Process Algebra Extension
for UxAS

UAV Control & Simulation System

Presented to:

United States Air Force Research Lab

By:

Angela Scott
Paul Coen
Winston Smith
Index

1.) Background
2.) Customer Needs
3.) Constraints
   a.) Style
   b.) Practical
   c.) Implementation
4.) Risks
5.) Verification
6.) Requirements
Section 1: Background

The AFRL assists the Air Force in its operations by improving upon or creating new technologies useful to Air Force missions. One such technology is the system used to process many autonomous vehicle missions, UxAS. UxAS needs to be able to efficiently compute the best routes for all tasks assigned to drones.

The current system has somewhat limited expressivity. It uses process algebra to describe tasks. They have three operators: AND, OR, and precedence. The AFRL would like three operators to be added to enhance the expressiveness of UxAS. The first, ~, will guarantee that p will start running before e. The second, dot, will guarantee that p ends before e begins to run. The AFRL would also like timing to be implemented in UxAS.
Section 2: Customer Needs

The added ability to control the timing of events in missions will be an invaluable asset for them.

- Add two process algebra operators
  - Tilde: p must begin before beginning e
  - Dot: p must finish executing before beginning e
- Add timing syntax to the process algebra operators:
  - Ex.
    - (p~e)_[0 .. 10]
      - a will start within 10 seconds of starting p
    - p-[14:00 .. 15:00]
      - p will start between 14:00 and 15:00
Section 3: Constraints

Stylistic Constraints:
The AFRL programs in a modified GNU style of C++. All three members of the team are experienced working in C++, but the GNU style is foreign to all of us. Additionally, there are thousands of lines of stylistic specifications that we need to adhere to, which means more time taken in code reviews/rewriting.

Practical Constraints:
UxAS must run as efficiently as possible, so for considerations other than hard limits, we should program our addition as if it would be run on a low-power embedded system. Additionally, we must decide whether to implement a new module(s) or modify existing one(s).

Implementation Constraints:
There are several constraints on what tools we can use in our implementation. These are probably meant to aid readability, efficiency, and memory use. This subsection details the subset of these constraints that may cause problems for our team, given our personal styles to solving problems.

1. Can’t use variable-size argument lists (variable-size arrays passed in, to simulate this?)
2. make_weak() and make_unique() are new functions to us
3. Some clarification needed - should we use pass by reference over pass by value, even when we’re just passing ints or other small-size types?
4. Public member functions of classes can’t return non-const pointers
5. Semi-convoluted constraints on classes, particularly inheritance
6. Team has limited experience with singleton objects, which are greatly preferred to global variables
Section 4: Risks

Implementing these new operators does not come without some sort of risk to the given software. Our operators could create interesting scenarios that can cause UxAS to fail to different degrees.

We must be prepared to deal with potential side-effects in their code when adding our operators. Understanding both how their code operates and how our code operates will be of utmost importance.

Verification of the system will be required to minimize risks, the level of validation the code will require is to be thought of as a life-critical system.

We must maintain the strict style requirements they have for code added to UxAS. Failing to do this may cause issues within their system that are incredibly hard to detect.

Large amount of work will be difficult to manage without some kind of software engineering approach. Failing to follow this could have a range of effects; From taking longer then expected to not finishing the project on time.

Overall, the risks can be minimized with enough time and discipline. Smart decision making will make this project much easier to manage and complete on time.
Section 5: Verification

As there is a potential that our code written for OpenUxAS may be merged with the master branch, there is a chance that the Air Force and other universities will use our code. It is therefore critical that there is some quality assurance of our code; that it follows the code standards for OpenUxAS and will be as error-free as possible when it goes in for final review at the AFRL.

To ensure quality, we will use several agile methods, which will be based around our senior project class meetings times in the spring semester (MWF). We will have sprints of roughly a week long, with the sprints starting and ending each Friday. On Fridays, we will: QA and code review, reflect on what did well/not so well and adjust our overall plan for the project based on that reflection, and assign tasks for the next week. On Monday and Wednesday we will have a meeting in class, ensuring that each member is on track and providing each other with any assistance that we may need.

Once we are satisfied that our code has met our goals, then we will have a final overall review. We will review the logic of our code, optimize where possible, and adjust the code style to meet the requirements of OpenUxAS as needed.
## Section 6: Requirements

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Add a ~ operator to UxAS. For the string “p ~ e”, p will begin before e does.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reasoning</td>
<td>It provides greater control over mission planning and task scheduling</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Add a dot operator to UxAS. For the string “p . e”, p will finish running before e begins.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reasoning</td>
<td>It provides greater control over mission planning and task scheduling</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Add an operator _[n1 .. n2]. For the string “(p ~ e) _[1 .. 3]”, e will start anywhere from 1 to 3 seconds after the start of p.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reasoning</td>
<td>Allows the user to ensure that time-sensitive chains of events happen in correct order</td>
</tr>
</tbody>
</table>