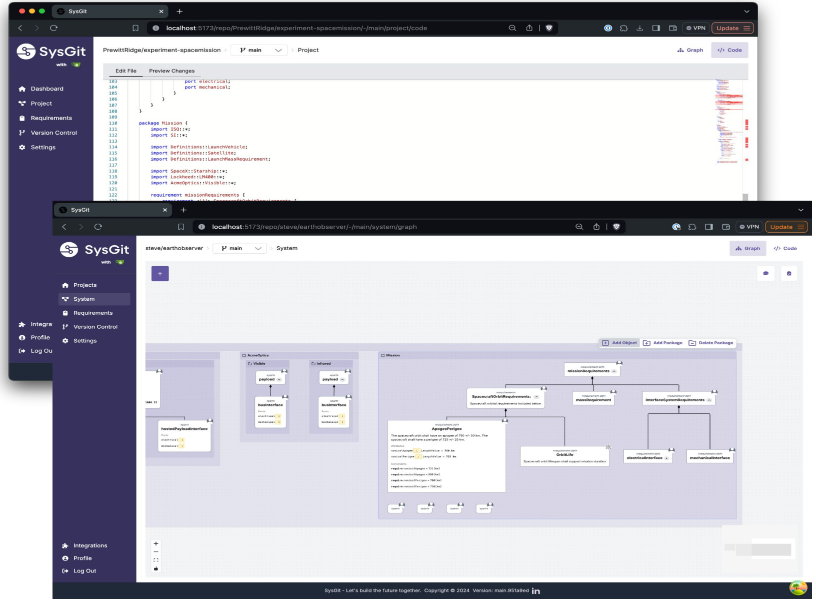
SysGit

Agile Engineering enabled by Agentic Co-Modelers

Hart Traveller, ML Engineer, SysGit

Zeke Brechtel, CTO, SysGit

Steve Massey, CEO, SysGit



```
                        port electrical;
                        port mechanical;
                    }
                }
            }
        }

        package Mission {
            import ISQ::*;
            import SI::*;

            import Definitions::LaunchVehicle;
            import Definitions::Satellite;
            import Definitions::LaunchMassRequirement;

            import SpaceX::Starship::*;
            import Lockheed::LM400::*;
            import AcmeOptics::Visible::*;

            requirement missionRequirements {
```

Authoring: Manage Requirements & Verifications, create MBSE models, generate trace matrices, and track changes.

Infrastructure: Interact with Requirements and System Models throughout the rest of your technology stack using our CI/CD mode, our Python ORM,
or the SysML v2 API.

Automation: Fully offline AI capability accelerates the processing of engineering artifacts into shareable System Models.

OMG Member

OMG Standards Development Organization.

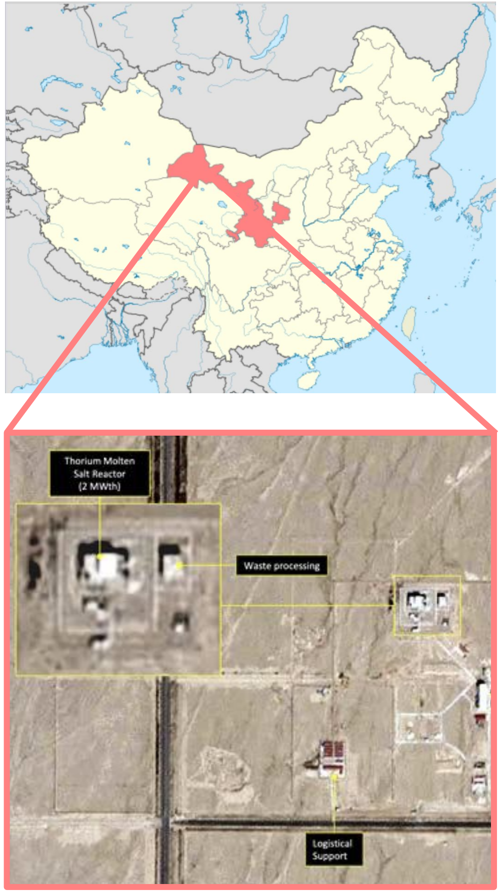
TREAT HARDWARE AS CODE

hello@sysgit.io | https://sysgit.io | https://prewittridge.com

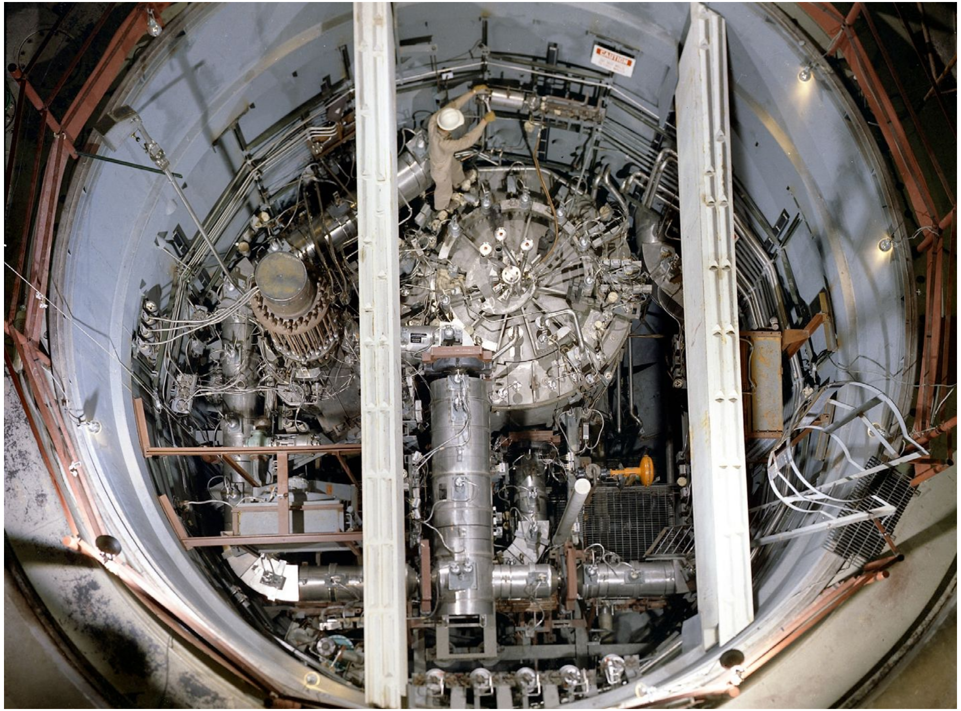
SysGit

TMSR-LF1 project

2011	**SINAP establishes TMSR center**
2018	Construction begins on a 2 thermal megawatt prototype near Wuwei
2021	Construction completes three years ahead of schedule
2023	**Criticality reached**
2024	The reactor reaches full power in sustained operation
2025	Chinese scientists accomplished first online refueling
2030	**60 MWt (10 MWe) reactor planned**



Origins at Oak Ridge

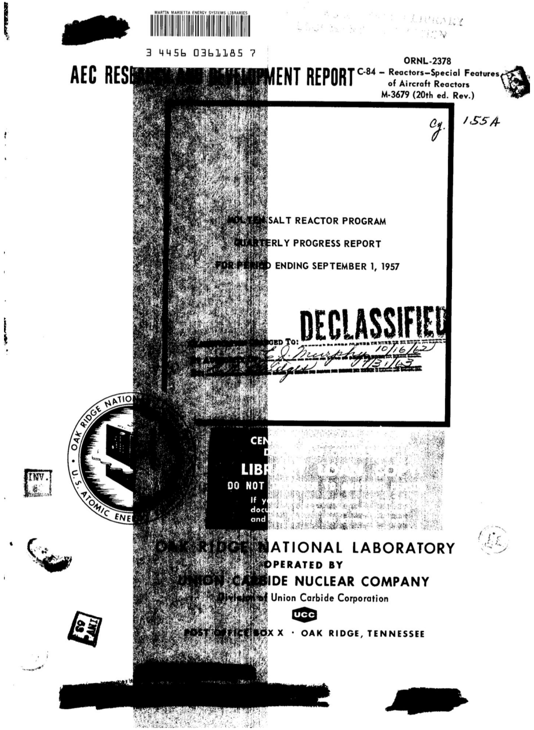


More recent view of the reactor experiment

The original reactor worked.

MSRE at ORNL in Aircraft Reactor Experiment building

Documentation is plentiful.



[redacted]

MARTIN MARIETTA ENERGY SYSTEMS LIBRARIES

3 4456 0361185 7

AEC RESEARCH AND DEVELOPMENT REPORT

ORNL-2378
C-84 – Reactors–Special Features of Aircraft Reactors
M-3679 (20th ed. Rev.)

Cy. 155A

MOLTEN SALT REACTOR PROGRAM
QUARTERLY PROGRESS REPORT
FOR PERIOD ENDING SEPTEMBER 1, 1957

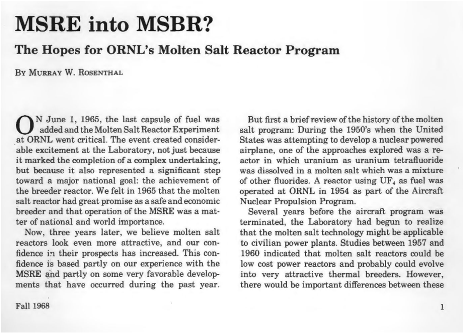
DECLASSIFIED

CLASSIFICATION CHANGED TO:
BY AUTHORITY OF: E.J. Murphy 10/16/62
BY: [illegible] Edges 7/31/63

OAK RIDGE NATIONAL LABORATORY
OPERATED BY
UNION CARBIDE NUCLEAR COMPANY
Division of Union Carbide Corporation
UCC
POST OFFICE BOX X · OAK RIDGE, TENNESSEE

[redacted]

[redacted]



MSRE into MSBR?

The Hopes for ORNL's Molten Salt Reactor Program

By Murray W. Rosenthal

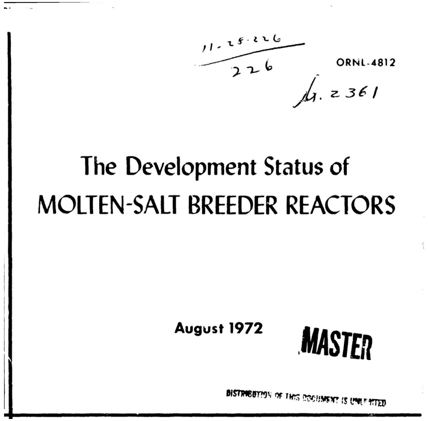
On June 1, 1965, the last capsule of fuel was added and the Molten Salt Reactor Experiment at ORNL went critical. The event created considerable excitement at the Laboratory, not just because it marked the completion of a complex undertaking, but because it also represented a significant step toward a major national goal: the achievement of the breeder reactor. We felt in 1965 that the molten salt reactor had great promise as a safe and economic breeder and that operation of the MSRE was a matter of national and world importance.

Now, three years later, we believe molten salt reactors look even more attractive, and our confidence in their prospects has increased. This confidence is based partly on our experience with the MSRE and partly on some very favorable developments that have occurred during the past year.

