Model-based Programming of Cooperating Explorers

Brian C. Williams
CSAIL
Dept. Aeronautics and Astronautics
Massachusetts Institute of Technology
Programming Long-lived Embedded Systems

Large collections of devices must work in concert to achieve goals

- Devices indirectly observed and controlled
- Need quick, robust response to anomalies throughout life
- Must manage large levels of redundancy
Coordination Recapitulated At The Level of Cooperating Explorers

(Courtesy of Jonathan How. Used with permission.)
Coordination Issues Increase For Dexterous Explorers

(Courtesy of Frank Kirchner. Used with permission.)
Outline

• Model-based Programming
• Autonomous Engineering Operations
  – An Example
  – Model based Execution
  – Fast Reasoning using Conflicts
• Cooperating Mobile Vehicles
  – Predictive Strategy Selection
  – Planning Out The Strategy
Approach

Elevate programming and operation to system-level coaching.

- Model-based Programming
  - State Aware: Coordinates behavior at the level of intended state.

- Model-based Execution
  - Fault Aware: Uses models to achieve intended behavior under normal and faulty conditions.
Why Model-based Programming?

Polar Lander Leading Diagnosis:

- Legs deployed during descent.
- Noise spike on leg sensors latched by software monitors.
- Laser altimeter registers 40m.
- Begins polling leg monitors to determine touch down.
- Read latched noise spike as touchdown.
- Engine shutdown at ~40m.

Programmers often make commonsense mistakes when reasoning about hidden state.

Objective: Support programmers with embedded languages that avoid these mistakes, by reasoning about hidden state automatically.

Reactive Model-based Programming Language (RMPL)
Model-based Programs
Interact Directly with State

Embedded programs interact with plant sensors and actuators:
- Read sensors
- Set actuators

Model-based programs interact with plant state:
- Read state
- Write state

Programmer must map between state and sensors/actuators.

Model-based executive maps between state and sensors/actuators.
RMPL Model-based Program

Control Program
- Executes concurrently
- Preempts
- Queries (hidden) states
- Asserts (hidden) state

System Model

Titan Model-based Executive

Generates target goal states conditioned on state estimates

State estimates
- Tracks likely plant states
- Tracks least cost goal states

Observations

Commands

Plant

Control Program

- inflow = outflow = 0
Outline

• Model-based Programming
• Autonomous Engineering Operations
  – An Example
  – Model based Execution
  – Fast Reasoning using Conflicts
• Cooperating Mobile Vehicles
  – Predictive Strategy Selection
  – Planning Out The Strategy
Motivation

Mission-critical sequences:

- Launch & deployment ★★★
- Planetary fly-by ★★★
- Orbital insertion ★★★
- Entry, descent & landing ★★★

(Courtesy of Mitch Ingham. Used with permission.)
Mars Entry Example

(Courtesy of Mitch Ingham. Used with permission.)
Mars Entry Example

Descent engine to "standby":
- Engine to standby
- Planetary approach
- Switch to inertial nav
- Rotate to entry-orient & hold attitude
- Separate lander

30-60 sec

(Courtesy of Mitch Ingham. Used with permission.)
Mars Entry Example

Spacecraft approach:
- 270 mins delay
- relative position wrt Mars not observable
- based on ground computations of cruise trajectory

(Courtesy of Mitch Ingham. Used with permission.)
Mars Entry Example

engine to standby

planetary approach

switch to inertial nav

rotate to entry-orient & hold attitude

separate lander

Switch navigation mode:

“Inertial” = IMU only

(Courtesy of Mitch Ingham. Used with permission.)
Mars Entry Example

engine to standby

planetary approach

switch to inertial nav

rotate to entry-orient & hold attitude

separate lander

Rotate spacecraft:

• command ACS to entry orientation

(Courtesy of Mitch Ingham. Used with permission.)
Mars Entry Example

engine to standby

planetary approach

switch to inertial nav

rotate to entry-orient & hold attitude

separate lander

Rotate spacecraft:
• once entry orientation achieved, ACS holds attitude

(Courtesy of Mitch Ingham. Used with permission.)
Mars Entry Example

- Engine to standby
- Planetary approach
- Switch to inertial nav
- Rotate to entry-orient & hold attitude
- Separate lander from cruise stage:
  - Pyro latches
  - Cruise stage
  - Lander stage

