Distributed Resource Management Using Active Supervisory Predicate Control *

Alex I. Tomlinson and **Greg M. Hoagland** and **Vijay K. Garg**
Department of Electrical and Computer Engineering
The University of Texas at Austin
Austin, Texas 78712

Abstract

We model the problem of Distributed Resource Management using supervisory predicate control, and develop a supervisor to manage the resources according to a desirable predicate. The desirable predicate states that resource requests (uncontrollable event) must be serviced according to some cost constraint. Traditional methods for developing the supervisor fail because an unconstrained sequence of uncontrollable events will always lead to a state in which the desired predicate does not hold, hence the controllable predicate would be $false$. We place a limit on the number of consecutive uncontrollable events for which cost-effective service is guaranteed, and define a dynamic controllable predicate. The dynamic controllable predicate enables the plant to utilize more of its state space than a static controllable predicate would allow. A dynamic controllable predicate requires the supervisor to actively manipulate the state of the plant using $forced$ events in addition to the usual task of resticting controllable events.

1 Introduction

Considerably more research effort has focused on supervisory control theory [?, ?, ?], than on the applicaiton of this theory[?, ?]. In this paper we present the problem of Distributed Resource Management (DRM) and apply supervisory control theory to model DRM. A distributed resource is one in which the pool of available resources is not centralized, but dispersed throughout a distributed system. Resource requests are also distributed; they originate from various points in the distributed system. A client's resource request is satisfied either through remote interaction between the client and the resource, or by relocating the resource to the client's location. There is a cost associated with both methods, which creates new management challenges. The manager must decide where to keep available resources and which resources to allocate when a resource is requested. We apply supervisory control theory to design a supervisor which meets these management challenges.

Supervisory control can be loosely defined as the control of a discrete event system in order to restrict the system to some *desired predicate*. The discrete event system being controlled is referred to as the *plant*. The plant is driven by *events* and produces some form of output that the supervisor monitors. The supervisor disables certain events at the appropriate time to maintain the desired predicate. The event set is partioned into controllable, uncontrollable and forced events. A supervisor can prevent the occurence of a *controllable* event by disabling it; an *uncontrollable* event cannot be disabled. A *forced* event [?] is one which the supervisor commands. In essence, the supervisor creates a feedback loop to control the plant.

The desired predicate of the DRM plant is to guarantee service within a specified cost. The resources in the DRM problem are located at various nodes within the distributed system. A controllable event in DRM is a request to relocate a resource. The supervisor manages the locations of the resources by disabling or forcing the movement of resources. The resources must be managed in such a way that cost-effective service can be guaranteed. Traditional methods of developing a supervisor[?] determine a static controllable predicate and then restrict the plant to states where the controllable predicate holds. This approach fails with the DRM plant because the controllable predicate would be $false$. These problems can be supervised by placing a limit on the number of number of consecutive resource requests and using a dynamic controllable predicate. The dynamic controllable predicate allows the DRM plant to utilize a larger portion of its state space.

In section ?? we review the state machine model, predicate transformers, preconditions and controllability. In section ?? we present the supervisory pred-

This research was partially funded by the National Science Foundation through grant NSF-CCR-9110605 and by Microelectronics Computer Development Fellowships

icate control problem as introduced in [?]. We describe the DRM problem in section ??, and model it as a supervisory control problem in section ??. In section ?? we specify the control invariant and develop a supervisor, and in section ?? we outline a proof of the correctness of our supervisor.

2 Background

The choice of a plant model depends on the characteristics of the system being modeled. We will use a variant of the state machine model known as the predicate transformer model[?] because it concisely represents systems with large state spaces.

2.1 The State Machine Model

The plant is modeled with a state machine denoted by the tuple $G = (X, \Sigma, \delta, x_0)$; where X denotes the state set; Σ denotes the finite event set; $\delta : X \times \Sigma \rightarrow X$ denotes the partial state transition function and $x_0 \in X$ denotes the initial state. The event set Σ is partitioned into controllable events Σ_c, uncontrollable events Σ_u, and forced events Σ_f. The supervisor for the plant G is specified by a control map $m : X \rightarrow 2^{\Sigma}$. An event $\sigma \in \Sigma$ is enabled in state x if and only if $\sigma \in m(x)$, otherwise σ is disabled. The supervised plant induced by the control map m is described by the state machine $G_s = (X, \Sigma, \delta_s, x_0)$, where

$$\delta_s(x,\sigma) = \begin{cases} \delta(x,\sigma) & \text{if } \sigma \in m(x) \\ \text{undefined} & \text{otherwise} \end{cases}$$

In the predicate transform variant, the state space is modeled by plant variables; states are modeled by predicates on the plant variables; and events are modeled by predicate transformers.

