SYNTHESIS OF HIERARCHICAL PROCEDURAL CONTROLLERS

Ross Baird*, Maria Celeste Colantonio, Sandro Macchietto
Centre for Process Systems Engineering,
Imperial College of Science, Technology and Medicine, London SW7 2BY

Abstract

This paper introduces Hierarchical Procedural Control Theory. It builds on the properties of Decomposition Theory to provide a mechanism to tackle large scale problems in synthesising sequential controllers for discrete-event batch operations. The generic behaviour of all the controllers is modelled as a finte state machine (FSM), enabling them to be combined and operated in series and parallel by other sequential controllers in a hierarchical structure. An application to a Clean in Place process in a Batch Pilot Plant is shown.

1 Introduction

Procedural Control Theory (PCT) was first introduced by Sanchez (1996). It provides a formal framework for the synthesis of controllers that handle event-driven activities in automated process plants. PCT derives from Supervisory Control Theory (SCT) (Ramadge and Wonham, 1987) where the discrete-event system and its desired behaviour are modelled as finite state machines (FSMs) with controllable and uncontrollable transitions. Sanchez (1996) derived a formalism to allow the enforcing of control actions in SCT in order to synthesise sequential controllers for process systems. Specifications are given in terms of predicate and linear temporal logic and then translated into FSMs. The resulting controllers are provably correct.

Further development by Alsop (1996) addressed large problems in PCT by means of Decomposition Theory in order to avoid state explosion. The complete procedure can be decomposed in series and parallel operation of controllers. Although state explosion can then be avoided, no methodology for synthesising the resulting overall hierarchical controller was given. It should be noted this topic has been approached within SCT (Hubbard and Caines, 1998) but not within PCT where the focus is on its application to the process industries.

This work presents a step forward for the synthesis of provable correct hierarchical procedural controllers using Decomposition Theory. Smaller size low level controllers are synthesised and then integrated by a higher level procedural controller that contains all the information regarding the order of execution of the lower controllers. The theoretical developments are illustrated on a Clean in Place operation included in a flexible Batch Pilot Plant.

In Section 2, the main aspects of Decomposition Theory are presented. Section 3 contains some previous developments in Hierarchical PCT along with the new results. Section 4 includes an application to the Clean in Place operation on the Batch Pilot Plant. Finally, some conclusions are drawn in Section 5.

2 Decomposition Theory

PCT is based on FSMs, predicate and linear temporal logic and therefore faces state explosion when dealing with large problems. To be able to avoid this, Decomposition Theory was developed (Alsop, 1996). It recognises that not all elementary components are required by every procedure and therefore smaller procedures operating on subsystems could be synthesised. In this work, the lower level procedural controllers are considered as procedural components that can be switched on and off in series or parallel by the higher level controller. In this sense, procedural components represent an aggregation of the finer events and states at the "next layer" down in the system. Rules were defined regarding when these smaller procedural controllers can operate and how they can interact with each other. Three main areas of relevance to this work are Parallel Decomposition, Series Decomposition and Procedure Initiation.

2.1 *Parallel Decomposition*

Parallel decomposition can be used when a process can be partitioned into subsystems operating concurrently. These subsystems, say x and y, must conform to the condition that the alphabet, Σ, of shared transitions contains only uncontrollable transitions, that is:

$$\Sigma_x \cap \Sigma_y \subseteq \Sigma_u. \tag{1}$$

This assures that the controllers C_x and C_y, operate independently. If both controllers conform to their respective specification then so does the whole system. Thus, for a process M under parallel control from C_x and C_y, the closed loop behaviour is given by:

$$L(C_x \uparrow C_y / M) = \mathcal{P}_x^{-1} L(C_x) \cap \mathcal{P}_y^{-1} L(C_y) \cap L(M) \tag{2}$$

*email:r.baird@ic.ac.uk

where $\mathcal{P}^{-1}$ is the inverse projection operator defined by Wonham (1996). This is more easily understood if the projection operator is first defined. The projection operator $\mathcal{P}_{\Sigma_p}$ deletes from strings in a language all occurrences of events not in Σ_p:

$$\begin{aligned} \mathcal{P}_{\Sigma_p} &: \Sigma^* \rightarrow \Sigma_p^* \\ \mathcal{P}_{\Sigma_p}(\sigma) &= \begin{cases} \sigma & \text{if } \sigma \in \Sigma_p \\ \epsilon & \text{if } \sigma \notin \Sigma_p \end{cases} \end{aligned}$$

The inverse projection $\mathcal{P}_{\Sigma_p}^{-1}$ adds to all strings in a language all transitions that are not in Σ_p:

$$\mathcal{P}_{\Sigma_p}^{-1} L_p = \{ s \subseteq \Sigma^* | \mathcal{P}_{\Sigma_p} s \in L_p \}. \quad (3)$$

The inverse projection operator then allows the asynchronous product to synchronise on alphabets outside each of the elementary components alphabets.

