05ICES-453

Planner-Based Control of Advanced Life Support Systems

Nicola Muscettola
NASA Ames Research Center

David Kortenkamp[1], Chuck Fry[2], Scott Bell[3]
[1] NASA Johnson Space Center/ER2/Metrica Inc., [2] NASA Ames Research Center/QSS Group Inc., [3] NASA Johnson Space Center/ER2/SKT Inc.

ABSTRACT

The paper describes an approach to the integration of qualitative and quantitative modeling techniques for advanced life support (ALS) systems. Developing reliable control strategies that scale up to fully integrated life support systems requires augmenting quantitative models and control algorithms with the abstractions provided by qualitative, symbolic models and their associated high-level control strategies. This will allow for effective management of the combinatorics that emerge when integrating a large number of ALS subsystems. By focusing control actions at different levels of detail and reactivity we can use faster, simpler responses at the lowest level and predictive but complex responses at the higher levels of abstraction. In particular, methods from model-based planning and scheduling can provide optimal resource management over long time periods. We describe a reference implementation of an advanced control system using the IDEA control architecture developed at NASA Ames Research Center. IDEA uses planning/scheduling as the sole reasoning method for predictive and reactive closed loop control. We describe preliminary experiments in planner-based control of ALS carried out on an integrated ALS simulation developed at NASA Johnson Space Center.

INTRODUCTION

Advanced life support (ALS) systems require complex control strategies that can maintain stable system performance and balanced resources with small margins and minimal buffers. In closed-loop life support systems there are complex interactions between sub-systems such as air, water, food production, solids processing, and the crew. Recent research at NASA Johnson Space Center has led to significant insights into autonomous control of ALS systems [1,2,3]. Routine control of an ALS system is well within the reach of current techniques. For example, the autonomous control system described in [4] operated around the clock for 73 straight days during a 90 day crewed test with minimal human intervention. The autonomous control system for a recent test of an advanced water recovery system operated with minimal human intervention for over eighteen months [5]. However these control systems are not able to deal convincingly with the concurrent and interacting control of several subsystems. They also fail to effectively coordinate the effective long term management of resources with the planning of mission activities. Lastly, they can't demonstrate effective recovery from significant anomalies. A solution to these issues is needed in order to demonstrate life support systems amenable to efficient long-duration missions such as the human exploration of Mars.

An effective way to address the complexity of ALS control consists in representing the plant at multiple levels of abstraction and operating each increasingly abstract layer over an increased control horizon. Abstraction inevitably pushes the representation of the system from the realm of continuous feed-back control to the one of discrete or hybrid combinatorial optimization. At the abstract level it is more appropriate to determine optimized courses of action, both by the control plant and by the humans carrying out mission activities, with primary focus on the efficient long term utilization of resources. Long-term plans should then feed control actions and set points to lower-level control algorithms. The actual plant conditions should be communicated back up the control chain and appropriately abstracted if necessary. This allows long-term optimization to be continuously re-adjusted to account for actual operating conditions, in a way similar in spirit to Model-Predictive Control methods [6]. Multiple levels of abstractions also allow a more effective management of reactivity, with simpler but faster responses at the lowest level and more predictive and complex but slower responses at higher levels of abstraction.

The integration of long-term planning with short term action execution and control has been successfully demonstrated in the realm of spacecraft control [7]. It demonstrated “fail-operational” scenarios without human intervention and relying entirely on the on-board control

