

The Institute For Research In Cognitive Science

Subsumption Architecture and Discrete Event Systems: A Comparison

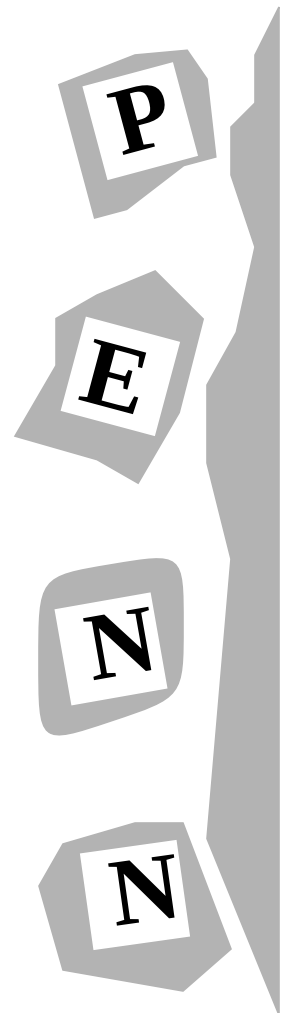
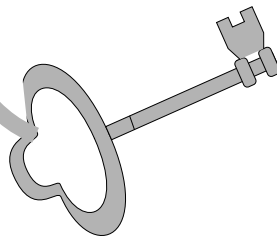
by

Luca Bogoni

**University of Pennsylvania
3401 Walnut Street, Suite 400C
Philadelphia, PA 19104-6228**

December 1994

Site of the NSF Science and Technology Center for
Research in Cognitive Science



Subsumption Architecture and Discrete Event Systems: A Comparison

Luca Bogoni

GRASP Laboratory
Department of Computer and Information Science
University of Pennsylvania, Philadelphia, PA 19104

December 17, 1993

Subsumption Architecture and Discrete Event Systems: A Comparison

Luca Bogoni

Abstract

In this paper we review Subsumption Architecture and Discrete Event Systems. These approaches present diverse methodologies for dealing with control of interactions. They often take diametrically opposite directions in addressing specific issues. Subsumption architecture expects limited knowledge of the environment, no explicit representation, limited reasoning capabilities and no centralized control. At the other extreme lies Discrete Event Systems, which require, at least in manufacturing and communication: a well-structured environment; explicit representations and models; and have limited reasoning capabilities and centralized control. Both offer benefits and limitations which should really be evaluated and traded off when attempting to build a system. However, combining aspects from these two approaches will not address and resolve all issues. We conclude that while both approaches are powerful there is more to intelligence than just behavior and control, and discuss the limitations and benefits entailed by both.

Contents

1	Introduction	1
2	Subsumption Architecture	3
2.1	Motivation, Philosophy and Principles	3
2.2	Levels and Layers	6
2.3	Task Achieving Behaviors: Formalization and Implementation	8
2.3.1	The Specification Language	9
2.3.2	Implemented System	12
3	Extension to the Basic Subsumption Model	14
3.1	Representation in Goal-Driven Behavior-Based Robots	14
3.1.1	Landmark Detection	15
3.1.2	Mapping Algorithm	16
3.1.3	Landmark Disambiguation	18
3.1.4	Path Planning	18
3.2	Hybrid Layered Architecture	19
3.2.1	Navigational Task	21
3.2.2	Experiments and Key Points	22
4	Discussion on Subsumption Architecture	23
4.1	Decomposition of Behaviors	23
4.2	The World as Its Own Model	24
4.3	Complexity and Scalability	25
4.4	Expressiveness of the Specification Language	26
4.5	Provability and Reliability of Behaviors	27
4.6	Future Directions in Subsumption Architecture	27

5	Discrete Event Dynamic Systems	29
5.1	Basic Formalization	29
5.2	Controlling Discrete Event Dynamic Systems	31
5.3	Observing the Behavior of the Plant	34
5.4	Review of The Formalism	35
6	Control of a Rapid Thermal Multiprocessor	37
6.1	Input and Output Interpretation	37
6.2	Logical versus Physical Plant	41
6.3	Refinement of the Supervisor Model	42
6.4	Synthesis of Supervisor	44
6.5	The Application System and Observations	45
7	Discussion on Discrete Event Dynamic Systems	46
8	Conclusions	48

1 Introduction

When describing an interaction with the environment we define actions, means of controlling their development and their effects either immediate or delayed. Additionally, we describe models of the environment in which actions are to take place, the characteristics of the agents participating, and the objects acted upon. In this paper we investigate two different philosophies for describing the development of tasks and the role that agents play in an environment.

The first one, subsumption architecture, is a behavior-based approach which takes a radical departure from the standard approach for defining interactions for navigating autonomous agents. It advocates a layered construction of a creature starting at very low, reactive behaviors and developing, in an evolutionary fashion, more complex behaviors. In this constructed hierarchy, more “evolved” layers have the capability of affecting the behavior of the lower level ones by overriding their outputs and inputs. It is named “behavior-based” since the subdivision of the roles played by the different layers is dependent on the behavior fulfilled. In each layer more than one behavior may be implemented and the contention between the different behaviors results in the emerging overall behavior of the creature. We investigate two approaches, beside the basic architecture. The first focuses on a distributed active representation for navigating. The second one proposes a departure from the basic architecture by introducing a hybrid system which uses subsumption architecture and symbolic reasoning capabilities.

The second group of papers focuses on an approach which originated from Control Theory. This method, named Discrete Event Dynamic System Theory, is based on formal language theory. It allows the specifications of interactions and their control. This is accomplished by introducing the notion of a supervisor whose role is that of controlling the development of a system. This control is expressed in the form of enabling and disabling possible transitions in the development of a task. This methodology has found its most successful applications in manufacturing environments and communication networks. The supervisor exerts its control on a plant (the system being controlled) and through a feedback loop monitors the developments of the plant. Only recently some researchers have begun to apply this methodology to other contexts to control, for instance, the behaviors of agents navigating in the environment.

While on first consideration the two approaches appear to deal with totally different areas, we consider each of the approaches and observe what each of them bring to the aspect of control of the interaction of an agent with the environment. In particular, we observe that the subdivision in terms of behaviors, as advocated by subsumption architecture, has allowed researchers to focus on different levels of interactions. Some of the interactions are best described at a reactive level and handled by subsumption method. Others, however, may require substantial supervision especially

to guarantee that the appropriate behavior is carried out.

Discrete Event Systems offer a formalism for expressing behaviors and developing control strategies even when conflicts arise. This aspect is rather loosely expressed in subsumption architecture in the form of object-like structures encapsulating a behavior. The interactions and the controls between the behaviors is not formalized and hence makes the approach infeasible for expressing complex systems. On the other hand, its closeness and simplicity of the reactive behaviors makes the developed agents capable of avoiding obstacles and navigating in the environment.

The paper is organized in three major parts. The first addresses subsumption architecture; the second one discusses discrete event dynamic systems; the third one presents a comparison between the two approaches. In each of the first two parts: the particular area is introduced, one or more reviews of papers are presented, and an intra-comparison is presented.

Specifically, subsumption architecture's philosophy and principles are introduced in section 2. These are investigated by focusing on the original paper by [Brooks, 1986] and augmenting the various points with further analysis presented in papers [Brooks, 1991b; Brooks, 1991c; Brooks, 1991d]. Two other papers addressing related examples, in which subsumption architecture is employed, are analyzed in section 3. The first one, [Mataric, 1992b] presents an interesting approach employing a distributed representation. The second one, [Connell, 1992], introduces a hybrid approach in which subsumption architecture and a symbolic reasoning system are interacting. Section 4 presents a discussion between the different approaches adopted in the papers presenting subsumption architecture. Discrete Event Systems are introduced in section 5. In it a brief overview and introduction to a large body of research is given by looking at [Ramadge and Wonham, 1989] and additionally [Ramadge and Wonham, 1987; Wonham and Ramadge, 1987]. In section 6 an application for a rapid thermal multiprocessor system is presented in [Balemi *et al.*, 1991]. A brief discussion on discrete event systems is presented in section 7. The two approaches are, finally, compared in section 8.

2 Subsumption Architecture

Subsumption architecture or behavior-based architecture came into being as the result of a new movement for studying intelligence bottom up, concentrating on physical systems, situated in the world, and autonomously carrying out tasks. In proposing a new architecture for constructing a robotic system, Brooks attacks the problem from a complete different angle, [Brooks, 1986]. This approach is based on engineering from first principles and finds inspiration on biological systems. However, the approach goes beyond the simple proposal of a new paradigm for constructing a system. In fact, Brooks proposes a rather iconoclastic view against the traditional *modus operandi* held by most Artificial Intelligence and Computer Vision researchers.

This section examines the motivation, philosophical aspects and implications of the behavior-based approach, and the basic principles underlying it. Subsequently, the formalism and implementation of the robotic system as described in [Brooks, 1986] are reviewed.

2.1 Motivation, Philosophy and Principles

The implementation of subsumption architecture is first presented in [Brooks, 1986]. The proposed goal is that constructing complete systems bottom-up, that are adequate for navigating in an unstructured environment and which will exhibit intelligence. The driving force can be best expressed by the following two key points:

- No Explicit Representation. Representation should be distributed in the individual behaviors.
- No Explicit Reasoning. Reasoning should be emergent from the different behaviors.

These two points and their implications are the topic of two separate papers, [Brooks, 1991c; Brooks, 1991d], in which Brooks argues for the behavior-based approach.

In [Brooks, 1991c], the author puts forward the thesis that the current status of Artificial Intelligence was greatly influenced by the aspects of computer architecture and conversely that the Von Neumann model of computation has lead Artificial Intelligence in particular directions. The underlying conclusion is that a new model based on engineering principles and based on Biology should be developed bottom-up. Furthermore, reasoning should be implicit and emergent as result of the interaction of the various behaviors.

He points out that the current conventions can not account for many aspects of what goes into intelligence. Most of these conventions on how reasoning and thought are carried out, such as planning and problem solving, have been obtained by introspection and are based on operations, such

as searches, performed on abstractions of how inputs and output should be handled and knowledge represented. Questioning the validity of these conventions puts the whole field of Artificial Intelligence on trial. He argues that the initial approaches, constrained by the technology in the early days of computing, have become to be adopted, over time, as principles. Hence, what was initially a unique field, attempting to describe intelligence and intelligent behavior, parted ways with the initial goals and developed into separate subfields – search, pattern recognition, learning, planning and induction¹. This partitioning allowed AI to focus on search strategies with the assumptions that, once the solution to the static environment problems would have been accomplished, it would have been simple to move in a more dynamic one. These expectations have not materialized.

Brooks, furthermore, argues that the solutions offered by Artificial Intelligence in general bear no resemblance at all to how biological systems work. Understanding biological intelligence through ethology, psychology and neuro-science, on the other hand, might provide constraints on how higher thought in human could be organized. While Biology provides motivations for an evolutionary approach to building systems based on a hierarchy of behaviors, one must be aware that evolution was not an optimizer in design, rather that certain aspects have been patched together and adapted. In fact, some of the solutions taken might, in fact, be sub-optimal and even some of the structures are vestigial and emulation may be a distraction. This consideration on evolution and the adaptability to an environment has suggested that intelligence could be interpreted as the product of behaviors conditioned on the complexity of the environment. Namely, the reactions of a creature in an environment and the attribution of its ability to reason could be imputed by an external observer noticing the behavior emerging in a complex environment rather than the creature's actual ability to coordinate and consciously interact. [Brooks, 1991c] states:

It is the observer of the Creature who imputes a central representation or a central control. The Creature itself has none: it is a collection of competing behaviors. Out of the local chaos of their interaction there emerges, in the eye of the observer, a coherent pattern of behaviors.

These observations, characterizing reactive type behavior, appear to find support in lower level creatures but, upon moving up in the evolutionary scale, it is not clear that such a strong case may be made for emergence. There are clear instances in which humans possess reactive type behavior and no conscious control is required especially in motor and stability control. Yet, there is a strong dissent on such behavior-type extensions governing actual reasoning, planning and abstraction.

Artificial Intelligence was founded around representation and has shared with Computer Vision, at least in its inception, the assumption that the purpose of Computer Vision was that of

¹This subdivision is due to Minsky.

reconstructing the external world as a 3D model. That view of the role of vision has since changed and many researchers tend to agree that reconstruction may not only be impossible, but also not necessary. This consensus has manifested in the origination of the Active Vision and the Purposive Vision paradigms, [Bajcsy, 1985; Bajcsy, 1988; Aloimonos *et al.*, 1988; Aloimonos, 1990]. Yet, what Brooks puts forward in [Brooks, 1991d] is more radical.

The thesis on the role of representation is that the world should be its own model and that representation is the wrong unit of abstraction. If intelligence is approached in an incremental manner, through perception and action, the reliance on representation disappears.

Representation has been used as means of interfacing between otherwise isolated modules. Intelligent system should be decomposed in parallel and independent activity producers. Intelligence is too complex and, since little is understood, decomposing it into representational units is not adequate. Brooks suggests that we still need to learn about the underlying mechanisms. As already mentioned above, he stresses the need to build complete systems, at each step of the way, capable of interacting with their environment thus ensuring validity of the overall system behavior. The interaction with the environment should be based on real sensing and real actions.

Abstraction is seen as a dangerous weapon which has lead to obsession on search algorithms rather than focusing on the actual world it abstracts from. In particular, abstraction has been used to factor out all aspects of perception and motor skills. In fact, for most AI planning system it is the experimenter who abstracts away most of the details to form a simple description in terms of atomic concepts for the system to manipulate. Similarly to what was pointed out about reasoning, Brooks suggests that introspective descriptions of internal representation may be quite different from what human use. Representation should relate more directly actions and perceptions.

The key ideas underlying the subsumption style of Artificial Intelligence can be characterized by the following properties, [Brooks, 1991c]:

SITUATEDNESS characterizing robots located in the world and hence concerned not with abstract descriptions but with *here* and *now* and with the ability to directly influence the behavior of the system.

EMBODIMENT identifying agents as having bodies and experiencing the world directly and hence having immediate feedback on their sensations.

