CAS 745
Supervisory Control of Discrete-Event Systems

Slides 5: Hierarchical Interface-based Supervisory Control

Dr. Ryan Leduc

Department of Computing and Software
McMaster University

September 1, 2022

Reference Material

1. R.J. Leduc. *Hierarchical Interface-based Supervisory Control.* Doctoral Thesis, Dept. of Elec. & Comp. Engrg., Univ. of Toronto, 2002

2. Ryan J. Leduc, Bertil A. Brandin, Mark Lawford, and W.M. Wonham, "Hierarchical Interface-Based Supervisory Control-Part I: Serial Case," *IEEE Trans. on Automatic Control,* vol 50, no 9, pp. 1322-1335, Sep. 2005.

3. Ryan J. Leduc, Mark Lawford, and W.M. Wonham, "Hierarchical Interface-Based Supervisory Control-Part II: Parallel Case," *IEEE Trans. on Automatic Control,* vol 50, no 9, pp. 1336-1348, Sep. 2005.

4. R.J. Leduc, M. Lawford, and P. Dai, "Hierarchical Interface-Based Supervisory Control of Flexible Manufacturing System," *IEEE Trans. on Control Systems Technology* vol. 14, no. 4, pp. 654-668, Jul. 2006.

Problem Definition

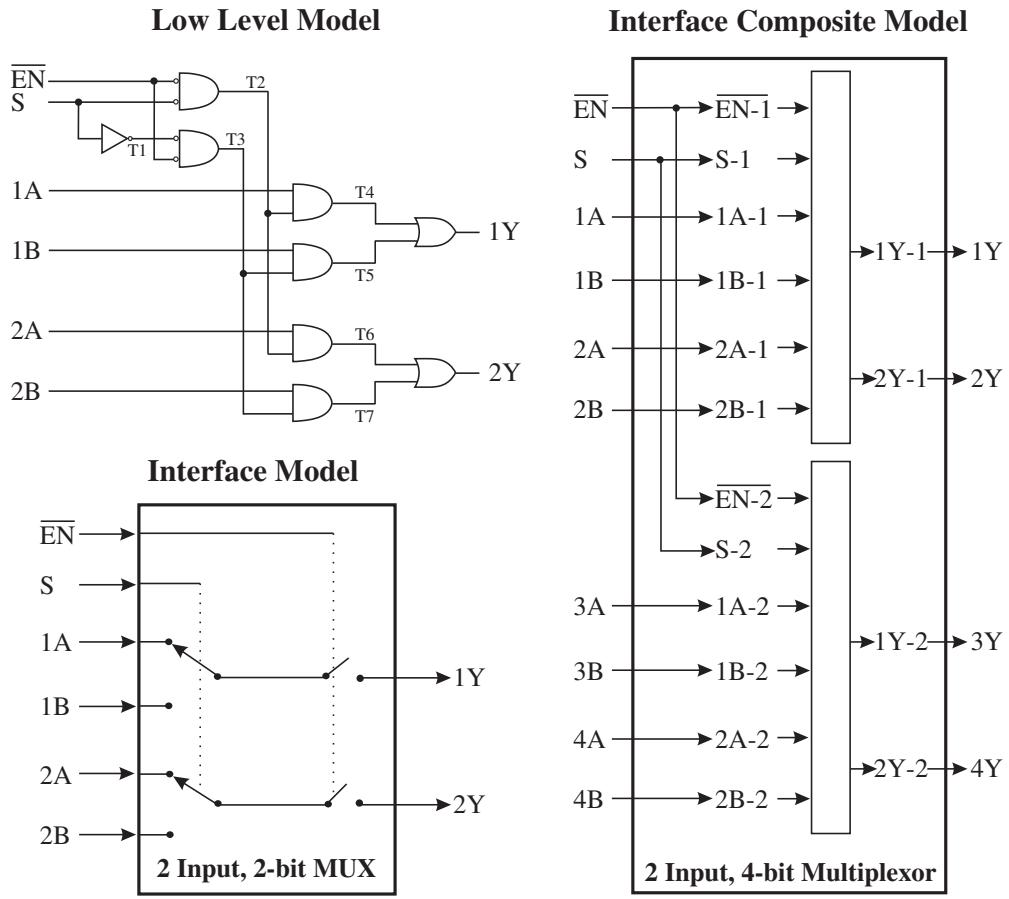
- We wish to be able to verify systems with large state spaces (ie with a statespace on the order of 10^{20} or larger).
- Main obstacle is the combinatorial explosion of the product state space.
- For example, if we took the synchronous product of m 10 state DES, the result could have as many as 10^m states.
- If we take $m = 6$, we are looking at around 1 million states.
- Could take around 1GB of RAM to compute using algorithms in which the states and transitions of the DES are explicitly represented.
- Our goal is to create an architecture for DES that supports a scalable method for the design and verification of nonblocking supervisory controllers.

Inspiration from Digital Logic

- Complexity routinely managed by designers of microprocessors for personal computers.
- Circuits hierarchical in nature, and are designed by using interfaces to limit the interaction between different levels of the hierarchy.
- Complex systems designed by first creating basic components, then adding an interface.
- Interface encapsulates the behaviour of the component and provides an abstract model of the component's operation with a well defined method of interacting with the component.
- These components are then combined to create a new, more complex, component, with its own interface.

Inspiration from Digital Logic - II

- Component treated as a black box; designer only uses the component's interface at the next higher level.
- Designer not allowed to modify the inner workings of a component or to "look below" the interface.



Inspiration from Software Engineering

- In software program design similar ideas are at work.
- To deal with the complexity of large scale systems, software engineers have long advocated the decomposition of software into modules (components) that interact via well defined interfaces (see work of David L. Parnas).
- This approach is referred to as *information hiding*.
- Some advantages are:
  - Limits complexity by hiding unnecessary detail behind interfaces.
  - Promotes independent development; once the module interfaces are defined, each module can be designed separately.

Inspiration from Software Engineering - II

- Advantages cont.
  - By encapsulating the behaviour of a module, its easy to change how module implemented.

    Such changes do not affect the modules that use it since they are not permitted to see internal details or interact with the internals of the module.

  - Makes a module easier to understand as information is localized in modules and unnecessary details are hidden by the interface.
  - Provides a well defined hierarchical structure.

    Structure guarantees we can remove upper levels of hierarchy, and what is left can be reused in another application.

Inspiration from Software Engineering - III

- Our goal is to develop a similar architectural approach for DES.
- We will present a method called *hierarchical interface-based supervisory control (HISC).*
- This method utilizes well defined interfaces between components that are themselves DES.
- These "interface DES" provide a structure allowing local checks to guarantee global properties such as controllability and nonblocking.

DES Preliminaries

- For language L, the eligibility operator $\text{Elig}_L : \Sigma^* \to \text{Pwr}(\Sigma)$ is defined as below. Let $s \in \Sigma^*$; then

$$\text{Elig}_L(s) := \{\sigma \in \Sigma \,|\, s\sigma \in L\}$$

Synchronous Product Versus Meet

- The supervisors are represented as automata and defined as below:

$$\mathbf{S} = (X, \Sigma_S, \xi, x_o, X_m)$$

- Normally we use the *synchronous product* to define the composite plant model, and the *meet* to define the composite supervisor model, as well as the closed-loop behaviour of the system.

- For HISC, will use the *synchronous product* operator for everything.

- **Reason:** makes data entry easier and less error prone, particularly when using a graphical editor to specify and display the supervisor.

Synchronous Product Versus Meet - II

- We define closed loop behaviour of a plant $\mathbf{G}_1 = (Y_1, \Sigma_1, \delta_1, y_{o_1}, Y_{m_1})$ under the control of a supervisor **S** as: **System** $:= \mathbf{G}_1 || \mathbf{S}$.
- To define controllability in this setting, we adopt the standard partition $\Sigma = \Sigma_u \dot{\cup} \Sigma_c$.
- We then define Σ as well as the natural projections and languages below:

$$\Sigma := \Sigma_1 \cup \Sigma_S \qquad P_1 : \Sigma^* \to \Sigma_1^* \quad P_S : \Sigma^* \to \Sigma_S^*$$
$$\mathcal{L}_{G_1} := P_1^{-1} L(\mathbf{G}_1) \qquad \mathcal{L}_S := P_S^{-1} L(\mathbf{S})$$

- Thus, $L(\textbf{System}) = \mathcal{L}_{G_1} \cap \mathcal{L}_S$
- A supervisor $\mathbf{S}$ is *controllable* for a plant $\mathbf{G}_1$ if:

$$\mathcal{L}_S \Sigma_u \cap \mathcal{L}_{G_1} \subseteq \mathcal{L}_S$$

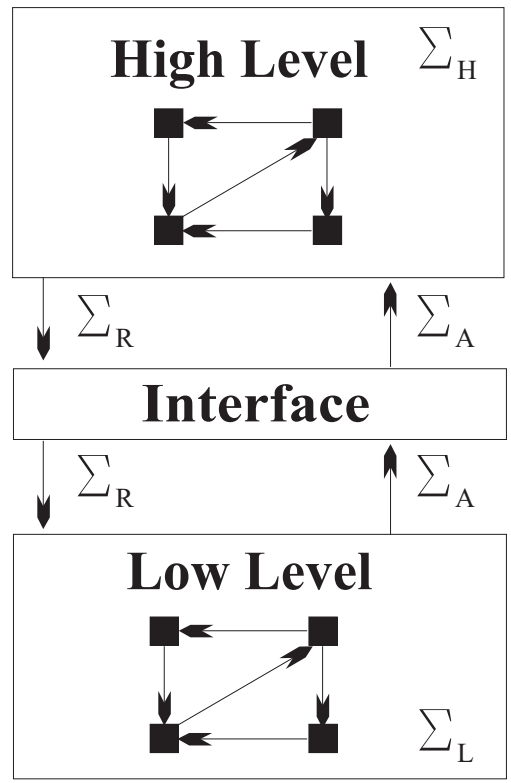
or, equivalently:

$$(\forall s \in \mathcal{L}_{G_1} \cap \mathcal{L}_S)\ \text{Elig}_{\mathcal{L}_{G_1}}(s) \ \cap \ \Sigma_u \subseteq \text{Elig}_{\mathcal{L}_S}(s)$$

Serial Case

- In the serial case of HISC, we have a master-slave system.
- We have a high-level subsystem which sends commands to a low-level subsystem.
- The low-level subsystem performs the indicated task, and then sends back a reply.

- Conceptual structure of system shown below:



Serial Case - II

- Want to be able to design and verify system on a per component basis.
- To achieve this, we interpose an interface between the two systems.
- Interface limits the two subsystem's interaction and knowledge of each other, and is itself a supervisor.
- To capture restriction of information imposed by interface, we partition Σ into the following pairwise disjoint alphabets:

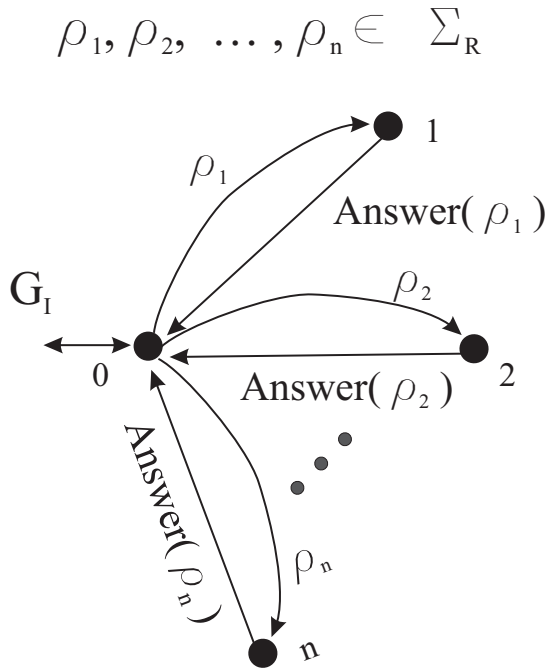
  Request events Σ_R: high-level commands.
  Answer events Σ_A: low-level responses.
  High-level events Σ_H: events visible only at the high-level.
  Low-level events Σ_L: events visible only at the low-level.
- We refer to request and answer events collectively as interface events.

Original Star Interfaces

- Sufficient to define a set of answer events for each request event.
- In essence, define a map **Answer** : $\Sigma_R \to \text{Pwr}(\Sigma_A)$.
- For $\rho \in \Sigma_R$, **Answer**(ρ) is the set of possible answers (referred to as the *answer set*) the low-level subsystem could provide after receiving request ρ.
- We add the additional constraints that the event set of the interface is equal to the set of interface events, that the low-level subsystem must provide at least one response for each request it receives, and that Σ_A does not contain any unused events.

