Verifying Interfaces and Generating Interface Control Documents for the Alignment and Phasing Subsystem of the Thirty Meter Telescope from a System Model in SysML

Sebastian J. I. Herzig[a], Robert Karban[a], Gary Brack[a], Scott B. Michaels[b], Frank Dekens[a], and Mitchell Troy[a]

[a]Jet Propulsion Laboratory, California Institute of Technology, Pasadena, CA, USA
[b]Thirty Meter Telescope, Pasadena, CA, USA

ABSTRACT

This paper presents a novel method for verifying interfaces and generating interface control documents (ICDs) from a system model in SysML™. In systems and software engineering, ICDs are key artifacts that specify the interface(s) to a system or subsystem, and are used to control the documentation of these interfaces. ICDs enable independent teams to develop connecting systems that use the specified interfaces. In the context of the Thirty Meter Telescope (TMT), interface control documents also act as contracts for delivered subsystems. The Alignment and Phasing system (APS) is one such subsystem. APS is required to implement a particular interface, and formulates requirements for the interfaces to be provided by other components of TMT that interface with APS. As the design of APS matures, these interfaces are frequently refined, making it necessary for related ICDs to be updated. In current systems engineering practice, ICDs are maintained manually. This manual maintenance can lead to a loss in integrity and accuracy of the documents over time, resulting in the documents no longer reflecting the actual state of the interfaces of a system. We show how a system model in SysML™ can be used to generate ICDs automatically. The method is demonstrated through application to interface control documents pertaining to APS. Specifically, we apply the method to the interface of APS to the primary mirror control system (M1CS) and of APS to the Telescope Control System (TCS). We evaluate the newly introduced method through application to two case studies.

Keywords: Interface Control Documents, Model-based Systems Engineering, SysML, TMT, APS

1. INTRODUCTION

The Thirty Meter Telescope (TMT) is a proposed astronomical observatory with an extremely large telescope. Its subsystems are developed by numerous institutions and organizations from around the world. To coordinate this largely independent development of the subsystems, interfaces between subsystems are actively managed and have been specified early in the design process. *Interface Control Documents* (ICD) can specify software, mechanical, optical, power, thermal, and other interfaces. Interfaces may evolve as the design is refined. A challenge is to verify whether the system under design remains in compliance with the specified interfaces, and detect any non-compliance as early as possible. Any discrepancy may require the interface control document, or the system specification to be updated. This paper takes one step towards streamlining interface verification and ICD generation, and introduces and evaluates a method for deriving the software interfaces of the Alignment and Phasing System (APS) to its interfacing subsystems through static analysis of the specified behavior of APS. A comparison of these derived interfaces to the change controlled ICDs (which act as the authoritative source for interface specifications) yields a basis for verification.

In current systems engineering practice, verifying compliance to interfaces is largely a manual effort, involving inspection, testing and reviews. TMT has developed an ICD database and management system,[1] in which interfaces are stored in a machine readable form. Several tools are available for processing and validating the interface specifications. However, the interface specifications still have to be produced by some external entity -

Further author information:
S. J. I. Herzig: E-mail: sebastian.j.herzig@jpl.nasa.gov, Telephone: +1 818 393 3059

e.g., by a systems engineering team - and adherence of the designed (sub-)system to the interface specifications has to be ensured manually.

The APS systems engineering team is following the Model-based Systems Engineering paradigm, and is developing a system model of TMT and APS in SysML^{TM}. This system model contains requirements, behavioral specifications, structural specifications, and relationships between these aspects. While SysML^{TM} also offers constructs for defining interfaces, these interface specifications cannot formally be connected to the behavior of the component realizing or using the interface. Therefore, their use is limited, requiring careful manual inspection to ensure compliance to specified interfaces. Part of the structural and behavioral specification are the definition of interactions between the various subsystems and their components. In the TMT / APS model, the software interactions are modeled using a *Signal* processing mechanism, in which various types of Signals are sent and received by components, triggering changes in their state. In the approach introduced in this paper, we derive software interfaces from these specified interactions. The working hypothesis is that sufficient information is contained in, and can be extracted from the system model in order to recreate an ICD. Given the validity of the hypothesis, a mechanism could be put into place for automatically updating ICDs based on the latest behavior specifications. Verification activities could be facilitated through comparison of the derived ICD to a change controlled, agreed-upon ICD (i.e., an authoritative, externally managed ICD). Clearly, such a comparison can reveal discrepancies and non-conformance, and check completeness of the current interface specification.

