\Sigma_c^{lo} \cap \Sigma_u^{lo} = \emptyset$.*

- *Nonblocking local low-level controllers $S_i^{lo}\colon L_i^{lo} \to \Gamma_i^{lo}$, where Γ_i^{lo} are the respective control patterns. The resulting low-level closed-loop languages are denoted*
  $L_i^{lo,c} := L(S_i^{lo}/G_i^{lo})$
  $L_{i,m}^{lo,c} := L_i^{lo,c} \cap L_{i,m}^{lo}$,
  $L^{lo,c} := ||_{i=1}^{n} L_i^{lo,c}$
  $L_m^{lo,c} := ||_{i=1}^{n} L_{i,m}^{lo,c} = L^{lo,c} \cap L_m^{lo}$.
  $G^{lo,c}$ is the canonical recognizer of the globally resulting locally controlled behavior such that
  $L^{lo,c} = L(G^{lo,c})$, $L_m^{lo,c} = L_m(G^{lo,c})$.[2]

- *High-level events $\Sigma^{hi} := \bigcup_{i=1}^{n} \Sigma_i^{hi}$ are introduced, where Σ_i^{hi} and Σ_i^{lo} are not necessarily disjoint: $\emptyset \subseteq \Sigma_i^{lo} \cap \Sigma_i^{hi}$, i.e. low-level events can also be defined as high-level events. Furthermore, $\Sigma_{ref} := \bigcup_{i=1}^{n} \Sigma_{ref,i} = \Sigma^{hi} - \Sigma^{lo} \subseteq \Sigma^{hi}$ is the set of all refinement events, i.e. all refinement events are high-level events. Moreover, all shared events are high-level events: $\bigcup_{j=1,i\neq j}^{n} (\Sigma_i^{lo} \cap \Sigma_j^{lo}) \subseteq \Sigma_i^{hi}$.*

- *The refinement automata are denoted R_i^{lo}. They generate the refinement languages*
  $K_{ref,i}^{lo} := L(R_i^{lo}) \subseteq (\Sigma_i^{lo} \cup \Sigma_{ref,i})^*$ *and mark the languages*
  $K_{ref,i,m}^{lo} := L_m(R_i^{lo}) \subseteq (\Sigma_i^{lo} \cup \Sigma_{ref,i})^*$
  with the set of refinement events $\Sigma_{ref,i}$ such that
  $L_{ref,i}^{lo} := L_i^{lo,c} || K_{ref,i}^{lo}$,

[2]Note that by defi nition $L^{lo,c} = \overline{L_m^{lo,c}}$.

$L^{lo}_{ref,i,m} := L^{lo,c}_{i,m} || K^{lo}_{ref,i,m}$
are the refined languages. The canonical recognizer of $L^{lo}_{ref,i,m}$ is denoted $G^{lo}_{ref,i}$,i.e $L(G^{lo}_{ref,i}) = L^{lo}_{ref,i}$ and $L_m(G^{lo}_{ref,i}) = L^{lo}_{ref,i,m}$.

- *Hierarchical abstractions $(G^{lo}_{ref,i}, p^{hi}, G^{hi}_i)$, with the reporter map to the high-level $\theta := p^{hi}$, where $p^{hi} : (\Sigma^{lo} \cup \Sigma_{ref,i})^* \rightarrow (\Sigma^{hi})^*$ is the natural projection to high-level events.*
  The local high-level languages are given by the abstraction of the refined languages:
  $L^{hi}_i = L(G^{hi}_i) := p^{hi}(L^{lo}_{ref,i})$,
  $L^{hi}_{i,m} = L_m(G^{hi}_i) := p^{hi}(L^{lo}_{ref,i,m})$

- *The high-level plant is given by $G^{hi} := ||^n_{i=1} G^{hi}_i$, so $L^{hi} := ||^n_{i=1} L^{hi}_i$ and $L^{hi}_m := ||^n_{i=1} L^{hi}_{i,m}$. Each pair G^{hi}_i, G^{hi}_j of abstracted subsystems is synchronized by shared events if $\Sigma^{hi}_i \cap \Sigma^{hi}_j \neq \emptyset$.*

- *The high-level supervisor is denoted $S^{hi} : L^{hi} \rightarrow \Gamma^{hi}$ with the high-level closed-loop language $L(S^{hi}/G^{hi})$ and a valid low-level supervisor implementation $S^{hi-lo} : L^{lo,c} \rightarrow \Gamma^{hi-lo}$ with the control pattern $\Gamma^{hi-lo} \subseteq 2^{(\Sigma^{lo} \dot\cup \Sigma_{ref})}$ has to guarantee that $p^{hi}(L(S^{hi-lo}/G^{lo}_{ref})) \subseteq L(S^{hi}/G^{hi})$.*

- *The decentralized implementation of S^{hi-lo} is given by valid supervisors S^{hi-lo}_i.*

4.2 Refinement Automata

One modification of the approach described in [13] is the notion of *refinement automata*, which are an important tool for the abstraction process. They can be used to identify unique low-level strings representing a certain detailed task and to introduce refinement events to report the beginning and the completion of these tasks to the high-level. The refinement automata have to fulfill the following requirements:

Definition 4.2.1 (Admissible Refinement Automaton) *Let G^{lo} be a finite automaton over the event set Σ^{lo} with $L(G^{lo}) = L^{lo} \subseteq (\Sigma^{lo})^*$ and $L^{lo}_m = L_m(G^{lo})$. Furthermore, let Σ^{hi} be the set of high-level events with $\Sigma_{ref} \subseteq \Sigma^{hi}$. Also let R^{lo} be an automaton over the event set $(\Sigma^{lo} \dot\cup \Sigma_{ref})$ generating the language $L(R^{lo}) = K^{lo}_{ref} \subseteq (\Sigma^{lo} \dot\cup \Sigma_{ref})^*$ and marking $L_m(R^{lo}) := K^{lo}_{ref,m}$. Then the refined languages are $L^{lo}_{ref} = L^{lo} || K^{lo}_{ref}$ and $L^{lo}_{ref,m} = L^{lo}_m || K^{lo}_{ref,m}$.*
R^{lo} is said to be an admissible refinement automaton with respect to G^{lo}, if:

1. *R^{lo} meets the following structural property:*
   Let $s,\ s' \in R^{lo}$ and $p^{lo}(s) = p^{lo}(s')$. If $\exists u \in (\Sigma_{ref})^*,\ \sigma_{uc} \in \Sigma_{uc}^{lo}$ such that $su\sigma_{uc} \in R_{ref}^{lo}$, then $\exists u'\sigma_{uc}$ such that $s'u'\sigma_{uc} \in R_{ref}^{lo}$.

2. *K_{ref}^{lo} and $K_{ref,m}^{lo}$ are not restrictive with respect to L^{lo} and L_m^{lo}.*

   $$p^{lo}(L_{ref}^{lo}) = L^{lo},\ \ p^{lo}(L_{ref,m}^{lo}) = L_m^{lo}$$

   with the natural projection to low-level events $p^{lo} : (\Sigma^{lo} \cup \Sigma^{hi})^* \to (\Sigma^{lo})^*$

3. *K_{ref}^{lo} and $K_{ref,m}^{lo}$ are consistent with respect to the controllability of low-level events:*

   $$(\Sigma^{lo} \cup \Sigma_{ref})^* \Sigma_{ref,c} \Sigma_{ref}^* \Sigma_{uc}^{lo} (\Sigma^{lo} \cup \Sigma_{ref})^* \cap L_{ref}^{lo} = \emptyset\,,$$
   $$(\Sigma^{lo} \cup \Sigma_{ref})^* \Sigma_{ref,c} \Sigma_{ref}^* \Sigma_{uc}^{lo} (\Sigma^{lo} \cup \Sigma_{ref})^* \cap L_{ref,m}^{lo} = \emptyset$$

The first requirement means that if one low-level string is represented in R^{lo} by several different refined strings, then all uncontrollable events possible after the low-level string have to be possible in R^{lo} after these refined strings.
The second requirement says that K_{ref}^{lo} is not allowed to restrict the behavior of G^{lo} in any sense.
The third requirement means that all sequences of refinement events containing a controllable event must not be followed by an uncontrollable low-level event, such that uncontrollable low-level events can never be disabled by a high-level supervisor disabling a high-level event. So it is guaranteed that any controllable behavior E_{ref}^{lo} can be implemented by an admissible low-level supervisor as stated in the following lemma.

Lemma 4.2.1 (Consistency of Controllability)
Let G^{lo} be a finite automaton and R^{lo} be a refinement automaton generating K_{ref}^{lo} with the refined language L_{ref}^{lo} as stated in Definition 4.2.1. The requirements on K_{ref}^{lo} in Definition 4.2.1 guarantee that any behavior E_{ref}^{lo} that is controllable with respect to the refined language L_{ref}^{lo} can be implemented by an admissible low-level supervisor:

$$\forall E_{ref}^{lo} \in \mathcal{C}(L_{ref}^{lo}) :\ \ p^{lo}(E_{ref}^{lo}) \in \mathcal{C}(L^{lo})$$

Proof 4.2.1 [3]
Let $s \in E_{ref}^{lo}$ such that $p^{lo}(s)\sigma_{uc} \in L^{lo}$ and $s\sigma_{uc} \notin E_{ref}^{lo}$ with $\sigma_{uc} \in \Sigma_{uc}^{lo}$.
Then, because of the first condition in Definition 4.2.1, $\exists u \in (\Sigma_{ref}^{lo})^*$ such that $su\sigma_{uc} \in L_{ref}^{lo}$.

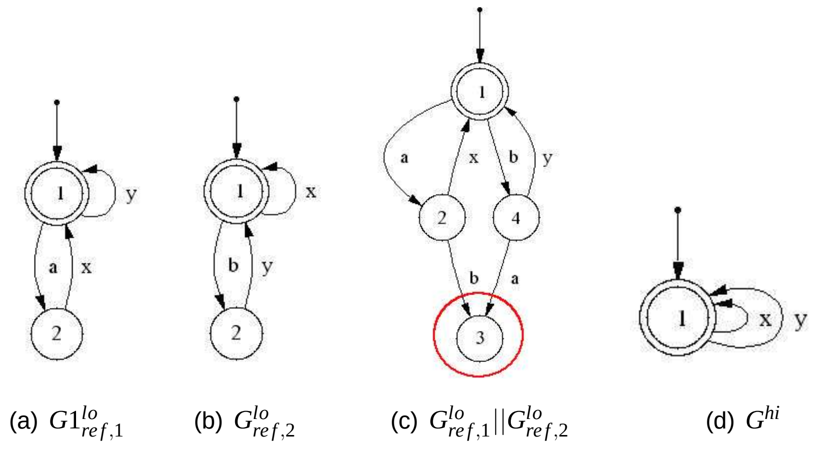
[3]The proof is the result of collaborative work with Dipl.Ing. Klaus Schmidt.

Let $u' \in \Sigma_{ref}^{lo}$ be the longest extension of s with refinement events, i.e. $su' \in E_{ref}^{lo}$ and $\nexists s\bar{u}' \in E_{ref}^{lo}$ with $su' < s\bar{u}'$. Then $su'\sigma_{uc} \notin E_{ref}^{lo}$ as $p^{lo}(su')\sigma_{uc} \in p^{lo}(E_{ref}^{lo})$.
Thus, with $u = u'u''$, and as E_{ref}^{lo} is controllable with respect to L_{ref}^{lo}, it holds that $u'' = \sigma_c\tilde{u}$ with $\sigma_c \in \Sigma_{ref,c}$.
Now, $su'\sigma_c\tilde{u}\sigma_{uc} \in L_{ref}^{lo}$ and $su' \in E_{ref}^{lo}$ and $su'\sigma_c \notin E_{ref}^{lo}$.
But as $su'\sigma_c\tilde{u}\sigma_{uc} \in L_{ref}^{lo}$ and $su'\sigma_c\tilde{u}\sigma_{uc} \in (\Sigma^{lo} \cup \Sigma_{ref})^*\Sigma_{ref,c}\Sigma_{ref}^*\Sigma_{uc}^{lo}(\Sigma^{lo} \cup \Sigma_{ref})^*$, it follows that $(\Sigma^{lo} \cup \Sigma_{ref})^*\Sigma_{ref,c}\Sigma_{ref}^*\Sigma_{uc}^{lo}(\Sigma^{lo} \cup \Sigma_{ref})^* \cap L_{ref}^{lo} \neq \emptyset$, which contradicts the third condition in Definition 4.2.1. Thus $p^{lo}(E_{ref}^{lo})$ is controllable with respect to L^{lo}.

A further modification of the approach described in [13] is the relaxation of the property of local nonblocking of the hierarchically abstracted subsystems in combination with a low-level supervisor implementation that is different from the standard supervisor implementation defined in [13].