(Courtesy of Mitch Ingham. Used with permission.)
Mars Entry Example

engine to standby

planetary approach

switch to inertial nav

rotate to entry-orient & hold attitude

Separate lander from cruise stage:

• when entry orientation achieved, fire primary pyro latch

(Courtesy of Mitch Ingham. Used with permission.)
Mars Entry Example

engine to standby
planetary approach
switch to inertial nav
rotate to entry-orient & hold attitude

Separate lander from cruise stage:
- when entry orientation achieved, fire primary pyro latch

(Courtesy of Mitch Ingham. Used with permission.)
Mars Entry Example

engine to standby

planetary approach

switch to inertial nav

rotate to entry-orient & hold attitude

Separate lander from cruise stage:

• in case of failure of primary latch, fire backup pyro latch

(Courtesy of Mitch Ingham. Used with permission.)
Mars Entry Example

- Engine to standby
- Planetary approach
- Switch to inertial nav
- Rotate to entry-orient & hold attitude

Separate lander from cruise stage:
- In case of failure of primary latch, fire backup pyro latch

(Courtesy of Mitch Ingham. Used with permission.)
What is Required to Program at This Level?

- simple state-based control specifications
- models are writable/inspectable by systems engineers
- handle timed plant & control behavior
- automated reasoning through low-level plant interactions
- fault-aware (in-the-loop recoveries)

(Courtesy of Mitch Ingham. Used with permission.)
Descent Example

Turn camera off and engine on

EngineA  EngineB

Science Camera

EngineA  EngineB

Science Camera
Control program specifies state trajectories:

• fires one of two engines
• sets both engines to ‘standby’
• prior to firing engine, camera must be turned off to avoid plume contamination
• in case of primary engine failure, fire backup engine instead

Plant Model describes behavior of each component:

– Nominal and Off nominal
– qualitative constraints
– likelihoods and costs

OrbitInsert()::

(\text{do-watching} \ ((\text{EngineA} = \text{Thrusting}) \ \text{OR} \ (\text{EngineB} = \text{Thrusting}))

(\text{parallel}

(\text{EngineA} = \text{Standby})
(\text{EngineB} = \text{Standby})
(\text{Camera} = \text{Off})

(\text{do-watching} \ (\text{EngineA} = \text{Failed})

(\text{when-donext} \ ((\text{EngineA} = \text{Standby}) \ \text{AND} \ (\text{Camera} = \text{Off}) \ )

(\text{EngineA} = \text{Thrusting})\\)

(\text{when-donext} \ ((\text{EngineA} = \text{Failed}) \ \text{AND} \ (\text{EngineB} = \text{Standby}) \ \text{AND}
(\text{Camera} = \text{Off}) \ )

(\text{EngineB} = \text{Thrusting})\\))
Plant Model

cOMPONENT MODES...

described by finite domain constraints on variables...

deterministic and probabilistic transitions

cost/reward

Engine Model

Camera Model

One per component ... operating concurrently
Example: The model-based program sets \textit{engine} = \textit{thrusting}, and the deductive controller . . . .

\begin{itemize}
  \item \textbf{Mode Estimation}
  \begin{itemize}
    \item Oxidizer tank
    \item Deduces that thrust is off, and the engine is healthy
  \end{itemize}
  \begin{itemize}
    \item Fuel tank
    \item Selects valve configuration; plans actions to open six valves
  \end{itemize}

  \item \textbf{Mode Reconfiguration}
  \begin{itemize}
    \item Determines valves on backup engine that will achieve thrust, and plans needed actions.
  \end{itemize}

  \item \textbf{Mode Estimation}
  \begin{itemize}
    \item Deduces that a valve failed - stuck closed
  \end{itemize}
\end{itemize}
Outline

• Model-based Programming

• Autonomous Engineering Operations
  – An Example
  – Model based Execution
  – Fast Reasoning using Conflicts

• Cooperating Mobile Vehicles
  – Predictive Strategy Selection
  – Planning Out The Strategy
Modeling Plant Dynamics using Probabilistic Concurrent, Constraint Automata (PCCA)

Compact Encoding:

– Concurrent probabilistic transitions
– State constraints between variables

Typical Example (DS1 spacecraft):

– 80 Automata, 5 modes on average
– 3000 propositional variables, 12,000 propositional clauses
The Plant’s Behavior

- Assigns a value to each variable (e.g., 3,000 vars).
- Consistent with all state constraints (e.g., 12,000).

