Modeling the CFS Middleware for CPN based Analysis

Charles Hartsell – CS 6388 Project Report

Introduction & Background

Cyber Physical Systems (CPS) are often used in mission- or safety-critical applications, and therefore require thorough system verification before deployment. The software component of such systems is commonly built on top of a suitable middleware. The Core Flight System (CFS) is one such middleware from NASA originally intended to reduce development time and cost of spacecraft flight software. CFS contains a core application environment providing a set of common software services, as well as a set of reusable flight-qualified applications. Many CPS are used in unmanned applications and require a suitable autonomy engine such as the Plan Execution Interchange Language (PLEXIL). PLEXIL is a general plan execution language intended for use in a variety of autonomous applications. PLEXIL plans can be executed by the PLEXIL executive, and typically act as a high-level, script-based controller.

Our ongoing work attempts to perform analysis and verification of systems built using the CFS framework and the PLEXIL autonomy engine. We have created a Colored Petri Net (CPN) model of the CFS framework which allows new system configurations to be loaded without changing the structure of the CPN model. Instead, relevant properties of the system, such as application behavior, message subscriptions, etc., can be encoded into large data structures. These structures can then be loaded into the CPN model as initial tokens and the new system configuration can be analyzed. However, generation of these data structures is currently done by hand, which is both time consuming and error prone.

The goal for this project is to implement a Domain Specific Modeling Language (DSML) for systems built on the CFS framework and using the PELXIL autonomy engine. This DSML must be able to encode all relevant system properties needed for CPN-based analysis. Additionally, a WebGME plugin for generating CPN initial tokens from the graphical model will be created, allowing new system configurations to be analyzed more quickly and reliably.

Domain Modeling

The meta model is broken into 3 primary aspects: the CFS software bus, applications, and the environment. One of the core services CFS provides is a publish/subscribe based messaging service called the software bus. Messages must belong to a topic, represented by an integer message ID, and may contain an arbitrary number of data entries. Messages are routed between message queues called pipes, and each application may own multiple pipes. Applications can register a subscription with the software bus, indicating that they wish to receive all messages on a given topic in a particular pipe. The software bus must maintain a subscription table containing a record of all subscriptions so that messages may be routed appropriately as they are received. Additionally, connections between pipes are also modeled and allow both publisher and subscriber pipes to be connected to the central software bus. A suitable meta-model was created based on the above description, and is shown in Figure 1.

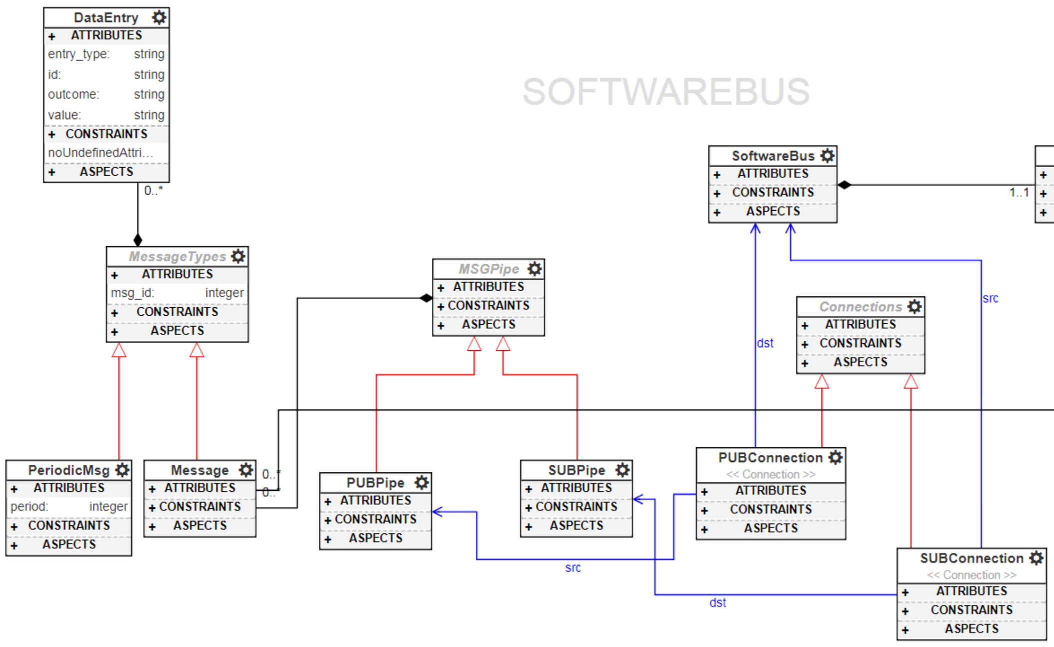


Figure 1. Meta-model of CFS software bus.