4.3 Modified Supervisor Implementation

We recall the Definition 3.1.9 of a locally nonblocking hierarchical abstraction.
In many application examples this property can be fulfilled only with restrictive models of the subplants, as different local extensions of exit strings usually decide on which subsequent high-level event is possible. So these alternative local extensions can be seen as local predecessors for different high-level events.
If local nonblocking is not fulfilled, the problem is to guarantee that two decentralized subsystems generate the local predecessors of the same shared event, otherwise blocking is possible that is not noticed by G^{hi}. The following figure illustrates a case of local blocking.



(a) $G1_{ref,1}^{lo}$ (b) $G_{ref,2}^{lo}$ (c) $G_{ref,1}^{lo}||G_{ref,2}^{lo}$ (d) G^{hi}

Figure 4.3: Two subsystems causing local blocking

The set of high-level events is given by the shared events $\Sigma^{hi} = \{x,y\}$. So, if the local event a is generated in the refined subsystem $G_{ref,1}$ and event b happens in $G_{ref,2}$, the concurrent behavior $G^{lo}_{ref} = G^{lo}_{ref,1}||G^{lo}_{ref,2}$ (Figure 4.3 (c)) of both subsystems has a blocking state, because the subsystems are not able to execute the same high-level event. This blocking is not noticed in $G^{hi} = p^{hi}(G^{lo}_{ref,1})||p^{hi}(G^{lo}_{ref,2})$, i.e. a nonblocking high-level supervisor S^{hi} does not prevent it. A way to avoid this problem is to design a low-level supervisor implementation that disables local predecessors of different high-level events.
Because of this, the implementation of a low-level supervisor is modified as follows.

Definition 4.3.1 (Modified Supervisor Implementation) *Given an extended hierarchical control system, the modified supervisor implementation is defined as*

$$L(S^{hi-lo}/G^{lo}_{ref}) := \overline{\kappa_{L^{lo}_{ref}}(L_m(S^{hi}/G^{hi})||L_m(G^{lo}_{ref}))}$$

The above expression points out, that the controlled high-level behavior can be seen as a specification for G^{lo}_{ref}, for which S^{hi-lo} is the nonblocking and admissible supervisor. With this supervisor implementation, only those exit string extensions in G^{lo} are allowed, from which one of the high-level events enabled by S^{hi} can be reached, while local strings leading to disabled high-level events are disabled. In a general EHDCS according to Definition 4.1.1, the low-level consists of several decentralized refined subsystems (as there are $G^{lo}_{ref,1}$ and $G^{lo}_{ref,2}$ in the above example), to which a decentralized form of the above supervisor implementation is applied.

Definition 4.3.2 (Decentralized Modified Supervisor) *Given an extended hierarchical control system of Definition 4.1.1, the decentralized modified supervisors are defined as*

$$L(S_i^{hi-lo}/G^{lo}_{ref,i}) := \overline{\kappa_{L^{lo}_{ref,i}}(p_i(L_m(S^{hi}/G^{hi}))||L_m(G^{lo}_{ref,i}))}$$

where $p_i : (\Sigma^{lo} \cup \Sigma^{hi})^* \rightarrow (\Sigma_i^{lo} \cup \Sigma_i^{hi})^*$ is the natural projection to the event set of each local subsystem .

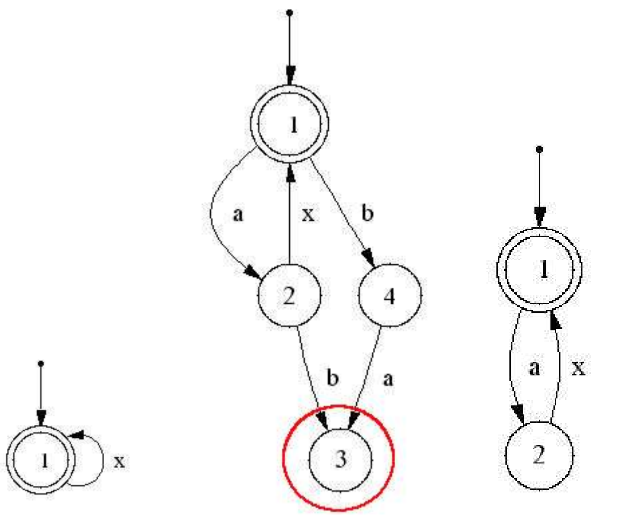
The overall behavior of the subsystems controlled by the decentralized modified supervisors is implemented according to the subsequent definition.

Definition 4.3.3 (Decentralized Modified Supervisor Implementation) *Given an extended hierarchical control system of Definition 4.1.1, the decentralized modified supervisor implementation is defined as*

$$L(S^{hi-lo}/G^{lo}_{ref}) = L(S^{hi}/G^{hi})||\big(||_{i=1}^{n}(L(S_i^{hi-lo}/G^{lo}_{ref,i})\big)$$

Note that the interaction of the subsystems is coordinated by the high-level. One important condition in that context is the fact that all shared events are high-level events and thus can be observed and controlled by S^{hi}.

Considering the example of Figure 4.3, the difference between the standard and modified supervisor implementation is as follows. Assume a high-level supervisor S^{hi} that disables the event y, as shown in Figure 4.4 a). As the standard supervisor implementation always enables all local events, $S^{lo}_{standard}$ allows the events a, b and x in $G^{lo}_{ref,1}||G^{lo}_{ref,2}$. This control action causes a blocking, whenever event b has happened (Figure 4.4 b)). Different from that, the decentralized supervisor S^{hi-lo}_2 disables event b, because after b no high-level event can be reached in $G^{lo}_{ref,2}$. So the resulting overall low-level supervisor implementation $S^{hi-lo}_{modified}$ is nonblocking, see Figure 4.4 (c).



(a) S^{hi}/G^{hi} (b) $S^{hi-lo}_{standard}/G^{lo}_{ref}$ (c) $S^{hi-lo}_{modified}/G^{lo}_{ref}$

Figure 4.4: Standard and modified low-level supervisor implementation

However, if both high-level events x and y are allowed in the high-level, i.e. $S^{hi-lo}_{modified}/G^{lo}_{ref} = G^{lo}_{ref}$ (Figure 4.3 c)), also the modified supervisor implementation causes a blocking as local paths leading to different events of the same set of shared events are still not disabled and thus there remain competitive paths in $G^{lo}_{ref,1}$ and $G^{lo}_{ref,2}$ that block each other in the synchronized behavior.

For that reason, we introduce an additional property concerning the high-level supervisor.

Definition 4.3.4 (Single Event Control) *Given an extended hierarchical and decentralized control system with a modified supervisor implementation as in Definition 4.3.1, single* *event control is defined as follows:*

$$\forall s^{hi} \in L(S^{hi}/G^{hi}), \forall \sigma \in \Sigma^{hi}(s^{hi}) \cap \Sigma^{hi}_{c,i} \cap S^{hi}(s^{hi}),\ \ \forall i = 1,2,\ldots,n:$$

$$S^{hi}(s^{hi}) \cap (\Sigma^{hi}(s^{hi}) - \sigma) \cap \Sigma^{hi}_{c,i} = \emptyset$$

This means that after any high-level string, S^{hi} enables at most one controllable high-level event out of each local alphabet, such that each pair of low-level subplants that share events agrees on the same shared high-level event. Thus there are no competing paths that block each other in the concurrent behavior. When considering application examples like the one of this work, this property is not very restrictive. A possible implementation is to set priorities of high-level events of one set of shared events, such that if S^{hi} enables a choice of more than one high-level event, only the event with the highest priority is implemented in the low-level.
A further useful property of the system models is the *single event controllability*, which says that low-level strings to different high-level events always begin with a controllable low-level event, i.e. these strings can always be disabled by a low-level supervisor implementation.

Definition 4.3.5 (Single Event Controllability) *Let H be an extended hierarchical and decentralized control system. For a given high-level string $s^{hi} \in L^{hi}_i$ with a controllable high-level successor event $\alpha \in \Sigma^{hi}_i(s^{hi}) \cap \Sigma^{hi}_{c,i}$, the language*

$$L_{s_{en},s^{hi},\alpha} := \{u\sigma | u \in (\Sigma^{lo} - \Sigma^{hi})^* \wedge s_{en}u\sigma \in L(G^{lo}_{ref,i}) \wedge \sigma \in \alpha \cup \Sigma^{hi}_{uc,i}\}$$

is the set of all local extensions of an entry string $s_{en} \in L_{en,s^{hi}}$ of s^{hi} that can be extended by α or an uncontrollable high-level event.
The high-level string s^{hi} is said to be single event controllable, if

$$\kappa_{L_{s_{en},s^{hi}}}(L_{s_{en},s^{hi},\alpha}) \text{ is locally nonblocking.}$$

H is said to be single event controllable if the above property holds $\forall s^{hi} \in L^{hi}_i,\ i = 1,2,\ldots,n$.

Remark: Note that it can be shown that an EHCDS that is locally nonblocking automatically is single event controllable, further suggestions on that issue are given in Chapter 6.
In the case that no high-level event is possible after a marked high-level string, *marked state controllability* guarantees that the low-level can be driven to a marked state. This property is identified in [12].

Definition 4.3.6 (Marked State Controllability)

Let $(G^{lo}_{ref,i}, p^{hi}, G^{hi}_i)$ be a hierarchical abstraction and let $\gamma^{hi} \in \Gamma^{hi}_i$ be the control patterns of S^{hi}_i with $L(S^{hi}_i/G^{hi}_i) = p_i(L(S^{hi}/G^{hi})$.
Choose $s^{hi} \in L^{hi}_{i,m}$ such that $\gamma^{hi} \cap \Sigma^{hi}_i(s^{hi}) = \emptyset$. The string s^{hi} is marked state controllable if

$$\forall s \in L_{en,s^{hi}} : \kappa_{L_{s,s^{hi}}}(L_{s,s^{hi},\gamma^{hi}}) \neq \emptyset$$

$(G^{lo}_{ref,i}, p^{hi}, G^{hi})$ is marked state controllable if s^{hi} is marked state controllable for all $s^{hi} \in L^{hi}_{i,m}$ with $\gamma^{hi} \cap \Sigma^{hi}_i(s^{hi}) = \emptyset$.

The above properties are met in the application example of this work.

Further required system properties are the *marked state acceptance* of the hierarchical abstractions and the *mutual controllability* of the abstracted subplants.

Theorem 4.3.1 (Main Result)

Let H be an extended hierarchical and decentralized control system with the following properties:

- *All refinement languages are admissible.*
- *All hierarchical abstractions are marked state accepting and marked state controllable.*
- *All local high-level languages L^{hi}_i are mutually controllable.*
- *The low-level supervisors are given by the decentralized modified supervisor implementation of Definition 4.3.2 implemented according to Definition 4.3.3.*
- *H is single event controllable.*
- *All high-level supervisors impose single event control according to Definition 4.3.4.*

Then H is hierarchically consistent and the resulting low-level control is nonblocking.

See the appendix for the proof of the above theorem.
With this result, the approach of [13] is extended to a multi-level hierarchy with refinements for the identification of low-level strings and a modified decentralized supervisor implementation that, in combination with system properties different from local nonblocking, results in nonblocking behavior of the controlled system.

We will see in the following chapter that this approach is applicable for practical examples of considerable computational size. Refinement languages provide a powerful tool to implement an unlimited number of abstraction levels consisting of subsystems of manageable size.

Chapter 5

Case Study: An Automated Manufacturing System

In this chapter, the application of the extended hierarchical and decentralized approach to a realistic model of an automated manufacturing plant will be presented. At first, an overview of the plant will be provided, followed by the main section which contains the description of the modeling process based on the approach of Chapter 4. Then the application example is used to evaluate the computational complexity of this approach. The chapter is concluded with suggestions for a modular implementation on a programmable logic controller (PLC) with online generation of the low-level control action.

5.1 Application Example: A Fischertechnik Production Plant Model

The application example is provided by the discrete event systems group of the Lehrstuhl für Regelungstechnik of the Universität Erlangen-Nürnberg (webpage: [1]) and represents a typical structure of an automated manufacturing system. It is implemented as a Fischertechnik model and controlled by a industrial standard PLC, a SIEMENS SIMATIC S7-300. The following picture contains an overwiew of the production plant.

