Model-based Programming of Reactive Systems: The Journey of Deep Space One

Brian C. Williams
MIT and NASA Ames Research Center

A new generation of sensor rich, massively distributed systems are emerging that offer the potential for profound economic and environmental impact, ranging from building energy systems to autonomous space probes. These "immobile" robots have the richness that comes from interacting with physical environments, together with the complexity of networked software systems. The goal is to make these systems efficient, capable and long lived, that is, able to survive decades of autonomous operation within unforgiving environments. This offers an overwhelming programming challenge. Hand coding functions for maintaining the system's internals traditionally requires the programmer to reason through system wide interactions, along lengthy paths between the sensors, control processor and control actuators. This reasoning requires thinking about the behavior of a hybrid system, composed of complex real-time software constructs, distributed digital hardware and continuous physical processes. The resulting code typically lacks modularity, is fraught with error and makes severe simplifying assumptions. Severe limitations of these codes are compensated by large operations teams.

Model-directed autonomy meets this challenge through two ideas. First, we note that programmers generate the desired function from their commonsense knowledge of how the software and hardware modules behave. The idea of model-based programming is to exploit this modularity by having engineers program model-directed systems by simply articulating and plugging together these commonsense models. The next challenge is the infeasibility of synthesizing a set of codes at compile time that envision all likely failure situations and responses. Our solution is to develop real time systems, called model-directed executives, that respond to novel situations on the order of hundreds of milliseconds, while performing extensive deduction, diagnosis and planning within the reactive control loop.

For the last several years my work, and that of the autonomous systems group at NASA Ames, has been directed towards developing an experimental autonomous control system, called Remote Agent (RA). Remote agent will soon be demonstrated as a technology experiment of the New Millennium spacecraft, Deep Space One (DS1). DS1, launched recently during the fall of 98, will fly by an asteroid and a comet, using its solar powered ion propulsion engine and navigating optically by following the stars. To demonstrate fully autonomy, Remote Agent will guide DS1 through a series of optical navigation maneuvers during May of 99, responding and replanning in response to a series of injected failures and changing mission goals.

In this talk I will examine the lessons learned from DS1 about model-based programming and model-directed execution. In the first part I will examine the representational needs of modeling DS1 as a hybrid system. From this I will develop the Reactive Model-based Programming Language (RMPL). RMPL uses transition systems and co-temporal interactions as a rallying point for unifying representational concepts from a diverse set of research areas, including qualitative modeling, model-based diagnosis, hidden Markov decision processes, synchronous reactive languages and concurrent constraint programming.

In the second part I will discuss the development of a series of increasingly more expressive model-directed executives. Each executive is formulated as a deductive form of an optimal, model-based controller in which models are specified through a combination of hierarchical, probabilistic transition systems and propositional logic. This framework allows us to achieve high levels of autonomy and responsiveness, by drawing inspiration from a diverse set of algorithms taken from model-based diagnosis, hidden Markov processes, search, planning and real-time propositional inference. I will conclude by discussing the role model-directed autonomy is playing within a variety of future NASA applications, from Mars Exploration to the search for Earth-like planets around other stars.

Model-based Programming of Reactive Systems: The Journey of Deep Space One

Brian C. Williams
Autonomous Systems, NASA Ames
Massachusetts Institute of Technology

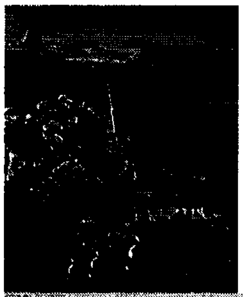
http://www.ic.arc.nasa.gov/ic/projects/mba/index.html

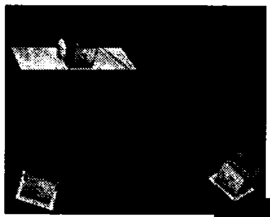
In collaboration with
P. Pandurang Nayak, Vineet Gupta, Jim Kurien

Outline

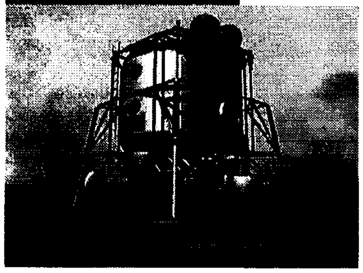
- Remote Agents and Deep Space One
- Model-based Autonomy
- Livingstone:Model-based Configuration Manager (with Pandu Nayak).
- Unifying Model-based and Reactive Programming (with Vineet Gupta).
- Conclusions

Emergence of Long-lived Systems

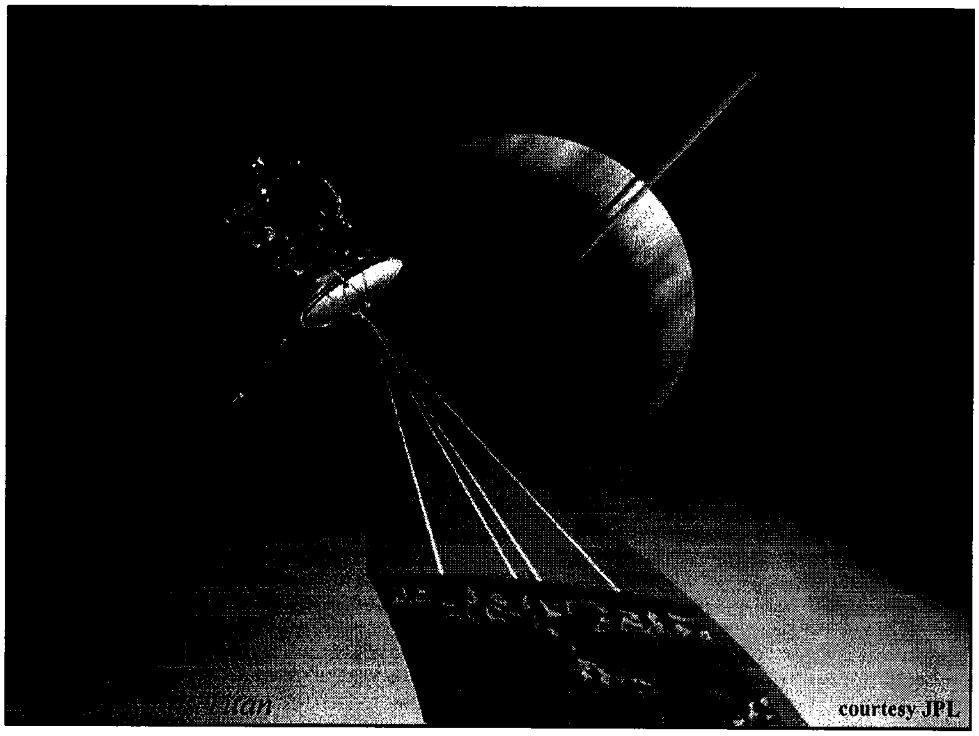

Rovers


Interferometers

In situ Propellant Production


Life Support

Explorers



Houston, We have a problem ...

