N94- 35047

On-Board Emergent Scheduling of Autonomous Spacecraft Payload Operations

Craig A. Lindley

CSIRO Division of Information Technology
Locked Bag 17, North Ryde NSW Australia 2113

Abstract

This paper describes a behavioural competency level concerned with emergent scheduling of spacecraft payload operations. The level is part of a multi-level subsumption architecture model for autonomous spacecraft, and functions as an action selection system for processing spacecraft commands that can be considered as "plans-as-communication". Several versions of the selection mechanism are described and their robustness is qualitatively compared.

Keywords: Spacecraft Control, Emergent Scheduling, Behavioural AI.

Introduction

This paper describes an autonomous control architecture for scheduling payload and user service operations of a low Earth orbiting microsatellite, AUSTRALIS-1. The control architecture is based upon a behavioural paradigm of artificial intelligence (see Maes, 1993). Previous work has defined a layered competency model for autonomous orbital spacecraft (Lindley, 1993a), and the detailed design of the level 1 competency for maintaining the spacecraft battery condition (involving power system fault detection, redundant unit switching, charge control, and discharge control; Lindley, 1993b). Planning and Scheduling of Data Acquisition and Transmission is a much higher level competency (level 7), representing very atypical functions for behavioural systems (that have generally addressed low-level robot motion control and guidance functions). Emphasizing the interface between the autonomous system and its human users, the level 7 competency takes into account specifically requested data objects, user requested instrument parameters, and the need to downlink particular data sets at particular times or locations. These needs are catered for while considering the battery charge requirements of Level 1. A *plan* in this system consists primarily of the current set of requests for user operations, generated by users and uplinked to the spacecraft. Hence "a plan" is a communicable list of representations of goals. The term "planning" can be used to refer to the generation of goal lists, or can be used more loosely to refer to the generation of activity that satisfies a set of current goals. A *schedule* is taken to be an association of specific activities with particular times, and hence is a more detailed form of planning in the loose sense. By these definitions, planning and scheduling can be emergent phenomena in that they do not need to involve the generation of action representations. A goal selection algorithm is described that achieves emergent planning and scheduling by choosing a particular represented goal as the basis of action generation with each control cycle of the system. Several versions of the algorithm are described, and their relative merits are qualitatively discussed.

Precedents for Autonomous Planning in Space Systems

Command languages for spacecraft can be viewed as languages for expressing plans that are uploaded to and then obeyed by a spacecraft. Increasing abstraction levels in spacecraft command languages (Pidgeon et al, 1992), or reducing command detail, requires greater spacecraft "intelligence" in the form of the autonomous planning ability needed to elaborate the details of uplinked plans to the appropriate level for the control of spacecraft hardware. For example, in normal operations, the Hipparcos spacecraft is controlled by processed commands that are sent to the onboard computer for distribution to other systems, and for possible time tagging. The ERS-1 spacecraft, which is in a low polar orbit with limited ground access, has a similar command macro system, with four command types providing different functions and levels of authority. The EURECA system, comprising fifteen separate payloads, uses an

onboard Master Schedule that contains a list of time tagged command macros for execution by the onboard data handling system. Further increasing the abstraction level of commands, or further decreasing their level of detail, involves the incorporation of more sophisticated techniques for autonomous planning and scheduling, drawn from research in artificial intelligence (AI).

Representational AI systems, or Knowledge-Based systems (Maes, 1993), use symbolic models of the robot, its operating environment, and the range of tasks and actions that the robot "knows how to perform". System behaviours are typically produced by an inference engine that reasons about the models to produce action plans. Plans and action representations are frequently hierarchically ordered, with high level representations of goals and actions at the top of the hierarchy. Subsequent levels of abstraction decompose goals into increasingly specific subgoals, ending with primitive machine commands that can be executed directly by hardware. Robotic plans are monitored during execution. A robot may attempt to deal with plan failures by replanning from some suitable level of abstraction, with various techniques being employed to deal with constraints of real-time operation. Brooks (1986) has described this traditional, representational approach as the sense/model/plan/act approach. Representational approaches (reviewed in Georgeff, 1987) have made substantial progress in addressing the requirements of many applications, but frequently founder in the inability of system designers to provide sufficient knowledge for the autonomous performance of useful tasks in real environments.

Most space applications of AI to date have concentrated upon the areas of mission planning, sequencing, and control. There have been numerous projects that have addressed the automation of ground-based planning and control of space systems, with many systems successfully prototyped, and a number now in routine operational use (Drabble,1991). These ground-based systems have been developed within the traditional, representational AI paradigm. Autonomous orbital spacecraft, surface exploration robots, and dextrous free-flying robots have particular requirements for producing appropriate behaviour quickly in the face of dynamic and uncertain circumstances. These issues have been addressed using techniques representing a convergence towards some of the techniques involved in behavioural artificial intelligence. Fast response times have been achieved by separating the control architecture into parallel functions, with a combination of deliberative planning and reactive planning being used to achieve goals at strategic and tactical levels, respectively (eg. Rokey and Grenander, 1990, and Erickson et al, 1989). Autonomous orbital spacecraft control models have adopted limited parallel and distributed processing models, but have still tended to rely upon world modelling as the basis of their intelligence (eg. Raslavicius et al, 1989).

Hierarchical approaches to action representation and control have dominated the field of spacecraft automation, and have been incorporated as fundamental structuring principles in proposed reference models for space automation and robotics developed both by NASA and ESA (Elfving and Kirchoff, 1991). The "Theory of Intelligent Machines" proposed by Valavanis and Saridis (1992) associates hierarchy with proposed fundamental analytical measures of intelligence, and the Rensselaer Polytechnic Institute Center for Intelligent Robotic Systems for Space Exploration has constructed a testbed that has the development and implementation of this theory as one of its primary purposes (Watson et al, 1992).

