Towards a Fault Management Analysis Tool for Model Centric Systems

Ksenia Kolcio[1], Maurice Prather[2], David Wagner[3], Maged Elaasar[3], Narek Shougarian[3,]

[1,2] *Okean Solutions, Inc, Seattle, WA, 98122, USA*
ksenia@okean.solutions
maurice@okean.solutions

[3] *Jet Propulsion Laboratory, California Institute of Technology, Pasadena, CA, 91109, USA*
David.Wagner@jpl.nasa.gov
Maged.Elaasar@jpl.nasa.gov
Narek.Shougarian@jpl.nasa.gov

ABSTRACT

In an effort to effectively develop more complex spacecraft fault management (FM) systems new technologies are sought to enable rapid diagnostic model generation and validation, and provide tools to perform FM analyses Model-based Systems Engineering approaches to FM system development are uniquely suited to be combined with model-based tools currently utilized in the design of other parts of flight systems. Combined tools utilizing information from a common system model can reduce design inconsistencies and gaps in analyses. Tighter integration of FM with other system-level and subsystem-level hardware/software development activities allows crucial redundancy and sensor placement trades to be performed earlier and throughout the mission lifecycle.

Our work has been towards the integration of a model-based fault management tool suite called MONSID®, with JPL's Computer Aided Engineering for Systems ARchitecture (CAESAR) platform as a way to improve FM system modeling and analysis. MONSID relies on application-specific models of the system being monitored. MONSID models consist of interconnected elements representing system hardware and measurement/command input points, called the topology. Model topology design is currently a manual process and often relies heavily on paper documentation such as hardware/software specs, engineering drawings, and interface control documents. CAESAR is a semantically- driven toolchain for model-based system engineering. At the core is a system model expressed in the Ontological Modeling Language (OML). It is intended to support semantic modeling, consistency validation, and continuous integration.

A goal of the combined toolset is to automate FM model development by directly extracting models from CAESAR and then analyzing them in MONSID. Analyses currently available in MONSID include model topology inspection and validation and fault isolation capability based on sensor placement. While we have focused on two specific tools, the integration approaches can be leveraged by other semantically driven model-centric platforms and tools.

This paper describes the evolution of our integration approaches as we evaluated different insertion points in the CAESAR toolchain with respect to MONSID model requirements. The MONSID-CAESAR tool is demonstrated on a simplified example of a spacecraft heat reclamation system. Results of the generated MONSID model are discussed, including levels of automation achieved and information surfaced to the users about the extracted model topology.

1. INTRODUCTION

New technologies are sought to effectively manage and streamline increasingly complex fault management (FM) systems, enable rapid diagnostic model generation and validation, and provide tools to perform FM analyses and trades e.g., fault isolation capability, FM model validation, and sensor placement.

Traditional fault detection systems monitor signals for known out of limit levels (e.g. over temperatures, excessive rates, stale data). Such monitors are designed to detect specific fault conditions. In contrast, the model-based fault detection approach utilized in this work relies on exposing deviations from modeled behavior which is assumed to be correct and as intended.

Systems Engineering approaches to FM system development are uniquely suited to be combined with model-based tools currently utilized in the design of other parts of flight systems.

Tighter integration of FM with other system-level and subsystem-level hardware/software development activities allows crucial redundancy and sensor placement trades to be performed earlier and throughout the mission lifecycle.

Our work in the first year of a two-year program has been towards the integration of a model-based fault management tool suite, called MONSID® (Kolcio 2016), with JPL's Computer Aided Engineering for Systems ARchitecture (CAESAR) (Elaasar, Rouquette, Wagner, Oakes, Hamou-Lhadj, Hamdaqa, 2023) platform as a way to improve FM system modeling and analysis. MONSID relies on application-specific models of the system being monitored. MONSID models consist of interconnected elements representing system hardware and measurement/command input points, called the topology. Model topology design is currently a manual process and often relies heavily on paper documentation such as hardware/software specs, engineering drawings, and interface control documents. It is also iterative, which itself is not problematic but when relying solely on manual updates, can lead to inconsistencies with the physical system it is supposed to be modeling. CAESAR is a semantically- driven toolchain for model-based system engineering. It is intended to support semantic modeling, consistency validation, and continuous integration. As such, CAESAR is an authoritative source of truth for the design of the physical system. The toolchain is used at JPL for flight system engineering design, mainly for power systems. It is also starting to be used by other disciplines including flight software, mechanical, and now fault management and safety and mission assurance. Where once systems engineering model development was siloed by discipline, CAESAR enables models to be developed and maintained from the same data source.

A goal of the combined toolset is to automate FM model development by directly extracting models from CAESAR and then analyzing them in MONSID. Analyses currently available in MONSID include model topology inspection and validation and fault isolation capability based on sensor placement. While we have focused on two specific tools, the integration approaches can be leveraged by other semantically driven model-centric platforms and tools.

The rest of this paper is organized as follows. We first introduce the MONSID and CEASAR technologies. Next, we describe our integration approaches and development of a MONSID-CAESAR adapter. We then discuss how the adapter was demonstrated on an example system modeled in CAESAR. Finally, we offer conclusions from the demonstration and discuss future work.

1.1. MONSID Background

The MONSID system is a model-based software package designed to detect and identify hardware faults and off-nominal behavior. MONSID is composed of a diagnostic engine and models of nominal behavior of the target system hardware. The generalized fault detection and identification approach is applicable to virtually any system with health and safety requirements. The physics-based models are application-specific. There are opportunities for model reuse for programs leveraging similar COTS or GOTS (commercial/government off-the-shelf) hardware. Examples of MONSID applications include rovers (Kolcio, Fesq, & Mackey, 2019), CubeSats (Mackey, Nikora, Fesq, & Kolcio, 2021), and high-fidelity testbeds (Kolcio & Prather, 2023). In this paper we focus on MONSID model generation and analysis.

1.1.1. MONSID Model Development and Analyses

MONSID models are comprised of interconnected components representing various system functions. By design they capture interactions among components and so provide a system-level view. At the top level, the MONSID model represents the system topology, i.e., how the various components are interconnected, and data insertion points where commands and sensor data enter the model. Model topology can be derived from system architecture and high-level operational descriptions.

For example, power system block diagrams, command and telemetry dictionaries, and operational concept documents can be used to specify the MONSID model topology of a power subsystem.

MONSID model developers use these artifacts to determine how the various power subsystem hardware components should be interconnected and where to place sensor and command insertion points. Factors considered as part of the high level MONSID model design include level of detail (assembly level, box level, or slice) and minimum sensor suite needed for fault detection and isolation. These design choices ultimately reflect the accepted single point failure for the mission risk classification per guidelines, for example as defined in NASA Procedural Requirements (NPR) 8705.4, Risk Classification for NASA Payloads. As FM design evolves through the program phases, so must model development, and it is crucial to avoid gaps and inconsistencies between the model and underlying system.

Model topology can be analyzed to determine regions in the model where it is not possible to identify a single fault source. These regions are called ambiguity groups. An ambiguity group with more than one member implies that every member is a fault suspect but there is insufficient information to resolve the fault to one member. For a perfectly diagnosable model, each component and sensor would be the only member in its own ambiguity group. Ambiguity groups with

more than a single member can be reduced with the addition of sensors, which may be possible when fault management design is closely coupled with flight system design.

MONSID model topology development and analysis are completed with the help of the MONSID tool suite. The tool suite includes a web app called MONSID Toolkit which provides visual model design and is also used to analyze how well faults can be isolated to a single component (diagnosis resolution (Kolcio, Fesq, & Mackey, 2017]). Model visualization and diagnosis resolution are the primary uses for models generated by the MONSID-CAESAR adapter. Although not further discussed here, Toolkit is also used to step through MONSID engine runs. Screenshots depicting various Toolkit usages are shown in Figure 1.