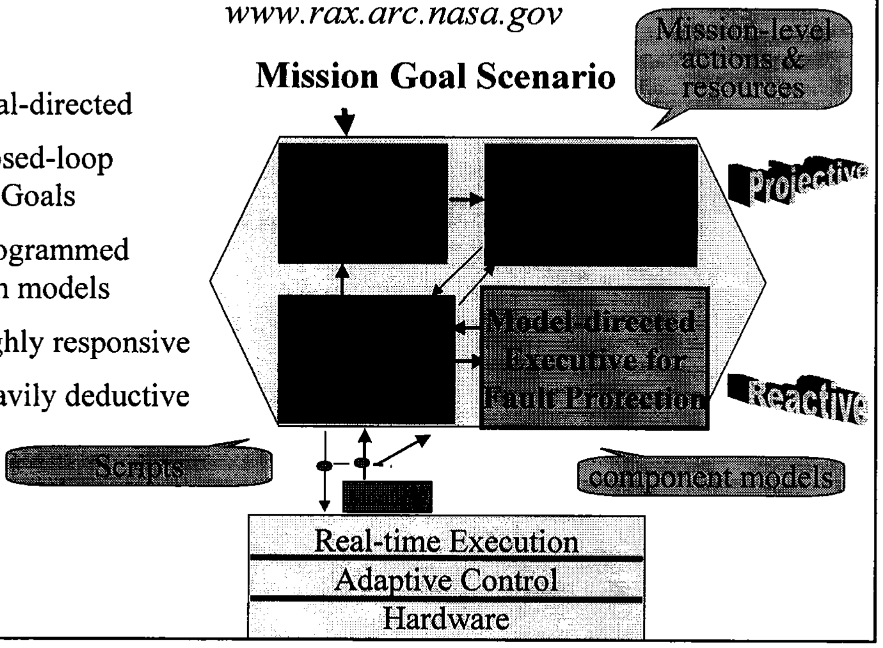
- Quintuple fault occurs (three shorts, tank-line and pressure jacket burst, panel flies off).
- Ground assembles novel repair.
- Swagert & Lovell work on Apollo 13 emergency rig lithium hydroxide unit.
- Mattingly works in ground simulator to identify novel sequence handling severe power limitations.

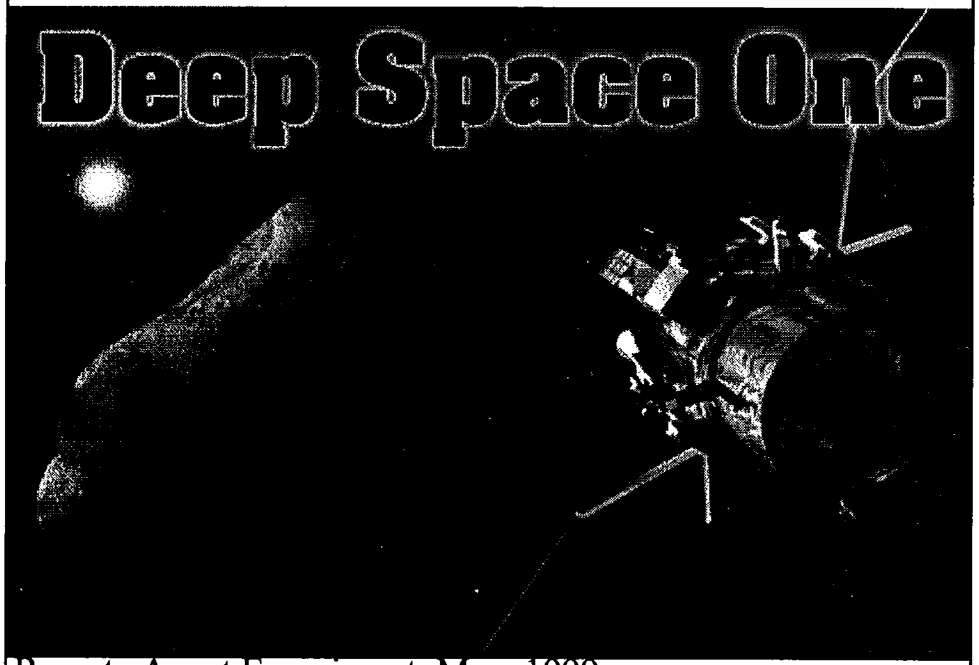
Autonomy should embody the innovation exemplified in Apollo 13 and other missions.

Remote Agent **AMES / JPL**

www.rax.arc.nasa.gov

- Goal-directed
- Closed-loop on Goals
- Programmed with models
- Highly responsive
- Heavily deductive

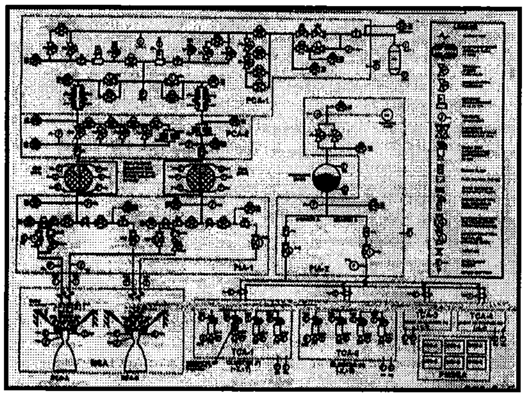




Remote Agent Experiment: May, 1999 **courtesy JPL**

Outline

- Remote Agents and Deep Space One
- Model-based Autonomy
- Livingstone:Model-based Configuration Manager
  (with Pandu Nayak).
- Unifying Model-based and Reactive Programming
  (with Vineet Gupta).
- Conclusions



Challenge: Programmers reason through interactions:

- monitoring
- confirming commands
- tracking goals
- detecting anomalies
- diagnosing faults
- reconfiguring hardware
- recovering from faults
- avoiding failures
- coordinating control policies

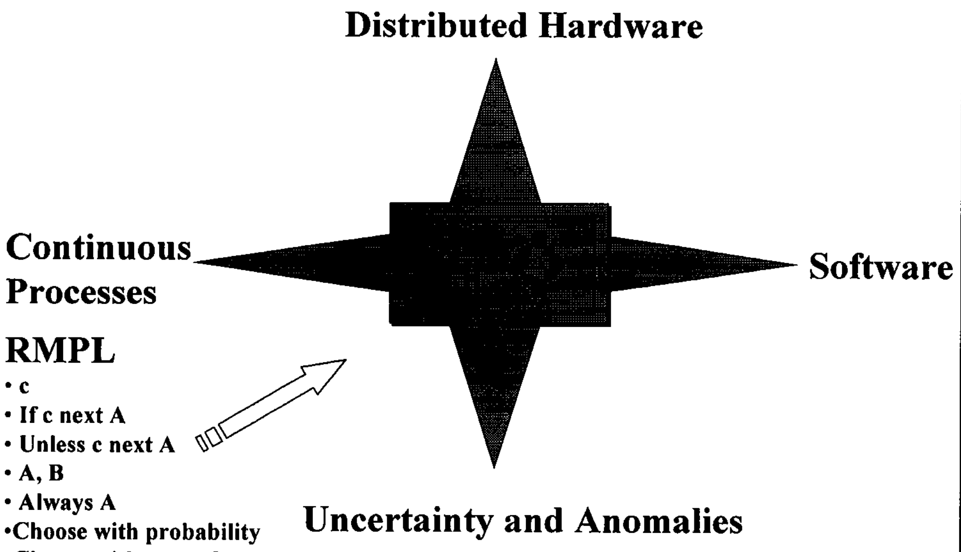
Model-based Autonomy

Programmers generate breadth of functions from commonsense hardware and software models in light of mission-level goals.