INTELLIGENCE describable by an external observer and whose limitations are not intrinsic of the computational engine used. Rather is originated from (a) situations in which the robot finds itself in the world (b) the signal transformation by the sensors (c) physical coupling of the robot with the world.

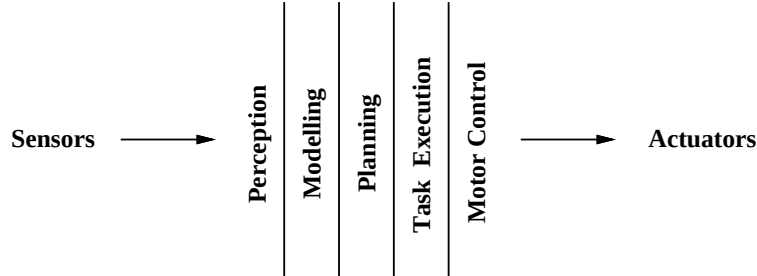


Figure 1: Traditional horizontal decomposition (adapted from [Brooks, 1986]).

EMERGENCE of intelligence from the interaction with the environment and sometime from indirect interaction between the components.

2.2 Levels and Layers

In the traditional approach, problems are generally decomposed into subparts (analysis); each of parts are solved; and then the individual solutions are composed (synthesis) to yield the final answer (see Figure 1). As a consequence of the new philosophy of portraying an incremental description of intelligence as levels of competence, the analysis of a problem, such as navigation in an environment, and the methodology for its solution take a whole different perspective. Namely, the problem is decomposed in different levels so that instead of a horizontal decomposition, a vertical decomposition is proposed. It allows a low level to run without waiting for the others to be developed. In order to accomplish this, a shift of focus from the sense-model-plan-act to a layered paradigm is necessary. The two methodologies can be characterized as a decomposition by function and a decomposition by activity:

- *Decomposition by Function.* The traditional view that perceptual modules deliver a symbolic description of the world. The action modules take a symbolic description of desired actions and make sure that they happen in the world. In this approach the central system behaves as a symbolic information processor (see Figure 1).
- *Decomposition by Activity* is orthogonal in that it slices the organization into activity producing subsystems. This approach allows an incremental description of intelligence. (see Figure 2).

In [Brooks, 1986] the author points out the requirements for the layers to operate at increasing level of competence and to be composed of asynchronous modules communicating over low-bandwidth channels. Each layer should identify an instance of a simple computational machine.



Figure 2: Levels of competence vertical decomposition: *subsumption architecture*(adapted from [Brooks, 1986]).

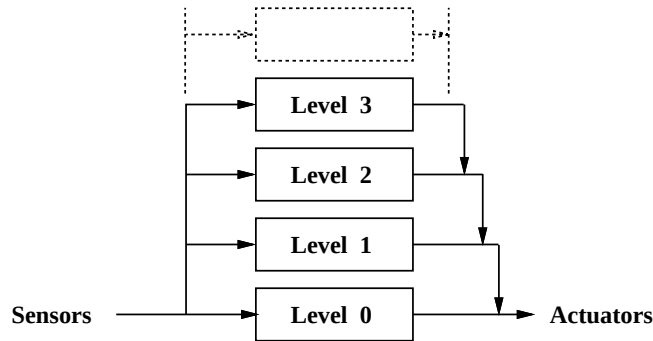


Figure 3: subsumption architecture: higher levels' subsumption.

High-level layers should subsume roles of the low-level by suppressing their outputs and low-level layers should continue to function independently from the addition of high-level layers.

This hierarchy of control in the layers is opaque to lower layers which continue to operate; only their output is possibly suppressed by the higher levels ones, (see Figure 3). This is the characteristic which has warranted the name “subsumption” for the proposed architecture.

Each of the individual layers has an associated task. Lower level layers are reactive and basic and higher level ones are able to alter the action of the lower levels. Multiple goals may coexist and some may be in conflict. The arbitration of these conflicts is resolved through the priority of the levels and their ability to subsume lower level ones. By letting each of the layers have independent goals, then the world can be used as its own model, continuously matching the preconditions of each goal to the real world. As pointed out, the representation is to be distributed between the different layers and the frequent interaction with the environment will provide the adequate feedback for the

action. By distributing the representation, individual layers extract only those aspects which they find relevant. Hence, each layer exhibits task-achieving behaviors from which emerges the behavior of the “creature”.

Robustness in the system can be achieved by developing layers from the lower, more reactive, to the higher ones, incrementally and by debugging each one thoroughly before proceeding with the next one in the hierarchy. Additionally, since changes of the world have less chances of being reflected in all the individual aspects, the overall behavior is going to be more robust by being distributed and depending on multiple sensors at different layers. The support of the independence and robustness derived is taken from biological systems. In the case of pigeons, for instance, who use both sight and reference to earth’s magnetic field for navigation, the sensing capabilities are not combined rather they employed selectively depending on the environmental conditions and operational level of the sensor subsystem. What Brooks is advocating is not sensor fusion but emergent, and hence implicit, behavior fusion.

According to the proposed architecture, once a behavior has been built, new ones may be added by guaranteeing that the appropriate type of subsumption occur. Since the goal of the design is that of constructing autonomous robots, any added behavior should be provided with independent source of power as not to impair the existing system. This requirement is identified as extensibility.

2.3 Task Achieving Behaviors: Formalization and Implementation

Having introduced the underlying philosophy and concepts which characterize subsumption architecture, the current section examines the formalization and the implementation presented of the paper in which Brooks introduced this approach [Brooks, 1986]. The goal of the implementation is that of constructing a mobile robot capable of wandering around unconstrained in laboratory areas and computer machine-rooms, eventually building maps of the surroundings and performing simple tasks.

The design decision for building a robot are based on the following principles:

1. Complex behavior is not necessarily the result of complex control system. Such behavior is the result of the robot interacting with a complex environment. It is the observer who imputes complexity to the system. Such complexity, however, is not necessarily in the design.
2. Simplicity in the design. When designing, the interface should not be too complex with respect to the role of the module designed. No component or collections of components should solve an ill-conditioned or unstable problem. Such a specialized module tends not to be too robust.

3. The robot should have a Map-making capability even when blue prints of environment are available. This allows the robot not to rely entirely on exact measurements matched to the map but to be able to adapt to new environment or environments in which unknown objects or obstacles are present.
4. The world to be navigating in should be three-dimensional. This aspect focuses on the requirement of dealing with a real environment rather than with an abstract or projected version.
5. Relational or qualitative maps should be employed rather than absolute coordinates. A map based on absolute coordinates is more prone to error. This leads to a change in the design space of the perception systems which become, in this way, more qualitative.
6. The world in which the robot is to interact should not be a special, artificial environment.
7. Use sonar for low level and vision for higher level purpose layers.
8. Self calibration should be built in and applied in all processing steps.
9. The robots should be self sustaining for long periods of time.

The unanswered question remains on the selection of the appropriate behaviors for expressing the task. In [Mataric, 1992a] design strategies for building an agent are given in terms of behaviors. These involve the identification of the basic set of reflexes for survival in the dynamic unstructured environment and the bottom-up design incorporating top-down constraints. In order to describe the process of using task-specific constraints to generate the behaviors, the following heuristic should be adopted. Specifically, one should specify the desired behavior(s) in qualitative terms, the behavior in terms of the actions in the observer space, and the actions in terms of the robots' effector and actuators. Once the behaviors have been identified, these need to be expressed using a formalism.

2.3.1 The Specification Language

Having outlined the principles for designing the robot, we now examine the formalism for the specification language. This is captured by two aspects: the internal structure of a layer and the communication between them.

The internal structure of a module is expressed in terms of a finite machine augmented with some instance variables, capable of holding Lisp-like data structures. A particular module (see Figure 4) is expressed in terms of inputs and outputs and a reset line to bring the state of the

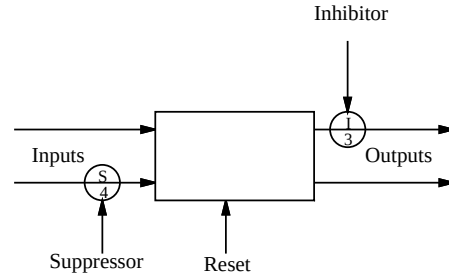


Figure 4: A black box description of a module with, output and input lines, reset line, and suppressor (acting on inputs) and inhibitor lines (adapted from [Brooks, 1986]).

```

(defmodule <module name> <level number>
  inputs (inputs wires)
  outputs (outputs wires)
  states
    (nil <condition> <action or side-effect>)
    (<state 0> ...)

    (<state N> ...))

```

Figure 5: The syntax for a module, defined in Lisp like structure. **nil** defines the default state the module is in at either startup or reset.

module to a known value. Suppressor lines control inputs and inhibitors lines affect outputs. The states characterizing a machine can be one of the following types:

OUTPUT. The output message is expressed as a function of module's input buffer and instance variables, sent to the output line. Once this occurs a new state is entered.

SIDE EFFECT. Occurs when an instance variable is set to a new value as result of functional computation from input buffers and variables. This causes a transition to a new state.

CONDITIONAL DISPATCH. A predicate on the module's instance variables and input values. Based on the result of computation one of two states is entered.

EVENT DISPATCH. Conditions and states to branch to are monitored until an event is true. Events are in combinations of arrivals on input lines and expirations of time delays.

Connectors:

```
(defwire <level number>
  (<first module> <parameter passed>)
  (<second module> <parameter received>)
)
```

Inhibitor or Suppressor:

```
(defwire <level number>
  (<module label> <parameter passed>)
  (modifier type <condition of control>)
)
```

Figure 6: Connectors between module defined as *wires*. The *modifier type* describes whether it is a inhibitor, suppressor or reset type.

The communication between modules is specified in terms of *wires* between the different modules. Semantically, they may be connectors between components or behaviors within a module, or inhibitors and suppressors, see Figure 6. In both cases the level number specifies the layer in which the module should belong to. Both the above module definition and wire definitions can be easily seen as separate processes or objects in an object-oriented language².

Messages are communicated through wires to buffers in the destination modules: the system allows for no shared memory. Such messages, intended to be of minimal content (usually few bits), may overwrite previous messages since the destination buffers can only contain one message at a time. This aspect is part of the asynchronicity and the real-time property built into the system. In the system, as structured, since the interaction with the environment is continuous and the signal processing done in real-time, the loss of a message is not crucial and the overwriting of a message is simply interpreted as an updating. Thus, the destination module has in its buffer either no input or the most recent and, since the system is meant to be reactive, most relevant information on the current situation.

²[Mataric, 1992b] and [Connell, 1992] present simple examples of behaviors. [Abelson *et al.*, 1985] provides examples of similar data abstractions in flavor to the ones presented above for defining wires connections between different components in an electrical circuit system.

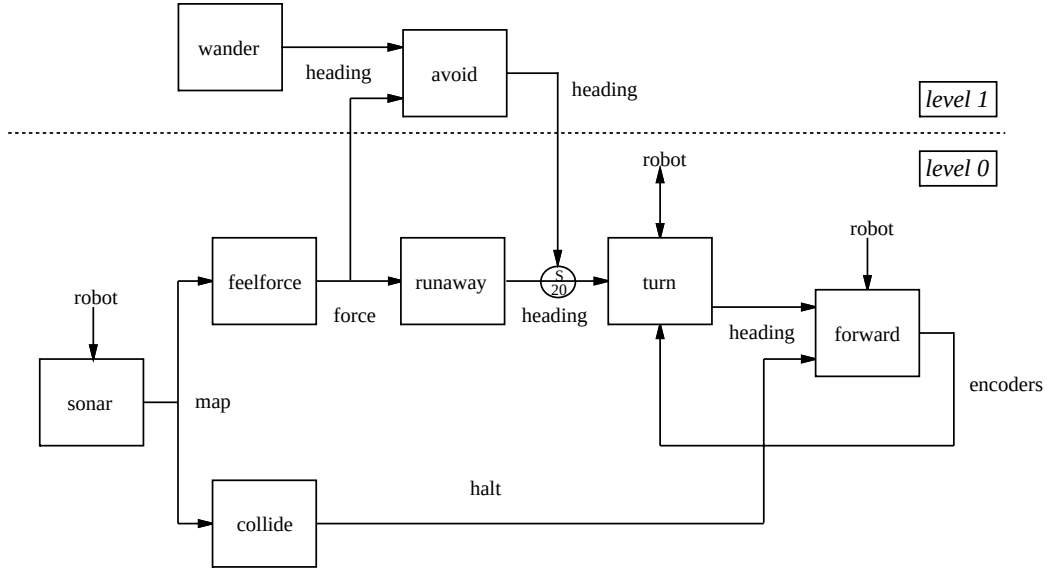


Figure 7: Schema of interconnections describing the control of level 0 and level 1 (adapted from [Brooks, 1986]).

2.3.2 Implemented System

The interaction between three lower levels was implemented while the higher ones were simulated. Figure 7 shows the connections between level 0 (Avoidance) and level 1 (Wandering). These can be described as:

Zero Level guarantees that the robot does not get into contact with other objects. Essentially endowed with very coarsely calibrated sonar sensors, this level causes the robot to flee approaching obstacles.

First Level wanders in the environment with a little planning for avoiding obstacles. It relies on the lower level for avoiding obstacles.

Second Level, not shown in Figure 7, locates interesting places to wander to. These destinations, free corridors, are selected using stereo. This level inhibits wandering when it needs to take pictures to decide where to head.

We notice that, while in the principles of the design one layer should have no knowledge about other layers above, in this very low level, provision is made to receive communication for the direction of motion or turning. As we shall observe, the distinction between layers is not so clear cut at all times and higher level layers may have “special” receiving modules or components within one of the lower layers, rather than just being able to affect (suppress or inhibit) the communication lines.

The simulation presented for the non-implemented levels appears promising and can be useful in designing stage to illustrate aspects which might otherwise require a redesign at a later state. The implemented system shown, however, even though built of only three layers is quite impressive for it achieves the specified goals of navigating in the environments. The system constructed has shown that the decomposition in terms of behaviors is a viable one. We will continue the discussion of the subsumption architecture in section 4 comparing the Brooks' approach with others discussed next.

