KNOWLEDGE-ASSISTED REASONING OF REQUIREMENTS

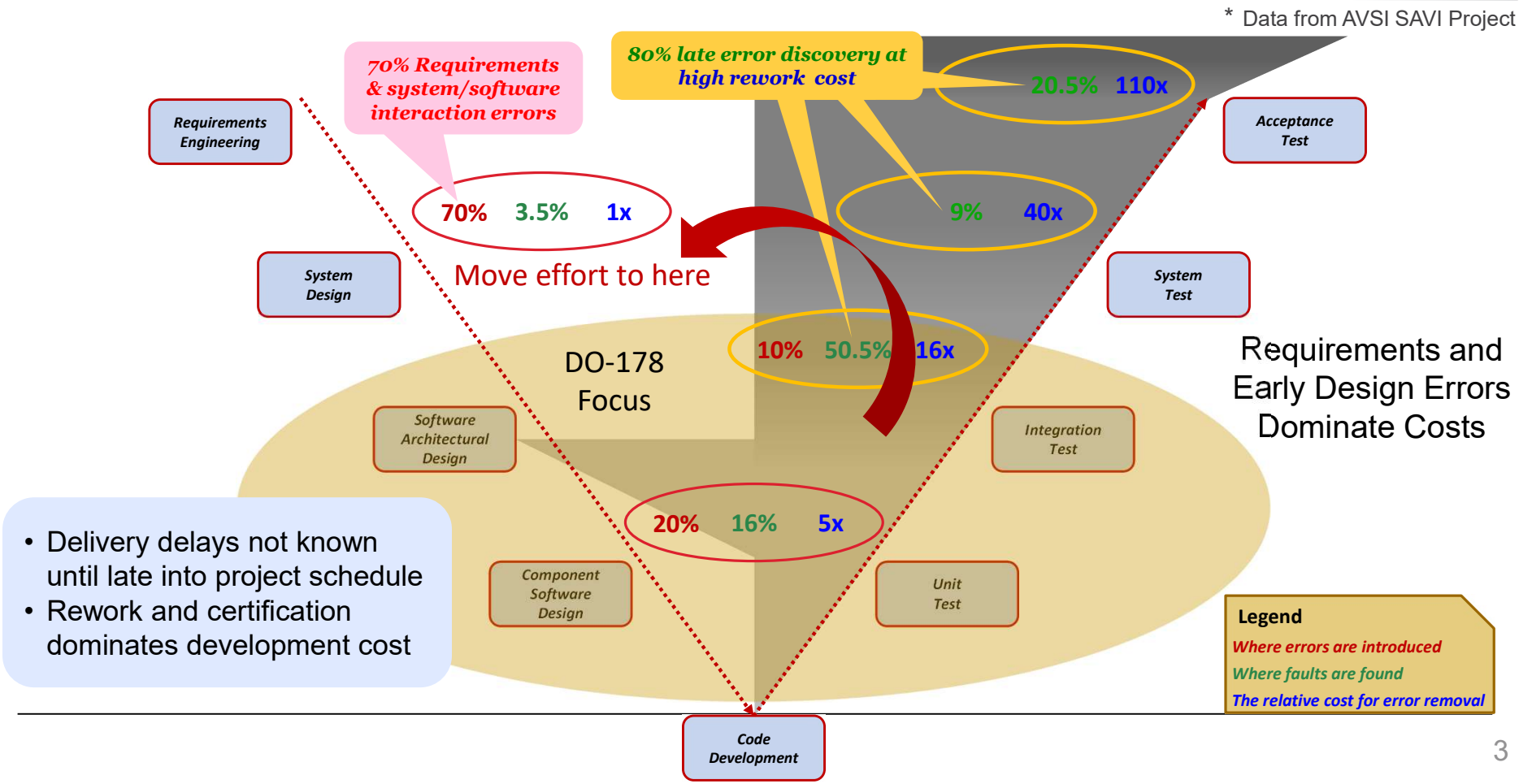
Using Event Calculus and Goal-Directed Answer Set Programming

