The Monterey Phoenix Approach and Tool for Behavior Modeling: A Tutorial

Kristin Giammarco, Ph.D.
Department of Systems Engineering

Mikhail Auguston, Ph.D.
Department of Computer Science
Tutorial Objectives

Upon successful completion of this tutorial, you will be able to:

1. Explain the motivation for Monterey Phoenix (MP) modeling
2. Identify the basic concepts of and appropriate use cases for MP
3. Extract information needed for a behavior model from a provided mission narrative
4. Develop, run, and inspect an MP model using the MP-Firebird tool
MP Tutorial Part 1: Motivation

Kristin Giammarco, Ph.D.
Department of Systems Engineering

Mikhail Auguston, Ph.D.
Department of Computer Science
What is Monterey Phoenix?

• MP is a Navy-developed formal approach and language for modeling human, technology, and environment behaviors all in one modeling environment

  – The MP-Firebird tool is publicly available at http://firebird.nps.edu
Purpose of MP Modeling

• To answer questions about behavior
  – structure of behavior
  – dependencies between actions involved in the behavior
  – constraints on behaviors
  – simple queries about behavior
  – to provide a source for different visualizations or views of behaviors

“Simplicity does not precede complexity, but follows it.”
– Alan J. Perlis

MP Tool High Level Architecture

Web browser

Internet

Public user

firebird.nps.edu

MP Modeler

Create a model

Run the model

Inspect scenarios

MP Tool

Compile MP code

Generate all possible scenarios in scope

Display generated scenarios
How do we currently describe system behavior?

Some have not yet transitioned to model-based methods and are still using office automation software, e.g.:
- PowerPoint
- Excel
- Word
Why do we need MP?

<table>
<thead>
<tr>
<th>Prevailing problems</th>
<th>MP value proposition</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Incompleteness</strong>: Only a very small subset of possible behaviors are modeled showing all actors and interactions on the same diagram</td>
<td><strong>Scope-completeness</strong>: Generates set of possible event traces (use case extensions) exhaustively up to a user-defined limit on iterations</td>
</tr>
<tr>
<td><strong>Ambiguity</strong>: Behavior models describe general activities but are unclear about who is doing each activity, or are otherwise unclear about activities performed</td>
<td><strong>Separation of concerns</strong>: Behaviors are separated by actor, and interactions between events in different actors are separately layered on as constraints; modeling in MP enables discussion and clarification of the behavior logic</td>
</tr>
<tr>
<td><strong>Inefficiency</strong>: People continue to do work that an automated computing device could do faster and with far fewer errors</td>
<td><strong>Efficient task allocation</strong>: Humans focus on using their experience, creativity, and pattern detection skills to inspect and evaluate, and use automated tools to compute, generate, and search</td>
</tr>
<tr>
<td><strong>Unwanted behaviors</strong>: Built systems that may meet requirements, but also permit extra undesired behaviors</td>
<td><strong>Behavior pruning</strong>: Enforces the necessary model structure for exposing and purging unwanted behaviors in the design before they emerge in actual system</td>
</tr>
</tbody>
</table>
Addressing Incompleteness

CSI Cargo Screening Process

Efficient Task Allocation

Human
- error prone
- lived experience
- imagination
- creativity
- brain
- inspection
- evaluation
- pattern detection

Machine
- can't do what it's not programmed to do
- searches on large data sets
- computation
- automation
- generation
- view projection
The system shouldn’t continuously accept an unlimited number of access attempts.

Requirement: If a User’s credentials are invalid more than three times, the System shall lock the User’s account.
Example Emergent Behaviors Found Using MP Modeling

• **An order processing system** enters a waiting state after a transaction is cancelled. (Pilcher 2015)

• The **International Space Station** is unaware of a hazardous condition within a supply spacecraft as that spacecraft approaches to dock. (Nelson 2015)

• **A first responder** administers rescue medication to an unconscious patient, unaware that the medication was already administered. (Bryant 2016)

• **A UAV** on a search and track mission reaches a return-to-base condition, then finds and begins to track a new target. (Revill 2016)

• **A UAV** on a humanitarian assistance and disaster relief mission reports acceptable system status, then the operator suddenly commands the UAV to abort the mission without provocation (Reese 2017; Beaufait, Constable, and Jent 2017).