2.2 *Series Decomposition*

Series decomposition simplifies the problem of specification by dividing operating procedures into smaller manageable phases of operation. Defined within the operating specification are *subgoals*, which are states that the process must pass through on the way to the overall operation goal. The system is then decomposed between these subgoals. For a process M under series control from C_1 and C_2, the closed loop behaviour is defined by:

$$L(C_1 \rightarrow C_2/M) = L(C_1).\sigma_{c_1}.L(C_2) \cap L(\mathcal{S}_{\sigma_{c_1}}^{q_1} M) \quad (4)$$

where $\sigma_{c_1} \notin \Sigma$ is the completion transition or event that signals that the system has reached the subgoal, q_1, and $\mathcal{S}_{\sigma_{c_1}}^{q_1}$ defines the selfloop of σ_{c_1} at the subgoal state q_1 in M. Note that σ_{c_1} is an uncontrollable transition.

Procedure Initiation

If a procedural controller begins execution when the system it is controlling is not in the correct initial state, problems of system deadlock, livelock or uncontrollability may arise. Procedure Initiation Theory (Alsop, 1996), assures that a procedural controller's initial state was satisfied before it began execution. For this purpose, system pre-checks were defined as:

$$L(C_1/M) = \Sigma_1^*.\sigma_{s_1}.L(C_1) \cap L(\mathcal{S}_{\sigma_{s_1}}^{q_1} M), \quad (5)$$

where Σ_1^* is the set of all possible strings that can be created from the alphabet Σ_1 and $\sigma_{s_1} \notin \Sigma_1$ is the transition that represents the starting of the procedural controller. This is an external event not in the controller's alphabet that is forced by an external source. Note that for the controller being started it is an uncontrollable event.

The decomposition forms discussed above were proven to conform to PCT with regard to maintaining the properties of nonblocking, conformance to specification and controllability. However, Decomposition Theory does not properly address the implementation of the resulting procedural controllers as part of a controller hierarchy. In parallel and series decomposition, the order of execution of the procedural controllers was determined *a priori*. This makes the procedural controllers inflexible and removes the option of reusability, a very desirable practice.

The consideration of open loop behaviour before a controller executes has been addressed but has been neglected after operation, an especially important factor in series operation. Otherwise, a controller may inhibit the operation of other controllers that follow in the series.

Finally, the implementation of the higher procedure that coordinates the operation of those procedures that resulted from decomposition needs to be considered. In fact, the coordinating controllers were implemented manually thus reducing the impact of formally synthesising the lower controllers.

In the following section, we present an extension to the theory that enables a complete formal synthesis of a hierarchy of procedural controllers.

3 Hierarchical Procedural Control

Hierarchical PCT aims to overcome the above shortcomings by:

- developing techniques that create self contained procedural controllers, thus increasing their reusability, and
- creating a basis for modelling generic procedural controller behaviour so that it can be used to formally synthesise higher level coordinating procedural controllers.

3.1 *Before and After Execution*

In this section, previous modular PCT for pre-checking, series decomposition and parallel decomposition (Alsop, 1996) is extended to consider the open loop behaviour of a process before and after controller execution to enable their proper use in hierarchical procedural controllers.

In PCT, the language produced by two controllers operating in parallel is given by Eq. 2. If the alphabets Σ_x and Σ_y are disjoint, parallel operation is trivial. But, if transitions are shared between controllers, only when both controllers are operating will they be permitted to occur. This is due to the controllers not considering the open loop behaviour of the process before they begin execution. To consider open loop behaviour, pre-checks are added. Likewise, for controllers operating in series the open loop behaviour of the process before and after all controllers have executed must be considered.