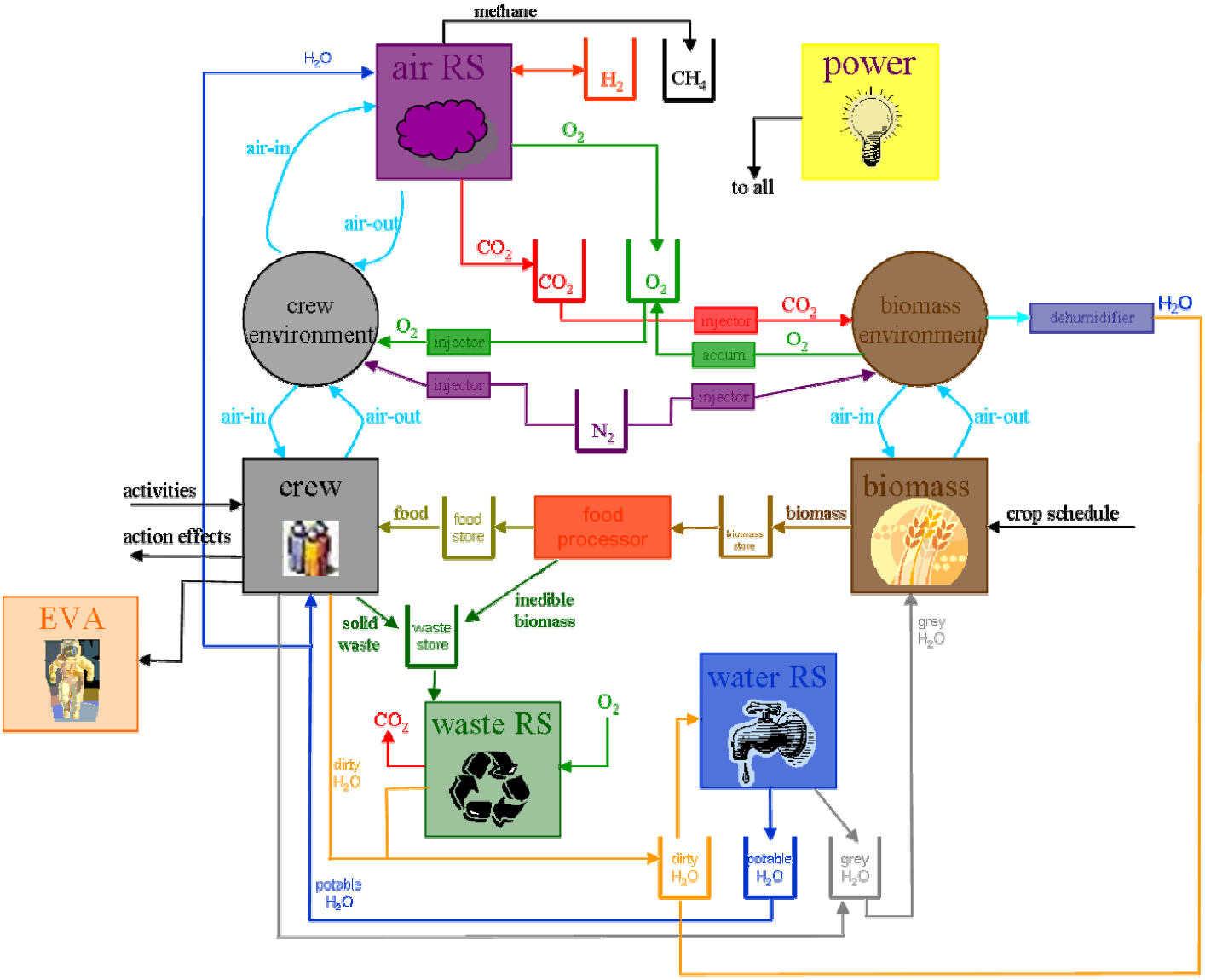

Figure 1: An integrated simulation of a life support system

software. At the highest level these systems build and maintain action plans/schedules that allow the evaluation of resource utilization tradeoffs among a set of competing activities. For example, in the Deep Space 1 Remote Agent experiment the planner had the responsibility of trading off the allocation of the spacecraft attitude resource among competing requests of beacon asteroid imaging and thrusting with the ion engine in an appropriate direction. (confusing)

This paper reports on a novel control system for ALS that integrates long-term planning and plan monitoring with short-term quantitative control. Our main focus is the augmentation of quantitative models with qualitative models on which we apply modern temporal planning algorithms. To ease the evaluation of several possible competing approach to temporal planning, we are encoding the ALS planning models into PDDL+. PDDL+ is a representation language broadly used in the planning and scheduling community. To test our ALS models we use an existing planner-based controller called the Intelligent Distributed Execution Architecture (IDEA) system to control a simple advanced life support scenario. IDEA was derived from the Remote Agent experiment and the Mars Exploration Rover MAPGEN system [8].

In the next section of this paper we describe a life support simulation that we will use to test our planner-based controller. Then we discuss the issues in merging qualitative and quantitative models. Next we introduce our prototype planner-based controller. Finally, we give preliminary results and conclusions.

SIMULATION OF ALS SYSTEMS

Most existing models of advanced life support systems are quantitative and continuous models, e.g., [9]. While these models are useful for analysis it is difficult to translate them into the qualitative, symbolic representations that advanced planning and scheduling systems use. These continuous models don't make

explicit the types of information that are required for integrated, qualitative reasoning. Quantitative, continuous models are necessary for understanding subsystem dynamics, but cross-system analysis offer the ability to reason over multiple subsystems and to project consequences of actions into the future [10].

Our approach to simulation of an ALS system has been developed at NASA Johnson Space Center. We have simulated most of the advanced life support modules using the best available information. The simulation is a process model in that each module takes in certain resources and produces other resources. They are not component models, that is, they do not model physical objects such as valves, pumps, etc. The simulation consists of multiple modules, each representing a subsystem of an advanced life support system. Figure 1 shows the modules and connections in our simulation. The simulator is fully described in [11].