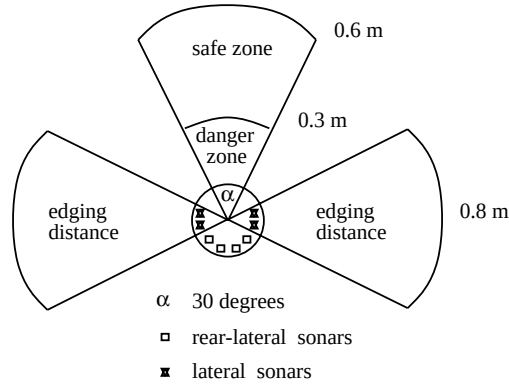


Figure 8: Perceptual Zones around the Robot (adapted from [Mataric, 1992b]).

3 Extension to the Basic Subsumption Model

In this section we review the work by [Mataric, 1992b] and by [Connell, 1989].

3.1 Representation in Goal-Driven Behavior-Based Robots

An approach integrating a distributed map representation into a reactive, subsumption-based mobile robot is presented in [Mataric, 1992b]. The environment in which the robot is to navigate is an office area. The architecture presents an alternative to hybrid systems, discussed in section 3.2, which separates reactive and traditional planning parts of the control system. In this case, no distinction is made between control and map. The incrementally designed behaviors developed are: collision avoidance, dynamic landmark detection, map construction and maintenance, and path planning. The topological representation uses primitives suited to the robot's sensors and its navigation behavior. The map, unlike traditional centralized maps, can be characterized as a distributed collection of behaviors responding to the various landmarks, allowing constant-time localization and linear-time planning. The approach presented is qualitative and tolerant of sensor inaccuracies, unexpected obstacles and course changes.

The behaviors can be distinguished as having three competences built into a homogeneous behavior-based representation. This include *basic navigation* (obstacle avoidance and boundary tracing), *landmark detection*, and *map-related computation* (map construction, update and path planning). Basic navigation was designed to facilitate map construction and landmark recognition. Its behavioral description and formalization is quite similar to the one described in [Brooks, 1986]. The sensors employed in the robot are sonars and a compass. Figure 8 shows how the sonar sensors are located in the robot and the qualitative quantizations of the sensing in the perimeter. The

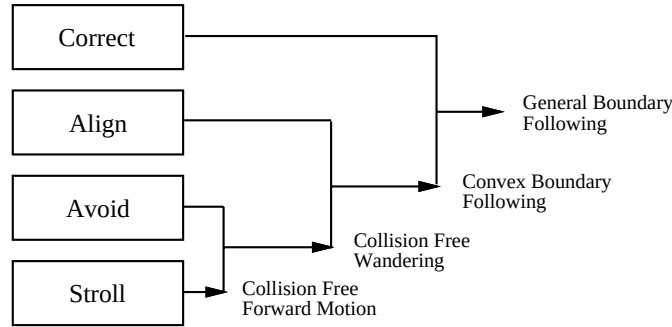


Figure 9: Incremental interaction of behaviors (adapted from [Mataric, 1992b]).

basic behaviors are summarized in Figure 9. While navigating the robots stays within edging distance from walls and, should an obstacle appear in front, it will stop. Differently from Brooks one can notice the addition of the “correcting” behavior which was introduced to keep track of sharp boundaries and navigate around them.

3.1.1 Landmark Detection

The detection of landmarks is dynamic and it is accomplished by continuously monitoring and relying on the tracing behavior of lower level. This particular reliance on the tracing behavior follows the subsumption architecture philosophy. It is, in fact, a good example in which one module *implicitly* depends on another for deriving the appropriate action. In this case the action derivable is the activation of a particular landmark behavior as a result of having gained a special vantage point in the environment. Each of the active landmarks can obtain information by accessing both sonars and compass directly, see Figure 10.

The features characterizing landmarks are: physical extent over time and confidence level associated with the landmark. These are recovered as consistency in sensor data on which error dynamic averaging is applied. The conditions to be met in order to characterize a landmark are: compass bearing and sonar sensor measurements. The measurements from the sonar sensor allow to identify whether the information is obtained from one of the side, identifying a wall-landmark, or from both side simultaneously, identifying a corridor. Once a short reading is obtained and stable bearings are observed, a confidence level on the occurrence of a landmark may be either defined, if new, or incremented, if previously observed.

A threshold confidence value τ is set to the width of the shortest landmark. A landmark extension is expressed in multiples of τ 's. This measurement is based on the assumption of constant velocity. However, while it is not explicitly stated, one must infer that the landmark recognition

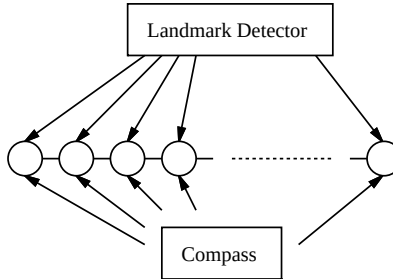


Figure 10: Distributed access of the nodes to the landmark detector and compass (adapted from [Mataric, 1992b]).

behavior is inhibited when the robot starts up or slows down. If this were not so, incorrect estimations of sizes of landmarks would be obtained. This would suggest that there is a mechanism to inhibit the output of the sensors. Therefore, since both avoidance and motion may not affect layers above them, this suggests that they probably inhibit the output from those sensors feeding the map building. This aspect of the inhibition control is clearly according to the subsumption architecture, however, the motivation for the signal suggests an implicit knowledge that the data might otherwise cause undesired side-effects on upper level modules. This approach may be suggested as a standard policy, however, there is no reason to support it. In fact, future models or different goals in the creature may in fact require that this information not be inhibited and hence result in a problem.

The landmark types are labeled as: **LW**, left wall, **RW**, right wall, **C**, corridor, and **I**, irregular. The selection of the landmarks to recognize is dependent of the sensors ability to detect them and on the parameter τ . These landmarks may lead to a sparse representation of space. Mataric suggests that by adding other sensor modalities, position control and additional behaviors may be possible to obtain a refined granularity. No mention is made of a possible multi-scaled approach in which a graph construction could have different types of nodes based on the granularity of the representation. This approach might lead to a less sparse representation with no major drawbacks in the matching and planning phase. Top of Figure 11 shows an environment with labeled landmarks.

3.1.2 Mapping Algorithm

The goal of the mapping algorithm is to produce a coarse scale map to allow the robot to get within sensing range of the goal. Graphs, unlike Cartesian maps, which are usually centralized, are convenient for encoding topological qualitative information about the environment. Each node identifies unique landmark neighbors and the links define the adjacency relations. The obtained structure in the graph is, hopefully, isomorphic to environment.

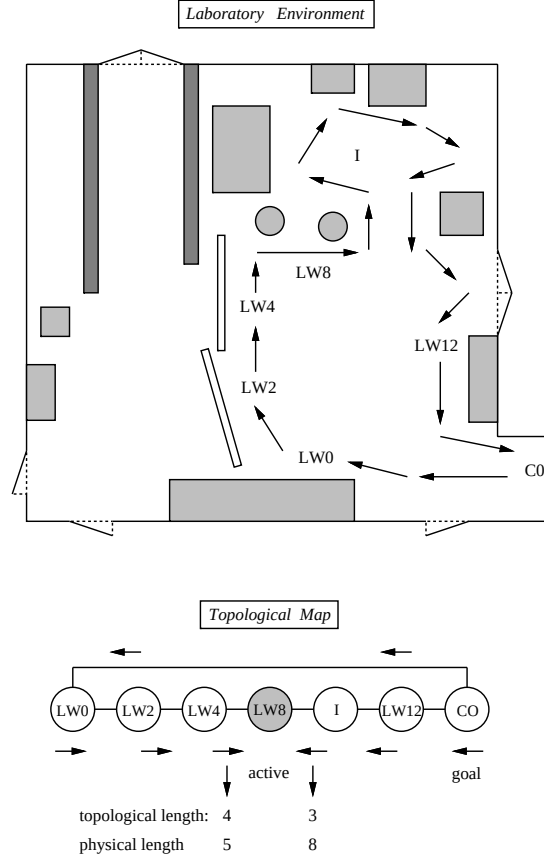


Figure 11: Environment and the topological representation recovered. The graph shows also the propagation of the activation obtained when the robot is located by landmark $LW8$ and has $C0$ as the goal destination. (adapted from [Mataric, 1992b])

The nodes in the graphs are concurrently active behaviors and encode topological relationships between landmarks, see bottom Figure 11. The graph, unlike in centralized representations, is not manipulable as a whole, rather it constitutes a network. It is presented as a natural extension of the architecture. The behavior of a node is described in terms of simple rules to match the detected landmarks and sends various activation messages for planning.

A landmark is described as a tuple $\langle T, C, L, P \rangle$:

$T \in \{LW, RW, C, I\}$ qualitative landmark type.

$C \in [0 \dots 15]$ compass bearings.

$L \in [0 \dots 127]$ rough estimate of landmark length.

$P = (x, y)$ such that $-128 \leq x, y \leq 127$ is coarse position estimate.

Once a node is created it is activated and added to the network. The matching criterias between the tuples for the first tuple, T , is simple equality; the second tuple, C , is considered to be matching if the compass heading is within a tolerance of 30 degrees. The second one is based on the heuristic that corridors and walls usually meet at right angles. Hence the position estimate L is re-calibrated upon revisiting a landmark and scaled by an error e proportional to size of L . Each landmark has a dual due to the direction it is encountered in navigation. Matching a landmark is carried out in constant time since it is performed in parallel regardless the size of the graph. The graphs shown in the paper are, however, minimal in size.

3.1.3 Landmark Disambiguation

The activation of a node spreads an expectation to its neighbors in direction of robot's travel priming them for an upcoming activation. The notion of expectation provides a contextual clue. By expanding the matching window to two nodes it becomes simpler to disambiguate nodes with same bearing. However it is not clear that this criteria is sufficient when the maps becomes large.

When the robot returns to a previously visited location the node at the location is activated once the topological link to the node is established. The tolerance in the matching is bounded by the size of the landmark. The position is not used for control but as an auxiliary source for landmark disambiguation. The combination of expectation and position estimation allow to disambiguate identical landmarks. A no match suggests a new node and that will be inserted in the graph.

Missing a landmark recognition would not affect the continuation toward the goal. This presumes that, after inserting the newly recovered landmark, the one following is recognized. No mentioned is made when several landmarks adjacent to one another are not recognized. While the graph is said to be dynamic, no mention is made about node removal. It would make sense that a node is removed if it finds no correspondence and in particular one might want to remove the Irregular nodes when other landmarks are inserted in their place. For instance, if a box is placed against the wall that will be perceived as an additional wall and with its removal should be manifested in the map by a removal of the landmark. Alternatively, a mechanism for decay or forgetting could be inserted, as presented in [Salganicoff, 1992].

3.1.4 Path Planning

The active and distributed nature of the representation allows for efficient path planning. A spreading activation from the goal in all directions eventually reaches the currently active node which identifies the locus of the robot. Since the number of nodes in the map is known, cycles may be

eliminated for these will yield paths longer than the maximum number of nodes. Alternatively a coloring schema message passing could be used to eliminate cycles and investigate the network but no suggestion of other methods is presented. The resulting shortest topological path is weighted by the physical length. A situation in which multiple paths of equal lengths may be encountered is not addressed. It can be assumed that in such an instance an arbitrary decision could resolve the ambiguities. On the other hand, a criteria involving ease in landmark recognition should be integrated to make the choice depend not only on length but on the ability of recognizing a landmark. It would make sense to select a path with obvious ease in recognition of landmarks rather than others with multiple nodes. Such weight could be defined inversely proportional to the ease of recognition of the landmark.

As it was mentioned in the beginning of the section, the most interesting asset to the paper is the representation adopted. In particular the three following characteristics are noteworthy:

QUALITATIVE Qualitative sensor characteristics are employed to construct a fault-tolerant navigation behavior to facilitate landmark recognition.

PROCEDURAL Each individual landmark identifies a specific behavior in the representation network.

DISTRIBUTED The nature of the distributed representation allows constant time localization and linear time planning.

It represents an alternative to the hybrid approach which separates the reactive and planning parts of the control system. Additionally, from a hardware standpoint, the quantized directions requires only fewer connections to broadcast between the various behaviors. Data and code for activations are blurred and 10 nodes require only 51K of memory. Larger maps would only increase linearly with the number of landmarks. However, large maps are never investigated neither in simulation nor experimentally.

3.2 Hybrid Layered Architecture

The hybrid layered architecture, SSS (servo, subsumption and symbolic system), presented in [Connell, 1992] addresses the bridging of the gap between servo and subsumption layers, by building situations recognizers, and the linking of subsumption and symbolic layers, by introducing event detectors. Such system is implemented in an indoor navigating robot.

This approach attempts to combine the best features of conventional servo-systems and signal processing with multi-agent reactive controllers and state-based AI systems. Servo-controllers have

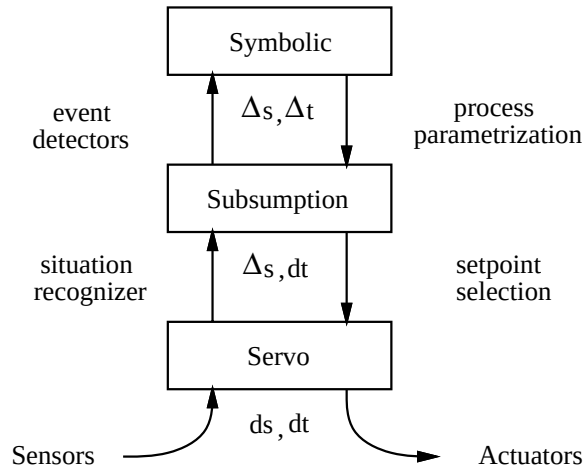


Figure 12: SSS architecture combining 3 control techniques (adapted from [Connell, 1989]).

problems since many real-world phenomena are either not well understood to be modeled accurately or are non-linear. Behavior-based or subsumption architectures do not impose modeling constraints and are good at making rapid radical decisions in a very limited domain. Since behavior-based systems are distributed, there is not an appropriate location for placing a world model. Connell points out that by bringing together the two type of architectures one must contend with different aspects in the representation. However, by introducing a hierarchy in the control structure one can benefit from the best of both approaches. A centralized representation can be introduced at a high level since it characterizes one of the capabilities of symbolic programming. On the other hand, problems real-time control can be delegated to subsumption and tactical control. This approach is similar in flavor to what [Stein and Paul, 1993] propose. Stein uses local real-time subsumption architecture to control local behaviors for telerobotics applications. The higher level decisions are carried out by a human operator while the local control for the various behaviors is emergent.