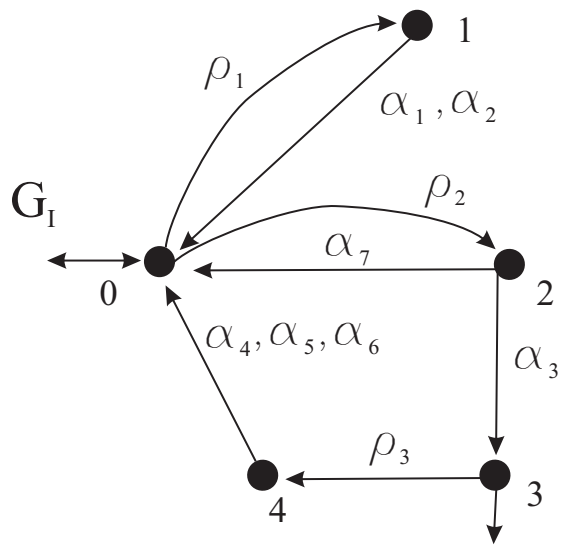
Original Star Interfaces - II

- We can now automatically construct the corresponding star interface in the form shown in the figure below.



Command-pair Interfaces

- Does not require "star" shape.
- Still has request events followed by answer events.
- Can now show additional state information as in figure below.



Command-pair Interfaces Definition

Definition

A DES $G_I = (X, \Sigma_R \dot{\cup} \Sigma_A, \xi, x_o, X_m)$ is a *command-pair interface* if the following conditions are satisfied:

1. $x_o \in X_m$
2. $(\forall x \in X_m)(\forall \sigma \in \Sigma_R \,\dot{\cup}\, \Sigma_A)\, \xi(x, \sigma)! \Rightarrow \sigma \in \Sigma_R \wedge \xi(x, \sigma) \in X - X_m$
3. $(\forall x \in X - X_m)(\forall \sigma \in \Sigma_R \,\dot{\cup}\, \Sigma_A)\, \xi(x, \sigma)! \Rightarrow \sigma \in \Sigma_A \wedge \xi(x, \sigma) \in X_m$

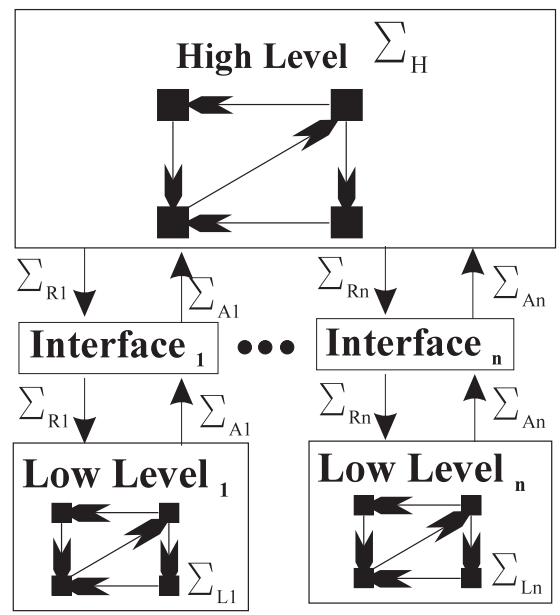
- Clear that star interfaces are a special case of command-pair interfaces.

HISC Parallel Case

- So far, we have restricted ourselves to systems where the number of low-level subsystems, or *degree* of the system (n), is restricted to one.

- We now examine the parallel case where we have $n \geq 1$ low-level subsystems and the serial case is just the special case of $n = 1$.

- Here, a single high-level subsystem, interacts with n independent low-level subsystems, communicating with each through a separate interface.

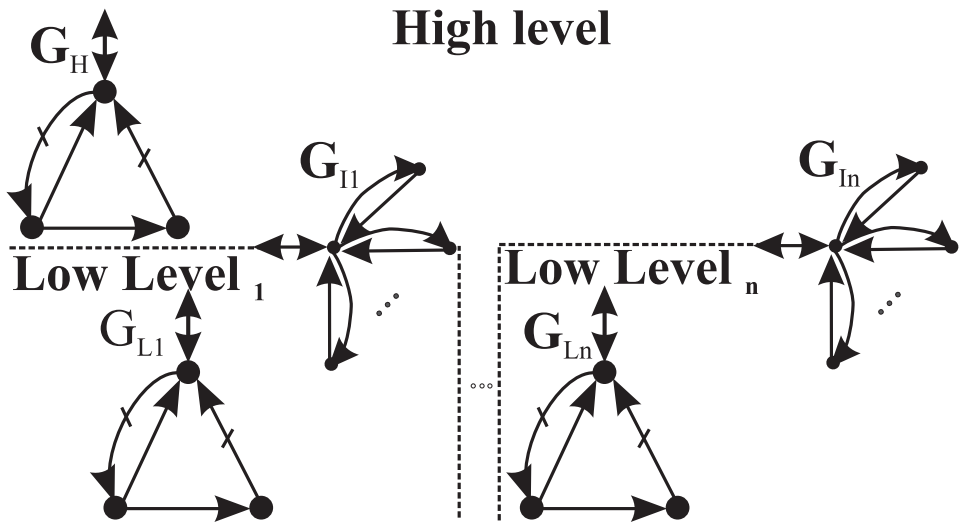


HISC Parallel Case : Nonblocking

- We partition the system alphabet into the following analogous pairwise disjoint alphabets:

$$\Sigma := \Sigma_H \dot{\cup} (\dot{\bigcup_{k=1,\ldots,n}} [\Sigma_{L_k} \dot{\cup} \Sigma_{R_k} \dot{\cup} \Sigma_{A_k}]) \qquad (1)$$

- We take the high-level subsystem to be $\mathbf{G}_H$, and for $j \in \{1, \ldots, n\}$, the j^{th} low-level subsystem is $\mathbf{G}_{L_j}$, and the j^{th} interface is $\mathbf{G}_{I_j}$.



HISC Parallel Case : Nonblocking - II

- The j^{th} low-level is $\mathbf{G}_{L_j}||\mathbf{G}_{I_j}$.
- The high-level is $\mathbf{G} = \mathbf{G}_H||\mathbf{G}_{I_1}||\ldots||\mathbf{G}_{I_n}$.
- We define the term flat system, $\mathbf{G}$, to refer to the equivalent DES that would represent our system if we ignored the interface structure.

$$\mathbf{G} = \mathbf{G}_H||\mathbf{G}_{L_1}||\ldots||\mathbf{G}_{L_n}||\mathbf{G}_{I_1}||\ldots||\mathbf{G}_{I_n}$$

Parallel Case Notation

- For the remainder of this section, we take the index j to have range $\{1, \ldots, n\}$.

- **Event Sets:**

$$\begin{aligned} \Sigma_{I_j} &:= \Sigma_{R_j} \cup \Sigma_{A_j} \\ \Sigma_{IL_j} &:= \Sigma_{L_j} \cup \Sigma_{I_j} \\ \Sigma_{IH} &:= \Sigma_H \cup \bigcup_{k \in \{1,\ldots,n\}} \Sigma_{I_k} \end{aligned}$$

- **Natural Projections:**

$$\begin{aligned} P_{I_j} &: \Sigma^* \to \Sigma_{I_j}^* \\ P_{IL_j} &: \Sigma^* \to \Sigma_{IL_j}^* \\ P_{IH} &: \Sigma^* \to \Sigma_{IH}^* \end{aligned}$$

Parallel Case Notation - II

- **Languages over Σ^*:**

$$\begin{aligned} \mathcal{H} &:= P_{IH}^{-1}(L(\mathbf{G}_H)), & \mathcal{H}_m &:= P_{IH}^{-1}(L_m(\mathbf{G}_H)) \subseteq \Sigma^* \\ \mathcal{L}_j &:= P_{IL_j}^{-1}(L(\mathbf{G}_{L_j})), & \mathcal{L}_{m_j} &:= P_{IL_j}^{-1}(L_m(\mathbf{G}_{L_j})) \subseteq \Sigma^* \\ \mathcal{I}_j &:= P_{I_j}^{-1}(L(\mathbf{G}_{I_j})), & \mathcal{I}_{m_j} &:= P_{I_j}^{-1}(L_m(\mathbf{G}_{I_j})) \subseteq \Sigma^* \end{aligned}$$

- **Language of flat system:**

$$L(G) = \mathcal{H} \cap \mathcal{L}_1 \cap \mathcal{I}_1 \cap \ldots \cap \mathcal{L}_n \cap \mathcal{I}_n$$

- **Marked language of flat system:**

$$L_m(G) = \mathcal{H}_m \cap \mathcal{L}_{m_1} \cap \mathcal{I}_{m_1} \cap \ldots \cap \mathcal{L}_{m_n} \cap \mathcal{I}_{m_n}$$

Level-Wise Nonblocking

Definition

The n^{th} degree ($n \geq 1$) parallel interface system composed of DES $\mathbf{G}_H, \mathbf{G}_{I_1}, \mathbf{G}_{L_1}, \ldots, \mathbf{G}_{I_n}, \mathbf{G}_{L_n}$, is said to be *level-wise nonblocking* if the following conditions are satisfied:

(I) *nonblocking at the high-level*:

$$\overline{\mathcal{H}_m \cap \bigcap_{k=1,\ldots,n} \mathcal{I}_{m_k}} = \mathcal{H} \cap \bigcap_{k=1,\ldots,n} \mathcal{I}_k$$

(II) *nonblocking at the low-level*: for all $j \in \{1, \ldots, n\}$,

$$\overline{\mathcal{L}_{m_j} \cap \mathcal{I}_{m_j}} = \mathcal{L}_j \cap \mathcal{I}_j$$

Interface Consistent

Definition

The n^{th} degree ($n \geq 1$) parallel interface system composed of DES $\mathbf{G}_H, \mathbf{G}_{I_1}, \mathbf{G}_{L_1}, \ldots, \mathbf{G}_{I_n}, \mathbf{G}_{L_n}$, is *interface consistent* with respect to the alphabet partition given by (1), if for all $j \in \{1, \ldots, n\}$, the following conditions are satisfied:

Multi-level Properties

1. The event set of $\mathbf{G}_H$ is Σ_{IH}, and the event set of $\mathbf{G}_{L_j}$ is Σ_{IL_j}.

2. $\mathbf{G}_{I_j}$ is a command-pair interface.

Interface Consistent - II

High-Level Properties

3. $(\forall s \in \mathcal{H} \cap \bigcap_{k=1,\ldots,n} \mathcal{I}_k)$
$\mathsf{Elig}_{\mathcal{I}_j}(s) \cap \Sigma_{A_j} \subseteq \mathsf{Elig}_{\mathcal{H}}(s)$

This property asserts that answer events must be eligible in the high-level subsystem (G_H), if the event is eligible in the interface.

Low-Level Properties

4. $(\forall s \in \mathcal{L}_j \cap \mathcal{I}_j)\ \mathsf{Elig}_{\mathcal{I}_j}(s) \cap \Sigma_{R_j} \subseteq \mathsf{Elig}_{\mathcal{L}_j}(s)$

This property asserts that request events must be eligible in the low-level subsystem (G_{L_j}), if the event is eligible in the interface.

Interface Consistent - III

5. $(\forall s \in \Sigma^*.\Sigma_{R_j} \cap \mathcal{L}_j \cap \mathcal{I}_j)$
$\mathsf{Elig}_{\mathcal{L}_j \cap \mathcal{I}_j}(s\Sigma^*_{L_j}) \cap \Sigma_{A_j} = \mathsf{Elig}_{\mathcal{I}_j}(s) \cap \Sigma_{A_j}$

where $\mathsf{Elig}_{\mathcal{L}_j \cap \mathcal{I}_j}(s\Sigma^*_{L_j}) := \bigcup_{l \in \Sigma^*_{L_j}} \mathsf{Elig}_{\mathcal{L}_j \cap \mathcal{I}_j}(sl)$

This property asserts that immediately after a request event has occurred, there exists a path via strings in $\Sigma^*_{L_j}$ to each answer event that can follow the request event. Some paths may vanish after low-level events occur.

6. $(\forall s \in \mathcal{L}_j \cap \mathcal{I}_j)$
$s \in \mathcal{I}_{m_j} \Rightarrow (\exists l \in \Sigma^*_{L_j})\ sl \in \mathcal{L}_{m_j} \cap \mathcal{I}_{m_j}.$

This property asserts that every string marked by the interface and accepted by the low-level subsystem, can be extended by a low-level string to a string marked by the low-level.