Most applications within CFS follow a set of general execution guidelines, and can be adequately modeled with a single, generic application model. These applications typically run at a set frequency, but may also be triggered by events, typically the arrival of a particular message type, if so desired. Normally, an application will begin an execution cycle after receiving a periodic wake-up message. The cycle begins by processing all messages which have arrived in the application's message queue since the last cycle ended. Each message type has an associated message handler which may perform some set of actions and will often produce output messages based on the content of the input messages. To model this behavior, applications may contain a variety of actions which are driven by message events. For generic applications, the primary actions are message handlers and periodic actions. Message handlers have both trigger messages and output messages, while periodic actions only contain output messages and a period attribute. Additionally, an application may subscribe to any message topic and must direct this subscription to one or more of its pipes. An application may only contain one publisher pipe, which publishes directly to the software bus. The application meta-model is shown in Figure 2.

Most CFS applications follow the general execution guidelines, and can be modeled as a generic application. However, there are no hard restrictions which force applications to follow these guidelines, and the behavior of some applications cannot be adequately modeled using the generic model alone. Two such applications we have considered are the CFS scheduler and the PLEXIL executive. The CFS scheduler allows other applications to register a periodic wakeup message which will trigger an execution cycle to begin. The scheduler is modeled individually, and contains a special schedule table indicating how often each application needs to be executed. The PLEXIL application provides an autonomy engine for CFS based systems, and may contain any number of PLEXIL plans.

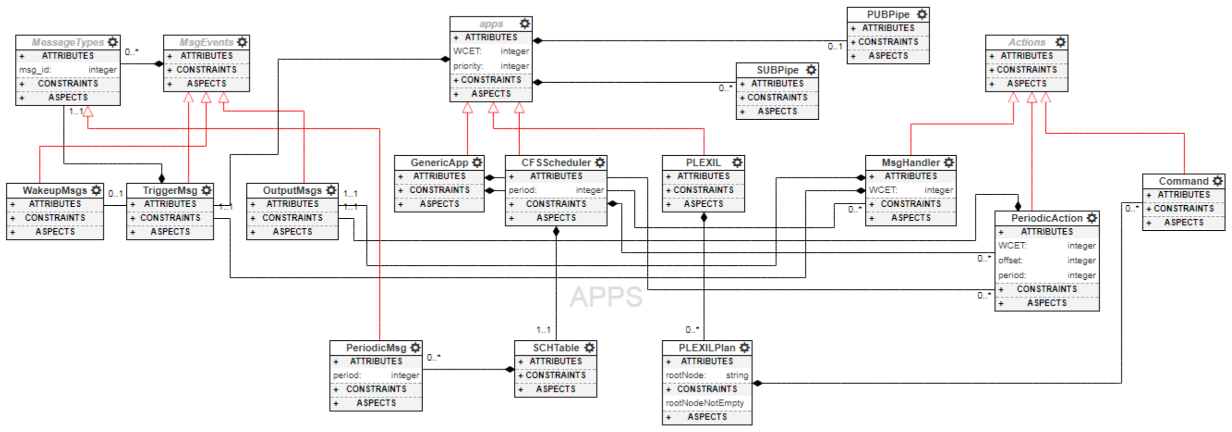


Figure 2. Meta-model of CFS applications.

Any CFS system is only allowed to contain a single environment model. Our current model of the environment is relatively simple and contains events, a single subscription pipe, and a single publisher pipe. Each event has a fixed time of occurrence and may contain any number of data entries, as shown in the meta-model in Figure 3. We are currently working on a more dynamic environment for our CPN model, and will port these changes to the DSML as appropriate.