Brendan Hall*, Sarat Chandra Varanasi†, Jan Fiedor‡§, Joaquin Arias¶, Kinjal Basu†,
Fang Li†, Devesh Bhatt*, Kevin Driscoll*, Elmer Salazar†, Gopal Gupta†
*Honeywell Advanced Technology, Plymouth, USA
†Department of Computer Science, The University of Texas at Dallas, Richardson, TX, USA
‡ Honeywell International s.r.o, Brno, Czech Republic
§Brno University of Technology, Brno, Czech Republic
¶Universidad Rey Juan Carlos, Madrid, Spain

OVERVIEW

- Our Motivation
  - Requirement Challenges
  - A Potential Paradigm Shift
- The MIDAS Approach
  - Model-Augmented Requirement Context
  - Constrained Natural Language Requirement Intent
- Formalizing MIDAS with Event Calculus and s(CASP)
- Example of Knowledge-Assisted Reasoning

MOTIVATION: UNDERSTANDING THE TRUE COSTS OF REQUIREMENT MATURITY ISSUES



- Delivery delays not known until late into project schedule
- Rework and certification dominates development cost

DEVELOPMENT EVOLUTION

1990
2000
2020

Document-centric design

-
-
-

Mo

Diagram Centric (e.g., SysML, Capella)

Simulation Centric (e.g., Simulink)

- Semantic-based models focussed on formal Intent Capture
- Model-Augmented Requirements - Hybrid-Requirements

Static notations 'alien' to domain experts

ulation

- *Limited provisions for software factoring and operational scenario support*

Constrained Natural Language

Common Information Model

+

- *Limited integration of dynamic and static views*
- *Informal semantics*
- *Tool specific workflows*

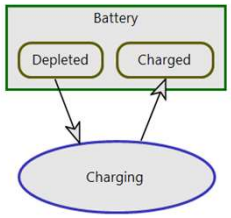
Model Augmented Formal Requirements

Knowledge-Enabled System Engineering

Requirements still monolithic text and not formal

ESTABLISHING REQUIREMENT CONTEXT: OBJECT–PROCESS METHODOLOGY (OPM)

- A language and methodology for modeling complex systems of any kind
- Recognized as ISO PAS 19450 standard
- Based on the minimal universal ontology of stateful objects
  - Processes transform objects by:
    - Creating new objects
    - Consuming existing objects
    - Changing states of existing objects



Charging changes Battery from Depleted to Charged.

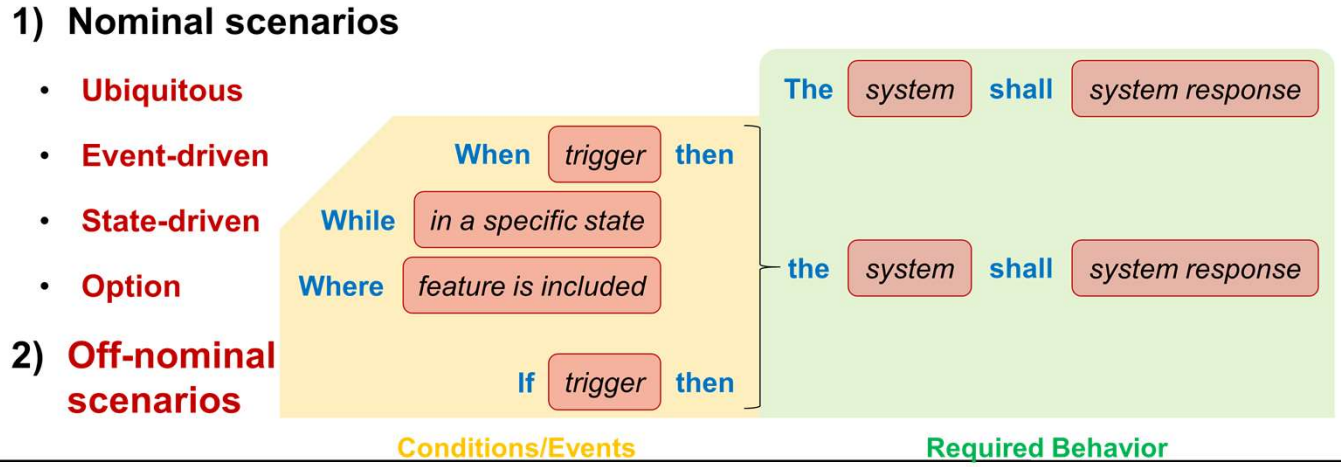
- Bimodal Representation
  - Graphical and Textual Form of Information Model
- Objects and processes can be refined to any desired level of detail via refinement-abstraction mechanisms:
  - In-zooming & Out-zooming (primarily for processes)
  - Unfolding & Folding (primarily for objects)