For the experiments in this paper we implemented a specific instance of the simulation to reflect a lunar habitat. The instance was designed with information from an internal JSC memo describing a lunar reference mission [12]. The reference mission assumes a four person crew with equal numbers of men and women. Mission length is 90 days with the habitat initiated and operating nominally upon crew arrival. The landing site is the lunar south pole with the sun above the horizon 80% of the time and surface temperatures between 210K and 230K during the day. The habitat atmosphere is composed of 29% oxygen at an overall pressure of 65.5 kPa and a leakage rate of 0.00224 kg/day. Food is shipped in most circumstances (although we looked at the addition of a small number of crops) and is 0.257 kg/crewmember-day moist food and 0.665 kg/crewmember-day of dry food. Air, water, and waste recovery systems are part of the habitat. One four-hour EVA by one crew member was performed each day of the mission. The EVA takes place through an airlock that is 3.7 m^3 in size and 10% of the airlock atmosphere is lost each time the airlock is used.

Air revitalization is obtained by multiple subsystems and is based on a recently completed test at the NASA Johnson Space Center. Gasses like CO_2 and O_2 produced by the system are either stored (as in the case of CO_2 and O_2), vented (as in the case of methane) or re-injected in other stages of the system. Injectors are available to take gases from the stores and inject them into the atmospheres. A control challenge requires three objectives. The first, and most important, is to maintain an optimal gas mixture in the crew and biomass environments. Secondly, the controller needs to minimize energy use by the accumulator and air revitalization module. Last, the controller should seek to minimize store use.

All stages of the system consume power in the form of electricity. The simulation has two models of power production. One simulates a nuclear-style power system that supplies a continuous, fixed amount of power. A second simulates a solar-style power system that supplies a varying amount of power. For our experiments a solar panel was used.

Testing effective and robust control strategies requires dealing with malfunctions in any component and any module. Each module of the simulation provides an application programmer's interface (API) to introduce these malfunctions at any time in the simulation. Each module can have malfunctions of varying degrees of severity and temporal length. For simplicity, the malfunctions have been divided into two categories based on temporal length: permanent and temporary; and three subcategories of severity: low, medium and high. These malfunctions are interpreted differently by each module. For example, a temporary but severe malfunction in the potable water store would be a large water leak. A permanent but low severity malfunction in the power production module would be the loss of a part of a solar array.

Each module can experience multiple malfunctions at the same time and the control system must detect them, schedule the crew to repair them (if repairable), and monitor to make sure the repairs went accordingly. Permanent malfunctions are non- repairable and require the control system to reallocate resources to continue the mission. A permanent malfunction with the water recovery system, for example, might cause a decrease in potable water. The control system could react by lowering available water to the plants to provide enough water to the crew.

The simulation also models stochastic processes. Because the real world is not deterministic, neither is the simulation. For example, the exact amount of air that is breathed in by a crew member is different with every breath. We model this by using a Gaussian function with adjustable parameters. The Gaussian can be set to zero to produce a deterministic simulation.

The simulation is controlled via sensors and actuators which model physical sensors and actuators of an advanced life support system. Sensors report on values of the underlying simulation. For example, an O_2 sensor would report the amount of O_2 in the atmosphere. Sensors in the real-world are noisy – that is they do not always return ground truth. We model sensors with an adjustable Gaussian noise function. Sensor noise can be turned off so that the sensors report ground truth.

Actuators are mirror images of sensors – they allow for control actions to be taken on the simulation. Like sensors, actuators in the real-world are noisy. For example, an injector that is told to open for one second will open for slightly more or less than one second given its mechanical tolerances. We model this noise as a Gaussian function. The parameters of the noise function are adjustable and the function can be turned off.

INTEGRATING QUALITATIVE AND QUANTITATIVE MODELS

The simulation described above is based on continuous differential equations. While this is excellent for continuous simulations these kinds of models do not translate easily into the declarative and procedural models required by intelligent control systems. They need to be augmented with models that capture the interactions amongst subsystems, the causes and effects of malfunctions and the duration and times of control actions. This poses many challenges including:

- **Abstraction**: How detailed do the models need to be for effective reasoning? A continuous curve, like the evolution of ambient temperature over time, could be represented as a discretized, piecewise constant/linear function. In how many pieces should the curve be split?