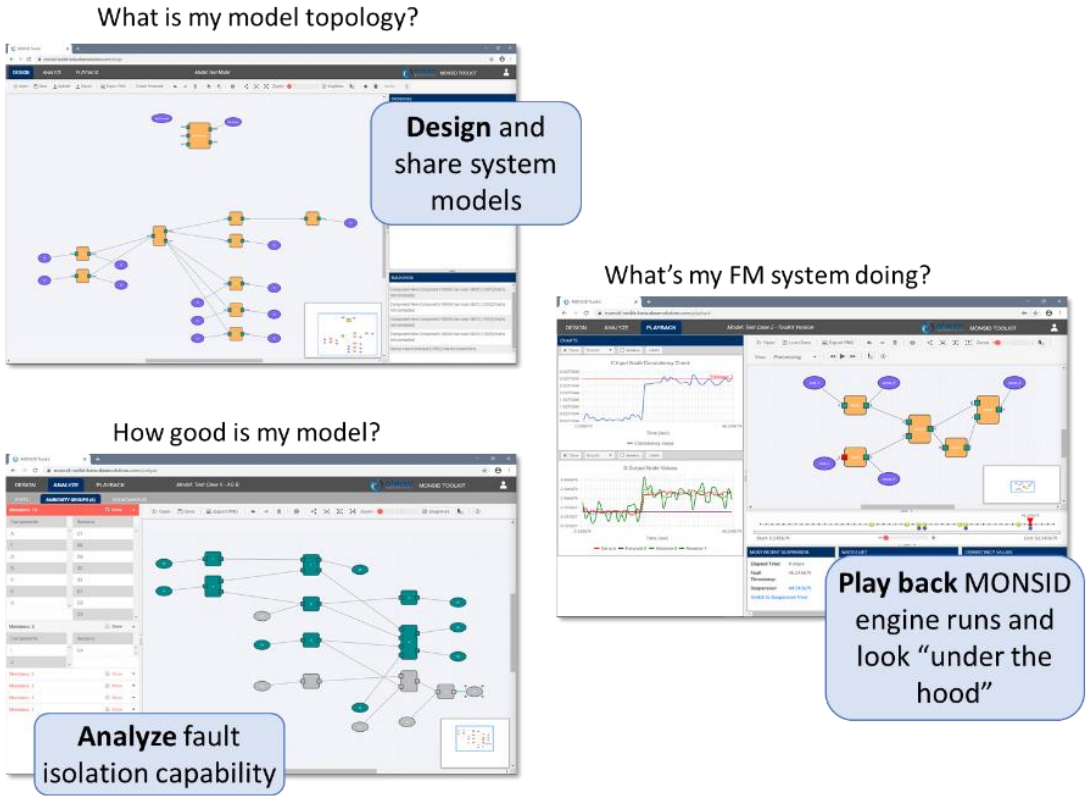

Figure 1. MONSID users can visualize and analyze and models, and inspect MONSID engine runs via the Toolkit web app.

The model development time depends on the scale of the program and available expertise. For example, it can take 3-4 person months for an engineer with an attitude control system background to develop an ACS MONSID model of a program on the scale of a CubeSat or rover. It can take considerably more time for engineers venturing outside their domain of expertise. Perhaps the largest factor affecting development time is the availability, correctness, and completeness of the resources used to develop the models. Sources are not always in familiar or easily navigable formats. Even when sources/data are available, it could take time and coordination to gain access to it, which can happen when different departments or organizations are involved. Heavy reliance on paper documentation which can be outdated or the wrong version can also result in design gaps and mistakes.

The resource issues are compounded for model development on larger scale programs with more complex systems, larger teams, and much more paper documentation to utilize. While the current model development process is viable and has resulted in validated, proven, and effective models, there is room for improvement. To date, MONSID has been applied to fairly small-scale systems like CubeSats, rovers, and hardware testing platforms. Without improvements in current processes, we can expect significant increases in model development time for large, complex systems.

1.2. CAESAR Tool Chain

CAESAR [1] is an open-source technology developed and maintained by NASA JPL designed to enable different

[1] GitHub - opencaesar/oml: Ontological Modeling Language (OML)

engineering teams to effectively share and manage program data in the process of performing domain-specific program development activities. It provides ontological description, or modeling, of various system data. In CAESAR, the data is modeled via ontological instances with precise syntax and logical semantics, and thus no longer tool-specific. Often referred to as a "semantic data warehouse" CAESAR is fully compatible with the Web Ontology Language (OWL-2, specified by the Worldwide Web Consortium). To make working with ontologies more efficient and easier, JPL has developed the Ontological Modeling Language (OML) that is a subset of OWL 2. OML provides more concise textual and XMI representations of OWL that precisely encode the expressive patterns and rules most commonly used in engineering descriptions. OML also uses description logic rules so that vocabulary consistency can be formally verified and models can be verified for consistency with the vocabulary using generic commercial or open-source reasoners. Logic reasoners are used to add inferences to the data which allows queries and database engines to pick up inferred associations.

The CAESAR environment consists of several tools to accomplish workflows from authoring models, called descriptions at one end to generating reports from various queries at the other end. The environment includes a conversion tool OML to OWL which allow open-source description logic reasoners such as Pellet to check model consistency. CAESAR is being utilized on NASA JPL programs including PSYCHE, Europa Clipper (Wagner, Kim-Castet, Jimenez, Elaasar, Rouquette, & Jenkins, 2020). and Mars Sample Return.

The initial focus has been on Electrical Flight System Engineering and harness design. Authoring and reporting tools were developed for this domain. On Europa Clipper and Psyche, electrical system assemblies, electrical interfaces and interconnectivity were captured. CAESAR was also used to integrate with telemetry and command specification data. Benefits to using CAESAR included a reduction on manual steps in harness specification and design, enabling system engineers to specify requirements instead of design, and re-usability (from Europa Clipper to Psyche). The MONSID-CAESAR interface will provide similar benefits, i.e. reducing manual steps to MONSID model creation.

At this time most of the models in CAESAR are higher-level system architecture specifications. Efforts at JPL have started to incorporate behavioral and analytical models. Near term plans include adding more domains such as mechanical and thermal. The FM piece has not yet been addressed, but could clearly benefit from a modeling platform.

2. MONSID AND CAESAR – THE BIG PICTURE

System engineering uses models to help automate engineering analysis. In this case, we are using a model transformation from CAESAR's descriptive system model to MONSID's analytical model to enable the kind of fault analysis that MONSID can provide. Model-based approaches enable different engineering teams to effectively share and manage a variety of data in order to perform domain-specific mission development activities. Government and industry organizations are seeking ways to better integrate the myriad tools through the use of authoritative source of truth modeling environments such as CAESAR. Figure 2 illustrates the capability in the context of FM system engineering activities.

The CAESAR modeling environment is the central area where program data is manipulated, and effectively kept under revision control such that information entered by hardware/software engineers on the left side of the figure can be utilized to generate a diagnostic model and update FM system analyses shown on the right side of the figure. The flow of information to/from CAESAR is facilitated by adapter interfaces allowing custom/COTS tool to work with the modeling environment.

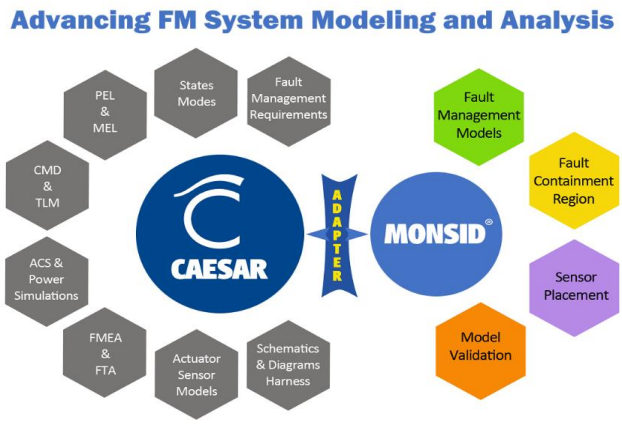

Figure 2. Model-based FM interacting with a modeling environment.