- A set of concurrent transitions, one per automata (e.g., 80).
- Previous & Next states consistent with source & target of transitions.
Control Sequencer:
Generates goal states conditioned on state estimates

Mode Estimation:
Tracks likely States

System Model:
• Executes concurrently
• Preempts
• Asserts and queries states
• Chooses based on reward

Control Program:
OrbitInsert():
(do-watching (EngineA = Firing) OR (EngineB = Firing))
(parallel
(EngineA = Standby)
(EngineB = Standby)
(Camera = Off)
(do-watching (EngineA = Failed)
(when-donext ( (EngineA = Standby) AND (Camera = Off) )
(EngineA = Firing))
(when-donext ( (EngineA = Failed) AND (EngineB = Standby) AND (Camera = Off) )
(EngineB = Firing)))))

Titan Model-based Executive

State estimates
State goals
RMPL Model-based Program

Control Program
- Executes concurrently
- Preempts
- Asserts and queries states
- Chooses based on reward

System Model

Titan Model-based Executive

Control Sequencer:
Generates goal states conditioned on state estimates

Mode Estimation:
Tracks likely States

Mode Reconfiguration:
Tracks least-cost state goals

State estimates

State goals

Valve fails stuck closed
Fire backup engine

Current Belief State

First Action

least cost reachable goal state
arg max $P_T(m')$
\[\text{s.t. } M(m') \land O(m') \text{ is satisfiable}\]

OpSat:
arg min $f(x)$
\[\text{s.t. } C(x) \text{ is satisfiable}\]
\[D(x) \text{ is unsatisfiable}\]

arg min $R_{T^*}(m')$
\[\text{s.t. } M(m') \text{ entails } G(m')\]
\[\text{s.t. } M(m') \text{ is satisfiable}\]
Outline

• Model-based Programming
• Autonomous Engineering Operations
  – An Example
  – Model based Execution
  – Fast Reasoning using Conflicts
• Cooperating Mobile Vehicles
  – Predictive Strategy Selection
  – Planning Out The Strategy
**Consistency-based Diagnosis:** Given symptoms, find diagnoses that are consistent with symptoms.

**Handle Novel Failures by Suspending Constraints:** Make no presumptions about faulty component behavior.

![Diagram of Diagnosis Formulation]

- **Symptom:** 1
- **Or1:** A, B, C, D, E
- **Or2:** B, C, D, E
- **Or3:** C, D, E
- **And1:** X, Y
- **And2:** Z
- **F:** 0
- **G:** 1
**Diagnosis Formulation**

**Consistency-based Diagnosis:** Given symptoms, find diagnoses that are consistent with symptoms.

**Handle Novel Failures by Suspending Constraints:** Make no presumptions about faulty component behavior.

![Diagram of diagnosis formulation](image)
Fast Reasoning Through Conflict

When you have eliminated the impossible, whatever remains, however improbable, must be the truth.

- Sherlock Holmes. The Sign of the Four.

1. Test Hypothesis
2. If inconsistent, learn reason for inconsistency (a Conflict).
3. Use conflicts to leap over similarly infeasible options to next best hypothesis.
Compare Most Likely Hypothesis to Observations

It is most likely that all components are okay.
The red component modes conflict with the model and observations.
Leap to the Next Most Likely Hypothesis that Resolves the Conflict

The next hypothesis must remove the conflict
New Hypothesis Exposes Additional Conflicts

Pressure_1 = nominal
Pressure_2 = nominal

Acceleration = zero

Another conflict, try removing both
Final Hypothesis Resolves all Conflicts

Implementation: Conflict-directed A* search.
Increasing Cost

A* (Algorithm)

Feasible

Infeasible
Conflict-directed A*
Conflict-directed A*

Increasing Cost

Conflict 1

Infeasible

Feasible
Conflict-directed A*

Increasing Cost

- Conflicts are mapped to feasible regions as implicants (Kernel Assignments)

- Want kernel assignment containing the best cost state.
Outline

- Model-based Programming
- Autonomous Engineering Operations
  - An Example
  - Model based Execution
  - Fast Reasoning using Conflicts
- Cooperating Mobile Vehicles
  - Predictive Strategy Selection
  - Planning Out The Strategy
Coordination is Recapitulated at the Level of Cooperating Explorers

(Courtesy of Jonathan How. Used with permission.)
• Explicit human guidance is at the lowest levels
What types of reasoning should the programmer/operator guide?

