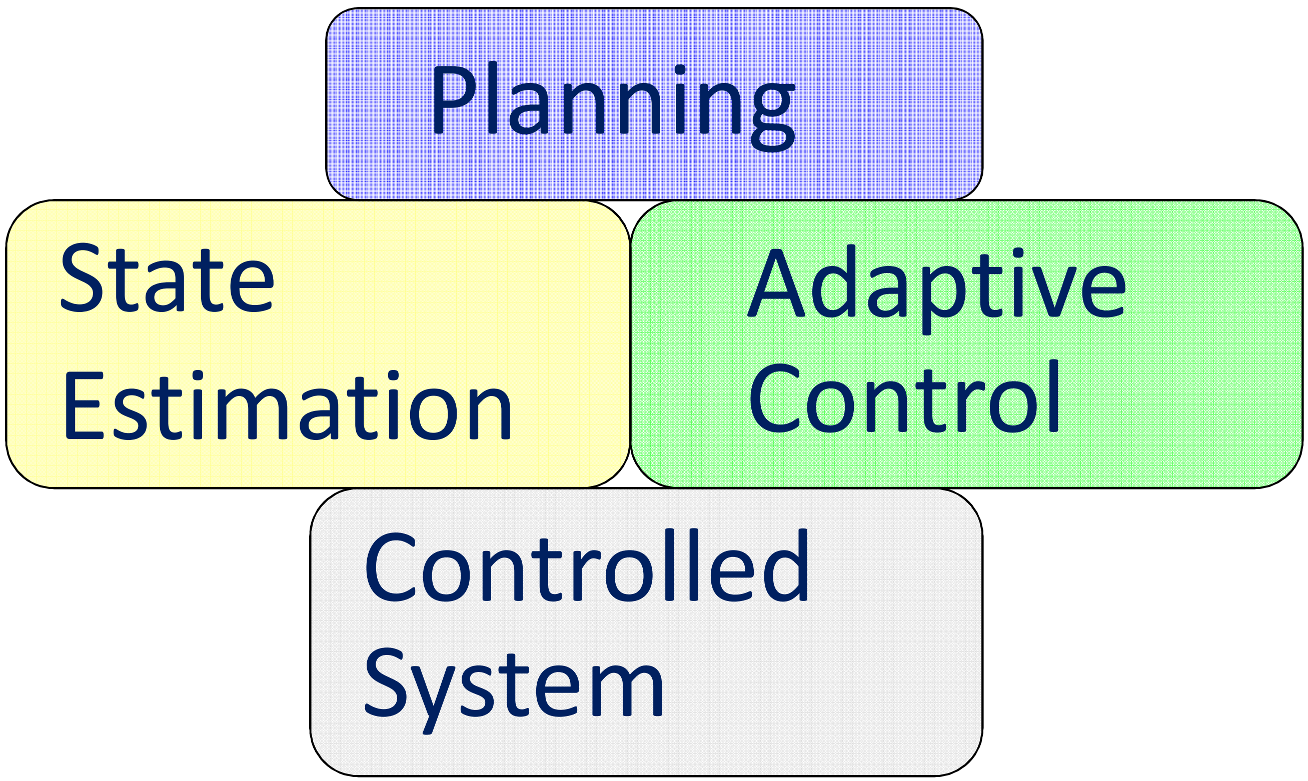
Intelliface

Interfacing Planning, Execution, and State Estimation Systems

Presented at: IEEE Aerospace 2015
Big Sky, MT
March 11, 2015

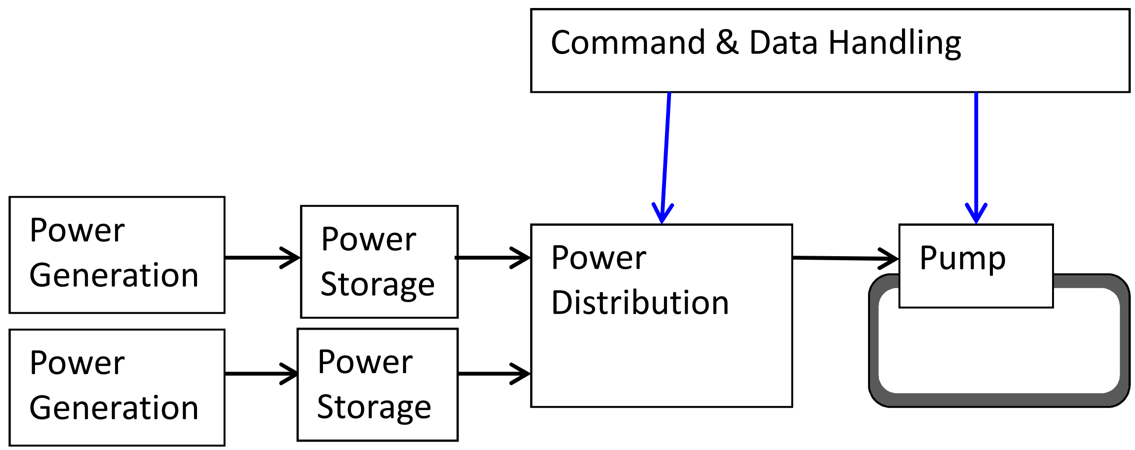
Stottler Henke
Smarter Software Solutions

Robust Autonomy Requires Integration



Modeling and Integration Challenges

Complex and Interdependent Resources



Must Revise Model When Assumptions Change

System capabilities and configuration
Resource capacities and usage patterns
Operating rules

Intelliface Project Goal

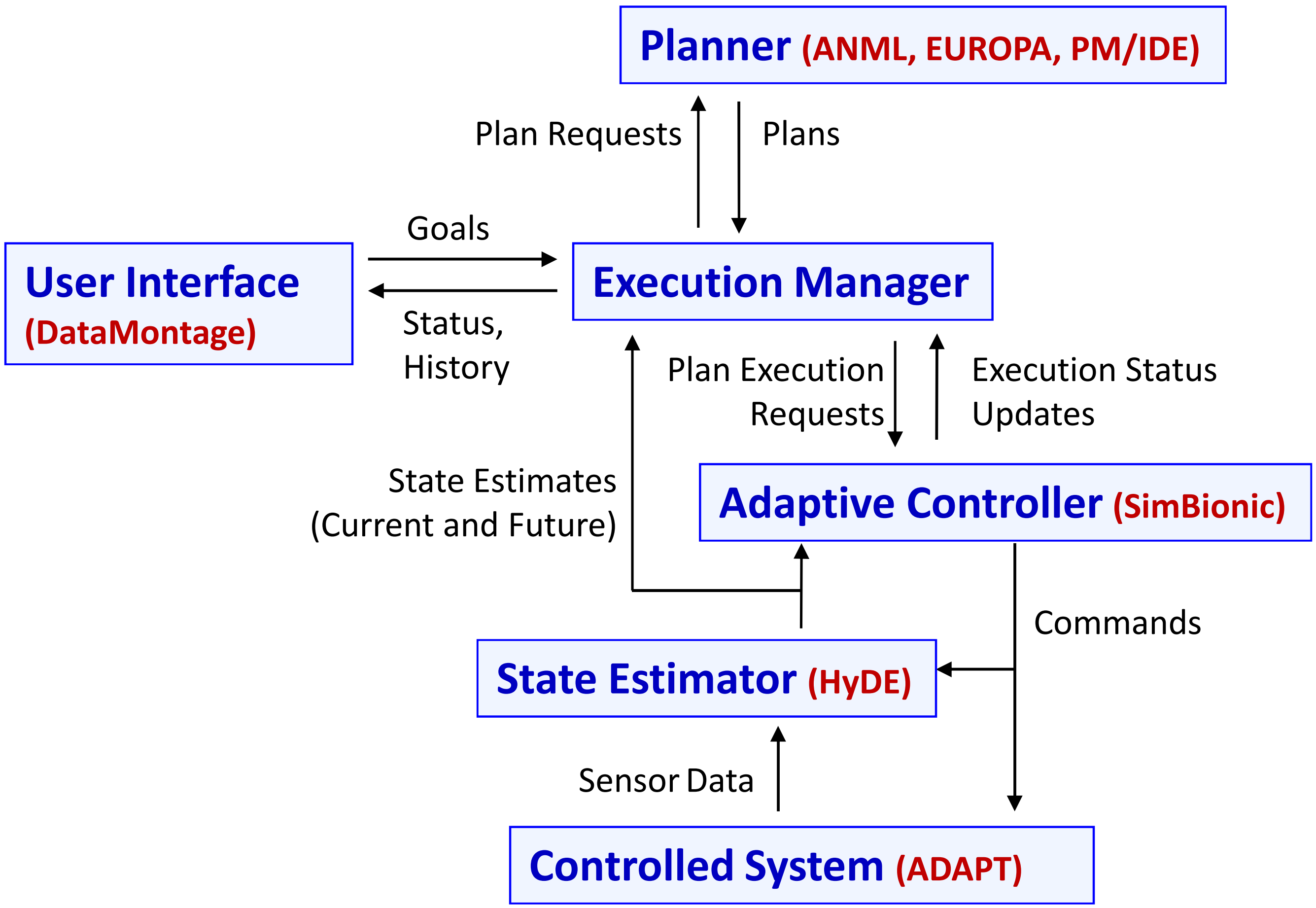
Integrate state estimation, planning, and execution systems quickly and consistently

Improve our understanding of the information exchanged among subsystems to support robust and efficient operations.

Design algorithms and data models that simplify the development of models and interfaces.