The core of modern system engineering processes are the models. Practically, there are many types of models, with different data types and levels of fidelity, used for various purposes. Although it is not feasible to think of "one model to rule them all", one can consider the authoritative and traceable representation of knowledge as sources of truth. That is what environments such as CAESAR are designed to do. A CAESAR model captures a set of facts about a system architecture and maintains change control over those facts. The facts describe elements that define the architecture, their properties, and relations, particularly including functional interface relations. It is the authoritative source of truth for this information on projects utilizing CAESAR.

Model-based FM can benefit from this source not only by extracting continuously up to date information needed to develop and test diagnostic models but also by the ability to automatically generate these models. MONSID models in

particular describe the nominal behavior of the underlying physical system.

However, there currently are no FM-specific interfaces to CAESAR for tools like MONSID. A goal of a MONSID-CAESAR adapter is to alleviate this problem. As emphasized in the NASA Fault Management Handbook (2012), tighter and earlier integration (w.r.t lifecycle) of FM with "nominal" system engineering activities such as system-level and subsystem-level hardware/software development can enable crucial trade studies, such as redundancy and sensor placement, to be performed earlier, more often, and more quickly.

Enabling the FM practice to be more tightly integrated with the other systems engineering activities makes it more accessible to non-FM experts to improve the overall system design. Providing this capability enables the systems team to more easily visualize models and glean precise information they need without necessarily having to understand all the technical details of the underlying FM design.

Through a MONSID-CAESAR adapter many of the previously mentioned issues with manual information collection can be alleviated and eventually removed. Information from disparate sources would be presented to FM engineers with consistent terminology and format. CAESAR would keep an authoritative running history of that data. This will shorten design time as well as improve model fidelity. As CAESAR evolves to include analytical models that capture behavior/operation, this information can be extracted to readily generate constraints for MONSID components, which are needed to realize an executable model that can be run with real data. Initially, the actual coding of MONSID components would still be done manually. The long-term goal would be to incorporate transformation tools to automate MONSID component development in support of automatically generated flight software.

FM analyses can greatly benefit from improved models. Model topology has been successfully used to determine how well the MONSID engine will be able to isolate faults based on the ambiguity groups. The results of ambiguity group analysis and sensor placement trades generated by MONSID can be fed back to CAESAR. This process would allow changes that influence the FM capabilities to be readily validated to ensure the changes don't adversely affect the established baseline functionality. For example, without this capability, if a sensor is removed from a subsystem, the system engineer may not know that removal would adversely affect the FM design.

A long-term goal of a MONSID-CAESAR adapter is to improve the determination and analysis of fault containment regions (FCR), which are key for fault tolerance. An FCR delineates a boundary around spacecraft components (hardware and/or software) such that a fault within the boundary of the FCR does not propagate outside (Avizienis, 1997). Partitioning of containment regions (Chau, Alkalai, & Tai, 2000) involves determination of a subsystem to system hierarchy, redundancy scheme, and design diversity (different hardware/software to accomplish the same task). Thus, faults can be contained in different ways through non-intersecting paths (e.g., separate data/power buses), and functionally or identically redundant hardware strings. Ensuring that the fault containment design will meet fail safe/operational requirements for single point failures is a manual, painstaking process. The number of FCRs is a primary factor in determining how many faults a system can tolerate and still preserve the spacecraft.

We postulate that MONSID models can also be used to analyze FCRs in a similar fashion to ambiguity groups. Through inputs from CAESAR, the identification of and tracing/analyzing FCR would be facilitated and eventually automated. It should be noted that, in general, diagnostic resolution and FCR are not the same. However, they can overlap when lower-level fault isolation is not needed or not possible. Rigorous assessment of FCR will provide assurance that mission risk classification requirements with respect to single point failure tolerance are being met.

While our effort has thus far focused on creating an adapter to work one way, from CAESAR to MONSID, it is desired to eventually bring results back into CAESAR. In the current process, such information would need to be manually authored into the CAESAR model. An automated two-way adapter would place MONSID model data and analyses into the continuous integration toolchain. Tighter integration of FM design with nominal system design would also be facilitated and would greatly benefit from a two-way adapter.

3. INTEGRATION APPROACHES

The task of converting one representation of data to another is simply a transformation exercise. The most important aspect of this task is identifying the places where data can be accessed and manipulated to achieve the desired goal. The CAESAR toolchain has several stages as shown in Figure 3, with the data cycle moving in a left to right fashion.

The key characteristic of that cycle is that model data will be stored in a database. This database breaks the data lifecycle into two phases, which we have simply termed as "pre-database" and "post-database". The CAESAR toolchain is responsible for populating the database with reasoned data, meaning the source OML model has instantiated, rules applied and the composite information has been run through a semantic reasoner. The database contains a well-formed and semantically correct model.

In the pre-database phase, referring to Figure 3, transformation is handled through the creation and maintenance of OML vocabularies and OML-based tools (not shown in Figure 3). We note that in this paper, vocabulary names are in lower case to distinguish them from

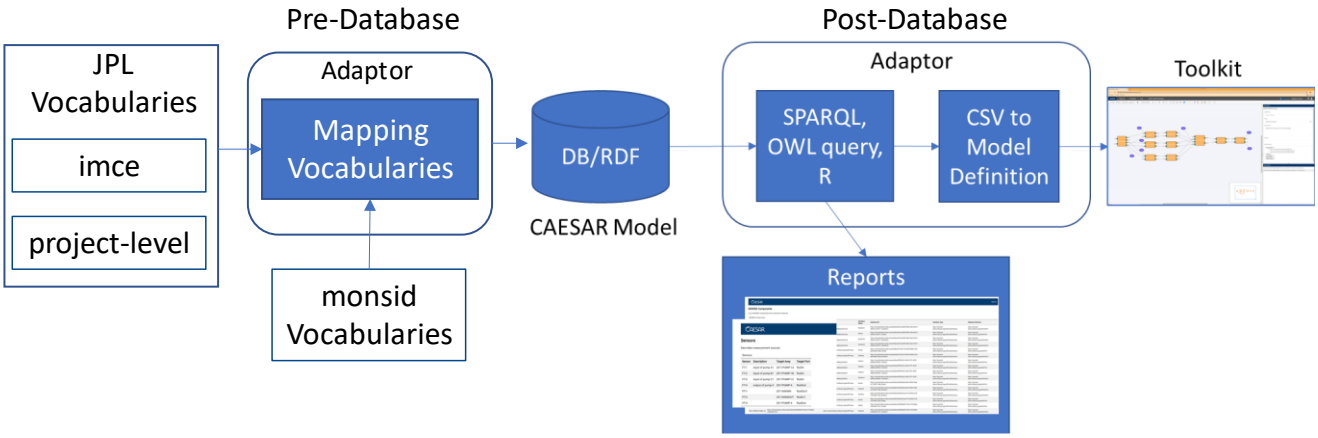

Figure 3. The MONSID adapter to CAESAR consists of portions before and after the CAESAR database to leverage the reasoning and query execution tasks of the CAESAR toolchain.

similarly named organizations or products. JPL, for example, relies on the integrated model-centric engineering (imce) vocabulary when constructing models for any project. Then project-level vocabularies can also be created to specialize imce vocabulary terms as well as additional terms specific to a particular project's needs. MONSID terminology is appended to the system model via the monsid and mapping vocabularies (discussed in the next section). The result is run through a semantic reasoner and loaded to a database, as represented by the CAESAR Model in Figure 3.

In the post-database phase, the transformation task takes on a more traditional ETL (extraction, transform, load) approach. Data contained in the database can be queried to produce a variety of reports and outputs that can be fed to other downstream processes. As illustrated in Figure 3, queries are used to generate HTML reports and text files (CSV). The adapter processes the text files to generate MONSID model definitions which can be visualized and analyzed in the Toolkit app. As with any ETL task, the source and target schemas must be well defined.