- State/mode inference
- Machine control
- Scheduling
- Method selection
- Roadmap path planning
- Optimal trajectory planning
- Generative temporal planning
RMPL Model-based Program  Kirk Model-based Executive

Control Program
- Executes concurrently
- Preempts
- non-deterministic choice
- \( A[l,u] \) timing
- \( A \) at location

Environment Model

Control Sequencer
- Predictive Strategy Selection
- Dynamic Scheduling
- Ensures Safe Execution

Deductive Controller
- Achieves State via Path Planning
- Estimates using Localization

Observations

Commands

Plant
Example Scenario

Mars rover operators have been leery of generative planners.
- Are more comfortable with specifying contingencies.
- Want strong guarantees of safety and robust to uncertainty.
- Global path planning is on the edge

Extend RMPL with planner-like capabilities . . . except planning
Reactive Model-based Programming

Idea: To describe group behaviors, start with concurrent language:

- \( p \)
- If \( c \) next \( A \)
- Unless \( c \) next \( A \)
- \( A, B \)
- Always \( A \)

- Add temporal constraints:
  - \( A \ [l,u] \)
  - Timing

- Add choice (non-deterministic or decision-theoretic):
  - Choose \( \{A, B\} \)
  - Contingency

- Parameterize by location:
  - \( A \) at \([l]\)
Example Enroute Activity:

Enroute

Rendezvous

Corridor 2

Corridor 1

Rescue Area
RMPL for Group-Enroute

\[
\text{Group-Enroute()} [1, u] = \{
\]
\[
    \text{choose} \ {\}
\]
\[
        \text{do} \ {\}
\]
\[
            \text{Group-Fly-Path(PATH1_1,PATH1_2,PATH1_3,RE_POS)} [1*90\% , u*90\% ];
\]
\[
                \text{maintaining PATH1_OK},
\]
\[
            \text{do} \ {\}
\]
\[
                \text{Group-Fly-Path(PATH2_1,PATH2_2,PATH2_3,RE_POS)} [1*90\% , u*90\% ];
\]
\[
                    \text{maintaining PATH2_OK}
\]
\[
\}
\]
\[
\{ \]
\[
    \text{Group-Transmit(OPS,ARRIVED)} [0, 2],
\]
\[
    \text{do} \ {\}
\]
\[
        \text{Group-Wait(HOLD1,HOLD2)} [0, u*10\% ]
\]
\[
            \text{watching PROCEED}
\]
\[
\} \text{ at RE_POS}
\]

Temporal Constraints:
RMPL for Group-Enroute

Location Constraints:

Group-Enroute()[l,u] = {
    choose {
        do {
            Group-Fly-
            Path(PATH1_1,PATH1_2,PATH1_3,RE_POS)[l*90%,u*90%];
            } maintaining PATH1_OK,
            do {
                Group-Fly-
                Path(PATH2_1,PATH2_2,PATH2_3,RE_POS)[l*90%,u*90%];
            } maintaining PATH2_OK
        };
        {
            Group-Transmit(OPS,ARRIVED)[0,2],
            do {
                Group-Wait(HOLD1,HOLD2)[0,u*10%]
            } watching PROCEED
        } at RE_POS
    }
Group-Enroute()\texttt{[l,u]} = \{ 
  \textbf{choose} \{ 
    \textbf{do} \{ 
      \textbf{Group-Traversal-Path} (PATH1\_1,PATH1\_2,PATH1\_3,RE\_POS) \texttt{[l*90\%,u*90\%]};
    \}\textbf{maintaining PATH1\_OK},
    \textbf{do} \{ 
      \textbf{Group-Traversal-Path} (PATH2\_1,PATH2\_2,PATH2\_3,RE\_POS) \texttt{[l*90\%,u*90\%]};
    \}\textbf{maintaining PATH2\_OK}
  \} \\
  \{ 
    \textbf{Group-Transmit} (OPS,ARRIVED) \texttt{[0,2]},
    \textbf{do} \{ 
      \textbf{Group-Wait} (HOLD1,HOLD2) \texttt{[0,u*10\%]}
    \}\textbf{watching PROCEED}
  \} \textbf{at RE\_POS}
\}
Outline