But first a brief review of the history of the molten salt program: During the 1950's when the United States was attempting to develop a nuclear powered airplane, one of the approaches explored was a reactor in which uranium as uranium tetrafluoride was dissolved in a molten salt which was a mixture of other fluorides. A reactor using UF_4 as fuel was operated at ORNL in 1954 as part of the Aircraft Nuclear Propulsion Program.

Several years before the aircraft program was terminated, the Laboratory had begun to realize that the molten salt technology might be applicable to civilian power plants. Studies between 1957 and 1960 indicated that molten salt reactors could be low cost power reactors and probably could evolve into very attractive thermal breeders. However, there would be important differences between these

Fall 1968 1



11-25-226
226

ORNL-4812

Ar. 2361

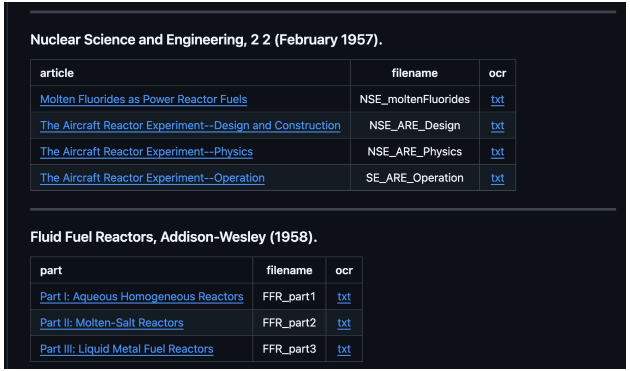
The Development Status of
MOLTEN-SALT BREEDER REACTORS

August 1972

MASTER

DISTRIBUTION OF THIS DOCUMENT IS UNLIMITED

The openmsr project specifically maintains repositories with extensive documentation, CAD models, and simulation code.



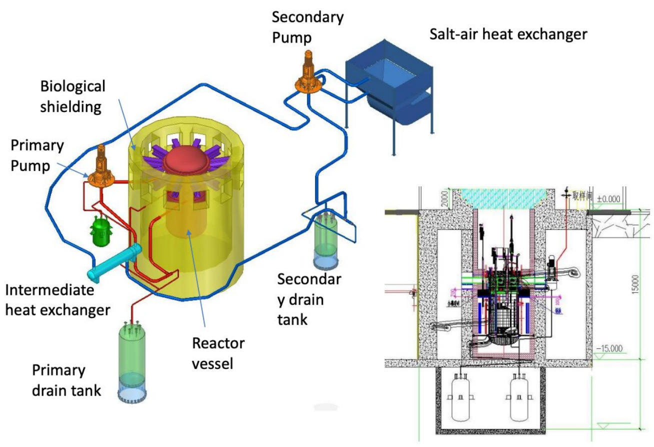
Nuclear Science and Engineering, 2 2 (February 1957).

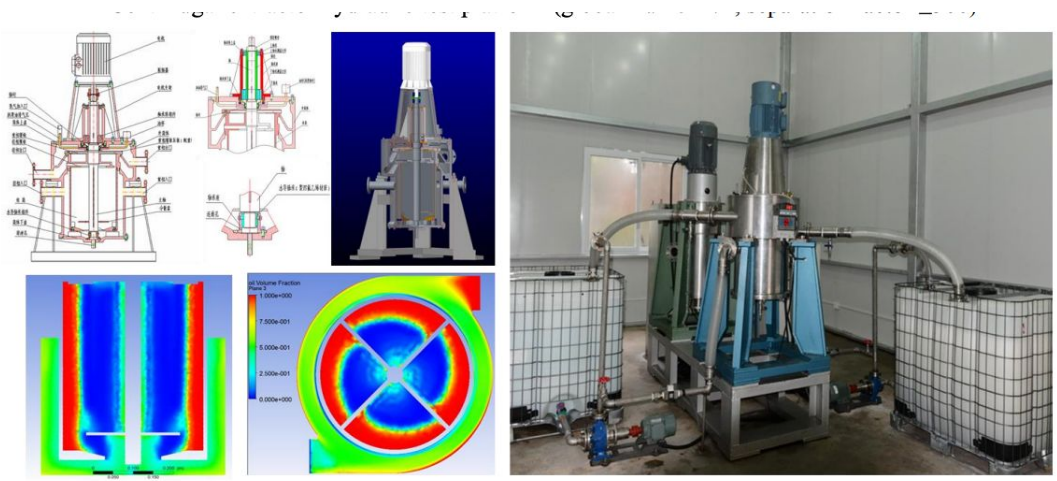
article	filename	ocr
Molten Fluorides as Power Reactor Fuels	NSE_moltenFluorides	txt
The Aircraft Reactor Experiment--Design and Construction	NSE_ARE_Design	txt
The Aircraft Reactor Experiment--Physics	NSE_ARE_Physics	txt
The Aircraft Reactor Experiment--Operation	SE_ARE_Operation	txt

Fluid Fuel Reactors, Addison-Wesley (1958).

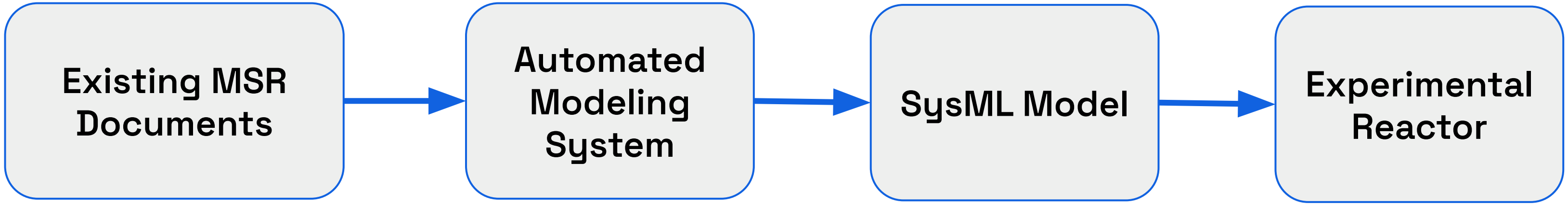
part	filename	ocr
Part I: Aqueous Homogeneous Reactors	FFR_part1	txt
Part II: Molten-Salt Reactors	FFR_part2	txt
Part III: Liquid Metal Fuel Reactors	FFR_part3	txt

Modern teams recreated the reactor with that documentation.





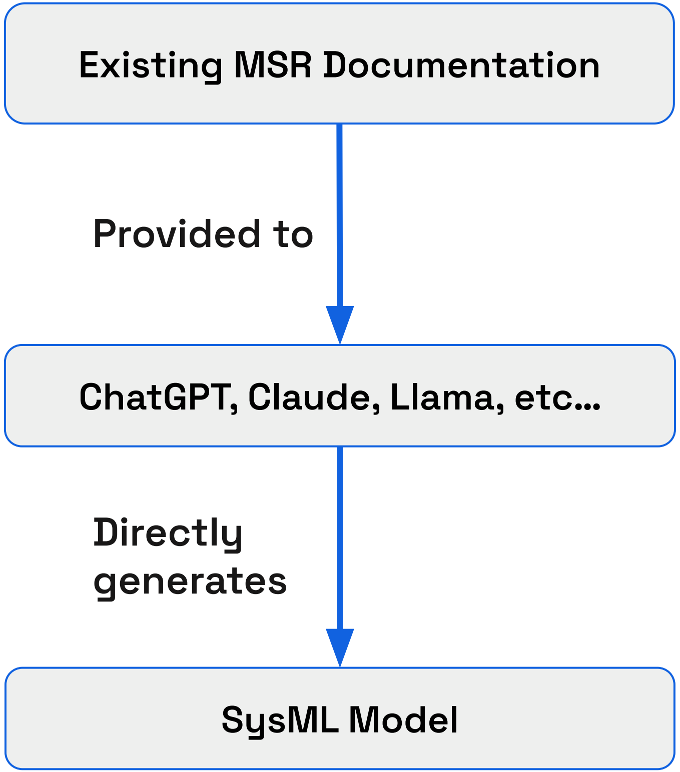
A concrete objective



Success criteria for reactor:

1. Turns on.
2. Provides power.
3. Doesn’t blow up.

One possible approach



In this case, we're implicitly including LLMs that do/do not have access to SysML documentation, and LLMs that are/aren't trained / fine tuned on SysML textual notation.

Syntactic and semantic quality metric details

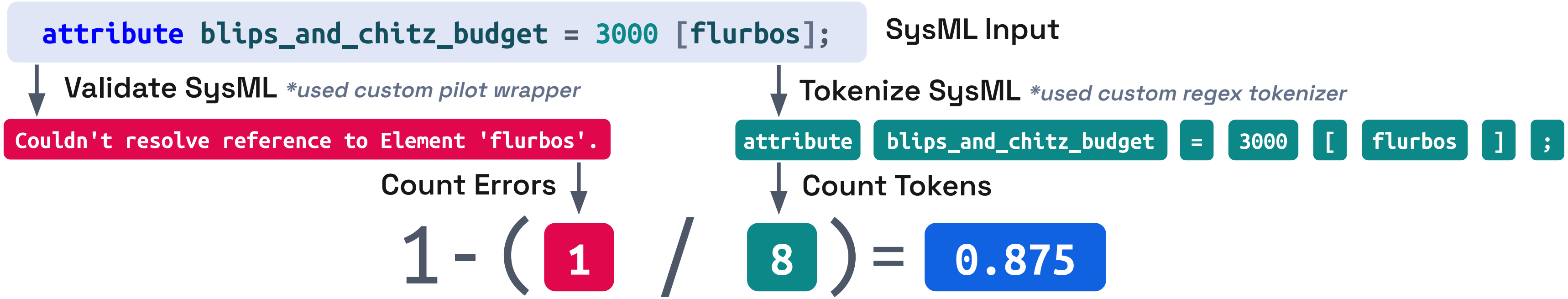
SUMMARY A continuous value between 0 and 1 that validity of syntax / semantics in a SysML model.[1][2]

INTERPRETATION Higher is better. 1 means there are no errors. Less than 1 means the textual notation is invalid, but the lower the value the worse. This is roughly akin to the percentage

Calculation

Subtract P(Error | Token)[3] from 1.

$$\text{float SSQ} = 1 - \left(\underset{\text{Number of Errors}}{\text{int } n_e} \Big/ \underset{\text{Number of Tokens}}{\text{int } n_t} \right)$$



SysML Input

```
attribute blips_and_chitz_budget = 3000 [flurbos];
```

Validate SysML **used custom pilot wrapper*

```
Couldn't resolve reference to Element 'flurbos'.
```

Count Errors

Tokenize SysML **used custom regex tokenizer*

```
attribute | blips_and_chitz_budget | = | 3000 | [ | flurbos | ] | ;
```

Count Tokens

$$1 - \left(\boxed{1} \ / \ \boxed{8} \right) = \boxed{0.875}$$

[1] This metric isn’t a ’percent valid’ metric - the minimum value isn’t necessarily 0, but go below 0 there would need to be more errors than tokens.
[2] The metric is continuous to help rank different strategies, and to function as a feedback signal.
[3] The error count is divided by the token count to normalize the error rate; this accounts for varied sysml input / translator output lengths.