3.1. MONSID-CAESAR Adapter

There were two main requirements defined for the adapter:

- Generate MONSID model topology from a CAESAR project
- Format shall be ingestible by MONSID Toolkit web application

To meet the first requirement, key MONSID model topology elements were identified as things to be mapped to a CAESAR model. The MONSID topology elements are defined by:

Models

- Contain Components
- Represent subsystems

Components

- Contain Nodes
- Represent hardware and physical behaviors

Nodes

- Effectively represent state variables
    - *Conserved* state variables (e.g., fluid flow, electric current, momentum)
    - *Potential* state variables (e.g., pressure, voltage)
- Used as endpoints for each Connection

Connections

- Used to describe a link between 2 nodes

Sensors

- Provide commanded and reported data from the physical system
- Provide data directly to Nodes

In order to extract existing model information for utilization in the MONSID tool suite, we establish a set of guiding principles for the MONSID-CAESAR adapter.

- Create a repeatable process that is not ad hoc for projects to add FM analysis.
- Keep the setup cost to a minimum. Ideally, we wanted all users of a specific vocabulary to gain the same benefit as past projects.
- Make it adjustable, knowing that each project will have some customizations that can only be addressed with bespoke adjustments to the adapter.

By providing a general framework to extract any CAESAR model and bringing that information into MONSID, we needed to explore how to best utilize the CAESAR toolchain. We examined two strategies leveraging post-and pre-database phases of the CAESAR toolchain starting with post-database.

3.1.1. Post-database

We created a prototype adapter to convert an existing CAESAR model of an example heat reclamation system by solely leveraging the reasoned data contained within database. We decided on this approach as a first step to better understand the mechanics of the CAESAR toolchain and build a suite of tools that would allow the ingest of any CAESAR model.

The prototype adapter was implemented in two parts. The post-database adapter and data flow are shown on the right side of Figure 3. The first part was the extraction of MONSID model elements from the CAESAR database. This was accomplished by creating queries and scripts to generate output files (CSV format) that contained MONSID model topology elements. The second part of the adapter was the creation of an executable that generated a MONSID model definition file from the csv output files.

When working with CAESAR models, there are effectively two model states. The first, is the native OML format, which we refer to as the source model. In this state, the model reflects the exact representation as intended by the authoring team and, more importantly, has not been altered by the reasoner. Once the source model is fed to the reasoner, the model becomes the reasoned model. In this state, a reasoned model has been validated/verified by the reasoner as well as contains additional data that may injected by other parties. This injection is discussed in the Pre-database section that follows. The reasoned data can be considered a more complete and expanded representation of the initial source model.

Although it is possible to design an adapter to exclusively work in the post-database realm, we found that quite a bit of knowledge of the vocabulary and source model was required. The database queries had to be extensively tailored to the source CAESAR model. Although some tailoring from project to project is to be expected, one of our objectives was to minimize project-specific tailoring to make the adapter more generalizable and easier to introduce to other projects. The problem with queries for the post-database only case is that the query authors need to really understand the vocabularies that are in use for the CAESAR model. This presents a substantial learning curve for MONSID users as some expertise of vocabulary structure and how it's used is required. Moreover, completely new queries would be needed for applications with different vocabularies.

The verbosity of the source model is highly dependent on institutional knowledge and practices. Thus, reasoned data alone does not give us more context and it becomes very difficult to determine what source model elements are needed in the resulting MONSID model. The prototype was dependent on a priori knowledge of the source model and that information was embedded throughout the two parts of the adapter.

Lastly, the executable needed to incorporate a variety of assumptions in order to process the data produced by the queries. For example, state variables were not explicitly implemented in the source model but needed to be realized in the final transformation step when creating the MONSID model. It should be mentioned that this is a limitation of the source model used to demonstrate our work, not the CAESAR platform itself. One could also argue that any source model designed for other system engineering purposes may need some augmentation in order to be useful for MONSID. However, any augmentation done exclusively in the post-database space introduces project-specific artifacts which we would like to minimize.

Heavy use of hard-coded knowledge may serve for a prototype; however, incorporating this type of information would not make for a scalable and general solution that could be easily transported to another project.

3.1.2. Pre-database

By working with data in the pre-database phase, it becomes easier to transform any CAESAR model into the desired representation. This is done by leveraging the strength of CAESAR's OML vocabulary and vocabulary mapping constructs. Work in this phase utilizes the semantic reasoner to associate and transform the data. In a typical ETL project, the Transformation step follows extraction; however, by utilizing OML, the transformation step is handled first which allows the overall solution to become more readily project agnostic.

There are three key elements to this process:

1. Create a vocabulary that represents the target vernacular. In this case, we need to create an OML vocabulary that represents the MONSID topology elements defined in the MONSID-CAESAR Adapter section.
2. Identify the most common vocabulary (e.g., upper ontology) for the source project. In this case we used JPL's imce vocabulary.
3. Create a mapping vocabulary between the monsid vocabulary and common vocabulary. This may be further expanded to include a project-level mapping vocabulary to accommodate project customizations.

Creating a vocabulary that represents MONSID elements is a task that only needs to be completed once. This vocabulary represents all the tangible items and relationships that are

needed to describe a MONSID model within OML. This task is effectively a one-time task and becomes a crucial component in the overall toolchain process.

The monsid vocabulary represents elements of the MONSID Model Library API. Vocabulary OML classes were created for each of the MONSID model topology elements listed above. See Figure 4 for a sample representation of the monsid vocabulary.

```
vocabulary <http://okean.solutions/monsid/foundation#> as m {

  extends <http://www.w3.org/2001/XMLSchema#> as xsd

  // ------------------------------------------------------------
  // Aspects
  // ------------------------------------------------------------
  aspect IdentifiedElement
  aspect NamedElement
  aspect SuspendableItem
  aspect StateVariable
  aspect ConservedStateVariable :> StateVariable
  aspect PotentialStateVariable :> StateVariable

  // ------------------------------------------------------------
  // Concepts
  // ------------------------------------------------------------
  concept Model
  concept Component
  concept Sensor
  concept Node
  concept Parameter

  concept InputNode :> Node
  concept OutputNode :> Node

  concept Connection

  // ------------------------------------------------------------
  // Enums
  // ------------------------------------------------------------
    enumerated scalar NodeTypeEnum [
      "Unknown",
      "Input",
```

Figure 4. Subset of monsid vocabulary

In the second step, identifying the common vocabulary that a given organization prefers is straightforward as most projects within that organization will use a single vocabulary along with project-level customizations. Time invested in understanding this vocabulary will make it easy to consume any project that leverages the same vocabulary.

Mapping Vocabulary

The third step involves establishing a mapping between the vocabulary of the CAESAR model to the vocabulary of the analysis tool. Thus, the goal is to understand how the imce vocabulary is utilized and then create a single mapping vocabulary between monsid and imce (the monsid-imce vocabulary).

The OML language syntax provides a mechanism by which a vocabulary can be extended by another vocabulary. This is a powerful way to easily transform the original representation created by the source vocabulary. For example, in the monsid vocabulary there exists an OML Concept called Component. In the imce vocabulary, there is an OML Concept called Assembly. With OML, it's straightforward to create a mapping that states an imce Assembly is a specialization of monsid Component. Figure 5 illustrates a subset of the monsid-imce mapping vocabulary.

```
vocabulary <http://okean.solutions/monsid-map/monsid-imce#> as map {

  extends <http://okean.solutions/monsid/foundation#> as m

  extends <http://imce.jpl.nasa.gov/foundation/base/base#> as base
  extends <http://imce.jpl.nasa.gov/discipline/fse/fse#> as fse
  extends <http://imce.jpl.nasa.gov/foundation/mission/mission#> as
  mission

  ref concept fse:Assembly :> m:Component, m:NamedElement

  ref concept fse:Subsystem :> m:Model, m:NamedElement

  rule AggregatesToCompContainer [ m:Model(mm) ^ m:Component(c) ^
  base:aggregates(mm,c) -> containsComponent(mm,c)
  ]

  ref concept mission:Interface :> m:Node, m:NamedElement
```

Figure 5. Subset of the monsid-imce mapping vocabulary.