• Model-based Programming
• Autonomous Engineering Operations
  – An Example
  – Model based Execution
  – Fast Reasoning using Conflicts
• Cooperating Mobile Vehicles
  – Predictive Strategy Selection
  – Planning Out The Strategy
Control Program
- Executes concurrently
- Preempts
- non-deterministic choice
- A[1,u] timing
- A at l location

Environment Model

RMPL Model-based Program  Titan Model-based Executive

Selects consistent threads of activity from redundant methods

Executive
- pre-plans activities
- pre-plans paths
- dynamically schedules [Tsmardinos et al.]

Tracks location

Finds least cost paths

Observations

Plant

Commands

location goals

location estimates
Enroute Activity Encoded as a Temporal Plan Network

- Start with flexible plan representation

![Temporal Plan Network Diagram](image-url)
Enroute Activity Encoded as a Temporal Plan Network

- Add conditional nodes
Enroute Activity Encoded as a Temporal Plan Network

- Add temporally extended, symbolic constraints

![Diagram of Enroute Activity](image_url)
Instantiated Enroute Activity

- Add environmental constraints

**Group-Enroute**

[500,800]

1. **Ask**(PATH1=OK)
2. **Tell**(PATH1=OK)
3. **Ask**(PATH2=OK)
4. **Tell**(PROCEED)
5. **Ask**(PROCEED)
6. **Tell**(PATH1=OK)
7. **Tell**(PROCEED)
8. **Tell**(PROCEED)

- Activity (or sub-activity)
- Duration (temporal constraint)
- Conditional node
- Symbolic constraint (Ask,Tell)
- External constraints
Generates Schedulable Plan

To Plan, . . . perform the following hierarchically:

- **Trace trajectories**
- **Check schedulability**
  - **Supporting and protecting goals (Asks)**
Supporting and Protecting Goals

Unsupported Subgoal

- Goal: any UCAV at Target
- Activity: UCAV1 at Target

Threatened Activities

- Activity: UAV1 at Base
- Activity: UAV1 at Target

Close open goals

Activities can’t co-occur

Resolving Unsupported Subgoals:
- Scan plan graph, identifying activities that support open sub-goals; force to co-occur.

Resolving Threatened Subgoals:
- Search for inconsistent activities that co-occur, and impose ordering.

Key computation is bound time of occurrence:
- Used Floyd-Warshall APSP algorithm $O(V^3)$. 
Randomized Experiments for Assessing Scaling and Robustness

Randomized Experiments:

- Randomly generated range of scenarios with **1-50 vehicles**.
- Each vehicle has **two scenario options**, each with **five actions** and **2 waypoints**:
  1. Go to waypoint 1
  2. Observe science
  3. Go to waypoint 2
  4. Observe science
  5. Return to collection point

- **Waypoints generated randomly** from environment with uniform distribution.

Strategy Selection:

- TPN planner chooses **one option per vehicle**.
- **Combined choices** must be **consistent** with **timing constraints** and **vehicle paths**.
Kirk Strategy Selection: Scaling and Robustness

Each vehicle visits 2 science sites and returns to collection point

Performance Improvement Through
- Incremental temporal consistency
- Conflict-directed Search (in progress)
RMPL Model-based Program        Titan Model-based Executive

Control Program
- Executes concurrently
- Preempts
- non-deterministic choice
- $A[1,u]$ timing
- $A$ at $l$ location

Environment Model

Executive
- pre-plans activities
- pre-plans paths
- dynamically schedules [Tsmardinos et al.]

Selects consistent threads of activity from redundant methods

Tracks location
Finds least cost paths

location goals

location estimates

Observations

Commands

Plant

SCIENCE AREA 1'

Landing Site: ABC
Landing Site: XYZ
Achieving Program States Combines Logical Decisions and Trajectory Planning
Explorers Will Need to Be Dexterous

(Courtesy of Frank Kirchner. Used with permission.)
Outline

• Model-based Programming
• Autonomous Engineering Operations
  – An Example
  – Model based Execution
  – Fast Reasoning using Conflicts
• Cooperating Mobile Vehicles
  – Predictive Strategy Selection
  – Planning Out The Strategy
Example:
Coaching Heterogeneous Teams

- Search and Rescue
- Ocean Exploration

A dozen vehicles is too many to micro manage

→ Act as a coach:
  - Specify evolution of state and location.