Syntactic and semantic validity metric validity

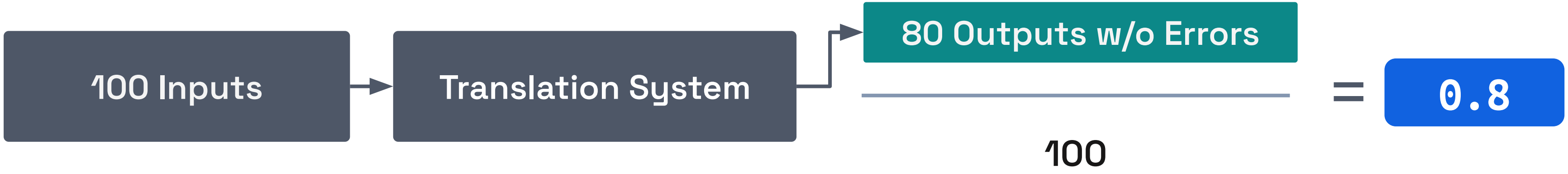
[computed across *n* sample outputs]

SUMMARY A value between 0 and 1 that measures the percent of outputs without syntax errors.

INTERPRETATION Higher is better. This metric is akin the probability that the output code generated is syntactically valid.

RATIONALE No errors is a soft requirement for code in production environments; SSV < 1 should disqualify a generation system.

CALCULATION The average floor of the S_q metric for each output in the sample.



Syntactic and semantic quality and validity

Quality = 1 - (ne / nt) = 1 - (0 / 35) = 1.0

```
private import SI::*;
private import ISQ::*;
requirement 'LRO Radiometric Doppler Measurement Accuracy Requirement' {
   doc /*
   LRO and its GDS shall achieve a radiometric doppler
   measurement accuracy of less than 1 mm/sec.
   */
   attribute measurementAccuracy : LengthValue;
   require constraint { measurementAccuracy < 0.001 [m/s] }
}
```

Quality = 1 - (ne / nt) = 1 - (3 / 22) = 0.864

```
requirement def LRODirectLunarTransfer {
   doc "The LRO shall utilize a direct lunar transfer trajectory."
   subject LRO : Lander;
   require constraint { LRO.transferTrajectory == 'Direct Lunar Transfer' }
}
```

```
2:5 no viable alternative at input 'doc'
```

```
2:9 mismatched input '"The LRO shall utilize a direct lunar transfer trajectory."' expecting RULE_REGULAR_COMMENT
```

```
4:5 missing EOF at 'require'
```

Quality = 1 - (ne / nt) = 1 - (38 / 86) = 0.558

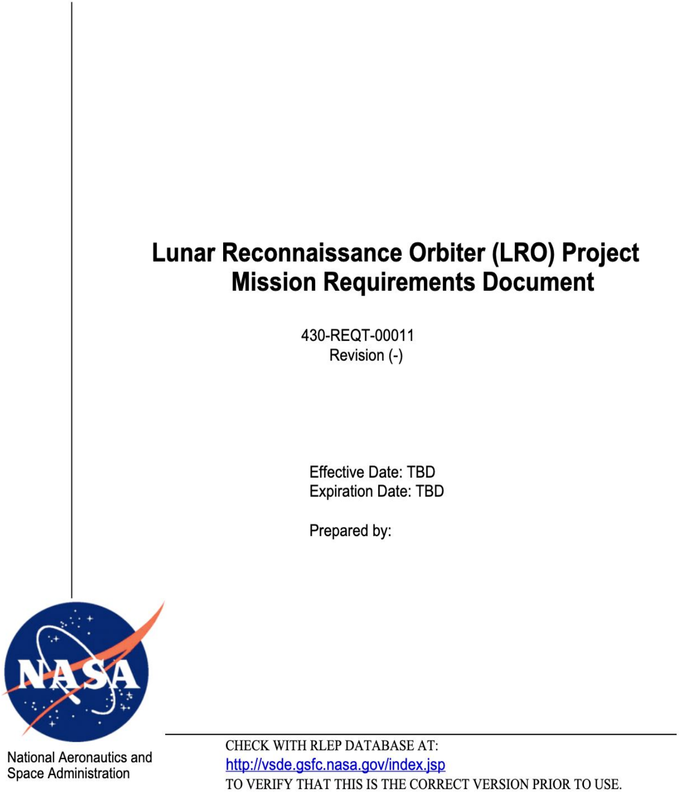
```
function LRODesign(orientationToSun: Orientation) : Design
   function SunOrientedMode() : Mode
       function CalculateSunDirection() : Vector3D
           // Code to calculate the sun direction based on time and location
       end
       function RotateLRO(direction: Vector3D) : Design
           // Code to rotate LRO towards the calculated sun direction
       end
       var sunDirection = CalculateSunDirection()
       return RotateLRO(sunDirection)
   end
   var modes = {NormalMode(), SunOrientedMode()}
   // Code to determine if current inertial position allows for sun orientation
   function FindSunOrientedMode() : Mode
   end
   if (FindSunOrientedMode())
       modes.add(SunOrientedMode())
   return Design(modes)
end
```

Total of 38 errors...

Validity = 1 / 3 = 0.33...

LRO syntax and semantics input data

Example Requirements



Lunar Reconnaissance Orbiter (LRO) Project
Mission Requirements Document

430-REQT-00011
Revision (-)

Effective Date: TBD
Expiration Date: TBD

Prepared by:

National Aeronautics and
Space Administration

CHECK WITH RLEP DATABASE AT:
http://vsde.gsfc.nasa.gov/index.jsp
TO VERIFY THAT THIS IS THE CORRECT VERSION PRIOR TO USE.

LRO's launch mass shall not exceed 1480 kg.

LRO shall fit within a 9.5 ft (diameter) fairing.

The orbit inclination shall be 90 degrees +/- 1 degree.

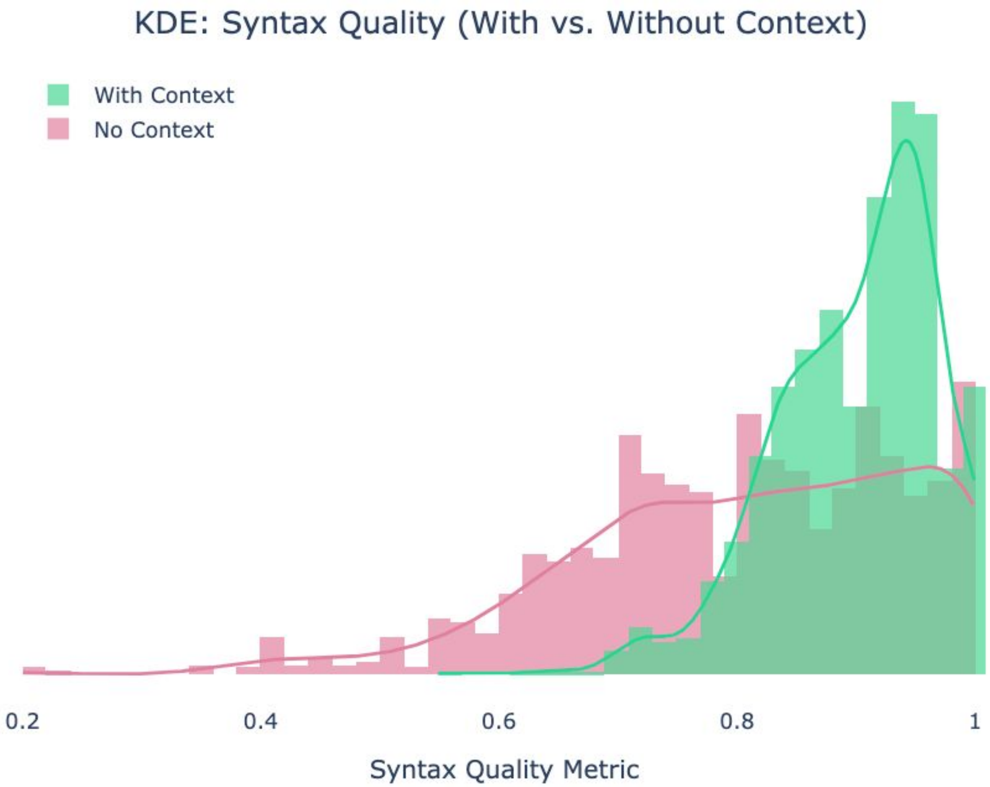
LRO shall be designed to have a minimum mission duration of 14 months.

The LV shall despin LRO to a rate < 2 rpm.

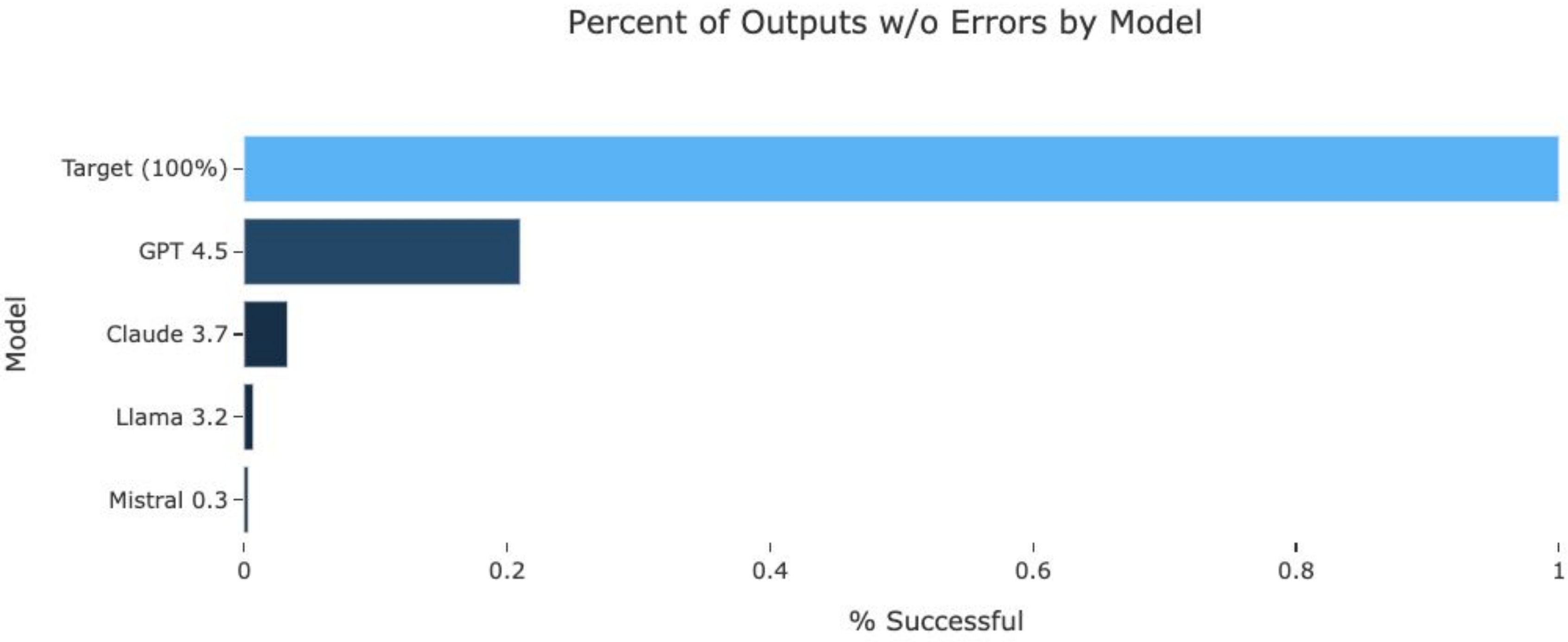
The launch vehicle must be capable of delivering a 1480 kg payload to a trajectory with a C3 > -1.85.