By that same extension, if a project creates a project-specific vocabulary, a project-level mapping vocabulary can also be created as seen in Figure 6. For example, using the same model used in the first prototype adapter, we were able to clearly define items that could be treated as monsid StateVariables and subsequently eliminate the hard-coded assumptions that involved them in the post-database tools (see yellow box in Figure 6).

```
vocabulary <http://monsid-demo/monsid-hrs/project-map#> as map {

  extends <http://monsid-demo/library/types#> as hrs-types

  extends <http://okean.solutions/monsid/foundation#> as m
  extends <http://imce.jpl.nasa.gov/foundation/base/base#> as base


  // Project specific mapping
  ref concept hrs-types:Sensor :> m:Sensor
  ref relation entity hrs-types:Senses :> m:NodeDependency

  //This will provide friendly names for nodes
  ref scalar property base:hasAlternateName :> m:Name

  ref concept hrs-types:InPort :> m:InputNode
  ref concept hrs-types:OutPort :> m:OutputNode

  // let psv and csv be superclasses of the proj-specific SV's
  aspect FlowType :> m:ConservedStateVariable
  aspect PressureType :> m:PotentialStateVariable
```

Figure 6. Subset of project-level mapping vocabulary.

By leveraging mapping vocabularies, all the benefits of the semantic reasoner come into play. All the rules, properties, and associations will be honored once the mapping is utilized. Further, by leveraging OML, the expertise needed to incorporate project customization is already available. More importantly, though, once the data has been reasoned and loaded into the database, the tool adapters will no longer need to know anything about the source vocabulary or project-specific details. Everything can be queried relative to our monsid vocabulary. This greatly simplifies the database queries as everything is relative to MONSID elements. Hard-coded knowledge, string lookups, and other assumptions about the project's contents can be removed from the database queries and any of the tools that consume those queries.

Finally, and perhaps most importantly, aside from injecting the mapping vocabularies into the source project, there is no need to adjust the source project. This is a key characteristic as additional tooling should benefit but not require altering existing CAESAR models in order to use them. A combination of pre- and post-database interaction offers the most flexibility and scalability for any model-based tool that would like to leverage the data within a CAESAR model.

4. EXAMPLE DEMONSTRATION

The combined pre- and post- database adapter was demonstrated on the fluid domain of a simplified heat reclamation system (HRS) based on the Europa Clipper mission. The HRS circulates a working fluid to heat and cool spacecraft components as needed to maintain thermal stability. The working fluid was assumed inviscid and isothermal, thus volumetric fluid flow and pressure were the only sensed state variables. A block diagram of the pump assembly portion of the HRS is shown in Figure 7.

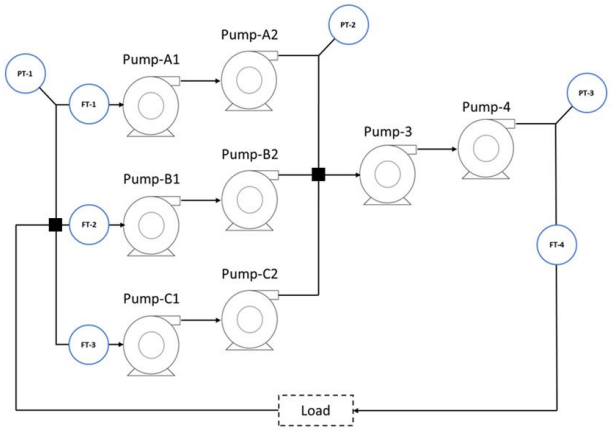

Figure 7. Block diagram of Heat Reclamation System.

The system includes three parallel banks of two pumps connected to two pumps in series for a total of eight pumps. Sensors are represented by the blue circles. Pressure sensors are labeled PT and flow sensors are labeled FT. The lines connecting the pumps represent pipes. The small black squares represent pipe junctions. The pump configuration of the example system is representative of actual spacecraft HRS. Simplifications such as ideal pipes and incompressible fluid were deemed appropriate for demonstrating the feasibility of the adapter.

A CAESAR model of HRS was specified using the imce vocabulary and specializations thereof to describe the individual pumps, flow and pressure sensors, and ideal pipes and pipe junctions. The load was not modeled. Pumps and pipe junctions included fluid in-port and out-port interfaces (not shown in the figure). Each pipe created a fluid flow connection between out port and in ports. Fluid flow and pressure state variables were not explicitly modeled in CAESAR. The fluid state variable was implied by the pipe connections. The pressure sensors implied pressure state variables.

The monsid-imce mapping vocabulary (see Figure 5) described above was used directly in the pre-database portion of the adapter. A project level vocabulary (see Figure 6) was created to map the HRS sensors to MONSID sensors, and in-port and out-port type to the corresponding MONSID input and output node types. Flow and pressure types were created in the HRS mapping vocabulary to separate and make explicit fluid flow and pressure, which were implied in the CAESAR model. The FlowType maps to ConservedStateVariable and PressureType maps to PotentialStateVariable, as shown in the yellow box in Figure 6.

The adapter executable generated additional nodes and connections as needed based on state variable type (conserved or potential). For example, each input/output node on a pump component was expanded into separate nodes for fluid flow and pressure. For a pipe junction component, one additional input node and output node (and connections) were needed for the pressure state variable.

A snapshot of the MONSID model generated by the adapter executable is shown in Figure 8. It is annotated for comparison with the HRS block diagram in Figure 7. Comparing this to the block diagram in Figure 7, a similar configuration of pump and junction components and sensors is seen.

The MONSID model topological elements introduced previously in 3.1 are now described for the HRS MONSID model shown in Figure 8. The orange boxes are the components which represent system hardware such as the pumps. Pipe junctions are also represented as components to model fluid flow splitting into different paths. The small green boxes on either side of the components are the input/output nodes; nodes on the left side of a component are input and nodes on the right are output. Each node is either a pressure or fluid type, as appropriate for a fluid domain model. The black lines between nodes are the connections. In

MONSID models there are separate paths for each state variable, in this case for flow and pressure. The adapter ensures that each fluid node is connected only to another fluid type node and similarly for pressure node types. The purple ovals represent sensors, which for this model are pressure and flow sensors. MONSID sensors represent points in the system where there is information available about its behavior. Sensors play an important role in the formation of ambiguity groups as will be discussed later.

On the Analyze page sensors can be added to the model to see how ambiguity group membership reduces. Ambiguity groups are automatically recalculated when model topology changes. To include the junction component and pressure sensor in the calculation, new sensors are added to the two input nodes of the left junction component. The result of adding the new sensors (labeled Sensor 8 and Sensor 9) is shown in Figure 11. First notice that now all items in the model are considered in the ambiguity group calculations (all

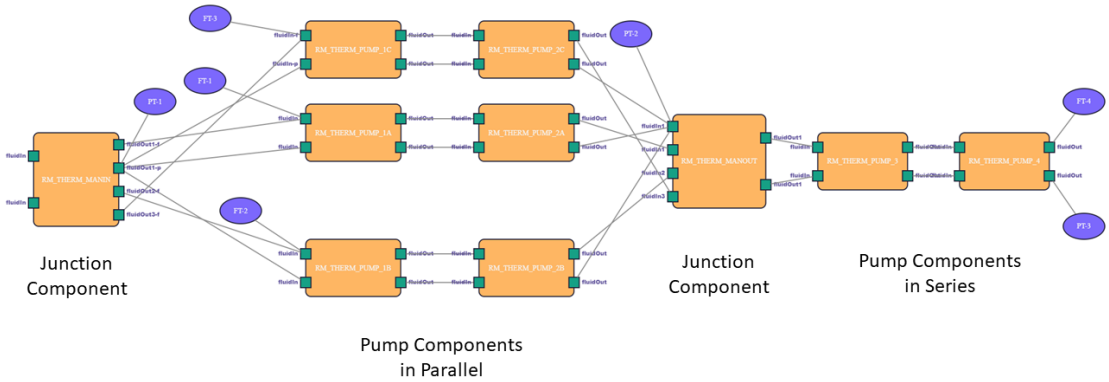

Figure 8. HRS MONSID model generated by the adapter and visualized in Toolkit.

The model generated by the adapter can be loaded into Toolkit to visualize and review its topology.

The Toolkit checks models for unconnected nodes and unattached sensors and displays a list of any such occurrences to the user. For the HRS model, the topology review feature noted that two input nodes on the left junction component were not connected. A screenshot of the model opened in Toolkit is seen in Figure 9; the two unconnected nodes are listed in the review pane located on the right side of the page.