The partitioning of the architecture into three layers is derived from a quantization of Space, first, and Time, afterwards, see Figure 12. Thus, the servo-style system operates in a continuous domain of space and time. The behavior-based stratum characterizes the actions to be carried out at intermediary stages of the task execution. This is accomplished by continuously checking the sensors to recognize special contexts in which sets of behaviors may be activated. This constitutes a discretization of space (the action to be carried out) while the time of the situation is still continuous since it requires continuous monitoring. The symbolic system brings the quantization one step further identifying events and hence discreteizing time as well. The quantization defined above identifies the ability to monitor, recognize and react to changes in the world.

The robot is mounted on a 3-wheeled omni-directional base and uses sonar ranging and infrared

proximity sensors for navigation. Path commands are communicated via a radio link.

The interfaces between the layers can then be characterized as:

- a command transformation between behavior-based layer and servos level by way of matched filters.
- a symbolic and subsumption ability to turn off behavior selectively and to parameterize some of the modules. In this manner event-like commands remain latched.
- an interface between behavior and symbolic level is accomplished through event generations and a contingency table lookup (acting as behavior based indexing with parameters).

3.2.1 Navigational Task

The goal for the robot is that of mapping a collection of corridors and doorways and, upon having acquired the map, navigate from one office to another within the building. Two major problems had to be addressed. The first one related to the changeability of the environment and the second one to the recognition of an already visited locus. The first problem was solved by restricting to operate with very coarse geometric maps, recording distance and orientation between relevant intersections, and by allowing the robot the capability of following segments. The second problem could not be solved exclusively on odometry because of the obvious drifting problems in navigation. The issue was resolved by taking advantage of the geometry of the environment: corridors intersect at 90 degrees angles. This solution is similar to the one suggested in similar cases by [Mataric, 1992b].

Two types of navigation strategies were employed: a *tactical* and a *strategic* one. Tactical navigation addressed the moment-to-moment control between the servo and the subsumption layer. It was based on a basic behavior carried out as a wall following side-seeking behavior using infrared proximity detectors. Cumulative odometry measurements were also employed and average heading was implemented as a correction behavior for steering around a obstacles and placing the robot back on track.

Strategic navigation allowed to focus on where to go next and was carried out by the symbolic layer. A coarse geometric map of the robot's world was constructed using landmarks annotated with paths' distances between them. Once the map was created, a spreading activation algorithm, similar to the one presented by [Mataric, 1992b], was employed to determine the route to destination. Thusfar, it would seem that the process is no different than the one described by other researchers. The major difference, beside the architecture, is the role played by the symbolic layer in the

system. It enables the appropriate set of subsumption modules according to the adequacy of the situation. Additionally it can reconfigure the behavior when specific events occur. The alterations are performed in timely fashion by using a contingency table which contains the association between the behaviors and the parameters accessible at the symbolic level. For instance, the symbolic level may perform some checking with the overall average heading and current map. As a result of the consultation, the symbolic level can prevent the robot from wandering in an office, taking a wrong turn, simply because it has located an open door. Hence the symbolic level can prevent the wrong set of behaviors to be active by disabling them.

3.2.2 Experiments and Key Points

The paper presents two experiments. The goal of the first one was to validate the usage of odometry for loop navigation. The path description was rather roughly defined in integral number of degrees and integral number of inches. The symbolic map recovered matches, the important intersections and corners identification. In the map construction no adjustments were made for the width of corridor. The wide tolerance in matching nodes and flexibility in matching opening lead to some skewness in the map; however, traveling the path a second time yielded only minimal changes.

The second experiment shows the competence of the subsumption level in local navigation while using a coarse map. To verify this competence, the initial heading of the robot was altered. As a result the absolute map recovered varied in angular orientation; however, it remained consistent with respect to the landmarks and hence allowed the robot to navigate through the environment to the desired destination.

The key points characterizing this hybrid system can be identified as:

- The continuous interaction between the symbolic level layer and the subsumption and servo layers. In other implemented system, the symbolic level is often left out of the loop of control while the task is performed.
- The contingency table decouples symbolic and most rapid form of control, yet allowing their interaction.
- The handling of certain part of the navigation problem by different levels and technologies allows to construct a fast-responding, goal-directed robot control system.
- The communication between the levels can be characterized in terms of their directions. In the upward direction, links are based on temporal concepts of situation and events; while in the downward direction it relies on parameters adjustment and setpoint selection.

4 Discussion on Subsumption Architecture

Sections 2 and 3 have provided a description of the basic subsumption architecture and two applications. In this section we view some of the critical issues presented in the papers and highlight some of the outstanding problems with the architecture. Some examples from [Hartley and Pipitone, 1991] and [Arnold, 1989] are presented to clarify some of the points discussed. The first paper investigates the applicability of subsumption architecture in the design of the controls for an aircraft. The second one identifies experiences encountered in designing a simple system by using a subsumption environment for testing of the ideas.

4.1 Decomposition of Behaviors

One of the critical points of the overall approach presented by this architecture is the focusing on the subdivision in terms of behaviors rather than a subdivision based on the function that a module accomplishes. In this new approach, reactivity is identified as a fundamental property for a mobile robot. *Allen*, the first robot [Brooks, 1986], combined non-reactive capabilities with reactive ones. *Herbert*, the second robot, wandered in the office collecting cans and brought them to the a collection site. This task was performed by having a laser range system and a gripper. The gripper would react when an object of the adequate measure would break the inter-gripper beam. A detailed description of a similar implementation is given by [Connell, 1989]. A small time delay maintaining a state (three seconds) was introduced to ensure that the can was within the grasp. While this is an adequate solution to the situation, it seems very much *ad hoc*. It would seem that a better method would introduce the ability to learn these parameters rather than hard-wire them.

Subsumption architecture defines a hierarchical relation between the various modules, however, it is not clear that such a strict hierarchical decomposition is always adequate or even possible. At times issues of mutual exclusion need to be addressed often allowing only one of the behaviors to be active at a single time. [Hartley and Pipitone, 1991] present an example in which a simulated aircraft executes both the behavior of flying to a point in space and following a straight line trajectory. In such a case, the competition and interference of the behaviors lead the plane to fly off into infinity. The problem was resolved by adding another behavior being able to recognize and handle that specific instance. However, this solution does not seem to guarantee that, unless possible scenarios are investigated, the correct behavior for the system will occur. This suggests the need for controlling and perhaps supervising the various behaviors.

Mataric gives some heuristics on directions for problem decomposition. However, nowhere in the papers presented is there an explanation on the criteria governing the decomposition. It would

be of great benefit if the authors were to explain the reasoning behind the selection of a particular decomposition of behaviors.

A more crucial issue regards the partition between upper and lower layers. These can not really be designed independently. This was pointed out above when observing the actual construction of the second and third level in Brooks original robot. Hence, the claimed independence in the components highlighted in the basic philosophy was violated in the first attempts of the design. While the clear partition yields robustness, it requires the implementor to have a lot of foresight to provide for the loss of flexibility that is lost for not allowing to redesign the lower levels. Additionally, the designer would need to maintain a rather complete knowledge of the possible implications in the system appearing at all levels rendering, thus, the task too complex. This consideration suggests also that small changes at the lowest level can't be made without keeping in mind the propagation of the changes through the system and redesigning the control structure.

There are clearly behaviors which are not related hierarchically through priorities in terms of spatial properties. These are those, such as picking up of an object and putting it down, which are better expressed in terms of contextual temporal ordering rather than hierarchical. Furthermore, they are mutually exclusive. Additionally, behaviors may exhibit different priorities relative to different actuators.

In the hierarchical structure, higher-level layers can inhibit lower levels, however, there are clearly situations in which this should not happen as in the case of the detection of a potential source of trouble by the lower-level. This can be compared to the reactive motion which makes us pull away our hand from a hot object.

4.2 The World as Its Own Model

Brooks solution of employing the world as means of communication presents serious problems in some cases. [Hartley and Pipitone, 1991] suggest that such a situation would arise when, for instance, the communication refers to the use of the landing gear of a plane. The authors, however, seem to identify a specific problem which is, in this case, resolvable by allowing the landing gear to be endowed with its own altimeter and hence having the ability to open when approaching ground. This consideration, however, opens a door to a whole range of situations dealing with real-time decision making and the ability to actively control the action to be taken at any given instance in time. Reactive type behaviors are triggered by preconditions matched in the environment and many such conditions could be triggerable at any instance in time if the system is complex. A possible solution could be the propagation of the context through the various behaviors, as in the case of the hormone system presented in [Brooks, 1991a]. In the original implementation of the

navigating robot, Brooks resolved to wait out the situation. That type of approach can clearly not be adopted when real-time constraints and deadlines are crucial as in the case of flying an airplane.

[Hartley and Pipitone, 1991] discuss the simulation of flying an airplane from take-off to landing using a behavior-based approach. While it is only a simulation and hence contrary to the basic philosophy of having an embodied system, it identifies some problems discussed so far. A compiler was built to be able to translate the behavior descriptions into *C* code which could then be tested. The construction of this simulation unveiled several problems addressing the lack of good definition of modularity in the subsumption architecture. [Arnold, 1989] also points out that a simulation of a subsumption architecture provides the ability to experiment with different approaches without having to go through the trouble of completely implementing a system. It is clear that some issues can not fully be understood unless a complete system is developed. On the other hand, with some easy to manipulate environment³ some ideas can be eliminated in the early stages saving time for better implementation strategies. In the simulation of the aircraft described in [Hartley and Pipitone, 1991], the issues of control are resolved by introducing some external form of arbitrators in between the modules as part of the interfaces.

The control of the lower level, however, brings forward considerations on the handling of the inhibition and the suppression of signals. There may be situations in which it may be desirable to alter one of the behaviors only partially with respect to some aspect of the interaction. In [Brooks, 1991a] the hormone approach is suggested, yet it is not clear that this approach would resolve and address the problems completely.

4.3 Complexity and Scalability

When the issue of complexity is raised, Brooks points out that no artificial constraints were introduced in the lab environment to allow the robots to navigate. He adds that a similar approach would work in an outdoor environment. This argument seems to be flawed by the same problem that Brooks suggests AI researchers had when addressing the scalability of toy-world solutions to real environments. In a sense, the lab environment with the tasks that the robot had undertaken is slightly more complex than the block world environment [Brooks, 1991c]. Mataric [Mataric, 1992b] represents as landmarks straight walls and corridors and irrerecognizable landmarks. In the implicit representation used the consistency in measurements from the sensors environment characterizes the invariance for recognition. In the outdoors, assuming flat terrain, obstacle avoidance might be feasible, but navigation by landmark recognition becomes extremely complex.

³[Arnold, 1989] uses the simulation system called BRIE, Brown Robotics Implementation Environment.

The increase in complexity due to multiple actuators and sensors (the most complex robot *Attila* has 23 actuators and 150 sensors) is handled remarkably well by subdividing the actuators and sensors into the various components. The complexity of the multiple layers added in are answered by Brooks by bringing in activations schemas for the various behaviors. In [Brooks, 1991d], he presents a method which models the hormonal system of the lobster. While the implementation is interesting, it suggests that the original schema of control and passing of simple message scheme is not sufficient when attempting to carry out a more interesting task. The switching on and off of the behaviors is taken further in [Connell, 1989] where the context for a switch in behavior is recognized and carried out using contingency table.

4.4 Expressiveness of the Specification Language

The specification language presented above and employed by the various researchers is not fully developed with constructs beside what finite machines, provide with loops and conditions to be matched. That aspect allows very little machinery for expression and combination of behaviors. To this respect the effort taken both by [Hartley and Pipitone, 1991] and [Arnold, 1989] proved very helpful in characterizing some of the problems especially in the issues of interfaces between the modules.

[Arnold, 1989] points out issues of code redundancy in two areas: one addressing the parameterization of behaviors and the other noticing the need for behaviors which are subsets of those found in other layers. In the implementation of the example presented, this is accomplished by introducing a controller layer governing a fully tested library of common routines.

However, there are more severe problems with the mechanism which are provided by this architecture. In particular, it requires the ability to express each behavior in terms of some finite state machine. This problem poses major limitations in the computational power of the system. In particular, from Automata Theory we know that there are classes of problems which require memory to be resolved and mechanism for accessing it and modifying it. It is very doubtful that a purely reactive behavior-based system could be applied to solve complex problems. Furthermore, even simple problems, such as the tower of hanoi, would be hard to express and solve in terms of reactive behaviors.

According to the original specification, memory can not be either centralized or shared, however, if a level is to carry out any abstract reasoning it may require to be able to communicate with various level to acquire different type of information. If a level is not allowed to access other layers' information, this can pose a great computational burden on a particular layer essentially reproducing the behavior of other layers. It would seem that this could be avoided by allowing less

stringent constraints with the increasing level of competence of a layer.

4.5 Provability and Reliability of Behaviors

[Hartley and Pipitone, 1991] chose the flying of an aircraft as a the type of problem to try to resolve using subsumption architecture. While this example reveals some issues of conflicts resolutions and inadequacy in resolving some problems, it brings out the question of reliability of given behaviors. Brooks states, [Brooks, 1991d], that emergence is a sign of intelligence and it should not be easily definable the cause, within the behaviors, which motivated a specific action. While this consideration may be appealing when addressing some emulation of evolutionary behaviors, it certainly does not provide the requisite for guaranteeing repeatability. Namely, when flying a plane one wants to be able to rest easy and enjoy a nice flight rather than wonder whether the behaviors of the engine are not in conflict with other parts of the airplane. While the above example is slightly sarcastic, it suggests that the complexity of the task may require that the behavior be guaranteed rather than just feasible with a some probability.