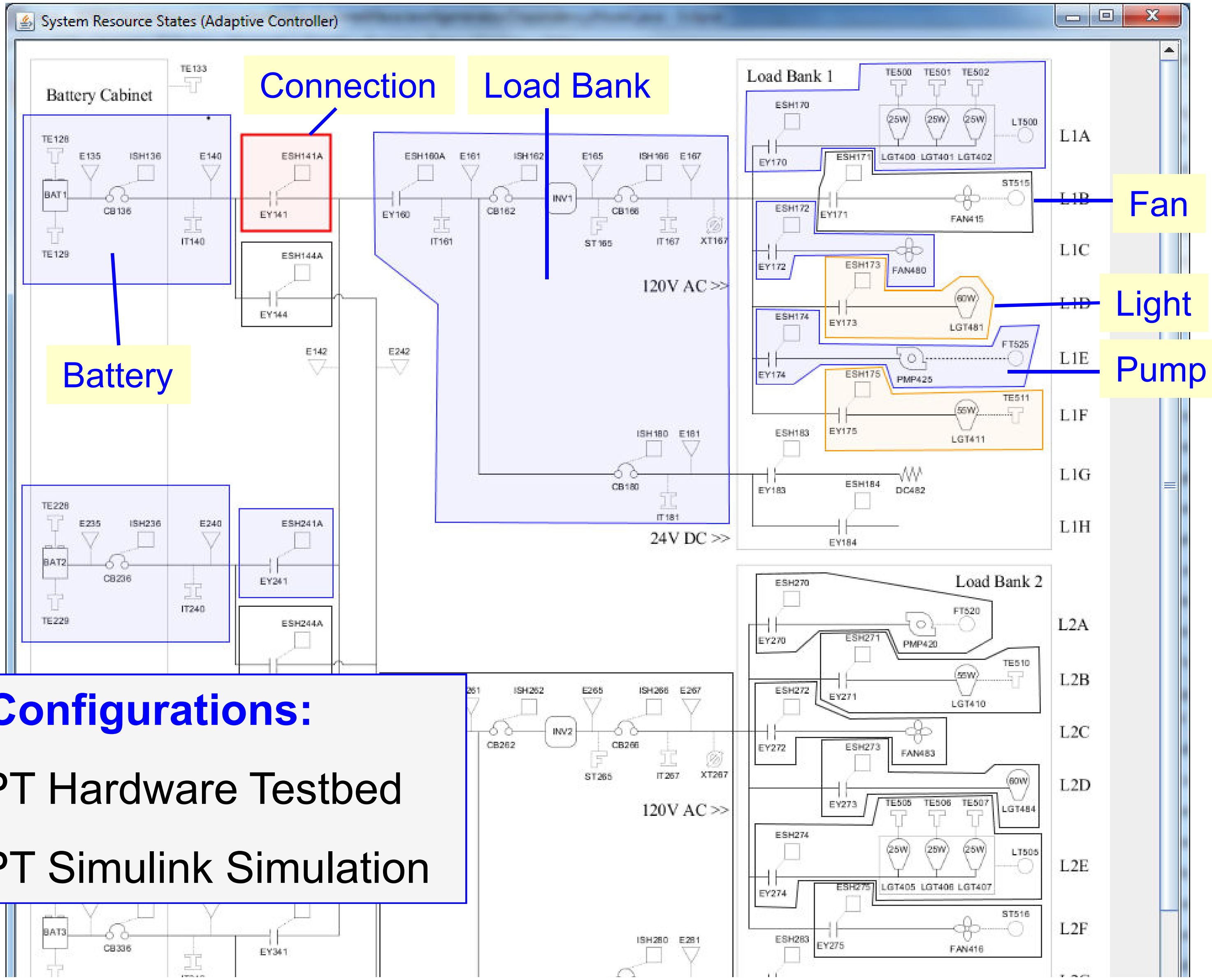
Create a test bed for developing and evaluating autonomous system integration strategies.

Intelliface/ADAPT Testbed



Intelliface/ADAPT

Controlled System - ADAPT

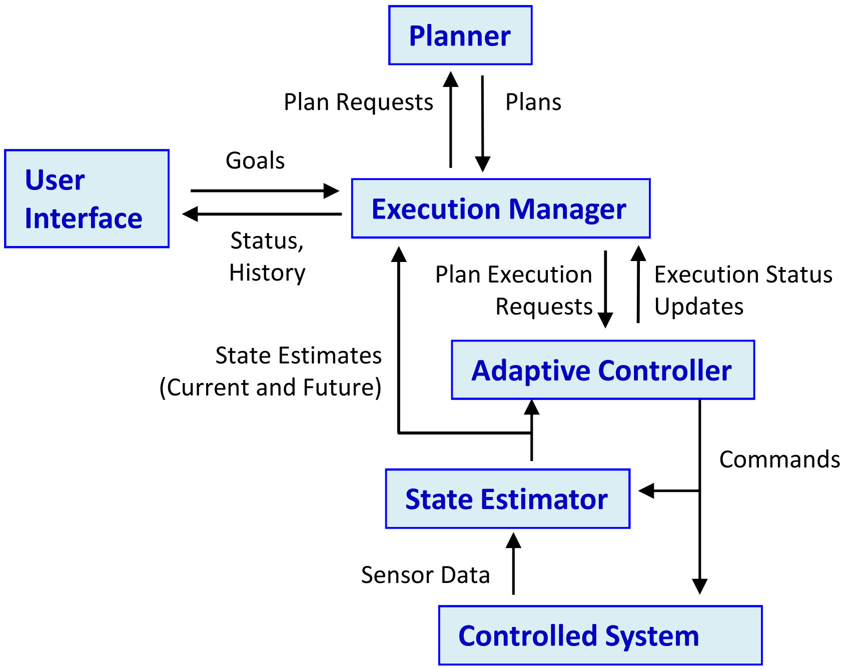


Two Configurations:

ADAPT Hardware Testbed

ADAPT Simulink Simulation

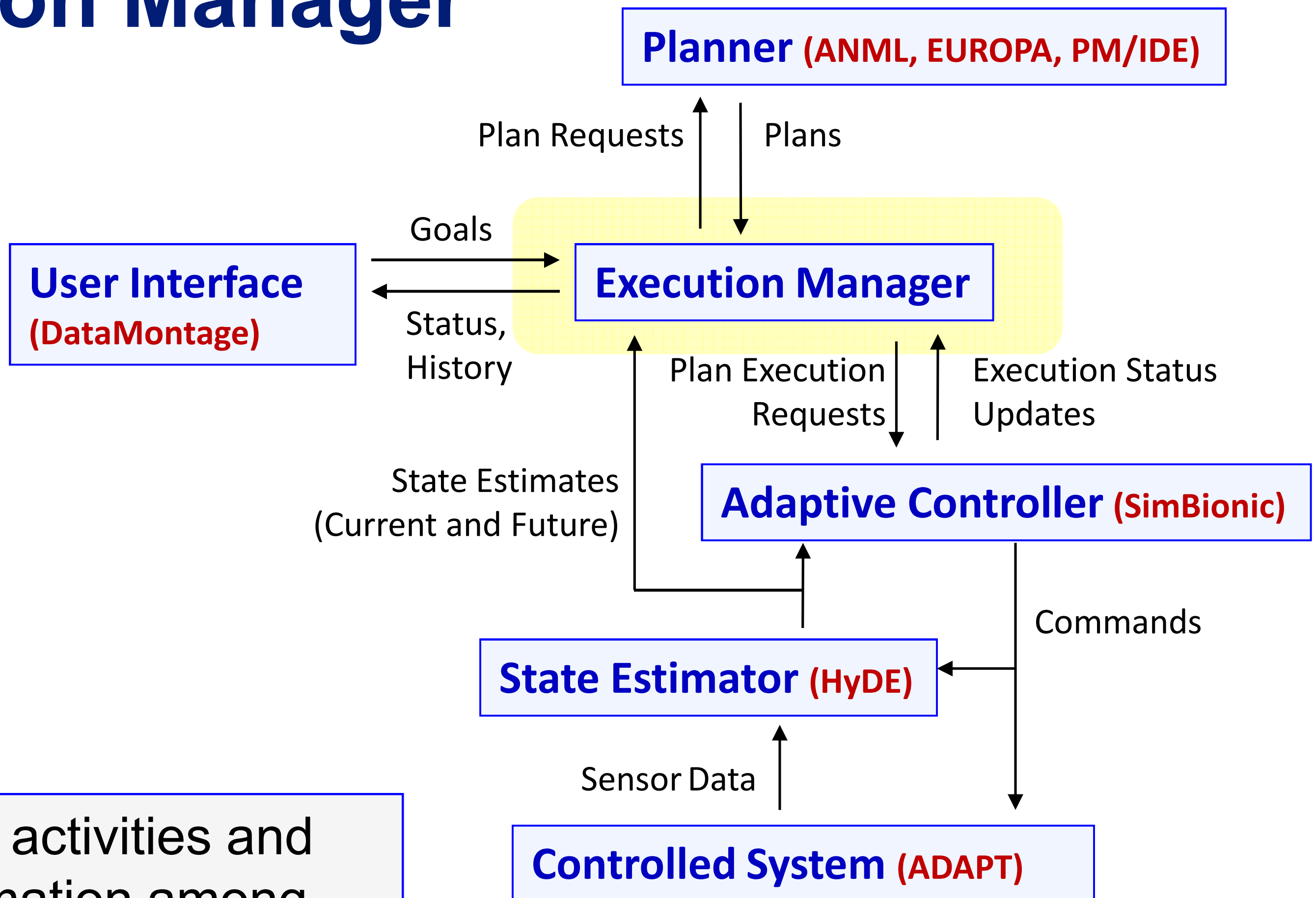
Basic Scenario: Replanning after a Fault



User	enters tasks to perform
Execution Manager	requests plan
Planner	generates plan
Execution Manager	sends plan to Adaptive Controller for execution
Adaptive Controller	executes each plan action by sending commands.
Controlled System	outputs sensor data.
User Interface	displays sensor data and plan in graphs and timelines.
User	injects fault
State Estimator	detects, diagnoses fault.
Execution Manager	evaluates fault's impact on the plan. Requests new plan.
Planner	generates new plan
Execution Manager	sends new plan to Adaptive Controller for execution
User Interface	shows new plan and old plan