- Model-based Programming
  - Program by specifying commonsense, compositional models.
- Model-directed Executives
  - Performs reasoning through system interactions, by performing significant search & deduction within the reactive loop.

How do we model and reason about hybrid software/hardware systems simply and uniformly?

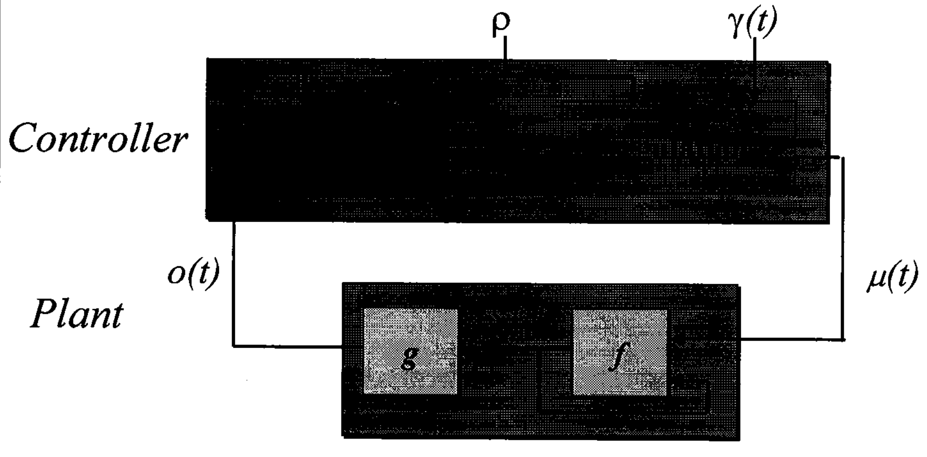
A Unified Modeling Paradigm



RMPL

- **c**
- **If c next A**
- **Unless c next A**
- **A, B**
- **Always A**
- **Choose with probability**
- **Choose with reward**

Model-directed Execution is Formulated as Model-based Stochastic Optimal Control



$$\mu = \mathrm{argmin}\; C(s', \gamma, \mu') \text{ s.t. } \mu' \in M$$

Reformulating the Autonomy Coding Challenge

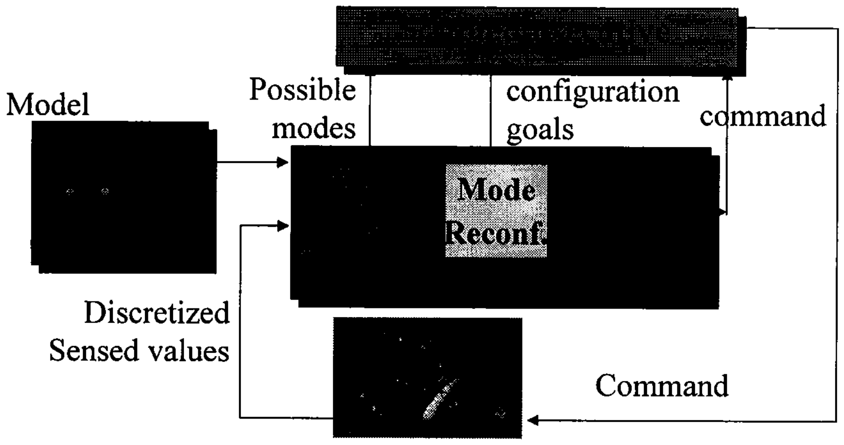
Programmers must reason through system-wide interactions to generate codes for:

- monitoring
- confirming commands
- tracking goals
- detecting anomalies
- isolating faults
- diagnosing causes

- reconfiguring hardware
- recovering from faults
- safing the system
- avoiding failures
- coordinating control policies

Identifying Modes — Reconfiguring Modes

Livingstone and Burton Model-directed Executives



Model-based, Stochastic, Optimal Controller where:
- variables have finite domains
- model specified in a propositional temporal logic
- implemented as queries to a fast, best-first, inference core

Outline

" Remote Agents and Deep Space One

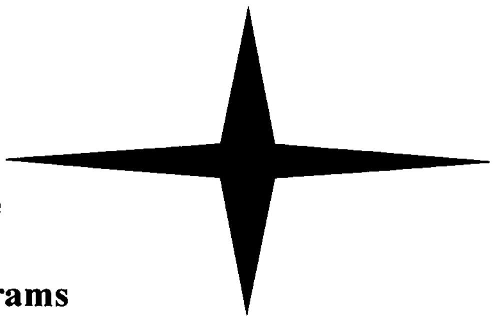
" Model-based Autonomy

- Livingstone:Model-based Configuration Manager (with Pandu Nayak).
- Unifying Model-based and Reactive Programming (with Vineet Gupta).
- Conclusions

A Unified Modeling Paradigm

Distributed Hardware

Continuous Processes



Software

Qualitative Algebra & State Diagrams

Uncertainty and Anomalies

Modeling: Co-temporal Interactions

Valve Driver				Valve			
On	■	□	Resettable failure	Open	⧓	⋈	Stuck open
Off	□	■	Permanent failure	Closed	⋈	⧓	Stuck closed

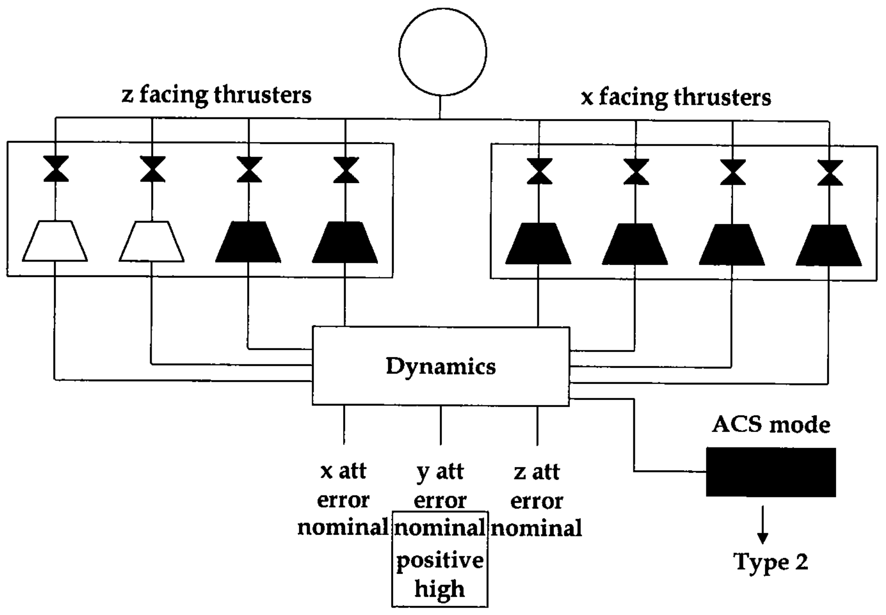
vdriver=on => Out= In;
vdriver=off=> Out=close; ....

vlv=open => Sgn(Inflow)= Sgn(Outflow);
vlv=closed=> Sgn(Outflow)=0; ....

