DEPARTMENT OF COMPUTER SCIENCE

Abstract Process Definitions in CSP - A Case Study

P.N. Taylor

Technical Report No. 283

April 1997

Abstract Process Definitions in CSP

- A Case Study Example -

P. N. Taylor.

Department of Computer Science, Faculty of Information Sciences,

University of Hertfordshire, College Lane, Hatfield, Herts. AL10 9AB. U.K.

Tel: 01707 284763, Fax: 01707 284303, Email: p.n.taylor@herts.ac.uk

April 14, 1997

Abstract

This paper presents a CSP specification of a case study for an environmental control system for a building. The modification of CSP process definitions to incorporate reuse via inheritance is the main issue presented in this work. Incremental and behaviour modification are two techniques that are presented in order to model object-oriented inheritance.

In its current form CSP does not provide many facilities for reusing existing components of a specification. This paper presents an addition to the language of CSP by parameterising initial actions and subsequent behaviour in the signature of a process which then permits behaviour to be modified in situ as well as by the extension of action choice.

Keywords CSP, Reuse, Inheritance, Object-oriented specification.

Introduction

This paper presents an addition to the language of CSP that permits both strict and casual inheritance to be implemented. These two forms of inheritance can be categorised in the following way.

Strict Inheritance : The reuse of existing behaviour with possible extension to the choice of behaviour initially offered by a process. Behavioural compatibility with the parent is guaranteed.

Casual Inheritance : The reuse of existing behaviour with the modification to the sequence of actions initially offered by a process. Behavioural compatibility with the parent is not guaranteed.

Strict inheritance is the simple form of reuse as it simply requires the 'bolting on' of new branches of behaviour to an existing framework.

Casual inheritance alters the behaviour of the inherited process as it splices new actions into the existing action sequences to yield a new sequence of behaviour given the same initial actions.

A process definition that both extends initial choice and modifies existing action sequences is said to inherit in both a strict and casual sense.

The vehicle for discussion in this paper is a case study of an environmental control system (ECS) that provides many opportunities for the reuse of behaviour within its components.

Elements of the ECS that exhibit common behaviour are identified as targets of reuse and this behaviour is placed in common shared templates that form classes for the inheriting processes. A shared process is a template class (parent) process, where many individual (child) processes all reuse its behaviour; these child processes can in turn be template classes which may then be inherited from (forming a transitive inheritance hierarchy from the original parent to the latest generation of child process).

Object-oriented design, with its techniques for classifying objects, aids in the identification of generic reusable class descriptions which are encoded in CSP as processes.

Three template classes are identified within the ECS. Specialisations of these template classes form the next level of child class templates from which the ECS is eventually built. Instantiations of the child class templates are the actual physical objects within the ECS.

1 The Environmental Control System (ECS) Case Study

This section contains both an informal and formal description of the environmental control system (ECS). The ECS is itself a prime example of a concurrent communicating system and consequently is well suited to being formally specified in a process algebra notation, such as CSP.

In this paper we use standard CSP notation [1], plus a few enhancements of our own, to enable us to formalise the behaviour of the ECS monitored components. Our enhancements provide more flexibility in specification than are originally afforded by the standard CSP notation (namely in the areas of both behavioural reuse and modification).

1.1 ECS Informal Description

The ECS maintains safety, security, air quality and heating in a building. A hygrometer controls water saturation and a thermostat controls temperature. A fan and air conditioner control the air circulation in the building. An alarm sounds should a problem arise in specific ECS components. Smoke detectors can also signal an alarm, activate window controls and fire doors and warning lights are also activated in the event of an alarm being triggered. Finally, movement sensors control the efficiency of lights in public corridors and act as intruder sensors out-of-hours.

Three class templates (containing common shared behaviour) are identified from which the individual ECS components (i.e: members of the template class) are derived. These class templates are SWITCH, VALVE and SENSOR and their children are described as follows:

Templates **LightSwitch, WarningLight, Alarm and Fan** behave like SWITCH and can be turned on or off. WarningLight also sends a signal to a control room, external to the system. Alarm and Fan can also timeout and turn themselves off automatically. Also, Fan can be turned on or off externally via the environment as well as internally by the system.