Open loop behaviour is modelled by the term Σ_i^*. It represents any transition strings that occur when controller i is not operating. External commands, modelled by the external transitions σ_{s_i} and σ_{c_i}, are also required to start the controller and monitor when it finishes. They are uncontrollable transitions, associated with process responses, that model the assessment of the current process state when an external source requests controller i to be started. Because they are uncontrollable, the rules of prohibiting sharing transitions are not violated. So, Eq. 2 becomes:

$$\begin{aligned} L(C_x \uparrow C_y/M) &= \mathcal{P}_x^{-1}\{\Sigma_x^*.\sigma_{s_x}.L(C_x).\sigma_{c_x}.\Sigma_x^*\} \\ &\cap \mathcal{P}_y^{-1}\{\Sigma_y^*.\sigma_{s_y}.L(C_y).\sigma_{c_y}.\Sigma_y^*\} \\ &\cap L(\mathcal{S}_{\sigma_{s_x}}^{q_{0_x}}\mathcal{S}_{\sigma_{s_y}}^{q_{0_y}}\mathcal{S}_{\sigma_{c_x}}^{Q_{m_x}}\mathcal{S}_{\sigma_{c_y}}^{Q_{m_y}}M), \quad (6) \end{aligned}$$

where q_{0_i} is the initial state of controller i and Q_{m_i} is the set of marked states for controller i.
Similarly,

$$\begin{aligned} L(C_1 \rightarrow C_2/M) &= \Sigma^*.\sigma_{s_1}.L(C_1).\sigma_{c_1}.\sigma_{s_2}.L(C_2).\sigma_{c_2}.\Sigma^* \\ &\cap L(\mathcal{S}_{\sigma_{s_1}}^{q_{0_1}}\mathcal{S}_{\sigma_{c_1}}^{Q_{m_1}}\mathcal{S}_{\sigma_{s_2}}^{q_{0_2}}\mathcal{S}_{\sigma_{c_2}}^{Q_{m_2}}M). \quad (7) \end{aligned}$$

Traditionally, series controllers have been synthesised from the same transition alphabet. However, this is a conservative approach which is not necessary. Then Eq. 7 is augmented by adding the inverse projection operator $\mathcal{P}^{-1}$:

$$\begin{aligned} L(C_1 &\rightarrow C_2/M) \\ &= \mathcal{P}_1^{-1}\{\Sigma_1^*.\sigma_{s_1}.L(C_1).\sigma_{c_1}.\sigma_{s_2}.\Sigma_1^*.\sigma_{c_2}.\Sigma_1^*\} \\ &\cap \mathcal{P}_2^{-1}\{\Sigma_2^*.\sigma_{s_1}.\Sigma_2^*.\sigma_{c_1}.\sigma_{s_2}.L(C_2).\sigma_{c_2}.\Sigma_2^*\} \\ &\cap L(\mathcal{S}_{\sigma_{s_1}}^{q_1}\mathcal{S}_{\sigma_{c_1}}^{Q_{m_1}}\mathcal{S}_{\sigma_{s_2}}^{q_2}\mathcal{S}_{\sigma_{c_2}}^{Q_{m_2}}M). \quad (8) \end{aligned}$$

On closer inspection, Eq. 8 (series operation) is very similar to Eq. 6 (parallel operation). The difference between the two modes of operation, parallel and series, lies in the use of the start and completion transitions in the controller terms. For parallel operation, each controller term only contains the start and completion transitions for that controller. This is because during parallel operation each procedural controller is not concerned when the others start or complete their execution. For series operation, where the order of controller execution is very important, all controller terms contain the start and completion transitions of all the controllers operating in the series.

3.2 *Procedural Components*

As it has been already said, Hierarchical PCT uses start transitions σ_{s_i} and completion transitions σ_{c_i}. These are external transitions not in the controllers' alphabets ($\sigma_{s_i}, \sigma_{c_i} \notin \Sigma_i$), that are forced or observed by another controller. The group of external transitions

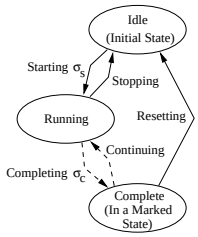

Figure 1: Finite State Machine representing a procedural component.

that control the operation of a procedural controller are termed *Procedural Transitions*. These transitions can include *start* and *complete* transitions plus *pause*, *hold*, *stop* and *abort*, to name a few. The ISA-S88.01 international standard present a way of modelling these procedural transitions as an FSM. In PCT, the model can be reduced to a simple three state elementary component as shown in Fig. 1. These FSMs, termed *procedural components* (Baird, 2000), can be used like any other elementary component to form part of another procedural component. Operating specifications are determined by either the rules regarding which controllers are permitted to operate concurrently, or specifications given by the designer regarding the order of execution of the lower level procedural controllers. Using these specifications, a *provably correct* higher level procedural controller is synthesised. In doing so, no flag variables (Elsag-Bailey, 1992) are required as the use of procedural components eliminates their need.