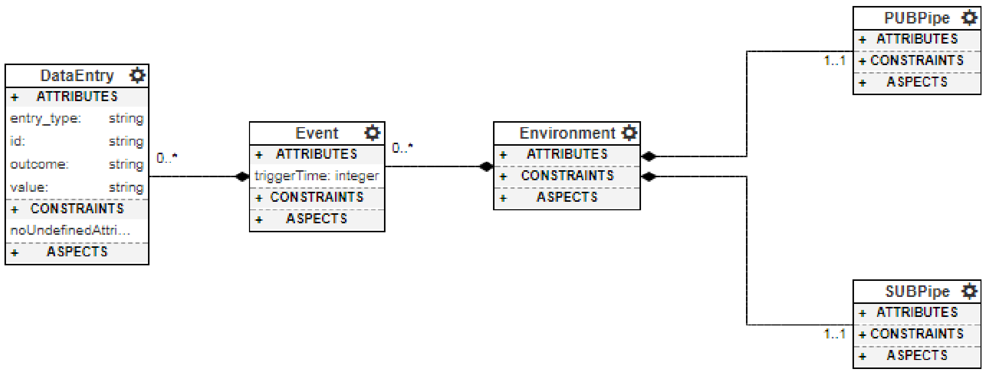


Figure 3. Meta-model of environment.

CPN Plugin

Our CPN model for CFS systems is structured in a hierarchical manner, with each major component having its own CPN model, all linked together by a single top-level model. Figure 4 shows each of these component models as a rectangle, along with their connections to other components. The ovals represent places in the model where information about a given system configuration is loaded as an initial token.

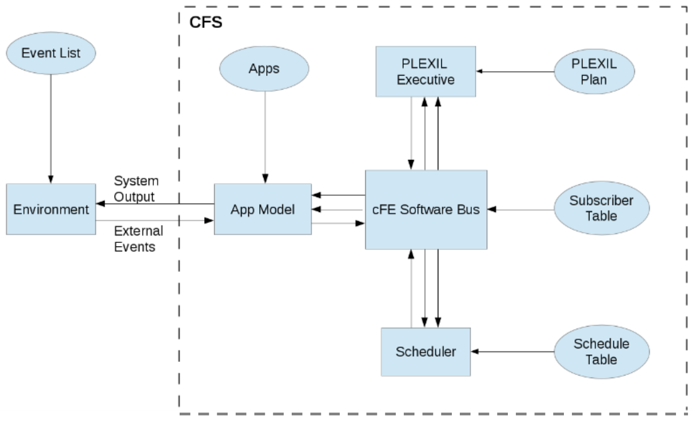


Figure 4. Structure of CFS model.

The CPN plugin performs two primary functions. First, it searches for all publisher and subscriber pipes in a CFS system model and collects all the published message topics, specified with human-readable names. Duplicate message topics are removed from the set of publisher pipes, and each published topic is assigned a unique ID. The set of subscriber pipes is searched to identify all message subscriptions within the system. The plugin then removes any outdated subscriber table from the software bus before creating an updated subscriber table. The plugin also searches the entire system model and updates all message instances with their correct topic ID before committing the changes to the model. Secondly, the plugin generates a suitable data structure for most of the initial token places in Figure 4, with the only exception being the "PLEXIL Plan" place. PLEXIL plans are modeled at a high-level within the DSML, but are not sufficiently detailed to generate the required tokens. This is because an external python program has previously been developed which generates these tokens directly from the PLEXIL plan itself. The plugin returns a zip file containing a total of 5 generated text files.

As an example, we considered a basic aircraft controller built using CFS that consists of 4 generic applications, the CFS scheduler, PLEXIL, an environment, and the CFS software bus as shown in Figure 5. Various messages were added to each application's message pipes, with all applications publishing at least one message topic. The plugin was able to successfully assign unique IDs to all message types, update all messages in the system to the correct IDs, and generate a new subscription table as shown in Figure 6. The generated text file containing a subscription table and message name to ID map is also shown in Figure 7. Currently, the user must manually copy these generated data structures from the text files into the CPN model for analysis. Also note that the rest of the system (application behavior, environment events, etc.) was created and generated successfully as well, but is omitted here for brevity.