- **Compactness**: We want the models to be as compact as possible. Some qualitative and quantitative modeling approaches lead to a proliferation of state variables and task types. Also the models must describe the states in which a subsystem does *not* want some other subsystem to be in over time. For example, suppose that a constraint wanted to declare that while it is in state A, another subsystem cannot be in state B. This does not prescribe a single value that has to hold for an extended period of time, e.g., containing A. It can actually be satisfied by a series of contiguous transitions, e.g., while in state A the other subsystem can transition periodically between C and D an arbitrary number of times. Representing these constraints is a key challenge. (example confusing)

- **Maintenance**: As we model more and more complicated systems maintaining accurate models becomes important. First there is the verification and validation of the models themselves – are they correct? Second, how can we easily change models when the underlying system changes?

The field of Artificial Intelligence planning relies on representations of actions and dynamic processes. A planner [13] uses a compositional model of the world expressed as a collection of operators. An operator represents the pre-conditions, post-conditions and maintenance conditions around a change of state in the world. Pre-conditions must hold before the change. Post-condition will hold after, and maintenance conditions must occur throughout the change. Two kinds of change can be modeled. The first, *actions*, are typically used to represent state transitions explicitly initiated by the control system. The second, *events,* typically represent a spontaneous change in the world without the explicit interventions of a controller. An operator-based representation gives the means to address the challenges described before. Since one of our goals is to establish a strong connection between the planning community and the life-support community, it is important that our representation of the life support domain be expressed in a language that is as accessible and standard as possible. This led us to select the PDDL+ domain modeling language [14], an extension to the original Planning Domain Definition Language (PDDL) [15] that has become the standard mean for expressing benchmark problems in the bi-annual planning competition [16]. PDDL+ extends PDDL by allowing expression of continuous processes that act on states in the system.

Figure 2 shows the representation of a fragment of the ALS system in PDDL+. The action `increase-flow`, for example, allows the controller to increase the flow through the pipe `?pipe` in order to raise the level of the material contained in the tank above the allowable lower limit. The lower limit is represented as the difference of the set-point level and a given deadband, `(- ?target ?deadband)`. The action can be executed any time the level of material in the tank, `(level ?tank)`, reaches the lower limit. The effect of the action is to increase flow through the pipe. We can then give the planning system initial parameters (e.g., store targets, deadbands, etc.) and a goal, e.g., `(:goal (level-maintained potable_water_store)` and the planning system will achieve and maintain that goal using the actions and processes. Between control actions, the tank evolves

```
(:action increase-flow
    :parameters
        (?tank ?pipe ?target ?deadband)
    :precondition
        (< (level ?tank) (- ?target ?deadband))
    :effect
        (increase (commanded-flow ?pipe)
                  (calculate-amount ?pipe (level ?tank)
    ?target))
)

(:action decrease-flow
    :parameters
        (?tank ?pipe ?target ?deadband)
    :precondition
        (> (level ?tank) (+ ?target ?deadband))
    :effect
        (decrease (commanded-flow ?pipe)
          (calculate-amount ?pipe (level ?tank) ?target))
)

(:process maintain-flow
    :parameters
        (?tank ?pipe ?target ?deadband)
    :precondition
        (and
                (> (level ?tank) (- ?target ?deadband))
                (< (level ?tank) (+ ?target ?deadband)))
    :effect
        (= (level-maintained ?tank) TRUE)
)

(:event store-level-not-within-target
    :parameters
        (?tank ?target ?deadband)
    :preconditions
        (or
                (> (level ?tank) (+ ?target ?deadband))
                (< (level ?tank) (- ?target ?deadband)))
    :effect
        (= (level-maintained ?tank) FALSE)
```