OPM Model Provides Visual and Ontological Context for the **Boundary of Intent**

DEFINING INTENT: EASY APPROACH TO REQUIREMENTS SYNTAX (EARS)

- A simple pragmatic requirements approach developed by Alistair Mavin and a team whilst working at Rolls Royce PLC
- Comprises constrained natural language templates to help structure and target requirements for better clarity and consistency

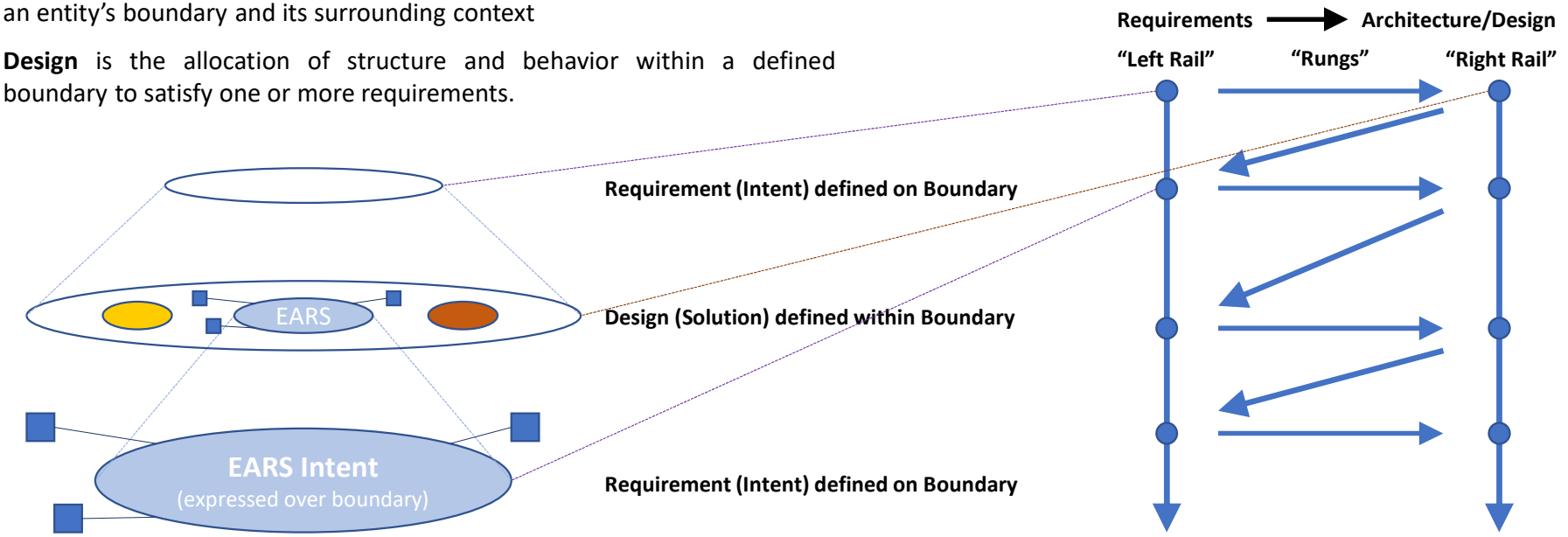
- Simple Keywords used to separate and guide focus to different requirement concerns
  - Nominal & Off-Nominal Scenarios
  - Event and State-Driven System Responses
  - Feature-Driven Requirements
  - System Invariants & Properties



SEPARATING WHAT FROM HOW: MIDAS DEFINITIONS AND THE Z-LADDER

Intent is an unambiguous statement of the intended interaction between an entity's boundary and its surrounding context

Design is the allocation of structure and behavior within a defined boundary to satisfy one or more requirements.