Behavioural theories of artificial intelligence suggest a very different approach to autonomous action generation and control, requiring new reference models for robotic systems (Lindley, 1993a) and new analytical formulations that do not intrinsically depend upon a hierarchy of functional or control flow (Lindley, 1993c). Maes (1993) has described the following characteristics of behavioural systems:

- systems have multiple, integrated, and typically low-level competences
- the system is "open" or "situated" in its environment, having many interactive interfaces with a complex, dynamic, and unpredictable world
- the emphasis is upon autonomy
- systems are designed to produce behaviour, rather than to have knowledge in a

representational sense, thus avoiding the *circumspection problem* of providing sufficiently comprehensive world models for the tasks at hand
- particular functions may emerge from the activity of more primitive behaviours
- there is a strong emphasis upon adaptation

Lindley (1994) has argued that behavioural approaches are justified by philosophical positions that regard represented knowledge as a cultural, discursive, and pragmatic artefact, rather than being the substance of intelligence; in effect, knowledge is reified practice, codified by representation in order to coordinate behaviour. From this viewpoint, *action*, rather than representation, is at the core of intelligence. Behavioural theories constitute a new and vigorous paradigm that has shown superior performance to representational methods in the control of simple robotic functions (Brooks, 1991).

Planning and Behavioural Action Selection

Agre and Chapman (1990) attribute problems with the representation-centred view of "plans-as-programs" to a mistaken notion of what activity is like, and the role plans can play in activity. The plan-as-program view regards activity as a matter of problem-solving and control. The world presents an agent with a series of formally defined problems that require solutions, and a planner produces the solutions. The executive then "implements these solutions by trying to make the world conform to them". Agre and Chapman hold that this view is incorrect, since "Acting in the world is an ongoing process conducted in an evolving web of opportunities to engage in various activities and contingencies that arise in the course of doing so". Hence an agent can be viewed as participating in the flow of events, rather than solving problems.

This view leads to the notion of *plans-as-communication* as part of a general theory of situated activity. Plans-as-communication have a reduced role, with the need for improvisation on the part of the user. A plan is used as a resource among other resources in a process in which an agent engaged in rational, goal-directed activity continually reevaluates what to do. Plan use is a matter of figuring out how to make a plan relevant to a situation at hand. The meaning of the plan is heavily dependent upon the context, and must be constantly reassessed. Plans-as-communication have a fuzzy boundary with other forms of communication, such as lists, schedules, images, and mnemonics. The current context of an agent is a resource that can be used to intepret the plan. Agre and Chapman present the example of a person following a plan which has the form of another person's instructions for how to reach a particular location. Lessons generalised from the example are:

- the list of shared understandings is innumerably long
- all plans depend on shared understandings in this way
- action in the real world is sufficiently difficult to suggest that plans must depend on innumerable shared understandings in order to be expressible
- these points apply regardless of whether the plan's maker and user are the same agent or different agents

Attempts to capture the meaning of plans by representation must therefore encounter the circumspection problem of needing to represent an unspecifiable amount of contextual knowledge. Agre and Chapman suggest that the plans-as-communications view is more plausible than plans-as-programs as an account of human plan creation and use, and may provide a more effective model for artificial autonomous agents, although its tendency to appear like the general problem of automated natural language comprehension makes it impractical for the design of autonomous agents in the short to medium term. However, this is overly pessimistic, since the ability to understand (ie. use) limited forms of plans for particular purposes in a specific domain can be a much simpler competency than the general-purpose natural language skills of human beings.

An agent capable of processing plans-as-communication can be designed using a behavioural goal selection system for action generation without deliberative reasoning. Maes (1990) suggests that the action selection mechanism of an autonomous agent in a complex

dynamic environment should be reactive and fast, favouring actions relevant to the current situation, exploiting opportunities, and adapting to unpredictable and changing situations. The mechanism must favour actions that contribute to any current ongoing goal or plan, and should look ahead, especially to avoid hazardous situations and to handle interacting and conflicting goals. The action selection mechanism must also be capable of operating with minimal, incomplete or incorrect world knowledge, with limited computational and time resources, and be robust (degrading gracefully when components fail).

A goal selection mechanism proposed by Maes (1990), that appears to have these desirable characteristics, views an autonomous agent as a set of modules each having its own specific competence. These modules resemble the operators of a classical deliberative planner. A competence module has a list of preconditions to be fulfilled before the module can become active, lists of expected effects of the module's action in terms of an add list and a delete list, and a level of activation for the module. If all the preconditions of a module are true at a particular time, then the module is executable at that time. Competence modules are linked to form a semantic network, with activating and inhibiting links between modules allowing the activation energy to accumulate in the modules that represent the "best" actions to take in the current situation and given the current set of goals. The pattern of spreading activation among modules, and the input of new activation energy into the network, are determined by the current situation and the current global goals of the agent. Activation iterates through cycles, thereby accumulating until a threshold is exceeded and an action is performed. A decay function ensures that the overall activation level remains constant through continuous iterations.

Initial results with this approach are reported to be very encouraging, with networks exhibiting planning behaviour. Plans are not explicitly represented, but the "intention" of an agent to carry out certain actions is expressed by high activation levels of the corresponding modules. Action selection is non-hierarchical and highly distributed. Global parameters serve as controls for mediating smoothly between different action selection characteristics, and therefore between adaptivity, speed, and reactivity on the one hand and "thoughtfulness" and certainty on the other. The approach is computationally much less expensive that search-based planning. Different paths are evaluated in parallel, and the system does not start from scratch when a particular path fails to produce a solution, nor "replan" at each timestep. The action selection mechanism is capable of processing complex goal interrelationships, and complex, possibly hierarchically organised goal substructures, although particular applications may have very simple, flat goal representations with few interconnections.