Nonblocking Theorem

Theorem

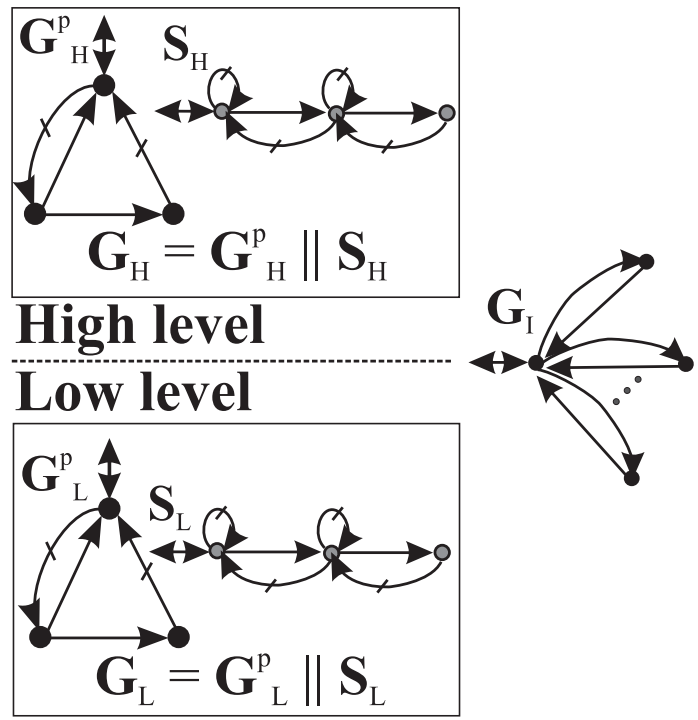
If the n^{th} degree ($n \geq 1$) parallel interface system composed of DES $\mathbf{G}_H, \mathbf{G}_{I_1}, \mathbf{G}_{L_1}, \ldots, \mathbf{G}_{I_n}, \mathbf{G}_{L_n}$, is level-wise nonblocking and interface consistent with respect to the alphabet partition given by (1), then

$$\overline{L_m(\mathbf{G})} = L(\mathbf{G}), \text{ where } \mathbf{G} = \mathbf{G}_H||\mathbf{G}_{I_1}||\mathbf{G}_{L_1}||\ldots||\mathbf{G}_{I_n}||\mathbf{G}_{L_n}$$

- For example, let's assume $\mathbf{G}_H$ has 10^5 states, $\mathbf{G}_{L_1}$ has 10^5 states, and $\mathbf{G}_{I_1}$ has 10 states ($n = 1$).
- Worst case, $\mathbf{G}$ could have 10^{11} states so checking nonblocking directly would likely run out of memory.
- However, $\mathbf{G}_H||\mathbf{G}_{I_1}$ and $\mathbf{G}_{L_1}||\mathbf{G}_{I_1}$ would have maximum 10^6 states which can easily be handled.

Parallel Case: Controllability

- For controllability, we need to decompose the system down into its plant and supervisors.
- The general form of a bi-level system is shown in the figure below, for $(n = 1)$.
- We define *high-level plant* to be $\mathbf{G}^p_H$, and *supervisor* to be $\mathbf{S}_H$.
- Similarly, the j^{th} *low level plant* and *supervisor* are $\mathbf{G}^p_{L_j}$ and $\mathbf{S}_{L_j}$.



Parallel Case: Controllability - II

- For consistency, we define the following identities for the high and j^{th} low-level subsystems as below:

$$\mathbf{G}_H := \mathbf{G}^p_H || \mathbf{S}_H \qquad \mathbf{G}_{L_j} := \mathbf{G}^p_{L_j} || \mathbf{S}_{L_j}$$

- We can now define our *flat supervisor* and *plant* as well as some useful languages over Σ^*:

$$\begin{aligned}
\textbf{Plant} &:= \mathbf{G}^p_H || \mathbf{G}^p_{L_1} || \ldots || \mathbf{G}^p_{L_n} \\
\textbf{Sup} &:= \mathbf{S}_H || \mathbf{S}_{L_1} || \ldots || \mathbf{S}_{L_n} || \mathbf{G}_{I_1} || \ldots || \mathbf{G}_{I_n} \\
\mathcal{H}^p &:= P^{-1}_{IH} L(\mathbf{G}^p_H), \quad \mathcal{S}_H := P^{-1}_{IH} L(\mathbf{S}_H), \subseteq \Sigma^* \\
\mathcal{L}^p_j &:= P^{-1}_{IL_j} L(\mathbf{G}^p_{L_j}), \quad \mathcal{S}_{L_j} := P^{-1}_{IL_j} L(\mathbf{S}_{L_j}), \subseteq \Sigma^*
\end{aligned}$$

Level-Wise Controllable

Definition

The n^{th} degree ($n \geq 1$) parallel interface system composed of DES $\mathbf{G}_H^p$, $\mathbf{G}_{L_1}^p, \ldots, \mathbf{G}_{L_n}^p$, $\mathbf{S}_H$, $\mathbf{S}_{L_1}, \ldots, \mathbf{S}_{L_n}$, $\mathbf{G}_{I_1}, \ldots, \mathbf{G}_{I_n}$, is *level-wise controllable* with respect to the alphabet partition given by (1), if for all $j \in \{1, \ldots, n\}$ the following conditions hold:

(I) The alphabet of $\mathbf{G}_H^p$ and $\mathbf{S}_H$ is Σ_{IH}, the alphabet of $\mathbf{G}_{L_j}^p$ and $\mathbf{S}_{L_j}$ is Σ_{IL_j}, and the alphabet of $\mathbf{G}_{I_j}$ is Σ_{I_j}

(II) $(\forall s \in \mathcal{L}_j^p \cap \mathcal{S}_{L_j} \cap \mathcal{I}_j)$
$$\text{Elig}_{\mathcal{L}_j^p}(s) \cap \Sigma_u \subseteq \text{Elig}_{\mathcal{S}_{L_j} \cap \mathcal{I}_j}(s)$$

(III) $(\forall s \in \mathcal{H}^p \cap \mathcal{I} \cap \mathcal{S}_H)$
$$\text{Elig}_{\mathcal{H}^p \cap \mathcal{I}}(s) \cap \Sigma_u \subseteq \text{Elig}_{\mathcal{S}_H}(s), \text{ where } \mathcal{I} := \bigcap_{k=1,\ldots,n} \mathcal{I}_k$$

Controllability Theorem

Theorem

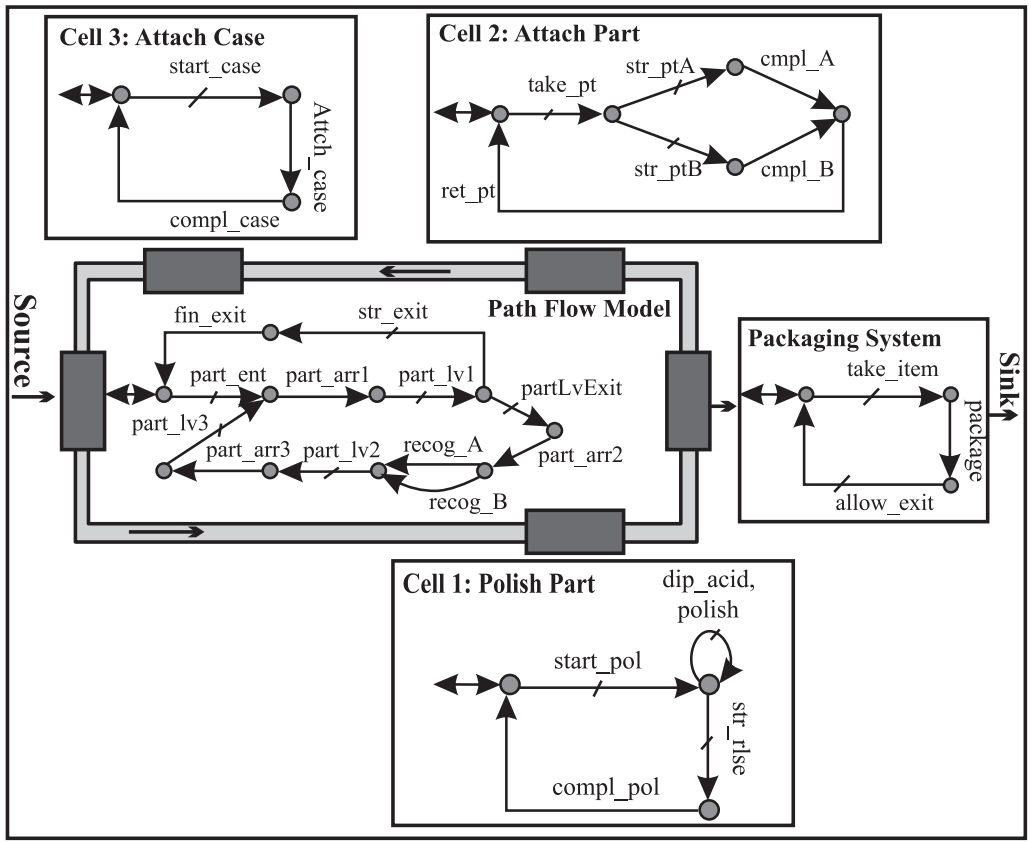
If the n^{th} degree ($n \geq 1$) parallel interface system composed of plant components $\mathbf{G}^p_H, \mathbf{G}^p_{L_1}, \ldots, \mathbf{G}^p_{L_n}$, supervisors $\mathbf{S}_H, \mathbf{S}_{L_1}, \ldots, \mathbf{S}_{L_n}$, and interfaces $\mathbf{G}_{I_1}, \ldots, \mathbf{G}_{I_n}$, is level-wise controllable with respect to the alphabet partition given by (1), then

$(\forall s \in L(\textbf{Plant}) \cap L(\textbf{Sup}))$

$$Elig_{L(\textbf{Plant})}(s) \cap \Sigma_u \subseteq Elig_{L(\textbf{Sup})}(s)$$

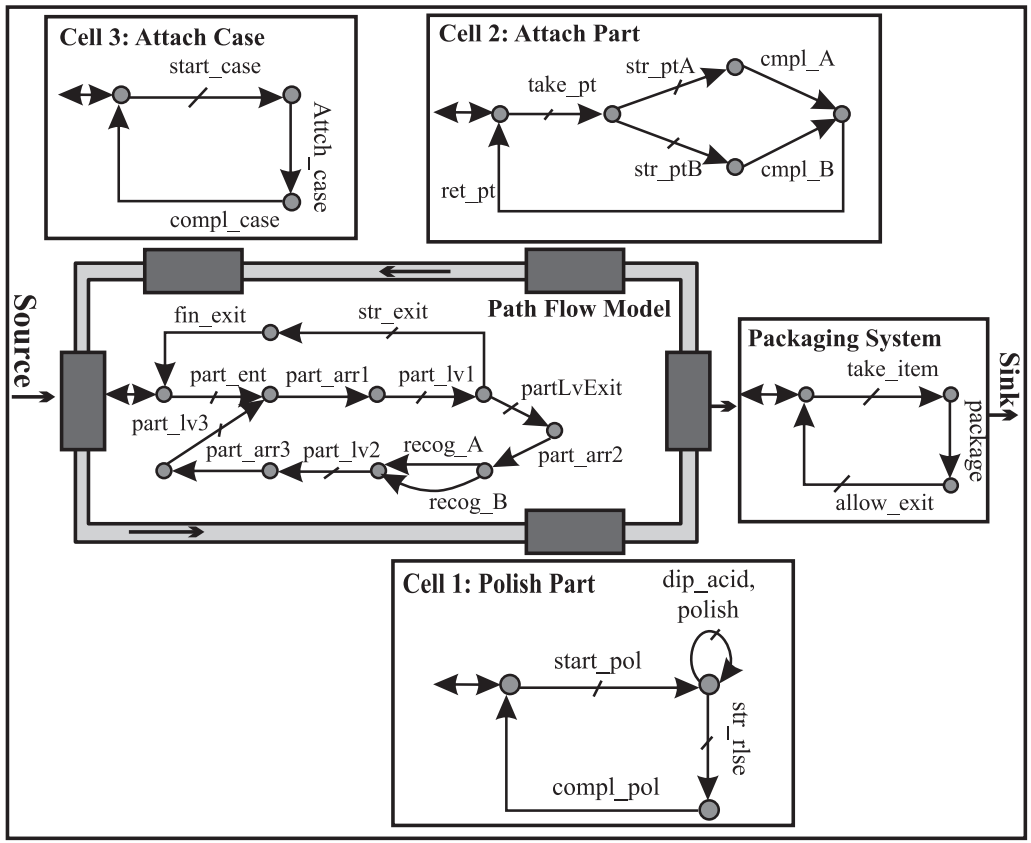
Simple Manufacturing Example

- The manufacturing unit, shown below, is composed of three cells connected by a conveyor belt.
- Before each cell, there is a part acquisition unit that automatically stops a part and holds it until it is given a release command.
- After the item exits the conveyor system, it goes to a packaging machine.
- Parts enter the system at the far left and exit at the far right.



Plant Models

- Diagram shows flat view of plant (supervisors will be added later).
- Shows plant models for cell one (polishes part), cell two (attaches part of type A or type B to the assembly of what's being built), cell three (attach case to assembly).
- In the middle, we see the path flow model that shows how parts enter the system, travel around the belt, and finally leave the system.
- On the far right, we see the model for the packaging system.