Templates **AirConditioner, Heater, DoorControl and WindowControl** behave like the template VALVE which can be set to a specific level within a range. VALVE can be opened or closed within that range. Any attempts to set VALVE out-of-range result in an error signal being sent from VALVE which activates Alarm. The template AirConditioner has a gauge denoting internal temperature. Heater displays its current setting and internal temperature. DoorControl can also be opened and closed via the environment, together with WindowControl.

Templates **SmokeAlarm, Hygrometer and Thermostat** behave like SENSOR, reading data from the environment. Boundary values for high and low readings can be set. A warning signal occurs on the high or low channels should an appropriate boundary value be broken. SmokeAlarm can also trigger Alarm, WarningLight, WindowControl (to close) and DoorControl (to close), as well as incorporating a test button. Hygrometer includes a water saturation reading and Thermostat registers the air temperature reading.

The components (i.e: child instances) of the three main templates SWITCH, VALVE and SENSOR are joined together to complete the ECS. The diagram in figure 1 shows how the components are connected together. Note that Hygrometer, Thermostat and SmokeAlarm are the key com-

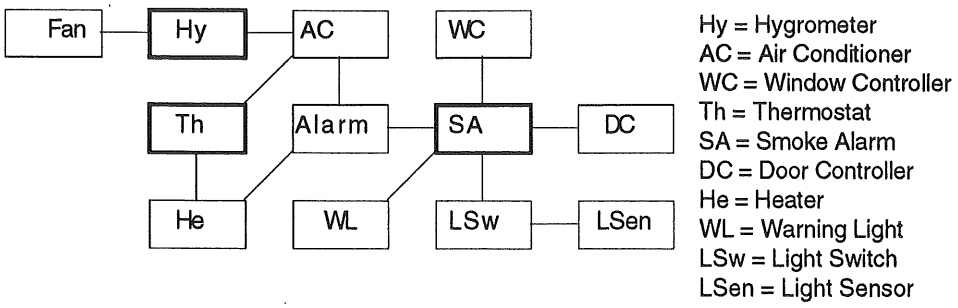

Figure 1: Simple View of the Environmental Control System (ECS).

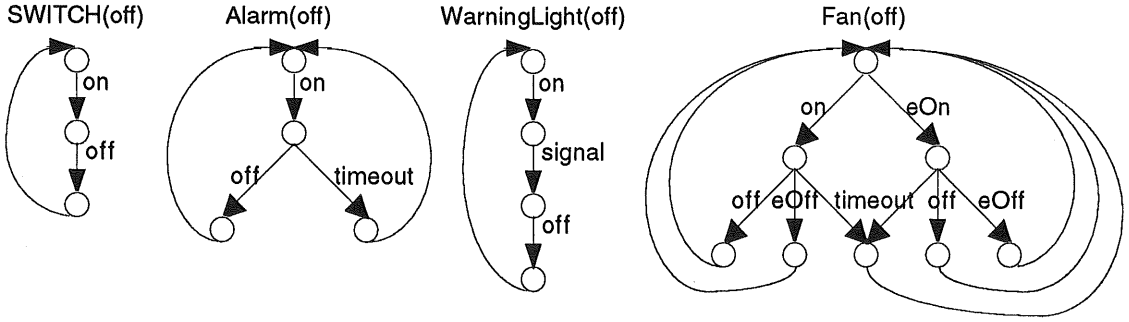

Figure 2: Transition Graphs of SWITCH-based ECS Templates

ponents in the ECS as they monitor the environment of the building and regulate the remaining components of the system. User interaction with the ECS is, of course, permitted (via externally visible channels—prefixed with 'e') but not necessary for the system to function.

1.2 The ECS described in CSP

The CSP language is now used to describe the ECS components. Firstly, each class template is specified, then the inheriting child class templates of each original parent class template. Reuse is shown as reference to an existing process definition used as part of a new process definition.