While the adapter generated a nearly complete MONSID model, some manual manipulation was needed in order to perform meaningful ambiguity group analysis. A snapshot of the model in the Analyze page is shown in Figure 10**.** This page displays the current state of ambiguity group determination, including a list of items that were not considered in the calculation because of topological issues (bottom left pane in the figure). In this case, the left junction component and, by extension, the pressure sensor attached to its second from the top output node (appearing in lighter gray in the figure) are not considered because of the two unconnected input nodes previously mentioned. Note that intentionally leaving items out of the ambiguity group calculations allows some analysis of partial or incomplete models to be performed. This is particularly useful in the development cycle when models are under construction.

are same shade of gray). Second, new ambiguity groups appear on the left side. There are two groups with more than a single member, one with 11 members shown with teal highlights and one with 3 members. The 3-member group includes the left junction component and the two new sensors.

The larger 11-member group is a result of the lack of sensors in between components. This is clearly undesirable from an FM perspective because a fault occurring in that group cannot be further narrowed down to any specific members. The diagnostic resolution would be improved if there are more ambiguity groups with fewer members in each group. The ideal ambiguity group is composed of a single member. Sensors located between components form an ambiguity group with themselves and so are single member groups. The HRS model naturally has four such groups, the PT-2, FT-1, FT-2, and FT-3 sensors.

There are a few ways to reduce the 11-member group. One way would be to add three more sensors in between the output flow nodes of the parallel pumps and the input nodes of the right junction component as shown in Figure 12. This action splits the 11-member group into 8 smaller groups. The pumps in series are now in a separate group. Each of the three branches of the parallel pump configuration is now a separate group consisting of two members each. The pumps in series are in a 5-member group with the right junction component and two sensors. Finally, the three additional flow sensors are single members of their own group. Another way would be

to add sensors to the output nodes of the right junction component. This costs less additional sensors than the previous option but puts the parallel pumps into a 7-member group with the right junction component. These are just examples of the kinds of sensor placement trades that can be performed.

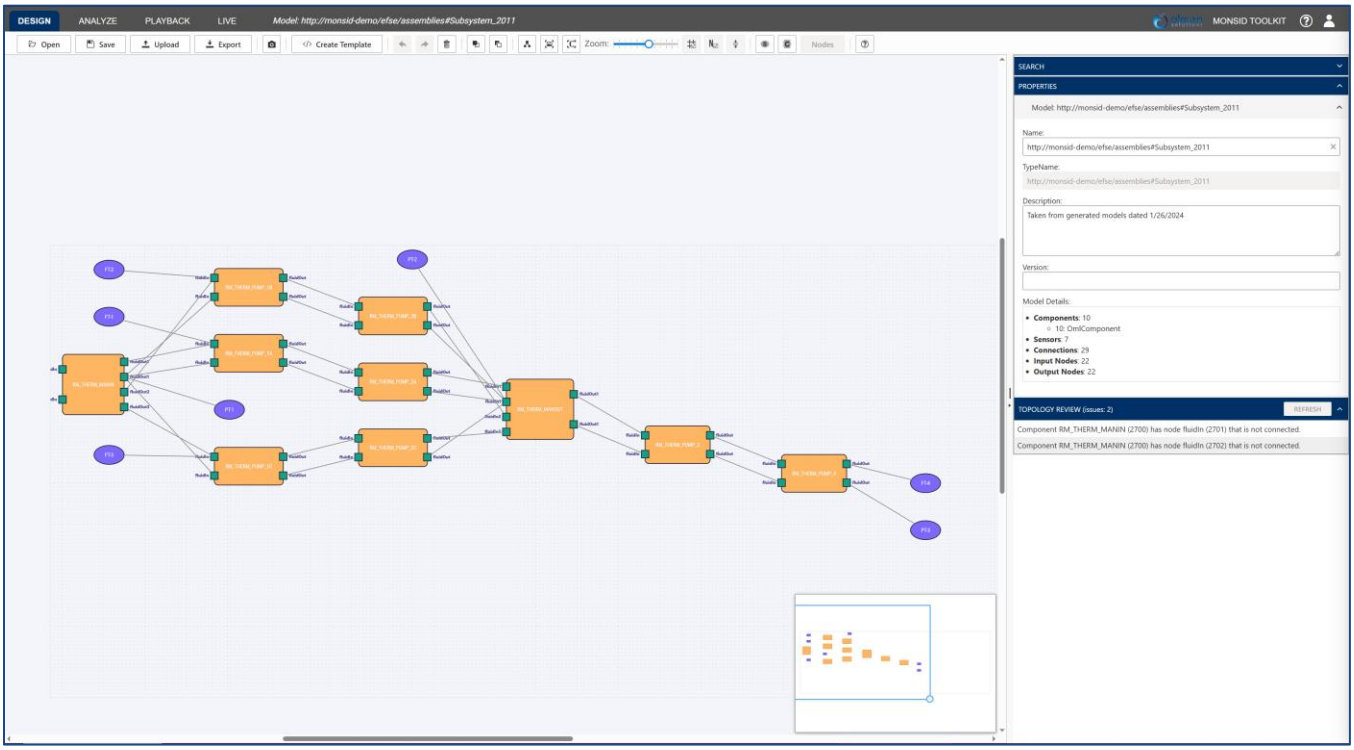

Figure 9. Screenshot of the adapter-generated model opened in Toolkit. The topology review panel shown on the right side of the page alerts the user to the two unconnected node interfaces.

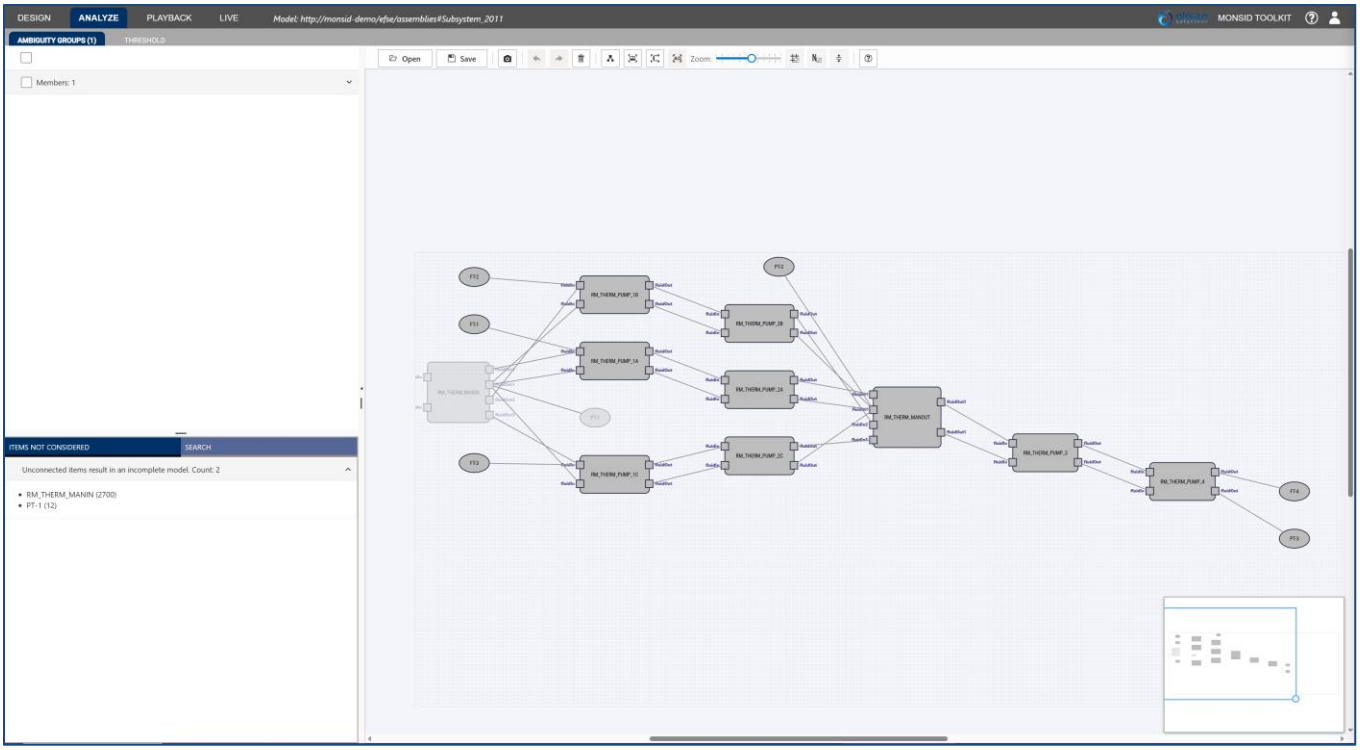

Figure 10. Screenshot of the model displayed in the Analyze page. Items not considered in ambiguity group calculation due to topology issues are listed in the pane on the bottom left. Item not considered are shown in a lighter shade of gray.

