Hierarchical Interface-based Supervisory Control: Serial Case

R.J. Leduc,[1,3] B.A. Brandin,[2] W.M. Wonham,[1] and M. Lawford[3]

[1]Dept. of Electrical and Computer Engineering, University of Toronto
[2]Siemens Corporate Research
[3]Dept. of Computing and Software, McMaster University
email: leduc@control.toronto.edu, bertil.brandin@mchp.siemens.de, wonham@control.toronto.edu, lawford@mcmaster.ca

Abstract - *In this paper we present a hierarchical method that decomposes a system into two subsystems, and restricts the interaction of the subsystems by means of an interface. We present a definition for an interface, and define a set of interface consistency properties that can be used to verify if a discrete-event system (DES) is nonblocking and controllable. Each clause of the definition can be verified using only one of the two subsystems; thus the complete system model never needs to be constructed, offering significant savings in computational effort. Additionally, the development of clean interfaces facilitates re-use of the component subsystems.*

1 Introduction

In the area of Discrete-Event Systems (DES), two common tasks are to verify that a composite system, based on a cartesian product of subsystems, is (i) nonblocking and (ii) controllable. The main obstacle to performing these tasks is the combinatorial explosion of the product state space. Although many methods have been developed to deal with this problem (modular control [1, 8, 17, 21], decentralised control [11, 18, 22], model aggregation methods [2, 3, 5, 20, 23], and multi-level hierarchy [4, 7, 12, 13, 19, 24]), large-scale systems are still problematic, particularly for verification of nonblocking.

To deal with the complexity of large scale systems, the software engineering community has long advocated the decomposition of software into modules (components) that interact via well defined interfaces (e.g., [14, 15, 16]). Recently the supervisory control community has begun to advocate a similar approach [9, 10, 6]. These approaches develop well defined interfaces between components to provide the structure to allow local checks to guarantee global properties such as controllability [6] or nonblocking [9].

In this paper, we present an interface-based hierarchical method to verify if a system is nonblocking and controllable, extending the work in [9]. While in general the method can decompose the system into multiple "parallel" subsystems (see [10]), for the purposes of the present paper we restrict ourselves to the special case where the system is split into two subsystems that interact via an interface DES that regulates the subsystems' interaction. The most significant feature that distinguishes the work from [6] is the results regarding nonblocking.

In the remainder of the paper we first describe the general setting and provide the preliminary definitions. We then present a set of (local) consistency requirements that the interface and subsystems must satisfy to guarantee global nonblocking. Next, we examine controllability in this setting and provide a small illustrative example. We close by discussing the application of the method to a large industrial example (750 million reachable states).

2 Setting and Preliminaries

With *hierarchical interface-based supervisory control,* what we are proposing is essentially a master-slave system, where a *high level subsystem* sends a command to a *low level subsystem,* which then performs the indicated task and sends back a reply. Figure 1 shows conceptually the structure and information flow of the system. We call this the serial case as communication occurs in a serial fashion between the two subsystems.

To capture the restriction of the flow of information imposed by the interface, the alphabet of the plant (Σ) is split into four disjoint alphabets: Σ_H, Σ_L, Σ_R, and Σ_A. The events in Σ_H are called *high level events* and the events in Σ_L *low level events* as these events appear only in the high level and low level models, respectively.

The alphabets Σ_R and Σ_A are called collectively *inter-*

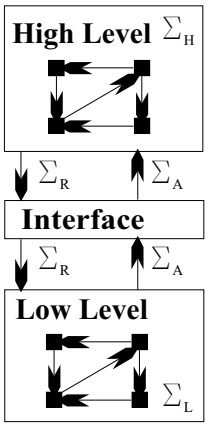

Figure 1: Interface Block Diagram.

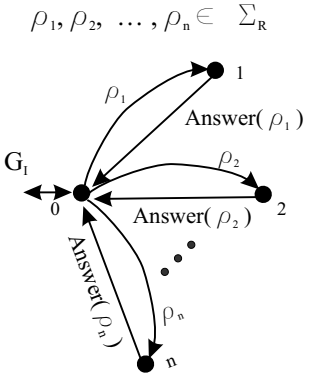

Figure 2: Interface Specification.