Intelliface/ADAPT

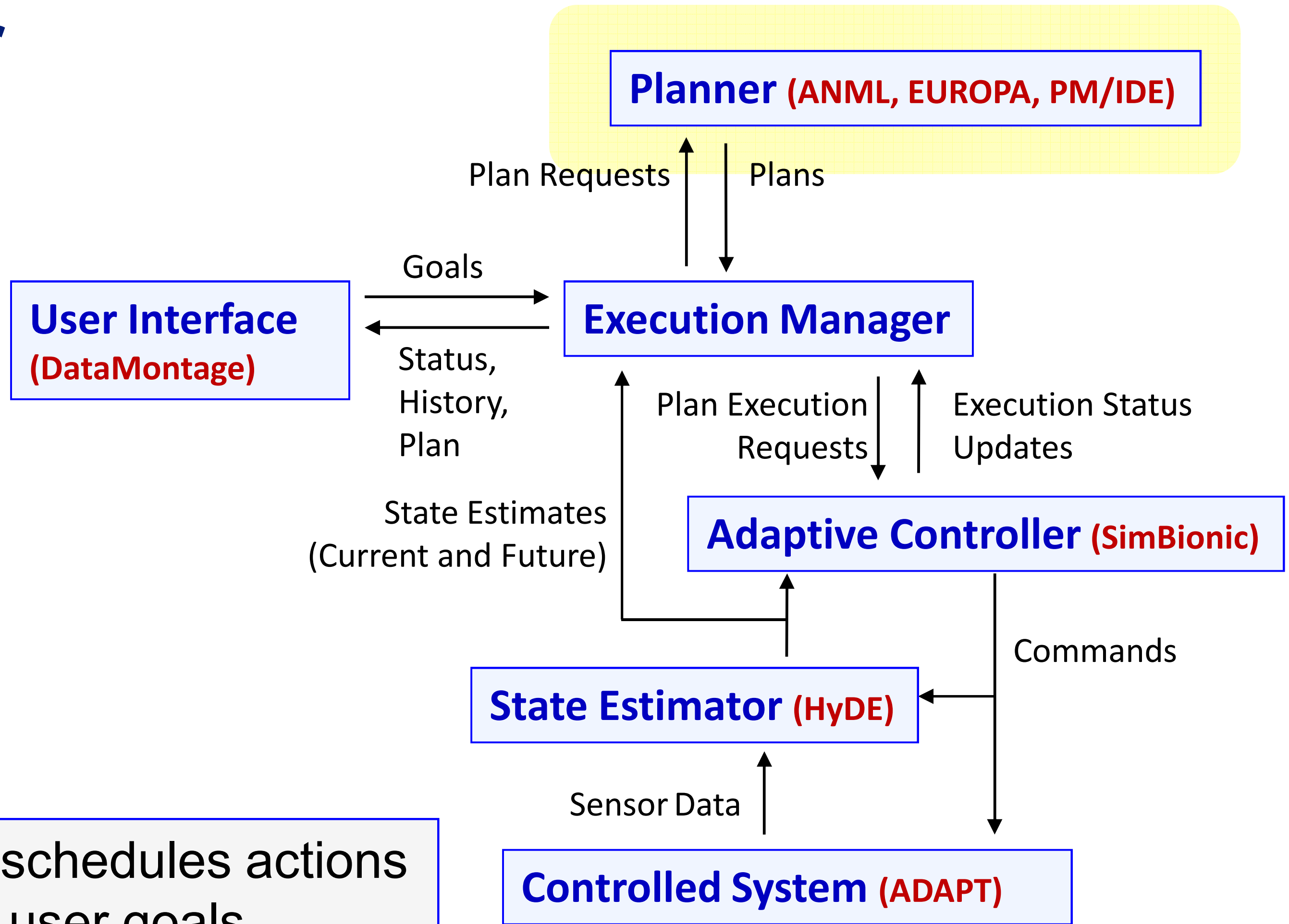
Execution Manager



Coordinates activities and routes information among other modules.

Intelliface/ADAPT

Planner



Selects and schedules actions that achieve user goals

ANML

Action Notation Modeling Language

Language Origins	Developed by NASA Inspired by PDDL, NDDL, AML Expressive, high-level, represent time
Fluents	Represent the state of world as discrete or continuous time-varying variables and functions
Actions	Specify effects on world by assigning values to fluents Specify conditions that must be satisfied Optional decomposition into subactions Have quantitative duration

Example ANML

Fluent Declaration

```
fluent Location atLocation(Robot r);
```

Action Definition

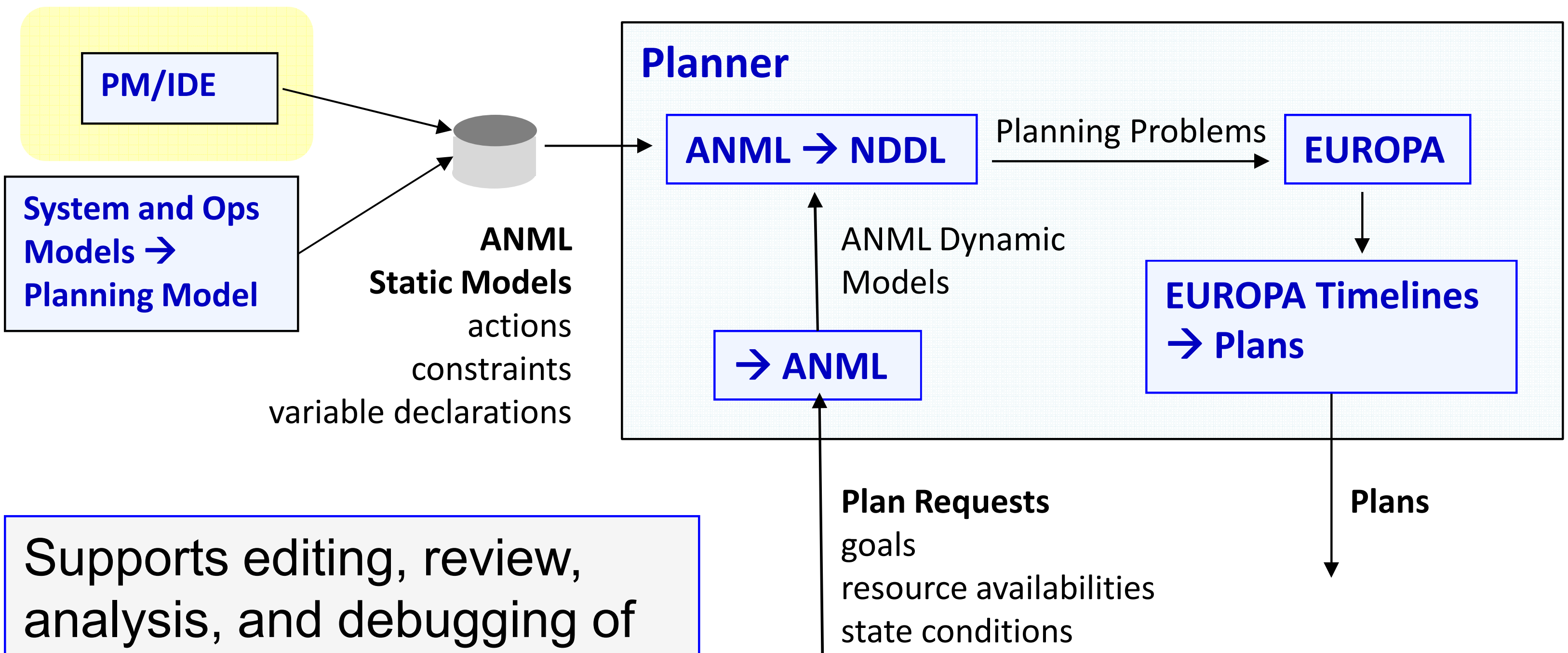
```
action recharge(Robot robot) {
  [start] batteryLevel(robot) > 0;
  [end] batteryLevel(robot) := 100;
}
```

Hierarchical Actions

```
action takeImage(
        Robot robot, Location location) {
  duration := 10;
  [all] contains
    ordered(
      calibrateCamera(robot),
      getImage(robot, location));
}
```

Planner

Planning Model Integrated Development Environment (PM/IDE)