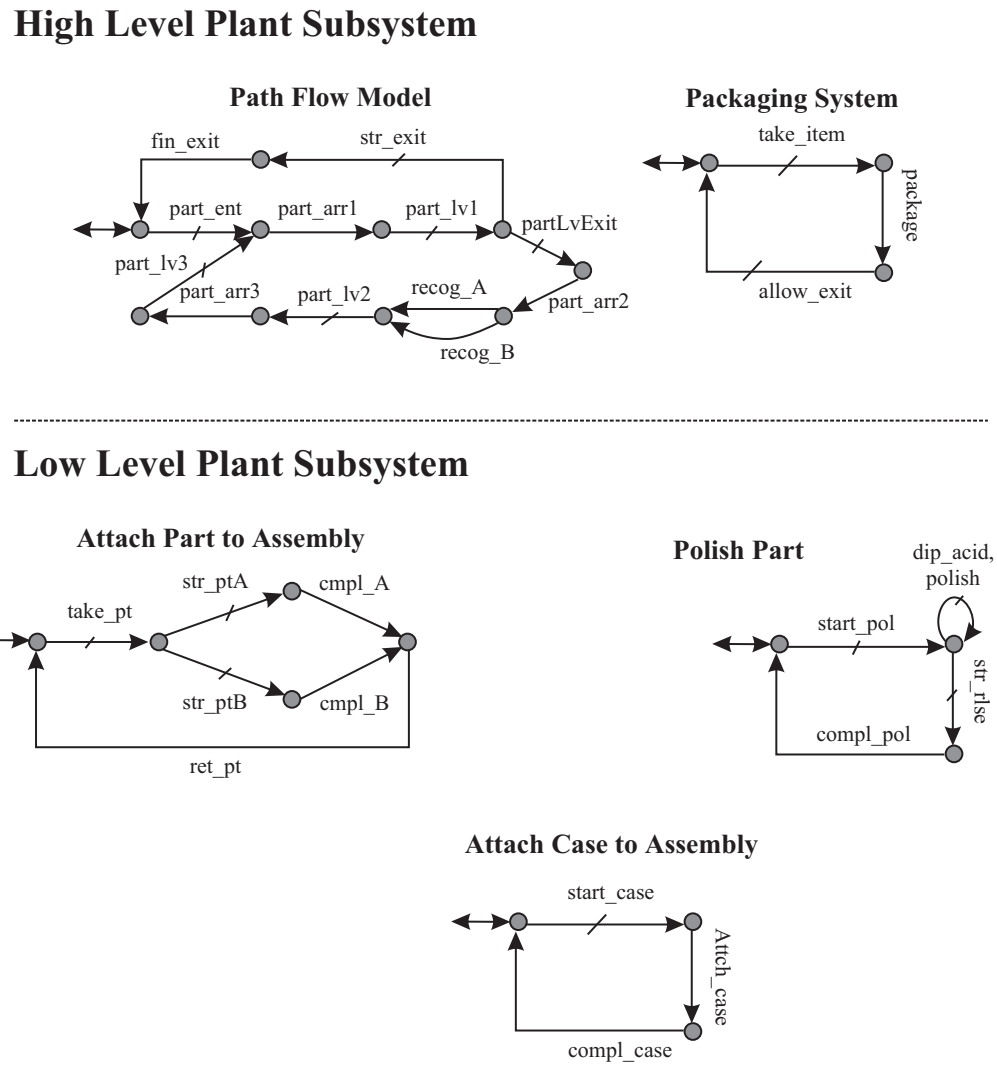


Defining Infrastructure

- First step is to decide which plant models belong at which level.

- High level decisions and info about how components interact goes at high-level.

- In general, localized information about how a component performs a task goes at the low-level.

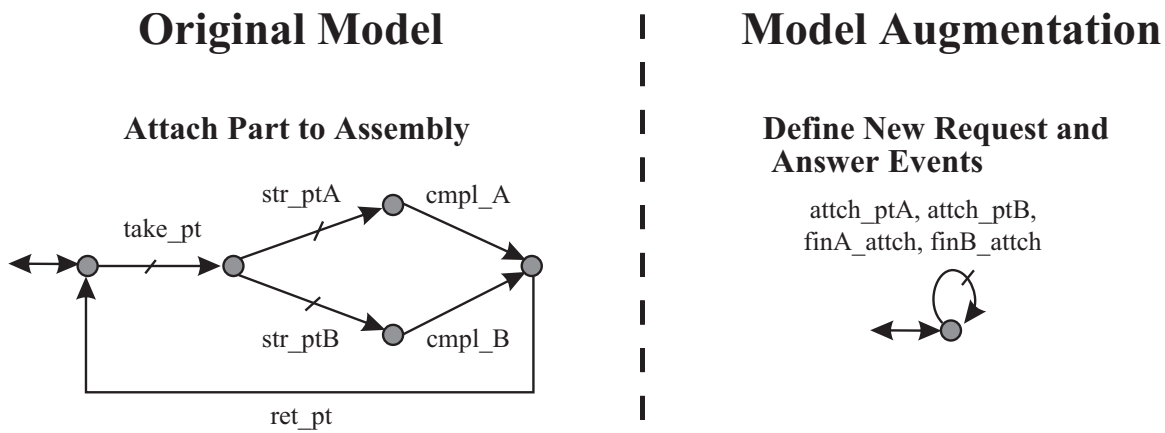


Augmenting Low-Level Plant

- Problem: the model for cell two is not well suited to being accessed through an interface.
- Why? It requires that the decision to attach part A or part B be made after event *take_pt* occurs.
- Means we have to pass in which part to choose.
- To achieve this, we augment the model by adding the DES **Define New Request and Answer Events** shown on next slide.

Augmenting Low-Level Plant - II

- The new request events (*attch_ptA* and *attch_ptB*) allow high -level to specify desired part.
- New finish events (*finA_attch* and *finB_attch*) tell high-level that their respective tasks is complete.
- We refer to these new events as expansion events.

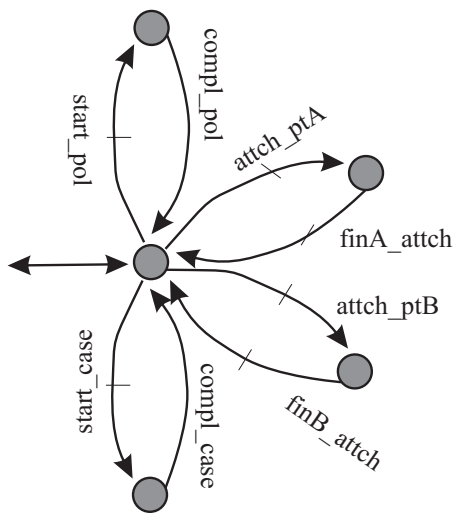


Defining Interface and Event Partition

- We define our interface DES as in figure below.
- We next define the alphabet partition $\Sigma := \Sigma_H \dot{\cup} \Sigma_L \dot{\cup} \Sigma_R \dot{\cup} \Sigma_A$ as follows:

$$
\begin{aligned}
\Sigma_R &= \{\textit{start\_pol}, \textit{attch\_ptA}, \textit{attch\_ptB}, \textit{start\_case}\} \\
\Sigma_A &= \{\textit{comp\_pol}, \textit{finA\_attch}, \textit{finB\_attch}, \textit{compl\_case}\} \\
\Sigma_H &= \{\textit{part\_ent}, \textit{part\_arr1}, \textit{part\_lv1}, \textit{partLvExit}, \\
&\quad \textit{str\_exit}, \textit{fin\_exit}, \textit{part\_arr2}, \textit{recog\_A}, \textit{recog\_B}, \\
&\quad \textit{part\_lv2}, \textit{part\_arr3}, \textit{part\_lv3}, \textit{take\_item}, \\
&\quad \textit{allow\_exit}, \textit{package}\} \\
\Sigma_L &= \{\textit{take\_pt}, \textit{str\_ptA}, \textit{str\_ptB}, \textit{compl\_A}, \textit{compl\_B}, \\
&\quad \textit{ret\_pt}, \textit{dip\_acid}, \textit{polish}, \textit{str\_rlse}, \textit{attch\_case}\}
\end{aligned}
$$

$\mathbf{G_I}$



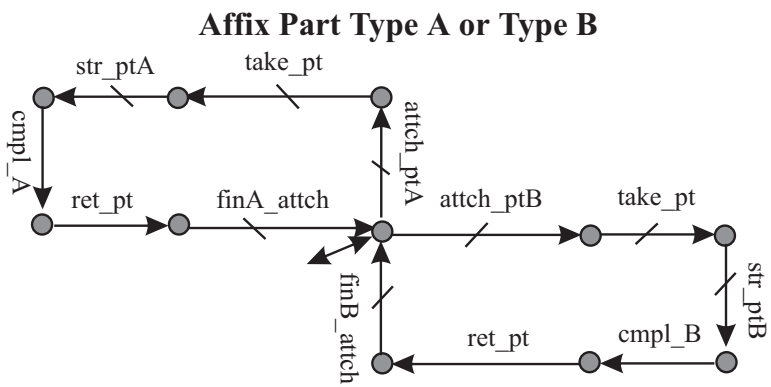
Control specifications

1. For cell one, we want the sequence *dip_acid-polish* to be repeated twice, after a *start_pol* event occurs.
2. For cell two, we should perform correct sequence to attach the appropriate part based on events *attch_ptA* and *attch_ptB* occurring. We should then allow the appropriate answer event to occur.
3. Only one part allowed in loop at a time.
4. Processing order is cell 1, cell 2, cell 3, and then the part is allowed to leave the loop to go to the packaging machine.
5. There is a two slot buffer between the loop, and the packaging machine. It should not underflow or overflow.

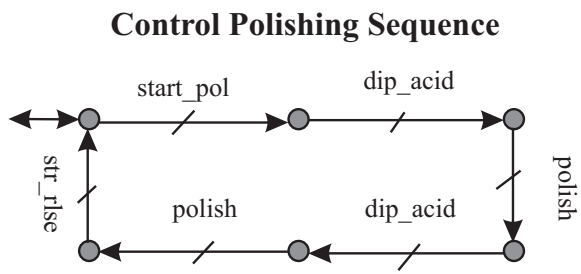
Designing Low-Level Supervisors

- After defining the interface, we next design the low-level supervisors. They will provide the functionality for the request events, and give meaning to the answer events.
- The idea is for the low-level to offer well-defined "services" to the high-level.
- The low-level may also contain processes that are necessary for its function but are not directly related to the interface.
- The supervisors to satisfy points 1 and 2 are shown below:

For Attach Part

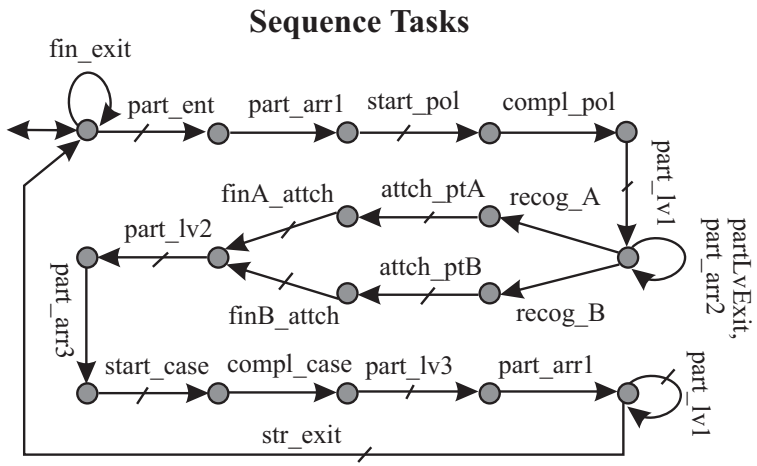


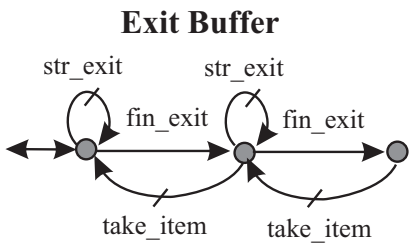
For Polish Part



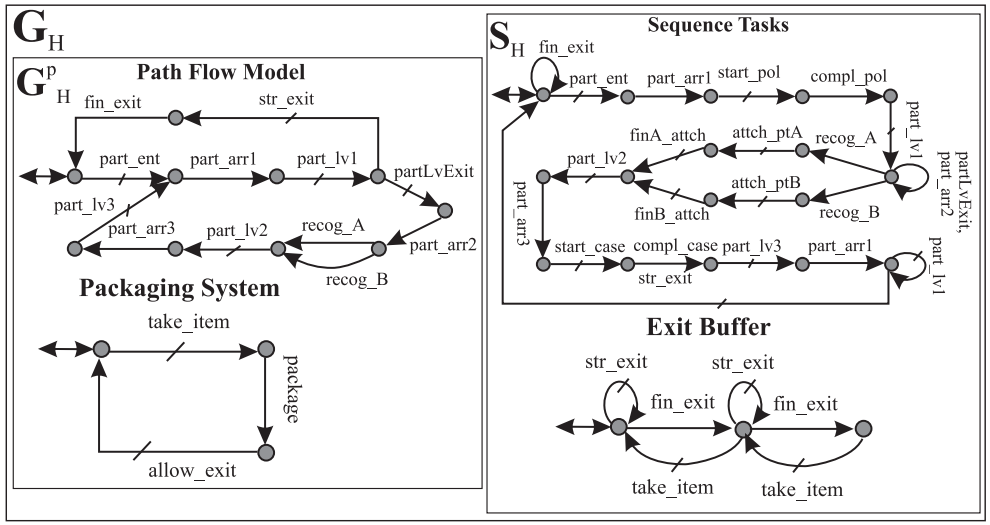
Designing High-Level Supervisors

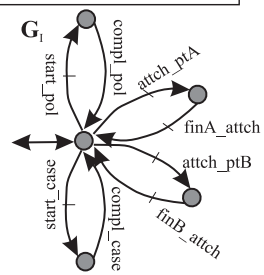
- High level supervisors are designed to manipulate the interface to get the low level perform the desired task.
- They may only express their control law in terms of high level and interface events.
- The supervisors to satisfy the remaining control specifications are shown below:





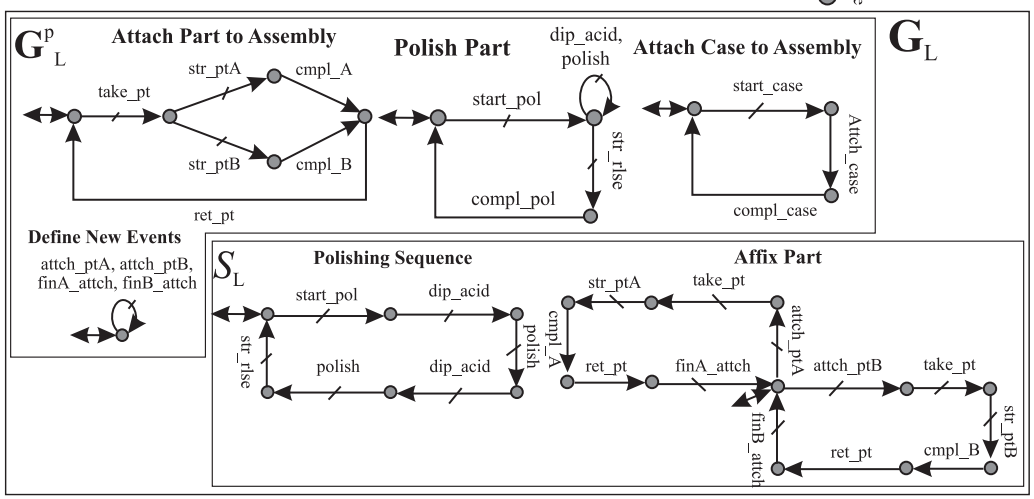
Final System





High Level Subsystem

Low Level Subsystem

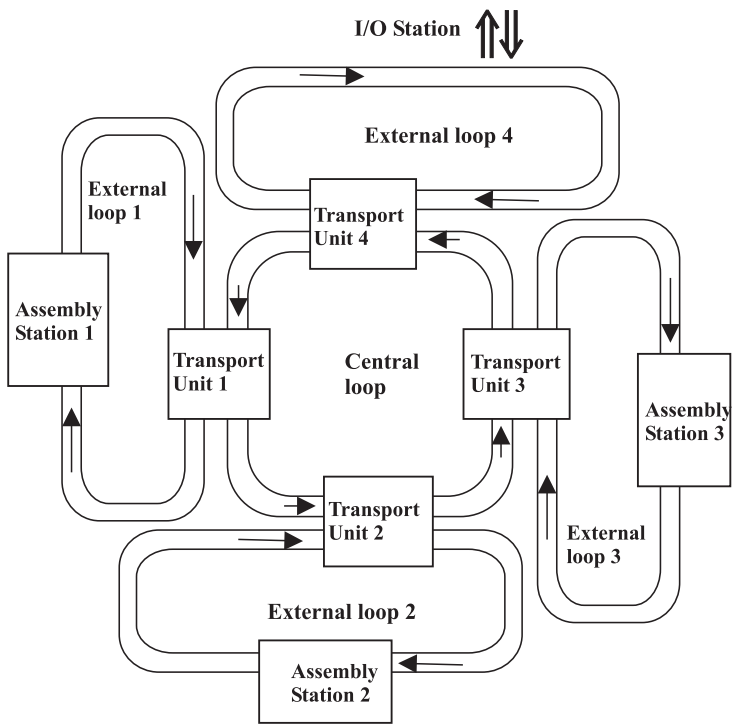


Final System: Results

- Can use a software tool (DESpot) to verify that the system is level-wise nonblocking and interface consistent.
- We can thus conclude via **Nonblocking Theorem** that the flat system is nonblocking.
- We then verify that the system is level-wise controllable, and thus conclude by **Controllability Theorem** that the flat supervisor is controllable for the flat plant.

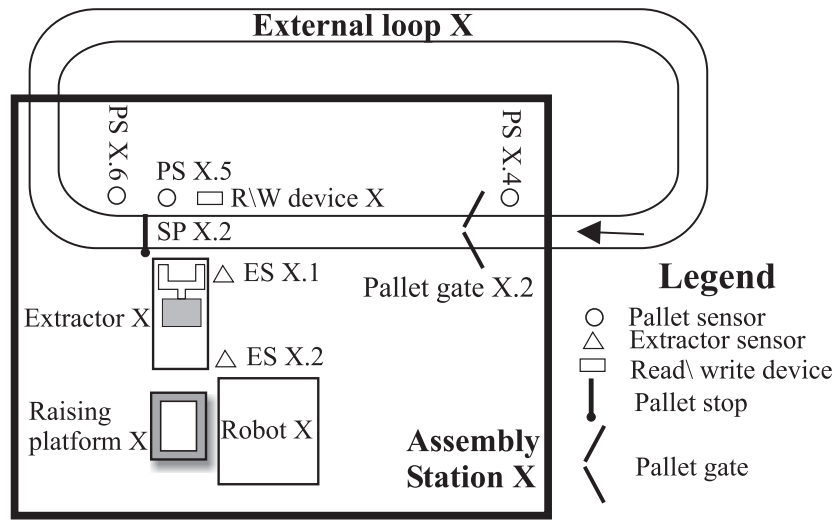
AIP Example

- Applied HISC method to the automated manufacturing system of the Atelier Inter-établissement de Productique (AIP).
- System consists of:
  - Central loop (CL) and four external loops (EL).
  - Three assembly stations (AS).
  - Four inter-loop transfer units (TU).
  - One I/O station where pallets enter and leave the station.



Assembly Station for External Loop $X = 1, 2, 3$

- Each station contains:
  - Robot to perform assembly tasks.
  - Extractor to transfer the pallet from the conveyor loop to the robot.
  - Sensors to determine the location of the extractor.
- Raising platform to present the pallet to the robot.
- Sensors to detect presence of pallet.
- Read/write (R/W) device to access the pallet's electronic label.



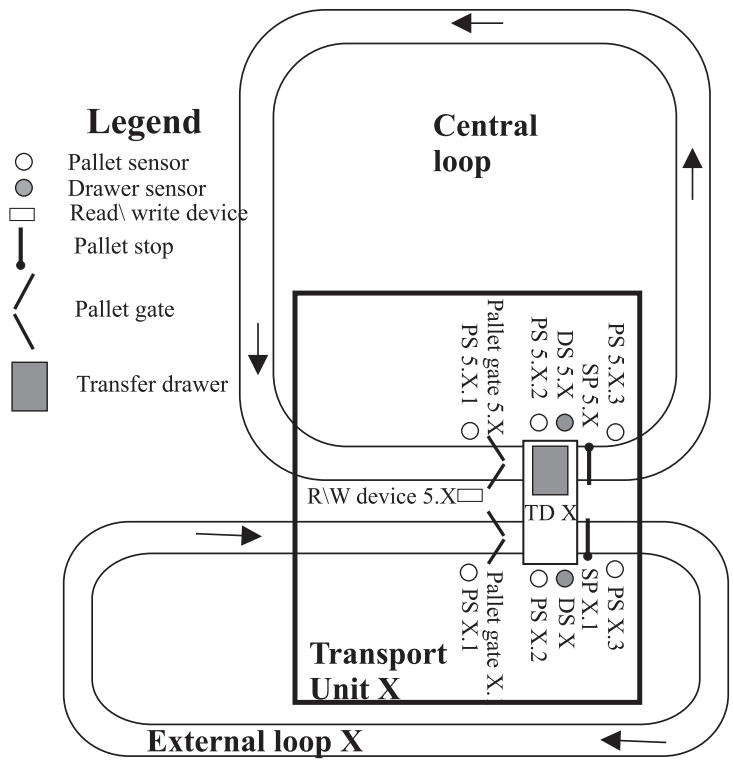
Assembly Stations - II

- Stations differ based on functionality and reliableness.
- Station one can perform task1A and task1B, while station 2 can perform task task2A and task2B.
- Station 3 can perform all four tasks, and can also repair a damaged pallet.
- Stations 1 and 2 can break down and must be repaired, while station 3 is of higher quality and is assumed never to break down.
- Station 3 is used as a substitute for the other stations when they are down.

Transport Unit for External Loop $X = 1, 2, 3, 4$

- Transport Units are used to transfer pallets between the central loop and the external loops.
- Each transport unit contains:
  - Transport drawer which physically conveys the pallet between the two loops.

- Sensors to determine the drawer's location.
- Each loop has a pallet gate and pallet stop.
- Sensors to detect presence of pallets.
- Read/write device.

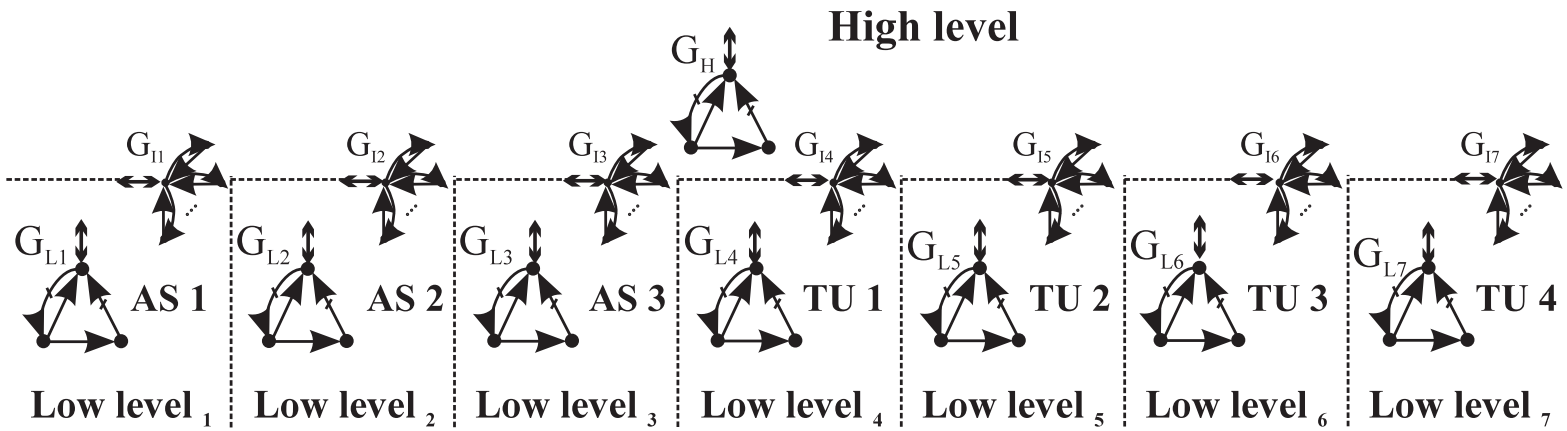


AIP Control Specifications

1. **Routing:** Type 1 pallet must go first to AS1, then AS2 before leaving the system. Type 2 pallets go first to AS2, then AS1 before leaving the system.
2. **Capacity:** External loops 1 and 2 are limited to one pallet at a time..
3. **Pallet Exit Ordering:** type 1, type 2, type 1, ...
4. **Assembly errors:** When pallet damaged, route to AS3 for maintenance then undergo assembly operation again.
5. **Station Breakdown:** When AS1 or AS2 down, pallets are routed to AS3 until station repaired.
6. **Capacity of AS:** Only one pallet is allowed in a given station at a time.
7. **Task Ordering:** For Type 1 pallets, do task1A before task1B, and task2A before task2B. For type 2, do task1B before task1A, and task2B before task2A.