4.6 Future Directions in Subsumption Architecture

The papers by Mataric and Connell identify directions in which subsumption architecture is heading⁴. [Mataric, 1992b], in particular, addresses the aspect of representation. This issue, however, goes beyond the simple representation of the map for navigation. Since it has implications addressing the issue of creating active representations of objects. An object recognition behavior could be inserted in a subsumption like architecture and trigger particular response or being inhibited by the task the robot is carrying out. Thus only certain objects could become of interest at a particular time and the rest could be ignored. This aspect, however, requires a mechanism for acquiring, or even better for learning, the object behaviors in first place. This aspects were not investigated and are only superficially outlined in [Brooks and Mataric, 1993]. [Connell, 1989] presents an alternative to the original subsumption style architecture marrying symbolic control and high-level planning with subsumption architecture. While the role of the symbolic level is primarily that of guaranteeing the correct contexts of behaviors and the construction of the map, this approach shows possible directions in which aspects of the traditional AI approach to problem solving may find applicabil-

⁴IBM is no-longer funding the research on the navigating robot as pursued by Connell. New papers by Mataric focus on interaction between multiple agents rather than focusing on further developments of subsumption architecture and Brooks has stopped pursuing this area actively. This suggests that the actual future of subsumption architecture is not too certain and future developments of this approach may diverge from the original inception of the criteria and goals.

ity. In particular, the approach seems to identify a partition in levels of intelligence: one, reactive and without explicit reason and representation, and the other, symbolic and capable of reasoning, learning, and in need of maintaining both memory and representation.

5 Discrete Event Dynamic Systems

In this section an overview of Discrete Event Dynamic Systems (DES) is given. The description was introduced by [Ramadge and Wonham, 1989] and [Ozveren and Willsky, 1990]⁵. According to DES theory the behavior of a dynamic system can be modeled as a non-deterministic finite automaton (Ndfa). In such a Ndfa arcs identify **events** and fragments of operational behaviors characterize the **states** of the system. The assertion of events may identify the beginning or the completion of an action or other semantically meaningful transitions. In particular, the theory provides for a methodology for the synthesis of a supervisor so as to impose a control structure over the various behaviors. The proposed structure of control defines a closed loop involving a supervisor, a plant and observer(s), see Figure 13. The objective of the formulated theory is that of examining theoretical ideas such as controllability, observability, and decentralized or hierarchical control for DEDS. In this section we will focus primarily on issues of controllability and observability.

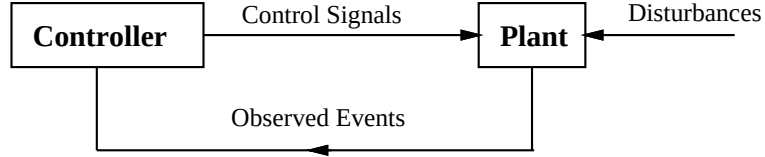


Figure 13: Controller and Plant.

According to the type of behavior focussed on, different model of DES are presented. These are:

- *Logical DES* models. In these the trajectory is simply specified by the sequence of the events. The time component is ignored in this case.
- *Timed or Performance Models* are concerned with timeliness of the events and can further distinguished in *non-stochastic*, Petri Nets and Max-Algebra, and *stochastic*, Markov chains, queueing networks. This distinction is made based on whether the time component is known a priori or modeled using statistical assumptions.

5.1 Basic Formalization

The formalism presented in [Ramadge and Wonham, 1989] falls in the category of Logical Models. These are characterized by trajectories of events. The formalism borrows from formal language

⁵A review of the DES formalism in comparison with Real-Time Temporal Logic Models is presented also in [Košecká, 1992] and a brief overview of [Ozveren and Willsky, 1990] is presented in [Sobh, 1991].

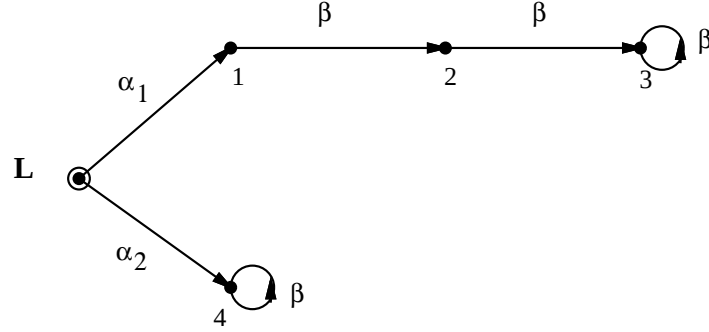


Figure 14: Example characterizing language $\mathcal{L} = \overline{\alpha_1\beta^2} + \alpha_2\beta^*$. The set of states $Q = \{0, 1, 2, 3, 4\}$, $\Sigma = \{\alpha_1, \alpha_2, \beta\}$, δ is implicitly represented in the transition, q_0 is state 0, and the marker states are $Q_m = \{0, 3, 4\}$.

theory [Hopcroft and Ullman, 1979]. Events characterize an alphabet and trajectories of events are expressed as strings over the specified alphabet, usually Σ . Then Σ^* identifies the set of all possible strings from Σ , including the empty string ϵ . The empty string represent no event. A Language L can thus be defined to represent all possible the trajectory of events from Σ^* . Since it is desirable to be able to recognize state of the system not only at completion but also in the progress, the language is required to be prefix closed. The *prefix closure* of $L \subset \Sigma^*$ can be expressed as:

$$L = \{u \mid uv \in L \text{ for some } v \in \Sigma^*\}.$$

The language L is accepted by some machine G . Alternatively, we can express L as the language generated by G . This generator can be characterized as $G = (Q, \Sigma, \delta, q_0, Q_m)$ where Q represents the set of all possible states, Σ the set of all possible events, δ the transition function $\delta : \Sigma \times Q \rightarrow Q$, q_0 the initial state and Q_m the subset of states called marker states. Marker states are similar in idea to what final states are for formal languages. Essentially languages thus expressed are *regular languages* and follow the same rules. A simple example of one such state machine is shown in Figure 14. Note the overbar on the first two terms of language $\mathcal{L}$ characterizing the prefix closure thus making both state 0 and 4 marked states.

Considering the marked states, a special subset of the language $L_m(G)$ identified as:

$$L_m(G) = \{s \mid s \in L \text{ and } \delta(w, q_0) \in Q_m\}$$

characterizes the set of states defining possible “tasks” or (sequences of tasks). Unlike in the definition of formal languages, once a marked state has been reached there is no implication of termination. Clearly, a trajectory could define a task with repetitions of subtasks in it. Connected to this completion characteristic a property of DES is identified: A generator G is said to be **non-blocking** if every event sample path in $L(G)$ can be extended to a complete path. This property

of a system prevents trajectories which would otherwise lead to catastrophic events (transitions to dead states) with no possibility to reach a recoverable state. We will examine this more closely when reviewing [Balemi *et al.*, 1991].

Unlike, in traditional acceptors for formal languages in which the input is read and the transition is taken, state transitions in DES occur instantaneously, spontaneously and asynchronously. These three characteristics can easily be seen in the context of a robotic task such as the coming to contact of an end-effector with a surface. Instanteneity refers to the fact that once an event occurs, the coming to contact, a transition to a different state occurs. In a certain sense, once the event has occurred the transition to the different state has already occurred. Spontaneity refers to the fact that the occurrence of an event is out of the control of the observer. Namely, if we consider an example applied to Robotics, the coming to contact does not depend on the observer but on the current kinematics and dynamics of the robots. Finally, asynchronicity refers to the non-timed aspect of the system, and again happening external to the observer. The non-determinism of the system suggests that for any given state a particular event may characterize a path to possibly different marker states. In the context of a robotic arm moving, the start up event is a common event which could be followed by other events distinguishing further the development of the task.

5.2 Controlling Discrete Event Dynamic Systems

The presentation given so far is essentially that of a regular language. When addressing a task, however, one would like to be able to establish control over the sequence of events which characterize the development of the task. Furthermore, it is important not only to control the events but to try to identify potentially critical situations which could develop next and, within possibility, prevent them from occurring.

In order to define the control over a task, events are differentiated into two classes: **controllable** and **uncontrollable**. The former class identifies events which can be prevented from occurring and the latter those on which has no means of preventing from taking place. Hence we say that events which are controllable can be either *disabled*, prevented from occurring, or *enabled*, allowed to take place. Since uncontrollable events are beyond the ability of the controller to be prevented, they are considered as always enabled. The two classes of events partition the event set

$$\Sigma = \Sigma_u \cup \Sigma_c; \quad \Sigma_u \cap \Sigma_c = \emptyset$$

Examples of uncontrollable events are machine breakdown in manufacturing applications, loss of packets in communication systems, the unintentional crushing of an object being grasped or the shattering of an object being pressed upon by an end-effector.

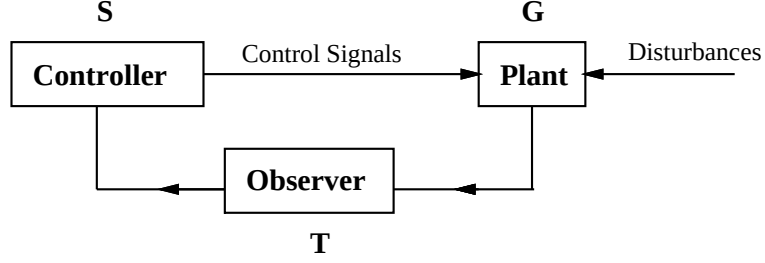


Figure 15: Controller (Supervisor: S), Plant (Event Generator: G) and Observer.

The original definition of the generator is then extended by allowing a sequence of control patterns associated with events and transitions to certain states. Hence all possible control pattern associated with events are characterized by Γ defined all the possible binary assignments for each element in Σ_c . The assignments of a control from Γ is expressed by the function $\gamma : \Sigma_c \rightarrow \{0, 1\}$

Now the transition function δ can be redefined to account for the control pattern as $\delta : \Sigma \times Q \times \Gamma \rightarrow Q$. Hence the transition defined by δ is enabled if $\gamma = 1$ and not defined otherwise. The generator can also be augmented to operate on the controllable strings as $G = (Q, \Sigma \times \Gamma, \delta, q_0, Q_m)$ is called *Controlled DES*.

Hence the behavior of the system can be modeled and a controlled pattern can be imposed on it. A **supervisor** then generates the control pattern to be sent to plant. In other words, the supervisor can be thought as of a function mapping from the language L to the control sequence Γ

$$f : L \rightarrow \Gamma$$

As a result the language $L(G, f)$ defines those strings which are generated by G and subject to the control sequence of the supervisor. This language, L_f can be expressed as:

- i) $\epsilon \in L_f$; and
- ii) $w\sigma \in L_f$ iff $w \in L_f$, $\sigma \in f(w)$, and $w\sigma \in L$

Additionally if the language is a marked language, then

$$L_m(G, f) = L(G) \cap L_f$$

characterizes a language of those tasks completed under supervision.

As shown in Figure 15 the generator G plays the role of the “plant”, the object to be controlled, T functions as the “observer” and S is the supervisor. In this manner, the loop of control between supervisor and plant may be closed. The formalization presented thusfar does not automatically guarantee that such supervisor can be constructed. Before formalizing this aspect, the main role

of the supervisor can be described as: to modify the open-loop behavior of a plant given some specified constraints expressed in the function Γ .

The problem then can be specified as: *Given a Controllable Discrete Event Dynamic System G with behavior L , what are the closed-loop behaviors $K \subseteq L$ which can be achieved by supervision?* The particulars on the existence of a supervisor relates to the notion of **controllability** of a system. In particular this aspect identifies the ability of reaching a desired state of the system from a given state. This aspect is related to the aforementioned non-blocking property of a system. A given behavior $K \subseteq \Sigma^*$ is said to be *controllable* if

$$\bar{K}\Sigma_u \cap L \subseteq \bar{K}$$

In the above $\bar{K}$ represents the prefix closure of the language identifying the desired behavior. Then the above condition states that for any string $w \in \bar{K}$ followed by an uncontrolled event $\sigma \in \Sigma_u$ so that $w\sigma \in L$; $w\sigma \in \bar{K}$. What this intuitively means is that if a sequence of events w is followed by an uncontrollable event σ , given that this event can't be prevented from occurring, then the string $w\sigma$ remains in $\bar{K}$. The role of the supervisor through language L_{mf} is then that of controlling the language L achieving non-blocking behavior. What is more important is that if the above property holds for the desired behavior expressed by K then, by Theorem 6.1 in [Ramadge and Wonham, 1987], the existence of such a supervisor is guaranteed.

The properties can be summarized with the following proposition:

Fix a non-blocking DES G with closed behavior L and marked behavior L_m

1. For nonempty $K \subseteq L$ there exist a supervisor f such that $L_f = K$ iff K is prefix closed and controllable.
2. For nonempty $K \subseteq L$ there exist a supervisor f such that $L_{fm} = K$, and the closed loop system is non-blocking iff K is controllable and $\bar{K} \cap L_m = K$

Additional properties allow to express approximations for behaviors expressed in terms of a closed set, under union and intersection, of classes of languages $C(K)$. We will not review these aspects here, however, we will point out the following:

- composition $||$ operator is described as: given language L_1 and L_2 , these can be composed using the *shuffle* product. The resulting language will have the cross-product of the states⁶.

⁶More sophisticated methods of module composition are described; We will discuss one of these in the paper by Balemi.

- an iterative method can be applied for reducing the number of states by introducing constraints and mutual exclusions to the resulting $L_1 \parallel L_2$ language (detailed in [Wonham and Ramadge, 1987]).
- the same composition can be applied to the languages described by the supervisor constructing the supremal of a controllable language. This is addressed as the synthesis of a supervisor, and, in addition, the construction of an *efficient supervisor*. Under the same considerations *non-conflicting* supervisors can be addressed.

5.3 Observing the Behavior of the Plant

Thusfar, we have talked about controlling the plant and that some feedback was obtained to close the loop. It is, therefore, necessary to investigate how this loop is closed. The feedback is obtained by incorporating an **observer** whose role is that of monitoring the behavior of the plant. [Ramadge and Wonham, 1989] do not present a detailed description of the observer role. [Ozveren and Willsky, 1990], however, stress this aspect more in detail. Additionally, not all events may be observable and hence **partial observability** needs to be addressed. Observability captures the notion of being able to determine at every point of the task-development the path taken to reach the particular state. Observable events are identified by the *observation alphabet* $\Sigma_o, \Sigma_o \subseteq \Sigma$. The events in Σ can then be interpreted as being filtered through the projection function P mapping: $P : \Sigma^* \rightarrow \Sigma_o^*$ where P is defined as

$$P(\sigma) = \begin{cases} \sigma, & \sigma \in \Sigma_o \\ \epsilon, & \sigma \notin \Sigma_o \end{cases}$$

The extended relation on string for the projection function can then be defined as:

$$\begin{aligned} P(\epsilon) &= \epsilon \\ P(w\sigma) &= P(w)P(\sigma) \end{aligned} \quad w \in \Sigma^*, \sigma \in \Sigma$$

The notion of partial observability is described with respect to distributed supervisors. We will not dwell with that aspect here.