Figure 2: A sample of PDDL

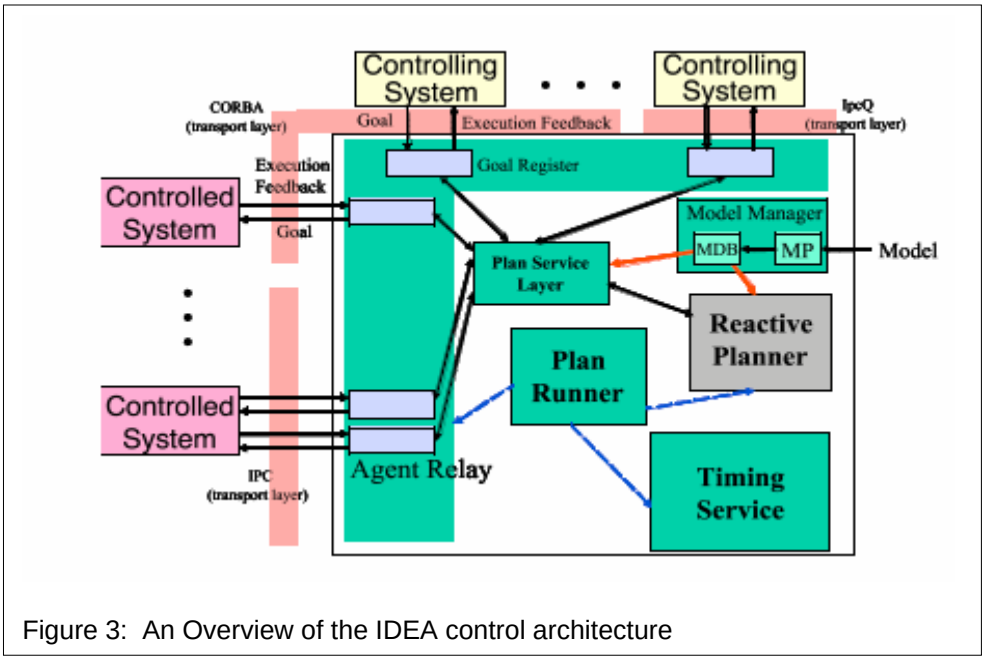

Figure 3: An Overview of the IDEA control architecture

according to a process, `maintain-flow`, which at this level of abstraction is simply represented as a period of time during which the tank level remain within a deadband around the target level. The process is terminated by an event, `store-level-not-within-target` which is triggered by the tank level falling outside the setpoint range. Depending on which "out of bounds" condition causes the event, the appropriate control action between `increase-flow` and `decrease-flow` should then be applied to restore the control goal.

Note that this is not a replacement for low-level, model-based control (e.g., [6]) that would turn on and off pumps, valves, etc. to actually fill and drain stores. Instead, this is a coarse qualitative model that describes overall system states and goals. This simple model only represents a nominal situation. In an off nominal situation it is possible that the controller will not be able to exert a control action immediately when the tank level is detected to be out of bounds, i.e., event `store-level-not-within-target` occurs. A full model of the device must represent also all of these off-nominal processes and events as well as the possible corrective actions to restore nominal operations.

PLANNER-BASED SUPERVISORY CONTROL

A representation of the system and control actions in terms of planning operators is not sufficient to build viable controllers. We also need a framework for supervisory control that can interpret these models, build plans, monitor their execution and modify the plan within the real-time constraints imposed by the physics of the plant.

The control framework that we are adopting is the IDEA system [17]. IDEA evolved from the experience of the Remote Agent. Different approaches, like Remote Agent and other three-layered control system, use reasoning mechanisms and control machinery at different levels. By contrast, each IDEA agent strictly adheres to a single formal virtual machine and uses a model-based reactive planner as its core engine for reasoning. The IDEA architecture is *service-based* in the sense that it provides unifying services for fundamental functions needed for a planner-based controller. It does not, however, impose the selection of a specific planning approach, planning algorithm, or reasoning method to select the control actions. IDEA defines a virtual machine that organizes these services and a set of expectations with regard to the functionalities needed by a planner. For example, the ability for a planning algorithm to operate on a plan-database concurrently with other programs, and specific rules for the kind of interactions that are possible when multiple planners modify the same section of the database. Any planner that is capable of satisfying the requirements of IDEA can be used as the core of an IDEA control agent. To this date, IDEA agents have been implemented using EUROPA [18]. These planning technology and planning algorithms derive from the on-board planner of the Remote Agent and have been successfully used throughout the Mars Exploration Rover (MER) mission to implement MAPGEN. MAPGEN is the science activity planning system successfully used by the ground operators that have been operating the Spirit and Opportunity rovers on Mars.

THE IDEA VIRTUAL MACHINE

Fig. 3 gives an overview of the components of an IDEA agent. The agent communicates with other agents (either controlling or controlled by the agent) using an *Agent Relay*. The agent relay maintains the IDEA agent's execution context by sending or receiving message invocations (respectively, goals sent to controlled agents or received from controlling agents) and receiving or sending method return values (i.e. the achievement of a goal). The execution context is