The remainder of this paper is structured as follows: in section 2 we briefly introduce the TMT system model, and review the state of practice in interface management with SysML^{TM}. This is followed by an introduction to our approach in section 3. Section 4 introduces methods for generating interface control documents from the extracted information. The approach is applied to two example interfacing subsystems in section 5. Results are analyzed, and key insights discussed in detail. The paper closes with a summary of the most important findings, conclusions and suggestions for future work.

2. BACKGROUND & RELATED WORK

Before introducing our approach, a brief introduction to the TMT / APS system model is provided in this section. Also introduced are the constituents of an interface control document, and the state-of-the-art of interface management with UML / SysML^{TM}.

2.1 The Thirty Meter Telescope System Model

The Thirty Meter Telescope (TMT),[2] under development by the TMT International Observatory (TIO), applies a hybrid systems engineering approach leveraging both traditional systems engineering and model-based techniques to perform the systems engineering tasks such as requirements management, technical resource management, and design management and analysis. The main objective of MBSE for the TMT is to capture operational scenarios and demonstrate that requirements are satisfied by the design. For this purpose, a SysML^{TM} model is created to better understand and communicate system behavior, motivated by optimizing the system design.

The Jet Propulsion Laboratory (JPL) participates in the design and development of several subsystems of TMT and delivers the complete APS. The TMT International Observatory (TIO) is the customer, which provides the APS team with requirements, and JPL delivers an operational realization of the system. The APS team pursues an MBSE approach to analyze the requirements, come up with an architecture design and eventually an implementation. Central to the implementation of MBSE on APS is the development of a system model in SysML^{TM}. This system model integrates requirements, structure, behavior, and parametric relations. Its development is driven by operational scenarios, the integration of each of which refines the behavioral and structural specification of APS.

Both the structure and behavior of APS is captured in the system model. The structure includes the encapsulation of properties of a system, and the decomposition of a system by either functional or physical boundaries. Interfaces formally capture any and all exchange of information, energy or matter between components. The captured behavior of each component specifies how the state and properties of a component change, and when and how components interact. Requirements are imposed on both structure and behavior.[3–5] In addition to APS and its components, TMT subsystems interfacing with APS are also modeled (see Figure 1). The behavioral