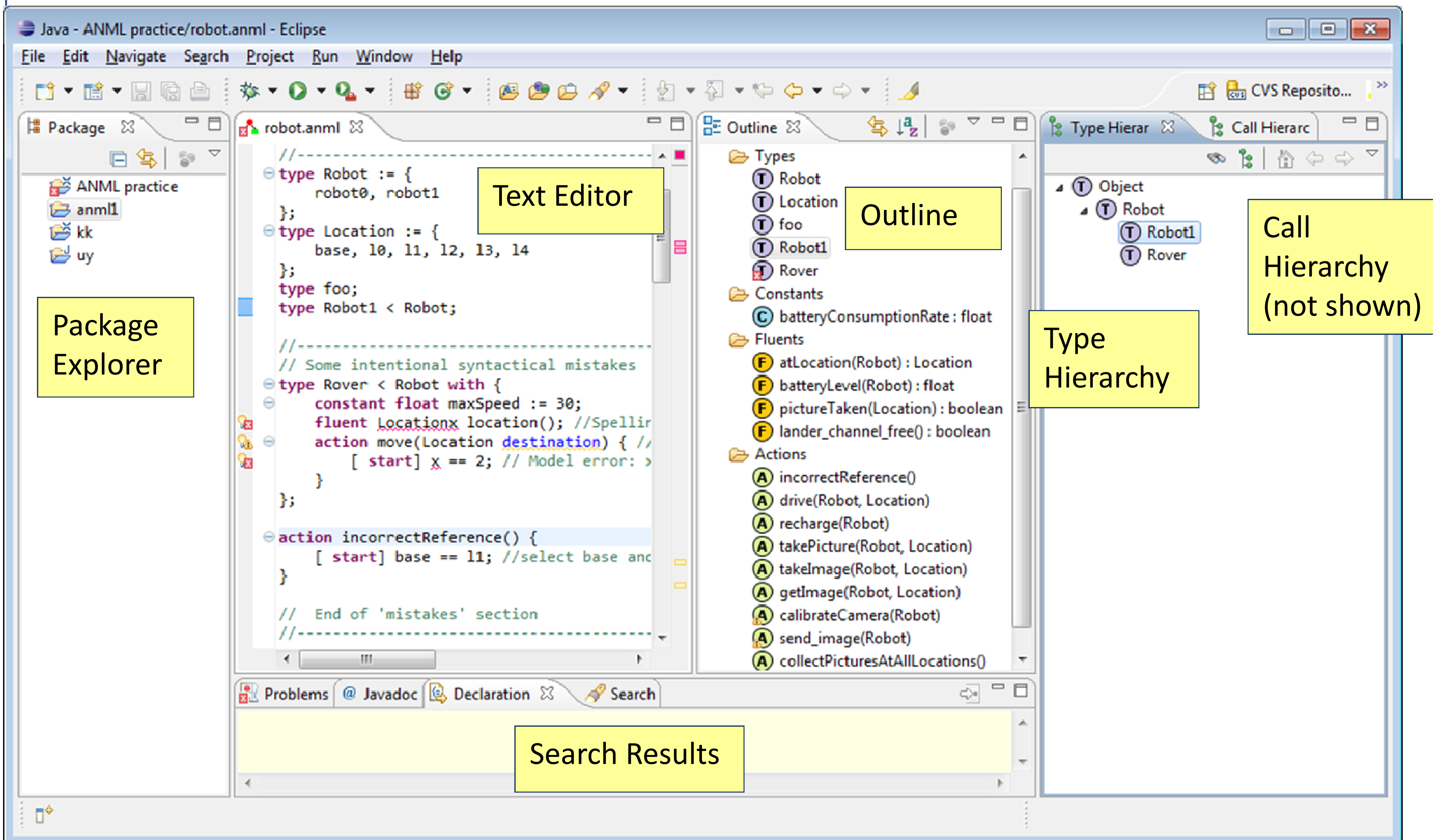


Supports editing, review, analysis, and debugging of ANML models

Provides syntax-aware text editor and visualizations

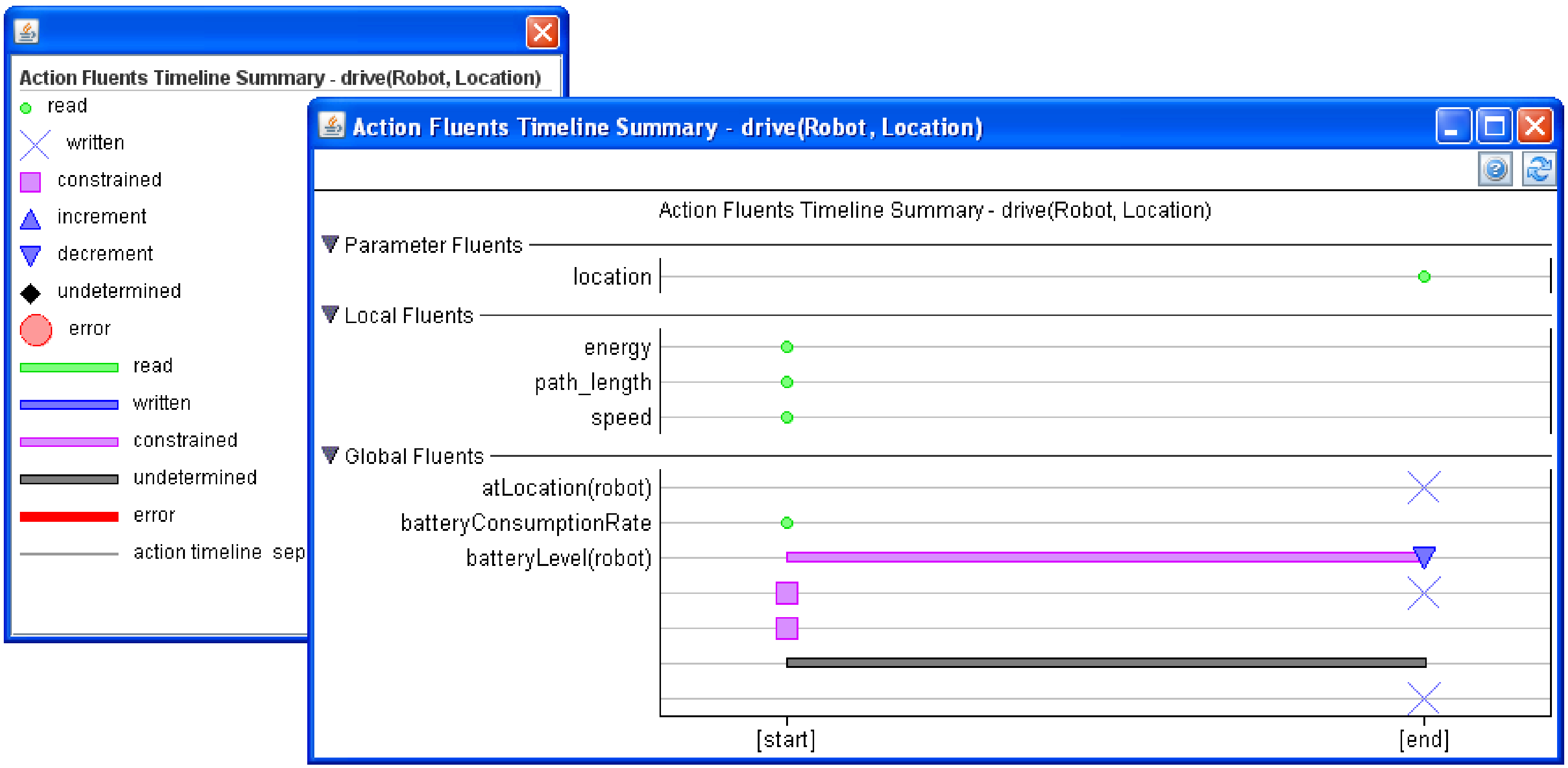
PM/IDE

Syntax-Aware Text Editor and Views



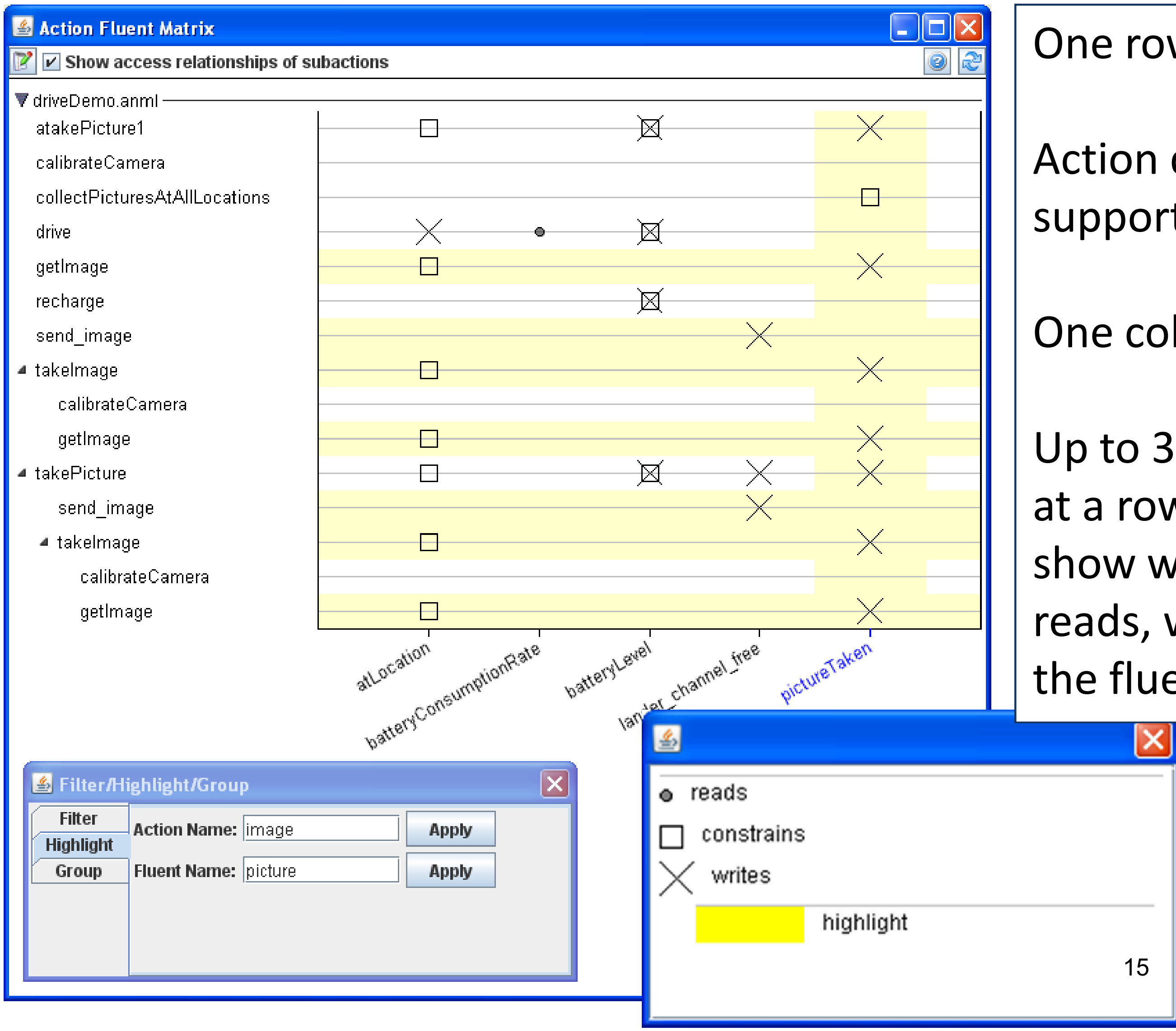
Action Fluents Timeline Summary

shows when a user-selected action reads, writes, or constrains the action's parameters and local and global fluents.