(Courtesy of Jonathan How. Used with permission.)
Forest Fire Rescue

• Goal: retrieve family from fire.

• Rescue cannot take place until the local fire is suppressed.

• Retrofit one rescue vehicle for fire suppression
• **Strategy Selection** determines the optimal rules / strategies to accomplish mission goals.

• **Activity Planning** figures out how to achieve mission goals within strategic framework using available low-level actions.

Mission Developer

Strategy Macro Library

Strategy Macro decomposition

Mission Controller

TPN Planner

state configuration goals

• RMPL control program

Operators, Tactics, Scenario Model

environment and action data

Visibility Graph

Generative Activity Planner

Human / Computer Interface

MILP Path-Planning

schedulable plan with rationale
RMPL Control Program

- (defclass rescue-team
  
  (execute ()
    (sequence
      (parallel [l1,u1]
        (tell-start(at uav1 Ambulance))
        (tell-start(at uav2 Ambulance))
        (ask-end(suppressed Fire))
      )
      (parallel [l2,u2]
        (tell-start(at family RescuePoint))
        (ask-end(rescued family))
        (ask-end(at uav1 Ambulance))
        (ask-end(at uav2 Ambulance))
      )
    )
  )
)

Initial State

Intermediate State

Goal State

Phase 1

Phase 2
Environment Model

• Terrain Map

• Object instantiations:
  – UAV uav1
  – UAV uav2
  – RESCU-READY uav1
  – RESCUE-READY uav2
  – IN-DISTRESS family
  – LOCATION Ambulance
  – LOCATION Fire
  – LOCATION RescuePoint
Vehicle Specifications

- Vehicle linearized dynamics

- Vehicle primitive operators:
  - Fly(V,A,B)
    - move UAV “V” from location “A” to location “B”
  - Refit(V)
    - Prepare UAV “V” to drop fire retardant
  - Drop(V,A)
    - Drop fire retardant at location “A” with UAV “V”
  - Rescue(V,P,A)
    - Rescue people “P” in distress with UAV “V” at location “A”
**Strategy Selection**

determines the optimal rules / strategies to accomplish mission goals.

**Activity Planning**

figures out how to achieve mission goals within strategic framework using available low-level actions.

- **Mission Controller**
  - RMPL control program
  - \(\rightarrow\) Strategy Selection
  - \(\rightarrow\) Activity Planning

- **Strategy Selection**
  - TPN Planner
  - \(\rightarrow\) state configuration goals

- **Activity Planning**
  - Visibility Graph
  - Generative Activity Planner
  - \(\rightarrow\) schedulable plan with rationale

- **Mission Developer**
  - Strategy Macro Library
  - Strategy macro decomposition
  - \(\leftarrow\) environment and action data
  - Operators, Tactics, Scenario Model

- **Human / Computer Interface**
  - MILP Path-Planning

\(\rightarrow\) Human / Computer Interface
Kirk Constructs Vehicle Activity Plan Using a Generative Temporal Planner

Approach:
• Encode Goal Plan using an LPGP-style encoding
• Prototype using LPGP [Fox/Long, CP03]

Mission Goal State Plan

Translate to Planning Problem with Atomic Operators

Use Atomic Generative Planner (GraphPlan – Blum & Furst) To Generate Operators and Precedence

Extract Temporal Plan and Check Schedulability

Vehicle Activity Plan
Kirk extracts a least commitment plan and generates a rationale
Kirk Model-based Execution System Overview

Strategy Selection
- TPN Planner

Activity Planning
- Visibility Graph
- Generative Activity Planner

Mission Developer
- Strategy Macro Library
- Operators, Tactics, Scenario Model
- MILP Path-Planning
- Human / Computer Interface

- Strategy Selection determines the optimal rules / strategies to accomplish mission goals.
- Activity Planning figures out how to achieve mission goals within strategic framework using available low-level actions.
Plan layered with rationale