The practice is propagated through the complete controller hierarchy by creating a procedural component for each higher level procedural controller and synthesising an even higher level procedural controller, with as many hierarchical levels as desired. In other words, the procedural controller at each level is a procedural component of the complete controller.

3.3 *Integrating Controllers of Different Levels*

The languages generated by procedural controllers operating in parallel and series are defined by Eqs 6 and 8. When a higher level procedural controller is synthesised, a language of procedural transitions is generated to specify the order of execution of the lower level procedural controllers. This language is added to produce the language generated under hierarchical control. In the following, the first three terms on the right hand side of the equations will be referenced as *controller terms* and the fourth term as the *modelling term*. The symbol $\triangle$ represents parallel operation with a coordinating hierarchical controller, C_{hier}. Then, Eq. 6 becomes:

$$\begin{aligned} L(C_x \triangle C_y/M) &= \mathcal{P}_{hier}^{-1}L(C_{hier}) \\ &\cap \mathcal{P}_x^{-1}\{\Sigma_x^*.\sigma_{s_x}.L(C_x).\sigma_{c_x}.\Sigma_x^*\} \end{aligned}$$

$$\cap \quad \mathcal{P}_y^{-1}\{\Sigma_y^*.\sigma_{s_y}.L(C_y).\sigma_{c_y}.\Sigma_y^*\}$$
$$\cap \quad L(\mathcal{S}_{\sigma_{s_x}}^{q_{0x}}\mathcal{S}_{\sigma_{s_y}}^{q_{0y}}\mathcal{S}_{\sigma_{c_x}}^{Q_{m_x}}\mathcal{S}_{\sigma_{c_y}}^{Q_{m_y}}M), \quad (9)$$

which represents the language generated by two procedural controllers operated in parallel by a higher level procedural controller.

For series operation of two procedural controllers under the control of a higher level procedural controller (symbolised by $\triangleq$), the order of procedural transitions, $\sigma_{s_1}.\sigma_{c_1}.\sigma_{s_2}.\sigma_{c_2}$ is contained within $L(C_{hier})$. Then, Eq. 8 results in:

$$\begin{aligned} L(C_1 \triangleq C_2/M) &= \mathcal{P}_{hier}^{-1}L(C_{hier}) \\ &\cap \mathcal{P}_1^{-1}\{\Sigma_1^*.\sigma_{s_1}.L(C_1).\sigma_{c_1}.\Sigma_1^*\} \\ &\cap \mathcal{P}_2^{-1}\{\Sigma_2^*.\sigma_{s_2}.L(C_2).\sigma_{c_2}.\Sigma_2^*\} \\ &\cap L(\mathcal{S}_{\sigma_{s_1}}^{q_{0_1}}\mathcal{S}_{\sigma_{s_2}}^{q_{0_2}}\mathcal{S}_{\sigma_{c_1}}^{Q_{m_1}}\mathcal{S}_{\sigma_{c_2}}^{Q_{m_2}}M), \quad (10) \end{aligned}$$

which is identical to Eq. 9. From the above, we can consider the general case of a higher level controller, C_{hier}, coordinating several procedural controllers. The set of lower level controllers is denoted by C^I, where $C^I = \{C_1, C_2, C_3, .., C_i, .., C_n\}$ and C_i is a generic lower level controller. Thus, the language generated by a higher level controller coordinating n lower level controllers in C^I is:

$$\begin{aligned} L(C_{hier} \quad \triangle \quad & C^I/M) \\ &= \mathcal{P}_{hier}^{-1}L(C_{hier}) \\ &\cap \left(\bigcap_{i=1}^{n}\mathcal{P}_i^{-1}\{\Sigma_i^*.\sigma_{i_s}.L(C_i).\sigma_{i_c}.\Sigma_i^*\}\right) \\ &\cap \left(\bigcup_{i=1}^{n}L(\mathcal{S}_{\sigma_{i_s}}^{Q_{0_i}}\mathcal{S}_{\sigma_{i_c}}^{Q_{m_i}}M)\right), \quad (11) \end{aligned}$$

where $\triangle$ stands for hierarchical coordination.

The inverse projection operator, $\mathcal{P}_{hier}^{-1}$, in the above, enables the high level controller to synchronise on events in the lower controllers, conforming to the rules of parallel and hierarchical decomposition, $\Sigma_{hier} \cap \Sigma_i = \emptyset$.