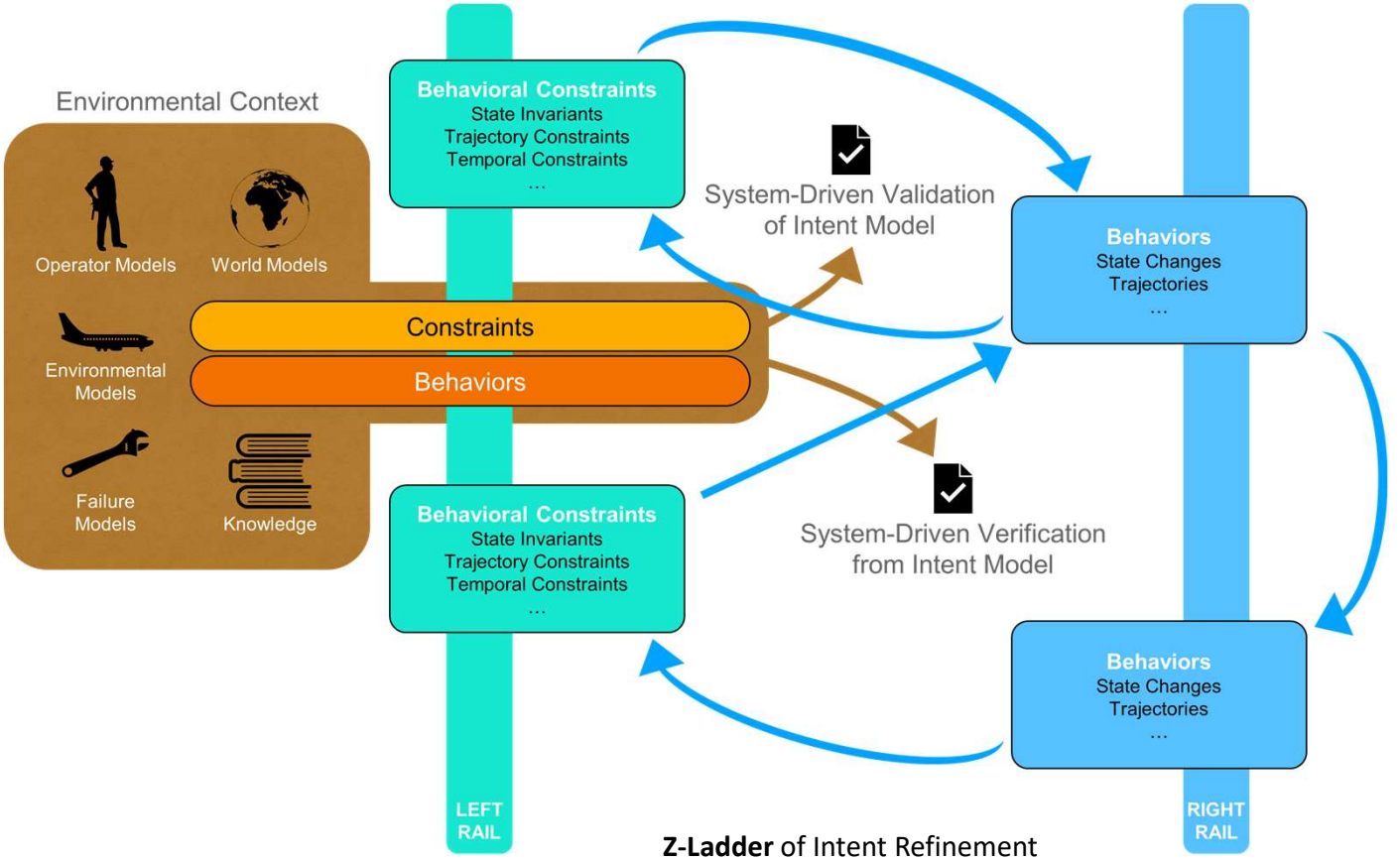
Boundaries relate to C*onceptual Abstractions* and may necessarily be Structural or Physical

For more details see: Hall, Brendan, Jan Fiedor, and Yogananda Jeppu. "Model Integrated Decomposition and Assisted Specification (MIDAS)." In INCOSE International Symposium, vol. 30, no. 1, pp. 821-841. 2020.