Transition graphs can be drawn to illustrate the behaviour of each derivative of SWITCH for example. Figure 2 represents the transition graphs for the original template SWITCH and its children. Each circle represents a state reached after the action on the labelled arc occurs. Recursion is shown as a state followed by an unlabelled arc representing an invisible transition back to the initial state of the process, forcing execution to repeat indefinitely.

1.2.1 Abstract Process Signatures

An addition to the language of CSP is proposed which permits the signature of a process to be parameterised. This parameterisation takes two forms:

1. initial actions

2. subsequent behaviour

We can write an expression to show the general form of an abstract process signature (APS).

$$\epsilon[a_1, \ldots, a_n]\langle P\rangle = (a_1 \rightarrow P \ldots \Box\ a_n \rightarrow P)$$

The first parameter denotes the initial action or actions of the process ϵ, followed by any subsequent behaviour P. If more than one initial action is offered then external choice connects the initial actions, followed by behaviour P.

The offering of multiple subsequent behaviour P_n is not available in this model as it remains unclear how the initial actions a_n may be related to P_n.

Recursion is defined using the standard recursive definition, found in [1, p.28]

$$\begin{aligned} X &= F(X) \\ \mu X : A.F(X) &= F(\mu X.F(X)) \end{aligned}$$

Applying the recursion law to ϵ we get the following result:

$$\epsilon[a_1, \ldots, a_n]\langle P\rangle = F(\epsilon[a_1, \ldots, a_n]\langle P\rangle)$$

$$\mu\, \epsilon[a_1, \ldots, a_n]\langle P\rangle : A.F(\epsilon[a_1, \ldots, a_n]\langle P\rangle) =$$
$$F(\mu\, \epsilon[a_1, \ldots, a_n]\langle P\rangle.F(\epsilon[a_1, \ldots, a_n]\langle P\rangle))$$

The previous expression states that a process with an abstract process signature (APS) can unfold an infinite number of times, providing the necessary recursion.

An example of recursion using the general case ϵ is now shown:

$$\epsilon[a_1, \ldots, a_n]\langle P\rangle = a_1 \rightarrow \epsilon[a_1, \ldots, a_n]\langle P\rangle \ldots \Box\ a_n \rightarrow \epsilon[a_1, \ldots, a_n]\langle P\rangle$$

1.3 Applying Abstract Process Signatures to the ECS

The foundations of the APS theory have now been introduced. The ECS SWITCH class template can now be defined using CSP incorporating APS theory.

$$SWITCH[a]\langle P\rangle = (a \rightarrow P)$$

An Alarm reuses and extends the behaviour of a SWITCH. Like so:

$$Alarm[a]\langle P \,\Box\, Q\rangle = SWITCH[a]\langle P \,\Box\, Q\rangle$$

The extra choice of behaviour in the Alarm template is passed as external choice in its behaviour parameter. Note that in CSP if P and Q are processes then so is $(P \,\Box\, Q)$, which meets the signature requirements for the definition of an abstract process signature. Note also that an Alarm template is simply a specialisation of a SWITCH template; an example of strict inheritance.

A WarningLight class template requires the insertion of an extra signal action into the inherited behaviour of SWITCH.

$$WarningLight[a]\langle P\rangle = SWITCH[a]\langle P\rangle$$

The signature for WarningLight is the same as for an original SWITCH. The renaming of SWITCH to WarningLight is performed to aid the readability of the final specification. Upon instantiation the extra action is inserted that separates SWITCH from WarningLight.

A LightSwitch template is a simple renaming of a standard switch therefore its definition as this abstract level is as simple as the definition for WarningLight.

$$LightSwitch[a]\langle P\rangle = SWITCH[a]\langle P\rangle$$

The class template for Fan is a further specialisation of an Alarm (see above). If offers a new initial action, followed by modified subsequent behaviour. Both strict and casual inheritance of the behaviour of Alarm are present in order to construct Fan.

$$Fan[a, b]\langle (P \,\Box\, Q) \,\Box\, R\rangle = Alarm[a]\langle (P \,\Box\, Q) \,\Box\, R\rangle \,\Box\, Alarm[b]\langle (P \,\Box\, Q) \,\Box\, R\rangle$$