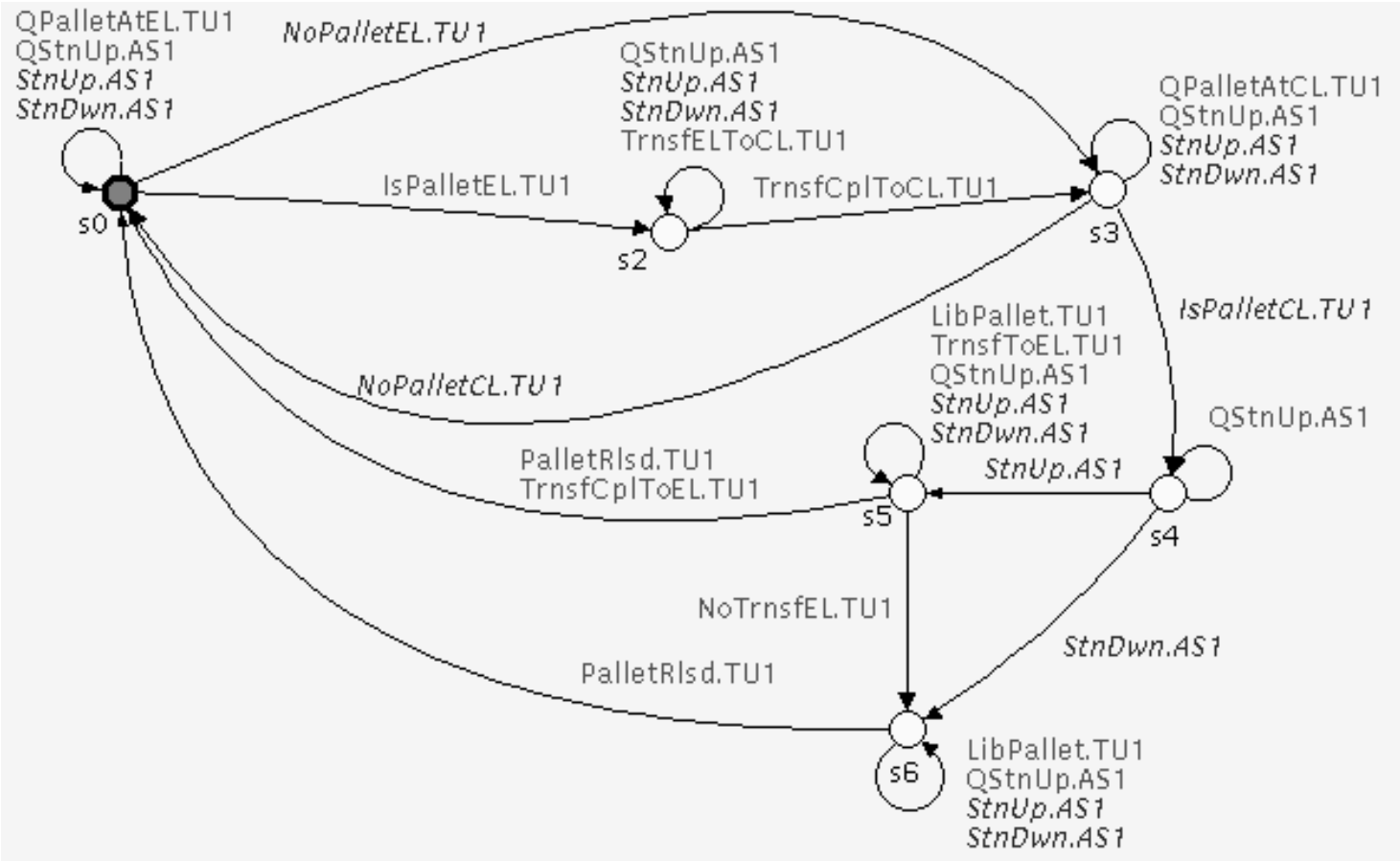
Structure of Parallel System

- Figure below shows breakdown of AIP system.
- Example contains 181 DES, so we will only look at a few representative DES.
- **Convention:**
  - Uncontrollable events are shown in italics; all other events are controllable.
  - Initial states have a thick outline, marker states are filled.



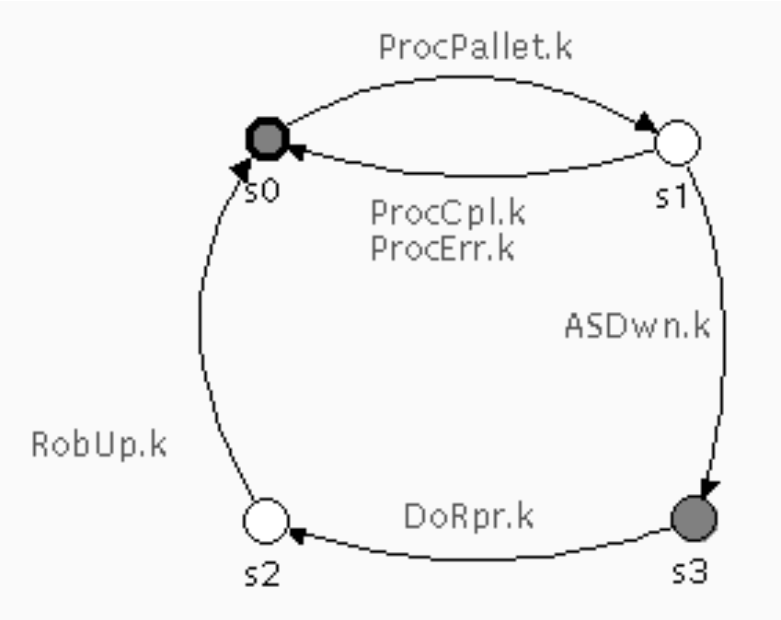
High-Level

- Keeps track of breakdown status of assembly stations.
- Enforces maximum capacity of external loops 1 and 2.
- Controls operation of all transport units and assembly stations.
- **Supervisor ManageTU1:** controls transfer of pallets between central loop and external loop 1.

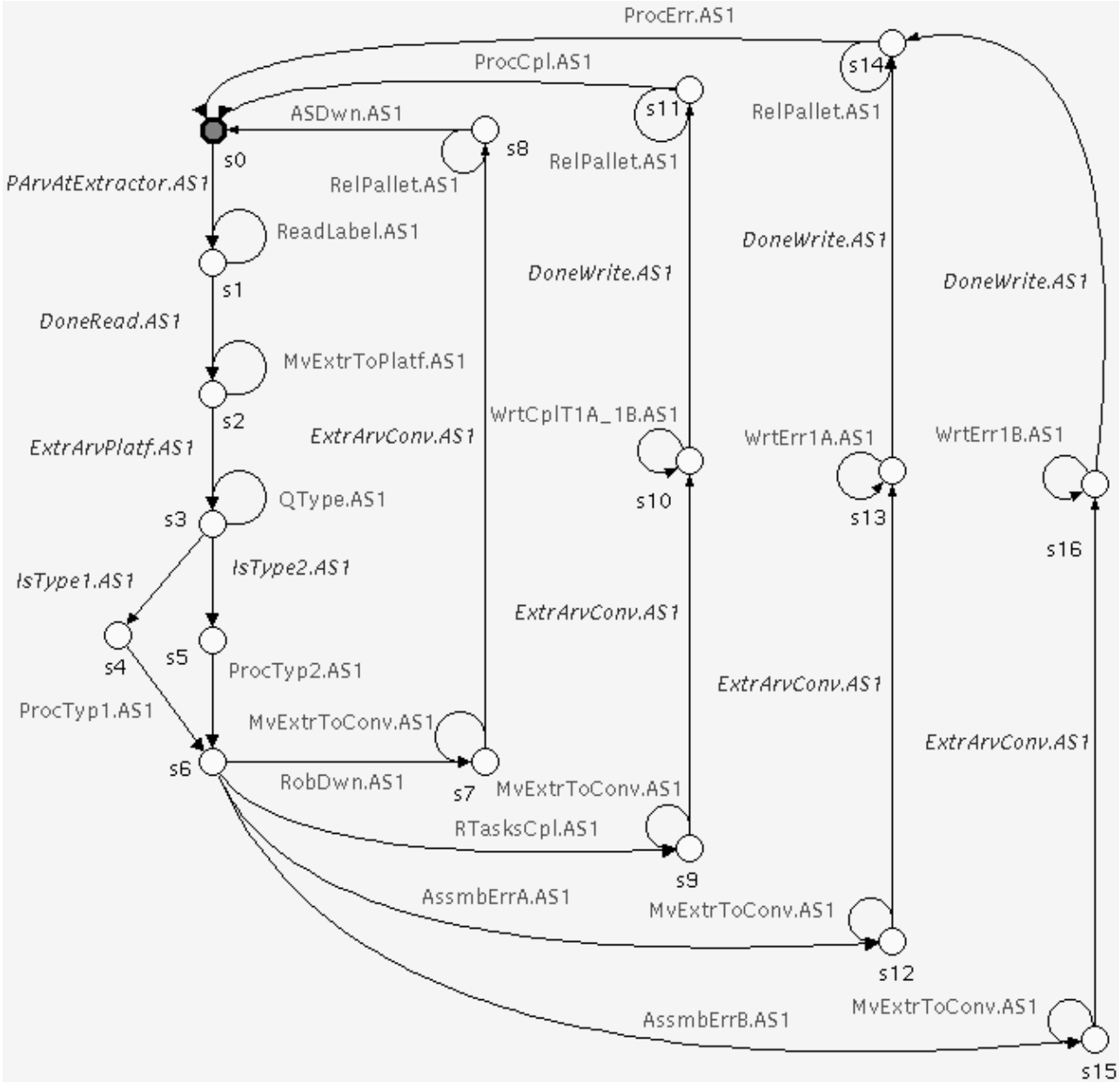


Interface to low level k = AS1, AS2

- They are identical so only have to design and verify once.
- Provides functionality indicated in interface.
- Accepts pallet at gate and presents to robot for assembly.
- Robot performs correct task based on pallet type.
- Pallet released and status of operation reported.

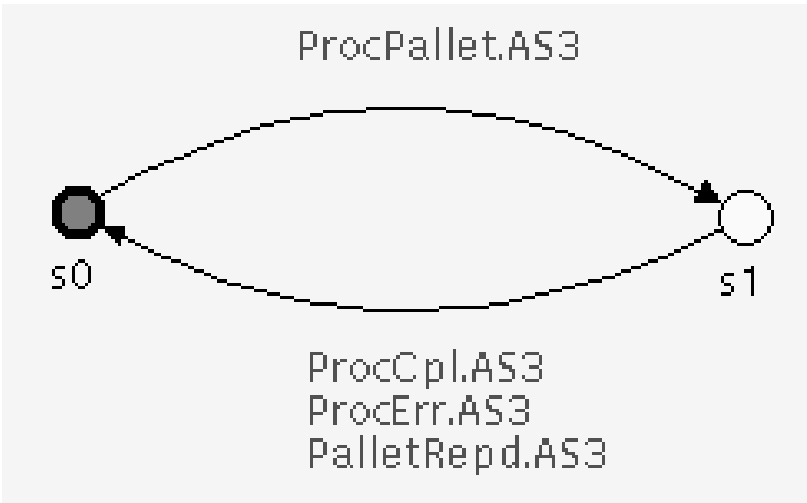


Supervisor: HndlPallet.AS1



Interface to AS3

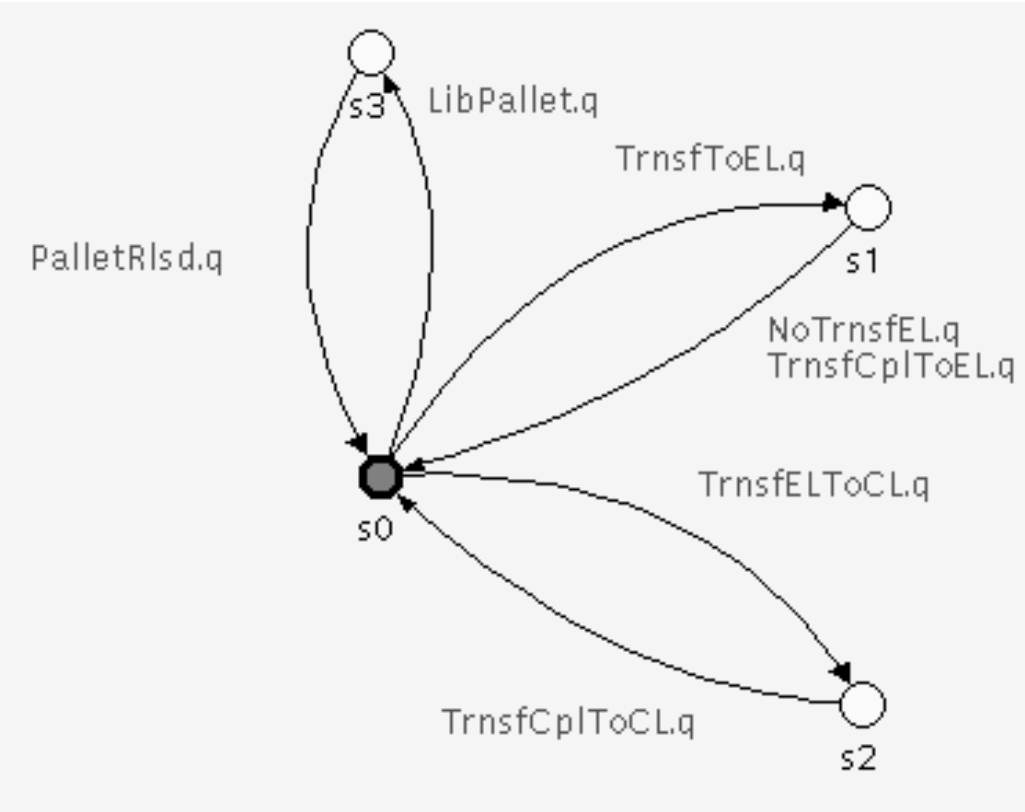
- Provides functionality in interface below.
- Behaviour similar to AS1 and AS2.
- **Differences:**
  - Can repair damaged pallets.
  - Assumed never to break down.
  - Can substitute for other stations when down.