3.4 *Properties of Hierarchical Control*

Let us show that the properties of PCT are preserved when a process is placed under the control of a hierarchical procedural controller.

Theorem 1 *For hierarchical systems, the closed loop behaviour generated by C_{hier} and C_i on M conforms to specification S if C_{hier} generates a closed loop behaviour on H_{hier} that conforms to specification S_{hier} and similarly C_i generates a closed loop behaviour on H_i that conforms to specification S_i.*

Proof

In hierarchical control theory, the language generated by the specification is:

$$L(S)' = \mathcal{P}_{\Sigma_{hier}}^{-1}L(S_{hier}) \cap (\bigcap_{i=1}^{n}\mathcal{P}_i^{-1}\{\Sigma_i^*.\sigma_{i_s}.L(C_i).\sigma_{i_c}.\Sigma_i^*\})$$

where S_{hier} defines how the specifications S_i interact; S_i specifies the operation of the process components, and $L(S)'$ represents specification that includes the starting and completion events, $\sigma_{i_s}, \sigma_{i_c} \notin \Sigma$, to coordinate the operation of the lower level controllers. Therefore,

$$L(S) = \mathcal{P}_{\Sigma}(L(S)').$$

For a hierarchical control structure to conform to specification, it must satisfy,

$$L(C_{hier} \triangle C^I/M) \subseteq L(S)'$$

from which it follows that

$$\begin{aligned} \mathcal{P}_{\Sigma}L(C_{hier} \triangle C^I/M) &\subseteq \mathcal{P}_{\Sigma}L(S)' \\ &\subseteq L(S). \end{aligned}$$

The language generated by a higher level controller coordinating n lower level controllers in C^I operating on M is:

$$\begin{aligned} &L(C_{hier} \triangle C_i/M) \\ &= \mathcal{P}_{\Sigma_{hier}}^{-1}L(C_{hier}) \cap \left(\bigcap_{i=1}^{n}\mathcal{P}_{\Sigma_i}^{-1}\{\Sigma_1^*.\sigma_{i_s}.L(C_i).\sigma_{i_c}.\Sigma_i^*\}\right) \\ &\quad \cap \left(\bigcup_{i=1}^{n}L(\mathcal{S}_{\sigma_{i_s}}^{Q_{0_i}}\mathcal{S}_{\sigma_{i_c}}^{Q_{m_i}}M)\right) \qquad \textit{from Eq. 11} \end{aligned}$$

Since,

$$\bigcup_{i=1}^{n}L(\mathcal{S}_{\sigma_{i_s}}^{Q_{0_i}}\mathcal{S}_{\sigma_{i_c}}^{Q_{m_i}}M) = \mathcal{P}_{\Sigma_{hier}}^{-1}L(H_{hier}) \cap \left(\bigcap_{i=1}^{n}\mathcal{P}_{\Sigma_i}^{-1}L(\mathcal{S}_{\sigma_{i_s}}^{Q_{0_i}}\mathcal{S}_{\sigma_{i_c}}^{Q_{m_i}}H_i)\right)$$

Substituting in equation 11

$$\begin{aligned} &L(C_{hier} \triangle C_i/M) \\ &= \mathcal{P}_{\Sigma_{hier}}^{-1}L(C_{hier}) \cap \left(\bigcap_{i=1}^{n}\mathcal{P}_{\Sigma_i}^{-1}\{\Sigma_1^*.\sigma_{i_s}.L(C_i).\sigma_{i_c}.\Sigma_i^*\}\right) \\ &\quad \cap \mathcal{P}_{\Sigma_{hier}}^{-1}L(H_{hier}) \cap \left(\bigcap_{i=1}^{n}\mathcal{P}_{\Sigma_i}^{-1}L(\mathcal{S}_{\sigma_{i_s}}^{Q_{0_i}}\mathcal{S}_{\sigma_{i_c}}^{Q_{m_i}}H_i)\right) \\ &= \mathcal{P}_{\Sigma_{hier}}^{-1}\{L(C_{hier}) \cap L(H_{hier})\} \\ &\qquad \cap \left(\bigcap_{i=1}^{n}\mathcal{P}_{\Sigma_i}^{-1}(\{\Sigma_1^*.\sigma_{i_s}.L(C_i).\sigma_{i_c}.\Sigma_i^*\} \right. \\ &\qquad\qquad \left. \cap L(\mathcal{S}_{\sigma_{i_s}}^{Q_{0_i}}\mathcal{S}_{\sigma_{i_c}}^{Q_{m_i}}H_i)\right) \\ &= \mathcal{P}_{\Sigma_{hier}}^{-1}L(C_{hier}/H_{hier}) \\ &\quad \cap \left(\bigcap_{i=1}^{n}\mathcal{P}_{\Sigma_i}^{-1}(\{\Sigma_1^*.\sigma_{i_s}.L(C_i/H_i).\sigma_{i_c}.\Sigma_i^*\}\right) \\ &\subseteq \mathcal{P}_{\Sigma_{hier}}^{-1}L(S_{hier}) \cap \left(\bigcap_{i=1}^{n}\mathcal{P}_{\Sigma_i}^{-1}\{\Sigma_i^*.\sigma_{i_s}.L(S_i).\sigma_{i_c}.\Sigma_i^*\}\right) \\ &\subseteq L(S)' \end{aligned}$$

