A Model-Based Design Tool for Systems-Level Spacecraft Design

Brandon Eames, Allan McInnes, Jared Crace, Joe Graham
Electrical & Computer Engineering
Utah State University
Overview

- Motivation
- Modeling Formalism
- Tool Infrastructure
- Tool Overview
Motivation

- At the subsystem level designs are mathematically modeled and analyzed to ensure that they are correct.

- Spacecraft system-level behavior is typically described in prose and diagrams:
  - Provides for documentation but not analysis.
Goal

- Enable capture of system-level behavior models that can be rigorously analyzed to detect errors earlier in the development cycle
Communicating Sequential Processes
- Mathematical formalism originally developed for modeling interacting software components
- History of industrial use (Daimler, Qinetiq, INMOS)
- Model behavior as processes composed of events

```plaintext
datatype T = t1 | t2 | t3
channel tuple: (T, T)
channel inx, iny, outx, outy: T

MergeXY =
  let
    Xin = inx?x -> [[]] y:T @ tuple.(x,y) -> Xin
    Yin = iny?y -> [[]] x:T @ tuple.(x,y) -> Yin
  within
    Xin ||{||tuple||}|| Yin
```
Using CSP

- Tool support for CSP models allows:
  - Manual exploration to improve understanding of system behavior
  - Exhaustive analysis to verify that modeled behavior is correct

- Model-builders must understand CSP language/theory

---

<table>
<thead>
<tr>
<th>Specification</th>
<th>Model</th>
<th>Implementation</th>
</tr>
</thead>
<tbody>
<tr>
<td>COPY [F= (RegTIMP(sto+4))(tack2))(CHAOS(tack2))(tack1,tack2)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>COPY [F= (RegTIMP(sto+3))(tack2))(CHAOS(tack2))(tack1,tack2)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>COPY [F= (RegTIMP(sto+2))(tack2))(CHAOS(stack2))(tack1,tack2)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TSpec(sto+4) [F= TIMP]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TSpec(sto+3) [F= TIMP]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TSpec(sto+2) [F= TIMP]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Clock [FD= (TABP/mail)(tack,error))(CHAOS(mail,error))(diff(Events(tack2)))</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Approach

- Develop a visual tool that supports system-level behavior modeling, and maps diagrams to CSP models
The Generic Modeling Environment

- Tool Infrastructure for implementing Domain-Specific Modeling Languages (DSMLs)
- High-level interfaces for interpreter creation: “compiler” for the visual language
  - Translate the captured diagrams into “something useful”
Spacecraft Design Workbench

- Modeling language supports three classes of diagrams:
  - Mode Transition Diagrams, FFBDs, Dataflow Diagrams
- Allows the capture of distinct views of system
- Rules and constraints govern interaction between views
- Defined mapping between diagram elements and CSP constructs
Mode Transition Diagrams

- Capture the modal behavior of the spacecraft
- States indicate system modes
- Transitions correspond to events
  - Events can be internal or external
  - Transition occurs on event being raised
- “Function” associated with each state
  - Models the spacecraft behavior executed while in that state

![Mode Transition Diagram](image)
Functional Flow Block Diagrams

- FFBDs define the order of execution of system functions
- Model concurrency, sequencing and iteration through simple associations
- Hierarchical: A function box can be another FFBD
Dataflow Diagrams

- Dataflow diagrams capture data exchange and sharing between functions
- Hierarchical function definition supported in dataflow view
  - Input-Output Transfer functions
  - FFBDs
- External ports capture “external” data sources/sinks
  - ex. sensors, telemetry system
- GME model traversal interfaces used to create model interpreters
- Interpreter prototype under construction to automatically generate CSP code from captured diagrams

ScienceModeFFBD =
let
  AttitudeBlock = FFBDblock(cmd_att, set_attitude)
  ScienceBlock = FFBDblock(sci_in_req, sci_out_ack)
  FFBD =
      (begin_ffbd.science ->
       FFBDbiteration(get_modestate, diff(Mode,{science}),
       FFBDo((AttitudeBlock, ScienceBlock)));
       end_ffbd.science -> FFBD)
    [] (end_ffbd.science -> FFBD)
within
  ((AttitudeCommand ||| Science)
   [||{cmd_att, set_attitude, sci_in_req, sci_out_ack}]))
FFBD) \ {{sci_in_req, sci_out_ack}
Analytical tools can help designers gain confidence in system-level designs
- Analysis requires formal semantics

SDW bridges the gap between formal semantics and intuitive visual constructs
- builds on industrially-proven formal methods
- allows rigorous system-level analysis
- makes system-level analysis accessible

Paper provides a brief example illustrating both the visual constructs and CSP verification