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

Technical Report

Sebastian Perk, Klaus Schmidt and Thomas Moor

Lehrstuhl für Regelungstechnik
Friedrich-Alexander Universität Erlangen-Nürnberg
Cauerstraße 7, D–91058 Erlangen, Germany
sebastian.perk@rt.eei.uni-erlangen.de

November 2004

Contents

1 Introduction

Over the past years several new methods for the control of discrete event systems (DES) based on the framework provided by P.J. Ramadge und W.M. Wonham [10] 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.

In [8], a new approach for the control of a class of decentralized discrete event systems has been presented that adresses the complexity combining a hierarchical and a decentralized design 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.

In this technical report, the above-mentioned approach is extended and modified. The extended method is then applied to an automated manufacturing system.

For a detailed introduction to discrete event systems and the supervisory control theory (SCT) refer to [10, 2]. Basic notations and definitions of the SCT and previous results of hierarchical control of decentralized DES are adopted from [8] and [7].

The outline of this report is as follows.
In Section 2, an extension and modification of the hierarchical and decentralized control system presented in [8] 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.
In Section 3, 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).
The report is concluded by a prospect to possible future research activities.

2 Hierarchical Control of Decentralized Discrete Event Systems: Multi-Level Extension of the Hierarchical and Decentralized Approach

In this section the approach presented in [8] 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.

2.1 Multi-Level Hierarchy

An example of this decentralized multi-level structure with 4 subsystems and 3 levels in the hierarchy is shown in Figure 2.1. Generally, there is no restriction on the number of levels and the number of subsystems at each level.

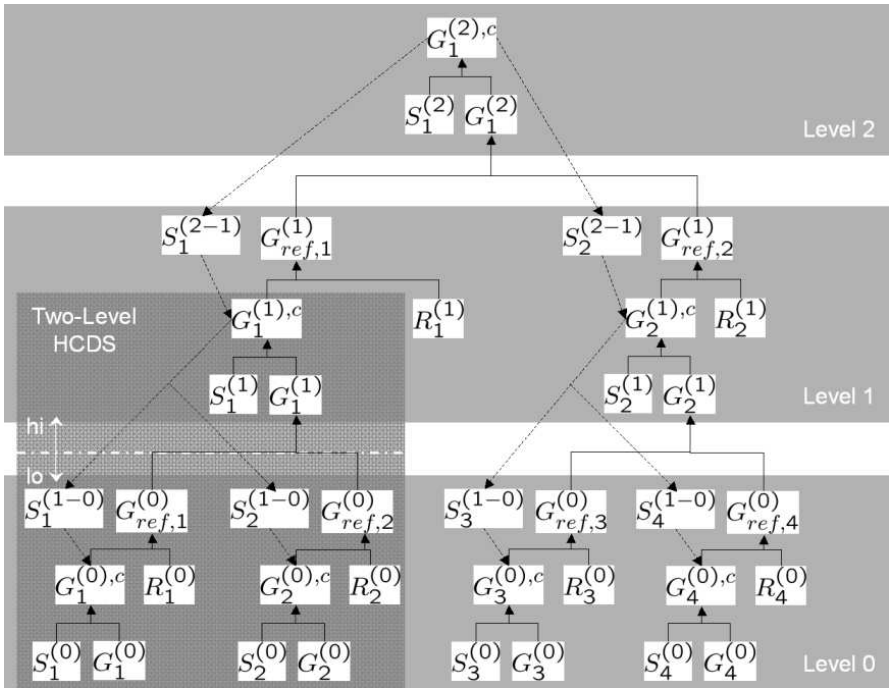


Figure 2.1: Example of a decentralized 3-level hierarchy

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 $S_i^{(0)}/G_i^{(0)}$ $(i = 1,\ldots,4)$ 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 [4].

For convenience and to be able to easily apply the notations of [8], 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 2.1. For the general structure of a two-level hierarchy see Figure 2.2.

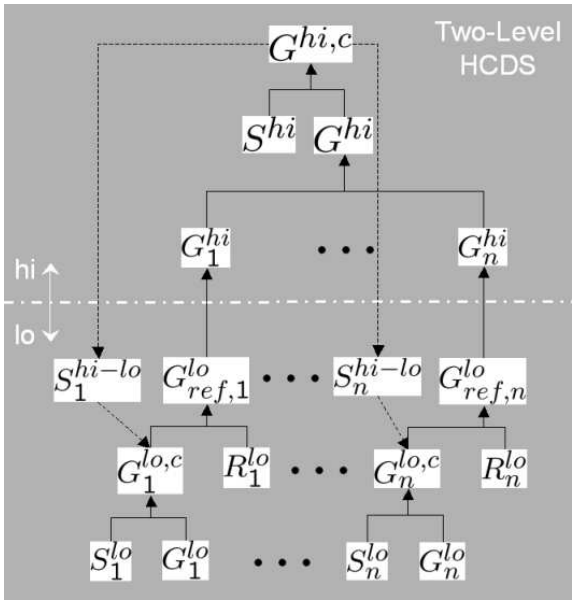


Figure 2.2: Two-level hierarchy

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]

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

Definition 2.1 (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 [8] 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 $S_i^{lo}: L_i^{lo} \rightarrow \Gamma_i^{lo}$, where Γ_i^{lo} are the respective control patterns. The resulting low-level closed-loop languages are denoted*

[1] For clarity reasons the local high-level subsystems G_i^{hi} are not shown in Figure 2.1.

$L_i^{lo,c} := L(S_i^{lo}/G_i^{lo})$
$L_{i,m}^{lo,c} := L_i^{lo,c} \cap L_{i,m}^{lo}$,
$L^{lo,c} := ||_{i=1}^{n} L_i^{lo,c}$
$L_m^{lo,c} := ||_{i=1}^{n} L_{i,m}^{lo,c} = L^{lo,c} \cap L_m^{lo}$.
$G^{lo,c}$ 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*
$L_{ref,i}^{lo} := L_i^{lo,c} || K_{ref,i}^{lo}$,
$L_{ref,i,m}^{lo} := L_{i,m}^{lo,c} || K_{ref,i,m}^{lo}$
are the refined languages. The canonical recognizer of $L_{ref,i,m}^{lo}$ is denoted $G_{ref,i}^{lo}$,i.e $L(G_{ref,i}^{lo}) = L_{ref,i}^{lo}$ and $L_m(G_{ref,i}^{lo}) = L_{ref,i,m}^{lo}$.