□

In the following we make use of:

Lemma 1 *$\mathcal{P}_{\Sigma_i}^{-1}\{\Sigma_i^*.\sigma_s.L_m(C_i).\sigma_c.\Sigma_i^*\}$ and $L(\mathcal{S}_{\sigma_s}^{Q_0}\mathcal{S}_{\sigma_c}^{Q_m}M)$ are nonconflicting if H_i is internally consistent with M.*

Theorem 2 *For hierarchical systems, the closed loop behaviour generated by C_{hier} and C_i on M is nonblocking if*

1. *C_{hier}, C_i and M are nonblocking, and*
2. *$\mathcal{P}^{-1}_{\Sigma_{hier}} L_m(C_{hier})$, $\mathcal{P}^{-1}_{\Sigma_i}\{\Sigma_i^*.\sigma_s.L_m(C_i).\sigma_c.\Sigma_i^*\}$ and $L(\mathcal{S}^{Q_0}_{\sigma_s}\mathcal{S}^{Q_m}_{\sigma_c} M)$ are nonconflicting and trim.*

Proof

$$
\begin{aligned}
&L(C_{hier} \triangle C_i / M) \\
&= \mathcal{P}^{-1}_{\Sigma_{hier}} L(C_{hier}) \cap \mathcal{P}^{-1}_{\Sigma_i}\{\Sigma_i^*.\sigma_s.L(C_i).\sigma_c.\Sigma_i^*\} \\
&\quad \cap L(\mathcal{S}^{Q_0}_{\sigma_s}\mathcal{S}^{Q_m}_{\sigma_c} M) \qquad \textit{and if cond. 1 holds} \\
&= \overline{\mathcal{P}^{-1}_{\Sigma_{hier}} L_m(C_{hier})} \cap \overline{\mathcal{P}^{-1}_{\Sigma_i}\{\Sigma_i^*.\sigma_s.L_m(C_i).\sigma_c.\Sigma_i^*\}} \\
&\quad \cap L_m(\mathcal{S}^{Q_0}_{\sigma_s}\mathcal{S}^{Q_m}_{\sigma_c} M) \qquad \textit{and if cond. 2 holds} \\
&= \overline{\mathcal{P}^{-1}_{\Sigma_{hier}} L_m(C_{hier}) \cap \mathcal{P}^{-1}_{\Sigma_i}\{\Sigma_i^*.\sigma_s.L_m(C_i).\sigma_c.\Sigma_i^*\}} \\
&\quad \cap L_m(\mathcal{S}^{Q_0}_{\sigma_s}\mathcal{S}^{Q_m}_{\sigma_c} M) \\
&= \overline{L_m(C_{hier} \triangle C_i / M)} \qquad \textit{using Eq. 11} \qquad \square
\end{aligned}
$$

4 Case Study

The Hierarchical PCT briefly introduced above has been applied to a Clean in Place (CIP) unit in the Flexible Batch Pilot Plant in the Centre for Process Systems Engineering at Imperial College. The unit is used to prepare cleaning fluid for its use in other sections of the plant. Fig. 2 shows a diagram of the CIP Unit.

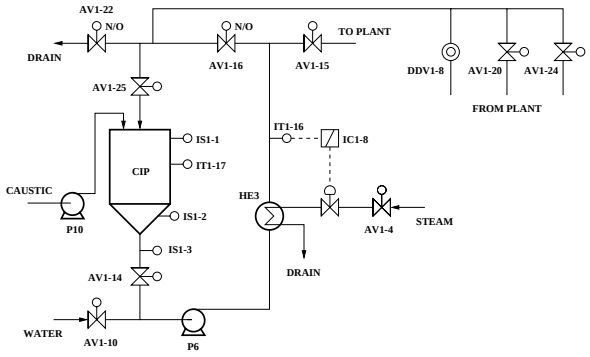

Figure 2: Batch Pilot Plant CIP Unit Schematic Diagram.