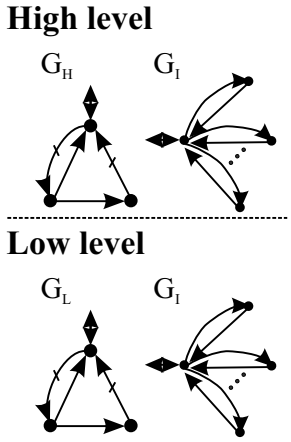

Figure 3: Two Tiered Structure of the System.

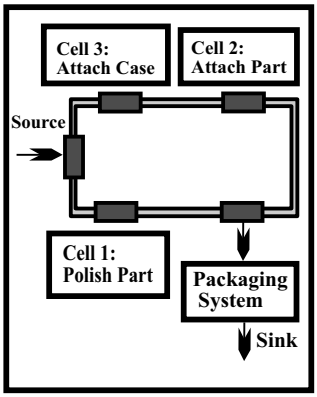

Figure 4: Block Diagram of Plant

face events. These events are common to both levels of the hierarchy and represent communication between the two subsystems. The events in Σ_R, called *request events*, represent commands sent from the *high level subsystem* to the *low level subsystem.* The events in Σ_A are *answer events* and represent the low level's responses to the *request events* (high-level commands).

Finally, we will be using the eligibility operator in our definitions. For a language $L \subseteq \Sigma^*$ and a string $s \in \Sigma^*$, the operator $\text{Elig}_L : \Sigma^* \to \text{Pwr}(\Sigma)$ is defined as follows:

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

2.1 Interface Definitions and Notation

To define an *interface,* the designer selects a set of *request events*, and then for each *request event*, the designer defines a set of *answer events.* In essence, the designer defines a map $\textbf{Answer} : \Sigma_R \to \text{Pwr}(\Sigma_A)$. For $\rho \in \Sigma_R$, $\textbf{Answer}(\rho)$ is the set of possible answers the *low level subplant* could provide after receiving request ρ. For consistency, we add the constraints 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. Figure 2 shows how an *interface* is expressed as a DES. The required structure for an *interface* is given by DES G_I.

For our setting, we assume the *high level subsystem* is modelled by DES G_H (defined over event set $\Sigma_H \cup \Sigma_R \cup \Sigma_A$), the *low level subsystem* by DES G_L (defined over event set $\Sigma_L \cup \Sigma_R \cup \Sigma_A$), and the interface by DES G_I (defined over $\Sigma_R \cup \Sigma_A$). Also, the *high level* will mean $\textbf{sync}(G_H, G_I)$,[1] and the *low level* $\textbf{sync}(G_L, G_I)$. The overall structure of the system is displayed in Figure 3.

We next assume that the alphabet partition is specified by $\Sigma := \Sigma_H \dot\cup \Sigma_L \dot\cup \Sigma_R \dot\cup \Sigma_A$ and take the *flat system* to be:

$$G = \textbf{sync}(G_H, G_L, G_I)$$

To simplify the notation in proofs, we introduce some terminology. As we will often be referring to different groupings of events, we define the following subsets:

$$\begin{aligned} \Sigma_I &:= \Sigma_R \dot\cup \Sigma_A && \textit{Interface Events} \\ \Sigma_{IH} &:= \Sigma_H \dot\cup \Sigma_R \dot\cup \Sigma_A && \textit{Interface \& High Level Events} \\ \Sigma_{IL} &:= \Sigma_L \dot\cup \Sigma_R \dot\cup \Sigma_A && \textit{Interface \& Low Level Events} \end{aligned}$$