PM/IDE

Action Fluents Matrix



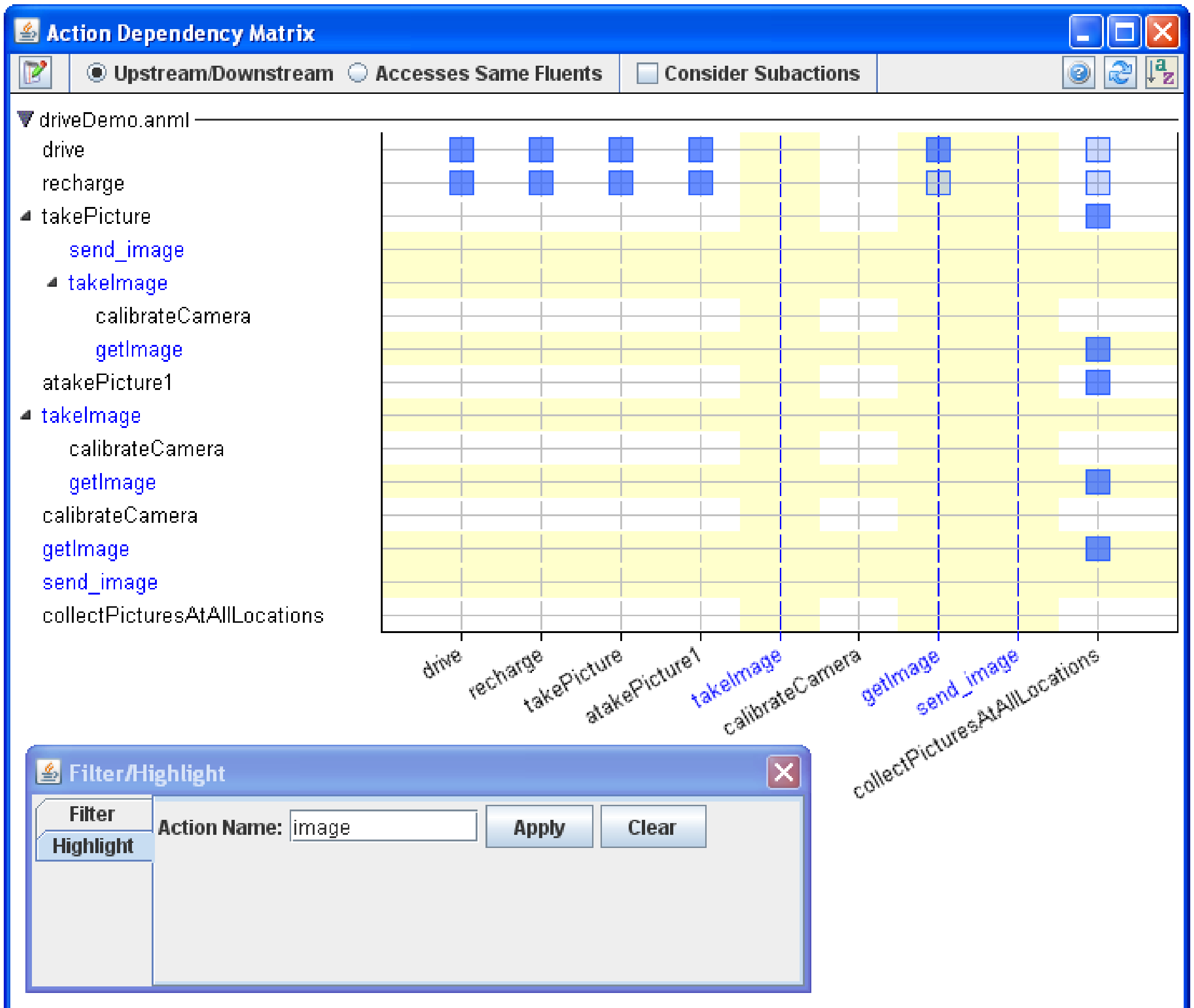
One row per action.

Action decomposition supported.

One column per fluent.

Up to 3 overlapping symbols at a row-column position show whether an action reads, writes, or constrains the fluent.

Action Dependency Matrix



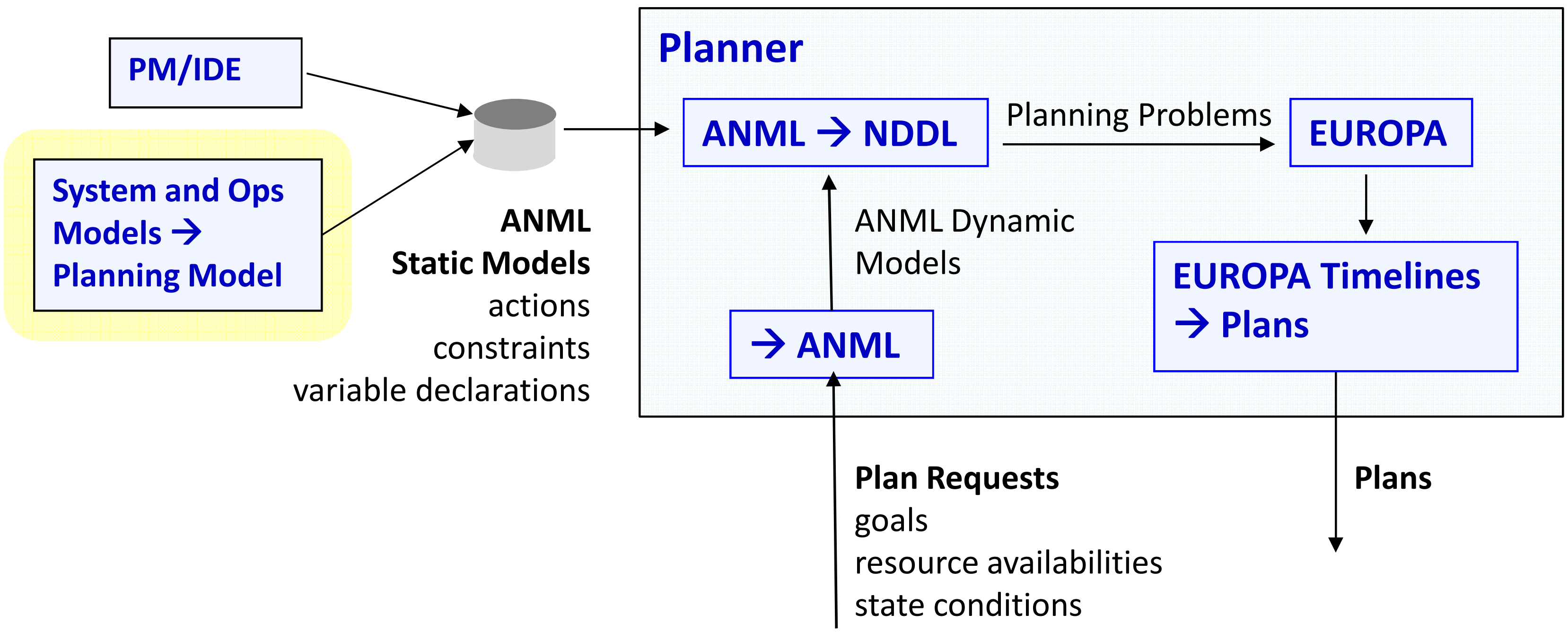
Shows pairs of actions that are:

1. Upstream / Downstream

or

2. Access Same Fluents

Planner
System and Operations Models → Planning Domain Model



Generates ANML from:

system models (subsystems, components)

operations models (flight rules)

ANML Code Generation Example:

Support Functions

Dependency Type table:

ID	DEPENDENT_TYPE	SUPPORTING_TYPE	DEPENDENCY_TYPE
dependency_load_loadbank	Load	Loadbank	shared
dependency_loadbank_connection	Loadbank	Connection	shared
dependency_connection_battery	Connection	Battery	exclusive

…combined with dependency table:

RESOURCE	SUPPORTING_SET	GROUPID
r_LGT400	set_lb1	dependency_load_loadbank
r_FAN415	set_lb1	dependency_load_loadbank
r_FAN480	set_lb1	dependency_load_loadbank

…

…generates Support Functions:

```
function boolean loadbankCanSupportLoad(Load l, Loadbank lk);
[all] {
   loadbankCanSupportLoad(r_LGT400,r_LB1) := true;
   loadbankCanSupportLoad(r_LGT400,r_LB2) := false;
   loadbankCanSupportLoad(r_FAN415,r_LB1) := true;
   // etc.
}
```

ANML Code Generation Example:

Actions

```
action useLoad(Load l) {

   duration >= 3;
   [all] loadState(l) == operational;

   [all] exists (Loadbank lb) {
      loadbankUsage(lb) == in_use and
      LoadbankCanSupportLoad(l,lb) == true;
   }

   [all] loadUsage(l) := in_use;
}
```

Action skeleton (always generated)

Extra preconditions inferred from dependency types

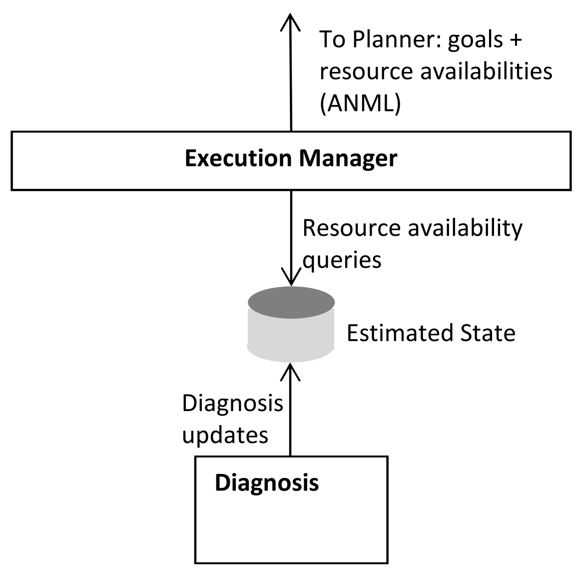
Dependency Type table:

ID	DEPENDENT_TYPE	SUPPORTING_TYPE	DEPENDENCY_TYPE
dependency_load_loadbank	Load	Loadbank	shared
dependency_loadbank_connection	Loadbank	Connection	shared
dependency_connection_battery	Connection	Battery	exclusive

Resource Availabilities

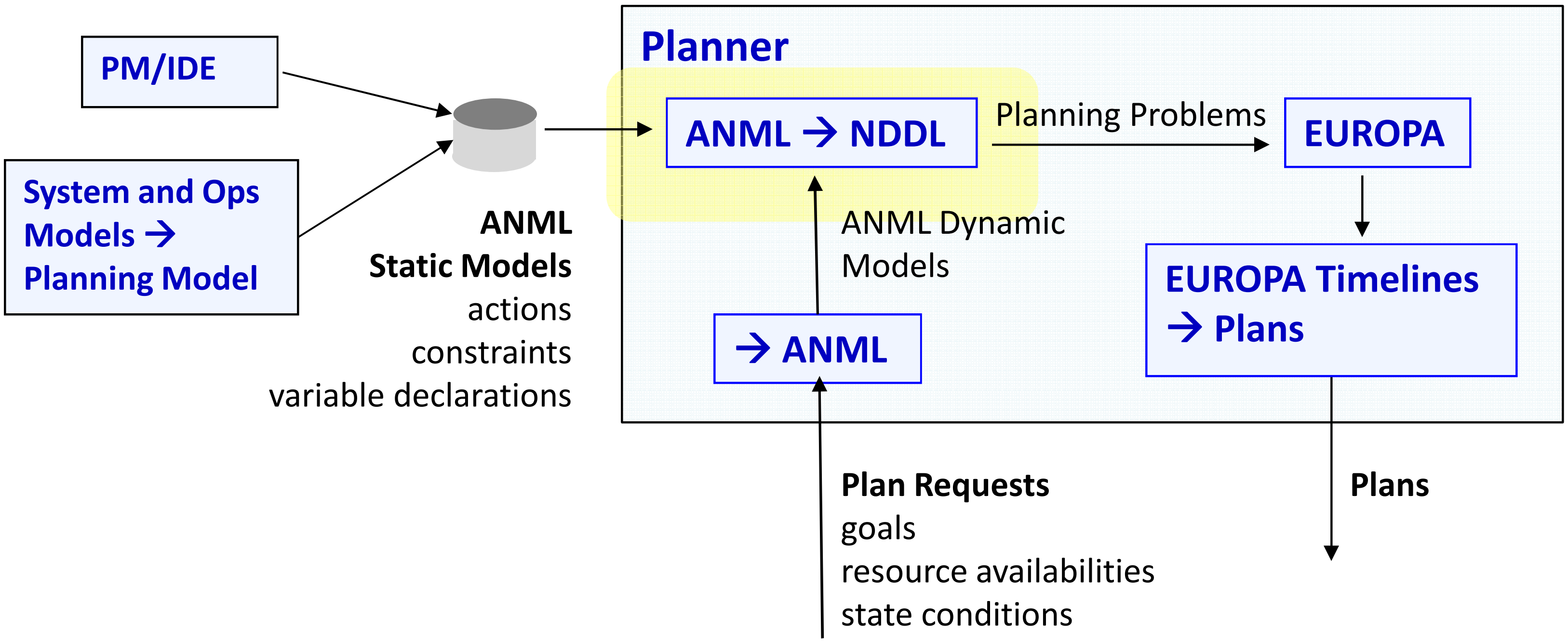
- ANML code is generated on the fly by the Execution Manager, to reflect real-time resource availabilities.
- Each resource availability maps to a simple ANML statement, for example:

r_LGT400 is operational -> `[all] loadState(r_LGT400) := operational;`



Planner

ANML → NDDL Translator



Partial prototype translates ANML model to NDDL

ANML Fluents →NDDL Timelines

For each ANML fluent declaration, create a NDDL timeline class whose properties correspond to the fluent parameters. The class' "value" token represents the fluent's value during a period of time.

```
fluent float [10.0,100.0] batteryLevel(Robot r);
```

```
class batteryLevel extends ANML_Fluent { //ANML_Fluent extends
Timeline
   Robot r;
   batteryLevel (Robot _r) {
      super();
      r = _r;
   }
   predicate value{float _value;};
   predicate value_undetermined {};
}
batteryLevel _batteryLevel_robot1_ = new batteryLevel(robot1);
batteryLevel _batteryLevel_robot2_ = new batteryLevel(robot2);
```

ANML → NDDL Translator

ANML Actions → NDDL Timelines

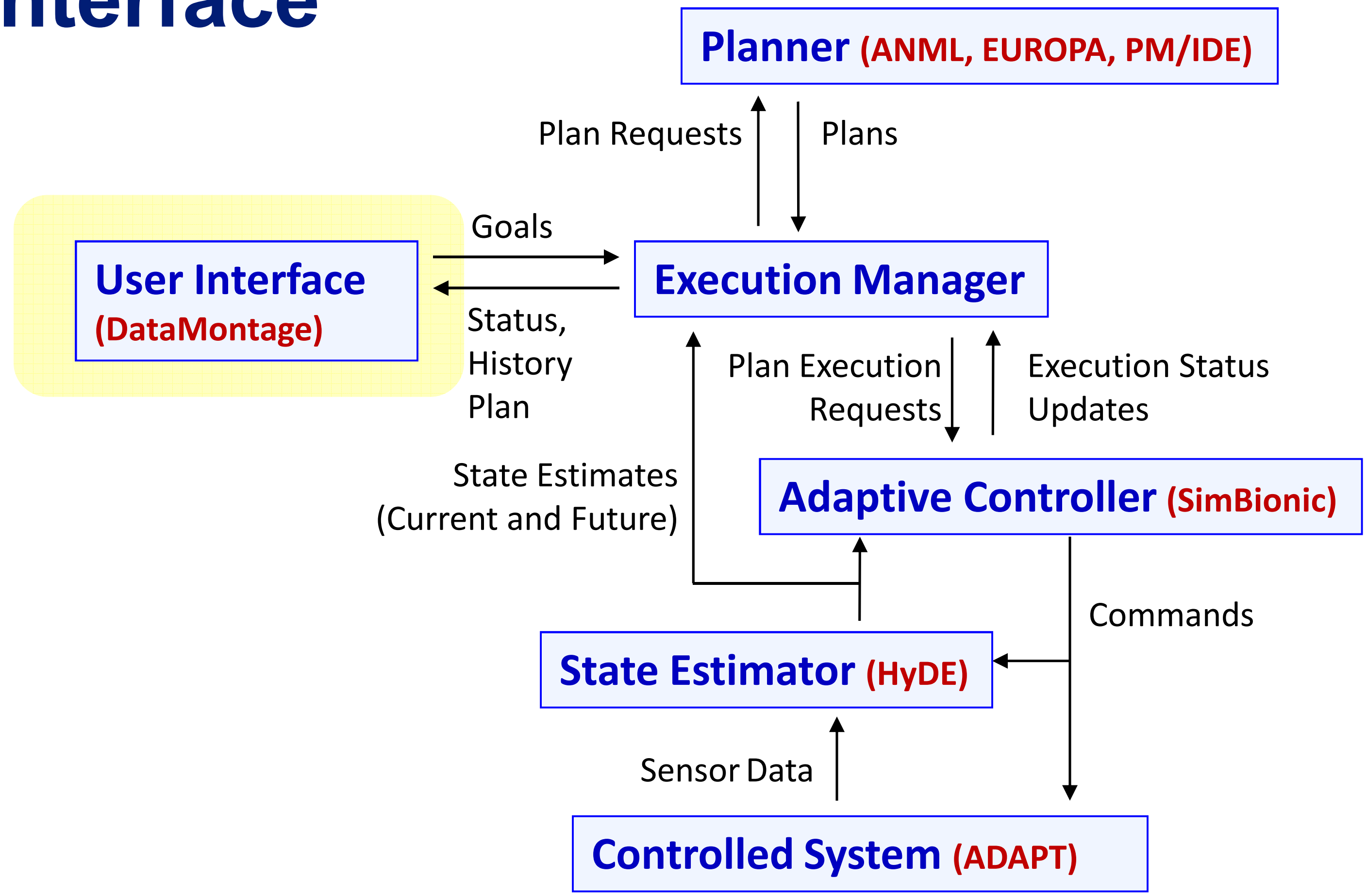
For each ANML action definition, create a NDDL timeline class whose properties correspond to the action parameters. An action has a token "exe" representing when the action is executed.

```
action drive (Robot robot, Location location) { … }
```

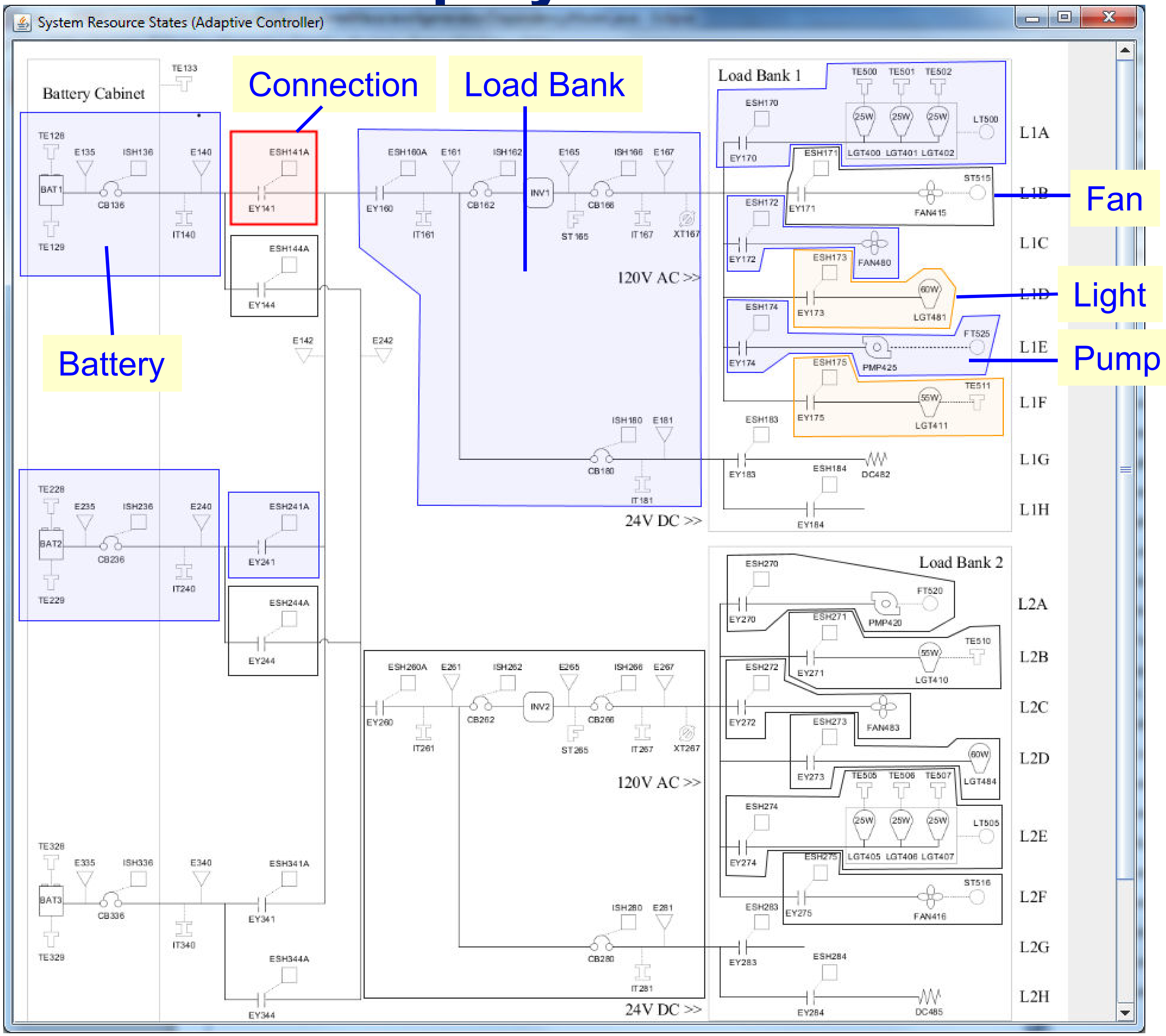
```
class drive extends ANML_Action { //ANML
   Robot robot;
   Location location;
   drive (Robot _robot, Location _location) {
      super();
      robot = _robot;
      location = _location;
   }
}
//grounding
drive _drive_robot1_base_ = new drive(robot1,base);
drive _drive_robot1_l1_ = new drive(robot1,l1);
drive _drive_robot2_base_ = new drive(robot2,base);
drive _drive_robot2_l1_ = new drive(robot2,l1);
```