With respect to issues of observability, it is important to emphasize the work of Ozveren and Willsky. [Ozveren and Willsky, 1990], present a theory for DEDS in which ambiguities and state indistinguishability are allowed to occur. The major differences with Ramadge and Wonham are characterized by the type of controllability and observability of the events. These are predicated on the notion that it is impossible to know the state of the system at every point during the execution of the plant. The controllability is geared toward maintaining the system into a set of specified states (*E-states*); hence, the focus in stability and stabilizability. Furthermore, the authors point

out that when dealing with real-systems it is important to address the error recovery capability of the system and to prevent “catastrophic error propagations” [Ozveren and Willsky, 1991]. In order to prevent the system from taking detours into infinite sequence of transitions to states other than E-states, they require that observations occur with a certain regularity. Namely, that current state of the system be observable at points in time separated by a bounded number of transitions. The indistinguishability in some states introduces other dilemmas in the modeling of the observer. In particular, it may be possible to reconstruct the state of the system only after having observed a sequence of events rather than being able to establish the exactly the state of the system based on a single observed event. This is termed as observability with a delay.

5.4 Review of The Formalism

The major points which the DES formalism provides can be summarized as:

- a qualitative expression of control;
- the control over the behavior of a plant through the introduction of supervision and observation;
- a methodology for constructing behaviors founded on formal language theory;
- the ability for modular composition;
- the capability of synthesizing a supervisor for guaranteeing that a given behavior can be enforced;
- a mechanism for reducing the possible state space expansion by composing the resulting language of behavior with a set of constraints;
- the closing of the loop between the plant and the controller level.

While in the above we have presented a formalization of a methodology for controlling a plant, there are limitations.

One of the hard problems is the model definition and the characterization of the constraints. This problem, common to the definition of models in general, is a bit exacerbated by the fact that the language for expressing the behavior is a formal language and hence not intuitive. Furthermore, the transition from the definition of the behaviors and supervisor to express the control as well as of the observer is not straight forward. In fact, the definition provided by the formalism presented does not map straight forward into the particular hardware to control. Additionally, while the

formalism provides means for analyzing the behaviors and control them, nothing is being done to address how the control is to be enforced from a Classical Control standpoint. No mention is made as to how and where PID type controllers should be introduced to control the behavior. In otherwords, DES allows for means of partitioning the event space and controlling, at a high-level, the behavior of the system. How that is carried out, at a lower-level, is dependent on the specific hardware specific means of control typical of Classical Control Theory.

When dealing with real-time constraints the model, as defined above, is limited since it is qualitative and event driven. This aspect becomes a limitation when it is important to introduce deadlines in the execution and control of the system. Research addressing the issue of real-time are presented in [Brandin and Wonham, 1992].

The observation of the events occurring in the plant are always addressed as observable with probability 1 or not observable at all. However in real-world situations, in which the environmental conditions are not clearly observable and monitored by sensors, it is necessary to introduce aspects of probability. While it might be possible to introduce probability by using stochastic models, Markov Random Fields for instance, it is not clear how multiple behaviors and their probability could be combined. Two aspects, in particular, need to be addressed:

- the assertion that an event was actually observable by one or more sensors, and
- the role that the observed event plays with respect to other observed events.

The first aspect was firstly addressed by [Sobh, 1991] with respect to the construction of a visual observer for tracking a moving target. This approach was, however, open loop and did not affected the control of the task. [Bogoni, 1993] presents a preliminary approach addressing both issues of determining the probability of the observation of a single event by multiple observers and the combination of observations of multiple events by multiple observers.

6 Control of a Rapid Thermal Multiprocessor

This section discusses an application of supervisor control theory to a semiconductor manufacturing workcell [Balemi *et al.*, 1991]. Rapid Thermal Multiprocessing, RTM, consists of a processing chamber capable of performing a number of processing steps such as cleaning, annealing, oxidation and chemical vapor deposition using a multitude of attached machinery. The goal of flexible manufacturing is that of being able to accommodate frequently changing manufacturing requirements.

This work, based on the theory of DES as developed by [Ramadge and Wonham, 1989], presents an input-output interpretation of supervisory control. It represents one of the first application of DES theory to manufacturing applications. The contributions of this work are derived mostly from the fact that it maps a theoretical methodology of control to a real problem. As a result the authors developed a tool for constructing supervisors for a manufacturing workcell. In having to map the theory to a real system, issues of interfaces and verifications of behaviors had to be dealt with.

In this review, we will focus on the re-interpretation of the closed-loop relation between the supervisor and the plant. The formalism for the construction of a supervisor and the subordinate processes implementing the sensing and actuating are presented by looking at an example. After this, considerations addressing the requirements of an interface between the logical and the physical processes controlling the workcell follow. As a result of the gained insight on the investigation of the requirements and constraints imposed by the interface, a refinement of the supervisor into an implicit and an explicit supervisor are presented. Having formalized and considered the constraints and requirements, the synthesis of the supervisor is then introduced. In order to verify the resulting automata, the authors introduce a symbolic approach, based on a fixed point iteration mechanism. In particular, the derived method allows to accomplish the goal, avoiding the otherwise exponential explosion in the verification methods which one would encounter when using the standard iterative approach presented in [Ramadge and Wonham, 1989]. Since the symbol manipulation is based on propositional logic, an efficient data structure encoding for the formulae is introduced. The review of this paper concludes with a brief block-diagram description of the application tool developed and some additional comments.

6.1 Input and Output Interpretation

[Balemi *et al.*, 1991] revise the plant model proposed by [Ramadge and Wonham, 1989], in which the plant generates events “wildly”, both controlled and uncontrolled, and replace it with an input/output model with *commands* and *responses*, see Figure 16. The supervisor can also force events as inputs to the plant. The new relation of the supervisor to the plant is shown in Figure 17.

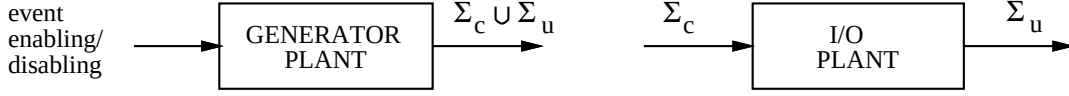


Figure 16: Ramadge and Wonham plant generator (left), inputs are enable and disabling commands and outputs are $\Sigma_c \cup \Sigma_u$. Balemi et al. Input/Output plant generator (right). Inputs are Σ_c (commands) and Outputs are Σ_u (plant's responses).(adapted from [Balemi *et al.*, 1991])

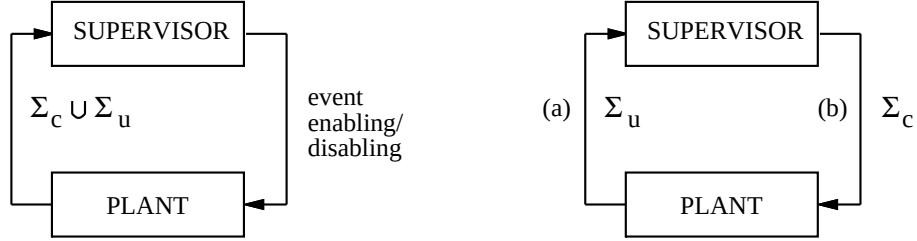


Figure 17: Asymmetric (left) and Symmetric (right) feed-back loops.(adapted from [Balemi *et al.*, 1991])

The modifications introduced are defined to account for a better behaved environment in which the response to commands is well-defined. However, there might be some complications in the process with some undesirable events taking place in the plant. The system to be designed should be non-blocking so that actions may be taken by the supervisor when these events take place.

In [Ramadge and Wonham, 1989] there is no issue of timing between the supervisor and plant and, in fact, the two are considered to work asynchronously. In this case, on other hand, there should be synchronization between the two processes. The authors distinguish two types of synchronizations:

- *Full Synchronization* when an event taking place in more than one process must be agreed by all the processes who carry out the event in their alphabet; and
- *Prioritized Synchronization* when an event is initiated by a process without requiring consensus by others.

Furthermore, with respect to the uncontrollability of the events, the property of receptiveness is described to account for the fact that when an uncontrolled event occurs in the plant it should be followed by the supervisor with a transition. Since the supervisor can not prevent it from happening, then it might as well be *receptive* and take the adequate measure to recover from the occurrence of the uncontrolled event. Likewise the plant can't prevent the supervisor to issue commands so it is best if it is receptive to them. This view allows to express the new close-loop relation between plant and supervisor. To account for the modification in relations to the plant

and to allow synchronization, the supervisory control problem is then modified to:

Given a plant P , find a supervisor S satisfying the following criteria:

- i) S is non-blocking for P – the partial execution in S can be extended to an marked execution in both S and P .
- ii) $M_{S||P} \subseteq L_{spec}$ – the set of distinguished traces should be in the prefix closed language of the specification.
- iii) S is receptive to P – uncontrolled events are not prevented from occurring in the plant.
- iv) P is receptive to S .

where L_{spec} is the language specifying the behavior. We note the symmetry condition (iii) and (iv) indicating mutual receptivity. In the above, the subscript $S || P$ identifies the composition of the process of the supervisor and the plant. A process $P = (L, M)$ is defined in terms of a language L and a set of traces. The composition operator for two processes P_1 and P_2 denoted as $P_1 || P_2$ is defined as:

$$P_1 || P_2 = (\text{del}(\Sigma - \Sigma_1)^{-1}(L_1) \cap \text{del}(\Sigma - \Sigma_2)^{-1}(L_2), \\ \text{del}(\Sigma - \Sigma_1)^{-1}(M_1) \cap \text{del}(\Sigma - \Sigma_2)^{-1}(M_2))$$

where $\text{del}(D)(L)$ represents the language obtained by removing all the occurrence of symbols in D from L , this is called L' . Its inverse is characterized by $\text{del}(D)^{-1}(L') = \sup\{ L \mid \text{del}(D)(L) = L' \}$. We can clarify what this mean by the following example.

Let $L_1 = (abc + abb + acb)$, with $\Sigma_1 = \{a, b, c\}$ and let $D_1 = c$ then the resulting language $\text{del}(D_1)(L_1) = (ab + abb) = L'_1$ then the inverse of $\text{del}(D_1)(L_1)$ denoted as $\text{del}(D_1)^{-1}(L'_1)$ is the supremal of L_1 to which by removing D_1 we can obtain back L'_1 . What this tell us is that the resulting language is the augmented language in which we have added in D_1 every where it was possible. One such languages is actually the original language L_1 .

It is worthed take a moment in the discussion to clarify this construction which is not explicitly described in the paper and which is of major importance for the definition of composition of the supervisor with other processes. This construction is a generalized version of the cross product or shuffle operator. [Ramadge and Wonham, 1989] also present this type of product, called synchronous product, of which the shuffle product is only a special case. The problem with employing the shuffle product with modules, which may contain similar events, is that in the shuffle the semantics will not be preserved. The semantics is noted not only by the events but by the marker states and hence sequences of events. Hence, since the cross product is insensitive to the events defined by the marker states, the resulting automaton would not preserve the same semantics.

Let's consider now an example. Let $L_1 = (ab)^*$ be a language and let the marker language be $M_1 = L_1$. Let $L_2 = (ac)^*$ be another language and let again the marker language be $M_2 = L_2$. Let $\Sigma = \{a, b, c\}$ then $\Sigma - \Sigma_1 = \{c\}$, let's call this D_1 ; likewise $D_2 = \{b\}$. Then the augmented languages are $L'_1 = (c + ab)'$ and $L'_2 = (b + ac)'$. Now taking the intersection $L'_1 \cap L'_2$, since by Theorem 3.3 in [Hopcroft and Ullman, 1979] regular languages are closed under intersection, this intersection exist. The proof of that theorem and the answer to the question is a constructive one. This is obtained by constructing the cartesian product of the states, with the resulting alphabet Σ but navigating the automaton by considering pairs of states and how the combined transition functions act in the individual automata. The resulting automata from $L_1 \parallel L_2$ is given by the language $\mathcal{L}_{L_1 \parallel L_2} = (ab + ac)^*$.

In order to describe how the modeling of a plant which follows the above characteristics the following three steps can be identified:

Step 1 Model the high-level behavior of the plant. This results in what is identified as the *fundamental* process Ξ which best matches the structure of control.

Step 2 Design a logical interface to the physical plant starting from the fundamental process Ξ . In order to do this low-level routines Ψ_i are chosen. These interact with the physical system by actuating the sensing processes.

Step 3 Compose a description of the fundamental process using the resulting routines.

Thus, the fundamental process $\Xi = (L_\Xi, M_\Xi)$ with alphabet $\Sigma_\Xi \subseteq \Sigma$ models the qualitative changes occurring in the subsystem. The *actuating* and *sensing* processes $\Psi_i = (L_i, M_i)$ with $1 \leq i \leq p$ define the underlying desired behaviors and the final plant process $P = (L_P, M_P)$ can be constructed as:

$$L_P = \overline{\text{del}(\Sigma - \Sigma_\Xi)^{-1}(M_\Xi) \cap [M_1 \cup M_2 \cup \dots \cup M_p]^*} \quad \text{and} \quad M_P \subseteq L_P$$

Hence the above expression identifies the language in which all the symbols and marked traces (notice the prefix closure) appear in the language of the supervisor M_Ξ and in those of the underlying processes $[M_1 \cup M_2 \cup \dots \cup M_p]^*$.