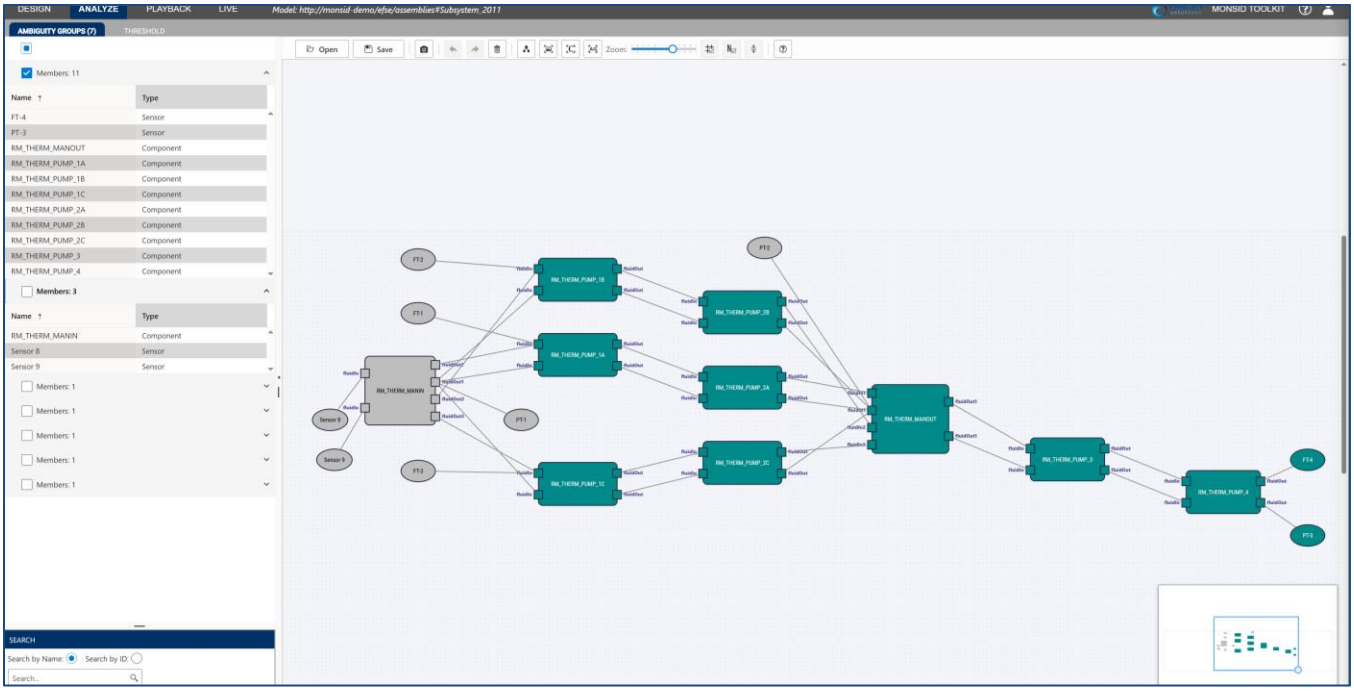

Figure 11. Screenshot of the model in Analyze page after sensors (sensor 8 and 9) are added to each input on the left junction component. New ambiguity groups and their members are listed on the left. The largest group with 11 members is highlighted in teal. All model items are now considered in the ambiguity calculations (the Items not Considered list is empty and is not displayed).

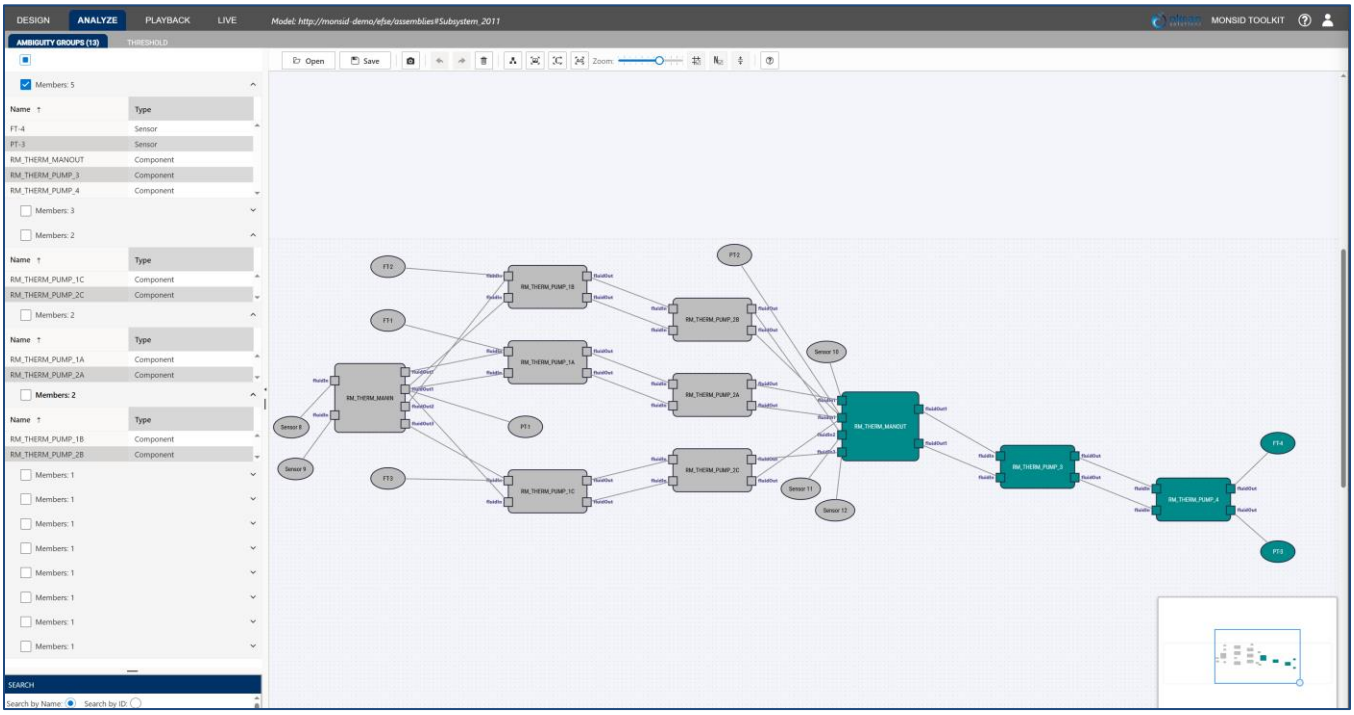

Figure 12. Screenshot of the model in Analyze after sensors 10, 11, and 12 are added to inputs on the right junction pipe. Additional ambiguity groups with fewer members are created thus improving diagnosability.

5. CONCLUSIONS AND FUTURE WORK

The MONSID adapter to CAESAR enables faster diagnostic model generation with less errors compared to a documents-based and hand-coding process.

This approach also constitutes a significant step towards integrating FM with nominal flight systems engineering. Model generation can occur as the CAESAR model is being populated and thus can start earlier in the project life-cycle. Since the adapter is effectively part of the CAESAR toolchain, MONSID models evolve concurrently with other systems engineering activities including harness design and flight software development.