2.2 Predicates and Predicate Transformers

We use $\mathcal{P}$ to denote the collection of predicates defined on the state set X. A predicate $P \in \mathcal{P}$ is a boolean map $P : X \rightarrow \{true, false\}$. There is a one to one mapping between subsets of the state space and predicates in the collection of predicates. Thus for notational convenience, we write $x \in X_A \Leftrightarrow P_A(x)$, where $X_A \subseteq X$ and $P_A \in \mathcal{P}$. For example, if we define $X_{x_0} = \{x_0\}$, then P_{x_0} corresponds to $\{x_0\}$, the initial state of the plant.

Often we will write "an event leaves R" to refer to an event that takes the plant from a state where R holds to a state where $\neg R$ holds. Likewise, "an event enters R" refers to an event that takes the plant from a state where $\neg R$ holds to state where R holds.

The set of predicates in $\mathcal{P}$ is partially ordered by the relation $\preceq$, defined as: $P_1 \preceq P_2 \Leftrightarrow \forall x \in X : P_1(x) \Rightarrow P_2(x)$. If $P_1 \preceq P_2$, we say that P_1 is *stronger* than P_2.

Let $\mathcal{F}$ denote the collection of all *predicates transformers*, i.e. if $f \in \mathcal{F}$, then $f : \mathcal{P} \rightarrow \mathcal{P}$. Since predicate transformers provide mappings from one set of states to another, they are used to model event occurrences, which take the plant from one state to another.

2.3 Preconditions and Postconditions

Consider an an event $\sigma \in \Sigma$, and an intitial set of states $X_A \subseteq X$ (where predicate P_A holds). This event will take the plant to a final state depending on the initial state. Let X_B be smallest set of states containing all possible final states (where predicate P_B holds). In this scenario, P_B is said to be the *strongest postcondition* of P_A and σ. Dijkstra and Scholten define the strongest postcondition P_B in [?]: P_B holds precisely in those final states for which there exist a computaion σ that starts in an initial state satisfying P_A. Formally, $sp : \mathcal{P} \times \Sigma \rightarrow \mathcal{P}$, where:

$$sp(P_A, \sigma) = P_B, \text{ where}$$
$$X_B = \{x \in X \mid \exists y \in X_A : \delta(y, \sigma) = x \}$$

The dual of the strongest postcondition is the *weakest liberal precondition*[?]. $wlp(P_B, \sigma)$ holds precisely in those states for which each computation of σ results in a state that satisfies P_B, or for which σ is undefined.

$$wlp(P_B, \sigma) = P_A, \text{ where}$$
$$X_A = \{x \in X \mid \delta(x, \sigma) \in X_B \ \vee \ \neg\delta(x, \sigma)! \}$$

The weakest liberal precondition includes all states for which σ is undefined, while the *weakest precondition* includes none of the states for which σ is undefined.

$$wp(P_B, \sigma) = P_A, \text{ where } X_A = \{x \in X \mid \delta(x, \sigma) \in X_B \}$$

The above definitions of sp, wlp and wp are based on the state transition function δ. We use sp_s, wlp_s and wp_s to denote the respective functions induced by the supervised state transition function δ_s.

2.4 Controllability

A plant is *controllable* with respect to an initial state P_{x_0} and a desired predicate $R \in \mathcal{P}$ if and only if uncontrollable events will not leave R and all states in R are reachable from P_{x_0}.[?] Desired predicates that are not controllable must be strengthened so that the plant can be controlled. This strengtened predicate is called the *controllable predicate*.

3 Supervisory Predicate Control

The Supervisor Predicate Control Problem (SPCP) can be stated as follows[?]: Given a plant $G =$

(X, Σ, δ, x_0) and a desirable predicate R, create a static supervisor $m : X \rightarrow 2^{\Sigma}$ such that $\bigvee_{\sigma \in \Sigma} sp_s^*(\sigma, P_{x_0}) = R$, where sp_s^* denotes the disjunctive closure of sp_s, which is the strongest postcondition transformer induced by δ_s.

It is assumed that $P_{x_0} \preceq R$ (the initial state satisfies the desirable predicate), otherwise SPCP is not solvable. If G is controllable with respect to R, then no uncontrollable events will leave R. In this case the solution is simple: disable all controllable events that leave R.