LRO and its GDS shall achieve a radiometric doppler measurement accuracy of less than 1 mm/sec.

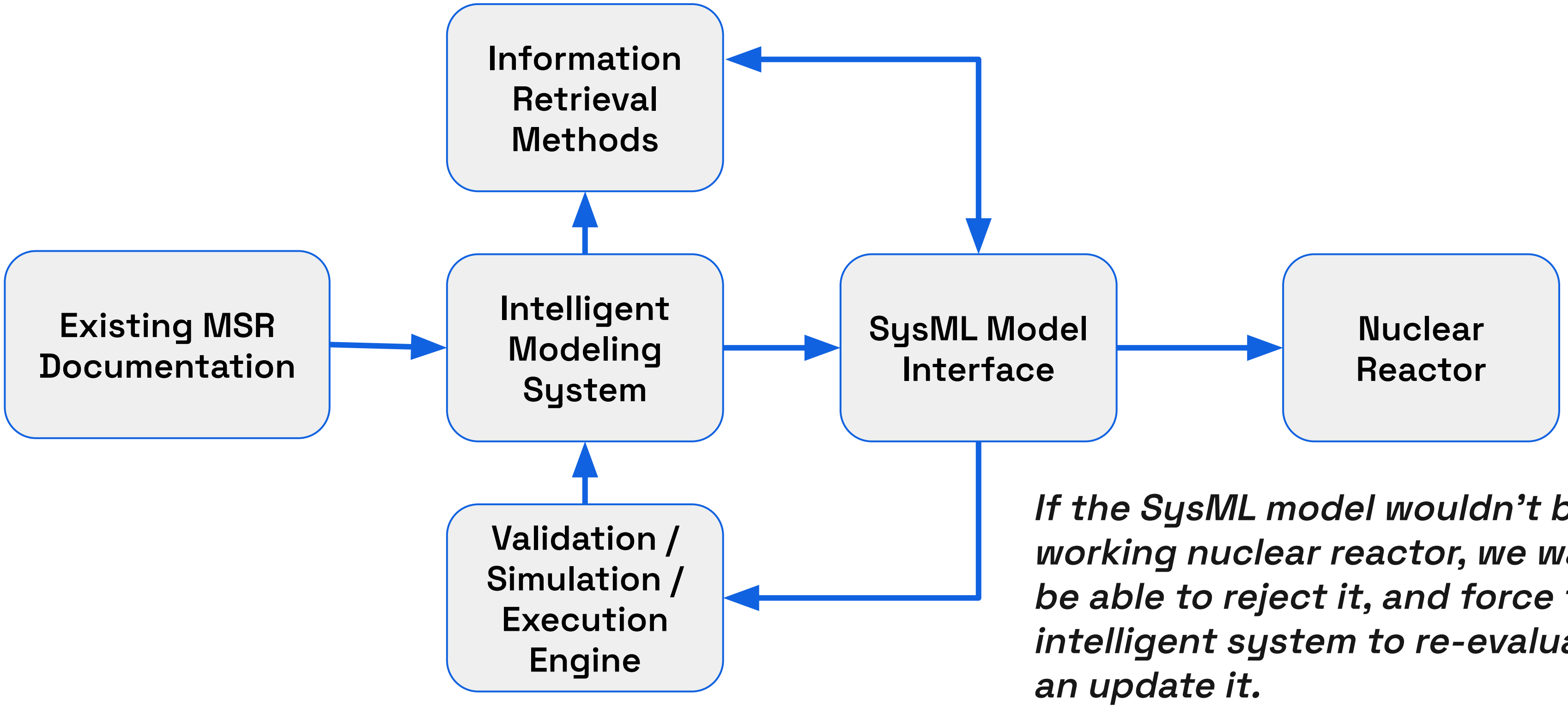
Quality metric result



Validity metric result

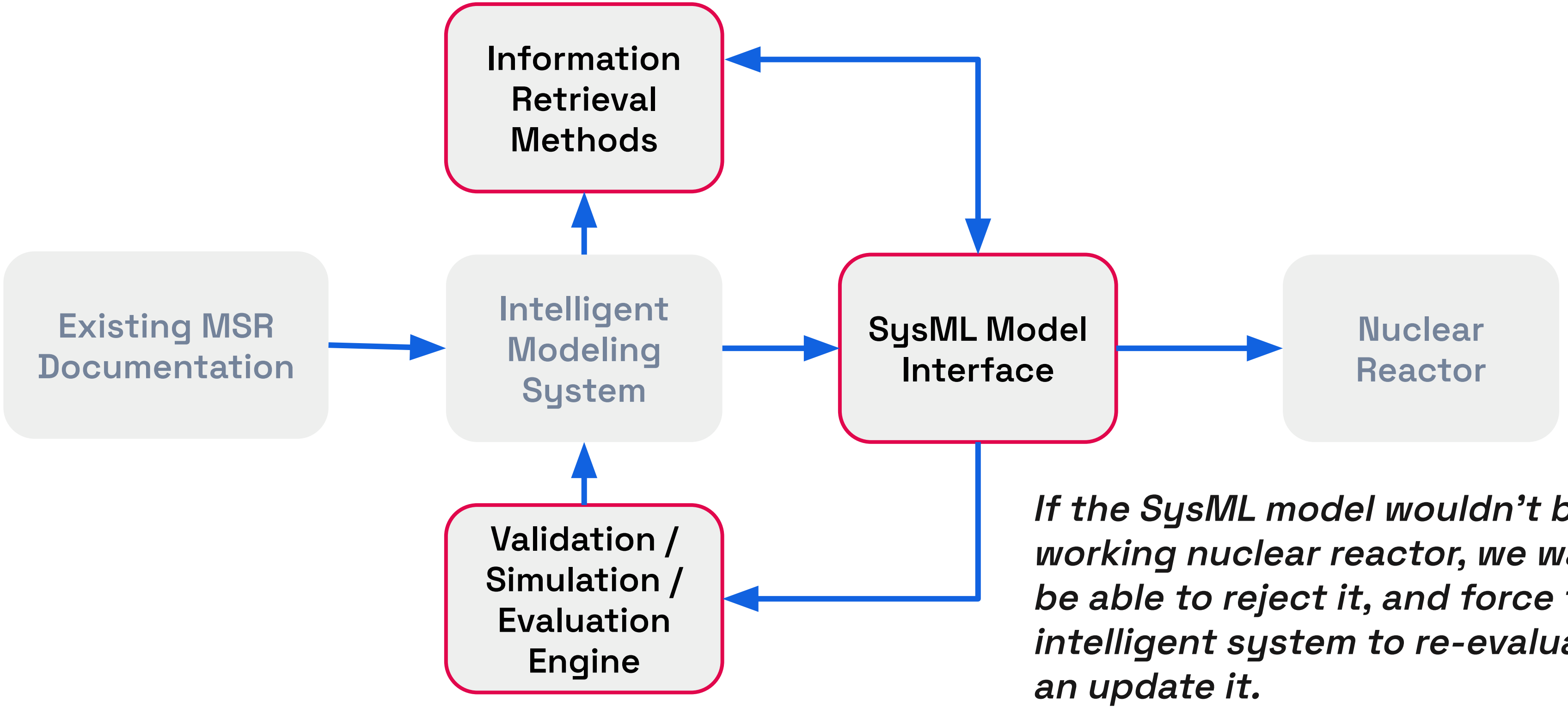


A refined approach


Information Retrieval Methods
Existing MSR Documentation
Intelligent Modeling System
SysML Model Interface
Nuclear Reactor
Validation / Simulation / Execution Engine

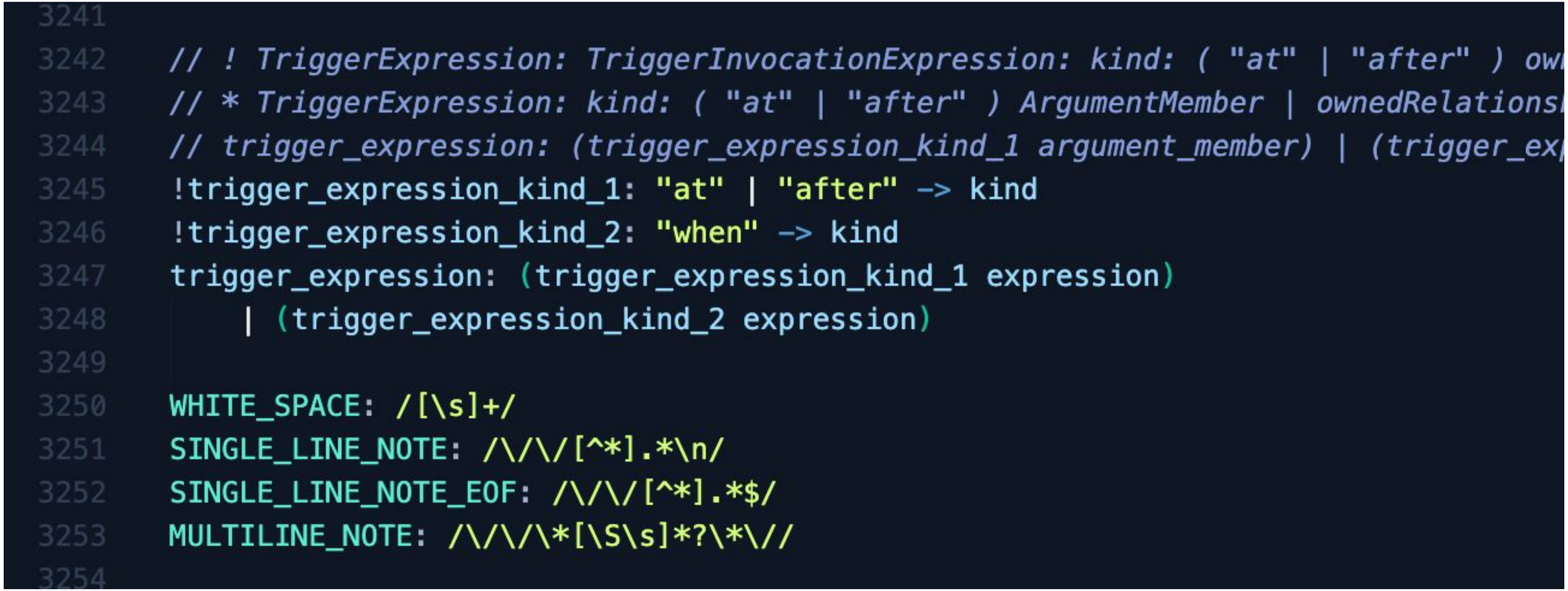
If the SysML model wouldn’t be a working nuclear reactor, we want to be able to reject it, and force the intelligent system to re-evaluate it an update it.