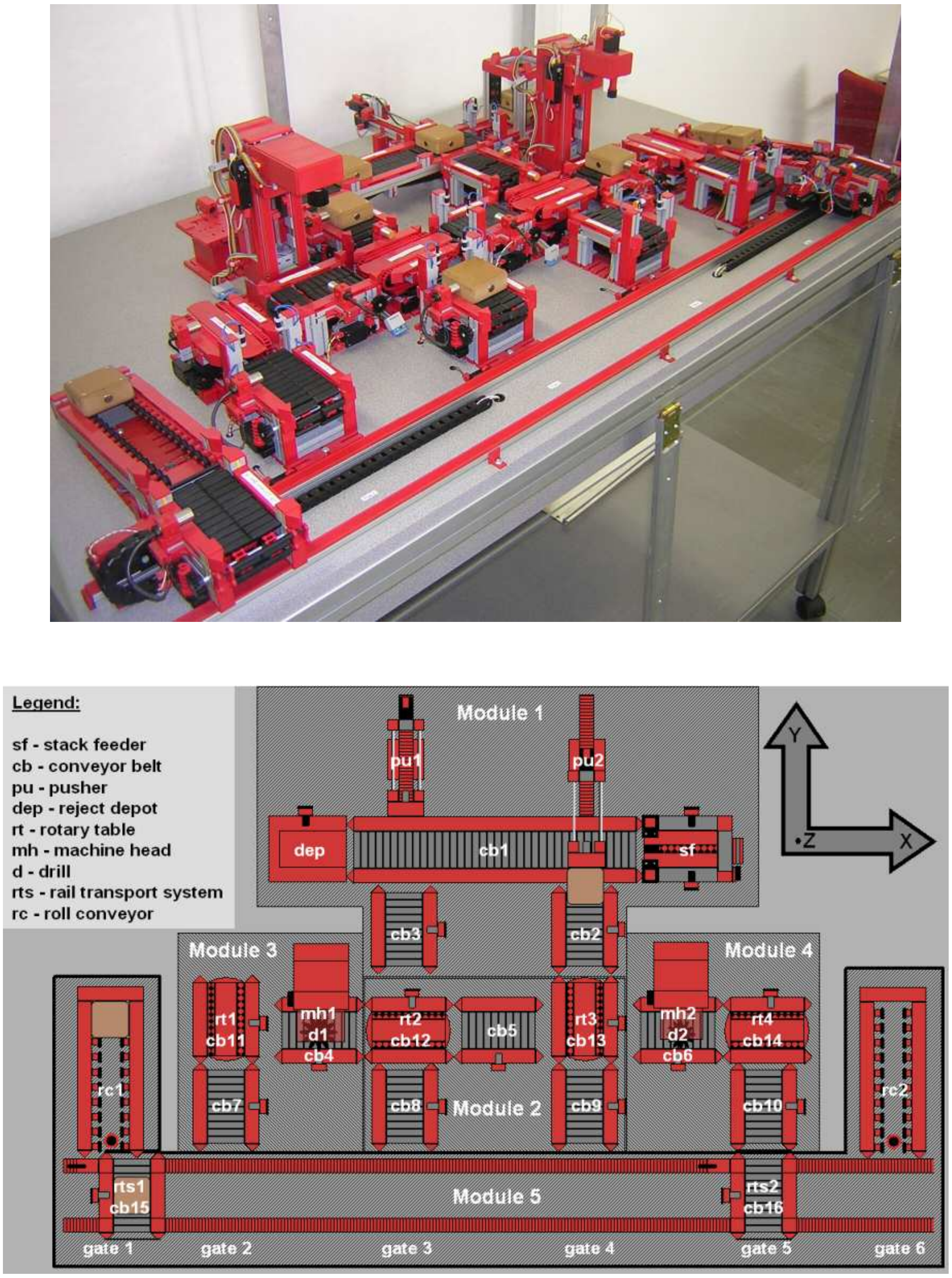


Figure 5.1: Fischertechnik automated manufacturing system with schematic overwiev

The production process starts at the stack feeder (sf) that inputs workpieces to the conveyor belt cb1, where they are distributed to either a reject depot (dep) or to conveyor belts

cb2 an cb3 via pushers pu1 and pu2.
The conveyor belts cb12 and cb13 on the rotary tables rt2 and rt3 can determine the direction of the workpieces arriving from cb2 and cb3 such that they can be transported to machine head mh1 equipped with a drill d1 positioned above conveyor belt cb4 or to machine head mh2 equipped with drill d2 above conveyor belt cb6, respectively, or they are transported to further conveyor belts cb8 and cb9. The rotary tables rt2 and rt3 are connected via conveyor belt cb5.
The exit of both manufacturing cells is either the way back to rt2 and rt3, or the workpieces are transported to conveyer belts cb7 and cb10 via conveyor belts cb11 and cb14 installed on rotary tables rt1 and rt4.
If a workpiece arrives at one of the conveyor belts cb7-cb10, it can be loaded on one of two rail transport systems rts1 and rts2 equipped with conveyor belts cb15 and cb16. The range of rts1 is delimited to the left by roll conveyor rc1, which is an exit buffer for up to 4 workpieces, and by conveyor belt cb9 to the right. Consequently, rts2 can serve conveyor belts cb8, cb9, cb10 as well as the second exit for workpieces, roll conveyor rc2. Note that the ranges of both rail transport systems overlap at coveyor belts cb8 and cb9. The positions, at which rts1 and rts2 can exchange workpieces with the rest of the system are denoted by gate 1-6.
Sensor signals, for example the arrival/departure of workpieces at each conveyor belt or the position of the rail transport systems at the respective gate are reported to the PLC, that can set actuator signals like the movement of each belt according to a control program. As any of these sensor and actuator signals is *binary*, the plant can be modeled as a discrete event system using finite state automata. The application of the theoretical approach presented in Chapter 4 to the plant is described in the following section.

5.2 Modeling and supervisor implementation

The theoretical approach presented in Chapter 4 has been implemented to the whole plant. Therefore, a hierarchy was developed containing the detailed lowest-level models of the plant up to 5 functional modules in the highest level of abstraction, as shown in Figure 5.1. In this report, we will focus on module 5, as it is examplary for the rest of the plant. It consists of the rail transport systems rts1 and rts2 combined with the conveyor belts cb15 and cb16 and the roll conveyors rc1 and rc2.

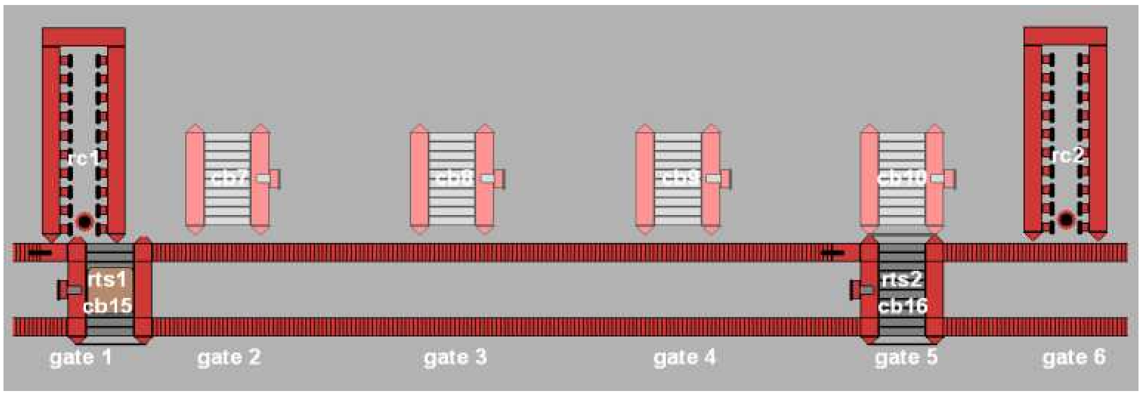


Figure 5.2: Module 5

The result of the application of the theoretical approach is the hierachical and decentralized structure shown in the subsequent figure.

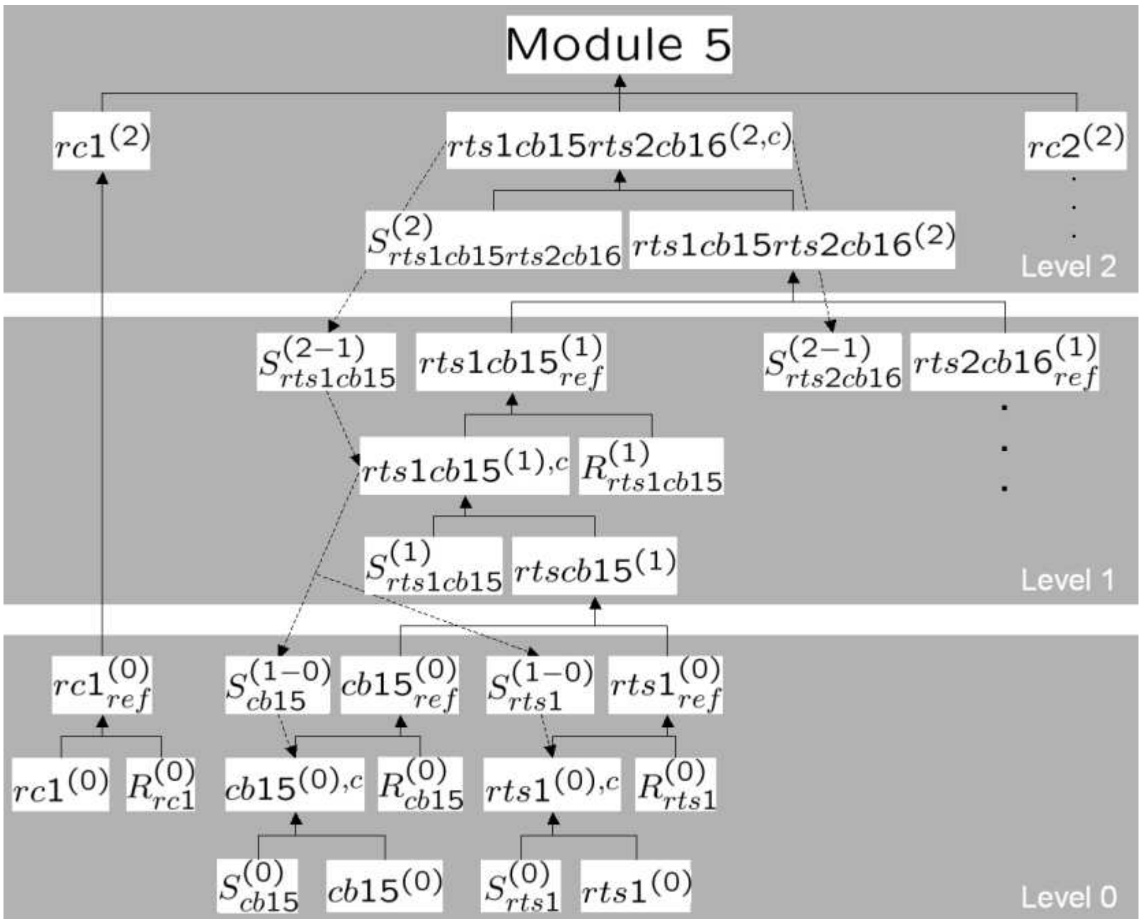
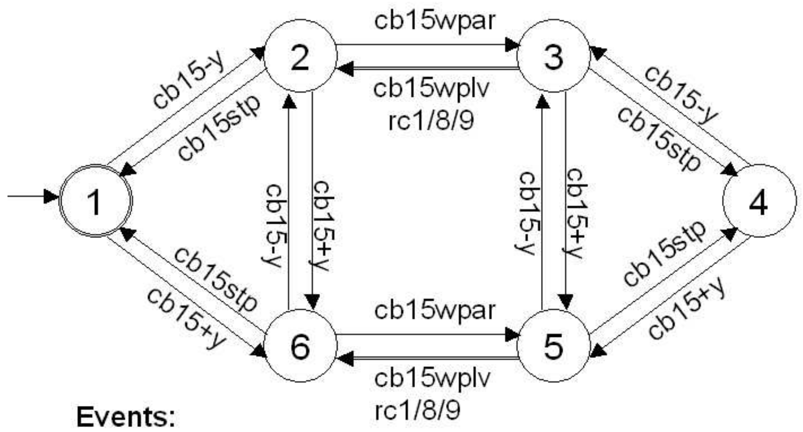


Figure 5.3: Hierarchy of module 5

Because of the symmetric structure of module 5, a description of the modeling process will be given for submodule rts1cb15, the results will then be transferred to submodule rts2cb16. We begin with the low-level model of conveyor belt 15.

5.2.1 Conveyor Belt cb15

The conveyor belt can be seen as an elementary component of the plant as it consists of one actuator (the belt drive) and one sensor for a workpieces presence, and its structure is the same for all conveyor belts except for cb1. The low-level model has already been developed in former works, e.g. [4, 14, 13] and is shown in Figure 5.4. For detailed explanations on how to derive a finite automaton from a discrete event system, see [4]. The resulting model of subplant $cb15^{(0)}$ is shown in the subsequent figure.



Events:

cb15+y:	belt moves in y-direction
cb15-y:	belt moves in negative y-direction
cb15stp:	belt movement stops
cb15wpar:	workpiece arrives at sensor
cb15wplvrc1/8/9:	workpiece leaves to rc1,cb8 or cb9

Figure 5.4: Low-level model of conveyor belt cb15: $cb15^{(0)}$

In the initial state the conveyor belt does not move and there is no workpiece at the sensor. The initial state is marked, which means that a task of the conveyor belt is not completed before the it is back in the initial state and thus ready for the next task. This property will be applied to all subsequent plant models as well.

Now that the low-level model of cb15 has been derived, we proceed to the computation of the low-level supervisor $S^{(0)}_{cb15}$, the first step of which is to formulate local specfications that are as follows:

- To avoid sudden changes of the direction of movement, the belt has to stop, before the direction is changed.