The above procedure is best illustrated through an example. The alphabets of commands and responses are

$$\begin{aligned} \Sigma_u &= \{\text{r_valve_failed}, \text{r_valve_opened}, \text{r_valve_closed}\} \\ \Sigma_c &= \{\text{c_open_valve}, \text{c_close_valve}, \text{c_repair_valve}\} \end{aligned}$$

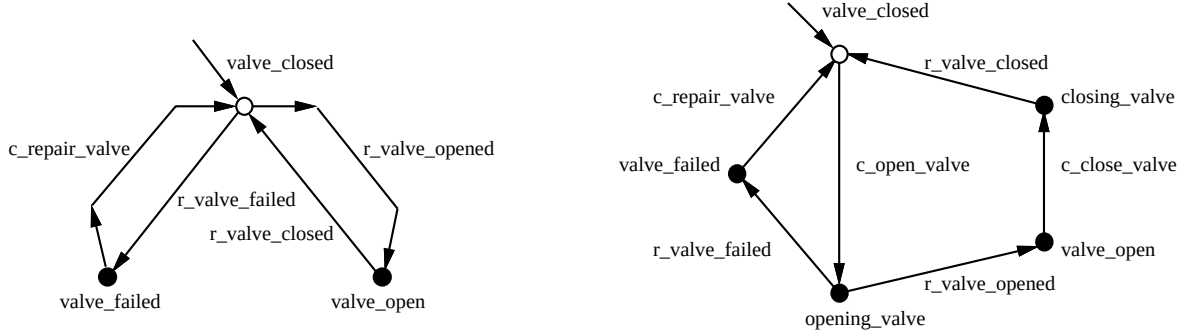


Figure 18: Automaton for the fundamental valve process (*left*); Automata model for the gas valve (*right*). (adapted from [Balemi *et al.*, 1991])

where the leading c stands for commands and r for responses. The fundamental process with alphabet Σ_{Ξ} is:

$$\Sigma_{\Xi} = \{r_valve_opened, r_valve_closed, r_valve_failed, c_repair_valve\}$$

with automaton shown in Figure 18 (left). In it the only marked states is the initial one. The marked language M_{Ξ} is given by

$$M_{\Xi} = \{((r_valve_opened \bullet r_valve_closed) + (r_valve_failed \bullet c_repair_valve))^*\}$$

and the language $L_{\Xi} = M_{\Xi}$. This defines step 1 in the above construction. The sensing processes Ψ_i are given by the languages:

$$\begin{aligned} M_1 &= \{c_open_valve \bullet (r_valve_opened + r_valve_failed)\} \\ M_2 &= \{c_close_valve \bullet r_valve_closed\} \\ M_3 &= \{c_repair_valve\} \end{aligned}$$

and again in this case $L_i = \overline{M_i}$. This constitutes step 2. The final step is basically the construction defined before. In this case the only marked state is the initial one and the resulting automaton shown on left in Figure 18 is expressed as the marked language:

$$M_P = \{(c_open_valve \bullet (r_valve_opened \bullet c_close_valve \bullet r_valve_closed + r_valve_failed \bullet c_repair_valve))^*\}$$

with $L_P = \overline{M_P}$.

6.2 Logical versus Physical Plant

Having constructed the process for controlling the system, it is necessary to bring the physical plant in connection to the logical plant. This is accomplished through the *interface* which extracts

the information to supply to the supervisor (the response) and at the same time for enforcing the commands for the plant.

The response are extracted from the system directly or indirectly by sensors. Amongst these can be distinguished three types:

Event sensors can be characterized as physical changes sensed directly. These are forwarded directly to plant output.

Discrete state sensors obtained by the interface by polling. Example of these are the states of a valve (open or closed) as define above.

Continuous state sensors monitored by the interface and mapped to a discrete state set, eg. *cold, hot, processing temperature*.

Additionally, **software messages** are communicated through the interface these are usually triggered by timeouts.

As mentioned above the role of the interface is that of interpreting the responses but also that of enforcing the commands from the supervisors. Again three types are identified:

Discrete actions to activate or disengage a state change in the system hardware.

Continuous actions to initiate a continuous action performed by a routine

Change in software execution to alter the execution of a program by pure software instructions. This could represent, for instance, a selection of a different algorithm to control the temperature subsystem.

6.3 Refinement of the Supervisor Model

Some of the specifications in carrying out a task are task-specific or *explicit* while others are equipment-specific and do not necessarily relate to a task and hence are *implicit*.

- Implicit specification are:
 - *Safety specifications* expressed through some language L_{spec} . These are meant to prevent the plant from executing certain sequences.
 - *Fundamental liveness specifications* to enforce repetitive type behaviors.

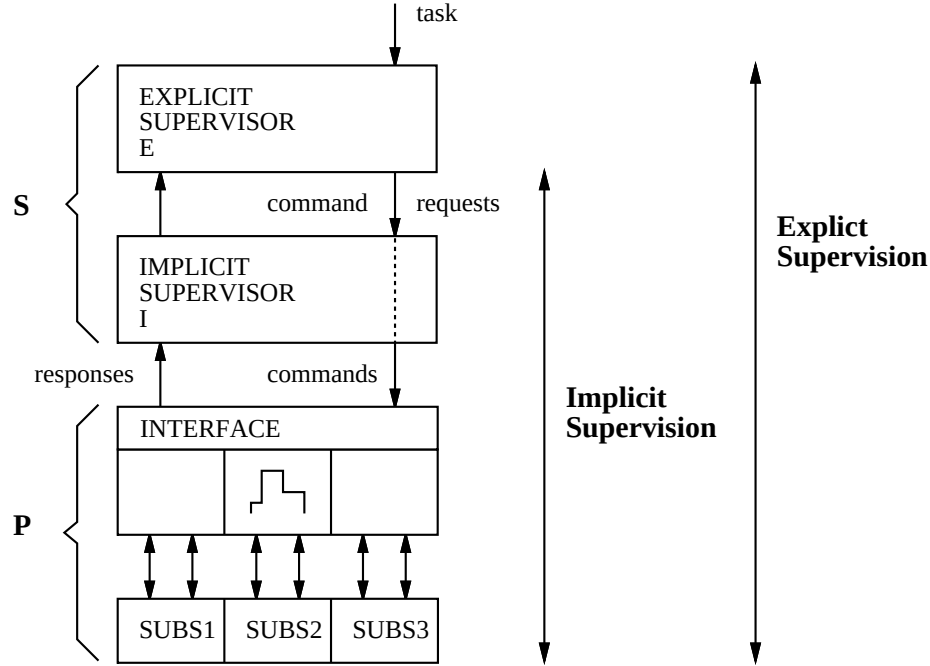


Figure 19: Generic scheme for the control of a discrete event system. (adapted from [Balemi *et al.*, 1991])

- Explicit specification are:
 - *Safety specifications* (similar to the implicit)
 - *General liveness specifications* to enforce the termination of certain logical sequences defining the task.

Based on the partitioning of the specifications, a partitioning of the role played by the supervisor is possible. This is accomplished by defining an *Implicit Supervisor* (I) and *Explicit Supervisor* (E). The partition has the advantage of reducing the state space to be controlled by what would be carried out by a single supervisor. The two supervisors are composed using the composition operator: $S = E \parallel I$.

The diagram in Figure 19 shows a generic scheme for the supervisory control of a system that is based on the above description. The physical plant, the interface and the implicit and explicit supervisors are shown. In this model full synchronization is assumed though it might be replaced by a prioritized synchronization.

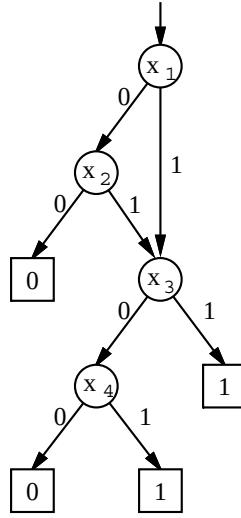


Figure 20: A Binary Decision Diagram representing the boolean function $(x_1 \vee x_2) \wedge (x_3 \vee x_4)$. (adapted from [Balemi *et al.*, 1991])

6.4 Synthesis of Supervisor

Up to this point the description of the supervisor has been presented in terms of processes and constraints, however, when describing a system, verification is required. Namely, the state space needs to be explored to verify that the resulting behavior is what has been obtained through the specifications. The authors point out that even in the simple cases the state explosion can render the actual implementation infeasible. To overcome this problem a novel approach is presented. The key issues are:

- a logical encoding of the fixpoint operator and
- an efficient data structure to investigate the logical encodings.

Theorem 3.1, on the fixed point iteration, basically tells that it is possible to compute a boolean function which expresses symbolically the next state relation of an automaton for the supremal of a controllable language. Namely, it provides a mechanism to progressively span an automaton which encodes boolean functions expressing the behavior of the system. In order to make this possible, the fixpoint operator expressed in a logical form allows to perform the computations symbolically avoiding the explicit enumeration of the states. Henceforth, it was necessary to express relevant information defining a plant's behavior as logical formulae. Additionally to gain efficiency a particular data structure called Binary Decision Diagrams were employed. These are binary decision trees with added restriction on the order of the decision. It is represented as an acyclic

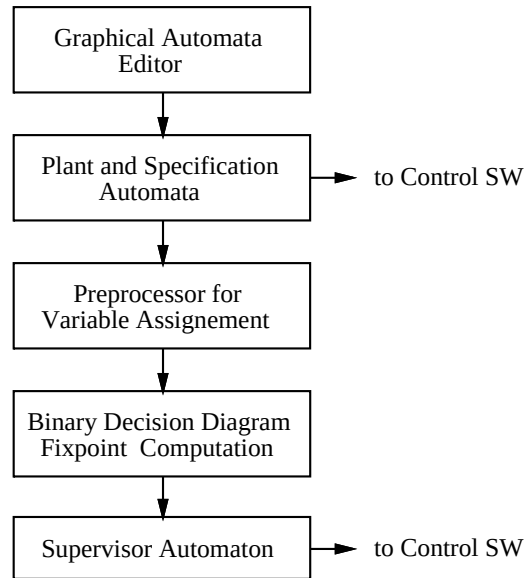


Figure 21: Block diagram of the synthesis program.(adapted from [Balemi *et al.*, 1991])

directed graph. An example is shown in Figure 20. Any particular path represents a possible evaluation assignment for the variables.

6.5 The Application System and Observations

The analysis of the problem for the Rapid Thermal Multiprocessor resulted in the development of package which allows a tractable approach to the synthesis of a supervisor automaton to control the plant. The block diagram representation is shown in Figure 21.

We have seen here a successful implementation of a system and the development of an application tool for constructing supervisors. The considerations and modifications of the original semantics were brought about by the well-structureness of the manufacturing environment. In a workcell, the behavior of the plant is well-behaved and hence the removal from the plant of the possible generation of “wild” events is well-founded. The further subdivision of the supervisor into an implicit and explicit one and the addressing of the role of an interface are extremely important for constructing a system. Issues relating to communication and synchronizations must be tackled when building a system. This work, while restricted to manufacturing environment, casts some lights on considerations and possible avenues of investigations to be kept in mind when developing a discrete event based system for a less structured environment.

7 Discussion on Discrete Event Dynamic Systems

In the preceding sections we have examined the formalism presented by [Ramadge and Wonham, 1989] and briefly made a comparison to other formalizations of Discrete Event Dynamic Systems, [Ozveren and Willsky, 1990] and [Ostroff, 1992], and the work by [Balemi *et al.*, 1991] was detailed in the previous section. A pressing question needs to be posed at this point, namely, “Given that the formalizations presented seem to define the panacea for solving the problems in control theory, why is it that these formalizations find wide-spread applications for the most part in manufacturing environments and communication protocols?”

The answer to this question rests on the issue of the transition from continuous real-time description of the world to its discretized representation. In the case of the above mentioned areas, and a few others, the mapping is easily achieved. In communication protocols, packets are sent, received or lost, etc. hence the discrete representation of the behavior of a network is quite suitable for the formalism. In addition, the arrival or departure are easily mapped in the an event-like structure⁷. In the manufacturing environment, as we have seen in the example presented in the previous section, the actions are easily quantizable. Operations dealing with a medium such as temperature or pressure, which are inherently changing in a continuous fashion, can be discretized because the processes governing the development of a manufacturing process are triggered at exact threshold values. Hence, the mapping to a discretized system is again easy to map.

When, however, we are dealing with controlling navigational behaviors, see for instance [Košecká and Bajcsy, 1993], the estimation of location in the environment or the description of controls which are non-linear, the mapping to discrete events is not easily achieved. Furthermore, the situations defined above present well-behaved and modelable environments. In the case of navigation or of tracking an object being manipulated, the time of the occurrence of a particular event is not well defined. [Sobh, 1991], for instance, bases the description of an intelligent observer primarily on Ozveren’s formulation of DEDS. The focus of the work is that of controlling an active observer tracking a robotic arm carrying out a task. In his application, he defines measures of confidence on the observation of an event through a camera.

The event space complexity for the verification can be curbed, as presented in [Balemi *et al.*, 1991]. That result makes the discrete event state approach much more feasible even for complex problems; however, how should a situation in which states are not predictable be handled? Such issues are not well-defined and, in fact, in cases where the uncertainty of the observation must be brought in, the verifiability and the predictability that Discrete Event Systems offer fall short of

⁷Even though this is a simplification of the behavior of a communication network, we are aiming at illustrating the ease of transformation into the formalism.

providing means of addressing these situations.

Issues of contexts switching, addressed in subsumption architecture by Connell, recognized by Brooks and pointed out in [Hartley and Pipitone, 1991], need to be addressed with much more efficacy. In addition, planning may be brought into the system as means of allowing the flexibility which is lost in the construction of a well-behaved but none-the-less fixed system. We will review some of the issues, more in detail, in section 8 when comparing Discrete Event System to Subsumption architecture.

8 Conclusions

This section focuses on the aspects of the two approaches and points out what can be gained from considering the individual architectures. The comparisons between Subsumption architecture and Discrete Event Systems have been summarized in table 22. This table reflects some of the major criterias for comparison. The discussions provided in the preceding sections have been used as basis for composing the table. However, new developments in research are addressing some of the limitations for Discrete Event Systems.

The two approaches considered are quite different. On the one hand, subsumption architecture requires limited knowledge of the environment, no explicit representation, limited reasoning capabilities, no centralized control. On the other, Discrete Event Systems, require, at least in manufacturing and communication⁸, a well-structured environment, explicit representations and models, have limited reasoning capabilities, and have centralized control.