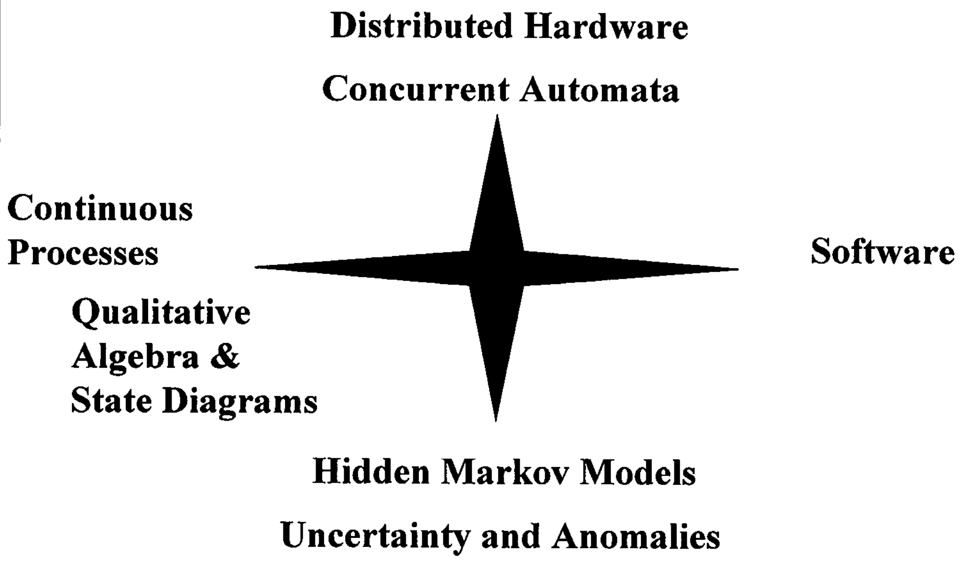
Formulae in propositional state logic $\{x_i = v_{ij}\}$

- Algebra on Signs : S = {+, 0, -,?}
  - $[\text{inflow}]_S = [\text{outflow}]_S$
- Algebra on Relative Values: R = {high, nominal, low,?}
  - $[\text{inflow}]_R = [\text{outflow}]_R$

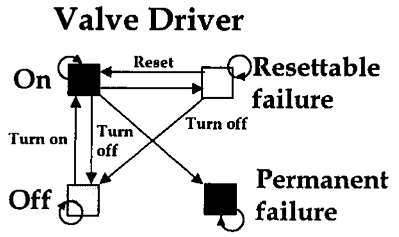
DS1 RCS failure scenario

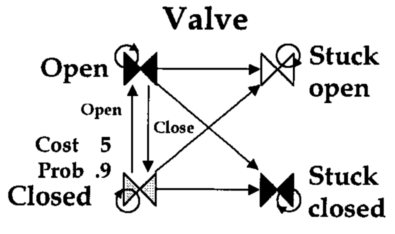


A Unified Modeling Paradigm



Modeling: Constraint-based, Concurrent, Probabilistic, Automata



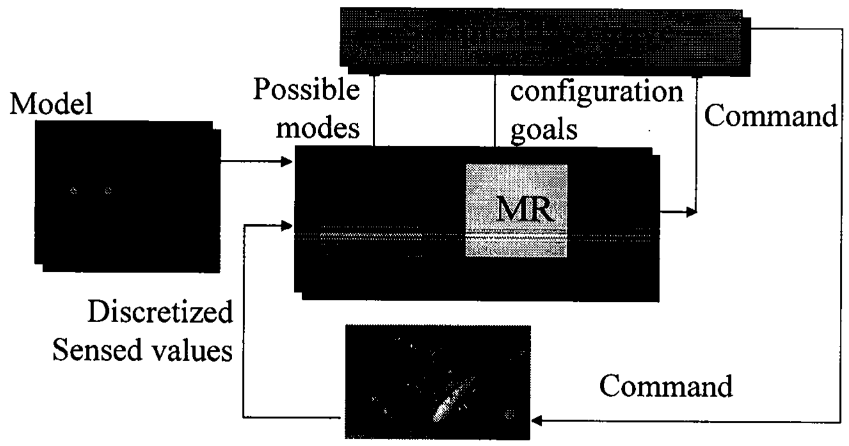


vdriver=on => Out= In;
vdriver=off=> Out=close; ....

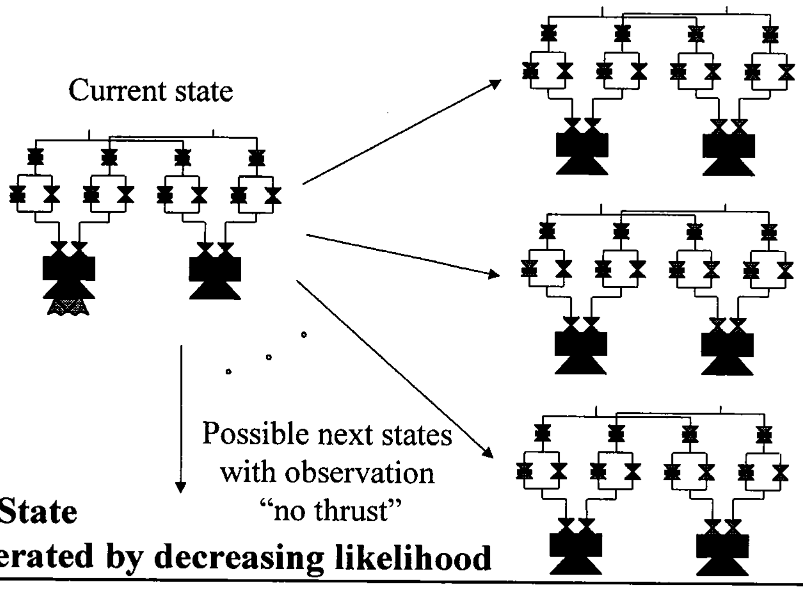
vlv=open => Sgn(Inflow)= Sgn(Outflow);
vlv=closed=> Sgn(Outflow)=0; ....

Compact encoding of a restricted POMDP

Livingstone: Model-based Configuration Manager



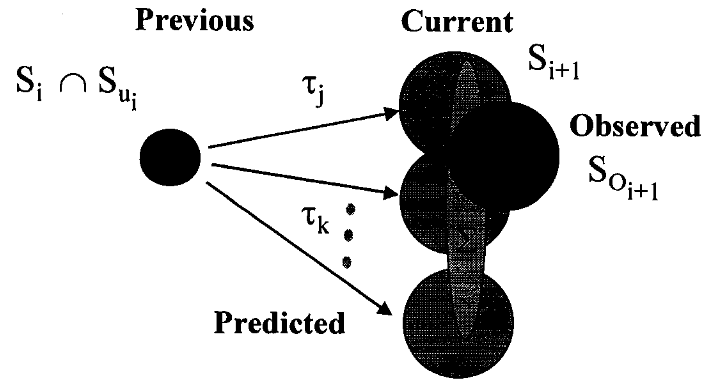
Mode Identification: Reachable Consistent Next States