Our primary focus


Information Retrieval Methods
Existing MSR Documentation
Intelligent Modeling System
SysML Model Interface
Nuclear Reactor
Validation / Simulation / Evaluation Engine

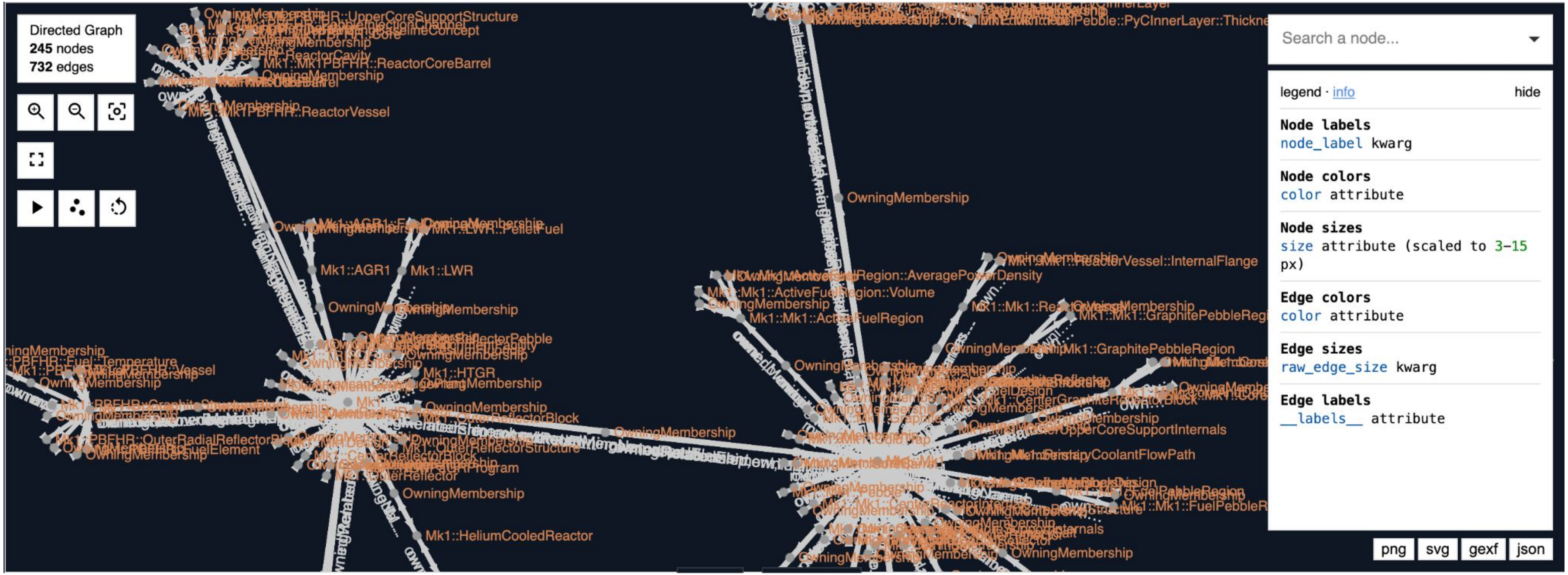
If the SysML model wouldn’t be a working nuclear reactor, we want to be able to reject it, and force the intelligent system to re-evaluate it an update it.

Grammar for parser



```
// ! TriggerExpression: TriggerInvocationExpression: kind: ( "at" | "after" ) ow
// * TriggerExpression: kind: ( "at" | "after" ) ArgumentMember | ownedRelations
// trigger_expression: (trigger_expression_kind_1 argument_member) | (trigger_ex
!trigger_expression_kind_1: "at" | "after" -> kind
!trigger_expression_kind_2: "when" -> kind
trigger_expression: (trigger_expression_kind_1 expression)
    | (trigger_expression_kind_2 expression)

WHITE_SPACE: /[\s]+/
SINGLE_LINE_NOTE: /\/\/[^*].*\n/
SINGLE_LINE_NOTE_EOF: /\/\/[^*].*$/
MULTILINE_NOTE: /\/\/\*[\S\s]*?\*\//
```

Intermediary graph representation



Directed Graph
245 nodes
732 edges

legend · info hide

Node labels
node_label kwarg

Node colors
color attribute

Node sizes
size attribute (scaled to 3–15 px)

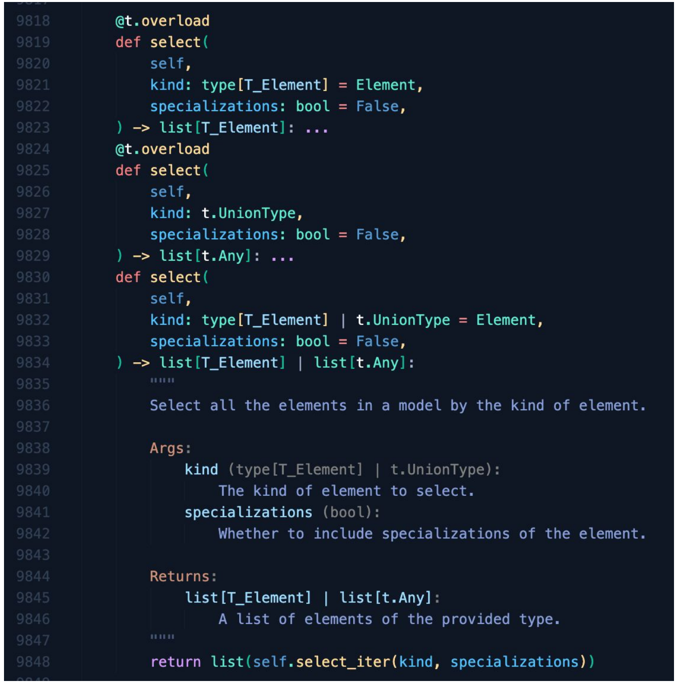
Edge colors
color attribute

Edge sizes
raw_edge_size kwarg

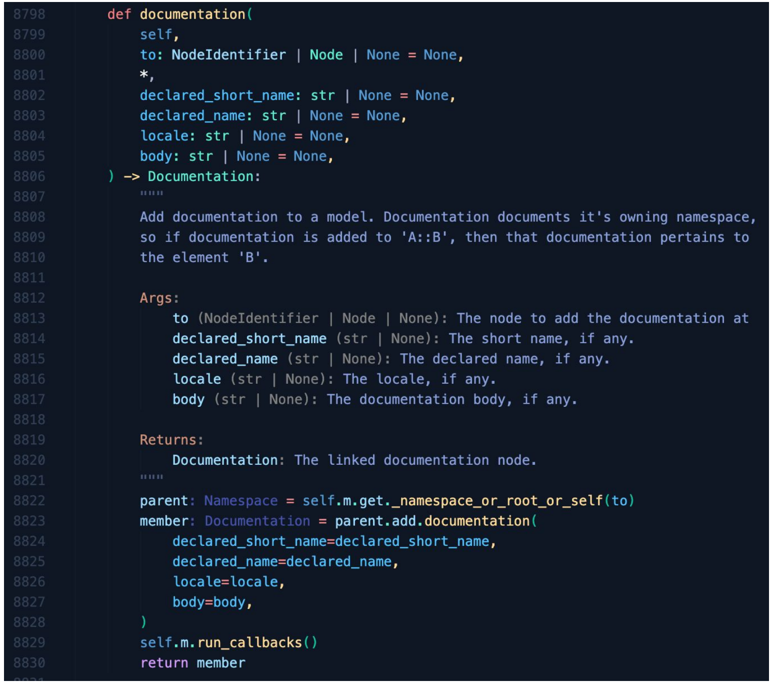
Edge labels
__labels__ attribute

Search a node...
png
svg
gexf
json
OwningMembership
Mk1::MK1PBFHR::ReactorCoreBarrel
Mk1::MK1PBFHR::ReactorVessel
Mk1::AGR1
Mk1::LWR
Mk1::LWR::PelletFuel
Mk1::HTGR
Mk1::HeliumCooledReactor
Mk1::OuterReflectorBlock
Mk1::OuterReflectorStructure
Mk1::PBFHR::OuterRadialReflectorBlock
Mk1::Mk1::ActiveFuelRegion
Mk1::ActiveFuelRegion::Volume
ActiveFuelRegion::AveragePowerDensity
Mk1::Mk1::ReactorVessel::InternalFlange
Mk1::Mk1::GraphitePebbleRegion
Mk1::GraphitePebbleRegion
Mk1::CenterGraphiteReflectorBlock
UpperCoreSupportInternals
CoolantFlowPath
Mk1::Mk1::FuelPebbleR
UpperCoreSupportStructure
Pebble::PyCInnerLayer::Thickne

Graph/node methods for information retrieval and mutation

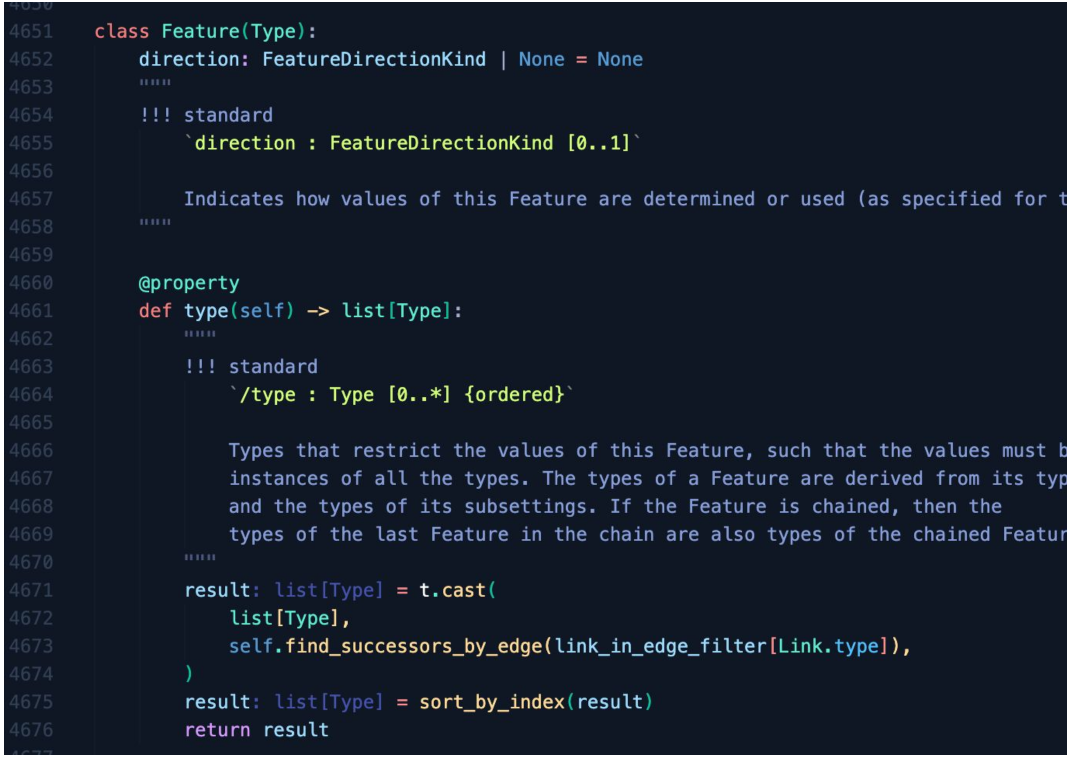


```python
    @t.overload
    def select(
        self,
        kind: type[T_Element] = Element,
        specializations: bool = False,
    ) -> list[T_Element]: ...
    @t.overload
    def select(
        self,
        kind: t.UnionType,
        specializations: bool = False,
    ) -> list[t.Any]: ...
    def select(
        self,
        kind: type[T_Element] | t.UnionType = Element,
        specializations: bool = False,
    ) -> list[T_Element] | list[t.Any]:
        """
        Select all the elements in a model by the kind of element.

        Args:
            kind (type[T_Element] | t.UnionType):
                The kind of element to select.
            specializations (bool):
                Whether to include specializations of the element.

        Returns:
            list[T_Element] | list[t.Any]:
                A list of elements of the provided type.
        """
        return list(self.select_iter(kind, specializations))
```



```python
    def documentation(
        self,
        to: NodeIdentifier | Node | None = None,
        *,
        declared_short_name: str | None = None,
        declared_name: str | None = None,
        locale: str | None = None,
        body: str | None = None,
    ) -> Documentation:
        """
        Add documentation to a model. Documentation documents it's owning namespace,
        so if documentation is added to 'A::B', then that documentation pertains to
        the element 'B'.

        Args:
            to (NodeIdentifier | Node | None): The node to add the documentation at
            declared_short_name (str | None): The short name, if any.
            declared_name (str | None): The declared name, if any.
            locale (str | None): The locale, if any.
            body (str | None): The documentation body, if any.

        Returns:
            Documentation: The linked documentation node.
        """
        parent: Namespace = self.m.get._namespace_or_root_or_self(to)
        member: Documentation = parent.add.documentation(
            declared_short_name=declared_short_name,
            declared_name=declared_name,
            locale=locale,
            body=body,
        )
        self.m.run_callbacks()
        return member
```