• **A Pressurized Water Reactor** experiences subsystem failures that cannot be traced to the source by the human operator. (Thrutchley, 2018)
What Has MP Been Used For?

- Modeling UAV swarm behavior (Launch, Ingress, Mission, Egress, Recovery)
- Modeling a UGV mission (Searching and clearing hazardous objects)
- Modeling fuze behavior (Specification to support LP/HC analysis)
- Modeling MQ-25 missions and a surrogate UAV system used for piloting SET
- Modeling a business process (Non-DOD conference approval process)

MP has been employed in theses, dissertations, teaching, and sponsored research efforts to model behaviors for system architecture, software architecture, software-intensive systems architecture, business processes, biological processes, geological processes, human interactions, medical procedures, operational missions, and entertainment events.
MP Tutorial Part 2: Basic Concepts

Kristin Giammarco, Ph.D.
Department of Systems Engineering

Mikhail Auguston, Ph.D.
Department of Computer Science
MP is used to define logical or physical objects in terms of their underlying behaviors.

– Human, technology, environment, etc.
Modeling of Behavior

- *Behavior* is defined as a set of events with two basic relations: precedence and inclusion.
MP Language: Event Grammar

Person:  
Check_weather_forecast
Drive_to_work;

Smart_Phone:  
Fetch_weather_data
Display_weather_forecast;

Environment:  
(Rainy | Sunny );
Let’s Try It!  Exercise 1

• Play MP-Firebird Video

• Go to http://firebird.nps.edu:3000

• Clear the default code and type:

```
SCHEMA Routine

ROOT Person:       Check_weather_forecast
                   Drive_to_work;

ROOT Smart_Phone:  Fetch_weather_data
                   Display_weather_forecast;

ROOT Environment:  ( Rainy | Sunny );
```
Exercise 1
Reading Event Traces

- **Person**
  - Check weather forecast
  - Drive to work

- **Smart Phone**
  - Fetch weather data
  - Display weather forecast

- **Environment**
  - Rainy
  - Sunny
Exercise 1
Constraining Interactions with Coordination

• Add:

```
COORDINATE $a: Check_weather_forecast FROM Person
          $b: Fetch_weather_data FROM Smart_Phone
DO ADD $a PRECEDES $b; OD;

Do loop
'a' event shall always precede 'b' event.
```

```
COORDINATE $a: Fetch_weather_data FROM Smart_Phone
          $b: ( Rainy | Sunny ) FROM Environment
DO ADD $a PRECEDES $b; OD;

Run
```

```
COORDINATE $a: ( Rainy | Sunny ) FROM Environment
          $b: Display_weather_forecast FROM Smart_Phone
DO ADD $a PRECEDES $b; OD;

Run
```

Run the model after each new constraint is added to check that it has the desired effect.
Exercise 1
Event Trace Inspection

- Person
  - Check weather forecast
    - Take umbrella
    - Drive to work

- Smart Phone
  - Fetch weather data
    - Display weather forecast

- Environment
  - Rainy

- Person
  - Check weather forecast
    - Take sunglasses
    - Drive to work

- Smart Phone
  - Fetch weather data
    - Display weather forecast

- Environment
  - Sunny
Exercise 1
Editing MP Code

• Edit:

```
ROOT Person:
Check_weather_forecast(
    Take_umbrella | Take_sunglasses
) Drive_to_work;
```

Add this line:

• Inspect:
Exercise 1
Editing MP Code

• **Edit:**

Add these lines:

```c
ROOT Smart_Phone:
Fetch_weather_data
Display_weather_forecast;

Display_weather_forecast: ( It_will_rain |
It_will_shine );
```

• **Inspect:**

1. Person → Smart Phone → Environment
   - Check weather forecast
   - Fetch weather data → Rainy
   - Display weather forecast
   - Take umbrella
   - Drive to work → It will rain

2. Person → Smart Phone → Environment
   - Check weather forecast
   - Fetch weather forecast → Sunny
   - Display weather forecast
   - Take umbrella
   - Drive to work → It will rain

3. Person → Smart Phone → Environment
   - Check weather forecast
   - Fetch weather forecast → Rainy
   - Display weather forecast
   - Take umbrella
   - Drive to work → It will rain

7. Person → Smart Phone → Environment
   - Check weather forecast
   - Fetch weather data
   - Rainy
   - Display weather forecast
   - Take sunglasses
   - Drive to work → It will shine