AUSTRALIS-1 Spacecraft Operations

The AUSTRALIS-1 spacecraft is currently being designed by an informal consortium of Australian universities. The baseline spacecraft is a 35 cm cube, having an expected mass of less than fifty kilograms. It is intended to operate within an altitude range from five hundred to one thousand kilometres. The spacecraft will carry a near infra-red CCD-based camera system, and telecommunications equipment to support a data store-and-forward communications service. AUSTRALIS-1 has particular requirements for autonomy derived from the need to serve a large number of users over a highly dispersed area, using very cheap and simple ground equipment with minimal centralised control or coordination.

The spacecraft subsystems most directly associated with supporting the level 7 user operational model are shown on Figure 1. The controller initiates image acquisition operations and controls downlinking of images and data units via the CMS (Command Management System) database. Hence, the level 7 control network functions at what will here be called a *transaction* level; that is, it is not concerned with detailed data transfer and timing control between devices, but with control of the transfer of complete data units.

The flow of user data between components does not pass through the level 7 controller, although operation selection and initiation is controlled by this level. The level 7 controller receives inputs from the relevant subsystem components, from the guidance and navigation system, and from the

level 1 competency concerned with battery conditioning, charge control, discharge control, and power control. The AUSTRALIS-1 power control system is described in detail in Lindley (1993b).

AUSTRALIS-1 users will be able to uplink data files (by *Store Data* command), request data broadcast (by *Broadcast Data* command), request image acquisition (by *Acquire Image* command), and request data deletion (by *Delete Data* command). The commands are stored on board the spacecraft, and function as goals within the payload planning and scheduling competency of the autonomous control system. In the data store and forward mode, ground stations can uplink a request for the spacecraft to broadcast data immediately, or broadcast to a distant ground station specified in terms of a latitude and longitude or a time. During image acquisition operations, the spacecraft receives an image acquisition request from a user, schedules and acquires the image, and then treats the image data in the same way that user data packets are treated.

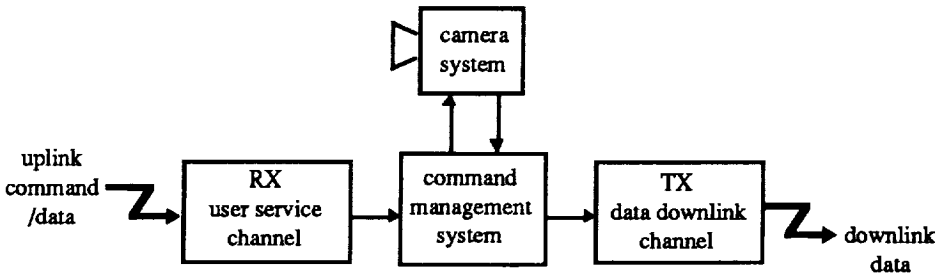

Figure 1. Level 7 subsystems.

Stored data is held in an onboard database within the Command Management System (CMS) for downlinking to specified destinations. Stored data can be deleted upon explicit user request. In all of these transactions the spacecraft will schedule and execute operations without coordination or mediation by a central ground station or command and control network. That is, user stations will interact directly with the spacecraft.

The Command Management System (CMS) consists of a command execution controller, a database management system (dbms), and a database residing in bank switched memory. The CMS command execution controller and other levels of the control system handle detailed protocols and data exchange between devices, allowing the level 7 control competency to be defined at a supervisory control level. The CMS database contains user data packets in addition to user commands, and maintains a file listing all data packets and their parameters or header information. This directory file is treated as a data packet with an identifier of 0. It can be downlinked (with filtering based upon user priority), but cannot be deleted. The directory is updated automatically by the CMS dbms.

A Store Data command has an associated data packet. Upon receipt of a Store Data command, the data packet is stored in a database within the CMS, along with command parameters. A data identifier and descriptor are entered into the directory file within the CMS database. Store Data commands are processed immediately if accepted (ie. within one processing cycle of the control system) to initiate data input to the CMS database from the receiver (channel) for eligible commands. Detailed timing of data storage transactions is not modelled at this level. If the level 7 control cycle time is fast enough, a Store Data command can be validated prior to uplinking any of the associated data to the spacecraft. If the cycle time is too slow for this, uplinked data can be buffered prior to transfer to the CMS database if the command is accepted, or deleted if the command is rejected.

Upon receipt, a Broadcast Data command is stored in the database within the CMS. As specified by the command parameters, broadcasting can be initiated immediately, at a specified time, or in the proximity of a particular target point specified by latitude and longitude. The Broadcast Data command includes parameters specifying the data to be broadcast.

A Delete Data command includes parameters specifying the data to be deleted from the CMS database. Deletion will occur only if the command

carries appropriate authorisation. When data is deleted, the associated data identifier and descriptor are removed from the directory file within the CMS database.

Upon receipt, an Acquire Image command is stored in the CMS database. The command includes all appropriate image acquisition parameters. Execution of an Acquire Image command is triggered when time or positional parameters are satisfied, and subject to resource availability. Once an image has been acquired, it is stored as a data unit in the CMS database, along with its parameters and acquisition details. An image identifier and a descriptor are entered into the directory file within the CMS database.

Level 7 Subsystem Interface Definition

Subsystem Sensors

Each subsystem or component has a range of data outputs, which may include sensor values and computed function values. These outputs are treated as sensors monitored by the level 7 controller. Sensors classified by associated component are as follows:

Command Management System (CMS):

CA	command arrival, 1/0
CREC	command record
BD	broadcasting data via tx, 1/0
SD	storing data from rx, 1/0
CMA	active memory blocks
CM	total memory blocks

Guidance and Navigation System (GNS):

SClat	spacecraft latitude
SClong	spacecraft longitude
SCalt	spacecraft altitude
SCtime	spacecraft system time

CREC records are derived (within the CMS) from information provided by the user. Information associated with all commands upon initial receipt includes: command identifier (CID), user priority (UPRI), command priority (CPRI), and command type (CMD).