Belief State
Enumerated by decreasing likelihood

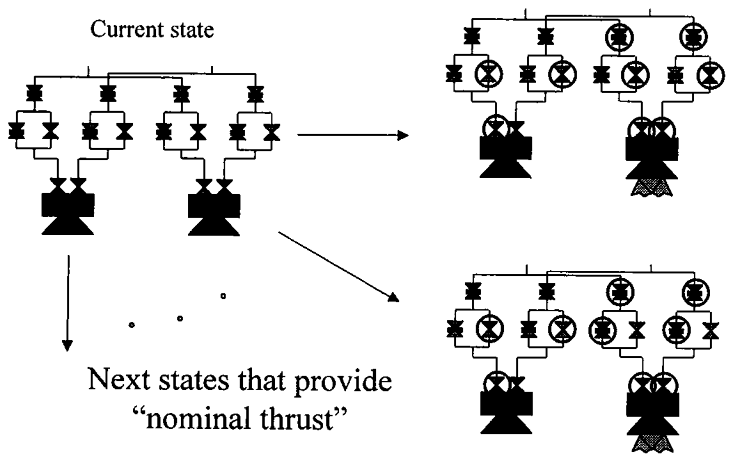
Characterizing MI

- Find feasible current states, given prior state, commands and current state observations.



- Probabilities computed as standard belief state update with conditional independence of transitions.

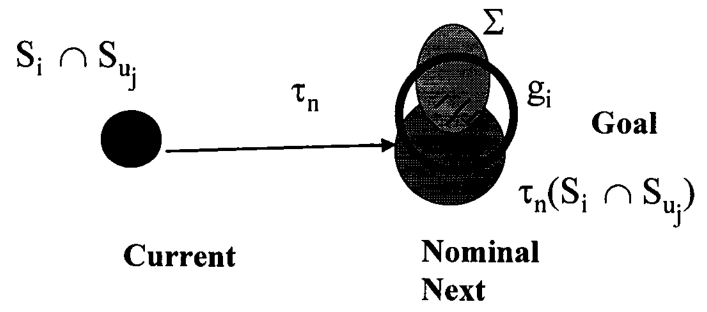
Mode Reconfiguration: Reaching Goals in the Next State



Enumerated by decreasing immediate reward

Characterizing MR

- Find least cost commands that achieve the current goal in the next state.



Propositional Reduction

- MI: Characterizing possible next states

$$\rho_{S_{i+1}} \equiv \bigvee_{\tau_j} \left(\rho_{S_i} \wedge \rho_{S_{\mu_i}} \text{ entails } \bigwedge \Phi_{jk} \; \Psi_{jk} \right) \wedge \rho_{\Sigma} \wedge \rho_{O_{i+1}}$$

where transition τ_j is specified by a conjunction of formulas $\Phi_{jk} \Rightarrow next(\Psi_{jk})$

- MR: Characterizing possible commands

$$M_i = \left\{ \mu_j \middle| \rho_{S_i} \wedge \rho_{\mu_j} \text{ is consistent and } \left(\rho_{S_i} \wedge \rho_{\mu_j} \text{ entails } \bigwedge \Phi_{nk} \; \Psi_{nk} \right) \wedge \rho_{\Sigma} \wedge \rho_{g_i} \text{ entails } \rho_{g_i} \right\}$$

Statistically Optimal Configuration Management

Statistical Mode Identification

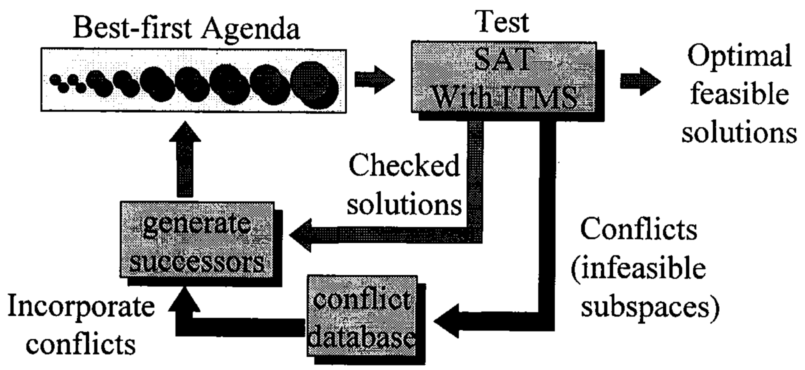
- $p(\tau_j \mid o_i) = p(o_i \mid \tau_j)\, p(\tau_j) / p(o_i)$ Bayes Rule
  $\propto p(o_i \mid \tau_j)\, p(\tau_j)$

- $p(o_i \mid \tau_j)$ is approximated from the model
  - $p(o_i \mid \tau_j) = 1$ if $\tau_j(a_{i-1})$ entails o_i
  - $p(o_i \mid \tau_j) = 0$ if $\tau_j(a_{i-1})$ is inconsistent with o_i
  - $p(o_i \mid \tau_j) = ?$ otherwise

Optimal Mode Reconfiguration

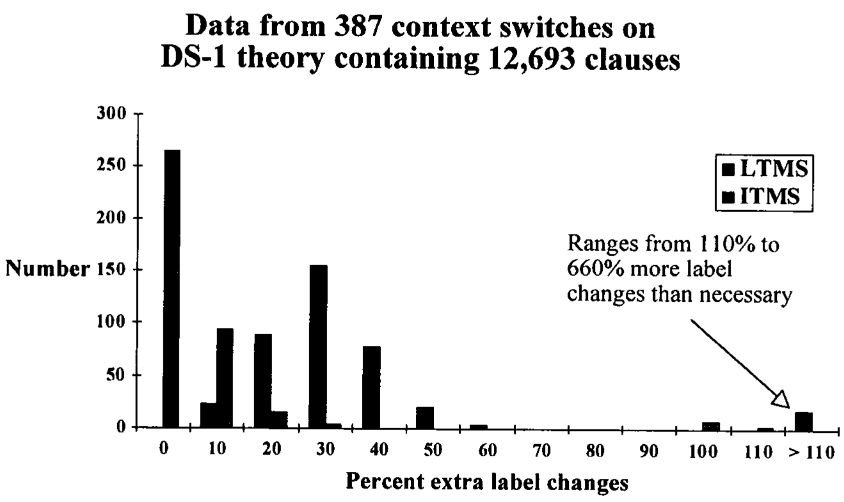
$\mu_i = argmin\ c(\mu_i')\ s.t.\ \mu_i'\ in\ \mathrm{M}_i$

OPSAT: RISC-like Best-first, Deductive Core



- **Conflicts and unit propagation highly focus search**
  - **(< 10 states visited per test)**
- **Unit propagation dominates, TMS significantly reduces.**

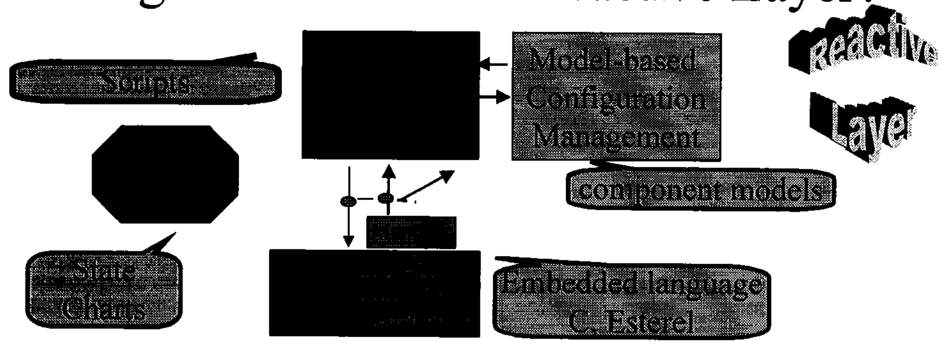
ITMS incremental unit propagation close to ideal