Intelliface/ADAPT

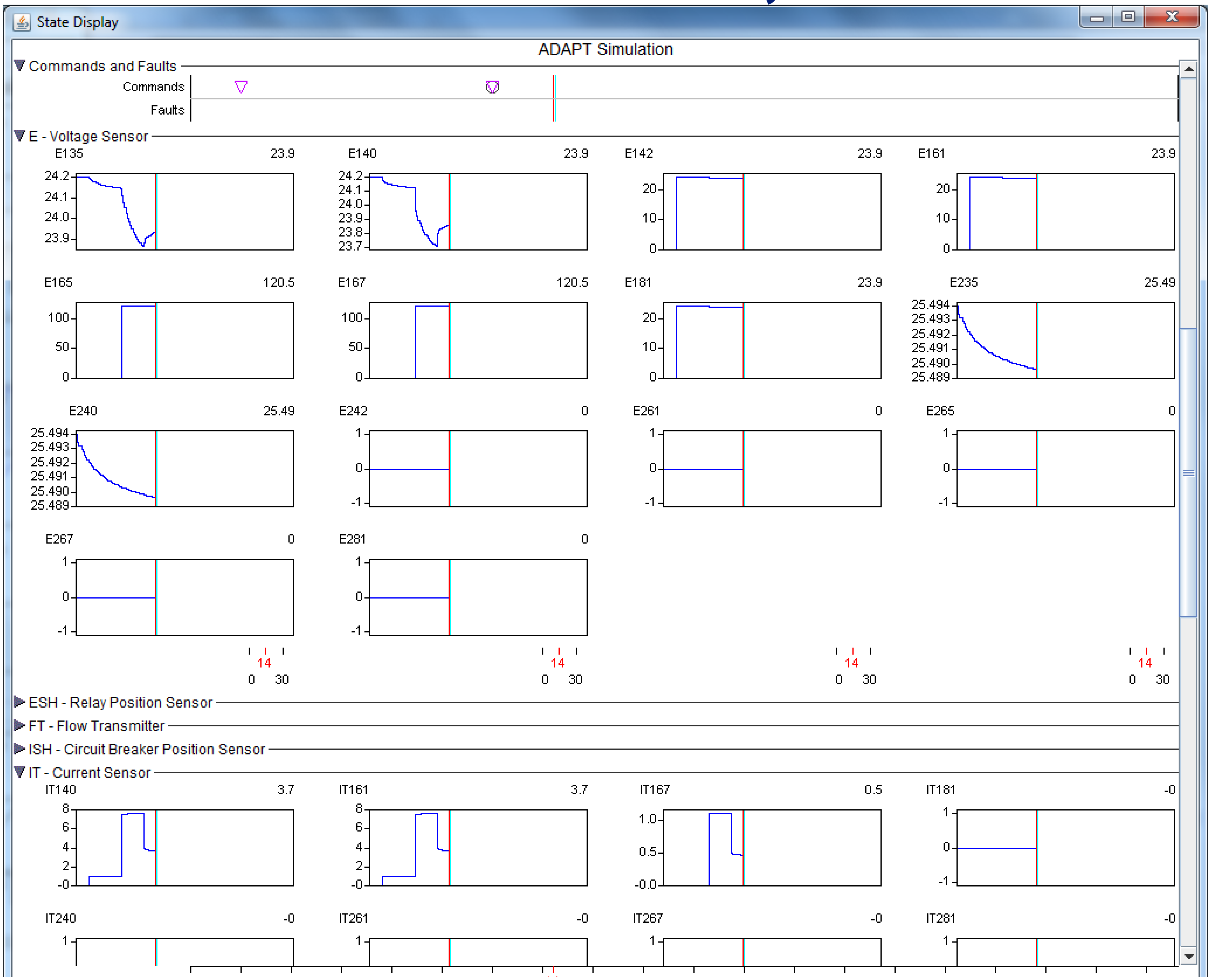
User Interface



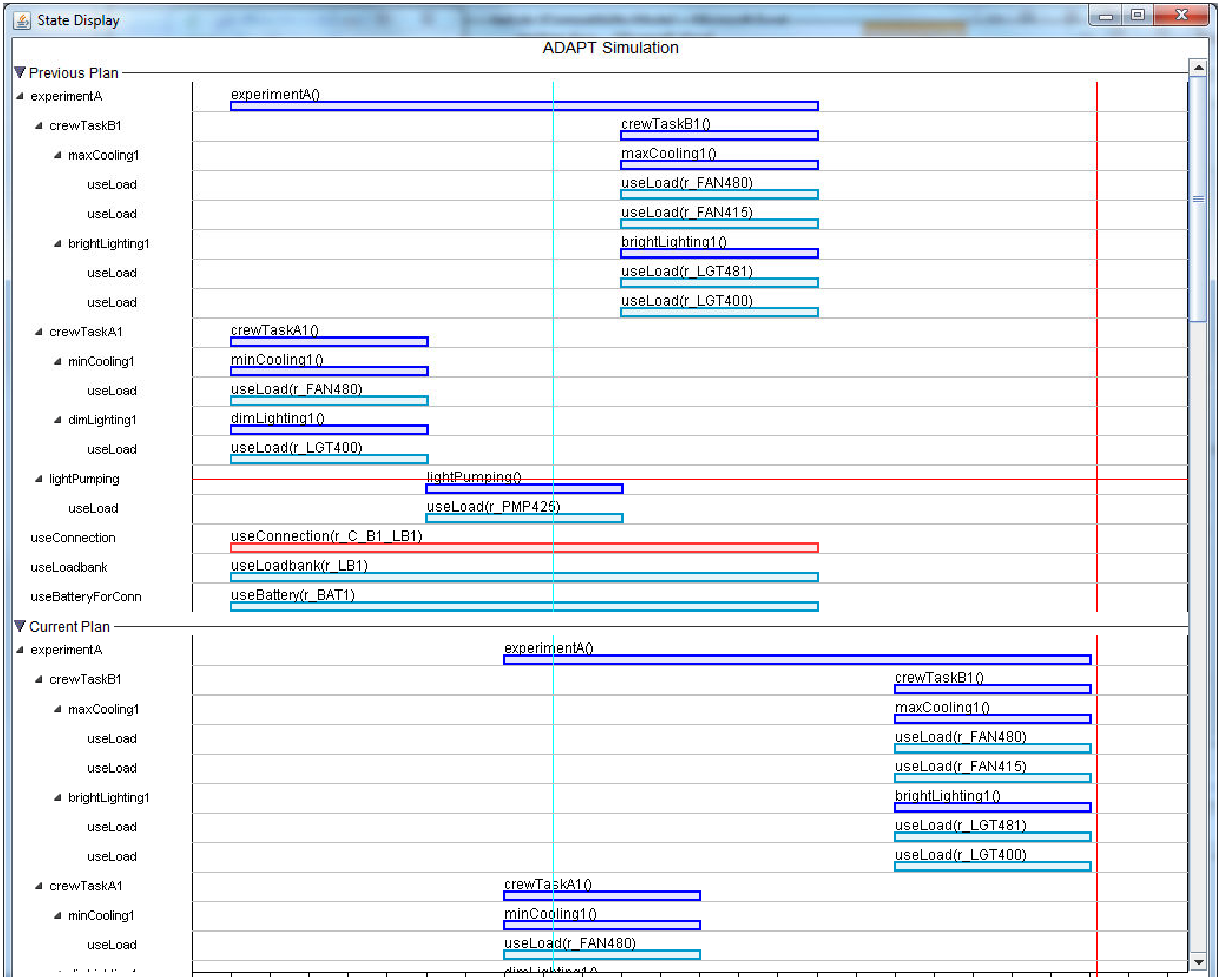
Schematic Display



Timeline: Commands, Sensor Data

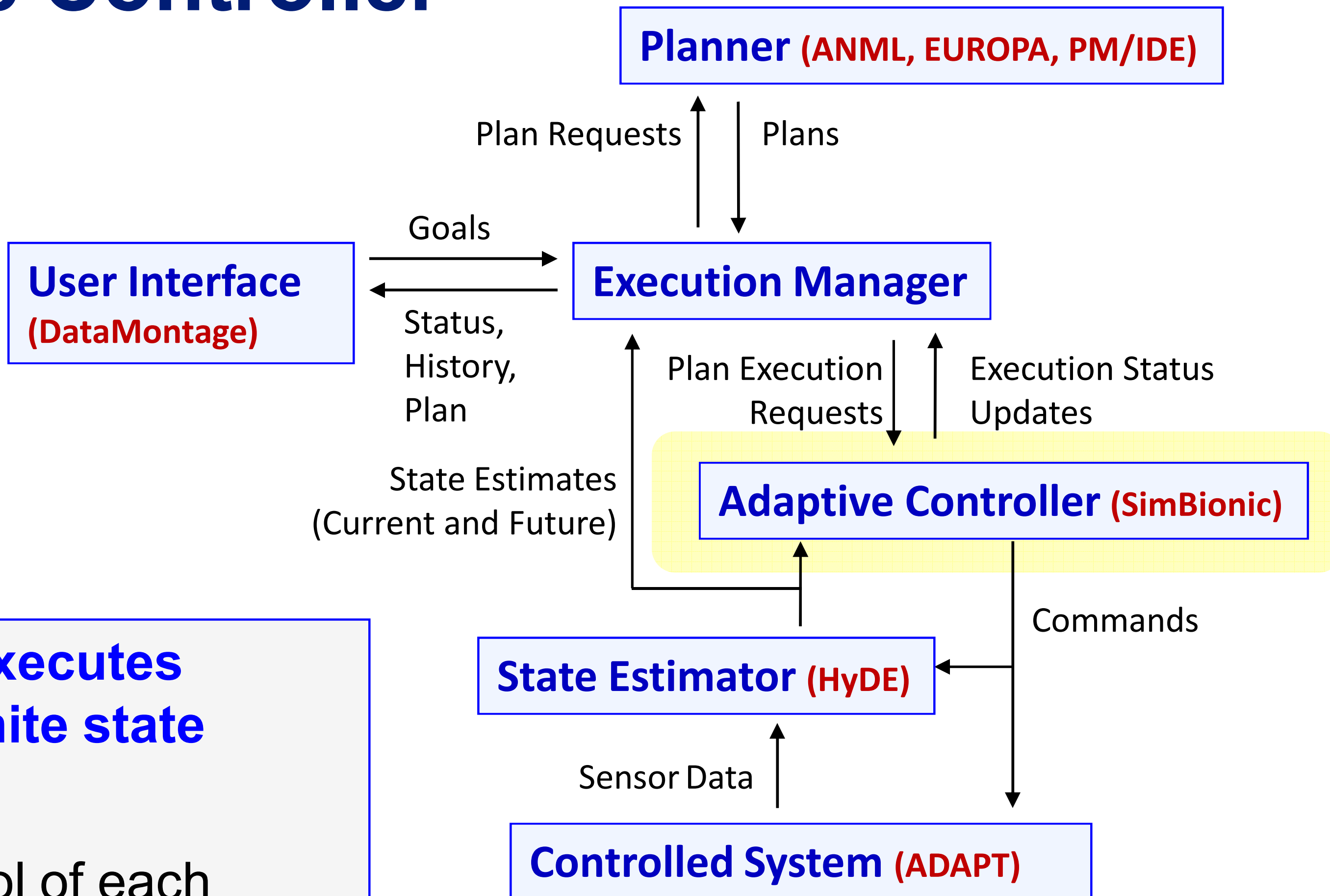


Timeline: Previous and New Plans



Intelliface/ADAPT

Adaptive Controller

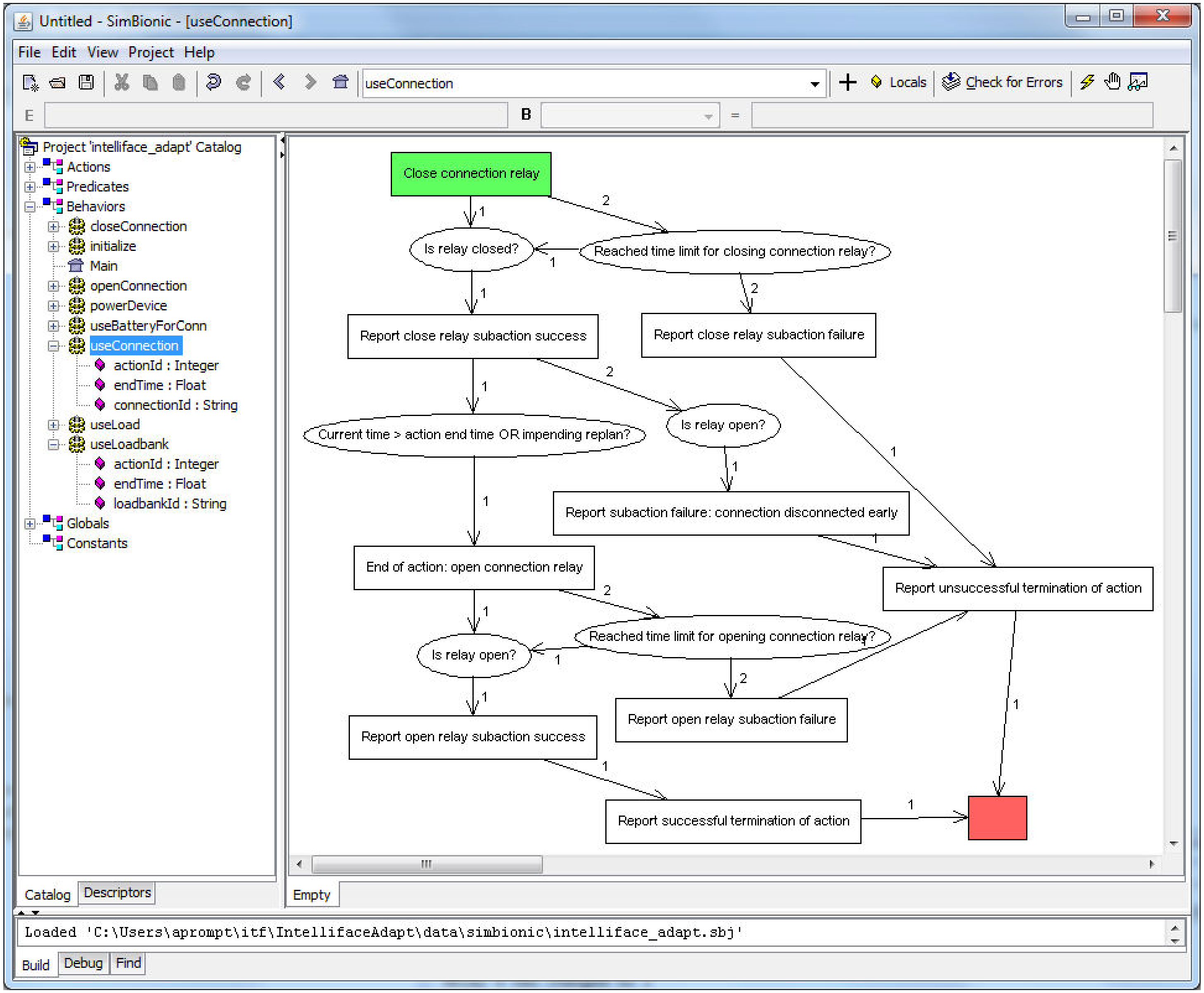


SimBionic® executes augmented finite state machines

Adaptive control of each action controlled by 1+ FSMs

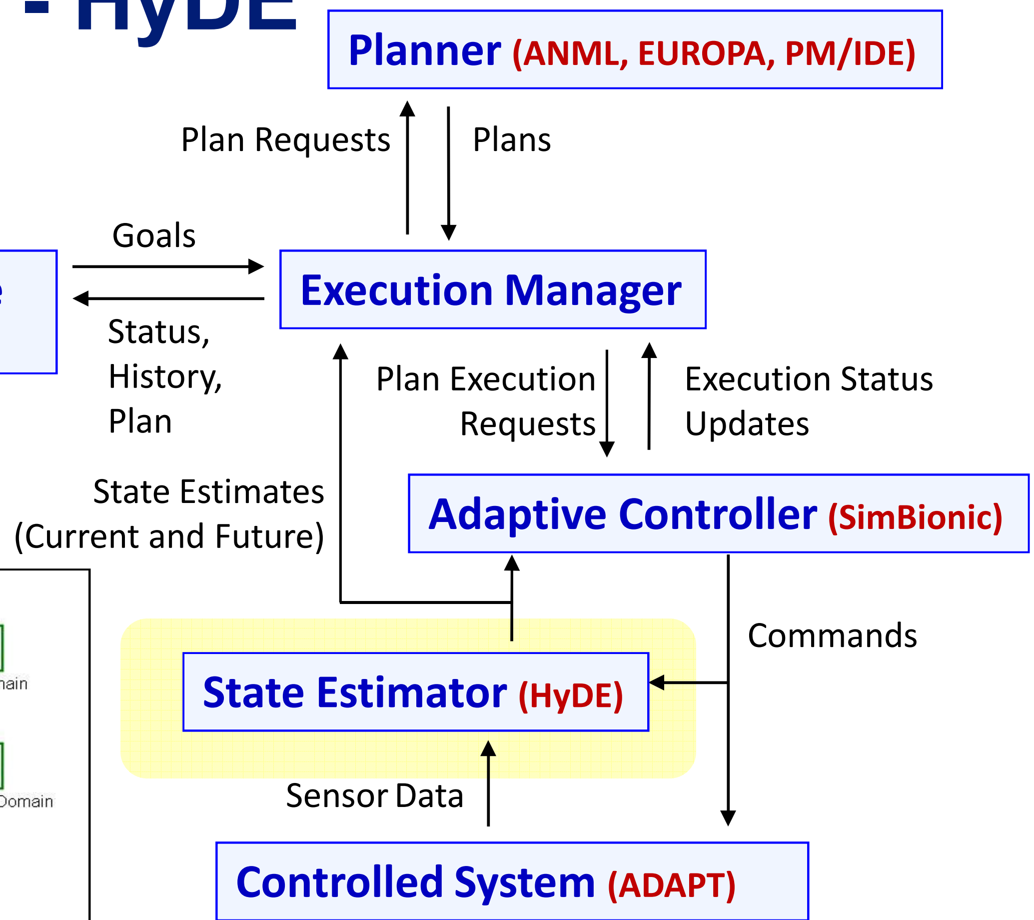
Multiple FSMs can run in parallel, branch, loop, wait

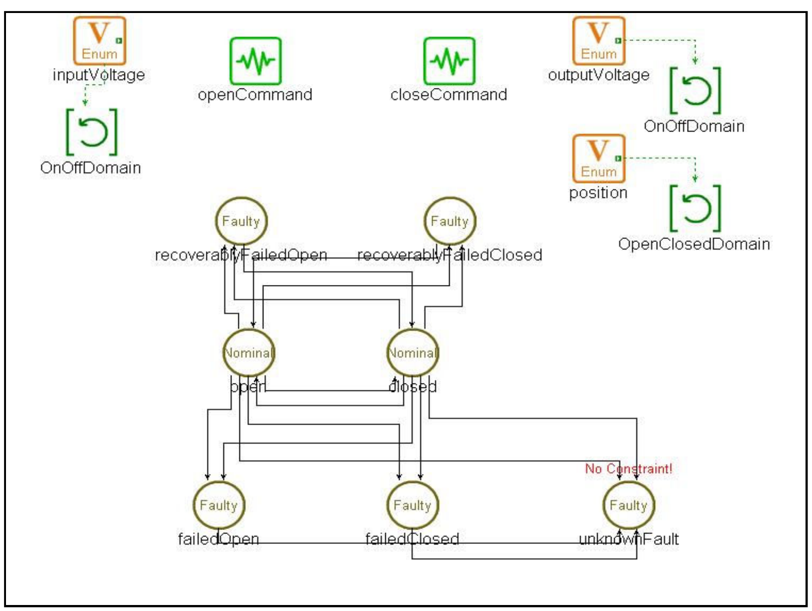
Adaptive Control Editor (SimBionic)



Intelliface/ADAPT

State Estimator - HyDE





Some Problems Not Yet Addressed

Preserve Work Already Performed

Minimize Changes to the Plan

Complex Mappings from Faults to Resource Changes

Ambiguous Diagnoses

Handle Resource Quantity and Quality

Diagnostic Actions

Summary

Robust autonomy requires integrating State Estimation, Planning, and Execution.

Integration imposes additional requirements on models.

Model translation and integration is labor-intensive and error-prone

We developed an initial version of a testbed for developing and evaluating diverse integration strategies

Generation of models and interfaces can be partially automated.