Further information within Acquire Image CREC input includes the target latitude and longitude or an image acquisition time, and the image data size. Store Data CREC inputs include the data size and the data itself. Broadcast Data CREC inputs include a data identifier, data size, and the downlink latitude and longitude or a downlink time. A data identifier of 0 indicates the CMS database directory file. Delete Data CREC inputs include a data identifier, and a delete latitude and longitude or a delete time. The level 7 CMS database user data storage format for each data file includes the data identifier, the identifier of the user who supplied the data, the data priority, the data size, and the stored data. A datablock generally includes command parameters from the initial Store Data or Acquire Image command associated with the data record.

Actuators

Control signals (including combined control and data output signals) are:

Command Management System (CMS):

Command id, CID	integer
Control enable, CE	1/0
Command Arrival Acknowledge, CAAK	1/0
Active Memory Blocks, AMB	2 bytes
Operation, CMS_OP	2 bits:
delete data	00
acquire image	01
store data	10
broadcast data	11

Transmitter:

tx_baud	2 bits

(encoding 0, 1200, 4800, or 9600 bps)

Controller Design

The overall goal of the level 7 competency is to maximise the provision of user services throughout the lifetime of the spacecraft. Hence the goal is to maximise the overall throughput of data in response to user demand. Users and their data requests are prioritised, and higher priority throughput is more important than lower priority throughput. Hence, the goal of this level can be expressed as the maximisation of the value function:

$$Q = \sum_{i=1}^{k} S_i \cdot P_{ui} \cdot P_i / P^2_{max} \quad (1)$$

where k is the number of data packets downlinked over the lifetime of the spacecraft, S_i is the size of the ith data packet downlinked during that time, P_{ui} is the priority of the user who requested data packet i, P_i is the priority requested by the user for data packet i, and P_{max} is the maximum possible user and data priority. P_{max} is an *a priori* constant established by convention. S_i, P_{ui}, and P_i appear to the controller as stochastic variables within respective predefined range limits. The controller has some control over the product $S_i \cdot P_{ui} \cdot P_i$ for the command i executed on any given occasion if there are a number of candidate commands from which to choose that have different values of $S_i \cdot P_{ui} \cdot P_i$. Then, other things being equal, the controller can maximise the value function by choosing to process the command with the maximum product $S_i \cdot P_{ui} \cdot P_i$. It is the goal selection system of the level 7 competency that has the role of maximising Equation (1). Inputs to the goal selection system include command goals representing demands for power and state variables indicating the power available after battery charging (from level 1 of the control system, concerned with power control). The amount of power available constrains the data throughput rate, and also constrains command execution by command type.[1]

There are many ways of defining and implementing a goal selection algorithm, manifesting varying features and degrees of a behavioural approach. Several alternative goal selection systems are described here, compared, and evaluated qualitatively in terms of that aspect of their *robustness* that concerns the tendency of the system to return to proper functioning following disruption due to loss of hardware and/or control logic.

[1]More generally, there is a tradeoff between the throughput and the overall lifespan of the spacecraft. This complicates the optimisation problem, but the complication is avoided here by designing the spacecraft to operate with an average power consumption over a fixed design lifetime of four years.

Design #1: A Procedural Algorithm

The algorithm for a single cycle of goal selection can be written as a conventional procedural algorithm:

1. select eligible commands from the general command list, according to:
   - satisfaction of latitude, longitude, and time constraints
   - Store Data commands are not eligible if a command is currently being stored (ie. SD is asserted)
   - Broadcast Data commands are not eligible if a command is currently being broadcast (ie. BD is asserted)

2. set best goal = NIL

3. set power available = array energy remaining - battery charge energy needed (from Level 1)

4. for each eligible command goal i
   if the command is a *Delete Data* command then
      - execute the command
      - remove the command goal from the controller goal list
   else
      calculate Q_i = UPRI . CPRI . Sd
      if the command is a *Store Data* command then
         if memory required ≤ memory available (that is, Sd ≤ (CM - CMA)) then
            $Cost_i$ = receiver power + memory power
         else
            $Cost_i$ = greater than max power
         end if
      else if the command is an *Acquire Image* command then
         if memory required ≤ memory available then
            $Cost_i$ = camera power + memory power
         else
            $Cost_i$ = greater than max power
         end if

```
    else if the command is a Broadcast Data
    command then
      for each selectable transmission bit
      rate j
            Cost_ij = transmission power for
                  rate j
      end for
  end if
  if Q_i > Q_best goal then
      if command i is a Broadcast Data
      command then
            set j_max = max j for which
              Cost_ij < power available
            if j_max is not zero then
              set tx_baud = bit rate j_max
              set Cost_i  = Cost_ij_max
            else
              Cost_i = greater than max
                    power
            end if
      end if
      if Cost_i < power available then
            set best goal = goal i
      end if
  end if
end for
5. execute best goal
```

This system differs from the goal arbitration and selection network described by Maes (1990) in that, unlike Maes' system, command goals do not have any represented dependencies, substructures of subgoals, or interrelationships; they are treated as independent requests for system resources. Also, the goal list used here is a dynamic structure that may vary in size from 0 to the maximum number of goals that the system memory can hold. The value, Q_i, of each goal i is similar to the activation level used by Meas.

The content of goals (within certain standard formats) and the frequency of goal arrivals are determined by the dynamics of the system environment. The goals have a critical role in determining the value of the autonomous system, since the ultimate purpose of the spacecraft is to satisfy its users on the ground by satisfying their requests communicated to the spacecraft in the form of the goals. However, the goals are partial in the sense that maximising the usefulness of the system over its projected lifetime requires balancing user-generated goals with other goals concerning the health and status of spacecraft subsystems and components. The overall goal structure of the spacecraft control system is partially defined by the levels of the multi-level competency model (see Lindley, 1993a), partially emergent (overall performance is maximised by many competencies acting together but achieving numerous local subgoals), and partially explicit in the form of the command goal list.

Advantages of this procedural algorithm include overall simplicity and computational complexity that is linear in the number of eligible goals. The primary disadvantage is that the procedure has no intrinsic robustness against failure, because it is written according to a model in which a single active process interprets a variable-length list of passive goals.