To work with languages defined over subsets of Σ, we define the following natural projections:

$$\begin{aligned} P_{IH} &: \Sigma^* \to \Sigma_{IH}^* \\ P_{IL} &: \Sigma^* \to \Sigma_{IL}^* \\ P_I &: \Sigma^* \to \Sigma_I^* \end{aligned}$$

and the following useful languages:

$$\begin{aligned} \mathcal{H} &:= P_{IH}^{-1}(L(G_H)), & \mathcal{H}_m &:= P_{IH}^{-1}(L_m(G_H)) & \subseteq \Sigma^* \\ \mathcal{L} &:= P_{IL}^{-1}(L(G_L)), & \mathcal{L}_m &:= P_{IL}^{-1}(L_m(G_L)) & \subseteq \Sigma^* \\ \mathcal{I} &:= P_I^{-1}(L(G_I)), & \mathcal{I}_m &:= P_I^{-1}(L_m(G_I)) & \subseteq \Sigma^* \end{aligned}$$

3 Interface Consistency and Nonblocking

We now present the interface properties that the system must satisfy to ensure that it interacts with the *interface* correctly.

Serial Interface Consistent: The system composed of DES G_H, G_L and G_I, is *serial interface consistent* with respect to the alphabet partition $\Sigma := \Sigma_H \dot\cup \Sigma_L \dot\cup \Sigma_R \dot\cup \Sigma_A$, if the following properties are satisfied:

[1] The operation **sync** is the synchronous product operation from CTCT [21].

Multi-level Properties

1. The event set of G_H is Σ_{IH}, and the event set of G_L is Σ_{IL}.
2. G_I is an *interface* for the alphabet partition $\Sigma := \Sigma_H \dot\cup \Sigma_L \dot\cup \Sigma_R \dot\cup \Sigma_A$

High Level Properties

3. $\mathcal{H}\Sigma_A \cap \mathcal{I} \subseteq \mathcal{H}$

Low Level Properties

4. $\mathcal{L}\Sigma_R \cap \mathcal{I} \subseteq \mathcal{L}$
5. $(\forall s \in \Sigma^*.\Sigma_R \cap \mathcal{L} \cap \mathcal{I})$
   $\text{Elig}_{\mathcal{L}\cap\mathcal{I}}(s\Sigma_L^*) \cap \Sigma_A = \text{Elig}_{\mathcal{I}}(s) \cap \Sigma_A$

   where $\text{Elig}_{\mathcal{L}\cap\mathcal{I}}(s\Sigma_L^*) := \cup_{l\in\Sigma_L^*} \text{Elig}_{\mathcal{L}\cap\mathcal{I}}(sl)$
6. $(\forall s \in \mathcal{L} \cap \mathcal{I})$
   $s \in \mathcal{I}_m \Rightarrow (\exists l \in \Sigma_L^*)\ sl \in \mathcal{L}_m \cap \mathcal{I}_m$

These properties mean the following:

1: This property asserts that the high and low levels can only share *request* and *answer events.*

2: This property states that DES G_I is a proper interface of the form defined in Section 2.1 for the specified alphabet partition.

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

4: This property asserts that *request events* must be eligible in *the low level subsystem* (G_L), if the event is eligible in the *interface.*

5: This property asserts that immediately after a *request event* has occurred, there exists a path via strings in Σ_L^* to each *answer event* that can follow the *request event.* These paths may vanish after *low level events* occur.

6: 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.*

3.1 Level-wise Nonblocking

In this section, we define the nonblocking requirements each level must satisfy.

Definition: The system composed of DES G_H, G_L, and G_I, is said to be *serial level-wise nonblocking* if the following conditions are satisfied:

(I) $\overline{\mathcal{H}_m \cap \mathcal{I}_m} = \mathcal{H} \cap \mathcal{I}$ *nonblocking at the high level*

(II) $\overline{\mathcal{L}_m \cap \mathcal{I}_m} = \mathcal{L} \cap \mathcal{I}$ *nonblocking at the low level*

We now present our main result, the *serial interface nonblocking theorem.*

Theorem 1 *If the system composed of DES G_H, G_L, and G_I is* serial level-wise nonblocking *and* serial interface consistent *with respect to the alphabet partition $\Sigma := \Sigma_H \dot\cup \Sigma_L \dot\cup \Sigma_R \dot\cup \Sigma_A$, then*

$L(G) = \overline{L}_m(G)$, *where* $G = \mathbf{sync}(G_H, G_L, G_I)$

Proof: *See [10].*

4 Controllability

Now that we have discussed nonblocking in the serial interface setting, we consider controllability.