FORMALIZING THE MIDAS MODEL

- Approach: Formalize the *MIDAS Intent Model* using the event calculus, implemented in goal-directed Answer Set Programming (the s(CASP) system)
- Event calculus devised by AI researchers to model dynamic systems:
  - Designed around events and fluents
  - Axioms to model their interaction
- EARS and OPM can be mapped to event calculus directly
  - Event triggers and state qualifies identified using EARS keywords
  - State of the OPM objects map to fluents
  - Level of *Temporal Abstraction* elegantly represented in the event calculus
- OPM Information model can also be used to capture *physical world assumptions and additional domain knowledge*
  - *The OPM and Event Calculus formalism may also be Intuitive than other* approaches such as hybrid automata or Kripke structures
    - Much smaller "semantic gap"; avoid "design" pollution
    - Completeness (to the extent possible) can also be checked

THE INTEGRATED MIDAS INTENT MODEL



Z-Ladder of Intent Refinement

EVENT CALCULUS

- Actuator actions map to events (switch on the home heating system)
- Sensor readings map to fluents (room temperature)
- Conditions on sensors map to triggers (temperature exceeds 80^{o}F)
- Basic Event Calculus Axioms

BEC1. $StoppedIn(t_1, f, t_2) \equiv$
$\exists e, t \ (\ Happens(e, t) \wedge t_1 < t < t_2 \wedge (\ Terminates(e, f, t) \vee Releases(e, f, t) \ ) \ )$

BEC2. $StartedIn(t_1, f, t_2) \equiv$
$\exists e, t \ (\ Happens(e, t) \wedge t_1 < t < t_2 \wedge (\ Initiates(e, f, t) \vee Releases(e, f, t) \ ) \ )$

BEC3. $HoldsAt(f_2, t_2) \leftarrow$
$Happens(e, t_1) \wedge Initiates(e, f_1, t_1) \wedge Trajectory(f_1, t_1, f_2, t_2) \wedge \neg StoppedIn(t_1, f_1, t_2)$

BEC4. $HoldsAt(f, t) \leftarrow InitiallyP(f) \wedge \neg StoppedIn(0, f, t)$

BEC5. $\neg HoldsAt(f, t) \leftarrow InitiallyN(f) \wedge \neg StartedIn(0, f, t)$

BEC6. $HoldsAt(f, t_2) \leftarrow$
$Happens(e, t_1) \wedge Initiates(e, f, t_1) \wedge t_1 < t_2 \wedge \neg StoppedIn(t_1, f, t_2)$

BEC7. $\neg HoldsAt(f, t_2) \leftarrow$
$Happens(e, t_1) \wedge Terminates(e, f, t_1) \wedge t_1 < t_2 \wedge \neg StartedIn(t_1, f, t_2)$

ANSWER SET PROGRAMMING & THE s(CASP) SYSTEM

- ASP is an extension of logic programming that supports negation-as-failure
  - Based on stable model semantics (negation is sound)
  - Possible world semantics
  - Various implementations available based on use of SAT-solver technology
- s(CASP) system
  - Current ASP systems have limitations
    - programs must be grounded due to use of SAT solvers
  - Current ASP systems cannot model time faithfully
    - no support for constraints over reals
  - s(CASP) system solves these problems by devising a goal-directed execution strategy
    - Based on coinduction, support for constructive negation, and other innovations
- The event calculus can be elegantly coded in Answer Set Prolog and executed on s(CASP)
  - The encoding can be used for checking for feasibility and consistency by executing appropriate queries against this encoding
  - Additional knowledge coded in ASP can be used for checking for completeness

EXAMPLE ENCODING: CONSISTENCY & FEASIBILITY OF REQUIREMENTS

- Modeling requirements using the event calculus (EC):

While the aircraft is on-ground, **when** the requested door position becomes open, the door control system **shall** change the state of the cargo door from closed to open **within** 10 seconds.

```
% Declare fluents for respective object states
% We use parametrized state fluents to simplify
% the encoding
fluent(door_requested_position(RP)).
fluent(door_state(DS)).
fluent(aircraft_state(AS)).

% Declare `happens' predicate to reflect the change
% of state trigger over the duration of the event
happens(pilot_requests_door_to_open, T1, T2) :-
      not holdsAt(door_requested_position(open), T1),
      holdsAt(door_requested_position(open), T2),
      T2 .=<. T1 + 10.

% Declare `initiates' and `terminates' predicates that
% map the changes influenced by the system response
% over the duration of the event

initiates(pilot_requests_door_to_open,
          door_state(open), T) :-
             holdsAt(aircraft_state(is_on_ground), T).

terminates(pilot_requests_door_to_open,
        door_state(closed),T):-
            holdsAt(aircraft_state(on_ground), T).

% A typical query for verifying a property (door
% operation behaves correctly) is shown below.

door_response_is_correct_condition:-
    not holdsAt(door_requested_position(open), TB),
    not holdsAt(door_state(open), TB),
    TE .>. TB, TE .-<. TB+ 10,
    TH .>=. TB,
    holdsAt(aircraft_state(on_ground),TH),
    holdsAt(door_state(open), TE).
```