Design #2: A Declarative Knowledge-Base

The goal selection system is a behavioural approach to goal selection in that it achieves planning and scheduling without world modelling. However, the system can in principle be implemented as a declarative knowledge base, representing a less paradigmatic example of a "behavioural system". The knowledge base can be arbitrarily complex, since there is no limit to the amount of knowledge about the spacecraft and its world that can in principle be modelled. The simplest usable solution is to code the procedural knowledge expressed in the procedural algorithm using a declarative representation language. This will ensure that no knowledge is represented that is not immediately relevant to the solution to the arbitration problem, and avoids the circuspection problem by implementing a behavioural goal selection mechanism that does not depend upon processing a world model. However, such a "knowledge-based" solution is at least as complex as the procedural solution, and will result in little generality - it is very unlikely that any of the knowledge will be reusable for any other purpose.

A declarative solution suffers from the robustness problem of the procedural solution if it depends upon a single inference process. Inference has the further disadvantage of realising the procedural algorithm in an inefficient and "unnatural" way as a search pattern over the declarative model of the algorithm. This inefficiency is avoided by

preunifying or compiling the declarative model into a "flat" rule base that can be executed without search, but at the expense of memory. The declarative solution can then be used to address some aspects of the robustness needed of the control system. In particular, in mapping the controller design onto available physical processors, the natural modularity of the declarative controller definition can be used to adaptively remap control functions across physical processors according to possible competing demands for processors or variable processor availability due to failures or power limitations (Lindley, 1994).

This method can result in a graceful degradation of temporal efficiency in controller execution with decreasing total processing power, and allows the executing controller to be completely rebuilt, if necessary, at any execution cycle (with the loss of some dynamic state information). In general, critically diminishing processor power can be managed systematically by eliminating control functions from the "top down" in a layered control system such as the subsumption architecture, thus eliminating competencies in decreasing order of criticality.

Deficiencies of this approach include the need to maintain a robust and reliable copy of the declarative definition of the control system, and the need for a robust and reliable monitoring and remapping process. The proposed solution is therefore not a general one, since the problems of robustness and reliability are shifted onto those features of the system designed to address the reliability and robustness of the executing controller.

For the proposed strategy to avoid endless regression, it is necessary to reach a point of control design that is *intrinsically* robust and reliable. If intrinsic techniques for achieving robustness and reliability can be defined, parsimony suggests that those techniques should be built into the initial model of the autonomous control system, rather than any level of meta-interpreter. It is the search for intrinsic reliability and robustness that leads to behavioural implentation strategies. As suggested by Lindley (1994), a behavioural control model can be specified declaratively, and formal properties of the syntax can aid design integrity and support verification. However, the use of declarative specifications is neutral in regard to other general characteristics of a design, and adds little to the search for principles of intrinsic robustness.

Design #3: A Subsumption Layer

Taking the basic sequence of operations defined in the procedural algorithm as a sequence of behavioural components, and allowing the messages passed between behavioural components to include lists of complex data items, the procedure can be represented as a behaviour network in the graphical notation used to describe layers of the subsumption architecture, as shown in Figure 2.

In this Figure, list-valued messages are represented by bold arrows. Each box represents a transformation of the data content of the messages passed by input arrows into the box to the data content of messages passed by output arrows. The function and result of the algorithm expressed in this form are the same as in the procedural algorithm. The only substantial change from the procedural algorithm is that instead of iteratively applying the sequence of operations to each command in turn and testing for the best command so far on each iteration, a single sequence of operations is used with each intermediate step in the sequence receiving a list of commands as its input, performing an operation on the list, and producing a new list (and/or single command) as its output. The network is activated once per control cycle, where the inputs (represented by elliptical sensor signal sources) are fixed (sampled) at the start of each cycle and the cycle time is long enough for the outputs (represented by elliptical actuator signal sinks) to stabilise by the end of each cycle.

Advantages of this version of the emergent planning and scheduling competency include those of the procedural algorithm, that is, simplicity and computational complexity that is linear in the number of commands. Additional advantages that might follow from implementation as a *single level* of a subsumption architecture using the mechanisms described by Brooks (1986), but *excluding* the general advantages of behavioural systems, include:

- each block in Figure 2 can be implemented as an augmented finite state machine (AFSM), minimising the complexity of its implementation
- blocks intercommunicate asynchronously, thereby simplifying their interfaces
- each block can be implemented as a separate physical process, thereby improving the robustness of the network against processor failure

The primary disadvantage of this model is that each block is critical to the correct operation of the overall competency, so the failure of a block (or its physical processor) will result in the failure of the competency. Remapping functional components (ie. blocks) to processors can redress this deficiency, but only at the cost of the regression of the robustness requirement (beginning with the remapping process), as in the case of knowledge-based inference processes. The basic subsumption mechanisms as described do not provide intrinsic robustness and fault-tolerance.