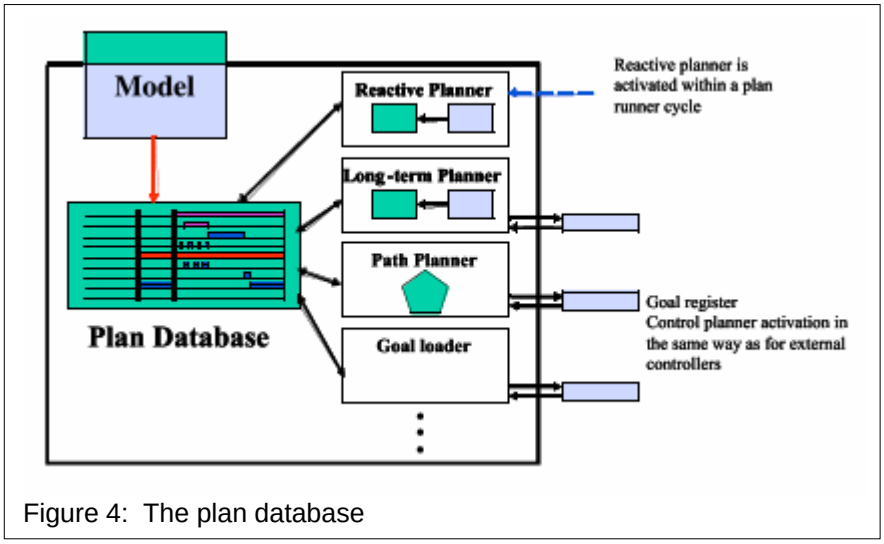

Figure 4: The plan database

synchronized with the internal state of a *Reactive Planner* (RP). The RP is the control engine of the IDEA agent: given a declarative (temporal) model of the agent activities (i.e. the planning model maintained by the *Model Manager*) and the execution context. It is responsible for generating the control procedure invocations. Although IDEA's modeling language is different from PDDL+, it uses very similar concepts as constructs, making the translation between the two straightforward.

IDEA EXECUTION CYCLE

The *Plan Runner* (PR) executes a simple, finite state machine that implements the sense/plan/act cycle of the IDEA agent. Each cycle must be completed within a finite *execution latency.* At present, an agent's latency corresponds to the minimum quantum of time that can be measured by an agent, the *agent tick.* Time is measured by a *Timing Service* that is also capable of warping time, a capability extremely useful in simulation to significantly compress the time needed to run multi-day scenarios. The PR operates as follows:

- The PR wakes up at the first tick after a message has been received from another agent, or at the tick when a wakeup timer has gone off;
- The state of the Agent Relay is updated with respect to the information resulting from the wakeup event (e.g., an event representing the return value of a control action has been received);
- The RP is invoked and the planner synchronizes its internal state with the Agent Relay through the *Plan Service Layer*.
- When the RP terminates, the agent relay loads the new context of execution and sends appropriate messages to the external agents. For example, if a control action has been terminated by the reactive planner, an event corresponding to the procedure's return value (determined by the RP) is sent to the agent's goal-setting interface (the controlling agent)
- The RP is invoked to determine what is the next time at which execution is expected to occur (barred any external communication). The time is set in the *Timing Services* module as the next wakeup time for the agent;
- The plan runner goes to sleep and waits for an external message or the expiration of a wakeup timer.

PLAN DATABASE

The reactive planner continuously updates a data structure, called *Plan Database* (PD) (see Fig. 4), which represents the I/O and internal state of the agent. The PD describes the past and the future execution state of the agent as a set of *timelines* (one for each state variable). A timeline represents the history of a state variable over a period of time. Each history is a sequence of tokens built by the RP keeping the consistency with respect to the IDEA model. The reactive planning is to refine the plan database checking for the consistency of the PD with respect to the current execution state and providing an execution plan up to a *planning horizon*. For a given timeline the past history represents ended activities and states while the future history is complete plan of activities with maximum flexibility, i.e., the start and end times are defined only if necessary.

Inconsistencies between expectations and actual events occurring in execution (e.g., a mismatch between actual and expected time of occurrence for a `store-level-not-within-target`) must be reconciled by RP before execution can continue (but still within the agent latency constraint). Only when the plan fully conforms to the model for a specified horizon following the current tick can the execution cycle be completed.

REACTIVE AND DELIBERATIVE PLANNING

In IDEA reactive planning determines the next action on the basis of sensory input and time lapse wakeups. More complex problem solving (e.g., long-term task planning) typically requires more time than the latency allows. IDEA provides a rich environment for integrating any number of deliberative planners within the core execution cycle (Fig. 4). Different specialized planners can cooperate in building a single plan coherently with the agent's model. Also in IDEA the activation for a deliberative planner is programmed in the model. This can be obtained by modeling the planner like any other subsystem, i.e., by specifying a timeline that can take tokens whose execution explicitly invokes the planner. This makes it possible to appropriately plan the time at which deliberate planning can occur compatibly with the internal and external state modeled by the agent.

PRELIMINARY RESULTS

We have just begun implementing an IDEA controller for the BioSim application. In this section we discuss our experimental scenario and we discuss the very first IDEA, planner-based controllers that we have implemented.

DEMONSTRATION SCENARIO

Our demonstration scenario is a 90-day expedition on the lunar surface. It assumes a four-person crew, performing one extra-vehicular activity per day. A separate biomass (plant growth) chamber contains two crops, wheat and white potatoes, and maintains a separate atmosphere. Both crew and biomass environments share a common drinking water supply; both environments also have small air leaks.