---

WWW.NPS.EDU
Exercise 1
Editing MP Code

• Edit:
  Comment out:
  /* ... */

• Inspect:

1. Person ➔ Smart Phone ➔ Environment
   - Check weather forecast
   - Take umbrella
   - Drive to work

2. Person ➔ Smart Phone ➔ Environment
   - Check weather forecast
   - Take umbrella
   - Drive to work
   - Display weather forecast
   - It will rain

3. Person ➔ Smart Phone ➔ Environment
   - Check weather forecast
   - Take sunglasses
   - Drive to work
   - Display weather forecast
   - It will rain

4. Person ➔ Smart Phone ➔ Environment
   - Check weather forecast
   - Take sunglasses
   - Drive to work
   - Display weather forecast
   - It will shine
Exercise 1
Editing MP Code

• Add:

COORDINATE $a: \{ \text{It will rain} \mid \text{It will shine} \} \quad \text{FROM Smart Phone, FROM Person}
DO ADD $a \text{ PRECEDES } \text{$b; OD;}

• Inspect:
Exercise 1
Editing MP Code

• Edit:

Comment out:

```c
/* ... */
```

Add

• Inspect:
Exercise 1

Saving Your Model

Export menu with options for different export types:
- Image Size: Medium
- File name: Authentication.mp
- Current Trace
- Selected Events
- Code
- Code and Graph
What is “Scope” in MP?

The Small Scope Hypothesis: most flaws in models can be demonstrated on small counterexamples.
Preloaded Example 1: Simple Message Flow

Exercise 2
Exercise 2

Preloaded Example 1: Simple Message Flow

```c
/*
example1_simple_message_flow.mp
Event grammar rules for each root define derivations for event traces, in this case a simple sequence of zero or more
events for each root.
The synchronized COORDINATE composition takes two root traces and produces a modified event trace, merging behaviors of Sender
and Receiver and adding the PRECEDES relation for the selected send/receive pairs. The coordination operation behaves as a "cross-cutting"
derivation rule.
Run for scopes 1 and up. The "Sequence" or "Swim Lanes" layouts are the most appropriate for browsing traces here.
*/

SCHEMA simple_message_flow
ROOT Sender: (send);
ROOT Receiver: (receive);
COORDINATE $x$: send FROM Sender, 
$y$: receive FROM Receiver
DO ADD $x$ PRECEDES $y$; OD;
```

- **Scope 1** will give you:

  - Sender sends message
  - Receiver receives message

- **Scope 2** will give you everything found at scope 1 plus:

  - Zero or more iterations

- **Scope 3** will give you everything found at scopes 1 and 2 plus:

  - Multiple send and receive interactions
Event grammar rules for each root define derivations for event traces, in this case a simple sequence of zero or more events for each root. The synchronized COORDINATE composition takes two root traces and produces a modified event trace, merging behaviors of Sender and Receiver and adding the PRECEDES relation for the selected send/receive pairs. The coordination operation behaves as a “cross-cutting” derivation rule. Run for scopes 1 and up. The “Sequence” or “Swim Lanes” layouts are the most appropriate for browsing traces here.
How MP Works to Expose Unwanted Behaviors

“prune” objectionable behaviors

leave behind only the desired behaviors
Authentication Model Demonstration

```
/* SCHEMA Authentication */

/* USER BEHAVIORS */

ROOT User: Provide_credentials
            [ CREDS_VALID Access_system ];

/* SYSTEM BEHAVIORS */

ROOT System: Verify_credentials
             ( CREDS_INVALID Deny_access )
             [ CREDS_VALID Grant_access ]
             [ Lock_account ];

/* INTERACTION CONSTRAINTS */

User, System SHARE ALL CREDS_VALID, CREDS_INVALID;

COORDINATE s₀: Provide_credentials FROM User
DO ADD s₀ PRECEDES s₁; DO;

COORDINATE s₁: Verify_credentials FROM System
DO ADD s₁ PRECEDES s₂; DO;

COORDINATE s₂: Deny_access FROM System
DO ADD s₂ PRECEDES s₃; DO;

COORDINATE s₃: Grant_access FROM System
DO ADD s₃ PRECEDES s₄; DO;

ENSURE #CREDS_INVALID <= 3;
ENSURE #Deny_access == 3 <-> #Lock_account == 1;
ENSURE #Grant_access == 1 <-> #Lock_account == 0;
```