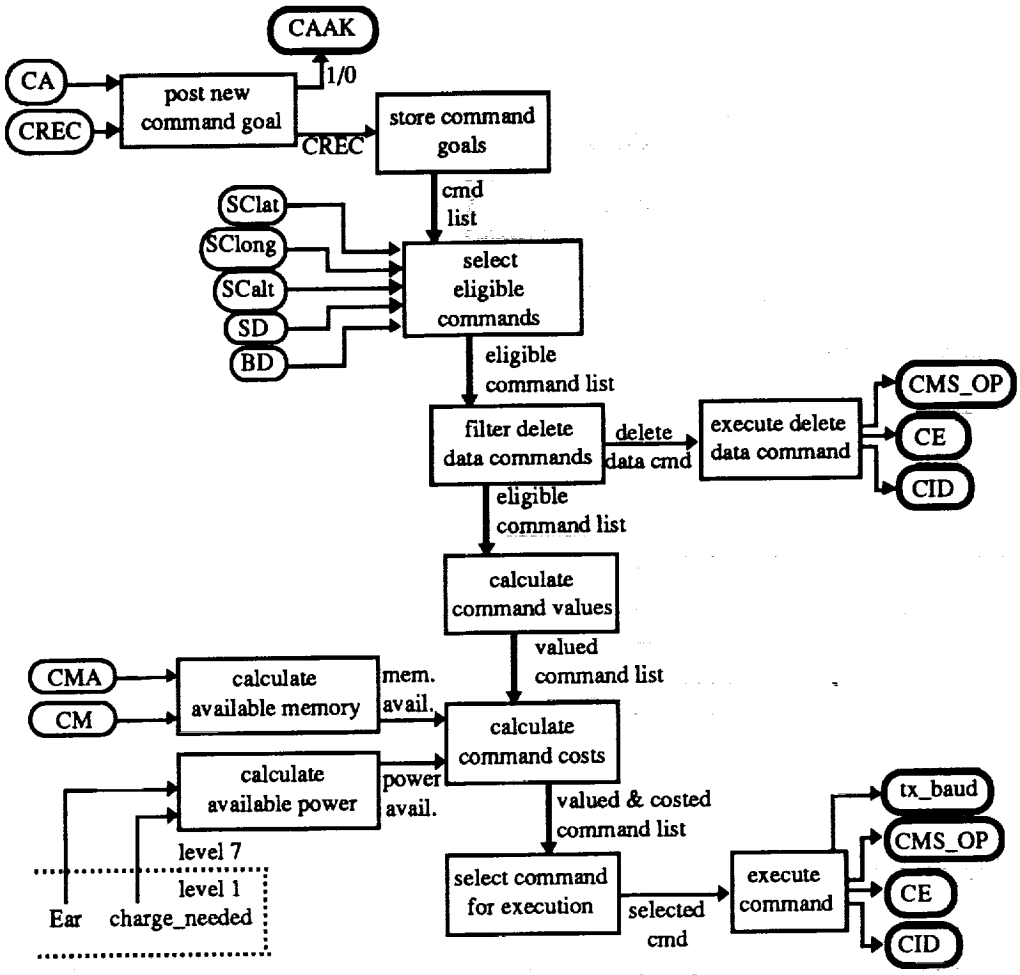

Figure 2. A subsumption level.

Design #4: Distributed Data

A major characteristic contributing to the robustness of a system is that incremental physical element failures merely degrade overall performance. A first step towards this characteristic is to distribute data structures across multiple hardware components. The list-processing functions of the subsumption layer shown on Figure 2 accept lists as input messages

and/or generate lists as output messages. A more robust approach would be to carry out the sequence of operations using a separate data structure to represent each command. The transformations carried out by the operations can be represented as a sequence of transformations of the state of each command structure, until the final selection of a command for execution. This approach is modelled on Figure 3.

Posting new goals is assumed to be asynchronous, although only goals that are present at the beginning of a control cycle are processed within

global memory area that is dynamically mapped onto multiple, modular hardware memory elements by the *post_new_command_goal* component. The wide arrows on Figure 3 indicate multiple independent data paths, and the double-headed paths indicate bidirectional transfer. Actual data transfer for each operation may be parallel or sequential. In the case of parallel transfer, each functional block acts as an SIMD (Single Instruction Multiple Data) processor, speeding up execution of the function. If data transfer is sequential, each block functions iteratively like the blocks of the subsumption

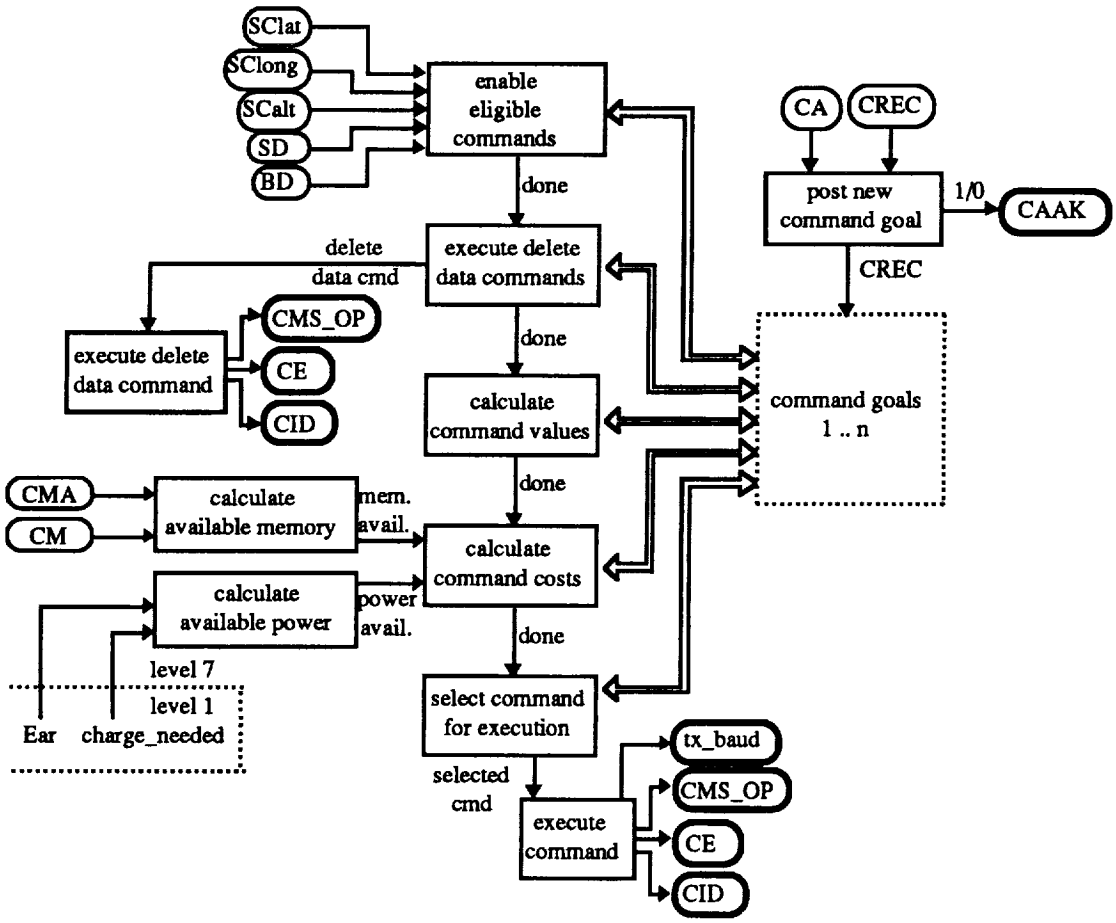

Figure 3. Controller with distributed data.