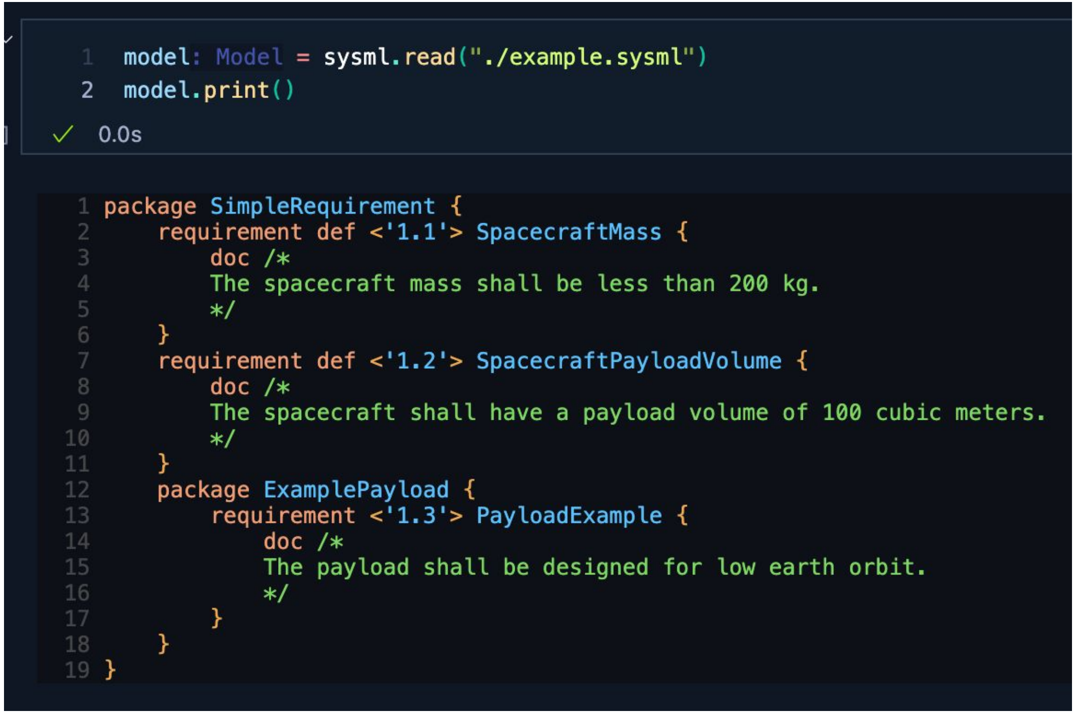
SysML standard compliant class hierarchy



```
class Feature(Type):
    direction: FeatureDirectionKind | None = None
    """
    !!! standard
        `direction : FeatureDirectionKind [0..1]`

        Indicates how values of this Feature are determined or used (as specified for t
    """

    @property
    def type(self) -> list[Type]:
        """
        !!! standard
            `/type : Type [0..*] {ordered}`

            Types that restrict the values of this Feature, such that the values must b
            instances of all the types. The types of a Feature are derived from its typ
            and the types of its subsettings. If the Feature is chained, then the
            types of the last Feature in the chain are also types of the chained Featur
        """
        result: list[Type] = t.cast(
            list[Type],
            self.find_successors_by_edge(link_in_edge_filter[Link.type]),
        )
        result: list[Type] = sort_by_index(result)
        return result
```

And why not serialize code inside nodes with references to global state?

Serialization algorithms to convert graph structure back into SysML

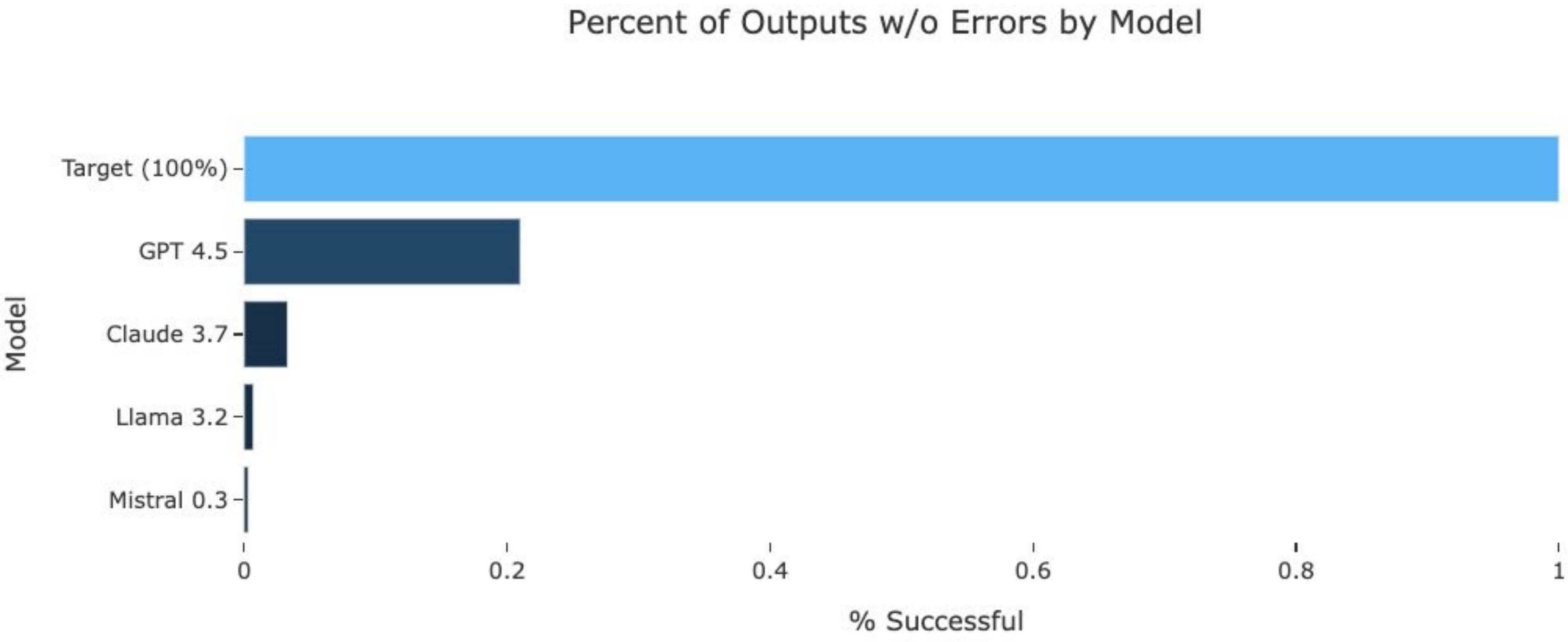


```
model: Model = sysml.read("./example.sysml")
model.print()
```

✓ 0.0s

```
package SimpleRequirement {
    requirement def <'1.1'> SpacecraftMass {
        doc /*
        The spacecraft mass shall be less than 200 kg.
        */
    }
    requirement def <'1.2'> SpacecraftPayloadVolume {
        doc /*
        The spacecraft shall have a payload volume of 100 cubic meters.
        */
    }
    package ExamplePayload {
        requirement <'1.3'> PayloadExample {
            doc /*
            The payload shall be designed for low earth orbit.
            */
        }
    }
}
```

Validity metric result



Piggyback on existing AI / scientific computing language

NumPy

LangChain

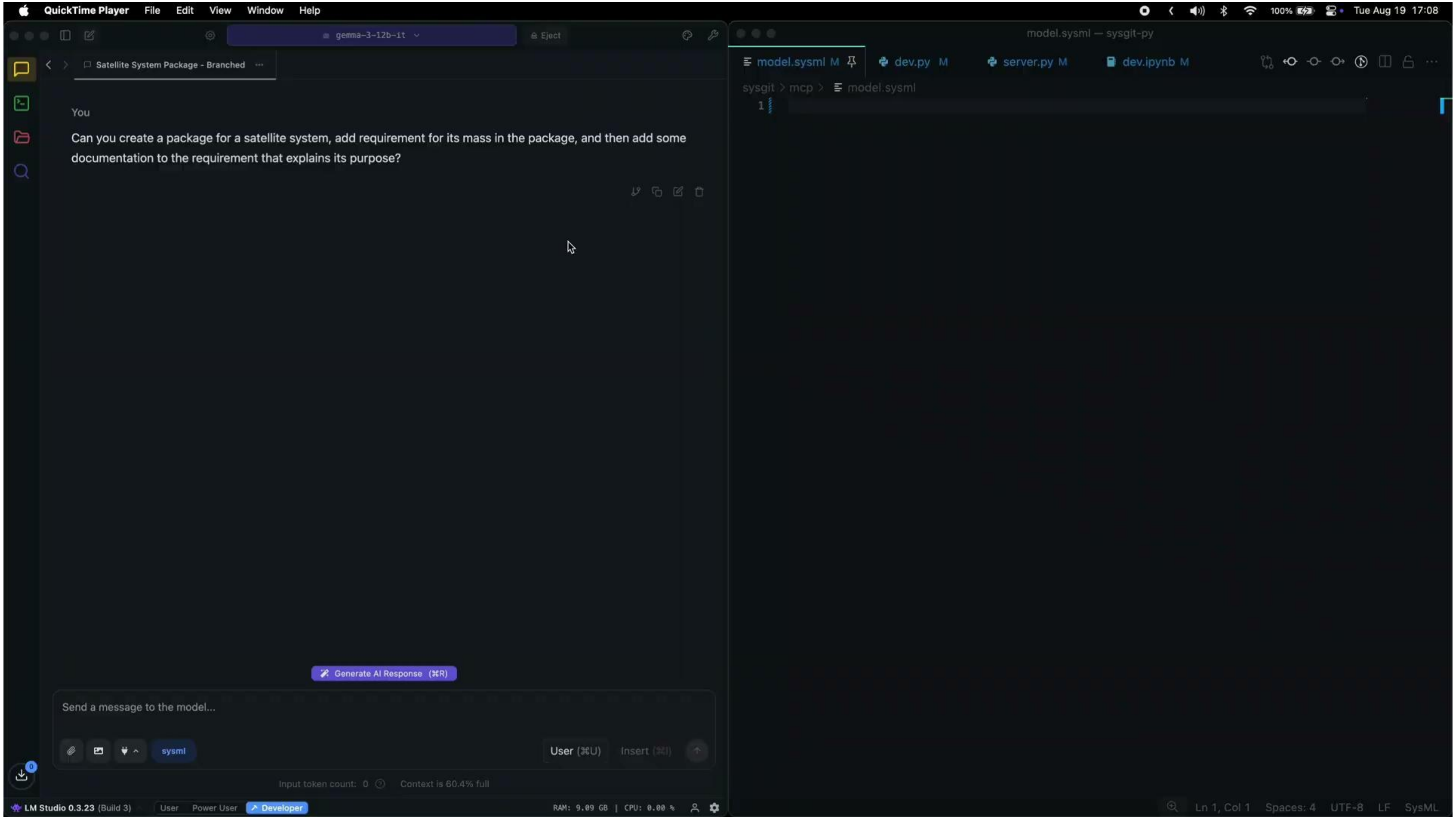
pandas

SciPy

Pydantic

SymPy

Pygments
Python syntax highlighter



You

Can you create a package for a satellite system, add requirement for its mass in the package, and then add some documentation to the requirement that explains its purpose?