4.1 Definitions

For nonblocking we were only concerned with the *high* and *low level subsystems,* ignoring distinctions between plants and supervisors. For controllability, we need to split the subsystems into their plant and supervisor components. We will do so as shown in Figure 5.

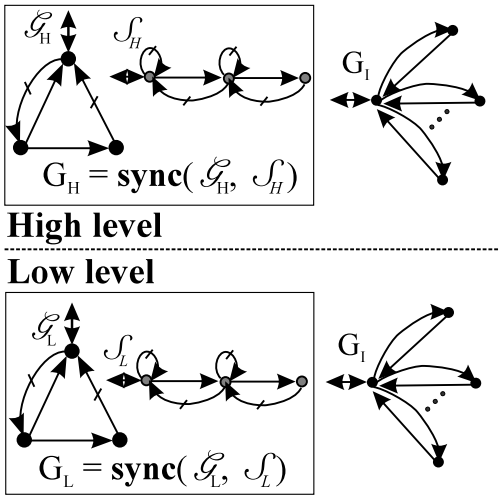

Figure 5: Plant and Supervisor Subplant Decomposition

We define the *high level plant* to to be $\mathcal{G}_H$, and the *high level supervisor* to be $\mathcal{S}_H$ (both defined over event set Σ_{IH}). Similarly, the *low level plant* and *supervisor* are $\mathcal{G}_L$ and $\mathcal{S}_L$ (defined over event set Σ_{IL}). We define the *high* and *low level subsystems* as follows:

$$G_H := \mathbf{sync}(\mathcal{G}_H, \mathcal{S}_H) \qquad G_L := \mathbf{sync}(\mathcal{G}_L, \mathcal{S}_L)$$

The reader should note that the definition of a *serial interface system* that we present here in terms of plant and supervisor components, is the general form of such systems. The form we defined in Section 2.1 (in terms of *high* and *low level subsystems*) is a special case of the general form, achieved by applying the above identities

for G_H and G_L. We will refer to the original form, used to simplify nonblocking definitions and proofs, as the *serial subsystem based form.*

We can now define our *flat supervisor* and *plant:*

$$\textbf{Plant} := \textbf{sync}(\mathcal{G}_H, \mathcal{G}_L) \qquad \textbf{Sup} := \textbf{sync}(\mathcal{S}_H, \mathcal{S}_L, G_I)$$

As we want to express the languages of **Plant** and **Sup** in terms of their components, we define the following terminology:

$$\begin{aligned} \mathbf{H} &:= P_{IH}^{-1}L(\mathcal{G}_H), \qquad \mathbf{H}_{\mathcal{S}} := P_{IH}^{-1}L(\mathcal{S}_H), \qquad \subseteq \Sigma^* \\ \mathbf{L} &:= P_{IL}^{-1}L(\mathcal{G}_L), \qquad \mathbf{L}_{\mathcal{S}} := P_{IL}^{-1}L(\mathcal{S}_L), \qquad \subseteq \Sigma^* \end{aligned}$$

Note that $\mathbf{H} = \overline{\mathbf{H}}$, $\mathbf{L} = \overline{\mathbf{L}}$, $\mathbf{H}_{\mathcal{S}} = \overline{\mathbf{H}}_{\mathcal{S}}$, and $\mathbf{L}_{\mathcal{S}} = \overline{\mathbf{L}}_{\mathcal{S}}$.