By identifying further behaviour within SWITCH-based templates we can simplify Fan.

$$3\,WaySwitch[a, b, c]\langle P\rangle = SWITCH[a, b, c]\langle P\rangle$$

The signature for Fan reverts to a more simple form provided that an instance of 3WaySwitch is available at the time of a Fan template instantiation.

$$\begin{aligned} 3ws &= 3\,WaySwitch[off, eOff, timeout]\langle self\rangle \\ Fan[a, b]\langle P\rangle &= (a \rightarrow P \,\Box\, b \rightarrow P) \end{aligned}$$

Instantiation of each SWITCH-based template can be expressed as follows:

$$\begin{aligned} sw &= SWITCH[on]\langle off \rightarrow self\rangle \\ al &= Alarm[on]\langle off \rightarrow self \,\Box\, timeout \rightarrow self\rangle \\ wl &= SWITCH[on]\langle signal \rightarrow off \rightarrow self\rangle \\ ls &= SWITCH[on]\langle off \rightarrow self\rangle \\ fan &= Fan[on, eOn]\langle 3ws\rangle \end{aligned}$$

Note that *self* is used to denote recursion in a process definition rather than 'hard-coding' the process name at the end of the action sequence (the current method of defining recursive behaviour). Therefore, *self* defers the naming used in the recursion in order to point the recursion to the calling process rather than the local 'owner' process. For more details of self in relation to process algebra notation the reader is referred to [2] and [3].

1.4 Further Reuse within the ECS

From the definition of Alarm above the reader can recognise that an instantiation of Alarm is complex. Alarm can be further simplified by encapsulating its subsequent behaviour into another shared process template, like 3WaySwitch.

$$2\,WaySwitch[a, b]\langle P\rangle = SWITCH[a, b]\langle P\rangle$$

Now the 3WaySwitch can be redefined, inheriting the behaviour of the 2WaySwitch.

$$3\,WaySwitch[a, b, c]\langle P\rangle = 2\,WaySwitch[a, b]\langle P\rangle \,\Box\, SWITCH[c]\langle P\rangle$$

A transitive link exists from SWITCH to each of its templates with varying degrees of reuse and complexity. Any component of the ECS that is formed from the behaviour of a SWITCH template can be built entirely from successive constructions of SWITCH.

Alarm and Fan are now redefined using the 2WaySwitch template.

$$
\begin{aligned}
2ws &= 2\,WaySwitch[off, timeout]\langle self\rangle \\
3ws &= 3\,WaySwitch[off, timeout, eOff]\langle self\rangle \\
al &= Alarm[on]\langle 2ws\rangle \\
fan &= Fan[on, eOn]\langle 3ws\rangle
\end{aligned}
$$

With the successive construction of complex components from simple ones the flexibility of the APS model is apparent.

2 Applying CSP Sequential Composition to the ECS

Using the given notation for CSP, taken from Tony Hoare's original book [1], we can attempt similar definition of the ECS. The results however are not flexible enough for casual inheritance to be attempted elegantly. Further modification of some of the following processes requires more from the CSP notation than it can originally give, hence the development of the abstract process signature (APS) notation in the previous sections.

$$
\begin{aligned}
P &= (on \rightarrow Skip) \\
Q &= (off \rightarrow Skip) \\
SWITCH_1 &= P;\ Q \\
SWITCH &= {}^*SWITCH_1
\end{aligned}
$$

Recursion is defined in SWITCH using the following rule, from [1, p.172]:

$$
\begin{aligned}
{}^*P &= \mu X.(P.X) = P;\ P;\ P;\ \ldots, \\
\alpha({}^*P) &= \alpha P - \{\checkmark\}
\end{aligned}
$$

To make casual inheritance possible SWITCH is defined as the sequential composition of two processes. Additions to Q allow new behaviour to be inserted into the centre of SWITCH. If SWITCH were defined as $(on \rightarrow (off \rightarrow self))$ then no alteration of the events occurring after the initial *on* event would be possible, only extra initial events.