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

- *The high-level plant is given by $G^{hi} := ||_{i=1}^{n} G_i^{hi}$, so $L^{hi} := ||_{i=1}^{n} L_i^{hi}$ and $L_m^{hi} := ||_{i=1}^{n} L_{i,m}^{hi}$. Each pair G_i^{hi}, G_j^{hi} of abstracted subsystems is synchronized by shared events if $\Sigma_i^{hi} \cap \Sigma_j^{hi} \neq \emptyset$.*

- *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 $S^{hi-lo} : L^{lo,c} \to$*

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

Γ^{hi-lo} *with the control pattern* $\Gamma^{hi-lo} \subseteq 2^{(\Sigma^{lo} \cup \Sigma_{ref})}$ *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_i^{hi-lo}.

2.2 Refinement Automata

One modification of the approach described in [8] 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 2.2 (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* $\exists u \in (\Sigma_{ref})^*$, $\sigma_{uc} \in \Sigma^{lo}_{uc}$ *such that* $su\sigma_{uc} \in R^{lo}_{ref}$, *then* $\exists u' \sigma_{uc}$ *such that* $s'u'\sigma_{uc} \in R^{lo}_{ref}$.

2. K^{lo}_{ref} *and* $K^{lo}_{ref,m}$ *are not restrictive with respect to* L^{lo} *and* L^{lo}_m.

   $$p^{lo}(L^{lo}_{ref}) = L^{lo}, \;\; p^{lo}(L^{lo}_{ref,m}) = L^{lo}_m$$

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

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

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

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^{lo}_{ref} 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^{lo}_{ref} can be implemented by an admissible low-level supervisor as stated in the following lemma.

Lemma 2.1 (Consistency of Controllability) *Let G^{lo} be a finite automaton and R^{lo} be a refinement automaton generating K^{lo}_{ref} with the refined language L^{lo}_{ref} as stated in Definition 2.2. The requirements on K^{lo}_{ref} in Definition 2.2 guarantee that any behavior E^{lo}_{ref} that is controllable with respect to the refined language L^{lo}_{ref} can be implemented by an admissible low-level supervisor:*

$$\forall E^{lo}_{ref} \in \mathcal{C}(L^{lo}_{ref}): \quad p^{lo}(E^{lo}_{ref}) \in \mathcal{C}(L^{lo})$$

A proof is given in [6]. A further modification of the approach described in [8] 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 [8].

2.3 Modified Supervisor Implementation

We recall the Definition of a locally nonblocking hierarchical abstraction (G, p^{hi}, G^{hi}) in [8].

Definition 2.3 (Locally Nonblocking Hierarchical Abstraction)
Let (G, p^{hi}, G^{hi}) be a a hierarchical abstraction. The high-level string $s^{hi} \in \overline{L^{hi}_m}$ 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 $p^{hi}(s)\sigma \in \overline{L^{hi}_m}$:

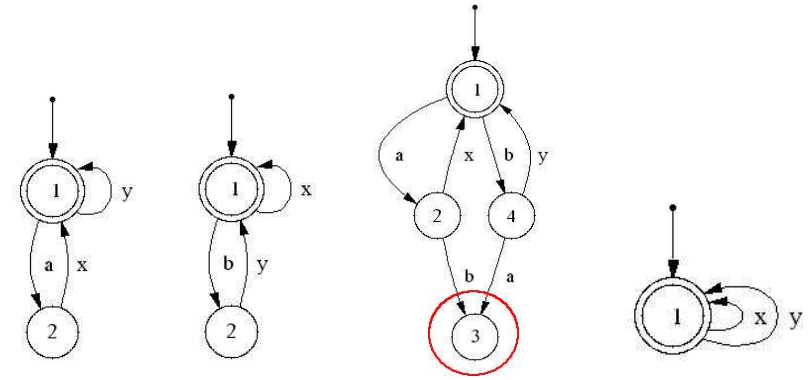
$$\exists u_\sigma \in (\Sigma - \Sigma^{hi})^* \text{ such that } s u_\sigma \sigma \in L(G)$$

where $\Sigma^{hi}(s^{hi}) := \{\sigma \in \Sigma^{hi} | s^{hi}\sigma \in L^{hi}\}$ is the set of all high-level events that are possible in G^{hi} after string s^{hi}.
(G, p^{hi}, G^{hi}) is locally nonblocking if the above condition holds $\forall s^{hi} \in \overline{L^{hi}_m}$.

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}. In many application examples this property can be fulfilled only with restrictive models of the subplants,

as different local extensions of low-level strings usually have influence on the subsequent high-level behavior. 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) $G1^{lo}_{ref,1}$ (b) $G^{lo}_{ref,2}$ (c) $G^{lo}_{ref,1}||G^{lo}_{ref,2}$ (d) G^{hi}

Figure 2.3: 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 2.3 (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 2.4 (Modified Supervisor Implementation) *Given an extended hierarchical control system of Definition 2.1, 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 successor strings are allowed at any state in G^{lo} , 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 2.1, 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 2.5 (Decentralized Modified Supervisor) *Given an extended hierarchical control system of Definition 2.1, 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 .*

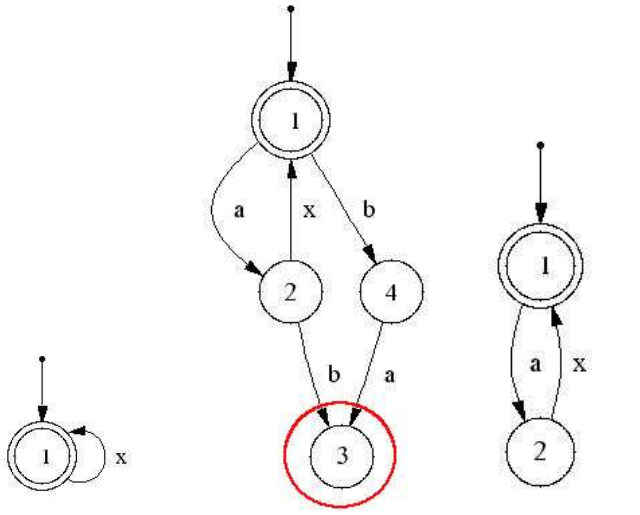
The overall behavior of the subsystems controlled by the decentralized modified supervisors is implemented according to the subsequent definition.

Definition 2.6 (Decentralized Modified Supervisor Implementation) *Given an extended hierarchical control system of Definition 2.1, the decentralized modified supervisor implementation is defined as*

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

So 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 2.3, 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 2.4 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 2.4 b)). Different from that, the decentralized supervisor S_2^{hi-lo} 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 2.4 (c).



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

Figure 2.4: 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 2.3 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 2.7 (Single Event Control) *Given an extended hierarchical and decentralized control system with a modified supervisor implementation as in Definition 2.4, 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 leading to different controllable 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.
For that purpose, we recall the definition of entry strings stated in [8].