that cycle. The procedural sequence is then implemented beginning with the *enable eligible commands* function, with each subsequent function being initiated by the "done" message from the function preceding it in the sequence. The command goals themselves are held in a

layer described in Design #3, but iterating through transformations of shared data structures, rather than processing separate lists.

Each command data structure can be mapped onto a single physical memory unit (or n command

structures can be flexibly mapped onto m physical memory units), so that failures of physical memory units result in a degradation of processing or storage capacity, but do not result in complete loss of functionality. The loss of memory units must be detected by the *post_new_command_goal* component, and that component then assigns new goals to the remaining memory elements. The only loss incurred by the failure of a working memory component is the current set of goals stored by that component. This distributed data solution has superior robustness to the preceding design solutions, but still has no intrinsic robustness against failures of transformational modules.

Design #5: Distributed Data and Distributed Functionality

evaluation functions to be embedded in an iterative algorithm. A simpler solution is to combine the functions into a monolithic goal arbitration module, as shown on Figure 4. Goals can be presented to the network in parallel. While the number of any given goals at any particular time is variable, that number is always in a range from zero to a fixed upper bound (as limited by storage space). The network can be designed for the maximum possible number of goals, with unused inputs assigned to null values and their associated goals having zero probability of selection.

This design will allow distributed data storage and distributed functionality, supporting robustness against hardware loss in both cases by the association of particular data structures and particular connectionist nodes (or sets thereof)

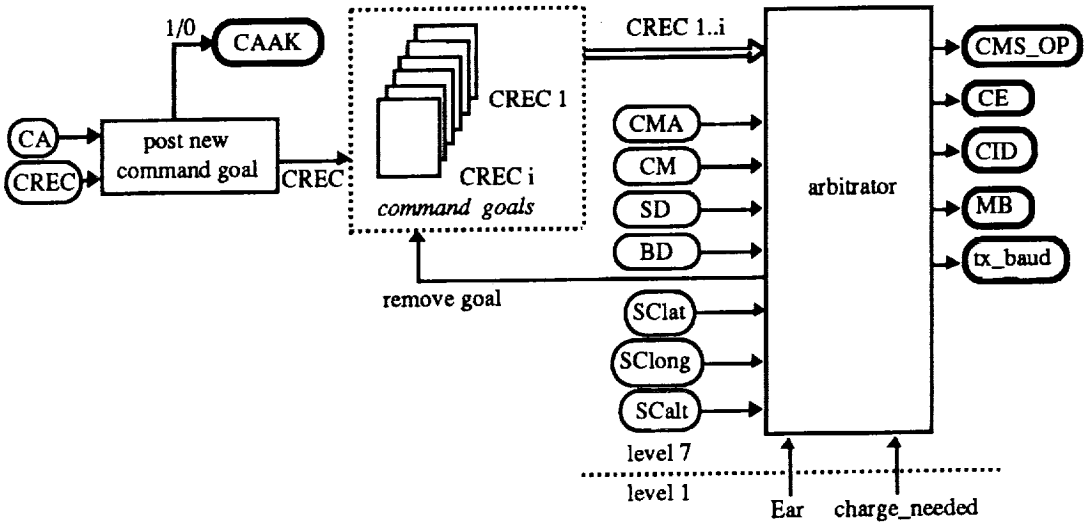

Figure 4. Distributed data with a monolithic arbitrator.

One method of achieving distribution of functionality is by implementing each functional module of the emergent scheduling algorithm as a connectionist network. That is, the transfer function implemented by the module can be implemented as a multi-layer network having an input layer for the inputs to the module, at least one hidden layer, and an output layer. Robustness is achieved because such networks are, in principle, highly tolerant of the loss of intermediate layer nodes. Iterative applications of the basic operations would require connectionist with different physical units. Hardware component failure will degrade performance, but will not immediately halt functionality unless a critical number of components are incapacitated. The detailed degradation characteristics for the goal selection task are currently unknown.

This form of solution may be the most robust considered here. The primary disadvantage of this approach is the need to create the weighted network of the arbitrator. A learning network could use a database of examples to learn from, if

available, or could learn in real time if a suitable reinforcement signal can be defined. The feasibility of this approach is a subject of ongoing research. A more desirable solution would be to derive a robust definition of the network from a formal specification of the transfer function that it is to implement.

Design #6: Distributed Objects/Agents

The final design option considered here is that of integrating data and functionality into discrete computational "objects", each of which represents a particular goal, and each of which can be allocated to a dedicated physical computational component. "Posting" a goal is then a matter of instantiating a goal object, based upon a common class or subclass (according to command type) definition but with particular characteristics defined by data unique to that command instance. Each goal has functionality for calculating its own activation level, based upon sensor inputs, state variables from other competency levels, and its own data values. Sensor inputs and state variables define a virtual environment within which goals compete for execution. The final selection is based upon the overall activation value of each separate goal, and so is a very simple mechanism that can be made robust by redundancy. If a hardware component is lost, the goals that depend upon that component will be lost; all other goals within the system will persist, with their own state and functionality intact.