The air recycling system includes an oxygen generator, which electrolyzes drinking water to oxygen and hydrogen gases, and a carbon dioxide scrubber. The biomass chamber also serves as part of the air recycling system, converting carbon dioxide into oxygen and plant matter via photosynthesis. Storage tanks hold oxygen, hydrogen, nitrogen, and carbon dioxide gases. A modest amount of oxygen, nitrogen, and carbon dioxide are provided at the start of the scenario, to account for losses due to leakage.

The water recycling system takes grey water and dirty water from the crew chamber and other sources, and converts them to potable water and solid waste by a variety of processes. Dehumidifiers in the crew and biomass chambers extract excess moisture from the air. A food processor converts biomass into food for the crew, dry waste, and grey and dirty water. Electrical power is generated by a solar cell array, and stored in a large battery.

The scenario allows for some uncertainty and calamity. The simulator can induce stochastic errors in the values reported by sensors, and the commanded positions of actuators. The simulator can also introduce failures in components and crops, and unexpected losses of air and water. The simulated failures we plan to demonstrate will range from the trivial to the catastrophic.

The IDEA agent is charged with managing several variables in this scenario:

- Potable H_2O tank level, via control of power to the water recycling system, and control of water flow rate to crops
- O_2 tank level, via control of the flow rate from the biomass chamber accumulator, and control of power to the oxygen generating system
- CO_2 level in biomass chamber, via control of CO_2 injector flow rate
- Wheat planting time and amount planted
- Wheat crop harvest time
- White potato crop planting time and amount planted
- White potato crop harvest
- Biomass chamber light level

The demonstration is a success if the IDEA agent can keep the crew and plants alive with smaller initial stocks of consumables (initial O_2, H_20 and crop seeds) and lower system mass (smaller potable H_20 tank, smaller O_2 tank, smaller power supply, smaller cabin volume, etc.) when compared to a default control scheme on the same variables and the same failures.

To do this, the IDEA agent must successfully integrate two kinds of control: fairly continuous, real-time control of variables such as the O_2 accumulator and CO_2 injector; and fairly discrete, long duration control of variables such as crop planting and harvesting. It must also successfully represent and reason with qualitative information (planting times, etc.) and quantitative information (diff. equations underlying plant growth, 0_2 and H_2O production and CO_2 consumption).

PRELIMINARY IDEA CONTROLLER

As a first step, we have implemented a simple reactive controller which maintains the levels in the potable water tank and the oxygen tank. The reactive controller is intended to be used at the lower level of a hierarchy of controllers. The desired minimum and maximum levels for both tanks can be specified manually, or by a master controller. The reactive control algorithm attempts to keep the actual levels within those limits, using only

knowledge of the current state of the system and its immediate past.

The potable water tank's level is controlled by managing electrical power to the water recycling system, and by directly controlling the water supply valve to the biomass chamber. If the level is too low, the water recycling system is powered to its maximum capacity, or the water supply valve to the biomass chamber is closed, or both. Likewise, if the level is too high, the water recycling system is switched off, or the output to the biomass chamber is opened to its maximum capacity, or both.

The level in the oxygen tank is controlled similarly by controlling two sources of oxygen. One of these is the oxygen generating system, which is controlled by managing its electrical power. The other source is the oxygen accumulator, which extracts oxygen produced by photosynthesis in the biomass chamber; it is controlled via a valve in the return line to the oxygen tank.

The controller currently uses a simple “on or off” algorithm, with local memory of the current state of the controls. For example, if the water tank has been below the target minimum level for 3 simulation ticks, and the water recycling system is already at full power and the water supply valve to the biomass chamber is already closed, it knows that it can do nothing further, and just waits for the water level to rise.

Commanding is closed-loop. The IDEA agent expects confirmation from the simulation that each command has been completed. This is to allow for recoveries in cases of transient or persistent faults. The agent does not currently support retrying commands in the event of a fault; it simply transitions to a fault state. Nor does the agent monitor the flow rate sensors to judge the effect of a command. These are extensions we plan to add in the near future.

At present the two tanks are controlled independently from each other. There is one direct interaction between the two tanks, and several potential indirect interactions. Control of the oxygen generating system has a direct effect on the level of the potable water tank. As more electrical power is fed to the Oxygen Generation System (OGS) in an attempt to produce more oxygen, its demand on the potable water the supply increases. Indirect interactions include contention for limited electrical power, water flow rate to the biomass chamber affecting the rate of photosynthesis, and so forth. Such interactions will be modeled in the future.