Definition 2.8 (Entry Strings) *Given $s^{hi} \in L^{hi}$, the set of entry strings of s^{hi} is defined as*

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

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}.

Definition 2.9 (Single Event Controllability) *Let H be an extended hierarchical and decentralized control system. For a given high-level string $s^{hi} \in L_i^{hi}$ with a controllable high-level successor event $\alpha \in \Sigma_i^{hi}(s^{hi}) \cap \Sigma_{c,i}^{hi}$, the language*

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

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

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

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

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 Section 4.
Further required system properties are the *marked state acceptance* of the hierarchical abstractions and the *mutual controllability* of the abstracted subplants, for the latter see also [5]. Another property identified in [7] and required for this approach is the *marked state controllability*.

Definition 2.10 (Marked State Acceptance) *Let $(G_{ref}^{lo}, p^{hi}, G^{hi})$ be a hierarchical abstraction. $(G_{ref}^{lo}, p^{hi}, G^{hi})$ 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 every exit string corresponding to a marked high-level string has a marked predecessor string in the low level.[8]

Definition 2.11 (Mutual Controllability, High-Level) *Given a EHDCS, two high-level languages L_i^{hi} and L_j^{hi} are said to be mutually controllable if* $\forall i, j = 1...n,\ i \neq j$

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

This means that an uncontrollable shared event possible in the abstracted subplant G_j^{hi} is never prevented from occuring because of the synchronization with another subplant G_i^{hi}.[8]
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.[7]

Definition 2.12 (Marked State Controllability) *Let $(G_{ref,i}^{lo}, p^{hi}, G_i^{hi})$ be a hierarchical abstraction and let $\gamma^{hi} \in \Gamma_i^{hi}$ be the control patterns of S_i^{hi} with $L(S_i^{hi}/G_i^{hi}) = p_i(L(S^{hi}/G^{hi})$. Choose $s^{hi} \in L_{i,m}^{hi}$ such that $\gamma^{hi} \cap \Sigma_i^{hi}(s^{hi}) = \emptyset$. The string s^{hi} is marked state controllable if*

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

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

The above properties are used in [8, 7] to show nonblocking behavior in the low level and are met in the application example of this work.

Theorem 2.1 (Main Result [6])
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_i^{hi} are mutually controllable.*
- *The low-level supervisors are given by the decentralized modified supervisor implementation of Definition 2.5 implemented according to Definition 2.6.*

- H *is single event controllable.*
- *All high-level supervisors impose single event control according to Definition 2.7.*

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

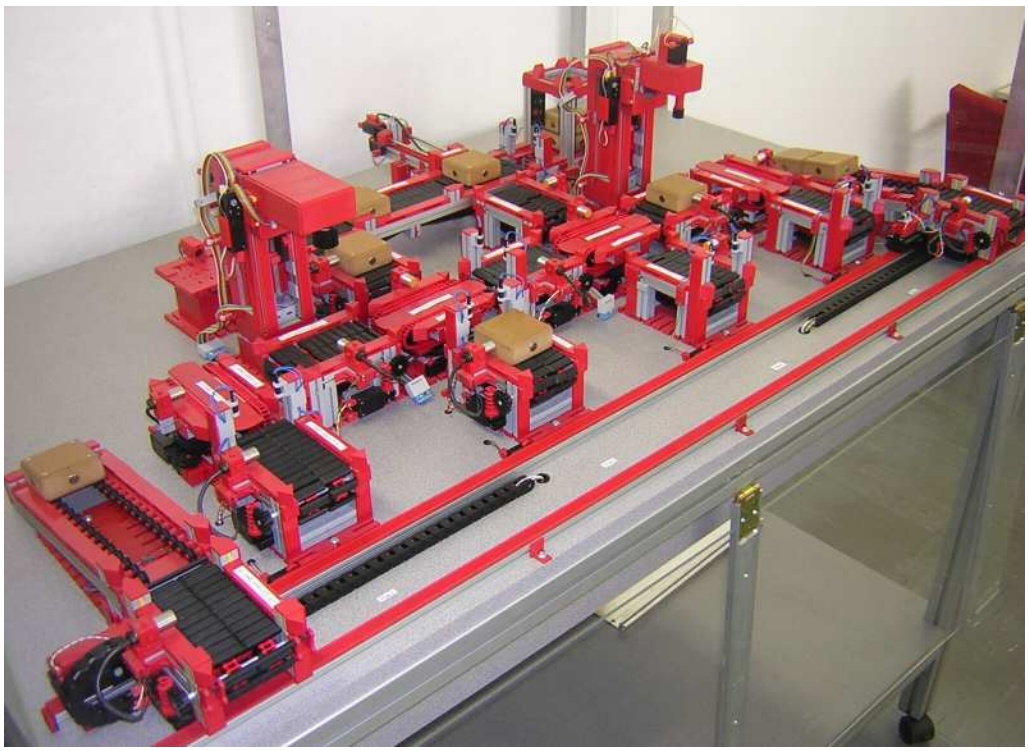
See [6] for the proof of the above theorem.
With this result, the approach of [8] is extended to a multi-level hierarchy with refinements for the identification of low-level strings and a modified decentralized supervisor implementation that, in combination with system properties different from local nonblocking, results in nonblocking behavior of the controlled system.
We will see in the following section 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.

3 Case Study: An Automated Manufacturing System

In this section, 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 Section 2. Then the application example is used to evaluate the computational complexity of this approach. The section is concluded with suggestions for a modular implementation on a programmable logic controller (PLC) with online generation of the low-level control action.

3.1 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 [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.



