Robust Autonomy Requires Integration

Planning

State Estimation

Adaptive Control

Controlled System
Modeling and Integration Challenges

Complex and Interdependent Resources

Power Generation → Power Storage → Power Distribution → Command & Data Handling → Pump

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

**Planner** *(ANML, EUROPA, PM/IDE)*

- Plan Requests
- Plans

**Execution Manager**

- Goals
- Status, History
- Plan Execution Requests
- Execution Status Updates

**User Interface** *(DataMontage)*

- Status, History

**Adaptive Controller** *(SimBionic)*

- Commands

**State Estimator** *(HyDE)*

- Sensor Data

**Controlled System** *(ADAPT)*

- State Estimates (Current and Future)
Intelliface/ADAPT
Controlled System - ADAPT

Two Configurations:
- ADAPT Hardware Testbed
- ADAPT Simulink Simulation
Basic Scenario: Replanning after a Fault

User Interface 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**

- **Planner** *(ANML, EUROPA, PM/IDE)*
  - Plan Requests → Plans
  - Goals
  - Status, History
  - State Estimates *(Current and Future)*

- **Execution Manager**
  - Plan Execution Requests
  - Execution Status Updates

- **Adaptive Controller** *(SimBionic)*
  - Commands

- **State Estimator** *(HyDE)*
  - Sensor Data

- **User Interface** *(DataMontage)*
  - Goals
  - Status, History

**Controlled System** *(ADAPT)*

Coordinates activities and routes information among other modules.
Intelliface/ADAPT

Planner

User Interface (DataMontage)

Execution Manager

Adaptive Controller (SimBionic)

State Estimator (HyDE)

Controlled System (ADAPT)

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)

PM/IDE
System and Ops Models ➔ Planning Model

ANML Static Models
- actions
- constraints
- variable declarations

ANML ➔ NDDL
Planning Problems ➔ EUROPA

EUROPA Timelines ➔ Plans

ANML Dynamic Models

Plan Requests:
- goals
- resource availabilities
- state conditions

Supports editing, review, analysis, and debugging of ANML models
Provides syntax-aware text editor and visualizations
PM/IDE
Syntax-Aware Text Editor and Views

Package Explorer
Text Editor
Outline
Type Hierarchy
Call Hierarchy (not shown)
Search Results
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.
PM/IDE

Action Dependency Matrix

Shows pairs of actions that are:

1. Upstream / Downstream

or

2. Access Same Fluents
Generates ANML from:
- system models (subsystems, components)
- operations models (flight rules)
ANML Code Generation Example:

Support Functions

Dependency Type table:

<table>
<thead>
<tr>
<th>ID</th>
<th>DEPENDENT TYPE</th>
<th>SUPPORTING TYPE</th>
<th>DEPENDENCY TYPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>dependency_load_loadbank</td>
<td>Load</td>
<td>Loadbank</td>
<td>shared</td>
</tr>
<tr>
<td>dependency_loadbank_connection</td>
<td>Loadbank</td>
<td>Connection</td>
<td>shared</td>
</tr>
<tr>
<td>dependency_connection_battery</td>
<td>Connection</td>
<td>Battery</td>
<td>exclusive</td>
</tr>
</tbody>
</table>

...combined with dependency table:

<table>
<thead>
<tr>
<th>RESOURCE</th>
<th>SUPPORTING_SET</th>
<th>GROUPID</th>
</tr>
</thead>
<tbody>
<tr>
<td>r_LGT400</td>
<td>set_lb1</td>
<td>dependency_load_loadbank</td>
</tr>
<tr>
<td>r_FAN415</td>
<td>set_lb1</td>
<td>dependency_load_loadbank</td>
</tr>
<tr>
<td>r_FAN480</td>
<td>set_lb1</td>
<td>dependency_load_loadbank</td>
</tr>
</tbody>
</table>

...generates Support Functions:

```java
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**

```plaintext
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;
}
```

**Dependency Type table:**

<table>
<thead>
<tr>
<th>ID</th>
<th>DEPENDENT TYPE</th>
<th>SUPPORTING TYPE</th>
<th>DEPENDENCY TYPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>dependency_load_loadbank</td>
<td>Load</td>
<td>Loadbank</td>
<td>shared</td>
</tr>
<tr>
<td>dependency_loadbank_conn</td>
<td>Loadbank</td>
<td>Connection</td>
<td>shared</td>
</tr>
<tr>
<td>dependency_connection_bat</td>
<td>Connection</td>
<td>Battery</td>
<td>exclusive</td>
</tr>
</tbody>
</table>
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:

  \[ r_{\text{LGT400}} \text{ is operational } \Rightarrow [\text{all}] \text{ loadState}(r_{\text{LGT400}}) := \text{operational}; \]
Partial prototype translates ANML model to NDDL
ANML → NDDL Translator

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.

```plaintext
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.

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

```java
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

- **Planner (ANML, EUROPA, PM/IDE)**
  - Plan Requests → Plans
  - Goals

- **Execution Manager**
  - Plan Execution Requests → Execution Status Updates
  - Status, History Plan

- **Adaptive Controller (SimBionic)**
  - Execution Status Updates → Commands

- **State Estimator (HyDE)**
  - Sensor Data

- **Controlled System (ADAPT)**

- **User Interface (DataMontage)**
  - Goals → Status, History Plan
  - State Estimates (Current and Future)
Schematic Display

- **Battery**
- **Connection**
- **Load Bank**
- **Fan**
- **Light**
- **Pump**
Timeline: Commands, Sensor Data
Timeline: Previous and New Plans
Intelliface/ADAPT

Adaptive Controller

Plan Requests → Plans
Goals
Status, History, Plan
State Estimates (Current and Future)
Plan Execution Requests
Execution Status Updates

User Interface (DataMontage)
Execution Manager
Adaptive Controller (SimBionic)
State Estimator (HyDE)
Controlled System (ADAPT)

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

---

**Planner (ANML, EUROPA, PM/IDE)**
- Plan Requests → Plans
- Goals
- Plan Execution Requests
- Execution Status Updates

**Execution Manager**
- Status, History, Plan
- State Estimates
- Current and Future

**User Interface (DataMontage)**
- Goals

**Adaptive Controller (SimBionic)**
- Commands
- Sensor Data

**Controlled System (ADAPT)**
- State Estimator (HyDE)
- User Interface (DataMontage)
- Planner (ANML, EUROPA, PM/IDE)
- Execution Manager
- Adaptive Controller (SimBionic)
- User Interface (DataMontage)
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.