Low Levels TU1 and TU2

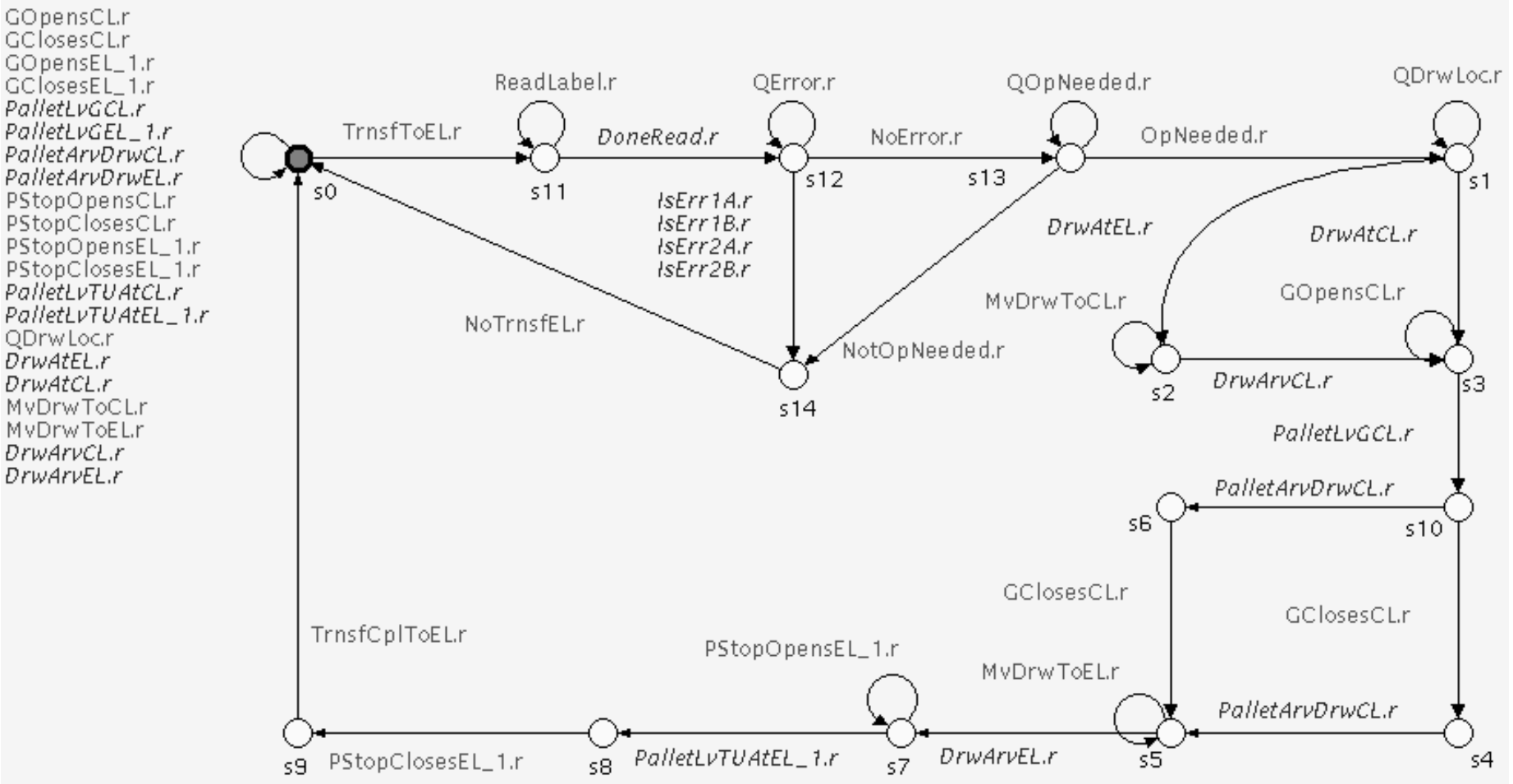
- Identical so only have to design and verify once.
- Provides functionality in interface below.
- Transfers pallets between CL and external loop.

- Pallets at EL always transferred.
- Pallet at CL gate can be liberated or transferred (only if pallet undamaged and next task pending).



Supervisor: HndlTrnsfToEL.r

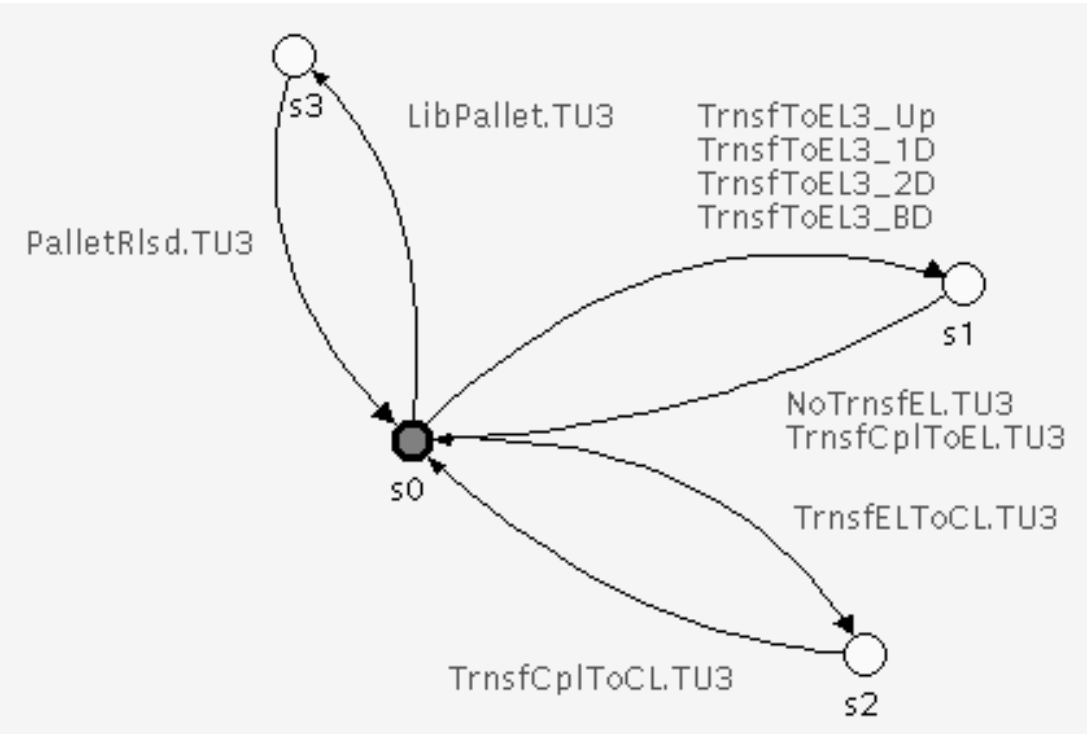
- Controls pallet transfers from CL to EL.



Low Level TU3

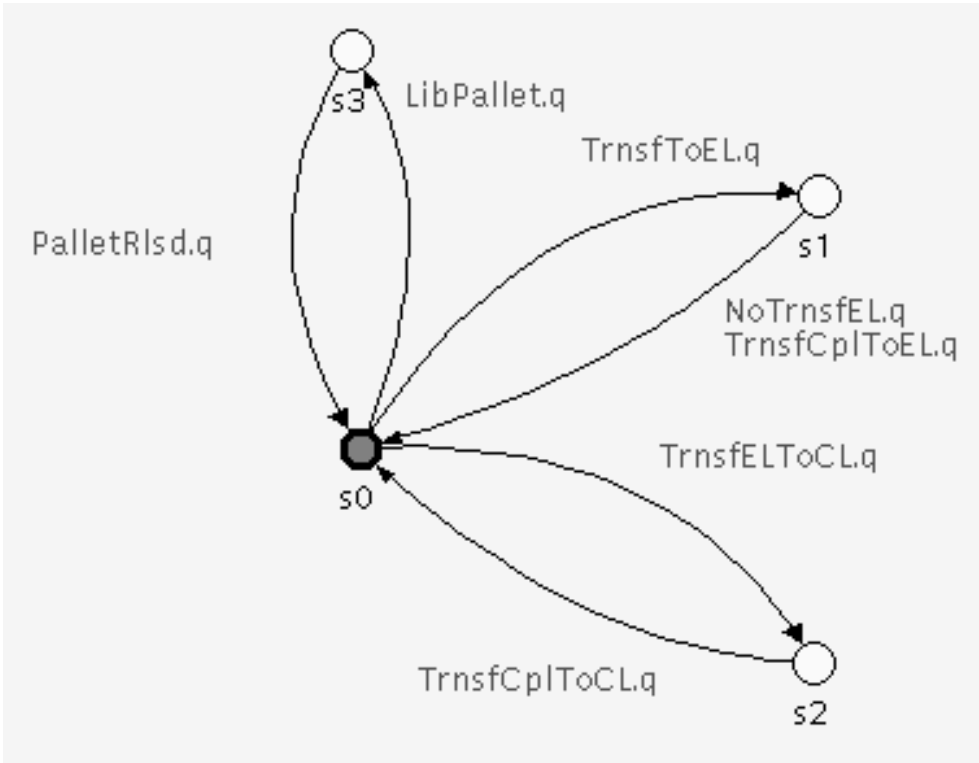
- Functionality similar to TU1 and TU2.
- Provides functionality in interface below.
- TU3 differs in criteria to send pallet to EL.

- Transfer all damaged pallets to EL.
- If pending assembly station down then transfer pallet to EL so AS3 can substitute for down AS.



Low Level TU4

- Similar to TU1 and TU2.
- Provides functionality in interface below (take $q =$ TU4).
- Differs in criteria to transfer pallet to external loop.
- Pallets required to leave system in order: type 1, type 2, type 1, ...
- Only transfers pallet if type fits order.



Verifying System

- Estimated worst case closed-loop state space: 7×10^{21}.
- Verified system level-wise controllable, nonblocking and interface consistent.
- Conclude by **nonblocking Theorem** and **controllability Theorem** that flat system nonblocking, and flat supervisor controllable for flat plant.
- Computation ran for 3 minutes, using 500MB of memory.
- Ran on 2Ghz Athlon 64 bit system, 4GB of RAM, and running Linux.
- A standard nonblocking verification failed due to lack of memory.

Algorithm Complexity

- Let N_H denote the size of the statespace of $\mathbf{G}_H$, while N_I and N_L are upper bounds for the statespace size of $\mathbf{G}_{I_j}$ and $\mathbf{G}_{L_j}$ $(j = 1, \ldots, n)$, respectively.

- The limiting factor for a monolithic algorithm would be $N_H N_L^n$ (ie for a flat system $\mathbf{G}_H || \mathbf{G}_{L_1} || \ldots || \mathbf{G}_{L_n}$) and similarly $N_H N_I^n$ (for high level $\mathbf{G}_H || \mathbf{G}_{I_1} || \ldots || \mathbf{G}_{I_n}$) for the HISC method.

- As long as $N_I \ll N_L$ (at least an order of magnitude), the HISC will provide significant improvement.

AIP Component Sizes

- Below are the size of various components for the AIP example.

Subsystem	States		
	Standalone	**$\|\|$ with $\mathbf{G}_{I_j}$**	**Size of $\mathbf{G}_{I_j}$**
$\mathbf{G}_H$	1,480,864	3,306,240	8,192
AS1	1,795	120	4
AS2	1,795	120	4
AS3	1,199	203	2
TU1	98	98	4
TU2	98	98	4
TU3	204	204	4
TU4	152	152	4

AIP Algorithm Complexity

- Substituting actual data from the AIP, we get:
  - $N_L^n = (120)^2(203)(98)^2(204)(152) = 8.71 \times 10^{14}$
  - $N_I^n = (4)^2(2)(4)^4 = 8,192.$
- This is a potential savings of 11 orders of magnitude!

HISC Tradeoffs

- The trade-off for this increase in computational efficiency is a more restrictive architecture.
- Interface approach restricts knowledge about internal details of components, and only allows supervisors to disable local and interface events.
- Result is we sacrifice global maximal permissiveness to obtain a (generally) suboptimal solution, but one that is more tractable.
- As similar interface-based approaches are common in both hardware and software, it's likely this method will be widely applicable.