User

Provide credentials

System

Verify credentials

CREDS INVALID

Deny access

Reenter credentials

CREDS INVALID

Deny access

Reenter credentials

Lock account
MP Tutorial Part 3:
Application Example: UAV Supporting Humanitarian Assistance / Disaster Relief Mission

Kristin Giammarco, Ph.D.
Department of Systems Engineering

Mikhail Auguston, Ph.D.
Department of Computer Science
Extracting Model Elements from a Mission Narrative

Humanitarian Assistance / Disaster Relief (HADR) Design Reference Mission (DRM)

1. Frame the problem
2a. Extract narrative of behavior
2b. Identify actors and actions

Mission Narrative

Ingress Phase

Pre-Conditions: Ingress Phase begins when the GCS Operator receives a launch clearance from the JTF C2

- GCS Operator receives launch command from JTF C2
- GCS Operator checks launch parameters for safety
- GCS Operator receives launch clearance from host platform
- GCS Operator commands Ground Crew to launch UAV
- Ground Crew commands UAV to launch
- UAV launches
- UAV maneuvers to clear obstacles then maneuvers to reach ingress altitude
- UAV transmits status (payload and systems) and position messages to GCS Operator (on-going throughout each phase of mission)
- If status and position are acceptable, GCS Operator commands UAV to proceed on ingress flight path, otherwise the GCS Operator commands UAV to return to base. Proceed to egress phase.
- UAV follows ingress flight path to reach on-station area and altitude
- GCS Operator monitors UAV status and position during flight path to on-station area and altitude
- The UAV reaches initial on-station waypoint and reports position to GCS Operator

Post-Conditions: The Ingress Phase ends when the UAV reaches the initial on-station waypoint
Ingress Phase

Pre-Conditions: Ingress Phase begins when the GCS Operator receives a launch clearance from the JTF C2

- GCS Operator receives launch command from JTF C2
- GCS Operator checks launch parameters for safety
- GCS Operator receives launch clearance from host platform
- GCS Operator commands Ground Crew to launch UAV
- Ground Crew commands UAV to launch

- UAV launches
  - UAV maneuvers to clear obstacles then maneuvers to reach ingress altitude
  - UAV levels off at ingress altitude
  - UAV transmits status (payload and systems) and position messages to GCS Operator (on-going throughout each phase of mission)
  - If status and position are acceptable, GCS Operator commands UAV to proceed on ingress flight path, otherwise the GCS Operator commands UAV to return to base. Proceed to egress phase.
  - UAV follows ingress flight path to reach on-station area and altitude
  - GCS Operator monitors UAV status and position during flight path to on-station area and altitude
  - The UAV reaches initial on-station waypoint and reports position to GCS Operator

Post-Conditions: The Ingress Phase ends when the UAV reaches the initial on-station waypoint
3. Identify event coordination

**ROOT JTF_C2:** Provide_launch_command;

**ROOT GCS_Operator:** Receive_launch_command;

**COORDINATE**

\[ a: \text{Provide\_launch\_command} \quad \text{FROM} \quad \text{JTF\_C2}, \]

\[ b: \text{Receive\_launch\_command} \quad \text{FROM} \quad \text{GCS\_Operator} \]

\[ \text{DO ADD} \quad a \text{ PRECEDES } b; \quad \text{OD}; \]

**Ingress Phase**

**Pre-Conditions:** Ingress Phase begins when the GCS Operator receives a launch clearance from the JTF C2

- GCS Operator receives launch command from JTF C2
- GCS Operator checks launch parameters for safety
- GCS Operator receives launch clearance from host platform
- GCS Operator commands Ground Crew to launch UAV
- Ground Crew commands UAV to launch
- UAV launches
- UAV maneuvers to clear obstacles then maneuvers to reach ingress altitude
- UAV levels off at ingress altitude

**Post-Conditions:** The Ingress Phase ends when the UAV reaches the initial on-station waypoint and reports position to GCS Operator
UAV Behavior During Ingress

*/

UAV MODEL

ROOT UAV:

UAV_Receive_navigation_reference
UAV_Receive_launch_command
Launch
Execute_climb
Maneuver_to_clear_obstacles
Maneuver_to_ingress_altitude
Level_off_at_ingress_altitude
Transmit_status_and_position

( Receive_command_to_proceed
  Follow_flight_path_to_reach_onstation_area_and_altitude
  Reach_onstation_waypoint
  )

  Receive_command_to_abort
  Abort_mission

;
Systems Are Modeled in Separate Roots

MP Models separate the behaviors in different systems...