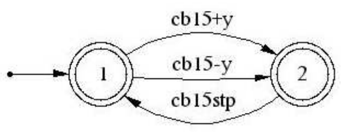


Figure 5.5: Specification $cb15^{(0)}_{spec1}$

- If the belt moves, it has to move until either a workpiece arrives or a present workpiece leaves to rc1, cb8 or cb9.

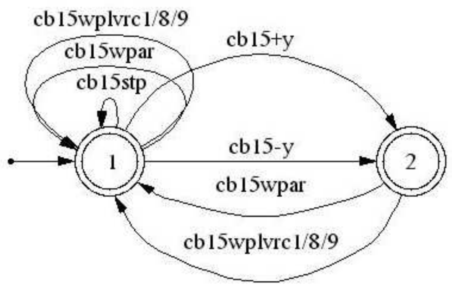


Figure 5.6: Specification $cb15^{(0)}_{spec2}$

- When a workpiece arrives (or leaves to rc1,cb8 or cb9), the belt has to stop. Event cb15stp is forcible, as it can force the belt to stop, before the workpiece leaves (or the next workpiece arrives).

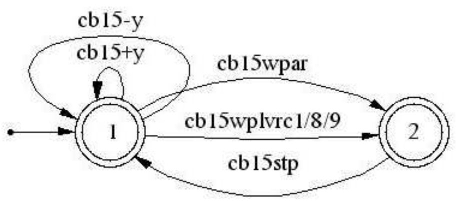


Figure 5.7: Specification $cb15^{(0)}_{spec3}$

- Until a workpiece arrives, movement only in the negative y-direction is allowed, then movement only in the y-direction is allowed until the workpiece leaves to rc1, cb8 or cb9.

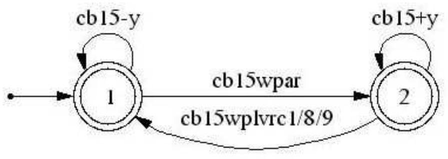


Figure 5.8: Specification $cb15^{(0)}_{spec4}$

These specifications are composed to form the overall specification $cb15^{(0)}_{spec} = ||^{4}_{i=1} cb15^{(0)}_{spec(i)}$, for which the supervisor $S^{(0)}_{cb15}$ is computed that leads to the following controlled behavior.

$$L(cb15^{(0),c}) = \kappa_{L(cb15^{(0)})}\big(L(cb15^{(0)}_{spec})||L(cb15^{(0)})\big) = L(S^{(0)}_{cb15}/cb15^{(0)})$$

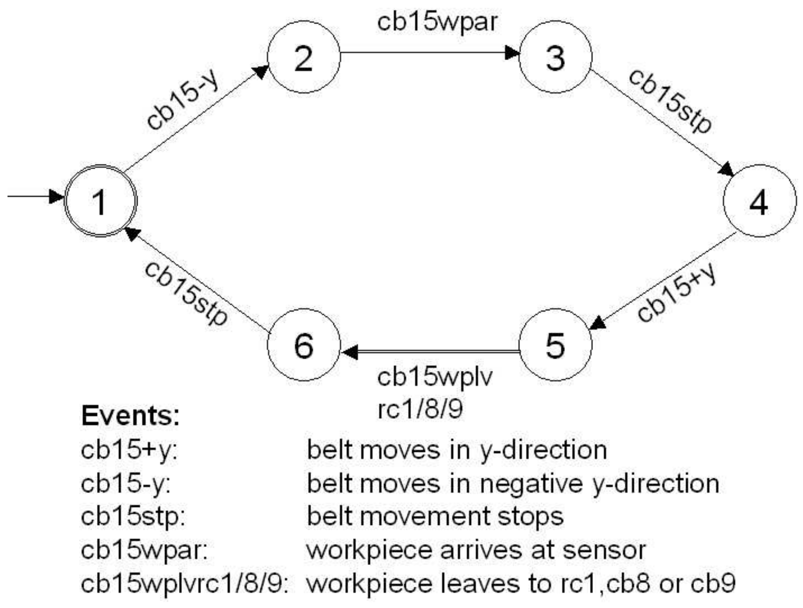

cb15wpar
cb15-y
cb15stp
cb15+y
cb15stp
cb15wplv rc1/8/9
1
2
3
4
5
6
Events:
cb15+y: belt moves in y-direction
cb15-y: belt moves in negative y-direction
cb15stp: belt movement stops
cb15wpar: workpiece arrives at sensor
cb15wplvrc1/8/9: workpiece leaves to rc1,cb8 or cb9

Figure 5.9: Controlled low-level behavior of cb15: $cb15^{(0),c}$

Because of their general nature, the specifications $cb15^{(0)}_{spec1}$ - $cb15^{(0)}_{spec3}$ can be transferred to all remaining conveyor belts.

Now, the locally controlled behavior of cb15 is refined by high-level events, which are events shared with the adjacent modules (rc1, cb7, cb8 and cb9) representing the beginning and the termination of a detailed low-level task. The introduction of starting and terminating events is useful for the identification of unique low-level strings. Note that this is similar to the notion of request and answer events in [7].

So if cb15 is in the initial state and starts moving, this means that the transport of a workpiece from cb7, cb8 or cb9 to cb15 is started. This beginning of the task shall be represented in the high-level by the refinement events $cb7-15,\ cb8-15$ and $cb9-15$. As the first event of the low-level task is controllable, also the respective refinement events $cb7/8/9-15$ are controllable. If the workpiece arrives at cb15 and the belt has stopped, this task is finished, so the refinement events $wp7-15,\ wp8-15$ and $wp9-15$ are introduced to designate the end of the task. These events are uncontrollable, because once a low-level task is started, the high-level has to wait for the completion of this low-level task and therefore is not allowed to disable a termination event. The resulting refinement automaton $R^{(0)}_{cb15,1}$ corresponding to this task is shown in Figure 5.10

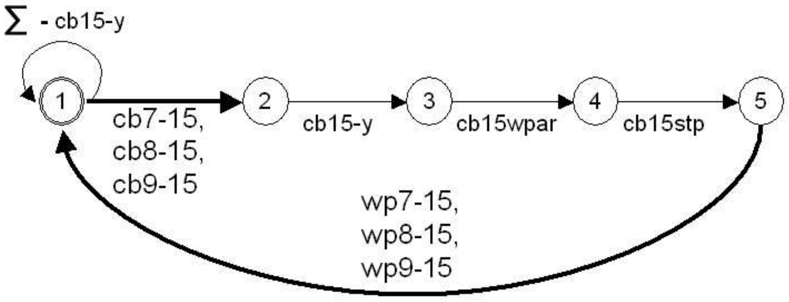


Figure 5.10: Refinement automaton $R^{(0)}_{cb15,1}$

In the automaton in Figure 5.10, the task *"transport of workpiece from cb7, cb8 or cb9 to cb15"* is identified.
Consequently, the remaining task to be refined is *"transport of workpiece from cb15 to rc1, cb8 or cb9"* [1], which is identified by the following refinement automaton $R^{(0)}_{cb15,2}$.

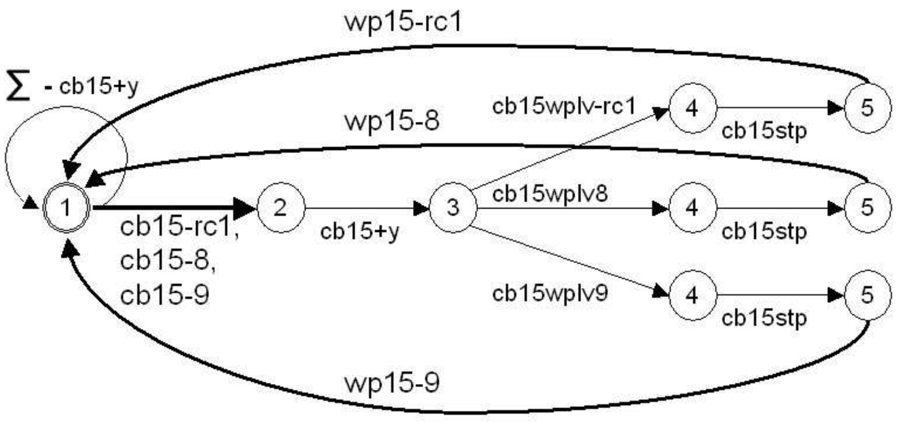


Figure 5.11: Refinement automaton $R^{(0)}_{cb15,2}$

[1]As cb7 will be used only in -y direction, it can not receive a workpiece from cb15.

Again, the refinement events representing the state of the low-level string are controllable, and the finishing refinement events are uncontrollable.
Both refinement automata are now applied to the controlled low-evel model $cb15^{(0),c}$ by parallel composition resulting in the refined automaton $cb15^{(0)}_{ref}$ shown in Figure 5.12.

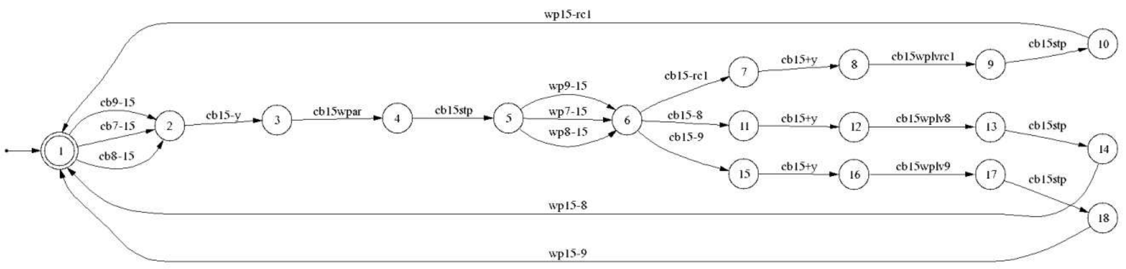


Figure 5.12: Refined automaton $cb15^{(0)}_{ref}$

Note that all strings possible in $cb15^{(0)}$ are contained in the strings of $cb15^{(0)}_{ref}$, and there is no uncontrollable low-level event that occurs after a controllable refinement event. Thus, $R^{(0)}_{cb15,1}$ and $R^{(0)}_{cb15,2}$ are admissible according to Definition 4.2.1.
As all low-level tasks are now uniquely identified, in this case the set of high-level events is given by the set of refinement events $\Sigma^{hi}_{cb15} = \Sigma_{ref,cb15} = \{cb7-15, cb8-15, cb9-15, wp7-15, wp8-15, wp9-15, cb15-rc1, cb15-8, cb15-9, wp15-rc1, wp15-8, wp15-9\}$. The refined automaton can now be projected to level 1. The result is shown in Figure 5.13.

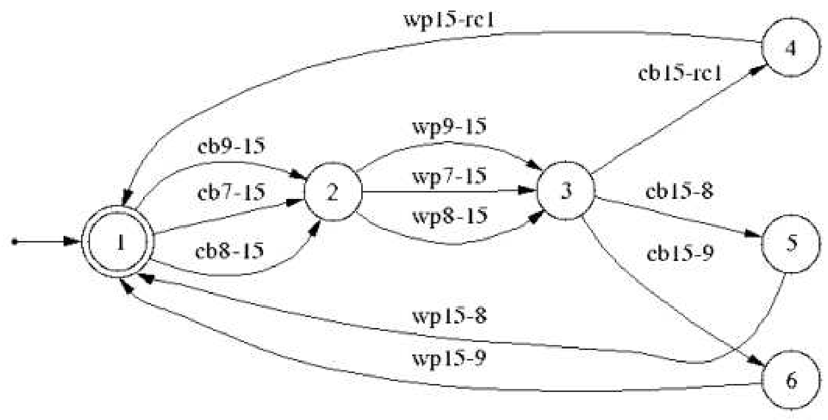
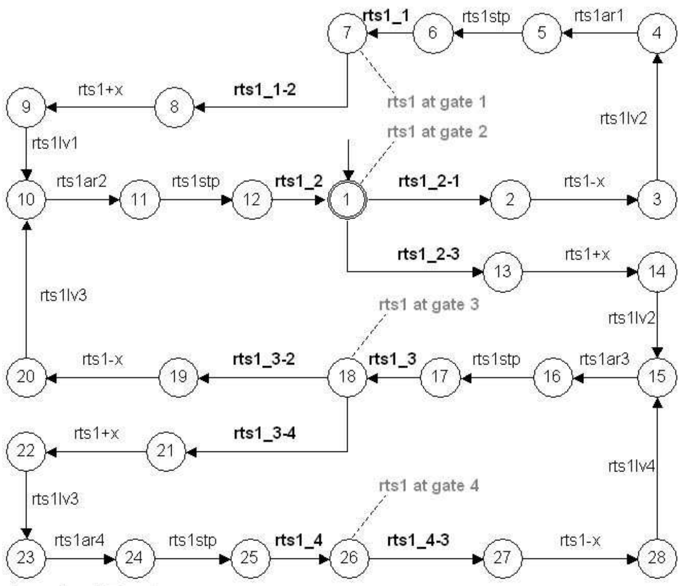


Figure 5.13: Projected level-1 automaton: $cb15^{(1)}$

For cb15, it is still not determinend with which one of the adjacent modules rc1, cb7,cb8 or cb9 it can momentarily interact. Controlling this task concerns cb15 as well as rts1. So in level 1, cb15 has to be composed with rts1, such that the concurrent behavior of both can be controlled by a level-1 supervisor.