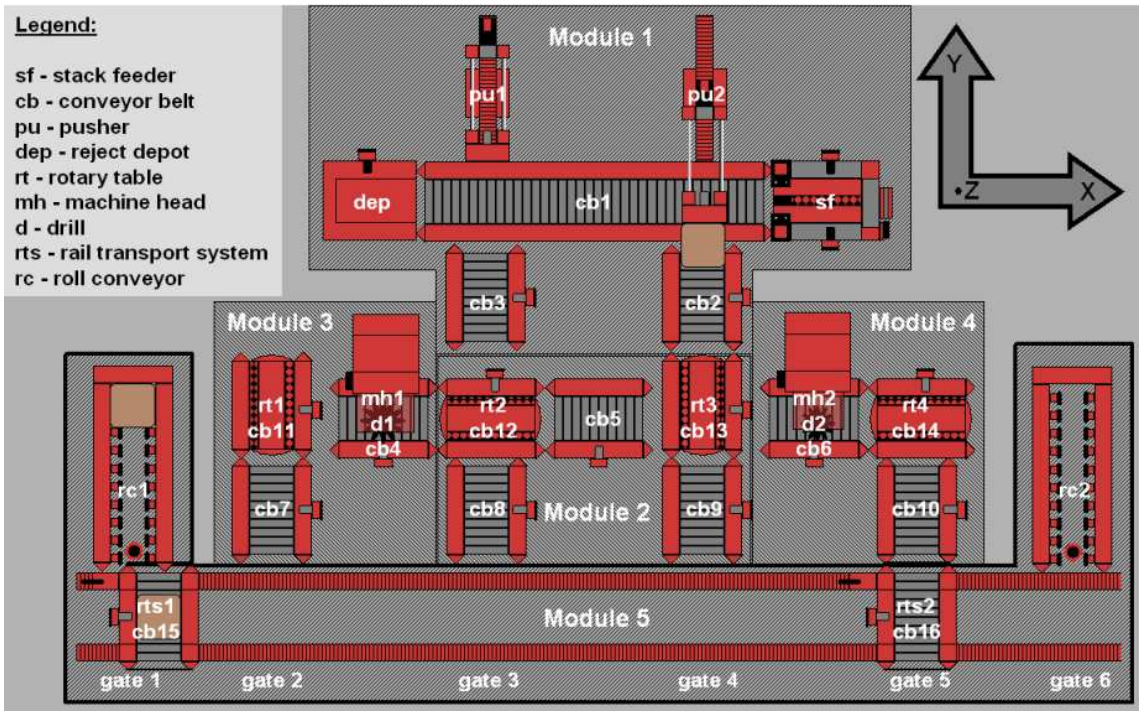


Figure 3.5: Fischertechnik automated manufacturing system with schematic overwiev

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 actor signals like the movement of each belt according to a control program. As any of these sensor and actor 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 Section 2 to the plant is described in the following section.

3.2 Modeling and Supervisor Implementation

The theoretical approach presented in Section 2 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 3.5. 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.

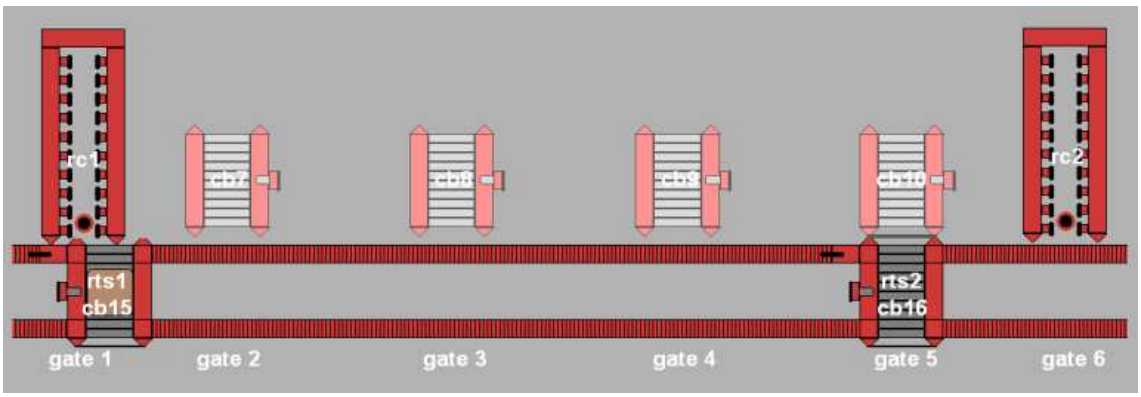


Figure 3.6: Module 5

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

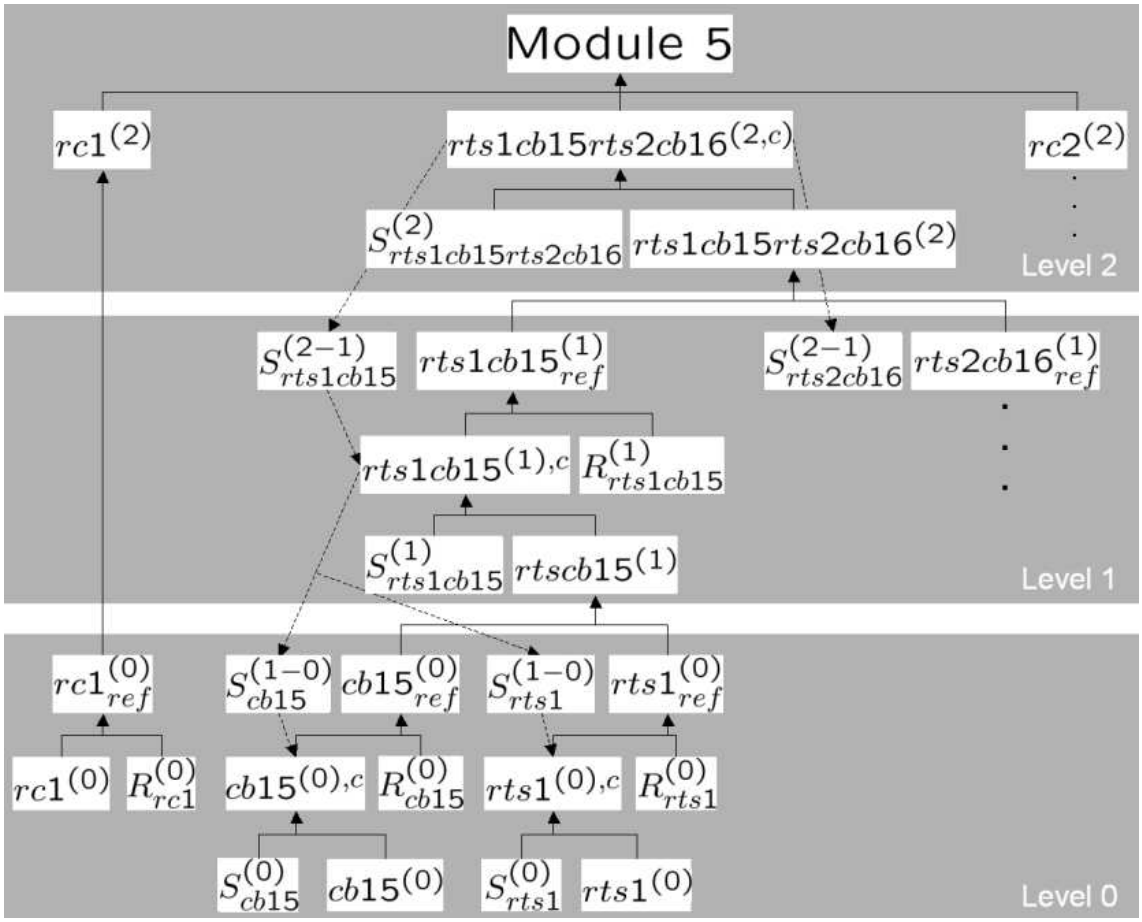


Figure 3.7: Hierarchy of module 5

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.