A serious disadvantage of this approach is the need to robustly maintain a prototypical object or class definition, and a robust instantiation mechanism. This results in a similar regression of the robustness requirement to that seen in the case of a declarative solution (Design #2 above). A strategy for reducing this regressed requirement might be to reduce object functions to a more primitive set of operations, with overall goal selection occurring as an emergent phenomenon of the primitive interactions of goal objects. This is an area of ongoing investigation.

There are many ways in which a fully distributed system might be defined, based upon an object analogy, or upon analogies of processes or agents. Each analogy emphasises different aspects of the behaviour and interaction of goals within the system. Clarifying the variety and respective merits of distributed approaches are important themes of ongoing research.

Conclusion

This paper has described an approach to the autonomous on-board scheduling of spacecraft payload operations, based upon behavioural action selection and upon processing plans as communications. Several variations of the action selection mechanism have been described and evaluated comparatively in terms of robustness against hardware component loss. It has been seen that increasing the distribution of data and functionality can provide greater robustness, validating behavioural approaches to artificial intelligence that seek greater overall functionality and robustness via low level distributed functions that do not rely upon high level representation models. Designs that minimise representation and distribute both data and functionality have the greatest potential robustness, and manifest the most distinctive features of behavioural systems.

The spacecraft model and control designs described in this paper are simplified in a number of ways. Detailed protocols for modelling possibly sequential input of formatted sensor values are not considered. The applicability of on-board image compression has not been considered in detail, although compression could yield substantial benefits in increasing virtual storage capacity and data throughput. It is assumed that FDIR functions are carried out within the level 5 competency (Acquire, Condition, and Downlink Instrument Data); without a detailed model of that level, the integration of operational planning and scheduling with FDIR functions is not clearly defined. This paper has not considered downlinking the contents of the command database to users, acknowledgement of command execution, or elimination of commands that cannot be executed. Finally, the quantisation of input and output variables is not considered, either in the definition of those variables or in the definition of functions that accept or define them, respectively. These simplifications are used to clarify the central issues in developing a behavioural control system for the autonomous scheduling and planning of user services, and do

not invalidate the overall analysis or conclusions obtained.

Current work is addressing the validation of the goal selection mechanisms in a spacecraft simulator, and the development of a more rigorous and quantitative characterisation of the merit of the different goal selection systems, accounting for their overall complexity in addition to their robustness. Ongoing research is extending this work to determine how greater levels of autonomy and robustness can be achieved by incorporating methods for learning, adaptation, and self-organisation into behavioural systems.

References

Agre P. E. and Chaman D. 1990 "What Are Plans For?", in Maes, P. (ed) *Designing Autonomous Agents: Theory and Practice from Biology to Engineering and Back*, MIT/Elsevier, 17-34.

Brooks R. A. 1986 "A Robust Layered Control System For A Mobile Robot", *IEEE Journal of Robotics and Automation*, RA-2, no 1, 14-23.

Brooks R. A. 1991 "Intelligence Without Reason", *12th International Joint Conference on Artificial Intelligence*, Morgan Kaufman, 569 - 595.

Drabble B. 1991 "Spacecraft Command and Control Using Artificial Intelligence Techniques", *Journal of the British Interplanetary Society*, Vol 44, 251-254.

Elfving A. and Kirchoff U. 1991 "Design Methodology for Space Automation and Robotics Systems", *ESA Journal*, Vol 15, 149-164.

Erickson J. D., Phinney D. E., Norsworthy R. S., Zacksenhouse M., Hartness K. T., Pham T. T., Merkel L. F. and Tu E. Y. 1989 "A Prototype Autonomous Agent for Crew and Equipment Retrieval in Space", *Proceedings, Second International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems*, 1052-1058.

Georgeff M. P. 1987 "Planning", *Annual Reviews in Computing Science*, 359-400.

Lindley C. A. 1993a "An Autonomous Satellite Architecture Integrating Deliberative Reasoning and Behavioural Intelligence", *Telematics and Informatics*, 10 (3).

Lindley C. A. 1993b "A Behavioural System for Autonomous Spacecraft Power Distribution and Control", *Artificial Intelligence and Knowledge Based Systems for Space*, 4th Workshop, May 17th - 19th, ESTEC, Noordwijk, The Netherlands.

Lindley C. A. 1993c "A Behavioural Theory of Intelligent Machines as a Framework for the Analysis of Adaptation", *AI-93 Workshop on Evolutionary Computation*, Melbourne, Australia.

Lindley C. A. 1994 "Extending the Behavioural Paradigm for Intelligent Systems", Special Track on Emerging Paradigms for Intelligent Systems, *Twenty-Seventh Annual Hawaii International Conference on System Sciences*, Maui, HI.

Maes P. 1990 "Situated Agents Can Have Goals", in Maes, P. (ed) *Designing Autonomous Agents: Theory and Practice from Biology to Engineering and Back*, MIT/Elsevier, 49-70.

Maes P. 1993 "Behaviour-Based Artificial Intelligence", *Proceedings of the Second International Conference on Simulation of Adaptive Behaviour* , MIT Press.

Pidgeon A., Howard G. and Seaton B. 1992 "Operational Aspects of Spacecraft Autonomy", *Journal of the British Interplanetary Society*, Vol 45, 87-92.

Raslavicius L., Gathmann T. P., and Barry J. M. 1989 "An Artificial Intelligence Framework for Satellite Autonomy", *Proceedings, Second International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems*, Vol 2, 536-543.

Rokey M. and Grenander S. 1990 "Planning for Space Telerobotics: the Remote Mission Specialist", *IEEE Expert*, 8-15, June.

Valavanis K. P. and Saridis G. N. 1992 *Intelligent Robotic Systems: Theory, Design and Applications*, Kluwer Academic Publishers.

Watson J. F., Lefebvre D. R., Desrochers A. A., Murphy S. H., and Fieldhouse K. R. 1992 "Testbed for Cooperative Robotic Manipulators", in Desrochers A. A. (ed.) *Intelligent Robotic Systems for Space Exploration*, Kluwer Academic Publishers.