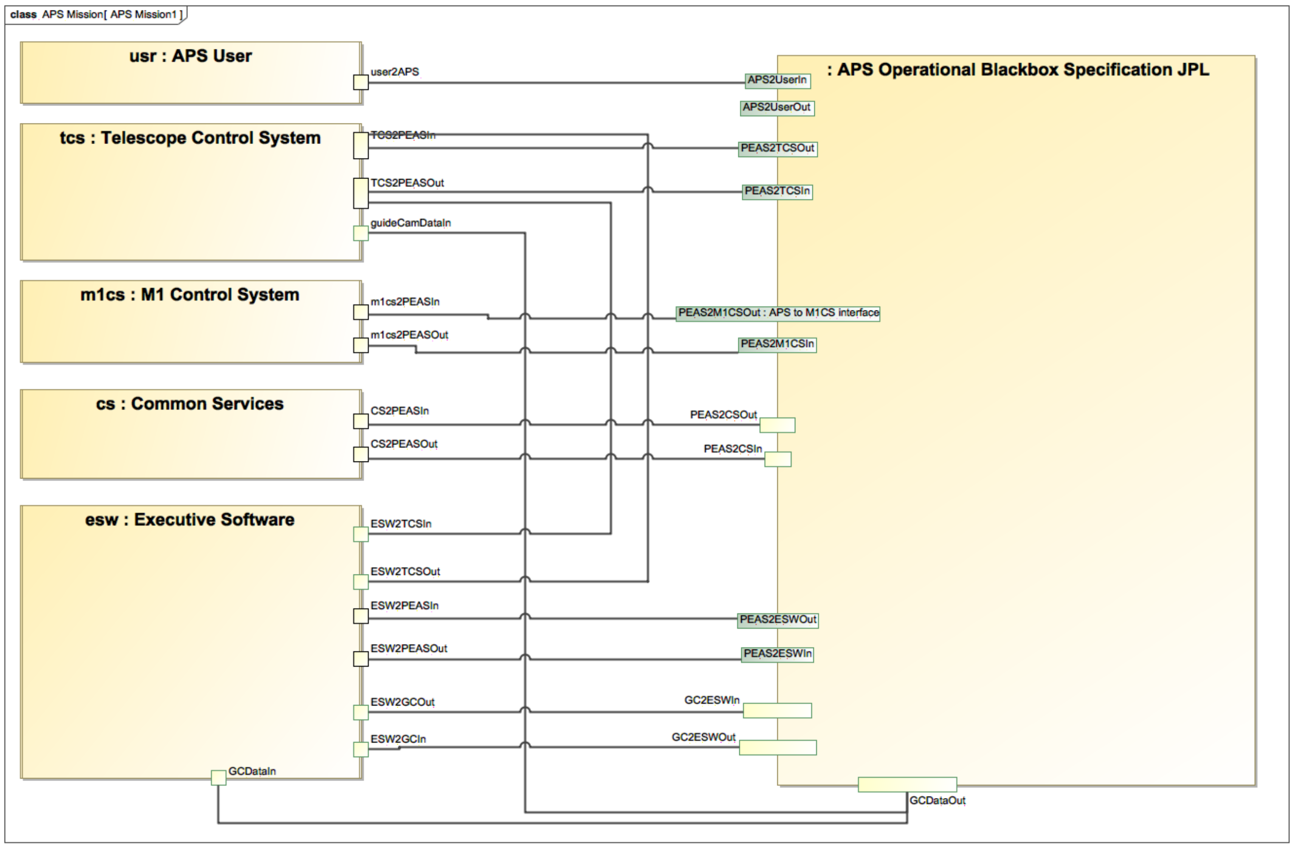


Figure 1. APS and TMT subsystems interfacing with APS (software interfaces).

specification of these interfacing components is limited to reactions that are observable by an external entity upon receiving appropriate stimuli. The behavior of these interfacing subsystems is based on heritage designs, and is regularly reviewed by representatives from the responsible teams (during reviews of the implementation of newly integrated operational scenarios). Ports and connectors between ports signify the existence of interfaces between APS and the subsystems.

By having adhered to the *Executable Systems Engineering Method* (ESEM),[5] and through use of appropriate tools, the TMT / APS SysML model is executable (i.e., it is built for simulation and analysis purposes). ESEM prescribes the functional and physical decomposition of the system into a nested tree of components, as well as the specification of the behavior of each. Requirements are formally captured through manual translation to mathematical constraints and the binding of any variables in these to behavioral or structural properties of the system. This allows for analytic links between requirements, use cases, system decomposition, system behavior and subsystem relationships to be captured. A number of (first-order) analyses can be performed automatically, including the verification of requirements on power, mass, timing, and optical and pointing errors. To facilitate requirements verification, we have modeled a number of operational scenarios of APS using SysML$^{\text{TM}}$. The system model is available for download under an open source license [*].

2.2 Elements of a Software Interface Control Document

An ICD is a document that describe the interface(s) to a system or subsystem. It may describe the inputs and outputs of a single system or the interface between two systems or subsystems. An ICD should only describe the interface itself, and not the characteristics of the systems which realize or use it. In general, an ICD does

[*]Available at http://www.github.com/Open-MBEE/TMT-SysML-Model

not have to be a textual document, and can come in many forms. For instance, an application programming interface (API) is a form of ICD. Interface descriptions can be created for software, optical, mechanical, thermal, and other interactions. Here, we focus on the content of software ICDs as used by TMT.

TMT has developed an ICD management system, in which certain information pertaining to software interfaces between subsystems is stored in HOCON notation (a superset of JSON) using a TMT-specific schema. From these definitions, document-based ICDs can be directly generated. Typically, these documents describe the interface between two systems. However, the interface information in the TMT database defines the inputs and outputs to *all* interfacing subsystems, allowing for the generation of artifacts that describe the interface(s) of just one subsystem.

In general, the software interface definitions in TMT ICDs describe *commands* that can be invoked or are supported, and *events* that components publish or can subscribe to. Therefore, a TMT ICD that defines the interface between two components A and B contains the following elements:

- Commands sent from A to B
- Commands sent from B to A
- Telemetry events published by B that A subscribes to
- Telemetry events published by A that B subscribes to

For each command, the following information is specified:

- Name of command
- Short description
- Detailed description
- Data format (the returned parameters / return type)
- Data units (the input parameters to the command)

For each input parameter (or "data unit"), as well as for the returned parameters / return type (or "data format") the following information is provided:

- Argument (parameter name)
- Data format (the type; e.g., *float*)
- Units (physical units; e.g., m (meters))
- Range (valid range of values for input parameter)
- Comments

In addition to the above, the return parameter specification also defines a *maximum* and a *typical time* that a command is expected to take to execute. Other, related documents, prescribe the communication protocols and technical standards to be used in the implementation.