3.2.1 Conveyor Belt cb15

The conveyor belt can be seen as an elemantary component of the plant as it consists of one actor (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. [3, 9, 8] and is shown in Figure 3.8. For detailed explanations on how to derive a finite automaton from a discrete event system, see [3]. The resulting model of subplant $cb15^{(0)}$ is shown in the subsequent picture.

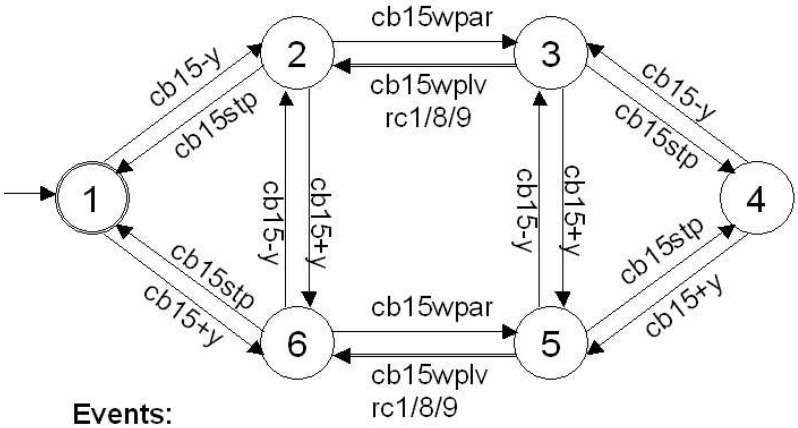


Figure 3.8: Low-level model of conveyor belt cb15: $cb15^{(0)}$

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.

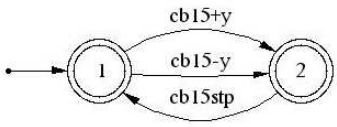


Figure 3.9: Specification $cb15^{(0)}_{spec1}$

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

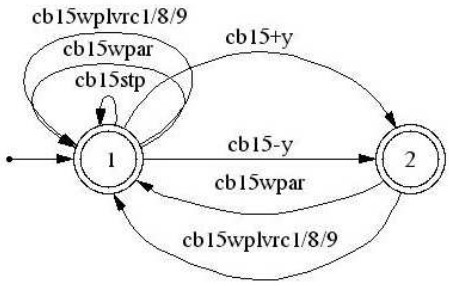


Figure 3.10: Specification $cb15^{(0)}_{spec2}$

- 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).

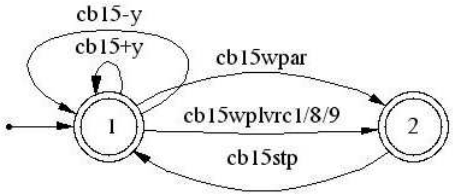


Figure 3.11: Specification $cb15^{(0)}_{spec3}$

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

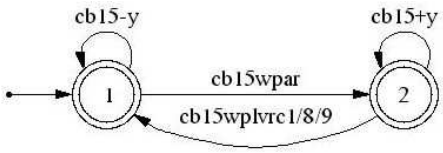


Figure 3.12: Specification $cb15^{(0)}_{spec4}$

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.

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

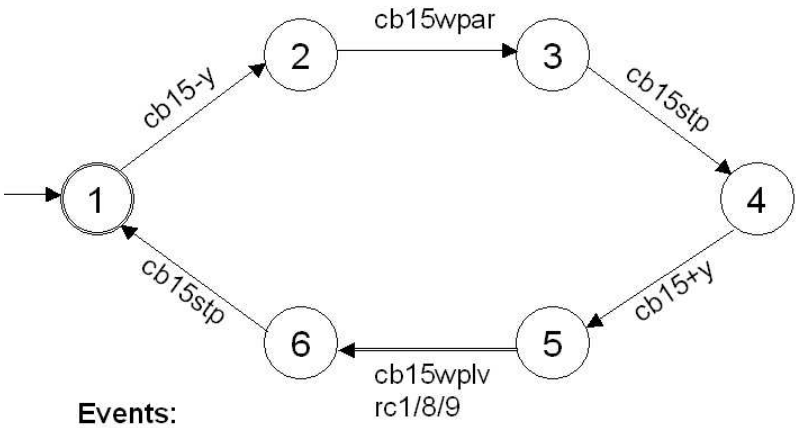


Figure 3.13: Controlled low-level behavior of cb15: $cb15^{(0),c}$

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 end 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 [4].

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 3.14

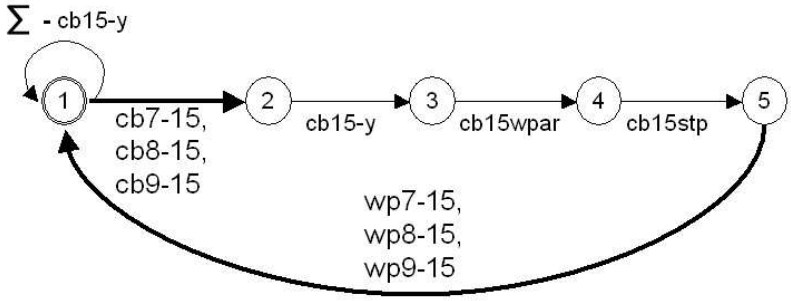


Figure 3.14: Refinement automaton $R^{(0)}_{cb15,1}$

In the automaton in Figure 3.14, 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"* [3], which is identified by the following refinement automaton $R^{(0)}_{cb15,2}$.

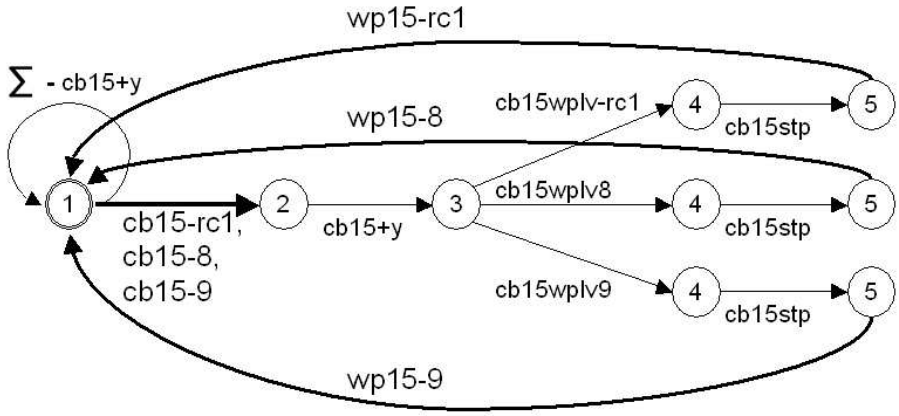


Figure 3.15: Refinement automaton $R^{(0)}_{cb15,2}$

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 3.16.