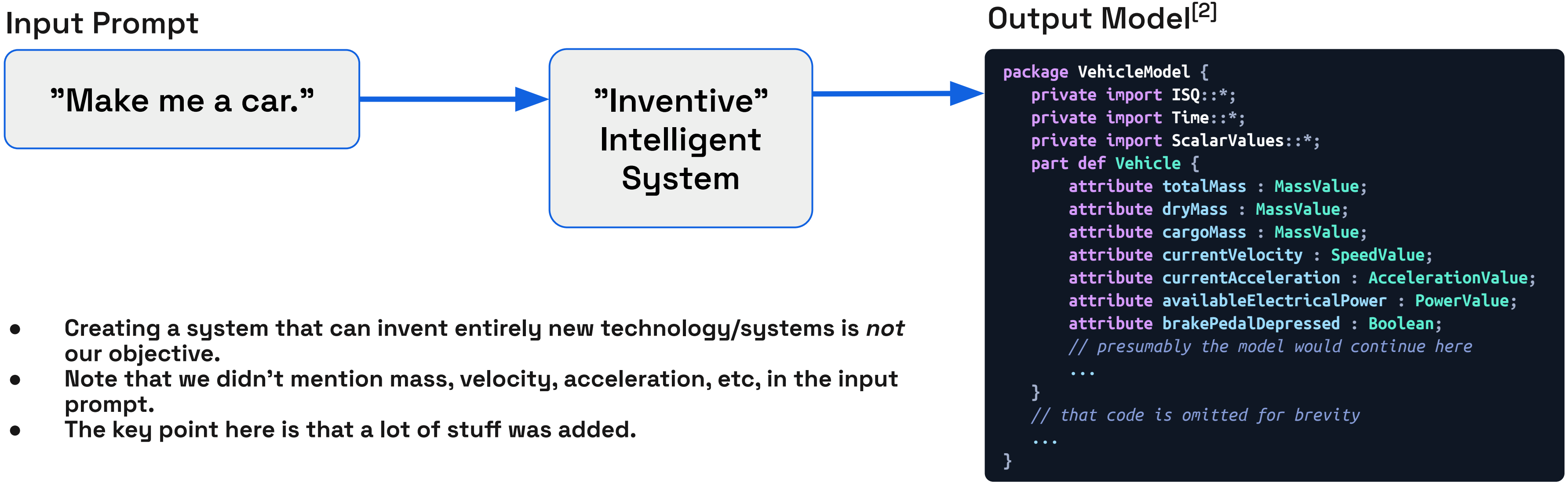
Issues, metrics, and vision

- Syntactic and semantic validity (distinguishable) [Solved]
- Physical/quantitative validity
- Output correctness / determinism (distinguishable)
- Information isomorphism
- Model connectedness/simulatability
- Information (de)duplication (in source material and in sysml model)
- Information deconfliction
- Traceability (to source material and for agent decisions)
- Model fidelity
- (Fuzzy) Model organization
- (Fuzzy) Convention correctness
- (Very Hard) Physical realizability
- Speed and scalability
- Review UI for human in loop
- Plug and play framework for building agents that create good SysML models
- And even if none of the agentic stuff works, there's still a very useful tool for working with SysML at the end anyways!

Extra details on script

If requested…

Inventive systems are *not* (currently) our objective[1]



Input Prompt

"Make me a car."

"Inventive" Intelligent System

Output Model[2]

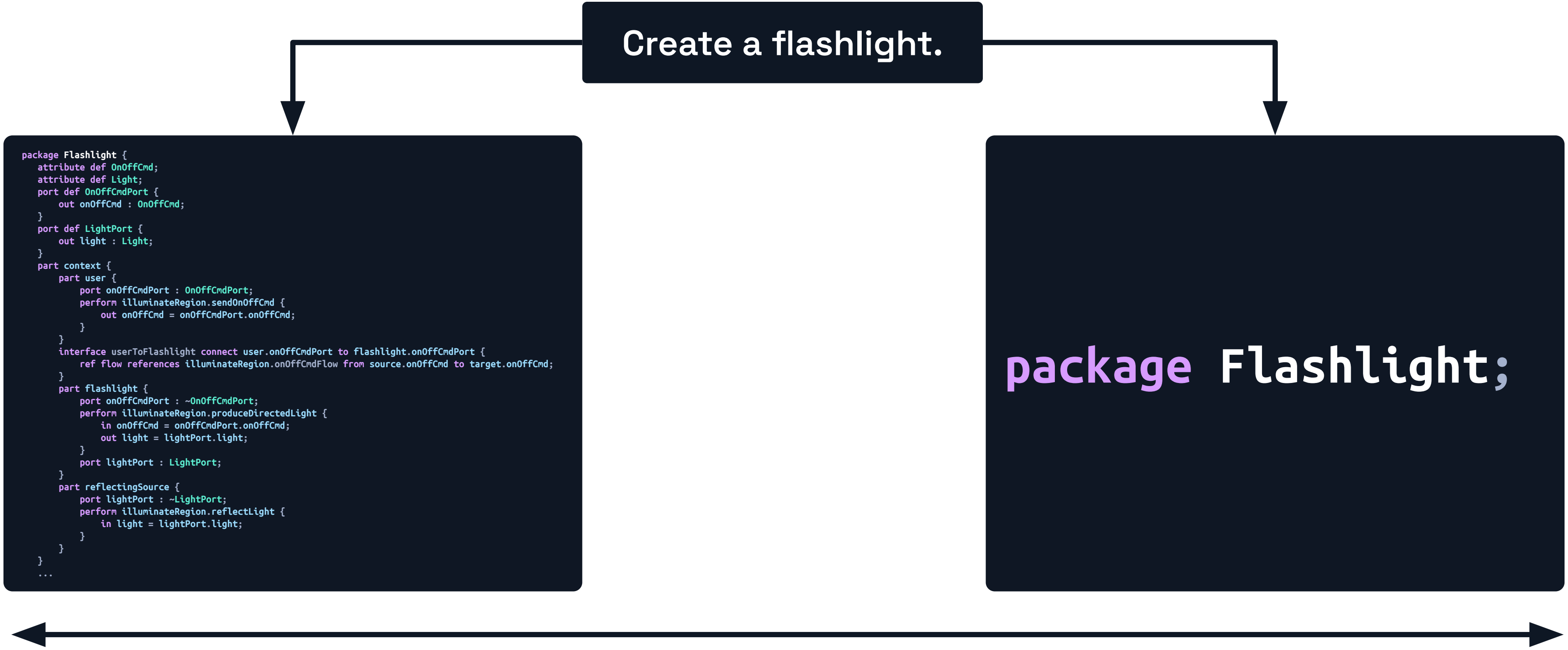
```
package VehicleModel {
    private import ISQ::*;
    private import Time::*;
    private import ScalarValues::*;
    part def Vehicle {
        attribute totalMass : MassValue;
        attribute dryMass : MassValue;
        attribute cargoMass : MassValue;
        attribute currentVelocity : SpeedValue;
        attribute currentAcceleration : AccelerationValue;
        attribute availableElectricalPower : PowerValue;
        attribute brakePedalDepressed : Boolean;
        // presumably the model would continue here
        ...
    }
    // that code is omitted for brevity
    ...
}
```

- Creating a system that can invent entirely new technology/systems is *not* our objective.
- Note that we didn't mention mass, velocity, acceleration, etc, in the input prompt.
- The key point here is that a lot of stuff was added.

[1] We're not saying that the objective can *never* be a system that can independently invent new and working technology. We're simply saying that this isn't our current objective. The full reasons why are out of scope, but it comes down to task complexity (it's very hard, possibly on the scale of centuries instead of funding rounds), vagueness of the target market, the fact that current LLMs are (debatably) not up to the task (as LLM creativity is more or less akin to probabilistic walk through a space of tokens, and not a deeper reasoning process, whatever "reasoning" really is), among other reasons - but this is a footnote and not an essay so I'll leave it at that.

[2] This is the SysML v2 Spec Annex A SimpleVehicleModel.sysml model from the official release, not LLM generated code.