Outline

" Remote Agents and Deep Space One
" Model-based Autonomy
" Livingstone:Model-based Configuration Manager (with Pandu Nayak).
• Unifying Model-based and Reactive Programming (with Vineet Gupta).
• Conclusions

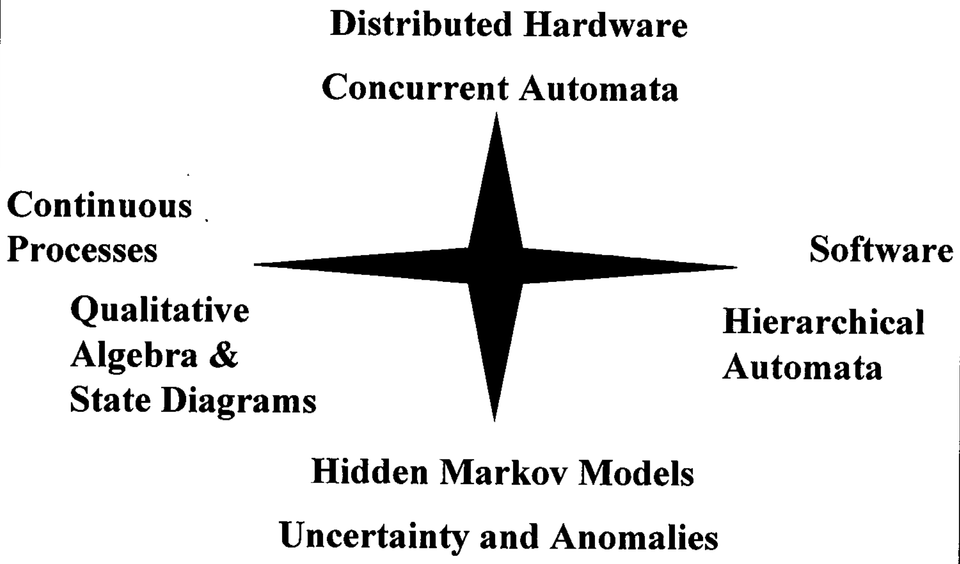
Does a Unified Representation and Paradigm Exist ... at the Reactive Layer?



Whats Missing from Livingstone's language?

- Goal Decomposition
- Method Selection
- Concurrency
- Preemption
- Strong Typing
- Encapsulations
- Polymorphism
- Inheritance ...

A Unified Modeling Paradigm



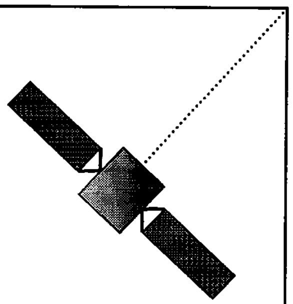
The Model-based Programming Language

Large-Scale Modeling Requires Good, Familiar Encapsulation Mechanisms.

```
class Valve {
  public ValveCmd cmdIn;
  private ValveFlow inflow, outflow;
  private ValveNominalModes mode;
  valve () {
      when (mode=open) {inflow = outflow;
                              when (cmdIn=close)
                                  next mode=closed;}
      when (mode=closed) { inflow = 0;
                                outflow = 0;
                                when (cmdIn=open)
                                   next mode=open;}}
}
```

Supports:
- –Familiar syntax
- –encapsulation
- –polymorphism
- –strong typing
- – multiple inheritance...

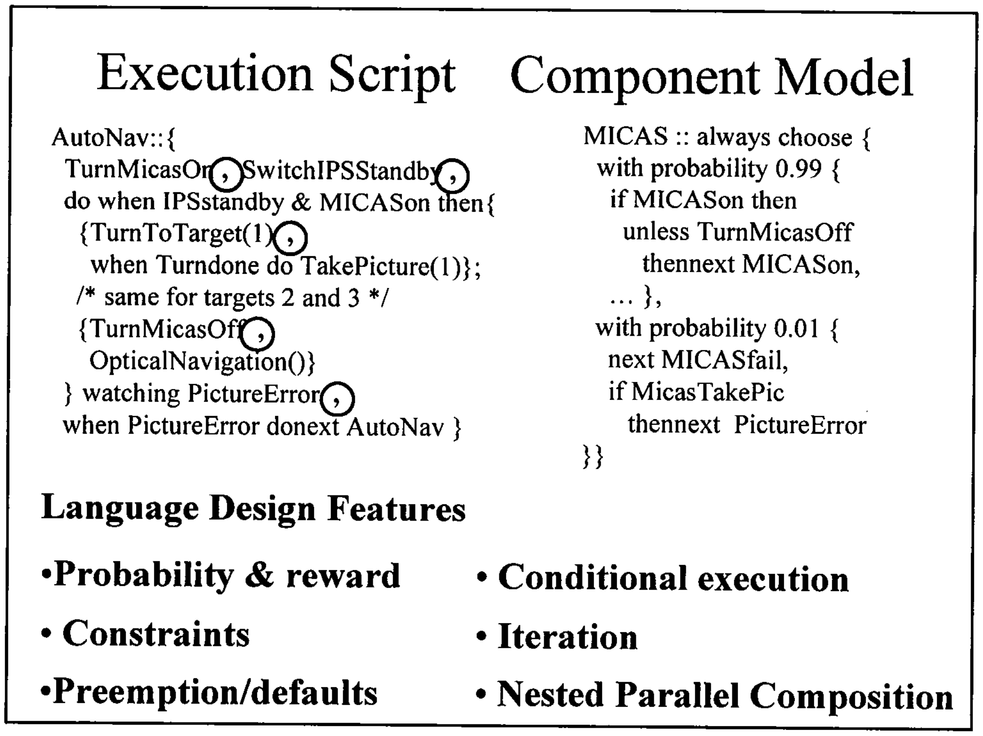
To model software MPL must support complex imperative behaviors.



DS1 Optical Navigation:

- Switch engine off, turn MICAS on
- Do:
  - Turn to first asteroid, take picture
  - Turn to second asteroid, take picture
  - Turn to third asteroid, take picture

  Watching if thrusters are broken, then switch to alternate set and repeat.
- Compute current position and course correction Watching if pictures are corrupted, then repeat sequence.