If G is not controllable with respect to R, the problem becomes more interesting. In this case we look for the minimally restrictive controllable predicate $R^{\uparrow} \preceq R$ and define a supervisor that restricts the plant to $R^{\uparrow}$. Such a supervisor is called a *minimally restrictive supervisor*.

Kumar et al. [?] present a formula for determining $R^{\uparrow}$. However, for many instantiations of SPCP, $R^{\uparrow} = false$. In these problems, an unconstrained sequence of uncontrollable events will always lead to the bad predicate space. Resource management problems fall into this category since an unlimited number of resource requests will deplete the system of all resources. In general, problems in which access to a limited resource must be controlled have no adequate solution when modeled in SPCP framework. We partition SPCP into the following problems:

Bounded SPCP: Instantiations of SPCP for which $R^{\uparrow} = false$.

Unbounded SPCP: Instantiations of SPCP for which $R^{\uparrow} \neq false$.

The term "bounded" reflects that the number of consecutive uncontrollable events must be bounded in order to maintain the desired predicate. Distributed resource management is one example of a Bounded SPCP problem. In the following sections we present a framework for controlling problems that fall into the Bounded SPCP category, using DRM as an example.

4 Distributed Resource Management

Distributed resources can be modeled by a weighted graph with tokens residing at the nodes. Each token represents one instance of a resource and has one of two states: *busy* or *avail*, corresponding to allocated and free resources. The tokens are free to roam around the graph, moving from node to node, subject to the approval of the resource manager. The manager also has the ability to forcibly relocate resources.

Resource requests occur at the nodes of the graph and are serviced by setting the state of a free resource to *busy* and relocating the resource to the requesting node. In order for a resource located at node v_1 to service a request originating at node v_2, there must exist a path from v_1 to v_2. Servicing a request has a cost, which we model by the the sum of the edge weights along the path used to service the request. When the resource is freed, its state is returned to *avail*.

An *incomplete request* is one in which the resource has not yet been freed. All resources are homogeneous, i.e, they are indistinguishable from each other except for their location and state. The goal of the manager is to maintain the locations of all the resources so that a given number of resource requests can be serviced within a specified cost. The DRM problem is formally specified below:

Graph: V =A Finite Set of Vertices
$E \subseteq V \times V$
$W : E \rightarrow \mathbb{R}$

Resources: R = Finite Set of Resources
$RS = \{avail, busy\}$
$L : R \rightarrow V$
$S : R \rightarrow RS$

Parameters: $Max \in \mathbb{N}$, $T \in \mathbb{R}$

Definitions: A *path* is a sequence of nodes $(v_1, \ldots, v_n)$ such that $\forall_{i:1 \leq i < n} : (v_i, v_{i+1}) \in E$
The *length* of a path $p = (v_1, \ldots, v_n)$ is: $length(p) = \sum_{i=1}^{n-1} W(v_i, v_{i+1})$
The *reachable set* $R_v \subseteq R$ of $v \in V$ is:
$R_v = \{r \in R \mid S(r) = avail \wedge$
$\exists$ path $p = (L(r), \ldots, v) : length(p) \leq T\}$

DRM Goal: Maintain $L{:}R \rightarrow V$ such that the following invariant holds:
$N < Max \Rightarrow \forall v \in V : R_v \| \, Max - N$,
N is the number of incomplete requests.

The goal is to maintain the positions of the resources such that all nodes can be serviced by at least $Max - N$ resources with a cost less than T. We write "a request can be serviced" as a short form for "a request can be serviced with a cost less than T".

5 State Machine Representation

We have described DRM from the token graph point of view; now we transform it into a plant $G = (X, \Sigma, \delta, x_0)$. The plant will formally define the events and serve as a basis for developing the supervisor.

$X = V^{\|R\|} \times RS^{\|R\|}$
$\Sigma = \Sigma_c \cup \Sigma_u \cup \Sigma_f$
$\Sigma_u = \{Request_v \mid v \in V\} \cup \{Free_r \mid r \in R\}$
$\Sigma_c = \{Move_r^v \mid r \in R, v \in V\}$
$\Sigma_f = \{FM_r^v \mid r \in R, v \in V\}$

The state space, X, consists of the locations and states of all the resources. $Request_v$ is a resource request at node v. $Free_r$ is the releasing of resource r. These events are both uncontrollable. $Move_r^v$ is a request by r to move to v, this is a controllable event. The forced event FM_r^v is a command issued by the supervisor to move r to v.