The Detergent Service Unit Operation supervises the preparation of hot caustic solution in the CIP tank. In total, 17 process devices are affected by this unit operation. The unit operation can be described as a series of three phases: filling the tank, preparing the detergent and rinsing the surrounding pipework. Procedure *Fill*, fills the CIP tank from empty with fresh water via pump P6 and valves AV1-16 and AV1-25. Procedure *Detergent*, comprises two sub procedures, *Heat* and *Dose* which are started once the contents of the tank are being recirculated through Heat Exchanger HE3 using P6. Procedure *Rinse* uses fresh water to flush the surrounding pipework and remove any detergent residue.

As shown by Fig. 3, the Unit Operation includes series and parallel decomposition resulting in a hierarchical procedural controller. Relating this to what was presenting in section 3, the procedures *Fill, Heat, Dose* and *Rinse* are low level controllers, $C_{Fill}, C_{Heat}, C_{Dose}, C_{Rinse} \in C^I$. Procedure *Detergent Service* is the higher controller that coordinates the others and thus contains the transitions $\sigma_{s_{Fill}}, \sigma_{c_{Fill}}, \sigma_{s_{Detergent}}, \sigma_{c_{Detergent}}, \sigma_{s_{Rinse}}$ and $\sigma_{c_{Rinse}}$. Procedure *Detergent* is both a lower level and higher level controller as it coordinates the operation of C_{Heat} and C_{Dose} but is coordinated by *Detergent Service* at a higher level. Thus, *Detergent* contains the transitions $\sigma_{s_{Heat}}, \sigma_{c_{Heat}}, \sigma_{s_{Dose}}$ and $\sigma_{c_{Dose}}$, and also has a procedural component representation in *Detergent Service*.

To show that the six smaller procedural controllers are permitted to operate in the given order, the alphabets of the languages generated by each of the controllers must be compared. This ensures the controllers can operate independently and without conflicting. The simplest way to do this is to compare the sets of elementary components, $E_{Fill}, E_{Detergent}, E_{Heat}, E_{Dose}$, and E_{Rinse}, to see what devices and thus transitions are shared between controllers. Fig. 3 lists each controller's elementary component set as part of the initial state specification.

We first consider the series operation of Procedure *Fill* and Procedure *Detergent*. They share 9 elementary components:

$$
\begin{aligned}
&E_{Fill} \cap E_{Detergent} \\
&\quad = \{AV1-22, AV1-25, AV1-16, IS1-1, \\
&\quad\ \ P10, AV1-14, AV1-10, P6, IC1-8\}.
\end{aligned}
$$

Checking the initial and goal states of each common elementary component in each of the controllers, we see that their goal states in Procedure *Fill* is the same as their initial states in Procedure *Detergent*.

Likewise, Procedure *Detergent* and Procedure *Rinse* can operate in series as the 7 common elementary components are in the correct state when Procedure *Detergent* terminates and Procedure *Rinse* starts:

$$
\begin{aligned}
&E_{Detergent} \cap E_{Rinse} \\
&\quad = \{AV1-22, AV1-25, AV1-16, \\
&\quad\ \ AV1-14, AV1-10, P6, IC1-8\}.
\end{aligned}
$$

Finally, Procedures *Heat and Dose* do not share any controllable transitions and thus can operate in parallel, as shown by:

$$E_{Heat} \cap E_{Dose} = \emptyset.$$

A comparison of results between implementing a single controller and multiple controllers in a hierarchy is given in Table 1. For a single controller with 17 elementary components the model's potential size is too large for a single overall controller to be synthesised.