Inventive vs. Translative



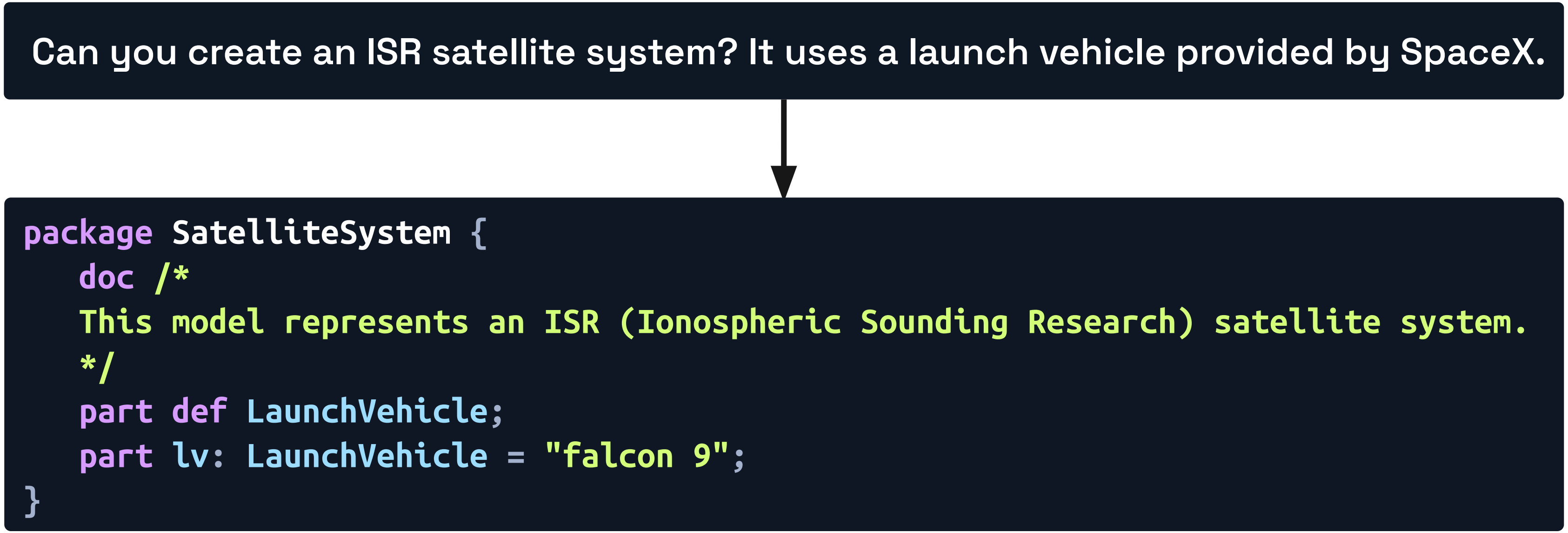
Create a flashlight.

```
package Flashlight {
    attribute def OnOffCmd;
    attribute def Light;
    port def OnOffCmdPort {
        out onOffCmd : OnOffCmd;
    }
    port def LightPort {
        out light : Light;
    }
    part context {
        part user {
            port onOffCmdPort : OnOffCmdPort;
            perform illuminateRegion.sendOnOffCmd {
                out onOffCmd = onOffCmdPort.onOffCmd;
            }
        }
        interface userToFlashlight connect user.onOffCmdPort to flashlight.onOffCmdPort {
            ref flow references illuminateRegion.onOffCmdFlow from source.onOffCmd to target.onOffCmd;
        }
        part flashlight {
            port onOffCmdPort : ~OnOffCmdPort;
            perform illuminateRegion.produceDirectedLight {
                in onOffCmd = onOffCmdPort.onOffCmd;
                out light = lightPort.light;
            }
            port lightPort : LightPort;
        }
        part reflectingSource {
            port lightPort : ~LightPort;
            perform illuminateRegion.reflectLight {
                in light = lightPort.light;
            }
        }
    }
    ...
```

```
package Flashlight;
```

This isn’t a binary category so much as a spectrum.

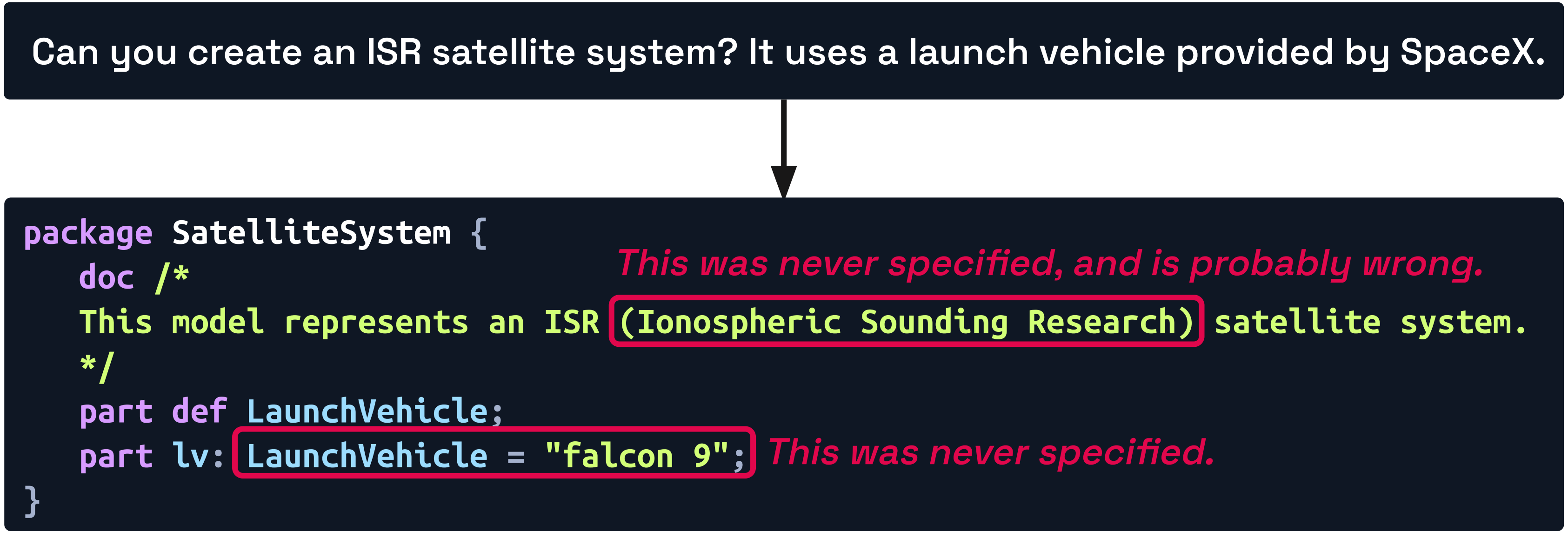
”Inventive” Intelligent Systems



Can you create an ISR satellite system? It uses a launch vehicle provided by SpaceX.

```
package SatelliteSystem {
   doc /*
   This model represents an ISR (Ionospheric Sounding Research) satellite system.
   */
   part def LaunchVehicle;
   part lv: LaunchVehicle = "falcon 9";
}
```

Information Isomorphism



Can you create an ISR satellite system? It uses a launch vehicle provided by SpaceX.

```
package SatelliteSystem {
    doc /*
    This model represents an ISR (Ionospheric Sounding Research) satellite system.
    */
    part def LaunchVehicle;
    part lv: LaunchVehicle = "falcon 9";
}
```

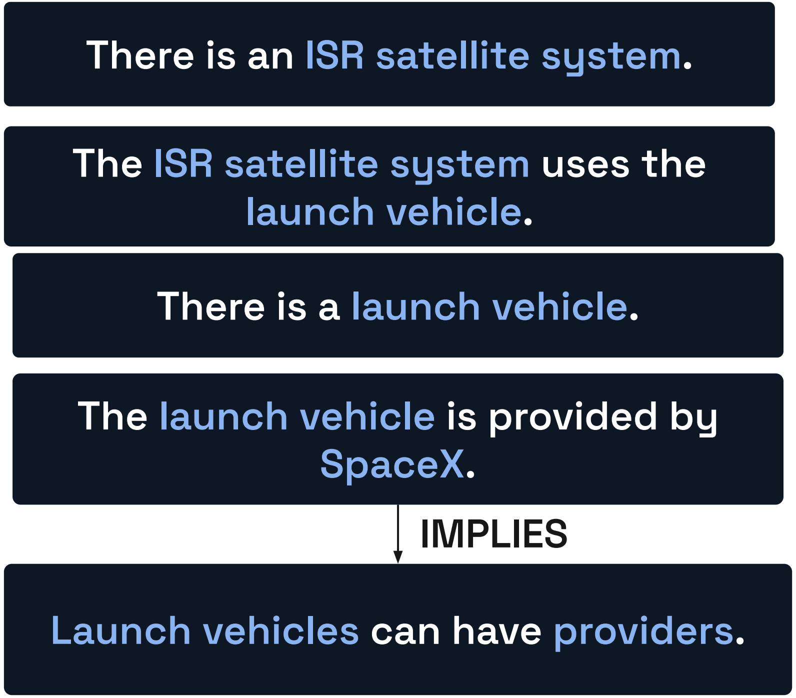
This was never specified, and is probably wrong.

This was never specified.

And where is the reference to SpaceX?
And (perhaps nitpicking) but the only reference to "ISR" is in the doc?

Statement Discretization

Can you create an ISR satellite system? It uses a launch vehicle provided by SpaceX.



There is an ISR satellite system.

The ISR satellite system uses the launch vehicle.

There is a launch vehicle.

The launch vehicle is provided by SpaceX.

IMPLIES

Launch vehicles can have providers.

CLAW 96, The First International Workshop on Controlled Language Applications, Katholieke Universiteit Leuven, 26-27 March 1996

Attempto Controlled English (ACE)

Norbert E. Fuchs, Rolf Schwitter
Department of Computer Science, University of Zurich
CH-8057 Zurich, Switzerland
{fuchs, schwitter}@ifi.unizh.ch

Attempto Controlled English (ACE) allows domain specialists to interactively formulate requirements specifications in domain concepts. ACE can be accurately and efficiently processed by a computer, but is expressive enough to allow natural usage. The Attempto system translates specification texts in ACE into discourse representation structures and optionally into Prolog. Translated specification texts are incrementally added to a knowledge base. This knowledge base can be queried in ACE for verification, and it can be executed for simulation, prototyping and validation of the specification.

1 Motivation

Somewhere between ridiculous pedantry and erroneous formulation there presumably exists a reasonably precise way of specifying a problem in English [Dodd 90].

Creating reliable software is hard. One of the worst obstacles to build a good software product grows out of shortcomings in writing a complete, consistent and unambiguous requirements specification. Managers and domain specialists often find it extraordinarily difficult to formulate specifications since at the beginning of the requirements engineering process the knowledge is usually informal, incomplete and opaque, and many – possibly conflicting – personal views of the system exists. Nobody knows what exactly the program should do until there exists a first version to run.

Requirements specifications are mostly written in natural language because they need to be understood by all participants. This involves a risk since the expressive power of unrestricted natural language can tempt people to write ambiguous or even incomprehensible statements. Apart from natural language people use arbitrary graphics, or semi-formal representations like structured analysis or entity-relationship diagrams that often have no formal semantics, or a poorly defined one, thus making formal reasoning impossible [Pohl 93].

Even when the software development team gets an acceptable requirements specification there can be problems because different people may understand the same document differently. To avoid disparate interpretations of a document, people have suggested to use formal methods [Hall 90]. However, formal languages are not easily understood by untrained users. Moreover, it is far from trivial to derive a formal specification from informal requirements since this derivation process cannot be formalised and cannot be formally validated [Hoare 87]. In the end, natural language comes back in through the back door when the formal specification must be accompanied by a natural language description that paraphrases *'what the specification means in real-world terms and why the specification says what is does'* [Hall 90]. It seems that introducing formal methods into the predominantly creative process of software development runs into immense difficulties.

But there is a way out. The specification language Attempto Controlled English (ACE) combines the familiarity of natural language with the rigor of formal languages. ACE enforces writing standards that restrict the grammar and the vocabulary, thus leading to documents containing more predictable and less ambiguous language. ACE helps people to find an agreement about the correct interpretation of a requirement specification. When domain specialists and software developers are guided to use the

Note: Inventiveness is not the same as Hallucination

The former relates to the addition of information, the latter relates to the inclusion of incorrect information.

Inventiveness

The tendency to add information to an output (correct or incorrect) that was not in the input. This seems to *co-occur* with the omission of input information in an output, but that is a separate issue.

Hallucination

… overconfident, plausible falsehoods, which diminish [the LLMs] utility and trustworthiness.

Definition Source: Why Language Models Hallucinate (OpenAI, arxiv.org, 2025)