JTF C2

GCS Operator

Ground Crew

Environment

... from interactions among the systems.

/*-----------------------------------------------------*/
UAV MODEL
---------
ROOT UAV:
  UAV_Receive_navigation_reference
  UAV_Receive_launch_command
  Launch
  Execute_climb
  Maneuver_around_obstacles
  Maneuver_altitude
  Level_off
  Maintain_altitude
  Transmit_status_and_position
  ( Receive_command_to_proceed
    Follow_flight_path_to_reach_onstation_area_and_altitude
    Reach_onstation_waypoint
    Receive_command_to_abort
    Abort_mission )
;

SCHEMA UAV_Ingress
/* Assumption: this model described a single UAV launch */
ROOT JTF_C2:
  JTF_Provide_launch_command;

GCS OPERATOR MODEL
---------------------
ROOT GCS_Operator:
  GCS_Receive_launch_command
  Check_launch_parameters_for_safety
  Receive_launch_clearance_from_host_ship
  ( Status_acceptable
    Command_UAV_to_proceed_on ingress)
    Status_unacceptable
    Command_UAV_abort )
;

GROUND CREW MODEL
--------------------
ROOT Ground_Crew:
  Crew_Provide_launch_command
;

NAV REFERENCE MODEL
----------------------
ROOT Environment:
  Provide_navigation_reference;

...
<table>
<thead>
<tr>
<th>Look for and address:</th>
<th>e.g.,</th>
</tr>
</thead>
<tbody>
<tr>
<td>Syntax errors</td>
<td>missing semicolons, misplaced parentheses</td>
</tr>
<tr>
<td>Typographical and transcription errors</td>
<td>misspelled event names, forgottenunderscores, wrong event names used</td>
</tr>
<tr>
<td>Notational errors</td>
<td>deviations from required notation and style guides adopted by the organization</td>
</tr>
<tr>
<td>Scope errors</td>
<td>running only at scope 1 when errors need scope 2 or 3 to manifest</td>
</tr>
<tr>
<td><strong>Required behaviors that are missing</strong></td>
<td>none of the event traces show a certain behavior that was required</td>
</tr>
<tr>
<td><strong>Prohibited behaviors that are present</strong></td>
<td>some event traces show a certain behavior that was prohibited</td>
</tr>
<tr>
<td>Unspecified valid behaviors</td>
<td>some event traces show wanted behaviors that were not explicitly required</td>
</tr>
<tr>
<td><strong>Unspecified invalid behaviors</strong></td>
<td>some event traces show unwanted behaviors that were not explicitly prohibited</td>
</tr>
</tbody>
</table>
Required Behaviors that are Missing

Identification of missing required behaviors involves cycling on requirement specification completeness

G.1.3.3.14. Ingress Phase:

Pre-Conditions: Ingress Phase begins when the GCS Operator receives a launch clearance from the JTF C2

- GCS Operator receives launch command from JTF C2
- If shipboard, the GCS Operator requests host ship maneuver to achieve launch parameters
- GCS Operator checks launch parameters for safety
- GCS Operator receives launch clearance from host platform
- GCS Operator commands Ground Crew to launch UAV
- Ground Crew commands UAV to launch
- UAV launches
- UAV maneuvers to clear obstacles then maneuvers to reach ingress altitude
- UAV levels off at ingress altitude
- UAV transmits status (payload and systems) and position messages to GCS Operator (on-going throughout each phase of mission)
- If status and position are acceptable, GCS Operator commands UAV to proceed on ingress flight path, otherwise the GCS Operator commands UAV to return to base. Proceed to egress phase.
- UAV follows ingress flight path to reach on-station area and altitude
- GCS Operator monitors UAV status and position during flight path to on-station area and altitude
- The UAV reaches initial on-station waypoint and reports position to GCS Operator

Post-Conditions: The Ingress Phase ends when the UAV reaches the initial on-station waypoint

---

Does GCS Operator confirm receipt?