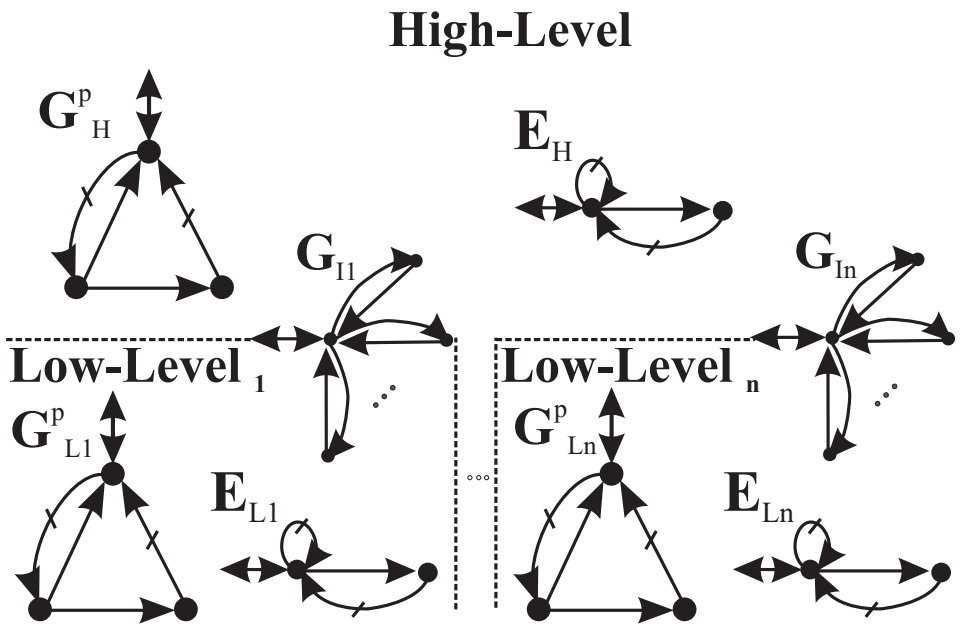
HISC Synthesis

- So far, we have provided supervisors and verified that the system satisfied the HISC conditions.
- If they do not, we must modify them until they do satisfy the conditions.
- Pengcheng Dai developed a per level synthesis method.
- Replaced high-level supervisor and each low-level supervisor with a corresponding *specification* DES.
- Constructs for each level a maximally permissive supervisor that satisfies the needed HISC conditions.

HISC System Structure For Synthesis

- Each level has its own specification.
- For further reading, see:

  R.J. Leduc, P. Dai, and R. Song, "Synthesis Method for Hierarchical Interface-Based Supervisory Control," *IEEE Trans. on Automatic Control*, vol 54, no 7, pp. 1548-1560, Jul. 2009.



Symbolic HISC Methods

- So far, HISC algorithms used explicit lists of DES states, and transitions.
- Means size of each level limited to around 10^7 states when using 1GB of memory.
- Want to represent states and transitions symbolically and implement using Binary Decision Diagrams (BDD) to save space and increase speed.
- Sets of states and transitions were represented as predicates.
- For instance, predicate $P : X \to \{0, 1\}$ might represent the set of reachable states such that for state $x \in X$, $P(x) = 1$ iff the state is reachable.
- Raoguang Song developed symbolic method for both verification and synthesis.

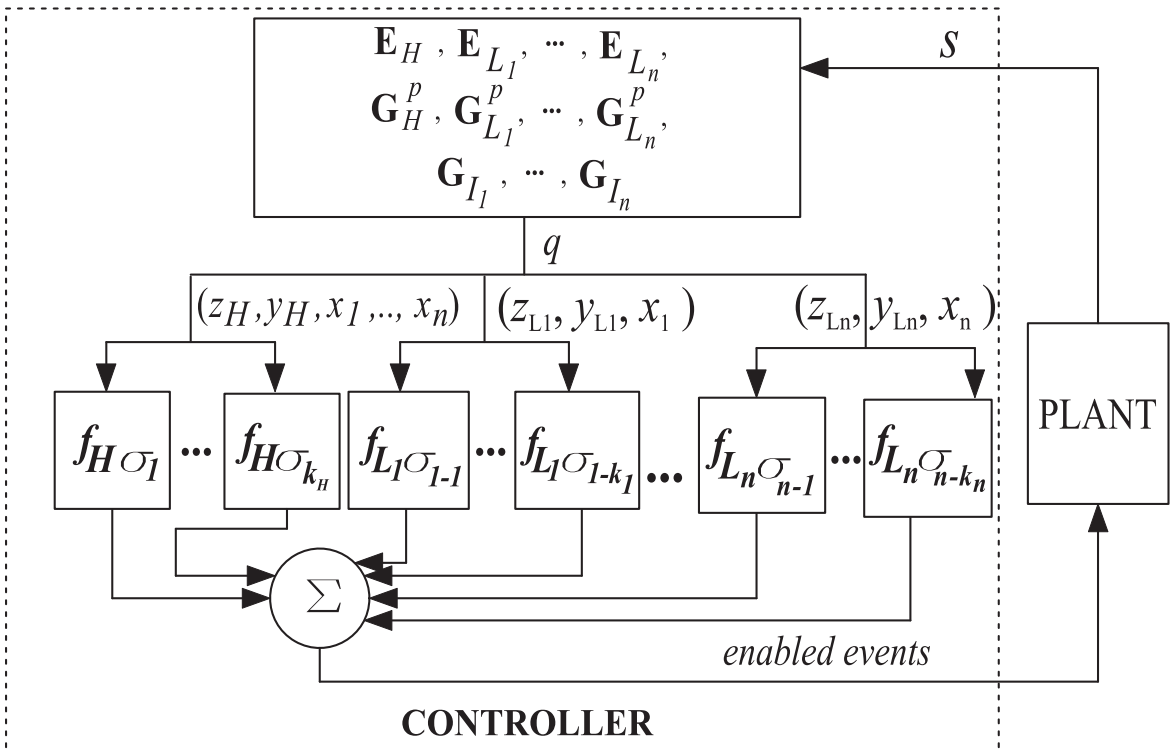
Symbolic Method and AIP Example

- Previously, AIP example took 3 minutes to verify, using 500MB of memory.
- With BDD-based software, Original AIP example took 2 seconds, and estimated 30 MB of memory.
- Song created greatly extended version of AIP example.
- Used less than 160MB of RAM and took 26 minutes to verify HISC conditions with high-level 10^{13} states, and system worst case 7.04×10^{26} states.
- Flat verification with BDD tool quickly used up all available RAM and failed to complete after 24 hours.
- Performed synthesis on system. Used less than 160MB of RAM and took 128 minutes to complete. High-level was 10^{15} states, and system worst case 1.51×10^{30} states.

HISC Controller Implementation

- Song also proposed a compact implementation for synthesized DES that takes advantage of the HISC method.
- For further reading on both topics, see:

  Raoguang Song and Ryan J. Leduc. Symbolic synthesis and verification of hierarchical interface-based supervisory control. Proc. of *WODES'06*, pages 419-426, Ann Arbor, USA, July, 2006.



HISC and Low Data Events

- Drawback of HISC was that when the interface is at a marked state, the low-level could not provide any information.
- This made it impossible for the interface to model "polling behavior" or low-levels that behaved like buffers.
- To increase the types of systems interfaces can model, a new type of interface events, called low data events Σ_{LD}, was added.
- Low data events provide means for low-level to send information (data) through the interface, independent of the standard request-answer structure.
- Needed to keep state of interfaces consistent with the state of their corresponding low-levels.

LD Interfaces

- Below extends command-pair interfaces to allow low data events, but also allows request events that are not followed by answer events.

Definition

The j^{th} interface DES $\mathbf{G_{I_j}} = (X_j, \Sigma_{I_j}, \xi_j, x_{o_j}, X_{m_j})$ is a *LD interface* if the following properties are satisfied:

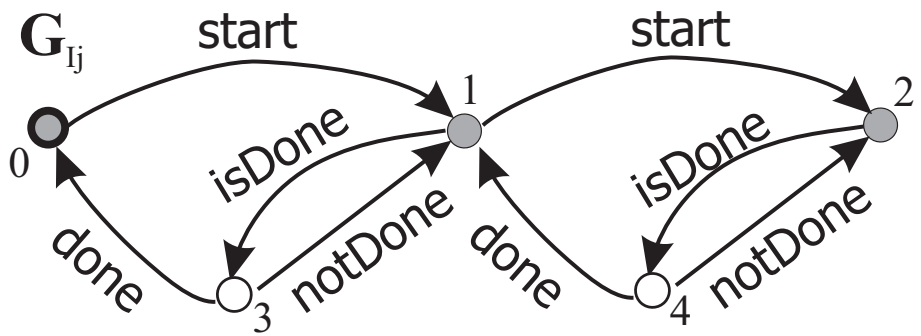
1. $x_{o_j} \in X_{m_j}$

2. $(\forall x \in X_{m_j})(\forall \sigma \in \Sigma_{I_j})\, \xi_j(x, \sigma)! \Rightarrow [\sigma \in \Sigma_{R_j}] \ \vee \ [\sigma \in \Sigma_{LD_j} \wedge \xi_j(x, \sigma) \in X_{m_j}]$

3. $(\forall x \in X_j - X_{m_j})(\forall \sigma \in \Sigma_{I_j})\, \xi_j(x, \sigma)! \Rightarrow$
   $[\sigma \in \Sigma_{A_j} \wedge \xi_j(x, \sigma) \in X_{m_j}] \vee [\sigma \in \Sigma_{LD_j}]$

LD Interface Example

- Interface could correspond to a machine at the low-level with an effective internal buffer of two.
- The *isdone*-{*done*, *notdone*} sequence shows an example of polling.



$\sum_{Rj}$={isDone,start}, $\sum_{Aj}$={done}, $\sum_{LDj}$={notDone}

LD Interface Consistent

Definition

The n^{th} degree $(n \geq 1)$ interface system composed of DES $\mathbf{G}_H, \mathbf{G}_{I_1}, \mathbf{G}_{L_1}, \ldots, \mathbf{G}_{I_n}, \mathbf{G}_{L_n}$, is *LD interface consistent* with respect to the alphabet partition given by (1), if for all $j \in \{1, \ldots, n\}$, the following conditions are satisfied:

Multi-level Properties

1. The event set of $\mathbf{G}_H$ is Σ_{IH}, and the event set of $\mathbf{G}_{L_j}$ is Σ_{IL_j}.
2. $\mathbf{G}_{I_j}$ is a LD interface.

High-Level Property

3. $(\forall s \in \mathcal{H} \cap \mathcal{I})\ \mathsf{Elig}_{\mathcal{I}_j}(s) \cap (\Sigma_{A_j} \dot{\cup} \Sigma_{LD_j}) \subseteq \mathsf{Elig}_{\mathcal{H}}(s)$

LD Interface Consistent - II

Low-Level Properties

4. $(\forall s \in \mathcal{L}_j \cap \mathcal{I}_j)\ \mathsf{Elig}_{\mathcal{I}_j}(s) \cap \Sigma_{R_j} \subseteq \mathsf{Elig}_{\mathcal{L}_j}(s)$

5. $(\forall s \in \Sigma^*.\Sigma_{R_j} \cap \mathcal{L}_j \cap \mathcal{I}_j)$
   $\mathsf{Elig}_{\mathcal{L}_j \cap \mathcal{I}_j}(s\Sigma^*_{L_j}) \cap \Sigma_{A_j} = \mathsf{Elig}_{\mathcal{I}_j}(s) \cap \Sigma_{A_j}$ where

$$\mathsf{Elig}_{\mathcal{L}_j \cap \mathcal{I}_j}(s\Sigma^*_{L_j}) := \bigcup_{l \in \Sigma^*_{L_j}} \mathsf{Elig}_{\mathcal{L}_j \cap \mathcal{I}_j}(sl)$$

6. $(\forall s \in \mathcal{L}_j \cap \mathcal{I}_j)$
   $s \in \mathcal{I}_{m_j} \Rightarrow (\exists l \in \Sigma^*_{L_j})\ sl \in \mathcal{L}_{m_j} \cap \mathcal{I}_{m_j}.$

LD Level-Wise Nonblocking and Controllability

- LD Level-Wise Controllability is essentially unchanged.
- For further reading, see:

  R.J. Leduc, "Hierarchical Interface-Based Supervisory Control with Data Events," *International Journal of Control*, vol. 82, no. 5, pp. 783 - 800, May, 2009.

Definition

The n^{th} degree $(n \geq 1)$ interface system composed of DES $\mathbf{G}_H, \mathbf{G}_{I_1}, \mathbf{G}_{L_1}, \ldots, \mathbf{G}_{I_n}, \mathbf{G}_{L_n}$, is said to be *LD level-wise nonblocking* if the following conditions are satisfied:

(I) *LD nonblocking at the high-level*:

$$(\forall s \in \mathcal{H} \cap \mathcal{I})(\exists s' \in (\Sigma - \Sigma_{LD})^*)$$
$$ss' \in \mathcal{H}_m \cap \mathcal{I}_m$$

(II) *nonblocking at the low-level*:

$$(\forall j \in \{1, \ldots, n\})\, \overline{\mathcal{L}_{m_j} \cap \mathcal{I}_{m_j}} = \mathcal{L}_j \cap \mathcal{I}_j$$