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

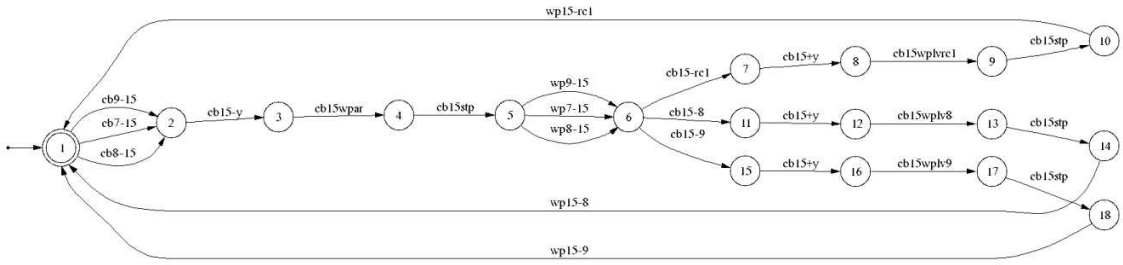


Figure 3.16: Refined automaton $cb15^{(0)}_{ref}$

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 2.2.

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 3.17.

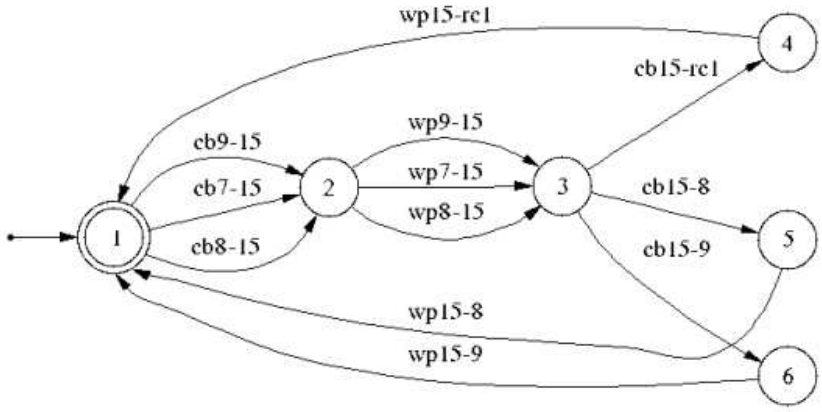


Figure 3.17: Projected level-1 automaton: $cb15^{(1)}$

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.

3.2.2 Rail Transport System rts1

The detailed low-level model of a rail transport system has been presented in [3]. Different from [3], 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 below.

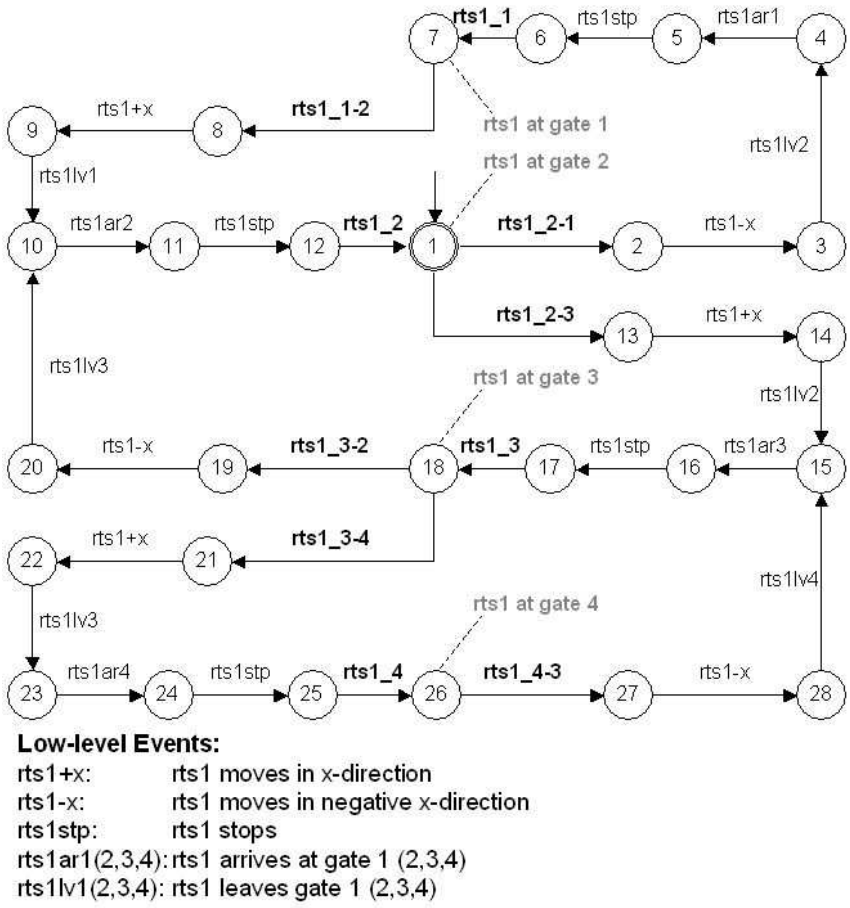


Figure 3.18: Refined low-level model of rts1: $rts1^{(0)}_{ref}$

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

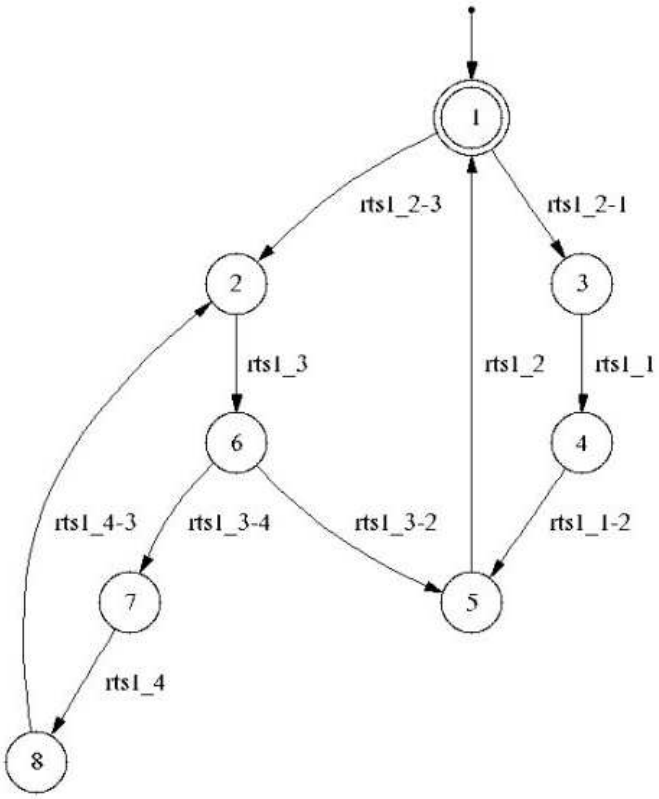


Figure 3.19: Projected level-1 automaton $rts1^{(1)}$

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

3.2.3 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 3.2.1 and 3.2.3. 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 3.7. 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 3.20.