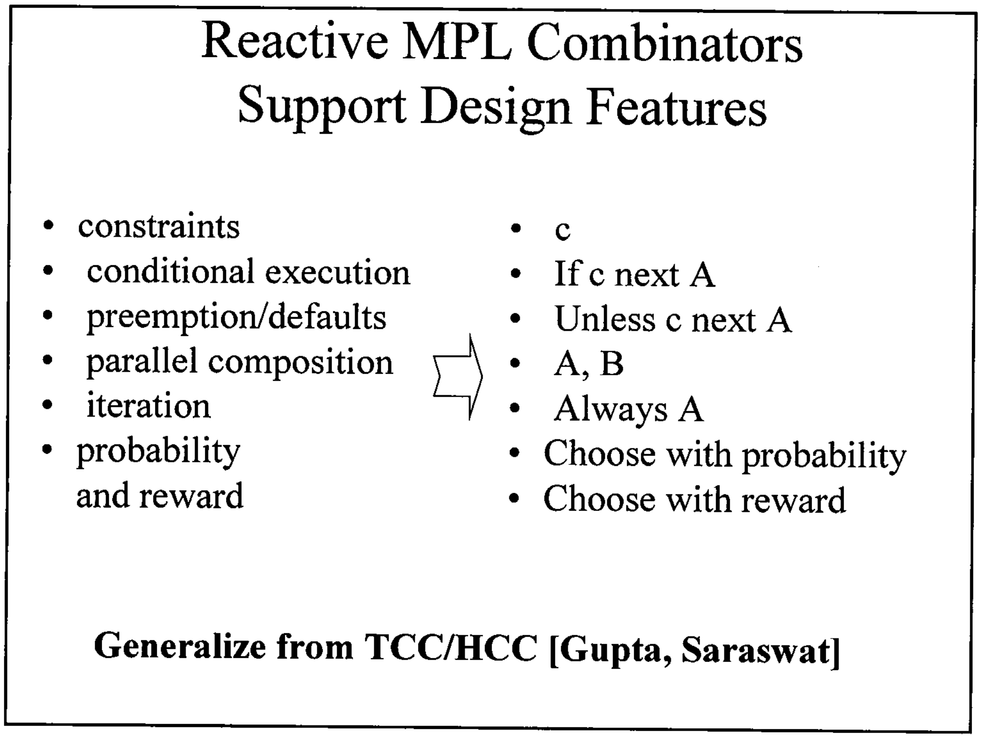
Execution Script   Component Model

```
AutoNav::{
 TurnMicasOn , SwitchIPSStandby ,
 do when IPSstandby & MICASon then{
  {TurnToTarget(1) ,
   when Turndone do TakePicture(1)};
  /* same for targets 2 and 3 */
  {TurnMicasOff ,
   OpticalNavigation()}
 } watching PictureError ,
 when PictureError donext AutoNav }
```

```
MICAS :: always choose {
 with probability 0.99 {
  if MICASon then
   unless TurnMicasOff
     thennext MICASon,
  ... },
 with probability 0.01 {
  next MICASfail,
  if MicasTakePic
    thennext  PictureError
}}
```

Language Design Features

- **Probability & reward**
- **Constraints**
- **Preemption/defaults**
- **Conditional execution**
- **Iteration**
- **Nested Parallel Composition**



Reactive MPL Combinators Support Design Features

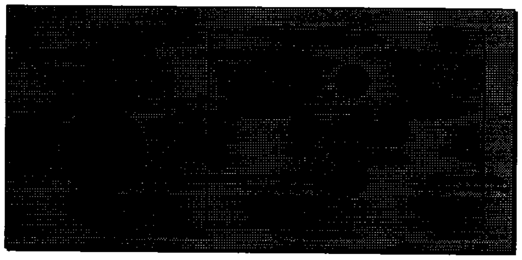
• constraints	• c
• conditional execution	• If c next A
• preemption/defaults	• Unless c next A
• parallel composition	• A, B
• iteration	• Always A
• probability and reward	• Choose with probability
	• Choose with reward

Generalize from TCC/HCC [Gupta, Saraswat]

Combinators of synchronous languages like Esterel can be derived from RMPL:

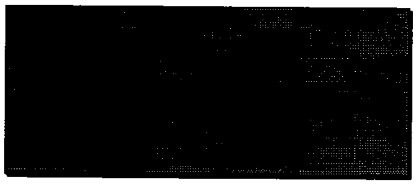
- do A watching c
- suspend A on c reactivate on d
- A; B
- **next** A
- **P ::= A[P]**
- when c do next A
- when c do A
- If c then A

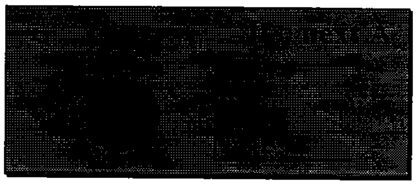
Encoding RMPL using Hierarchical Probabilistic Constraint Automata



- Generalize from concurrent constraint automata
- States may be automata (composite states).
- Multiple start and current states for each automata.
- Conditions allowed on negative information (not entailed).
- Transitions are distributions over next states, with rewards.
- Transitions from primitive states only.
- Constraints associated with primitive states only.

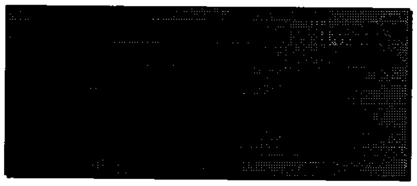
Combinators: Preemption & Conditional Execution

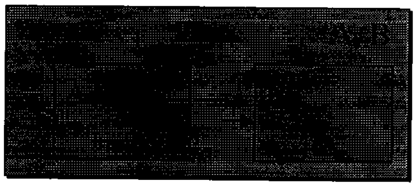




- Introduces non-monotonicity, but is stratified.
- Avoids causal paradoxes found in languages like Esterel.
- Only combinator using negative information.
- Livingstone used material implication to encode:
  - not c or next A
- RBurton uses intuitionistic interpretation,

Combinators: Iteration & Parallel Composition

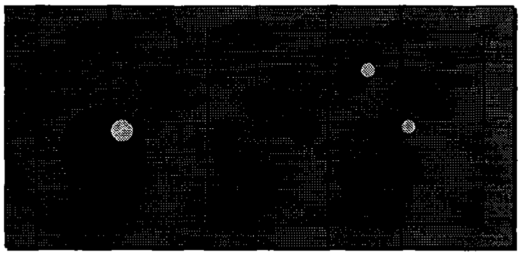




- Starts new copy of A at each time instant.
- The ability to mark multiple states simultaneously enables a compact encoding.