TMT ICDs also describe the telemetry events published by a component, and those that a component subscribes to. The specification of these events includes:

- Name
- Frequency

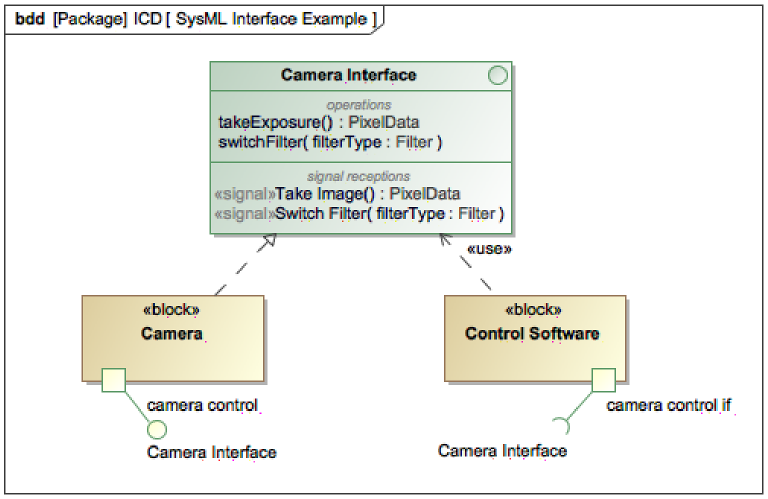

bdd [Package] ICD [SysML Interface Example]
Camera Interface
operations
takeExposure() : PixelData
switchFilter(filterType : Filter)
signal receptions
«signal»Take Image() : PixelData
«signal»Switch Filter(filterType : Filter)
«use»
«block»
Camera
«block»
Control Software
camera control
Camera Interface
camera control if
Camera Interface

Figure 2. Hypothetical interface specification in SysML^{TM} and Blocks that use and realize the interface.

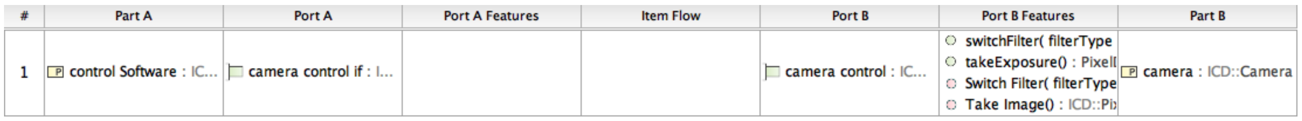
- Comment / description
- Attributes

Similar to parameters of commands, the specification of attributes of events include a specification of their types, units, value ranges, and feature a short description.

2.3 State of the Art of Interface Management with UML / SysML

SysML^{TM}, and more generally the *Unified Modeling Language* (UML) which SysML^{TM} is based on, define language constructs for defining interfaces. An example of such an interface definition is illustrated in Figure 2. For software interfaces, it is typical to define operations and signal receptions. In UML and SysML^{TM}, interfaces are virtual constructs that are *realized* or *used* by one or more components. Different symbols are used to distinguish between these relationships. For instance, in Figure 2, *Camera* realizes the *Camera Interface*, while *Control Software* simply *uses* the interface. This can be shown both by the different kinds of arrows pointing to the interface, and by the "ball and socket" notation.

Unfortunately, the UML and SysML^{TM} specifications are ambiguous about how a realizing classifier is expected to realize the features of its interfaces. The UML 2.4.1 specification stated that *"'for behavioral features, the implementing classifier will have an operation or reception for every operation or reception, respectively, defined by the interface."'* (7.3.25, semantics).[7] The current version, 2.5.1, has weakened this statement slightly, stating that *"'BehavioredClassifiers shall provide a public faade consisting of attributes, Operations, and externally observable Behavior that conforms to the Interface."'* (10.4.3 Semantics).[8] What this conformance consitutes is not unambiguously defined. The specification merely states that *"'if an Interface declares an attribute, this does not necessarily mean that the realizing BehavioredClassifier will necessarily have such an attribute in its implementation, but only that it will appear so to external observers."'*. Also stated is that *"'for BehavioralFeatures, the implementing BehavioredClassifier will have an Operation or Reception for every Operation or Reception, respectively, defined by the Interface"'*, a statement that can also be found in the SysML^{TM}



#	Part A	Port A	Port A Features	Item Flow	Port B	Port B Features	Part B
1	control Software : IC...	camera control if : I...			camera control : IC...	switchFilter(filterType takeExposure() : PixelI Switch Filter(filterType Take Image() : ICD::Pi	camera : ICD::Camera

Figure 3. Example of a tool-specific (here: NoMagic MagicDraw) ICD table.[6]

literature.[9] This implies having to create an operation and signal reception for every operation and signal reception defined by the interface. However, the formal relation between the realized feature and the declared feature is not further defined. Furthermore, no statement is provided about the relationship between an interface and the behavior of a BehavioredClassifier that is *using* the interface (such as the *Control Software* in Figure 2. This makes it unclear how, e.g., the operations specified by an interface can be invoked in behavioral specifications of the classifier relying on the interface.