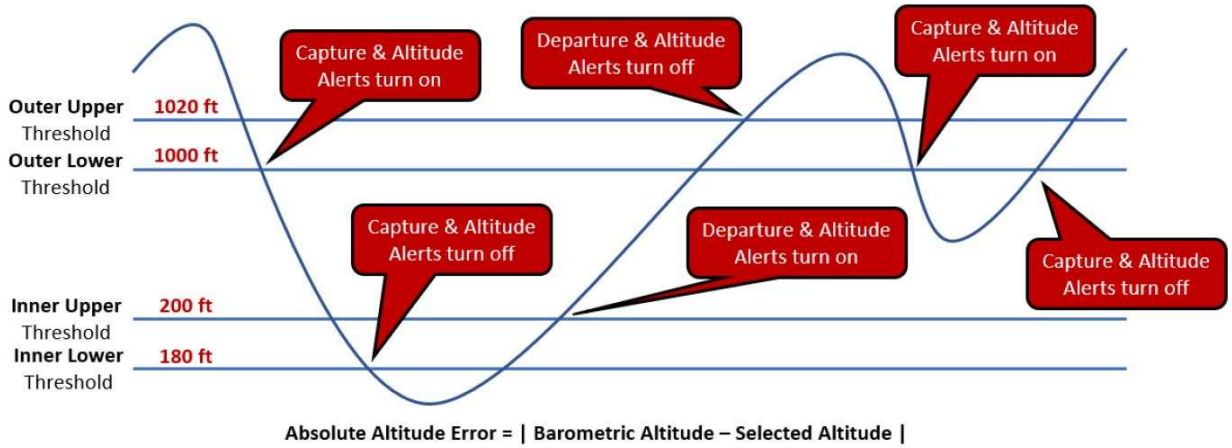
REASONING INTENT : CONSISTENCY & FEASIBILITY OF REQUIREMENTS

While the door is opening, the rate of change in door position shall increase positively with a maximum rate of 5 degrees per second.

```
trajectory(door_opening,T1,door_position(B),T2) :-
        holdsat(door_position(A),T1),
        holdsat(door_position(B),T2),
        B - A .>. 0,
        B - A .=<. 5,
        T2 - T1 .=. 1.

% Constraint on rate at which door opens
% is true throughout:
door_rate_ok :-
    trajectory(door_opening,T1,door_position(P),T2),
    holdsat(door_opening,TH), TH .>=. T1, TH .=<. T2.
```

ALTITUDE ALERTING SYSTEM



- Fluents: Absolute Altitude Error, Capture Alert On/Off, Departure Alert On/Off, Altitude Alert On/Off, Altitude Adjusting/Not Adjusting
- Events: Error Becomes > 200 ft, Error Becomes < 1020 ft, Init Capture Alert, Init Departure Alert, Init Altitude Alert, Term All Alerts

CONSISTENCY & FEASIBILITY OF REQUIREMENTS

REQ1: *When the **absolute_altitude_error** becomes less than 1000 ft, the Altitude Alerting System shall initiate the **capture_alert** and **altitude_alert** within 1 second.*

```
happens(init_capture_alert, T1, T2) :-
    holds_ae_becomes_lt(1000, T1),
    T2 .=<. T1 + 1.
holds_ae_becomes_lt(1000, T) :-
    holdsAt(altitude_error(V1), T1),
    holdsAt(altitude_error(V2), T2),
    V1 .>=. 1000, V2 .<. 1000,
    infimum(T2, T).
```

REQ2: *When the **absolute_altitude_error** becomes greater than 200 ft, the Altitude Alerting System shall initiate the **departure_alert** and **altitude_alert** within 1 second.*

```
happens(init_departure_alert, T1, T2) :-
   holds_ae_becomes_gt(200, T1),
   T2 .=<. T1 + 1.
```