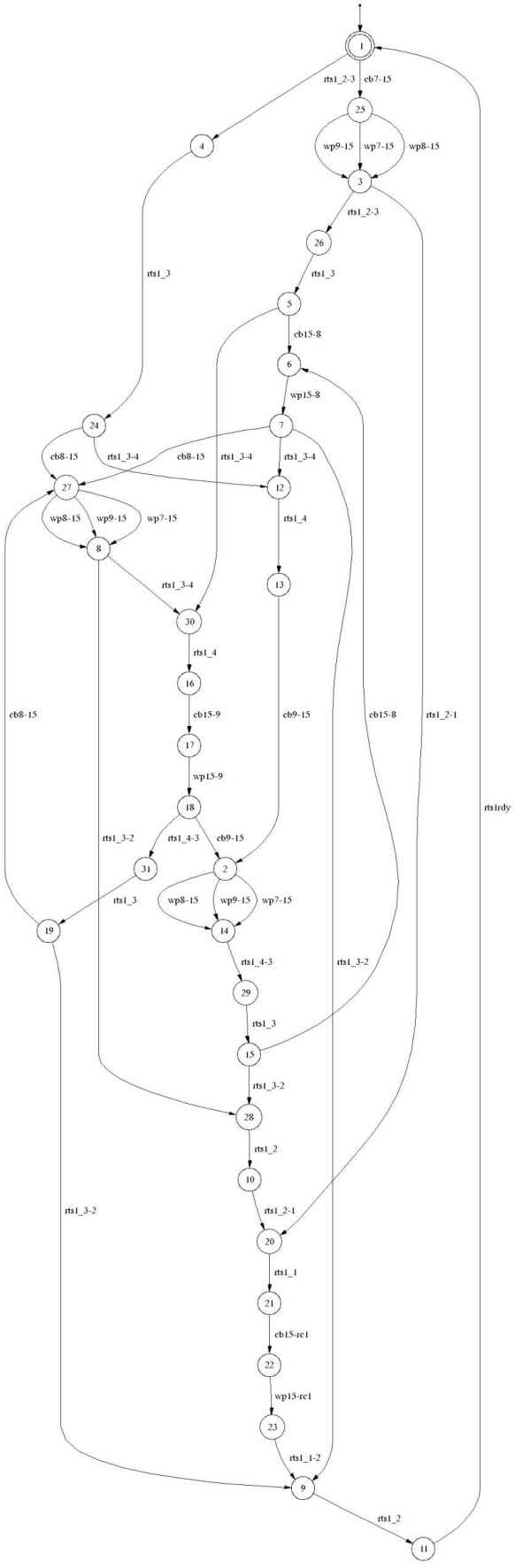


Figure 3.20: Refined automaton $rts1cb15_{ref}^{(1)}$

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 3.20. 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 2.9 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 2.4.

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

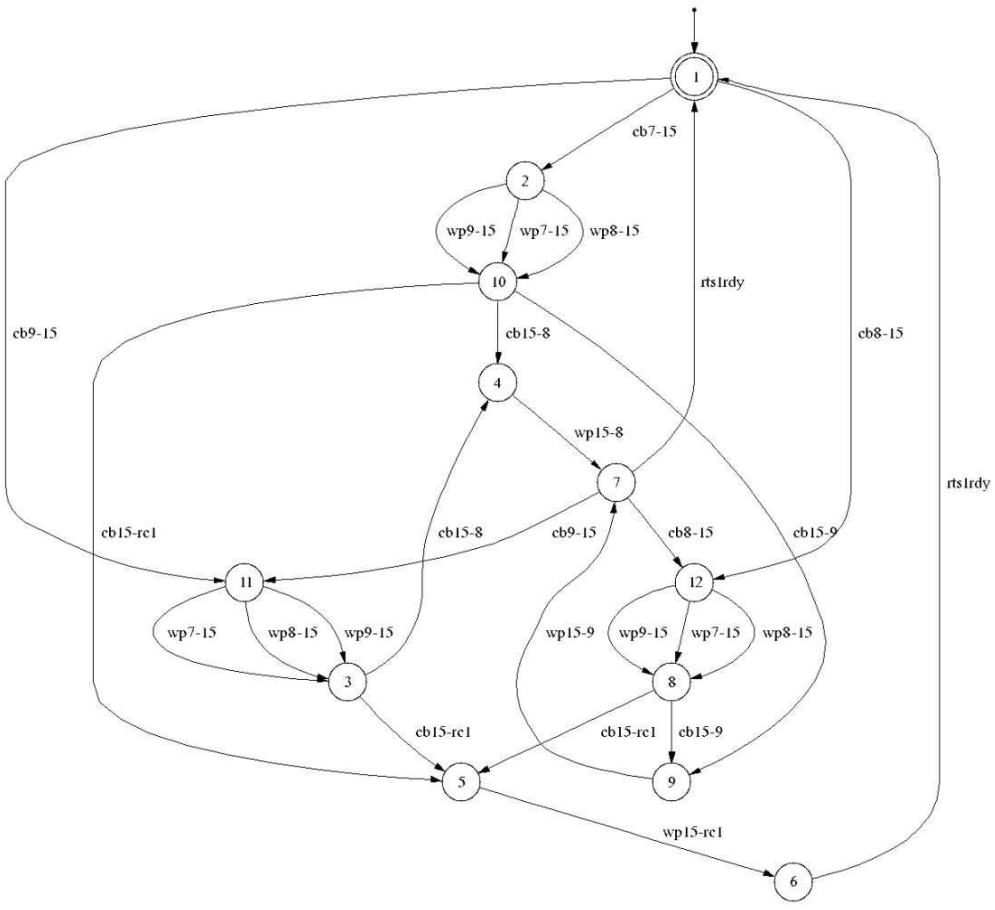


Figure 3.21: $rts1cb15^{(2)}$

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 $L(S^{(2)}_{rts1cb15}/rts1cb15^{(2)})$.