Rescue(UAV1, Troops, RSQ)
Refit(UAV1)
Fly(UAV1, Base, RSQ)
Fly(UAV1, RSQ, Base)

[10, +INF]  [20, +INF]  [30, 60]  [20, +INF]

[10, 20]  [0, 100]  [0, 100]

Control Program Phase I
Control Program Phase II

Fly(UAV2, Base, Radar)
Attack(UAV2, Radar)
Fly(UAV2, Radar, Base)
Kirk Ensures Plan Completeness, Consistency and Minimality

- **Complete Plan**
  - A plan is **complete** IFF every precondition of every activity is achieved.
  - An activity’s precondition is achieved IFF:
    - The precondition is the effect of a preceding activity (support), and
    - No intervening step conflicts with the precondition (mutex).

- **Consistent Plan**
  - The plan is **consistent** IFF the temporal constraints of its activities are consistent (the associated distance graph has no negative cycles), and
  - no conflicting (mutex) activities can co-occur.

- **Minimal Plan**
  - The plan is **minimal** IFF every constraint serves a purpose, *i.e.,*
    - If we remove any temporal or symbolic constraint from a minimal plan, the new plan is not equivalent to the original plan.
Plan-based HCI Proof of Concept:
Coaching through Coordinated Views

Activity UCAV2-FLY-TO-WAYPOINT beginning
UCAV2-FLY-TO-WAYPOINT establishes the following prerequisites:
UCAV2-FLY-TO-WAYPOINT helps establish (AT UCAV2 WAYPOINT) a prerequisite of UCAV2-FLY-TO-TARGET
UCAV2-FLY-TO-WAYPOINT establishes (AT UCAV2 WAYPOINT)

UCAV2 executes a FLY-TO operation on WAYPOINT1

Activity UCAV1-FLY-TO-SAM beginning
UCAV1-FLY-TO-SAM establishes the following prerequisites:
UCAV1-FLY-TO-SAM helps establish (AT UCAV1 SAM1) a prerequisite of UCAV1-ATTACK-SAM
UCAV1-FLY-TO-SAM establishes (AT UCAV1 SAM1)
UCAV1 executes a FLY-TO operation on SAM1

UCAV1-FLY-TO-SAM finished, go on? [default Yes]: 

(Courtesy of Howard Shrobe, Principal Research Scientist, MIT CSAIL. Used with permission.)
Plan & Geography View

Sequencing:

Action: FLY-TO  
Actor: UCAV2  
Objects: WAYPOINT1

Action: FLY-TO  
Actor: UCAV2  
Objects: HVT1

Action: ATTACK  
Actor: UCAV2  
Objects: HVT1 [10, 10]

Action: FLY-TO  
Actor: UCAV1  
Objects: SAM1 [5, NA]

Action: ATTACK  
Actor: UCAV1  
Objects: SAM1 [10, 10]

Action: FLY-TO  
Actor: UCAV1  
Objects: BASE1 [5, NA]

(Courtesy of Howard Shrobe, Principal Research Scientist, MIT CSAIL. Used with permission.)
Causal View

Causality

Explanation

UCAV2-ATTACK-TARGET Has the following prerequisites:
UCAV2-ATTACK-TARGET requires (AT UCAV2 HVT1)
This was established by
Activity UCAV2-FLY-TO-TARGET achieving (AT UCAV2 HVT1)

UCAV2-ATTACK-TARGET Establishes the following prerequisites:
UCAV2-ATTACK-TARGET helps establish (DESTROYED HVT1)
prerequisite of NEW
UCAV2-ATTACK-TARGET establishes (DESTROYED HVT1)

(Courtesy of Howard Shrobe, Principal Research Scientist, MIT CSAIL. Used with permission.)
Model-based Programming of Robust Robotic Networks

• Long-lived systems achieve robustness by coordinating a complex network of internal devices.

• Programmers make a myriad of mistakes when programming these autonomic processes.

• Model-based programming simplifies this task by elevating the programmer to the level of a coach:
  – Makes hidden states directly accessible to the programmer.
  – Automatically mapping between states, observables and control variables.

• Model-based executives reasoning quickly and extensively by exploiting conflicts.

• Mission-level executives combine activity planning, logical decision making and control into a single hybrid decision problem.