5.2.2 Rail Transport System rts1

The detailed low-level model of a rail transport system has been presented in [4]. Different from [4], the initial state is the inactive rts1 standing at gate2 in line with cb7. Analogously to the development process of cb15, at first several specifications of general kind are applied by a low-level supervisor to the low-level model of rts1. So rts1 is not allowed to leave the range between rc1 and cb8. Furthermore, it has to stop whenever one of the gate positions 1-4 is reached, but between two gate positions it is neither allowed to stop nor to change the direction of movement.
The locally controlled model of rts1, $rts1^{(0),c}$ is then refinened by high-level events that identify the low-level tasks *"movement from one gate to a neighboring gate"*. For example, if rts1 moves from gate 2 to gate 3, this low-level task is started by the refinement event "rts1_2-3" and terminated by "rts1_3". This results in the refined automaton $rts1^{(0)}_{ref}$ presented in Figure 5.14.



Low-level Events:

rts1+x:	rts1 moves in x-direction
rts1-x:	rts1 moves in negative x-direction
rts1stp:	rts1 stops
rts1ar1(2,3,4):	rts1 arrives at gate 1 (2,3,4)
rts1lv1(2,3,4):	rts1 leaves gate 1 (2,3,4)

Figure 5.14: Refined low-level model of rts1: $rts1^{(0)}_{ref}$

The projection of this model to level 1 is given by $rts1^{(1)}$ shown in the subsequent figure.

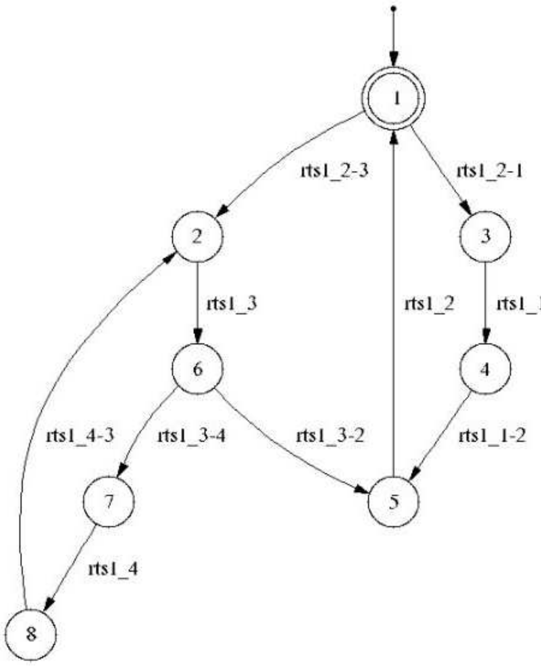


Figure 5.15: Projected level-1 automaton $rts1^{(1)}$

The local level-1 submodules $cb15^{(1)}$ and $rts1^{(1)}$ are now composed to the level-1 subplant $rts1cb15^{(1)}$.

5.2.3 Rail Transport System 1 and Conveyor Belt 15

For the level-1 subplant $rts1cb15^{(1)}$, the procedure of supervisory control followed by refinement is the same as it has been in Sections 5.2.1 and 5.2.3. There are two kinds of specifications for rts1cb15, one is the security aspect, which is the mutual exclusion of belt movement and movement of rts1, the second aspect is the efficiency, which is to avoid unnecessary paths.
According to that, the resulting specifications to rts1cb15 can be formulated as follows:

- No movement of the conveyor belt while rts1 moves and vice versa.
- Events cb8-15 and cb15-8 are only possible, if rts1 is at gate 3. Specifications with the same sense are formulated respectively for the remaining gates 1,2 and 4.
- If a workpiece is loaded on cb15 (e.g. from cb8), this workpiece has to be unloaded at a different gate (e.g. gate 1, 2 or 4, but not gate 3).
- Movement of rts1 to gate1 is only useful, if a workpiece is present at cb15.
- Every change of direction of the movement of rts1 is only possible after a load or unload task of cb15.

These specifications are composed to form an overall specification $rts1cb15^{(1)}_{spec}$ implemented in plant $rts1cb15^{(1)}$ by the supervisor $S^{(1)}_{rts1cb15}$ according to Figure 5.3. The controlled plant on level 1 is then given by $rts1cb15^{(1),c}$ with
$L(rts1cb15^{(1),c}) = \kappa_{L(rts1cb15^{(1)})}(L(rts1cb15^{(1)}_{spec}||rts1cb15^{(1)})) = L(S^{(1)}_{rts1cb15}/rts1cb15^{(1)})$
Now all movements of rts1 are synchronized correctly with the conveyor belt actions, so the events of rts1 are not reported to level 2. The only information about rts1, that has to be reported to level 2, is the return of rts1 from gates 3 and 4 to gate 2, which means that subplant rts2cb16 is now allowed to serve these gates. For that reason, the refinement event "rts1rdy" (rts1 ready) is introduced. The resulting refined automaton $rts1cb15^{(1)}_{ref}$ is shown in Figure 5.16.

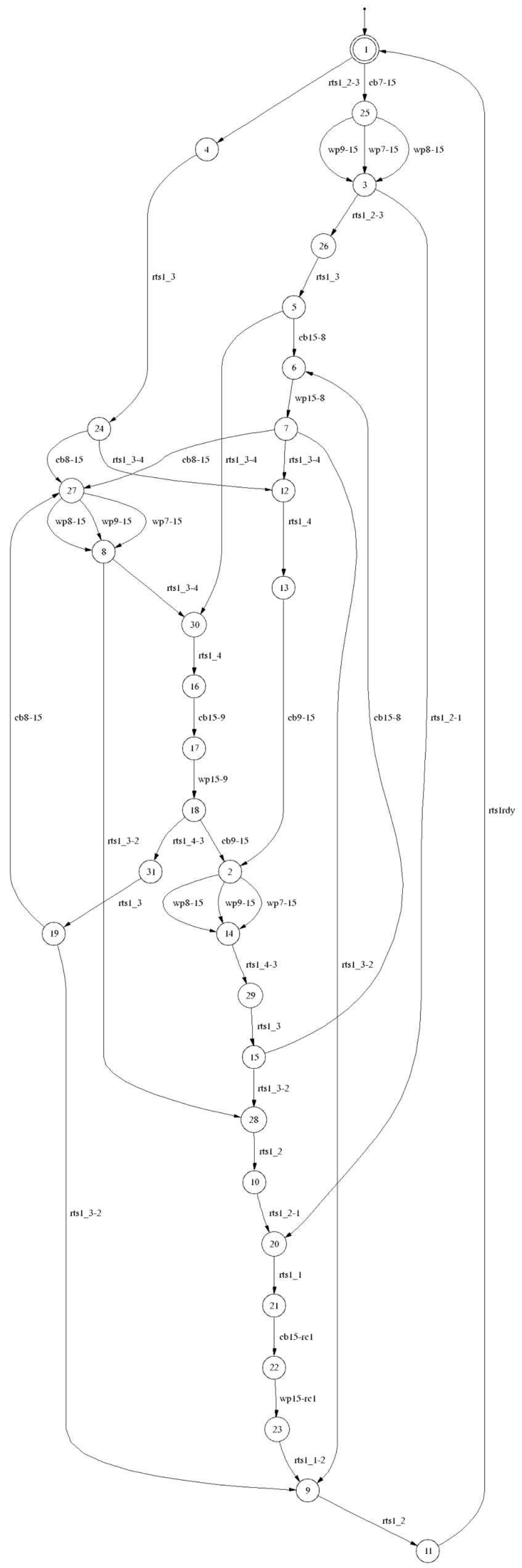


Figure 5.16: Refined automaton $rts1cb15^{(1)}_{ref}$

The set of high-level events is given by "rts1rdy" and all events of cb15, as all events of cb15 are shared with the adjacent modules.

Note that $rts1cb15^{(1)}_{ref}$ is not locally nonblocking. To show this, consider the initial state (state 1) in Figure 5.16. If a high-level supervisor enables cb7-15 and disables cb8-15 and cb9-15, there is a low-level string (rts1_2-3,rts1_3), after which cb7-15 is no longer possible. A further example is state 8, where we find a branching into two low-level signal pathes, each one ending with a different controllable high-level event (cb15-9 and cb15-rc1).

Here, the necessity of the property of single-event controllability in Definition 4.3.5 becomes obvious. So, whenever there is a branching to seperated low-level signal pathes, each ending with a different controllable high-levle event, the first event of each branch has to be controllable, such that it can eventually be disabled by the modified low-level implementation of the high-level control action given in Definition 4.3.1.

The high-level projection of $rts1cb15^{(1)}_{ref}$ denoted by $rts1cb15^{(2)}$ is given in the subsequent figure.

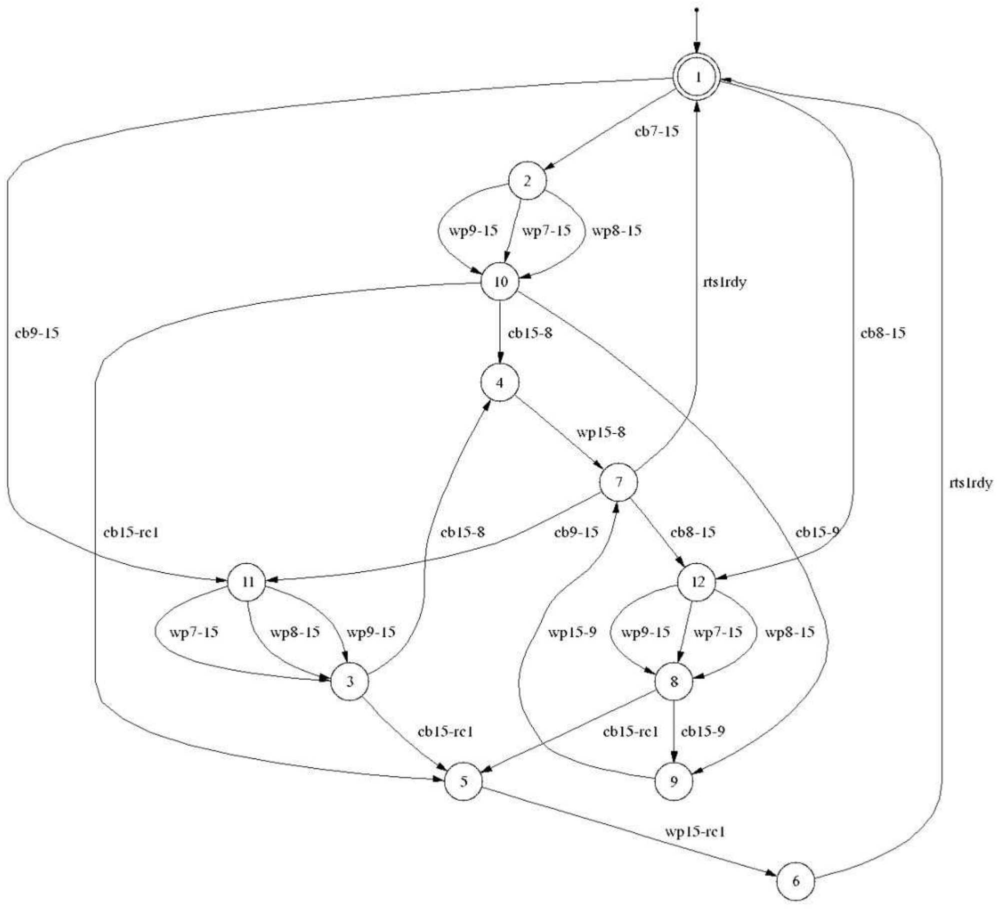


Figure 5.17: $rts1cb15^{(2)}$

For an illustration of the modified supervisor implementation $S^{(2-1)}_{rts1cb15}$, assume a level-2 supervisor $S^{(2)}_{rts1cb15rts2cb16}$ with the decentralized implementation $p_{rts1cb15}(S^{(2)}_{rts1cb15rts2cb16}) := S^{(2)}_{rts1cb15}$, that imposes the following control action: "A workpiece coming from conveyor belt 9 (cb9) shall be transported to roll conveyor 1 (rc1)." This results in the following controlled behavior $L(S^{(2)}_{rts1cb15}/rts1cb15^{(2)})$.