4.2 Level-wise Controllability

We now define the controllability requirements for each level. We adopt the standard partition $\Sigma = \Sigma_u \dot\cup \Sigma_c$ splitting our alphabet into *uncontrollable* and *controllable events.*

Serial Level-wise Controllable: The system composed of plant components $\mathcal{G}_H$, $\mathcal{G}_L$, supervisors $\mathcal{S}_H$, $\mathcal{S}_L$, and interface G_I, is said to be *serial level-wise controllable* with respect to the alphabet partition $\Sigma := \Sigma_H \dot\cup \Sigma_L \dot\cup \Sigma_R \dot\cup \Sigma_A$, if the following conditions are satisfied:

(I) The alphabet of $\mathcal{G}_H$ and $\mathcal{S}_H$ is Σ_{IH}, the alphabet of $\mathcal{G}_L$ and $\mathcal{S}_L$ is Σ_{IL}, and the alphabet of G_I is Σ_I

(II) $(\mathbf{L}_{\mathcal{S}} \cap \mathcal{I})\Sigma_u \cap \mathbf{L} \subseteq (\mathbf{L}_{\mathcal{S}} \cap \mathcal{I})$

(III) $\mathbf{H}_{\mathcal{S}}\Sigma_u \cap (\mathbf{H} \cap \mathcal{I}) \subseteq \mathbf{H}_{\mathcal{S}}$

To summarise the definition, **Point I** states that the plants, supervisors, and interface have the indicated event sets. **Point II** states that the *interface* and $\mathcal{S}_L$ are together controllable for the low level plant $\mathcal{G}_L$. **Point III** states that supervisor $\mathcal{S}_H$ is controllable for the high level plant $\mathcal{G}_H$, when it is already under the control of the *interface.*

4.3 Serial Controllability Theorem

In this section, we present our theorem for verifying controllability in this setting. In essence, this theorem asserts that if the system is *serial level-wise controllable*, then controllability can be checked for each level separately in order to determine that the system's *flat supervisor* is controllable for the system's *flat plant.*

Theorem 2 *If the system composed of plant components $\mathcal{G}_H$, $\mathcal{G}_L$, supervisors $\mathcal{S}_H$, $\mathcal{S}_L$, and interface G_I, is* serial level-wise controllable *with respect to the alphabet partition $\Sigma := \Sigma_H \dot\cup \Sigma_L \dot\cup \Sigma_R \dot\cup \Sigma_A$, then:*

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

Proof: *See [10].*

5 Simple Manufacturing Example

In this section, we present a simple manufacturing example to illustrate the *hierarchical interface-based supervisory control.*

5.1 Description of Manufacturing Unit

As shown in Figure 4, the manufacturing unit is composed of three cells connected by a conveyor belt. In front of each cell, is a part acquisition unit that automatically stops a part and holds it until it is given a release command. Parts enter the system at the far left and exit at the bottom right. After the item exits the conveyor system, it goes to a packaging machine.

The associated plant models can be seen in Figure 5.2, namely **Attach Case to Assembly**, **Polish Part**, **Attach Part to Assembly**, **Packaging System**, and **Path Flow Model**.

5.2 Defining Infrastructure

The first step in design is to decide which plant models belong to the *high level subsystem,* and which to the *low level subsystem.* The division we have chosen can be seen in Figure 5.2. We have added plant model **Define New Events** which introduces events *attch_ptA, attch_ptB, finA_attch,* and *finB_attch.* These events provide a more versatile means to interact with cell 2.

We define our *interface* to be the DES G_I shown in Figure 5.2. We define the alphabet partition $\Sigma := \Sigma_H \dot\cup \Sigma_L \dot\cup \Sigma_R \dot\cup \Sigma_A$ as follows:

$\Sigma_R =$ {*start_pol*, *attch_ptA*, *attch_ptB*, *start_case*}

$\Sigma_A =$ {*comp_pol*, *finA_attch*, *finB_attch*, *compl_case*}

$\Sigma_H =$ {*part_ent*, *part_arr1*, *part_lv1*, *partLvExit*, *str_exit*, *fin_exit*, *part_arr2*, *recog_A*, *recog_B*, *part_lv2*, *part_arr3*, *part_lv3*, *take_item*, *allow_exit*, *package*}

$\Sigma_L =$ {*take_pt*, *str_ptA*, *str_ptB*, *compl_A*, *compl_B*, *ret_pt*, *dip_acid*, *polish*, *str_rlse*, *attch_case*}

Our *uncontrollable/controllable* event partition can be seen immediately from Figure 5.2, after noting that controllable transitions are denoted by a slash across their arrow.