CONCLUSION

An effective life support control system can reduce system mass, reliance on ground controllers, and crew time spent monitoring life support functions. For these reasons, life support control systems are an enabling technology for long-duration space missions. This paper describes the need for qualitative modeling and reasoning in order to more effectively control the interactions and resource constraints of advanced life support systems. We hope to engage the planning community by representing the life support domain in a language that they understand. A first step is to encode life support models in PDDL+. A second step is to test those models in a planner-based control architecture. We are just beginning this process.

ACKNOWLEDGMENTS

This work is funded through a grant from NASA’s Office of Biological and Physical Research, Advanced Environmental Monitoring and Control program and from the Intelligent Systems Project of NASA's Exploration Mission Directorate

REFERENCES

1. Jorge Leon, David Kortenkamp and Debra Schreckenghost, “A Planning, Scheduling and Control Architecture for Advanced Life Support Systems,” *Proceedings of the NASA Workshop on Planning and Scheduling in Space*, 1997.
2. David Kortenkamp, R. Peter Bonasso and Devika Subramanian, “Distributed, Autonomous Control of Space Habitats,” *IEEE Aerospace Conference*, 2001.
3. Schreckenghost, Debra, Carroll Thronesbery, R. Peter Bonasso, David Kortenkamp and Cheryl Martin, “Intelligent Control of Life Support for Space Missions,” in IEEE *Intelligent Systems Magazine*, Vol. 17, No. 5, September/October 2002.
4. Debra Schreckenghost, Mary Beth Edeen, R. Peter Bonasso, and Jon Erickson, “Intelligent Control of the Product Gas Transfer for Air Revitalization,” *Proceedings of the 28th Conference on Environmental Systems*, 1998.
5. R. P., David Kortenkamp and Carroll Thronesbery, Intelligent Control of a Water Recovery System. In *AI Magazine*, Vol. 24, No. 1, Spring 2003.
6. Abdelwahed, S., J. Wu, G. Biswas, J. Ramirez and E. J. Manders, “Online Fault Adaptive Control for Efficient Resource Management in Advanced Life Support Systems,” *Habitation: International Journal for Human Support Research*, Vol. 10, No. 2, pp. 105-116, 2005.
7. Muscettola, N. P. Pandurang Nayak, B. Pell and B. Williams, “Remote Agent: To Boldly Go Where No AI System Has Gone Before,” Artificial Intelligence, Vol. 103, No. 1, pp. 5-47, 1998.
8. John Bresina, Ari Jonsson, Paul Morris, and Kanna Rajan, "Activity Planning for Mars Exploration Rovers", in Proceedings of 15th International Conference on Automated Planning and Scheduling (ICAPS), 2005.
9. Finn, Cory K. “Dynamic System Modeling of Regenerative Life Support Systems,” *29th International Conference on Environmental Systems*, SAE paper 1999-01-2040.

10. Benjamin Kuipers, *Qualitative Reasoning: Modeling and Simulation with Incomplete Knowledge*, MIT Press, Cambridge MA, 1994.
11. David Kortenkamp and Scott Bell, “Simulating Advanced Life Support Systems for Integrated Controls Research,” to appear in *33rd International Conference on Environmental Systems*, SAE paper 2003-01-2546, 2003.
12. Hanford, T., "Transient Modeling Challenge: A Lunar Reference Mission for a 90-Day Habitat," *NASA JSC Draft Document*, 2004.
13. David E. Smith, Jeremy Frank and Ari K. Jonsson, “Bridging the Gap Between Planning and Scheduling,” *Knowledge Engineering Review*, 15(1), 2000.
14. M. Fox and D. Long “PDDL+: Modeling continuous time dependent effects,” in Proceedings of the 3rd International NASA Workshop on Planning and Scheduling for Space.
15. M. Ghallab, A. Howe, C. Knoblock, D. McDermott, A. Ram, D. Weld and D. Wilkins, “PDDL – The Planning Domain Definition Language,” Technical Report, Yale Center for Computational Vision and Control, available as part of the PDDL distribution at http://ftp.cs.yale.edu/pub/mcdermott/software/pddl.tar.gz, 1998.
16. Drew McDermott, “The 1998 AI Planning Systems Competition,” *AI Magazine*, 21(2), 2000.
17. Muscettola, N.; Dorais, G. A.; Fry, C.; Levinson, R.; and Plaunt, C. IDEA: Planning at the core of autonomous reactive agents. In *Proc. 3rd Int. NASA WS on Planning and Scheduling for Spac*e, 2002.
18. Jeremy Frank, and Ari K. Jonsson, "Constraint-based Attribute and Interval Planning", in Constraints, 8(4), p 339-364, 2003.

CONTACT

Nicola Muscettola, Intelligent Systems Division, NASA Ames Research Center, MS-269-2, Moffett Field California 94035, mus@email.arc.nasa.gov