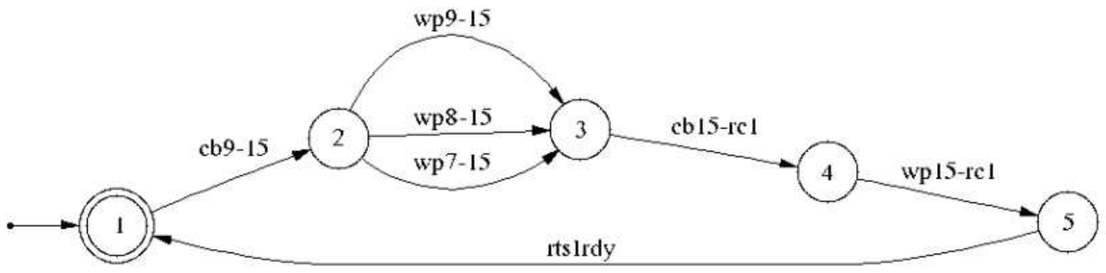


Figure 5.18: Controlled behavior of $rts1cb15^{(2)}$

According to the modified low-level supervisor implementation of Definition 4.3.1, this behavior can be used as a specification for the refined level-1 plant $rts1cb15^{(1)}_{ref}$, such that the nonblocking supervisor for this specification is the correct low-level supervisor implementation. The following figure shows the resulting controlled behavior of $rts1cb15^{(1)}_{ref}$, which is the control action for $rts1cb15^{(1),c}$.

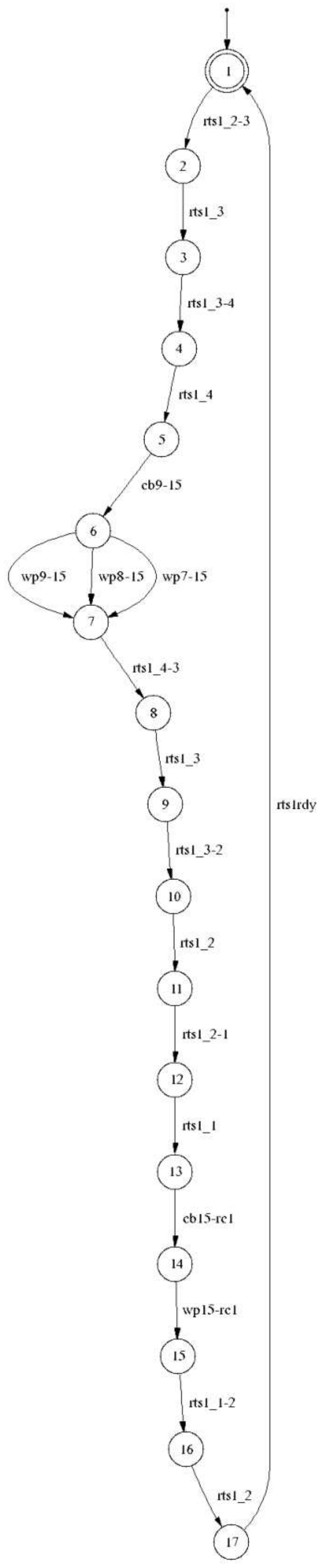


Figure 5.19: Controlled behavior of $rts1cb15_{ref}^{(1)}$ resulting from the modified low-level supervisor implementaion

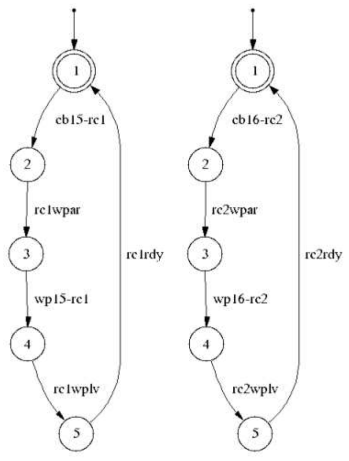
One can see, that the level-2 control action is implemented in nonblocking detailed level-1 control actions.

5.2.4 Module 5

With the development of the hierarchical and decentralized control architecture of submodule $rts1cb15$, the main steps of the implementation of the theoretical approach for module 5 are completed.

The models of submodule rts2cb16 can be derived directly from rts1cb15 using the symmetry of the plant structure. The composed subplant $rts1cb15rts2cb16^{(2)}$ is then controlled by $S^{(2)}_{rts1cb15rts2cb16}$, such that a collision of rts1 and rts2 in the overlapping range between cb8 and cb9 is avoided.

The remaining subplants are roll conveyor 1 and 2, which both just consist of a sensor that detects the arrival and departure of a workpiece, i.e. its low-level automaton model generates uncontrollable events only. Consequently, a level-0 supervisor is obsolete. These sensor signals "rc1wpar","rc1wplv" and "rc2wpar","rc2wplv" are translated by the refinement events "cb15-rc1","wp15-rc1" and respectively "cb16-rc2","wp16-rc2", which are shared with rts1cb15 and rts2cb16, and additional events "rc1rdy" and "rc2rdy". The purpose of these events is to report to level 2 that the respective roll conveyor can receive the next workpiece. The refined models $rc1^{(0)}_{ref}$ and $rc2^{(0)}_{ref}$ are shown in the subsequent figure.



(a) $rc1^{(0)}_{ref}$ (b) $rc2^{(0)}_{ref}$

Figure 5.20: Refined level-0 automata of roll conveyors 1 and 2

Note that if cb15-rc1 happened once, it can not happen a second time before rc1 has reported "rc1rdy". If rc1 has received 4 workpieces, it is full, and "rc1rdy" happens not

before at least one workpiece is removed from rc1 (which is done by hand). As rc1rdy follows the uncontrollable event rc1wplv, rc1rdy is also an uncontrollable event.
These refined automata are then projected directly to level 2, as they are composed with the level-2 subplant $rts1cb15rts2cb16^{(2),c}$ to form module 5.

Analogously to the example of module 5, the approach presented in [13] and modified in Chapter 4 has been applied to the whole Fischertechnik production plant.
It turns out that all subplants are of manageable size on any level of abstraction. This indicates that the presented method can be used for synthesizing supervisory controllers for large-scale systems.

5.3 Suggestions for Implementation on PLC

For implementation of finite automata we use Step7, which is one of several possible languages to program the SIMATIC S7-300 that controls the Fischertechnik plant model. The SIMATIC framework allows to define all events on the PLC introduced during the modeling process as user-defined datastructures. This could be for example a list of booleans with the name of the respective events. The value of each boolean indicates, whether the event is enabled or disabled. The current state of each automaton implemented on the PLC can simply be stored as an integer.
The remaining task is to correctly implement the hierarchical and decentralized structure. Given a specification for the highest level of abstraction, one possibility of implementation is to compute the controlled behavior $S_i^{(j,j-1)}/G_i^{(j-1)}$ resulting from the low-level supervisor implementations $S_i^{(j,j-1)}$ for each subsystem i of each level $j-1$ offline and then translate it to PLC-code. This results in a hierarchical and decentralized control program with instructions for all possible future behavior beginning at the initial state. One disadvantage of this procedure is, that whenever the highest-level specification is changed, the program has to be recalculated throughout the whole hierarchy.
For that reason we implement each locally controlled and refined behavior $G_{ref,i}^{(j-1)}$ of the subsystems of each level in program code, where at first all high-level events $\Sigma^{(j)}$ are disabled. The controlled behavior of level j according to a given specification in the highest level then enables the respective high-level events out of $\Sigma^{(j)}$ and thus starts the processes in the subsystems of the level $j-1$. Single event control is automatically guaranteed, if only one high-level event out of a choice of enabled high-level events is executed in the lower level by means of priority. If a low-level subsystem is in a state with a branching to several

low-level pathes leading to different high-level events, the PLC program first checks, which one of these high-level events is enabled and then executes only the low-level path corresponding to this high-level event. This is an online implementation of the modified low-level supervisor implementation.

The implementations of all locally controlled subsystems of each level can be seen as a hierarchical structure of general subroutines that are called with the respective high-level events as input variables and the low-level events as output variables for the level below.

If the high-level specification allows maximum performance of the plant, i.e. each state of each subsystem can be reached potentially at least once during the production process, this kind of implementation guarantees a control program of minimal size, as the automaton of each subsystem is the minimal recognizer of all possible future behavior of this subsystem. Furthermore, if the specification of the highest level is changed, only the highest-level supervisor has to be recalculated, the corresponding low-level tasks are generated automatically by the online low-level implementation.

5.4 State Comparison

The approch presented in this report shows the same computational benefits as the approach presented in [13]. This can be seen by the fact that all finite automata in the hierarchy are of manageable size. The following figure shows the size of the controlled plants of the hierarchy of module 5 that are supposed to be implemented on PLC.

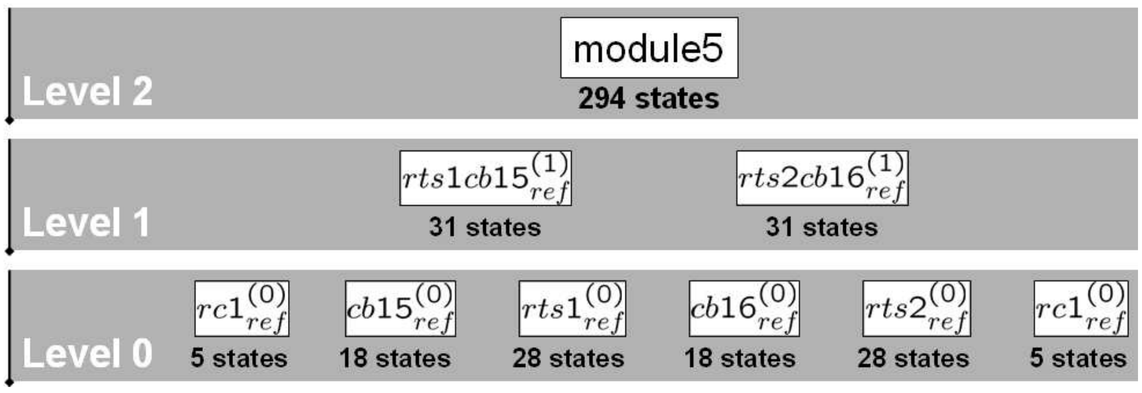


Figure 5.21: Number of states of each automaton of module 5 implemented on PLC

If the monolithic approach is applied, one has to compose all low-level subplants $G_i^{(0)}$ to the overall plant of module 5. This composition has 63 504 states, while the abstracted module 5 that results of the hierarchical and dezentralized approach results in a high-level plant with only 294 states. A monolithic supervisor S, that achieves the same controlled behavior that results from the approach of this report would result in a controlled behavior S/G, which also counts more than 3×10^4 states. Thus it is not advisable to implement this controller on PLC, opposed to the controller that consists of the hierarchical submodules shown in Figure 5.21.

Chapter 6

Conclusion

In this thesis, a method for the hierarchical and decentralized control of discrete event systems was presented, wheras previous results [13] were modified regarding industrial applicability.

Introducing refinement automata e.g. for the identification of unique low-level tasks, the approach in [13] was extended to a multi-level hierarchy with decentralized subsystems at each level, and it was modified such that it can be applied to a class of discrete event systems that do not necessarily fulfill the local nonblocking condition. In this context, the property of single event controllability has been identified as a structural requirement to the system models, together with a condition for the supervisors, which is the so called single event control. Furthermore, a decentralized low-level supervisor implementation was developed that – in combination with the aforementioned properties of the system and the supervisors – results in nonblocking behavior of the controlled system.

Furthermore the method has been applied to an automated manufacturing system with a large number of decentralized subsystems and a hierarchy of several levels of abstraction as a result. It could be shown that the computation of the approach is manageable for systems of praxis relevant size and that the resulting control actions can be implemented on a standard industrial PLC.

However, there are several points of interest remaining for future research efforts. As already indicated in [13], the maximal permissiveness of the control design in [13] and in this report compared to a monolithic approach is to be examined.

An interesting point in this context is the fact, that on the one hand requiring local nonblocking generally for all subsystems of a plant restricts the class of DES to which the approach can be applied, while on the other hand single event control can be too restrictive for those parts of the system that are locally nonblocking a priori. As locally nonblocking systems are always single event controllable as well, there is a way for a controller design to be found that takes into account systems that are composed of both, single event controllable subsystems as well as those that are additionally locally nonblocking.

A second point of interest is as follows. Given a specification for the highest level of the control architecture that designates the desired sequential tasks in the real system, a maximally permissive high-level supervisor often imposes control actions, that lead to branchings and circularities in the controlled behavior of the subsystems at the lowest level. This means that there are several different possibilities of low-level behavior that do not violate the specification for the system up to the worst case of theoretically infinite repitions of certain tasks. This degree of freedom can be used to optimize the controlled behavior with regard to the cost (energy, material, time, money etc.) of each alternative low-level task. So there is room for future investigations on the combination of this approach with new or existing optimization tools for discrete event systems.

Acknoledgement:

Special thanks to the Department of Electrical and Computer Engineering of the Carnegie Mellon University Pittsburgh, in particular to Professor Bruce Krogh, associate head of department, for the hospitality and support.