The ability and need to model the environment is substantially different. In the DES approaches considered, every aspect and states should be well-defined in order to be able to explore and verify the validity of the controllability of the plant. In Subsumption Architecture, the representation is implicit with no need for an exact explicit model. However, it is necessary to be able to recognize and match the preconditions for a behavior to be active. This aspect is essentially similar to having the observation of an event triggering an immediate response manifested in a reactive action.

The two approaches focus on different means for carrying out tasks which have different type of constraints and goals. At the one hand, in Subsumption Architecture, the task does not require a precise model, can be reactive, afford long time delay, ignore some situations and possibly get stuck. On the other hand, in DES, when addressing a manufacturing plant, there are strict specifications defining the task, real-time constraints and dead-lines to be met, and errors may result in extremely serious consequences. Currently DES formalism is being applied to problems in navigation in which the modeling of the environment is not quite possible to the extent that it is possible in manufacturing. The other direction, is however not been investigated. Primarily because Subsumption Architecture is not well suitable for those environments. The construction of extremely specific and sensitive behaviors capable of reacting to small variations in the parameters could be undertaken to account for the multiplicity of states and type of specific control. This, however, would be contrary to the philosophy of the approach.

⁸We have previously noted that new directions are taking Discrete Event System to consider environments which are not as well-structured. [Košecká and Bajcsy, 1993] uses DES to control vehicle's navigation. [Bogoni and Bajcsy, 1993b] address the use of Discrete Event System formalism, though slightly altered, to model and observe functional tasks. [Bogoni *et al.*, 1994] present an approach for qualitatively controlling a vehicle pushing a box.

CRITERIA	SUBSUMPTION	DED SYSTEMS
basic units	behaviors	processes
construction	emergence	composition
language	object-oriented specification	formal language theory
description environemt	simple uncertainty	no uncertainty allowed for
representation	distributed and implicit representation	explicit and implicit representation
models	world at its on model	as required for process
control	inhibiting/suppressing	enable/disable via supervisor
mutal exclusion/ conflicts resolution	inhibiting/suppressing at construction of circuit	addressed as part of composition of processes
memory	distributed	central, distributed or shared
communication	simple buffers/ minimal message passing	as needed: event and vectors
computational engine	finite state machines with instance variables	finite state machines, augumented by memory
reliability	depending on environment	verification, as good as model
learning	limited to special cases navigation: landmarks	essentially non-existent
planning feed-forward	limited in scope to one step: navigation	no planning / fixed a priori new direction with navigations
adaptability	reactive and designed specific environment	designed for particular environment new direction with reactive systems
reconfigurability on line	contingency table control of behaviors	depends on ability of recognize contexts of behaviors
parametrization	requires new behavior	feasible and depending on application in general limited
feed-back	reactive	through observer and reactive
type observations	qualitative and absolute	qualitative : as events
ease of design	complex, not structured depends on skills engineer	methodology for constructing particular behaviors and control
implementation of simple design	immediate from defined finte state machines	requires mapping to sensors through interface
implementation of complex design	hard to accomplish not well defined	requires mapping to sensors through interface
complexity in terms of states	hundreds	millions

Figure 22: Comparisons between Subsumption Architecture and Discrete Event Dynamic Systems.

Much is to there to be said about the role played by modeling. Defining a model is not just a way of constraining the reasoning capabilities in an attempt to match and reconstruct the environment but can be used as a virtual sensor. The model may be used as a means of defining expectations in what is to be observed, and what should be reacted to at a particular stage in a task. It provides the contextual reference for the activity to be carried out. Connell pointed out the importance of having a contingency table in which predefined behaviors can be matched to contexts and switched in. When the task becomes more specific, such as in brain surgery, there is little room for robots trying to react to activated behaviors. In such situations one would like the next action to be based not only on emerging behaviors, but being motivated by a long deliberation and by being part of a plan. What, then, the task defines is the degree of feasibility and satisfaction required. Hence, the evaluation of the utilities and the cost involved in allowing the task to be carried out at a sub-optimal condition must be weighted.

When constraints are removed or not easily definable and as assumptions on the ability to model the environment decrease, the ability to control and resolve conflicts, a strong-hold of DES, is severely impaired. In such situations, adaptability must be brought into the picture. DES, as currently constructed, are not adaptable to a changing environment or capable of dealing with unexpected modifications or unknown situations. Means for expanding the system, possibly through learning, and to allow for uncertainty must be investigated. Research to address these needs is carried out in two directions: on the one hand, uncertainties are beginning to be introduced in a system modeled by DES [Sobh, 1991; Bogoni and Bajcsy, 1993a] and, on the other, hybrid systems are being developed [Nerode and Kohn, 1992].

Perhaps, it would seems inadequate to attempt to apply DES system in an environment clearly unsuitable for them. Yet, as we have seen from both [Ramadge and Wonham, 1989] and [Balemi *et al.*, 1991], there is a lot to be gained. In particular, DES offer the powerfulness that comes with the formalism of language theory and the ability of verifying the state space of the behaviors. It might be beneficial to take advantage of the strong points from both approaches. Low level behaviors could still be constructed using the mechanism of Subsumption Architecture and conflict and mutual exclusions could be arbitrated by controlling the lower level behaviors using the structure that DES provide.

Why combining two possibly incompatible systems? When dealing with an environment and by introspecting on human abilities, one can notice that there are clearly some behavior-based mechanisms for dealing with the unstructured world in which the complexities, due to the uncertainty, are avoided by resorting to a competition of behaviors and compliance to the environment. Not knowing about the locus of a surface on which to place an object can be described in terms of events without having to model the interaction. Furthermore, these events are triggered by

feed-back from the complying actions. At the same time, while depositing an object on a surface, unconscious adjustments are made by the hand to accommodate the object for stability. In such a situation several behaviors are cooperating and concurrent.

Combining these two approaches is clearly not going to address and resolve all issues for clearly there is more than behavior and control to intelligence! Other aspects dealing with planning, learning, an abstract reasoning must be dealt with. [Dean and Wellman, 1991] investigate different types of control methodologies and the role played by planning. Some of the major underlying differences in the approach taken by planning can be expressed by considering that Control focuses on modeling a behavior while Planning addresses the ability to influence a behavior. Additionally, tradeoffs are gained through the use of Planning in flexibility and adaptability to varying situations but at the same time there are losses in the ability to respond in real-time to deadlines in the interaction.

Having considered the two approaches we see that, in spite of the limitations which each one presents, they both offer benefits and tradeoffs which should really be taken advantage off when attempting to build a system. However, while it is possible to construct better and more general systems, the general purpose Robotic System is still out of reach and perhaps not the correct avenue to pursue as an immediate goal.

References

- [Abelson *et al.*, 1985] H. Abelson, G.J. Sussman, and J. Sussman. *Structure and Interpretation of Computer Programs*. MIT Press, Cambridge, MA, 1985.
- [Aloimonos, 1990] Y. Aloimonos. Purposive and Qualitative Active Vision. In *Proc. DARPA Image Understanding Workshop*, pages 816–828, 1990.
- [Aloimonos *et al.*, 1988] Y. Aloimonos, I. Weiss, and A. Bandopadhyay. Active Vision. *International Journal of Computer Vision*, 2:333–356, 1988.
- [Arnold, 1989] J. E. Arnold. Experiences with the Subsumption Architecture. In *Proceedings - Fifth Conference on Artificial Intelligence Applications*, pages 93–100, 1989.
- [Bajcsy, 1985] R.K. Bajcsy. Passive Perception vs Active Perception. In *Proceedings of IEEE Workshop on Computer Vision*, pages 55–59, 1985.
- [Bajcsy, 1988] R.K. Bajcsy. Active Perception. *Proceedings of IEEE*, 76(8):996–1005, 1988.
- [Balemi *et al.*, 1991] S. Balemi, G. Hoffman, P Gyugyi, H. Wong-Toi, and G.F. Frankling. *Supervisory Control of a Rapid Thermal Multiprocessor*. Technical Report, Information Systems Laboratory, Department of Electrical Engineering, Stanford University, 1991.
- [Bogoni, 1993] L. Bogoni. *Overseeing Multiple Active Observers*. Technical Report MS-CIS-93, Dept. of Computer Science, University of Pennsylvania, 1993. Submitted to Journal of Robotics and Autonomous Systems.
- [Bogoni and Bajcsy, 1993a] L. Bogoni and R. Bajcsy. An active approach to characterization and recognition of functionality and functional properties. In *AAAI Workshop: Reasoning about Function*, pages 9–16, 1993.
- [Bogoni and Bajcsy, 1993b] L. Bogoni and R. Bajcsy. *Characterization of Functionality in a Dynamic Environment*. Technical Report MS-CIS-93, Dept. of Computer Science, University of Pennsylvania, 1993.
- [Bogoni *et al.*, 1994] L. Bogoni, M. Salganicoff, G. Sandini, and R. Bajcsy. Multiple-context mobile robot control tasks using deds, qualitative image measures and adaptation. In *Submitted to AAAI Spring Symposium: Toward Physical Interaction and Manipulation*, 1994.
- [Brandin and Wonham, 1992] B.A. Brandin and W.M. Wonham. *Supervisory Control of Timed Discrete Event Systems*. Technical Report: Systems Control Group Report 9210, Dept. of Control Science and Engineering, University of Toronto, 1992.

- [Brooks, 1991a] R.A. Brooks. Integrated Systems Based on Behaviors. In *Proceedings, AAAI Spring Symposium on Integrated Intelligence Architectures*, pages 46–50, 1991.
- [Brooks, 1991b] R.A. Brooks. Integrated Systems Based on Behaviors. *SIGART Bulletin*, 2(4):46–50, 1991.
- [Brooks, 1991c] R.A. Brooks. Intelligence without Reason. In *International Joint Conference of Artificial Intelligence*, pages 569–596, 1991.
- [Brooks, 1991d] R.A. Brooks. Intelligence without Representation. *Artificial Intelligence*, 47:139–159, 1991.
- [Brooks and Mataric, 1993] R.A. Brooks and M.J. Mataric. *Robot Learning*, chapter 8 Real Robots, Real Learning Problems, pages 193–213. Kluwer Academic Press, 1993.
- [Brooks, 1986] R. A. Brooks. A Robust layered Control System For a Mobile Robot. *IEEE Journal of Robotics and Automation*, RA - 2.(1):14–23, March 1986.
- [Connell, 1989] J.H. Connell. A Behavior-Based Arm Controller. *IEEE Transactions on Robotics and Automation*, 5(6):784–791, 1989.
- [Connell, 1992] J.H. Connell. SSS: A Hybrid Architecture Applied to Robot Navigation. In *International Conference on Robotics and Automation*, pages 2719–2724, 1992.
- [Dean and Wellman, 1991] T.L. Dean and M.P. Wellman. *Planning and Control*. Morgan Kaufmann Publishers Inc., San Mateo, CA, 1991.
- [Hartley and Pipitone, 1991] R. Hartley and F. Pipitone. Experiments with the Subsumption Architecture. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 1652–1658, 1991.
- [Hopcroft and Ullman, 1979] J.E. Hopcroft and J.D. Ullman. *Introduction to Automata Theory, Languages and Computation*. Addison Wesley, 1979.
- [Košecká, 1992] J. Košecká. *Control of Discrete Event Systems*. Technical Report MS-CIS-92-35, University of Pennsylvania, Dept. of Computer and Information Science, Philadelphia, PA., 1992.
- [Košecká and Bajcsy, 1993] J. Košecká and R. Bajcsy. Discrete Event Systems for Autonomous Mobile Agents. In *International Workshop on Intelligent Robot Control, Zakopane*, pages 502–506, 1993.

- [Mataric, 1992a] M. Mataric. Behavior-Based Control: Main Properties and Implications. In *Proceedings of IEEE International Conference on Robotics and Automation, Workshop on Intelligence Control Systems*, 1992.
- [Mataric, 1992b] M. Mataric. Integration of Representation into Goal-Driven Behaviour-Based Robots. *Transactions on Robotics and Automation*, 8(3):304–312, 1992.
- [Nerode and Kohn, 1992] A. Nerode and W. Kohn. *Models for Hybrid Systems: Automata, Topologies, Stability*. Technical Report, Mathematical Sciences Institute, Cornell University, 1992.
- [Ostroff, 1992] J.S. Ostroff. *Tutorial on Specification of Time*, chapter Survey of Formal Methods for the Specification and Design of Real-Time Systems. ., 1992.
- [Ozveren and Willsky, 1990] C.M. Ozveren and A.S. Willsky. Observeability of Discrete Event Dynamic Systems. *IEEE Transaction on Automatic Control*, 35(7):797–805, 1990.
- [Ozveren and Willsky, 1991] C.M. Ozveren and A.S. Willsky. Stability and stabilizability of discrete event dynamic systems. *Journal of the Association of Computing Machinery*, 38(3):730–752, 1991.
- [Ramadge and Wonham, 1987] P.J. Ramadge and W.M. Wonham. Supervisory Control of a Class of Discrete Event Processes. *SIAM J. Contr. Optimization*, 25(1):206–230, 1987.
- [Ramadge and Wonham, 1989] P.J.G. Ramadge and W. M. Wonham. The Control of Discrete Event Systems. In *Proceedings of the IEEE*, pages 81–97, 1989.
- [Salganicoff, 1992] M. Salganicoff. *A Robotic System for Learning Visually-Driven Grasp Planning*. PhD thesis, University of Pennsylvania, 1992.
- [Sobh, 1991] T. Sobh. *Active Observer: A Discrete Event Dynamic System Model for Controlling an Observer under Uncertainty*. PhD thesis, University of Pennsylvania, 1991. Technical Report MS-CIS-91-99.
- [Stein and Paul, 1993] M. Stein and R.P. Paul. *Behaviour Based Control in Time Delayed Teleoperation*. Technical Report, University of Pennsylvania, Dept. of Computer and Information Science, Philadelphia, PA., 1993.
- [Wonham and Ramadge, 1987] W.M. Wonham and P.H. Ramadge. On the Supremal Controllable Sublanguage of a Given Language. *SIAM Journal of Control and Optimization*, 25(3):637–659, 1987.