5.3 Designing Supervisors

Now that we have defined our *interface,* we design the low level supervisors that 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.*

For cell one, we want the sequence *dip_acid-polish* to be repeated twice, after a *start_pol* event occurs. The

supervisor is shown in Figure 5.2, and is labelled **Polishing Sequence**. For cell two, we have to provide supervisors so that the cell reacts appropriately when events *attch_ptA* and *attch_ptB* occur. We also must guarantee that *answer events finA_attch* and *finB_attch* only occur when they have the appropriate meaning. The DES **Affix Part** in Figure 5.2 shows how this is done.

We now design high level supervisors that use the *interface.* Figure 5.2 shows a supervisor (**Sequence Tasks**) that allows a part to visit each cell, executes the appropriate command for the cell and the part type, and then allows the part to leave the conveyor system. The figure also shows a supervisor (**Exit Buffer**) that implements a two item buffer for the packaging system. Finally, we note that the above supervisors were designed by hand, but we could have also employed synthesis methods.

5.4 The Final System

We our now ready to define our system components. Figure 5.2 shows our *high level subsystem, plant,* and *supervisor,* DES G_H, $\mathcal{G}_H$, and $\mathcal{S}_H$. We also have our *low level subsystem, plant,* and *supervisor,* DES G_L, $\mathcal{G}_L$, and $\mathcal{S}_L$. They are defined to be the synchronous product of the indicated automata.

Examining our system, we found it to be *serial interface consistent, serial level-wise nonblocking,* and *serial level-wise controllable.* We can thus conclude by **Theorem 1** that the *flat system* is nonblocking, and by **Theorem 2** that the *flat supervisor* is controllable for the *flat plant.*

5.5 Concurrency of Subsystems

Before concluding this example, we comment on the inherent concurrency of the *high* and *low levels.* Unlike state expansion methods such as Wang [19] and Gohari [7] that expand a high level state into a group of low level states, the interface method is based on the *synchronous product*, limiting information flow, and a set of consistency rules. This allows the *high level* to remain active while the *low level* is active, and thus operate concurrently.[2] In the cited state expansion methods, the high level state would remain fixed while the low level becomes active.