Bibliography

[1] *Forschungsgruppe Ereignisdiskrete Systeme, Lehrstuhl für Regelungstechnik, Universität Erlangen-Nürnberg*.
www.rt.e-technik.uni-erlangen.de/fgdes/index_en.html

[2] CASSANDRAS, C.G ; LAFORTUNE, S. *Introduction to Discrete Event Systems*. Kluwer. 1999

[3] CUNHA, A.E.C. da ; CURY, J.E.R. ; KROGH, B.H. *An Assume Guarantee Reasoning for Hierarchical Coordination of Discrete Event Systems*. WODES. 2002

[4] ERSOY, G. *Anwendung und Erweiterung Dezentraler Steuerungskonzepte in der Supervisory Control Theory*. Diploma Thesis, Lehrstuhl für Regelungstechnik, Universität Erlangen-Nürnberg. 2004

[5] HUBBARD, P. ; CAINES, P.E.: Dynamical Consistency in Hierarchical Supervisory Control. In: *IEEE TAC* (2002)

[6] JIANG, S. ; CHANDRA, V. ; KUMAR, R.: Decentralized control of discrete event systems with multiple local specializations. In: *Proc. ACC* (2001)

[7] LEDUC, R.J: *Hierarchical Interface-based Supervisory Control*, Department of Electrical & Computer Engineering, University of Toronto, Diss., 2002

[8] LEE, S-H. ; WONG, K.C.: Decentralised control of concurrent discrete-event systems with non-prefix closed local specifications. (1997)

[9] LEE, S-H. ; WONG, K.C.: Stuctural Decentralised Control of Concurrent Discrete-Event Systems. (2002)

[10] RAMADGE, P.J. ; WONHAM, W.M.: Modular Supervisory Contol of Discrete Event Systems. In: *Mathematics of Control of Discrete Event Systems* (1988)

[11] RUDIE, K. ; WONHAM, W.M.: Think globally, act locally: decentralized supervisory control. In: *IEEE TAC* (1992)

[12] SCHMIDT, K. ; MOOR, T. ; PERK, S. *A Hierarchical Architecture for Nonblocking Control of Discrete Event Systems*. Submitted to Mediterranean Conference on Control and Automation. 2005

[13] SCHMIDT, K. ; PERK, S. ; MOOR, T. *Nonblocking Hierarchical Control of Decentralized DES*. Accepted for IFAC. 2005

[14] SCHMIDT, K. ; REGER, J. ; MOOR, T. *Hierarchical Control of Structural Decentralized DES*. WODES. 2004

[15] TORRICO, C.R.C. ; CURY, J.E.R. *Hierarchical Supervisory Control of Discrete Event Systems Based on State Aggregation*. IFAC. 2002

[16] WONG, K.C. ; WONHAM, W.M.: Hierarchical Control of discrete-event systems. In: *Discrete Event Dynamic Systems* (1996)

[17] WONG, K.C. ; WONHAM, W.M.: Modular Control and Coordination of discrete event systems. In: *Discrete Event Dynamic Systems* (1998)

[18] WONHAM, W.M. *Notes on Control of Discrete Event Systems*. Department of Electrical & Computer Engineering, University of Toronto. 2001

[19] YOO, T. ; LAFORTUNE, S.: A generalized framework for decentralized supervisory control of discrete event systems. In: *WODES* (2000)

[20] ZHONG, H. ; WONHAM, W.M.: On the Consistency of Hierarchical Supervision in Discrete-Event Systems. (1990)

Appendix A

Proofs of the Main Result

Remark:

The following proofs are the result of collaborative work together with Dipl.Ing. Klaus Schmidt. Partial results and methods of reasoning have been adopted from [14] and [13].

A.1 Supporting Lemmas

Lemma A.1.1

$$\overline{L_m(S^{hi}/G^{hi})||L^{lo}_{ref,m}} = \overline{L_m(S^{hi}/G^{hi})}||\overline{L^{lo}_{ref,m}}$$

Proof A.1.1

To show:

a) if $s \in \overline{L_m(S^{hi}/G^{hi})||L^{lo}_{ref,m}} \;\Rightarrow\; s \in \overline{L_m(S^{hi}/G^{hi})}||\overline{L^{lo}_{ref,m}}$

This condition is always true.

b) if $s \in \overline{L_m(S^{hi}/G^{hi})}||\overline{L^{lo}_{ref,m}} \;\Rightarrow\; s \in \overline{L_m(S^{hi}/G^{hi})||L^{lo}_{ref,m}}$

To prove this property, let

$s \in \overline{L^{lo}_{ref,m}} \wedge s \in (p^{hi})^{-1}(\overline{L_m(S^{hi}/G^{hi})})$ *with* $s^{hi} = p^{hi}(s) \in \overline{L_m(S^{hi}/G^{hi})}$*.*

Then $\exists u^{hi} \in (\Sigma^{hi})^*$ *such that* $s^{hi}u^{hi} \in L_m(S^{hi}/G^{hi}) \subseteq L^{hi}_m = p^{hi}(L^{lo}_{ref,m})$*.*

Also $\exists u \in (\Sigma^{lo}\dot{\cup}\Sigma_{ref})^*$ *with* $p^{hi}(u) = u^{hi}$ *such that* $su \in L^{lo}_{ref,m}$*.*

As $s^{hi}u^{hi} = p^{hi}(su) \in L_m(S^{hi}/G^{hi})$ *it holds that* $su \in (p^{hi})^{-1}(L_m(S^{hi}/G^{hi}))$*.*

Consequently, $su \in L_m(S^{hi}/G^{hi})||L^{lo}_{ref,m}$ *and thus* $s \in \overline{L_m(S^{hi}/G^{hi})||L^{lo}_{ref,m}}$*.*

Lemma A.1.2

$$\kappa_{L^{lo}_{ref}}(L_m(S^{hi}/G^{hi})||L^{lo}_{ref,m}) \subseteq \overline{L_m(S^{hi}/G^{hi})}||\overline{L^{lo}_{ref,m}}$$

Proof A.1.2

Obviously,

$$\kappa_{L^{lo}_{ref}}(L_m(S^{hi}/G^{hi})||L^{lo}_{ref,m}) \subseteq \overline{L_m(S^{hi}/G^{hi})||L^{lo}_{ref,m}}$$

From Lemma A.1.1 we know

$$\overline{L_m(S^{hi}/G^{hi})||L^{lo}_{ref,m}} \subseteq \overline{L_m(S^{hi}/G^{hi})}||\overline{L^{lo}_{ref,m}}$$

So lemma A.1.2 is valid.

Lemma A.1.3 *Let $t \in L_m(S^{hi}/G^{hi})$. Then $\exists s \in L(S^{hi-lo}/G^{lo}_{ref})$ s.t. $p^{hi}(s) = t$.*

Proof A.1.3

As $L(S^{hi}/G^{hi})$ is nontrivial $\exists t = \sigma_1\sigma_2\cdots\sigma_m \in (\Sigma^{hi})^*$ such that $t \in L_m(S^{hi}/G^{hi}) \subseteq L_m(G^{hi}) = p^{hi}(L^{lo}_{ref,m})$. We want to show that $\exists s := s_1\sigma_1\cdots s_m\sigma_m u \in \kappa_{L^{lo}_{ref}}(\overline{L_m(S^{hi}/G^{hi})||L^{lo}_{ref,m}})$ s.t. $p^{hi}(s) = t$. Note that $\sigma_1 \in p^{hi}\big(\kappa_{L_{\varepsilon,\varepsilon}}(L_{\varepsilon,\varepsilon,\sigma_1})\big)$ as single event controllability according to Definition 4.3.5 is required $\forall i$ with $\sigma_1 \in \Sigma^{lo}_i$.

$$\Rightarrow \exists s_1 \in (\Sigma^{lo} - \Sigma^{hi})^* \text{ such that } s_1\sigma_1 \in \kappa_{L_{\varepsilon,\varepsilon}}(L_{\varepsilon,\varepsilon,\sigma_1})$$

Consequently, $s_1\sigma_1 \in \kappa_{L^{lo}_{ref}}(\overline{L_m(S^{hi}/G^{hi})||L^{lo}_{ref,m}})$. Now, respectively, assume $s_1\sigma_1 s_2\sigma_2\cdots s_{i-1}\sigma_{i-1} \in \kappa_{L^{lo}_{ref}}(\overline{L_m(S^{hi}/G^{hi})||L^{lo}_{ref,m}})$. Then $\sigma_i \in p^{hi}(\kappa_{L_{s_1\cdots\sigma_{i-1},\sigma_1\cdots\sigma_{i-1}}}(L_{s_1\cdots\sigma_{i-1},\sigma_1\cdots\sigma_{i-1},\sigma_i}))$ because of single event controllability. Thus $s_1\sigma_1\cdots s_i\sigma_i \in \kappa_{L^{lo}_{ref}}(\overline{L_m(S^{hi}/G^{hi})||L^{lo}_{ref,m}})$. As this holds for all $i = 1,\ldots,m$, $s_1\sigma_1\cdots s_m\sigma_m \in \kappa_{L^{lo}_{ref}}(\overline{L_m(S^{hi}/G^{hi})||L^{lo}_{ref,m}})$. Then, because $t \in L_m(G^{hi})$ and $s_1\sigma_1\cdots s_m\sigma_m \in L_{t,e}$, it follows from Lemma 3.2 in [13] that $\exists u \in (\Sigma^{lo} - \Sigma^{hi})^*$ such that $s_1\sigma_1\cdots s_m\sigma_m u \in L^{lo}_{ref,m}$. For showing that $s_1\sigma_1\cdots s_m\sigma_m u \in \kappa_{L^{lo}_{ref}}(\overline{L_m(S^{hi}/G^{hi})||L^{lo}_{ref,m}})$, we distinguish two cases. First, assume that $S^{hi}(t)\cap\Sigma^{hi}(t) = \emptyset$. Then, because of marked state controllability, $\kappa_{L_{t,s_1\cdots\sigma_m}}(L_{t,s_1\cdots\sigma_m,\emptyset}) \neq \emptyset$. Thus for $u \in \kappa_{L_{t,s_1\cdots\sigma_m}}$, it holds that $s_1\cdots\sigma_m u \in \kappa_{L^{lo}_{ref}}(L_m(S^{hi}/G^{hi})||L^{lo}_{ref,m})$ as $s_1\cdots\sigma_m u \in L^{lo}_{ref,m}$. In the second case $S^{hi}(t)\cap\Sigma^{hi}(t) \neq \emptyset$. Thus for any $\sigma \in S^{hi}(t)\cap\Sigma^{hi}(t)$, $\exists u'$ s.t. $s_1\cdots\sigma_m u'\sigma \in L(S^{hi-lo}/G^{lo}_{ref}) = \overline{\kappa_{L^{lo}_{ref}}(L_m(S^{hi}/G^{hi})||L^{lo}_{ref,m})}$. But then, because of marked state consistency, $\exists u \leq u'$ s.t. $s_1\cdots\sigma_m u \in L^{lo}_{ref,m}$ and thus $s_1\cdots\sigma_m u \in \kappa_{L^{lo}_{ref}}(L_m(S^{hi}/G^{hi})||L^{lo}_{ref,m})$.

$$\Rightarrow\ s_1\sigma_1\cdots s_m\sigma_m u \in \kappa_{L^{lo}_{ref}}(L_m(S^{hi}/G^{hi})||L^{lo}_{ref,m})$$

Lemma A.1.4 *Let H be a marked state controllable EHDCS and let $s \in L(S^{hi-lo}/G^{lo}_{ref})$, $s^{hi} := p^{hi}(s) \in L(S^{hi}/G^{hi})$. If i is such that $\nexists u_i \in (\Sigma_i - \Sigma^{hi})^*$ s.t. $s_i u_i \in L_m(S_i^{hi-lo}/G^{lo}_{ref,i})$ for $s_i := p_i(s)$, then $\exists t \in (\Sigma^{hi})^*$ s.t. $s^{hi}t \in L_m(S^{hi}/G^{hi})$ and $p_i(t) \neq \varepsilon$.*

Proof A.1.4 *Assume that $\Sigma_i^{hi}(s_i^{hi}) \cap S_i^{hi}(s_i^{hi}) = \emptyset$. Then $\exists u_i \in (\Sigma_i - \Sigma^{hi})^*$ s.t. $s_i u_i \in L_m(S_i^{hi-lo}/G^{lo}_{ref,i})$ because of marked state controllability. Thus it must hold that $\Sigma_i^{hi}(s_i^{hi}) \cap S_i^{hi}(s_i^{hi}) \neq \emptyset$. Thus $\exists t_i \neq \varepsilon$ s.t. $s_i^{hi}t_i \in L_m(S_i^{hi}/G_i^{hi})$. Also $\exists t \in (\Sigma^{hi})^*$ with $p_i(t) = t_i \neq \varepsilon$ and $s^{hi}t \in L_m(S^{hi}/G^{hi})$ as $L_m(S_i^{hi}/G_i^{hi}) = p_i(L_m(S^{hi}/G^{hi}))$.*