Post-database only transformation requires a lot of project specific information. It is viable but not scalable nor a very reusable solution. The problem lies in that the transformation logic becomes deeply embedded into multiple places, which is the antithesis of leveraging OML as the authoritative source of truth. Adding a pre-database component makes transformations more generic as the reasoner can apply a mapping vocabulary which allows the post-database portion to be more application-agnostic. The use of mapping vocabularies in the pre-database portion leverages the power of the reasoner to validate the semantic structure of the model. Hierarchical organization of the mapping vocabulary into upper-level and project-level allows re-use of the upper-level across projects within an organization (i.e. common vocabulary mapping that is project-agnostic) while allowing project-specific mappings and definitions to be constrained to the project-level.

The design of the MONSID-CAESAR adapter utilized the assumption that components in the source model have been specifically typed to accurately imply their behavior. Currently, CAESAR models for flight projects are primarily used to map interfaces and types are applied to define canonical interfaces. Here we assumed that this can also imply behavior. Whether this is sufficiently precise to enable useful fault analysis is the subject of ongoing investigation.

The benefits of a MONSID-CAESAR adapter were effectively demonstrated on the simplified HRS example system with less than 40 of any one topology element. We expect it to scale well for more complex flight systems. The adapter is particularly suited for real-world, larger systems that may have hundreds to thousands of elements. In the combined power and command & data handling subsystems on JPL's Mars Sample Return Lander (SRL) flight program, for example, there are upwards of one hundred assemblies, over two thousand interfaces, and thousands of sensors. The time consuming and error-prone tasks of manually making and keeping track of interface connections are eliminated with the adapter.

The next step in the adapter development is to apply it to a more complex CAESAR model utilized on an actual flight program. To that end, work is in progress to extend the adapter with the intent to demonstrate it on the CAESAR model of JPL's SRL flight program. Initial focus is on the SRL power system because it is the most populated. A goal of this effort will be to compare the results of diagnosis resolution analysis (also known as ambiguity group analysis) on the SRL MONSID model to fault containment regions specified in the CAESAR model.

ACKNOWLEDGEMENTS

This work was sponsored in part by a NASA Phase II SBIR contract under the Jet Propulsion Laboratory[2], California Institute of Technology.

REFERENCES

Avizienis, A., (1997). "Toward systematic design of fault-tolerant systems," *Computer*, vol. 30, no. 4, pp. 51-58, April 1997, doi: 10.1109/2.585154.

Chau, S., Alkalai, L., & Tai, A. T., (2000). "Analysis of Multi-Layer Fault Tolerant COTS Architecture for Deep Space Missions," *Symposium on Application-Specific Systems and Software Engineering and Technology*, Mar 24.

Elaasar, M., Rouquette, N., Wagner, D., Oakes, B., Hamou-Lhadj, A., & Hamdaqa, M., (2023). "openCAESAR: Balancing Agility and Rigor in Model-Based Systems Engineering", *Proceedings of SAM 2023*, Oct., Västerås, Sweden.

Kolcio, K., (2016). "Model-Based Fault Detection and Isolation System for Increased Autonomy", *AIAA SPACE 2016, AIAA SPACE Forum*, (AIAA 2016-5225), Sept 13-16 , Long Beach, CA.

Kolcio, K., Fesq, L. M., & Mackey, R., (2017). "Model-Based Approach to Rover Health Assessment for Increased Productivity", *IEEE Aerospace Conference*, Mar 5-10, Big Sky, MT.

Kolcio, K., Fesq, L. M., & Mackey, R., (2019). "Model-Based Approach to Rover Health Assessment - Mars Yard Discoveries", *IEEE Aerospace Conference*, Mar 2-9, Big Sky, MT.

Kolcio; K., & Prather, M., (2023). "Implementation and Evaluation of Model-based Health Assessment for Spacecraft Autonomy", *IEEE Aerospace Conference*, March 4-11, Big Sky, MT.

Mackey, R., Nikora, A., Fesq, L. M., & Kolcio, K., (2021). "On-Board Model Based Fault Diagnosis for CubeSat

[2] Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not constitute or imply its endorsement by the United States Government or the Jet Propulsion Laboratory, California Institute of Technology

Attitude Control Subsystem: Flight Data Results", IEEE Aerospace Conference, Mar 6-13, Big Sky, MT.

NASA Fault Management Handbook, (2012). (NASA-HDBK-1002), April 2.

Wagner, D., Kim-Castet, S. Y., Jimenez, A., Elaasar, M., Rouquette, N., & Jenkins, S., (2020). "CAESAR Model Based Approach to Harness Design", *IEEE Aerospace Conference*, Big Sky, MT, doi: 10.1109/AERO47225.2020.9172630.

BIOGRAPHIES

Ksenia Kolcio is the President of Okean Solutions, an SBC located in Seattle, WA, where she leads aerospace systems engineering technical activities. Ksenia has been the PI on several SBIR Phase I, II, and III programs with the DoD and NASA focusing on model-based fault management solutions that strive to increase spacecraft autonomy. Ksenia has also worked on other NASA Phase and DoD contracts in the areas of autonomy architecture development, spacecraft attitude control and navigation mission analysis. Prior to co-founding Okean Solutions, she was employed at Microcosm, Inc. as a Senior Systems/GN&C Engineer where she led and managed GN&C hardware & software development, CONOPS, and systems analysis. She started her career in aerospace at Northrop Grumman where she worked on a variety of NASA and DoD flight programs spanning proposal development to Integration and Test. Ksenia has a BS in Electrical Engineering from the University of Toronto and a Ph.D. in Electrical Engineering from University of Cincinnati. She is a senior member of AIAA and a member of the AIAA Intelligent Systems Technical Committee.

Maurice Prather is the co-owner of Okean Solutions, Inc. and currently serves as the company Vice President. Maurice received a BS in Aerospace Engineering and an MS in Mechanical Engineering from the University of Alabama. Maurice has spent over 25 years in the aerospace and software industries. He currently works as a system architect, developer and trainer, including lead development architect/manager for the MONSID fault management system. Maurice started his career as an aerospace engineer working on various software programs in Boeing's commercial airplane division in the noise and flight operations areas. Afterwards, he worked for Microsoft as a software design engineer on one of Microsoft's most popular business product lines. Maurice has worked as an IT consultant providing architectural guidance, leading development teams, and building custom applications for a large number of corporate and government clients. He is a member of AIAA.

David Wagner manages both the System Modeling and Methodology group as well as product development on the CAESAR project at the Jet Propulsion Laboratory. He holds a bachelor's degree in Aerospace Engineering from the University of Cincinnati, a master's degree in Aerospace from the University of Southern California, and over forty years of experience in systems engineering large and small projects at JPL.

Maged Elaasar is a senior software architect at the Jet Propulsion Laboratory (California Institute of Technology/NASA), where he technically leads the Integrated Model Centric Engineering program and the CAESAR product. Prior to that, he was a senior software architect at IBM, where he led the Rational Software Architect family of modeling tools. He holds a Ph.D. in Electrical and Computer Engineering and M.Sc. in Computer Science from Carleton University, (2012, 2003), and a B.Sc. in Computer Science from American University in Cairo (1996). He has received 12 U.S. patents and authored over 30 peer-reviewed journal and conference articles. He is a regular contributor to and leader of modeling standards at the Object Management Group like UML and SysML. Maged is also the founder of Modelware Solutions, a software consultancy and training company with international clients. He is also a lecturer in the department of Computer Science at the University of California Los Angeles. His research interests span model-based engineering, semantic web, big data analytics, and cloud-computing.

Narek Rouben Shougarian is a Systems Engineer in the Project Systems Engineering Group for Earth and Europa at the Jet Propulsion Laboratory, where he has worked as part of the Europa Clipper Project System Engineering Team since 2017. He is responsible for the Probabilistic Risk Assessments on Clipper and is on systems engineering leadership team for the system level EMI/EMC test. He holds a MEng degree in Aeronautical Engineering from Imperial College London and a PhD in Aerospace Systems from the Massachusetts Institute of Technology.