As mentioned above, we use the predicate transformer variant of the state machine representation.We define the set $\{r.location, r.state \mid r \in R\}$ to be the set of plant variables. There is a one to one mapping from the states characterized by the values of these variables to the states in the state space X. We describe the predicate transformers with the guarded command program in figure **??**. In the program, the keywords indicate the event type. The program is self-explanatory except for the following two points: pending forced events take precedence over other events, and $Recov_r$ is a sequence of forced events.

```
DO
   UNCONTROL  Request_v  → r := x | x ∈ R_v
                           r.location := v
                           r.state :=busy
   UNCONTROL  Free_r     → r.state :=avail
                           Recov_r
   CONTROL    Move_r^v   → r.location := v
   FORCED     FM_r^v     → r.location := v
OD
```

Figure 1: Predicate Transform Description

The task of the supervisor is to control $Move_r^v$ events and to define the recovery sequence, $Recov_r$, in order to ensure that the desired predicate is satisfied. Using the DRM Goal from section **??** we define the *desired predicate*: All nodes can be serviced by at least $Max - N$ resources with a cost less than T.

6 Bounded Supervisory Predicate Control of Distributed Resources

In the previous section we defined the plant and its desirable predicate. This section presents the framework for developing a supervisor that manages the events in the DRM problem. For the DRM problem, an unconstrained sequence of uncontrollable resource requests can always take the plant into a bad predicate because the system has a limited number of resources. The controllable predicate is therefore $false$ when using traditional supervisory control methods[**?**]. Placing an upper bound on the number of consecutive service requests that are guaranteed to be serviced within a certain cost makes it possible to find a controllable or desirable predicate that may not be $false$. However, an effect is that the controllable predicate becomes dynamic, dependent on the number of requests that are incomplete. Traditionally, the supervisor's job is to restrict controllable events that take the plant out of a static controllable predicate. With a dynamic controllable predicate, the supervisor must begin to take on the additional role of actively manipulating the state of the plant[**?**].

We start by defining a sequence of predicates which classify the states based on the number of requests for which service can be granted.

$$RP_0(x) \stackrel{\text{def}}{=} \exists v \in V : \neg\delta(x, Request_v)!$$

$$\forall n > 0 \quad : \quad RP_n(x) \stackrel{\text{def}}{=} \bigvee_{v \in V} wp(RP_{n-1}, Request_v)$$

By inspecting the program in figure **??**, it can be seen that $\delta(x, Request_v)$ is undefined if there exists a node v such that $R_v = \emptyset$. RP_0 holds when there exists a request event that cannot be serviced with a cost less than T. Hence, RP_n holds when there exists a sequence of $n+1$ requests such that at least one of the requests cannot be serviced with a cost less than T. Conversely, if the plant is in $\neg RP_n$ then for all sequences of $n+1$ service requests, it is possible to service each request with a cost less than T.

From the desirable predicate defined in the previous section, we define an equivalent *control invariant* that the supervisor must maintain: Given N incomplete requests, $\neg RP_{Max-N-1}$ should hold.

Initially $N = 0$, thus we assume that $P_{x_0} \preceq \neg RP_{Max-1}$. This means that $Max-1$ requests can be serviced. It is the responsibility of the supervisor to assure that the control invariant is satisfied. The supervisor accomplishes this using two methods: forcing recovery events and restricting controllable events.

For a given n, we define a *smart sequence* of forced moves, $F = (f_1, ..., f_k)$, where $f_i \in \Sigma_f$ for $1 \leq i \leq k$, to be one that satisfies: $sp(\neg RP_n, F) = \neg RP_{n+1}$. In figure **??**, $Recov_r$ is a sequence of forced moves; it must also be a smart sequence in order to maintain the control invariant.

Now we define the supervisor's role in relation to the event types which occur in the system. A controllable event will not change N, thus the controllable predicate will remain $\neg RP_{Max-N-1}$. The supervisor must disable all controllable events which leave $\neg RP_{Max-N-1}$. Using the weakest precondition defined in section **??**, we can determine which events enter a particular state. The predicate $wp(RP_{Max-N-1}, \sigma)$ holds when execution of σ results in a state where $RP_{Max-N-1}$ holds. Thus if $(wp(RP_{Max-N-1}, \sigma))(x) = true$, then σ should be

disabled in state x. The only controllable event in our example is $Move_r^v$, which transforms the state space with $r.location = v$, thus we define the control map m as follows:

$$m(x) \stackrel{\text{def}}{=} \Sigma_u \cup \{Move_r^v \mid (\neg wp(RP_{Max-N-1}, r.location = v))(x)\}$$

Recall that m is the set of *enabled* events, so wp is negated. Note that all uncontrollable events are included, so the supervisor doesn't try to disable uncontrollable events. Note also that the control map is a function of N, the number of incomplete requests, and x the current state of the plant.