$$
\begin{aligned}
Alarm_1 &= P;\ R \\
&\quad R = (Q \,\Box\, timeout \rightarrow Skip) \\
Alarm &= {}^*Alarm_1
\end{aligned}
$$

Inheritance in Alarm is the extension to the behaviour offered by the events that occur after the initial *on* event.

$$
\begin{aligned}
Fan_1 &= S;\ T \\
&\quad S = (P \,\Box\, eOn \rightarrow Skip) \\
&\quad T = (R \,\Box\, eOff \rightarrow Skip) \\
Fan &= {}^*Fan_1
\end{aligned}
$$

Transitive inheritance is evident in Fan as the extended behaviour of T uses the already extended behaviour of R. Therefore, buried within the behaviour of Fan is the behaviour of Alarm and buried within the behaviour of Alarm is the behaviour of SWITCH.

$$
\begin{aligned}
WarningLight_1 &= P;\ signal \rightarrow Skip;\ Q \\
WarningLight &= {}^*WarningLight_1
\end{aligned}
$$

The WarningLight template is created by simply inserting a new action in the sequence of existing actions for SWITCH. Casual inheritance is identified as being present in WarningLight. LightSwitch, as defined previously, is a simple copy of a SWITCH template.

3 Alternative Redefinable Process Signatures

The CSP specification of the ECS SWITCH template can also be attempted using different constructs from the CSP notation and assumptions regarding primed variables. Consider this second (alternate) strictly CSP specification:

$$
\begin{aligned}
s &= \{on, off\} \\
SWITCH &= (on \rightarrow Q) \\
Q &= (off \rightarrow self) \\
Alarm &= SWITCH \\
Q' &= Q \,\Box\, (timeout \rightarrow self) \\
Fan &= Alarm \,\Box\, (eOn \rightarrow (Q'')) \\
Q'' &= Q' \,\Box\, (eOff \rightarrow self) \\
WarningLight &= SWITCH \\
Q''' &= (signal \rightarrow Q)
\end{aligned}
$$

$$LightSwitch \quad = \quad SWITCH$$

A mathematical treatment of primed variables is assumed. Q' becomes Q immediately after the expression $Q' = Q$. Prime is used to show values before and after an assignment.

Note that the contents of Q are extended to offer more choice of behaviour. The initial event *on* must occur first except in the case of Fan where an alternative (external) *eOn* event is offered, hence the need to redefine the behaviour of Q as occurs after the initial event. Attempting to alter Q as an internal part of SWITCH would require casual inheritance which is a more complex procedure and not a formal operation within CSP.

Also note that *self* is again used to maintain the correct return address after invocation by an inheriting process.

4 Conclusions

The introduction of abstract process signature theory to CSP has permitted the generalisation of process specification such that any degree of complexity can be modelled as a set of simplifying stages. Each stage building upon the work of the former. Inheritance is the key to this reuse. Casual inheritance begin a more distruptive form of inheritance than strict inheritance due to its insertion of actions into an existing action sequence which invalidates behavioural compatibility.

Only a subset of the ECS case study was discussed in this paper; namely SWITCH and its derivatives. However, our restricted look at the behaviour of the ECS gave enough examples of reuse to highlight the important issues of reuse, inheritance and process definition.

Acknowledgments

I am grateful to David Smith for continued support regarding parameterised processes, a concept which has continually crept into our discussions over the past few years and has now, finally, been put down in writing.

References

[1] C.A.R. Hoare. *Communicating Sequential Processes.* Prentice-Hall, London, 1985.

[2] S. Rudkin. Inheritance in LOTOS. In K.R. Parker and G.A. Rose, editors, *Formal Description Techniques*, volume VI, pages 409–423. Elsevier Science Publishers B.V: North Holland, 1992.

[3] P.N. Taylor. Concrete examples (using CSP) of process algebra templates and their children. Technical Report TR-280, Department of Computer Science, Faculty of Information Sciences, University of Hertfordshire, Hatfield, U.K. AL10 9AB, April 1997.