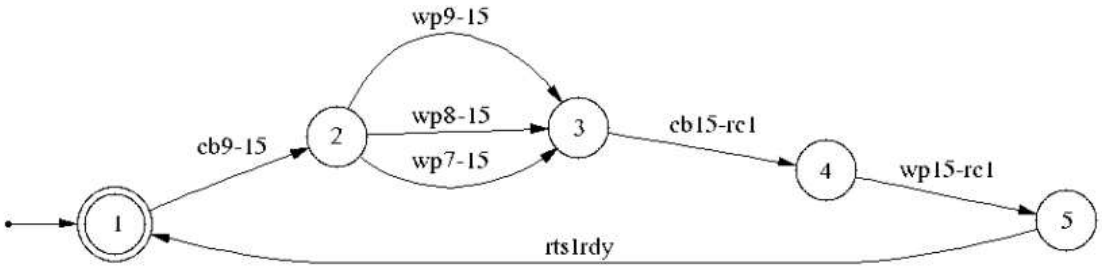


Figure 3.22: Controlled behavior of $rts1cb15^{(2)}$

According to the modified low-level supervisor implementation of Definition 2.4, 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 $rts1cb15^{(1),c}$.

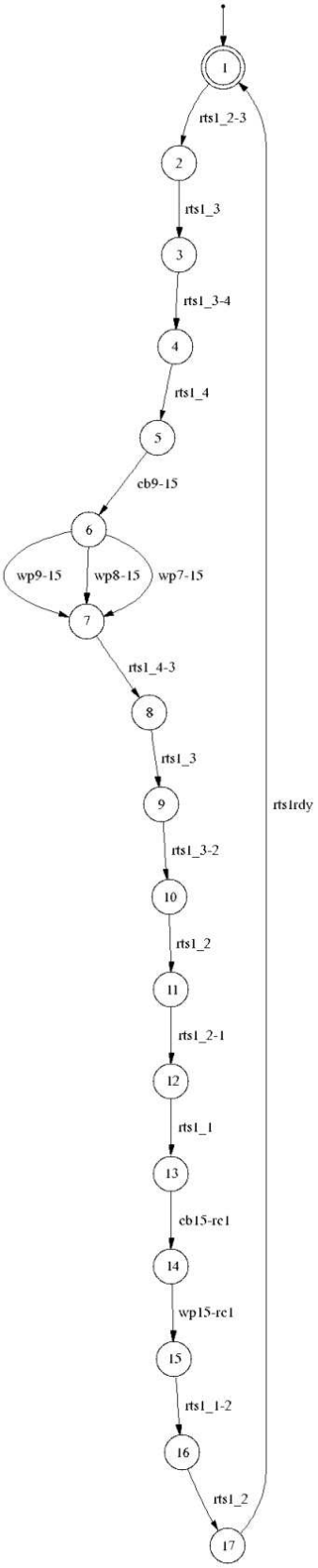


Figure 3.23: Controlled behavior of $rts1cb15_{ref}^{(1)}$ resulting from the modified low-level supervisor implementaion

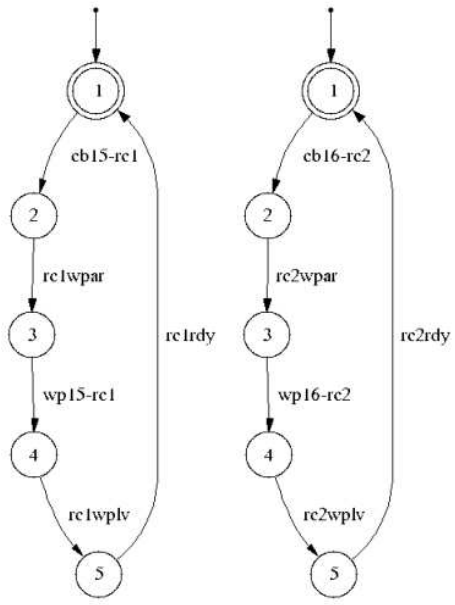
One can see, that the level-2 control action is implemented in nonblocking detailed level-1 control actions.

3.2.4 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 3.24: 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).
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 [8] and modified in Section 2 has been applied the whole Fischertechnik production plant.
It turns out that all subplants are of manageable size on any level of abstraction. This indicates that our method can be used for synthesizing supervisory controllers for large-scale systems.

3.3 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 calculate controlled behavior $S_i^{(j,j-1)}/G_i^{(j-1)}$ resulting of 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.

3.4 State Comparison

The approch presented in this report shows the same computational benefits as the approach presented in [8]. 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.

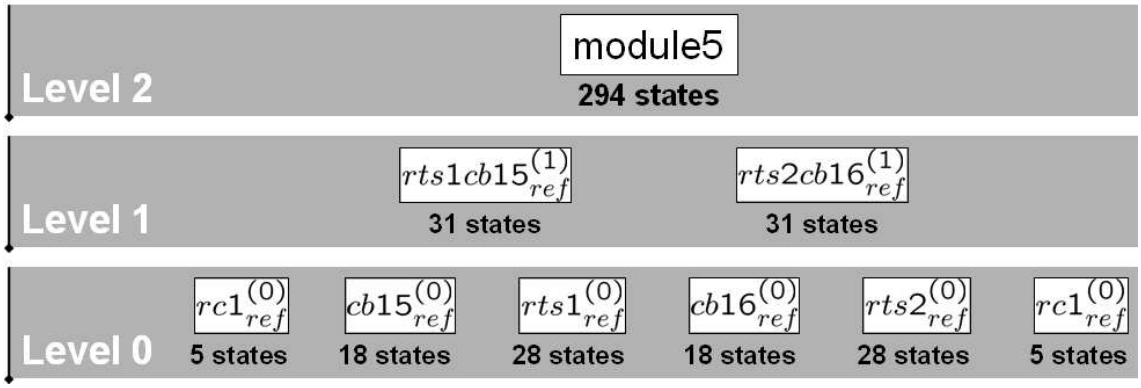


Figure 3.25: Number of states of each automaton of module 5 implemented on PLC

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 a number of 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 3.25.

4 Conclusion

In this report, the approach of [8] has been extended to a multi-level hierarchy, and it has been modified such that it can be applied to a class of discrete event systems that do not necessarily fulfill the property of local nonblocking. Also 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 [8], the maximal permissiveness of the control design in [8] 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 sequencial 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.

References

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

[4] LEDUC, R.J. *Hierarchical Interface-based Supervisory Control*. PhD-Thesis, Department of Electrical & Computer Engineering, University of Toronto. 2002

[5] LEE, S-H. ; WONG, K.C. *Structural Decentralised Control of Concurrent Discrete-Event Systems*. EJC. 2002

[6] PERK, S. *Hierarchical Design of Discrete Event Controllers: An Automated Manufacturing System Case Study*. Diploma Thesis, Lehrstuhl für Regelungstechnik, Universität Erlangen-Nürnberg. 2004

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

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

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

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