Should there be a similar command to the host ship to prepare/maneuver to receive the UAV?

Can anyone else give the launch command?

Are there situations in which the UAV wouldn't climb out?
Identification of missing required behaviors involves checking that all required behaviors in the specification are in fact represented in the model.

Since this event is coordinated with `Receive_command_to_proceed` in lines 97-99, and `Receive_command_to_proceed` is inside a composition of alternative events, these alternative events will always be rejected.
Required Behaviors that are Missing

To correct the issue, corresponding branches of coordinated behavior are established employing alternative events in the COORDINATE statement...

New alternative paths for Status_acceptable and Status_unacceptable are added...

...and coordinated with the preceding event Transmit_status_and_position...

... to provide two possible branches of coordinated behavior in both the GCS_Operator and the UAV.
Prohibited Behaviors that are Present

• The larger a behavior model, the more automated tools are needed for verifying absence of known unwanted behaviors

Are there any scenarios in which the UAV is launched without permission from the JTF C2?

Assertion Check:

CHECK #Launch
#JTF_Provide_launch_command
ONFAIL SAY("Unauthorized Launch");

Formal Requirement:

ENSURE #Launch
#JTF_Provide_launch_command
FROM UAV <= FROM JTF_C2;
Abort command could result from
• an overriding third party
• a problem observed through other means
• operator error
Main Body
- Motivation and Objectives
- Related Work
- Technical Accomplishments
- Conclusions
- FY18 Plans

Appendix A: List of Publications and Invited Talks
Appendix B: References Cited
Appendix C: Collaborator Courses that Integrate or Contribute Research Results
Appendix D: Monterey Phoenix Overview
Appendix E: Preliminary Catalog of Reusable Architecture Patterns
Appendix F: Instructions for Downloading UAV Models
Appendix G: Model Based V&V (MCSE MPT) Demonstration

Five key concepts support the development of system and SoS behavior models.

Model developers can expose both wanted and unwanted behaviors without specialized skills.

Model developers can generate all use case scenarios automatically up to a scope limit.

Given enough time, all behaviors not explicitly suppressed will probably eventually emerge.

Model developers can detect, classify, predict and control emergent behaviors with MP.

Probabilities for behavior outcomes can be calculated.

Templates for behavior patterns can be derived inductively.

All possible interactions for a model can be summarized in matrix form.

System behaviors could be formally specified consistently by both government (in requirements) and contractors (in implementation).

Many architecture modeling lessons learned are formalizable into anti-patterns.

MP may be mapped to SysML and other graphical notations.
• MP has simple and uniform concepts to model everything: system, actors, and environment (including other systems – SoS emerging behavior), it is not necessary to be a programmer to start using MP
• MP is useful for business process modeling as well as systems and software architecture
• MP is easy to use (behavior models are in fact pseudo-codes)
• MP has the ability to cover several layers of abstraction – from top level requirements and architecture to design level models in a uniform language – business process models are part of requirements
• MP has the ability to support developers and stakeholders interactions – humans understand examples of behavior (event traces) better than formal specifications
Way Ahead

- User-defined relations – potential use for network topology and other useful modeling aspects, to visualize event dependencies, etc.
- Event attributes to estimate time and other resources
- Custom architecture views and queries, consistency guaranteed
  - if MP model changes, all architecture views are updated automatically
- Potential to support project cost estimates early at the architecture level
MP Tutorial Part 4:
Practice Exercise: Model a System on MP-Firebird

Kristin Giammarco, Ph.D.
Department of Systems Engineering

Mikhail Auguston, Ph.D.
Department of Computer Science
MP Event Grammar Patterns

```
A: B C;       Ordered sequence of events (A includes B followed by C)
A: (B | C);   Alternative events (A includes B or C)
A: [B];       Optional event (A includes B or no event at all)
A: (* B *);   Ordered sequence of zero or more occurrences of event B in A
A: (+ B +);   Ordered sequence of one or more occurrences of event B in A
A: {B, C};    Unordered set of events B and C in A (B and C may happen concurrently)
A: {* B *};   Unordered set of zero or more occurrences of event B in A
A: {+ B +};  Unordered set of one or more occurrences of event B in A
```
Questions?

Monterey Phoenix and Related Work:

https://wiki.nps.edu/display/mp

firebird.nps.edu

kmgiamma (at) nps.edu