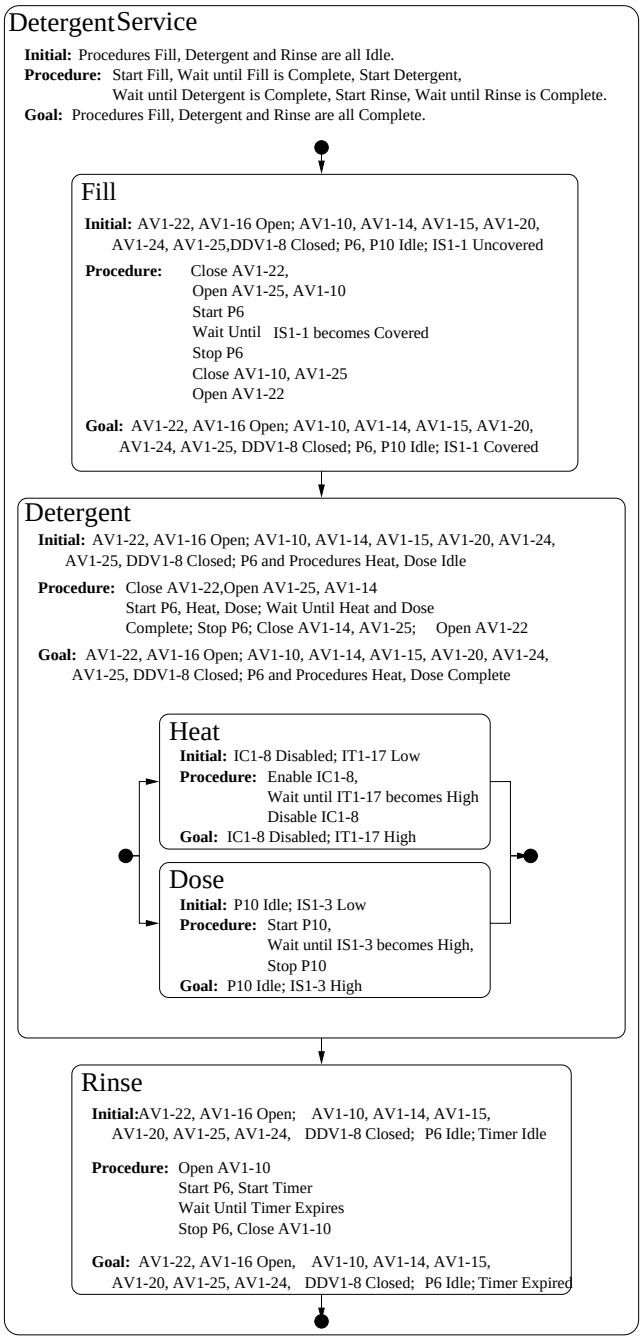

Figure 3: Controller Hierarchy for Detergent Service Unit Operation.

From the table, it is seen that the total number of potential states generated by the 6 smaller models is less than 6% of the potential size for the single model. From the practical point of view, specification of 6 problems each of order $\bigcirc(10^3)$ is much easier than to obtain one of order $\bigcirc(10^5)$.

The above shows that decomposition combined with hierarchical PCT aids in avoiding state explosion when synthesising sequential controllers for process plants, thus providing a practical tool for the operation designers. This example is part of a larger case study performed in Baird (2000) involving over 35 process devices.

Controller	No. of Comps	Potential Model Size	Cont. Size
Single	17	196 608	
Fill	12	4 096	15
Detergent	12	4 096	43
Heat	2	4	4
Dose	2	4	4
Rinse	11	3 072	7
Coordinating	3	27	7
Total		11 291	80

Table 1: Summary of Synthesis Results for CIP Preparation Unit Operation

5 Conclusions

The Hierarchical PCT presented extends the properties of Decomposition Theory within PCT to provide a mechanism for its application to large scale problems. Modelling the generic behaviour of a procedural controller as an FSM, they can be combined and controlled by other procedural controllers forming a Hierarchical Procedural Controller. The operation of a Clean in Place unit using the new development was illustrated.

References

Alsop, Nicholas (1996). Formal Techniques for the Procedural Control of Industrial Processes. PhD thesis. University of London.

Baird, Ross (2000). Synthesis and Implementation Procedural Controllers with Alarm Handling Capabilities. PhD thesis. University of London.

Elsag-Bailey (1992). *BATCH 90 User Manual*. Canada.

Hubbard, P. and P.E. Caines (1998). Trace-DC Hierarchical Supervisory Control with Applications to Transfer-Lines. In: *37th IEEE Conference on Decision and Control*. pp. 3293–3298.

Ramadge, P.J. and W.M. Wonham (1987). Supervisory control of a class of discrete-event processes. *SIAM Journal of Control and Optimization* **25**(1), 206–230.

Sanchez, Arturo (1996). *Formal Specification and Synthesis of Procedural Controllers for Process Systems*. Lecture Notes in Control and Information Sciences 212, Springer.

Wonham, W.M. (1996). *Notes on Control of Discrete–Event Systems*. Systems Control Group, Dept. of Electrical & Computer Engineering, University of Toronto. ECE 1636F/1637S.