REQ3: *When the **absolute_altitude_error** becomes either less than 180 ft or greater than 1020 ft, the Altitude Alerting System shall terminate all **altitude_alerts** within 1 second.*

```
happens(term_all_alerts, T1, T2) :-
  holds_ae_becomes_lt_or_gt(180,1020,T1),
  T2 .=<. T1 + 1.
holds_ae_becomes_lt_or_gt(180, _, T) :-
  holds_ae_becomes_lt(180, T).
holds_ae_becomes_lt_or_gt(_, 1020, T) :-
  holds_ae_becomes_gt(1020, T).
```

REQ4: *While the **altitude_selection_knob** has not been '**adjusting**' within the previous 5 seconds, the Altitude Alerting System shall determine **absolute_altitude_error** as the absolute difference between the **selected_altitude** and the **barometric_altitude**, otherwise **absolute_altitude_error** shall be 0.*

```
trajectory(altitude_adjusting, T1,
    altitude_error(0), T2) :-
    T2 .=< 5, T2 .>. 0.
trajectory(altitude_not_adjusting_5sec, T1,
    altitude_error(E), T2) :-
    T2 .>. 0,
    holdsAt(barometric_altitude(B), T2),
    holdsAt(selected_altitude_value(V), T2),
    absolute_difference(B, V, E).
```

REQ5: *Upon initialization, the Altitude Alerting System shall consider the **altitude_selection_knob** to be '**adjusting**'.*

```
initiallyP(altitude_adjusting).
initiates(reset, altitude_adjusting, T).
```

COMPLETENESS OF REQUIREMENTS: SINGLE EVENT UPSETS

- Single event upset can cause system reset; they are easy to miss and maybe counterintuitive
  - The destructive power of SEU modeled using a reset event
  - The reset event overrides the constructed internal state of the system, *forcing the initialization state to be re-established*
  - Such a reset tests robustness of the system initialization logic
  - Modeled as an ***abducible*** (reset may or may not happen) in ASP
  - Reset event can only override the cyber system software state and does not affect the continuous state of the physical world
  - SEU terminates every fluent and initiates it to an initial value

```
terminates(reset, Fluent, T) :-
        internal_state(Fluent).
initiates(reset, Default, T) :-
        default(Fluent, Default),
        internal_state(Fluent).
```

Query	s(CASP)	CLINGO
Query Q1 (Normal Run)	6 min 13 sec	> 50 min
Query Q2 (Reset, REQ5)	9 min 14 sec	> 50 min
Query Q3 (Reset, REQ5')	46 min 55 sec	> 50 min

Table I
QUERY EXECUTION TIMES

  - If designer fails to take into account that the altitude knob may be adjusting during SEU, we can detect that (e.g., check altitude_alert turned on after 5 seconds)

```
?- happens(reset, T, T),T1 .>=. T + 5,altitude_trajectory(T1), holdsAt(altitude_alert_on, T1).
```

FAILING RUN IN THE PRESENCE OF SEU

- If we change the assumption in REQ5, then we can show a failing run of the alerting system
- REQ5' : Upon initialization/reset, the altitude alerting system shall consider the altitude selection knob to be not adjusting (incorrect assumption)
- Scenario:
  - Plane is cruising at 32,000 ft at time *t*
  - Selected altitude value is 32, 300 ft, altitude not adjusting since *t – 5*
  - By REQ3, "*When altitude error goes above 300 ft, the altitude alert should be turned on*"
  - Under normal circumstances, the system issues the altitude alert
  - Consider a reset at time *t,* which resets the internal state by turning the altitude alert off
  - If REQ5' is used then, the altitude alert never turns back on
  - If REQ5 is used then, the altitude alert turns back on at *t+5*
  - The query shown before checks for erroneous behaviour of alerting with SEU

    ?- happens(reset, T, T),T1 .>=. T + 5, altitude_trajectory(T1), holdsAt(altitude_alert_on, T1).

FUTURE WORK

- Building out MIDAS IDE within MBSE Tool Chain
- Working classes of knowledge to aid application domain knowledge
  - E.g., Defeater Knowledge
    - How systems fail
    - Known specification gaps
  - Architectural & Distributed System Reasoning
    - E.g., Byzantine Vulnerability Detection
- Improving s(CASP) to search more efficiently
  - Refine the s(CASP) algorithm
    - Localize consistency checks to make the search more efficient
    - Generate efficient dual rules for Event Calculus encodings