We must consider $Free_r$ events in conjunction with forced events. When a resource has finished servicing a request, an uncontrollable $Free_r$ event is issued. Given N incomplete requests, $Free_r$ will decrease N to $N-1$. The controllable predicate changes from $\neg RP_{Max-N-1}$ to $\neg RP_{Max-N}$. To make sure the control invariant is satisfied it is necessary to couple a $Free_v$ event with a smart sequence of forced moves $Recov_r$. The supervisor chooses $Recov_r$ such that $sp(\neg RP_{Max-N-1}, Recov_r) = \neg RP_{Max-N}$.

7 Outline of Supervisor Proof

Theorem 1 states that a supervisor using the control map m satisfies the control invariant given in section **??**. We describe four supporting lemmas and then provide a simple proof of Theorem 1. For brevity, we use Req_v to represent the $Request_v$ event.

Lemma 1: $\forall n \geq 0 : RP_n \preceq RP_{n+1}$

$\neg RP_n$ holds when $n+1$ consecutive requests can be serviced. Lemma 1 claims the following: if the state of the plant is such that $n+2$ requests can be serviced, then $n+1$ requests can be serviced from the same state. This is easy to see once you realized that the statement of Lemma 1 is equivalent to $\neg RP_{n+1} \preceq \neg RP_n$ and use the interpretation of $\neg RP_n$ given in section **??**.

Lemma 2: $\forall n \geq 0, \forall v \in V : sp(\neg RP_{n+1}, Req_v) = \neg RP_n$

Lemma 2 claims that if a the plant is in a state that can service $n+2$ requests, and executes a service event, the resulting state will be able to service $n+1$ requests. This is a result of lemma 1 and the definition of the strongest postcondition function, $sp()$.

Lemma 3: $\neg RP_n(x) \Rightarrow$
$\forall seq = (Req_i, \ldots Req_k), k \leq n+1 : \delta(x, seq)!$

Lemma 3 claims that if $\neg RP_n$ holds in state x, then all sequences of $n+1$ (or less) consecutive request events can be serviced.

Lemma 4: $N < Max \Rightarrow \neg RP_{Max-N-1}$

Lemma 4 claims that if there are N incomplete transactions, and N is less than Max, then $\neg RP_{Max-N-1}$ holds. Theorem 1 proves the control invariant by a simple application of Lemmas 3 and 4.

Theorem 1 *$N < Max \Rightarrow Max - N$ requests can be serviced.*

Proof: By lemma 4, we know that $N < Max \Rightarrow \neg RP_{Max-n-1}$. And by lemma 3, we know that $\neg RP_{Max-n-1} \Rightarrow Max - N$ requests can be serviced. Therefore, $N < Max \Rightarrow Max - N$ requests can be serviced. ∎

We have outlined proof that a supervisor using the control map m satisfies the control invariant, which states that the plant should be able to service $Max - N$ requests if we have N incomplete transactions.

8 Applications of DRM

DRM is a good model for the problem police headquarters face when trying to manage patrol cars so that they can respond to 911 emergency calls quickly. The city is represented by a weighted graph in which the vertices represent different neighborhoods in the city, and the weighted edges represent the time it takes to travel between neighborhoods. Emergency calls are uncontrollable events. A patrol car moving from one neighborhood to another is a controllable event. Headquarters ordering a patrol car to move to a specific neighborhood is a forced event.

DRM can also model services in distributed computing system where the services must relocate to the node making the request. In this case the weighted graph is represented by the communication network. The vertices are the nodes on the network. The edge weights represent the cost of transferring the service over the link. The resources are the distributed services that float around the network. Although this type of distributed service computing model is not in widespread use today, as computer networks become more powerful and more widespread, this model may become more important.

9 Conclusion

We introduced the DRM problem as a useful method for modeling distributed resources. The DRM problem states that resource requests should be satisfied within a specified cost. Supervisory control provides a framework for modeling DRM and for determining a supervisor to manage the resources in such a way that the cost constraints are met. Traditional methods for determining a supervisor for the DRM plant failed because the controllable predicate was $false$.

We defined a new class of Supervisory Predicate Control Problems (SPCP) and labelled it Bounded-SPCP. DRM is an example of a Bounded-SPCP problem. We developed a supervisor that uses a dynamic controllable predicate to control the plant, which enables the plant to use more of its state space than a static controllable predicate.