Lemma A.1.5 *Let $s_i \in L(S_i^{hi-lo}/G^{lo}_{ref,i})$, $s_i^{hi} := p^{hi}(s_i)$ and s^{hi} with $p_i(s^{hi}) = p^{hi}(s_i)$. Then $\exists u_i \in (\Sigma_i - \Sigma^{hi})^*$ s.t. $s_i u_i \sigma \in L(S^{hi-lo}/G^{lo}_{ref,i}) \cap L_{en,s_i^{hi}}$ for $\sigma \in S^{hi}(s^{hi}) \cap \Sigma_i$.*

Proof A.1.5 *Let s_i, s_i^{hi} and s^{hi} as in Lemma A.1.5 and write $s_i = s_{en}u$ s.t. $s_{en} \in L_{en,s_i^{hi}}$ and $u \in (\Sigma_i - \Sigma^{hi})^*$. As $\sigma \in S^{hi}(s^{hi}) \cap \Sigma_i$, it holds that $\sigma \in S_i^{hi}(s_i^{hi})$. Because of local nonblocking of $\kappa_{L_{s_{en},s_i^{hi}}}(L_{s_{en},s_i^{hi},\gamma})$ (see single event controllability and single event control), $\exists u_i \in (\Sigma - \Sigma^{hi})^*$ s.t. $s_i u_i \sigma \in L(S^{hi-lo}/G^{lo}_{ref,i}) \cap L_{en,s_i^{hi}}$.*

Lemma A.1.6 *Let $s_i \in L(S_i^{hi-lo}/G^{lo}_{ref,i})$, $s_i^{hi} := p^{hi}(s_i)$ and s^{hi} with $p_i(s^{hi}) = p^{hi}(s_i)$. If $t \in (\Sigma^{hi})^*$ s.t. $s^{hi}t \in L(S^{hi}/G^{hi})$, then $\exists u_i \in \Sigma_i^*$ s.t. $s_i u_i \in L(S_i^{hi-lo}/G^{lo}_{ref,i}) \cap L_{en,s_i^{hi}p_i(t)}$.*

Proof A.1.6 *Write $p_i(t) = \sigma_0\sigma_1\sigma_2\cdots\sigma_m$ with $\sigma_i \in \Sigma_i^{hi}$ for $i = 1,\ldots,m$ and $\sigma_0 = \varepsilon$. We show that $\exists u_0\sigma_0 u_1\sigma_1 \cdots u_m\sigma_m \in L(S_i^{hi-lo}/G^{lo}_{ref,i})$ with $u_j \in (\Sigma_i - \Sigma^{hi})^*$ for $j = 0,\ldots,m$ by induction. For $j = 0$ we have $u_0 = \varepsilon$ and $s_i u_0 \sigma_0 \in L(S_i^{hi-lo}/G^{lo}_{ref,i})$. Now assume that $s_i u_0\sigma_0 u_1 \cdots u_{j-1}\sigma_{j-1} \in L(S_i^{hi-lo}/G^{lo}_{ref,i})$. Then, because of Lemma A.1.5, $\exists u_j \in (\Sigma_i - \Sigma^{hi})^*$ s.t. $s_i u_0\sigma_0 u_1 \cdots u_{j-1}\sigma_{j-1}u_j\sigma_j \in L(S_i^{hi-lo}/G^{lo}_{ref,i})$. As this is valid for all $j = 1,\ldots,m$, it holds that $\exists u := u_0\sigma_0 u_1\sigma_1 \cdots u_m\sigma_m$ s.t. $s_i u \in L(S_i^{hi-lo}/G^{lo}_{ref,i})$. Also $s_i u \in L_{en,s_i^{hi}p_i(t)}$ and thus $s_i u \in L(S_i^{hi-lo}/G^{lo}_{ref,i}) \cap L_{en,s_i^{hi}p_i(t)}$.*

Lemma A.1.7 *Let $(G, p^{hi}, G^{hi}, S^{hi}, S^{lo})$ be a EHDCS with marked state accepting hierarchical abstractions $(G^{lo}_{ref,i}, p^{hi}, G_i^{hi})$ and let S_i^{hi-lo} be decentralized modified supervisors for $i = 1,\ldots,n$. Also let $s_{en} \in L_{en,s_j^{hi}} \cap L(S_j^{hi-lo}/G^{lo}_{ref,j})$ for $s_j^{hi} \in L_m(S_j^{hi}/G_j^{hi})$. Then $\exists u_j \in (\Sigma_j - \Sigma^{hi})^*$ s.t. $s_{en}u_j \in L_m(S_j^{hi-lo}/G^{lo}_{ref,j})$.*

Proof A.1.7 *Assume $\nexists u_j \in (\Sigma_j - \Sigma^{hi})^*$ s.t. $s_j u_j \in L_m(S_j^{hi-lo}/G^{lo}_{ref,j})$. $L(S_j^{hi-lo}/G^{lo}_{ref,j}) = \overline{L_m(S_j^{hi-lo}/G^{lo}_{ref,j})}$ because of $L_m(S_j^{hi-lo}/G^{lo}_{ref,j}) = \kappa_{L^{lo}_{ref,j}}(L_m(S^{hi}/G^{hi}||L_m(G^{lo}_{ref,i}))$. Thus $\exists u \in \Sigma_j^*$ s.t. $s_i u \in L_m(S_j^{hi-lo}/G^{lo}_{ref,j})$. But then $u = u'\sigma u''$ with $u' \in (\Sigma_j - \Sigma^{hi})^*$, $\sigma \in \Sigma^{hi}$ and $u'' \in \Sigma_i^*$. Hence $u' \in L_{ex,s_j^{hi}}$ and because of marked state acceptance, $\exists u_j \leq u'$ s.t. $s_j u_j \in L_m(S_j^{hi-lo}/G^{lo}_{ref,j})$ which contradicts the assumption.*

A.2 Hierarchical Consistency

To show:

$$p^{hi}\big(L(S^{hi-lo}/G^{lo}_{ref})\big) = p^{hi}\big(\kappa_{L^{lo}_{ref}}(L_m(S^{hi}/G^{hi})||L^{lo}_{ref,m})\big) = L(S^{hi}/G^{hi})$$

This splits into:
(1) $p^{hi}\big(\kappa_{L^{lo}_{ref}}(L_m(S^{hi}/G^{hi})||L^{lo}_{ref,m})\big) \supseteq L(S^{hi}/G^{hi})$
(2) $p^{hi}\big(\kappa_{L^{lo}_{ref}}(L_m(S^{hi}/G^{hi})||L^{lo}_{ref,m})\big) \subseteq L(S^{hi}/G^{hi})$

(1) Proof:
Let $s^{hi} \in L(S^{hi}/G^{hi})$. Then $\exists t \in (\Sigma^{hi})^*$ s.t. $s^{hi}t \in L_m(S^{hi}/G^{hi})$. But because of Lemma A.1.3, $\exists s \in L_m(S^{hi-lo}/G^{lo}_{ref})$ s.t. $p^{hi}(s) = s^{hi}t$. Consequently, $\exists s' \leq s$ with $p^{hi}(s') = s^{hi}$ and it holds that $s' \in L(S^{hi-lo}/G^{lo}_{ref})$.

(2) Proof:
to show: if $s^{hi} \in L(S^{hi-lo}/G^{lo}_{ref}) \ \Rightarrow \ p^{hi}(s) \in L(S^{hi}/G^{hi})$
$\rightarrow$ proof by induction: if $s \in L(S^{hi-lo}/G^{lo}_{ref}) \ \Rightarrow \ p^{hi}(s\sigma) \in L(S^{hi}/G^{hi})$
1. It holds that $\varepsilon \in L(S^{hi-lo}/G^{lo}_{ref})$ as S^{hi-lo} is admissible. $p^{hi}(\varepsilon) = \varepsilon \in L(S^{hi}/G^{hi})$
2. Now let $s \in L(S^{hi-lo}/G^{lo}_{ref})$ with $p^{hi}(s) = s^{hi} \in L(S^{hi}/G^{hi})$ and $\sigma \in (\Sigma^{lo}\dot{\cup}\Sigma_{ref})$ such that $s\sigma \in L(S^{hi-lo}/G^{lo}_{ref})$
a) if $\sigma \in (\Sigma^{lo} - \Sigma^{hi})$, then $p^{hi}(s\sigma) = p^{hi}(s)p^{hi}(\sigma) = p^{hi}(s)\varepsilon = s^{hi}\varepsilon = s^{hi} \in L(S^{hi}/G^{hi})$.
b) now let $\sigma \in \Sigma^{hi}$: because of Lemma A.1.2 $L(S^{hi-lo}/G^{lo}_{ref}) \subseteq \overline{L_m(S^{hi}/G^{hi})}||\overline{L^{lo}_{ref,m}}$ and thus

$$s\sigma \in (p^{hi})^{-1}(\overline{L_m(S^{hi}/G^{hi})})$$

As S^{hi}/G^{hi} is nonblocking $\overline{L_m(S^{hi}/G^{hi})} = L(S^{hi}/G^{hi})$, so

$$s\sigma \in (p^{hi})^{-1}(L(S^{hi}/G^{hi})) \ \Rightarrow \ p^{hi}(s\sigma) \in L(S^{hi}/G^{hi})$$

A.3 Nonblocking

Proof A.3.1
For proving nonblocking behavior, we first note that it holds that $L(S^{hi-lo}/G^{lo}_{ref}) \neq \emptyset$, see Lemma A.1.3.
Now assume that $s \in L(S^{hi-lo}/G^{lo}_{ref})$ and $s^{hi} = p^{hi}(s) \in L(S^{hi}/G^{hi})$. It has to be shown

that $s \in \overline{L_m(S^{hi-lo}/G^{lo}_{ref})}$. Because of the definition of the low-level supervisor S^{hi-lo} and the decentralized implementation according to Definitions 4.3.2 and 4.3.3, $s_i := p_i(s) \in L(S_i^{hi-lo}/G^{lo}_{ref,i})$ and $s_i^{hi} := p_i(s^{hi}) \in L(S_i^{hi}/G_i^{hi}) := p_i(L(S^{hi}/G^{hi}))$. Considering all local subsystems $G^{lo}_{ref,i}$, $i = 1, 2, \dots, n$, we have to distinguish two cases.
Let $I_0 := \{i | \nexists u_i \in (\Sigma_i - \Sigma^{hi})^*$ such that $s_i u_i \in L_m(S_i^{hi-lo}/G^{lo}_{ref,i})\}$. The following algorithm provides a high-level string which leads to a marked high-level state and drives each controlled high-level subsystem S_i^{hi}/G_i^{hi} with $i \in I_0$ to a marked state.

1. *$k = 1,\ I = I_0$.*
2. *choose $i_k \in I$.*
3. *find a string $t_k \in (\Sigma^{hi})^*$ such that $s^{hi} t_1 \cdots t_k \in L_m(S^{hi}/G^{hi})$ and $p_{i_k}(t_k) \neq \varepsilon$.*
4. *remove all j with $p_j(t_{i_k}) \neq \varepsilon$ from I.*
5. ***if** $I = \emptyset$: set $k^* = k$ and **terminate***
   ***else** $k := k + 1$ and **go to** 2.*

Because of Lemma A.1.4, the string t_k in item 3. of the algorithm always exists. Also note that the algorithm terminates as I is a finite index set which is reduced in every loop of the algorithm.

Now define $t := t_1 \cdots t_{k^*}$. $s^{hi} t \in L_m(S^{hi}/G^{hi})$ and $\forall i, s_i^{hi} p_i(t) \in L_m(S_i^{hi}/G_i^{hi})$ and introduce $\mathcal{J} := \{j | p_j(t) \neq \varepsilon\}$. Because of Lemma A.1.6, $\forall i \in \mathcal{J}$, $\exists u_i \in \Sigma_i^*$ s.t. $s_i u_i \in L(S_i^{lo}/G_i^c) \cap L_{en, s_i^{hi} p_i(t)}$. Then, because of Lemma A.1.7, $\exists \bar{u_i} \in (\Sigma - \Sigma^{hi})^*$ s.t. $s_i u_i \bar{u_i} \in L_m(S_i^{lo}/G_i^c)$. For $i \notin \mathcal{J}$, we define $u_i := \varepsilon$ and we note that $\exists \bar{u_i} \in (\Sigma - \Sigma^{hi})^*$ s.t. $s_i u_i \bar{u_i} \in L_m(S_i^{lo}/G_i^c)$ as $i \notin I_0$. Then $\forall u \in ||_{i=1}^n s_i u_i \bar{u_i}$, it holds that $su \in ||_{i=1}^n L_m(S_i^{hi-lo}/G^{lo}_{ref,i})$ and $p^{hi}(su) = s^{hi} t \in L_m(S^{hi}/G^{hi})$ and thus $su \in L_m(S^{hi}/G^{hi}) || \left(||_{i=1}^n L_m(S_i^{lo}/G_i^c) \right) = L_m(S^{lo}/G^c)$. Hence $s \in \overline{L_m(S^{lo}/G^c)}$.