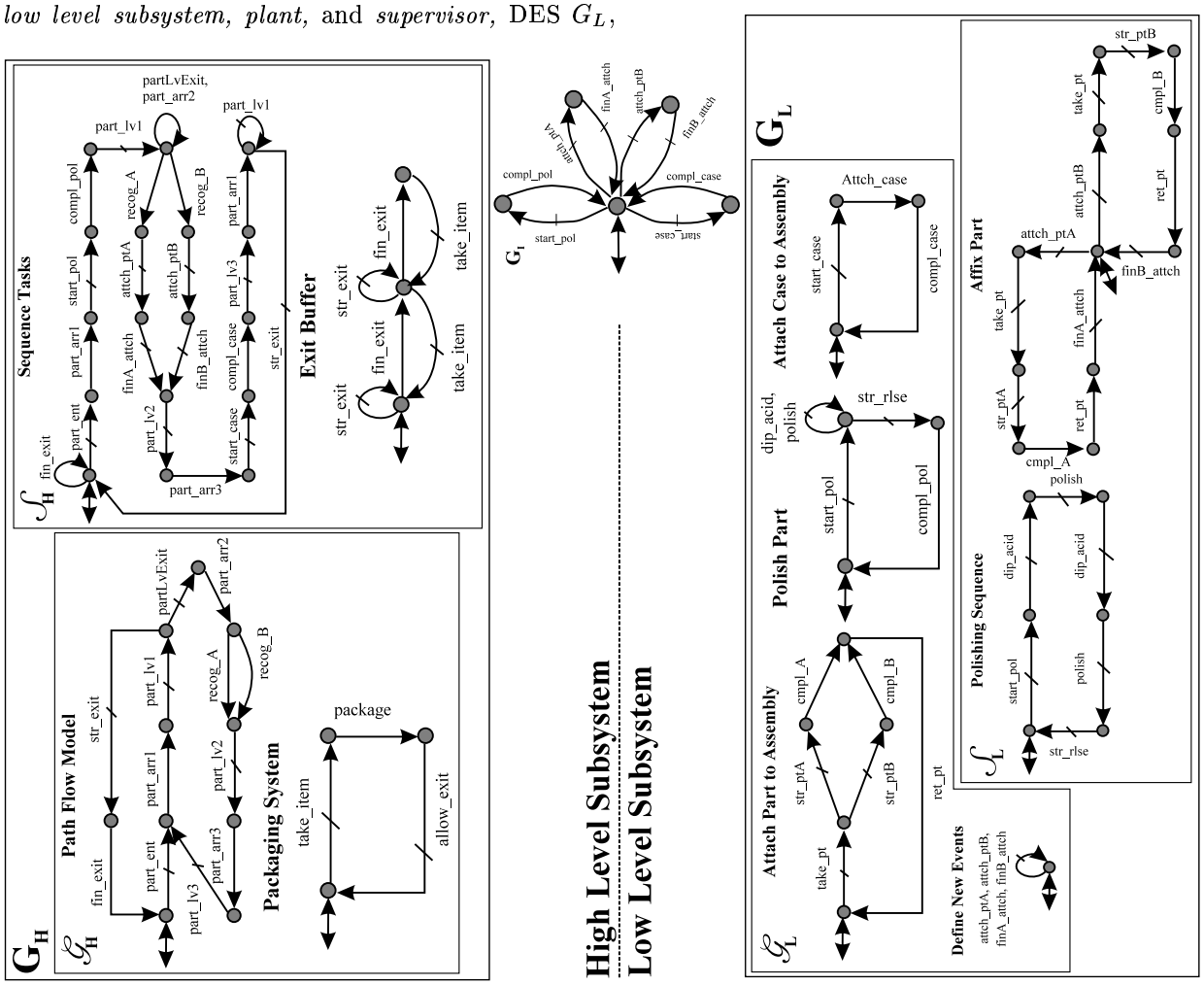

Figure 6: Complete System Definition

[2] Examine the behaviour of event *fin_exit* to see this.

6 Conclusions

We have also applied our method to a large car door locking example[3] which is composed of 54 DES, and has a composite model of approximately 750 million reachable states. The system was found to be *serial interface consistent*, *serial level-wise nonblocking*, *serial level-wise controllable,* hence nonblocking and controllable by **Theorem 1** and **Theorem 2**. The computation was run on an AMD K6 300 Linux workstation with 256MB of RAM, and 1.5GB of swap space. It took 19 minutes, and used 91MB of memory. A standard nonblocking verification was also attempted on the monolithic system. It ran for 38 minutes and consumed 1.7GB of memory before running out of memory.

Hierarchical interface-based supervisory control offers an effective method to model systems with a natural client-server architecture. The method offers an intuitive way to model and design the system. As each requirement can be verified using only one of the two subsystems, the entire plant model never needs to be constructed or traversed (in computer memory), offering potentially significant savings in computation.

It is clear from the definitions in Sections 3, 3.1, and 4.2, that once we have defined our *interface* and event partition, evaluating our *high* and *low level subsystems* for compliance can be done independently of each other. This means we can evaluate one *high (low) level subsystem* and use it with any *low (high) level subsystem* that satisfies the low (high) level portion of our definitions for the given *interface* and event partition. This provides us with the infrastructure required for component reuse.

[3] Unfortunately, the details of this example can't be released at this time for proprietary reasons.

References

[1] N. Alsop. *Formal Techniques for the Procedural Control of Industrial Processes.* PhD thesis, Dept of Chem. Eng. and Chem. Tech., Imperial College, London, 1996.

[2] R. Alur and T. Henzinger. Local liveness for compositional modelling of fair reactive systems. In *Proc. of 7th CAV*, LNCS, p. 166–179, 1995.

[3] A. Aziz, V. Singhal, and G. Swamy. Minimizing interacting finite state machines: A compositional approach to language containment. In *Proc. of IEEE Int. Conf. on Comp. Design: VLSI in Computers and Processors*, p. 255–261, Cambridge, MA, Oct 1994.