Executing Deterministic HCA

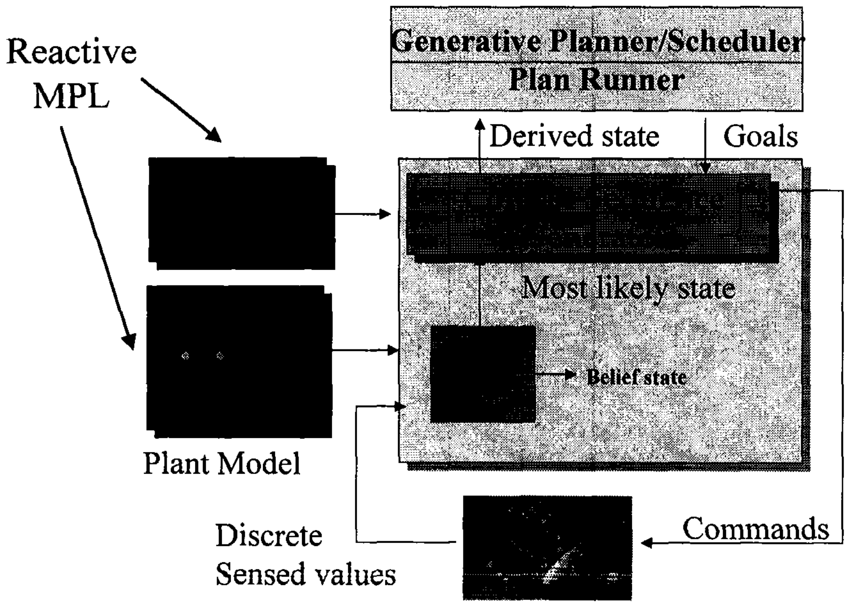
$\theta = \{c,d,e\}$



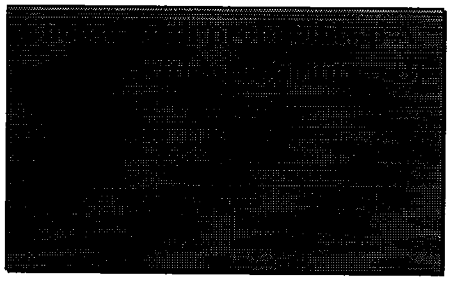
Given marking:

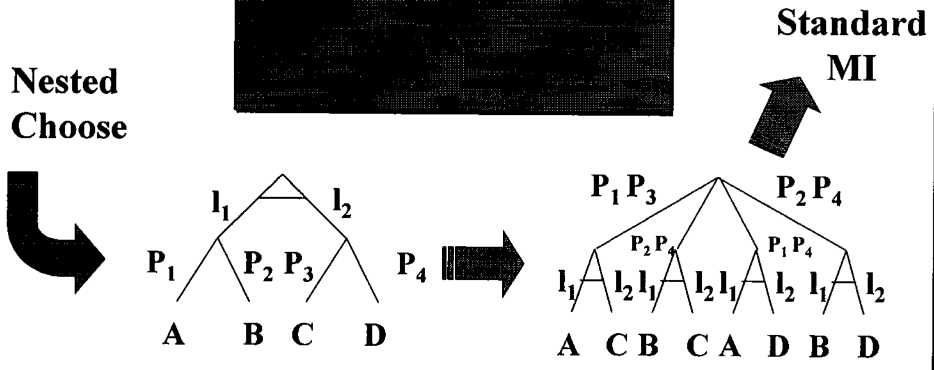
- Mark start states of all newly marked composite states.
- Let theory θ be the set of constraints of all marked states.
- Find enabled transitions of marked states.
  - Label d is satisfied if d is entailed by θ
  - Label $\bar{d}$ is satisfied if d is not entailed by θ
- Take Transition

RBurton: Model-directed Executive



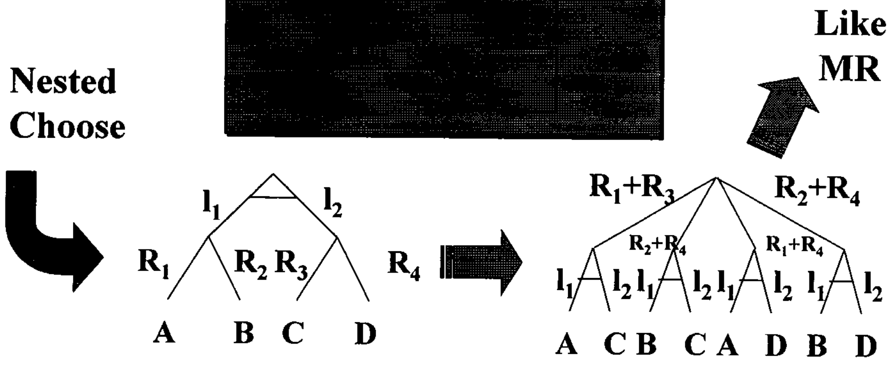
Probabilistic Execution: Enhanced MI



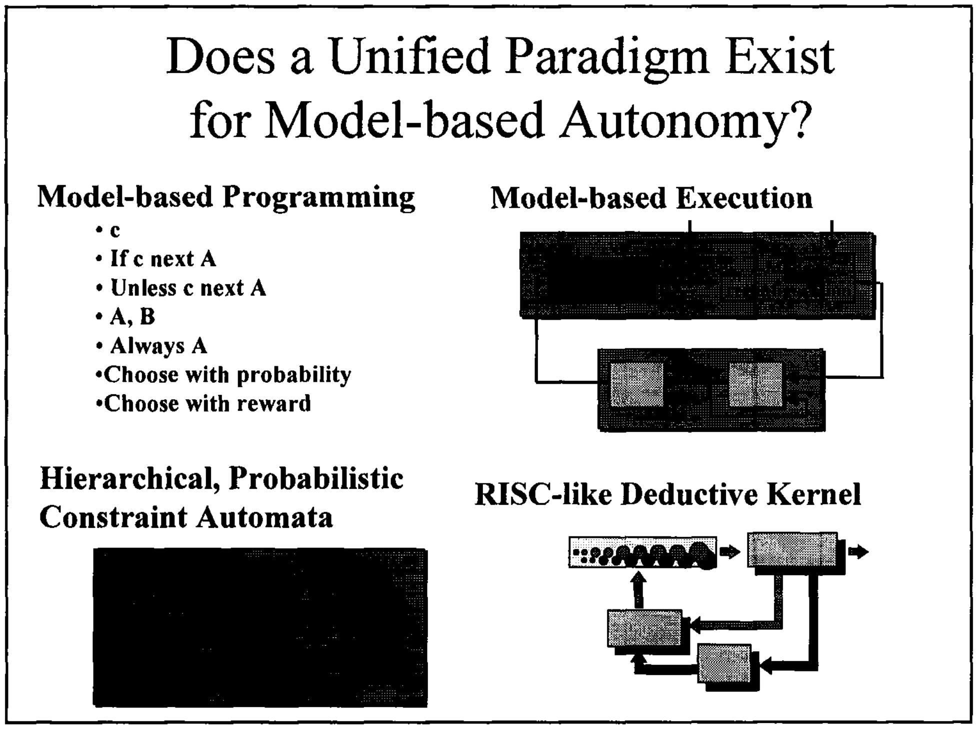


Encoding of a partially observable Markov Process

Sequence Combinator: Decision Theoretic Choice



Encoding of a partially observable Markov Decision Process

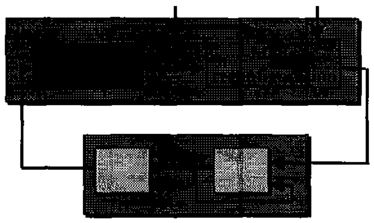


Does a Unified Paradigm Exist for Model-based Autonomy?

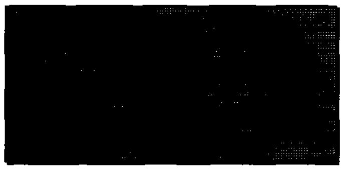
Model-based Programming

- **c**
- **If c next A**
- **Unless c next A**
- **A, B**
- **Always A**
- **Choose with probability**
- **Choose with reward**

Model-based Execution



Hierarchical, Probabilistic Constraint Automata



RISC-like Deductive Kernel