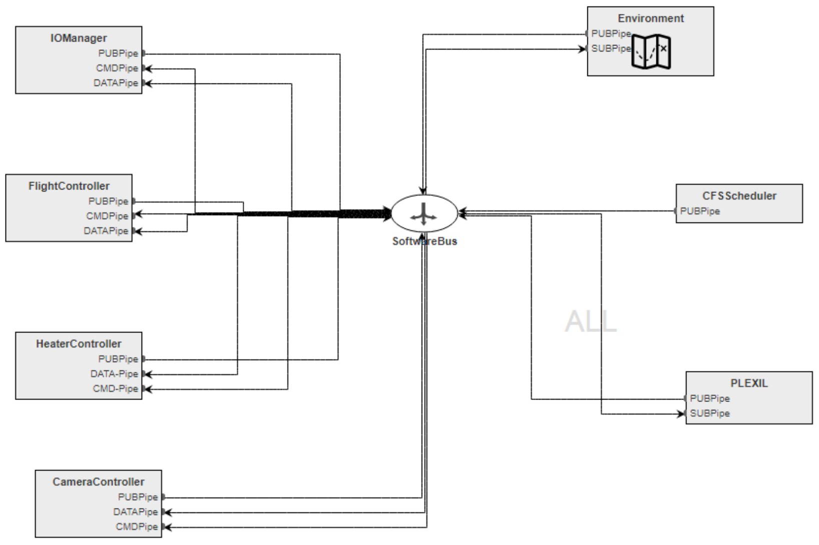


Figure 5. Aircraft controller system.

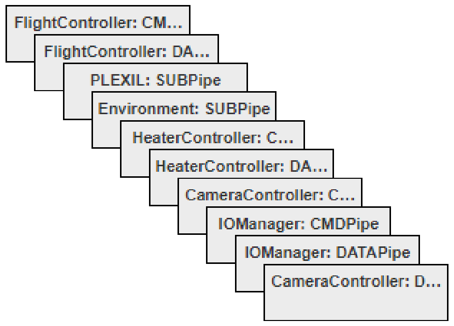
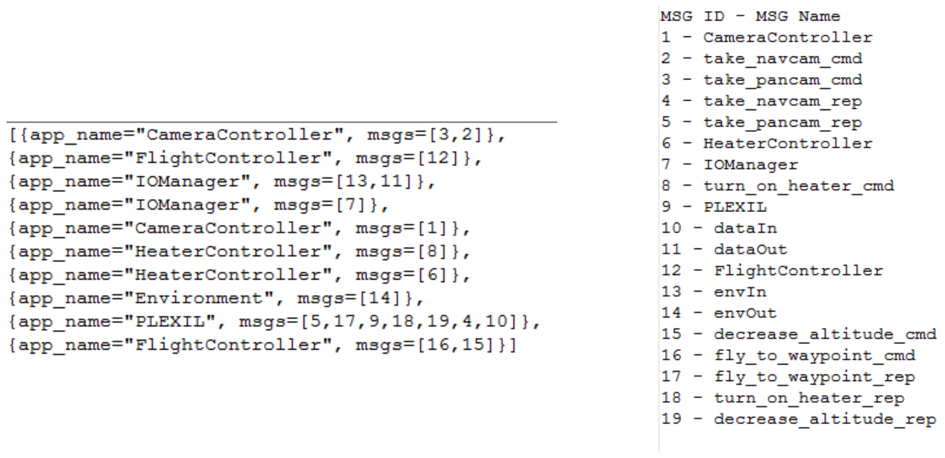


Figure 6. Generated subscription table and message map.



```
[{app_name="CameraController", msgs=[3,2]},
{app_name="FlightController", msgs=[12]},
{app_name="IOManager", msgs=[13,11]},
{app_name="IOManager", msgs=[7]},
{app_name="CameraController", msgs=[1]},
{app_name="HeaterController", msgs=[8]},
{app_name="HeaterController", msgs=[6]},
{app_name="Environment", msgs=[14]},
{app_name="PLEXIL", msgs=[5,17,9,18,19,4,10]},
{app_name="FlightController", msgs=[16,15]}]
```

```
MSG ID - MSG Name
1 - CameraController
2 - take_navcam_cmd
3 - take_pancam_cmd
4 - take_navcam_rep
5 - take_pancam_rep
6 - HeaterController
7 - IOManager
8 - turn_on_heater_cmd
9 - PLEXIL
10 - dataIn
11 - dataOut
12 - FlightController
13 - envIn
14 - envOut
15 - decrease_altitude_cmd
16 - fly_to_waypoint_cmd
17 - fly_to_waypoint_rep
18 - turn_on_heater_rep
19 - decrease_altitude_rep
```

Figure 7. Generated subscription table and message map tokens.

Conclusion

The DSML and accompanying plugin created in this project allow for generic CFS-based systems to be modeled within WebGME and interpreted into a data format compatible with our CPN model. Automating token generation in this way should result in significant time savings and reduced error rate when updating the CPN model for new system configurations. Future work on this project may include integrating the plugin with the CPN Tools software so that the user is not required to manually copy the tokens into the CPN model. Integration may also allow for model analysis to be at least partially automated, assuming CPN Tools provides some API functionality for this.