[4] Y. Brave and M. Heymann. Control of discrete event systems modeled as hierarchical state machines. *IEEE Trans. on Automatic Control*, 38(12):1803–1819, Dec 1993.

[5] P. Caines and Y. Wei. The hierarchical lattices of a finite machine. *Systems Control Letters*, 25:257–263, 1995.

[6] E. W. Endsley, M. R. Lucas, and D. M. Tilbury. Modular design and verification of logic control for reconfigurable machining systems. Submitted to Discrete Event Dynamic Systems: Theory and Applications.

[7] P. Gohari. A linguistic framework for controlled hierarchical DES. Master's thesis, Dept. of Elec. and Comp. Eng., University of Toronto, Toronto, Ont, 1998.

[8] R. Leduc. PLC implementation of a DES supervisor for a manufacturing testbed: An implementation perspective. Master's thesis, Dept. of Elec. and Comp. Eng., University of Toronto, Toronto, Ont, 1996.

[9] R. Leduc, B. Brandin, and W.M. Wonham. Hierarchical interface-based non-blocking verification. In *Proc. of the Canadian Conf. on Elec. and Comp. Eng.*, p. 1–6, May 2000.

[10] R.J Leduc, W.M. Wonham, and M. Lawford. Hierarchical interface based supervisory control: Bi-level systems. Technical report, Systems Control Group, University of Toronto, Toronto, ON, Canada, 2001. To appear.

[11] F. Lin and W.M. Wonham. Decentralized control and coordination of discrete-event systems with partial observations. In *Proc. 27th IEEE Conf. Decision Contr.*, p. 1125–1130, Dec 1988.

[12] H. Liu, J. Park, and R. Miller. On hybrid synthesis for hierarchical structured Petri nets. Tech report, Dept. of Comp. Sci., Univ. of Maryland, College Park, MD, 1996.

[13] C. Ma. A computational approach to top-down hierarchical supervisory control of DES. Master's thesis, Dept of Elec. and Comp. Eng., University of Toronto, Toronto, Ont, 1999.

[14] D. L. Parnas. On the criteria to be used in decomposing systems into modules. *Communications of the ACM*, December:1053–1058, December 1972.

[15] D. L. Parnas. Use of abstract interfaces in the development of software for embedded computer systems. NRL Report 8047, Naval Research Laboratory, 1977.

[16] D. L. Parnas, P. C. Clements, and D. M. Weiss. The modular structure of complex systems. *IEEE Trans. on Software Eng.*, SE-11(3):259–66, March 1985.

[17] M.H. de Queiro and J.E.R. Cury. Modular supervisory control of large scale discrete event systems. In *Proc. of WODES 2000*, p. 103–110, Ghent, Belgium, Aug 2000.

[18] K. Rudie and W.M. Wonham. Think globally, act locally: decentralized supervisory control. *IEEE Trans. on Automatic Control*, 37(11):1692–1708, Nov 1992.

[19] B. Wang. Top-down design for RW supervisory control theory. Master's thesis, Dept. of Elec. and Comp. Eng., University of Toronto, Toronto, Ont, 1995.

[20] K.C. Wong. *Discrete-Event Control Architecture: An Algebraic Approach.* PhD thesis, Dept. of Elec. and Comp. Eng., University of Toronto, Toronto, Ont, 1994.

[21] W.M. Wonham. *Notes on Control of Discrete-Event Systems.* Dept. of Elec. and Comp. Eng., University of Toronto, 1999. http://odin.control.toronto.edu/DES/.

[22] T. Yoo and S. Lafortune. A general architecture for decentralized supervisory control of discrete-event systems. In *Proc. of WODES 2000*, p. 111–118, Ghent, Aug 2000.

[23] H. Zhong and W.M. Wonham. On the consistency of hierarchical supervision in discrete-event systems. *IEEE Trans. on Automatic Control*, 35(10):1125–1134, Oct 1990.

[24] M. Zhou, D. Wang, and I. Mayk. Using Petri nets for object-oriented design of command and control systems. *Intl. Journal of Intelligent Control and Systems*, 2(2):287–300, 1998.