These ambiguities have resulted in a lack of support for verifying the specified behavior of a component against its specified interface. Current tool-specific extensions for interface management are limited to and based on the declared interfaces, rather than the specified behavior. For instance, NoMagic MagicDraw defines "black box" and "white box" interface tables (see Figure 3), which can be created for any two components that are connected through ports.[6] The "white box" tables contain the features of the connecting interfaces. While this allows for producing a summary view of the essential content typically found in an ICD, it still requires manual verification and "double bookkeeping", since the implementation of the interfaces must be carefully analyzed manually to ensure conformance to the specified interface. Therefore, no guarantee can be made about the conformance to, and the completeness and correctness of the interface with respect to the specified behavior.

3. EXTRACTING SOFTWARE INTERFACES THROUGH STATIC BEHAVIOR ANALYSIS

Here, we introduce an approach to *deriving* software interfaces from the specified behavior of a component using static analysis of the interactions described in the behavioral specifications. The derived information can be used as a basis for comparison and verification, and for the purpose of generating artifacts such as Interface Control Documents.

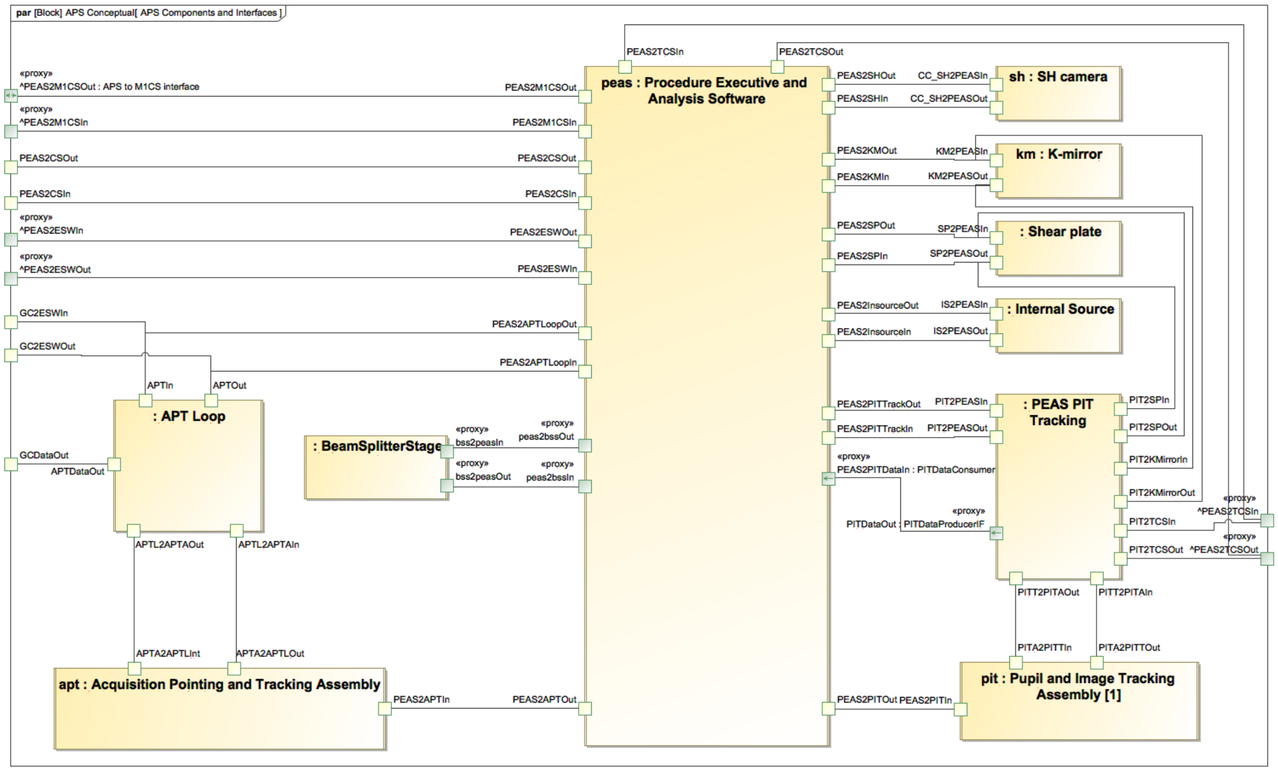


Figure 4. Conceptual model: interfaces, and components of APS with their respective APS-internal interfaces.

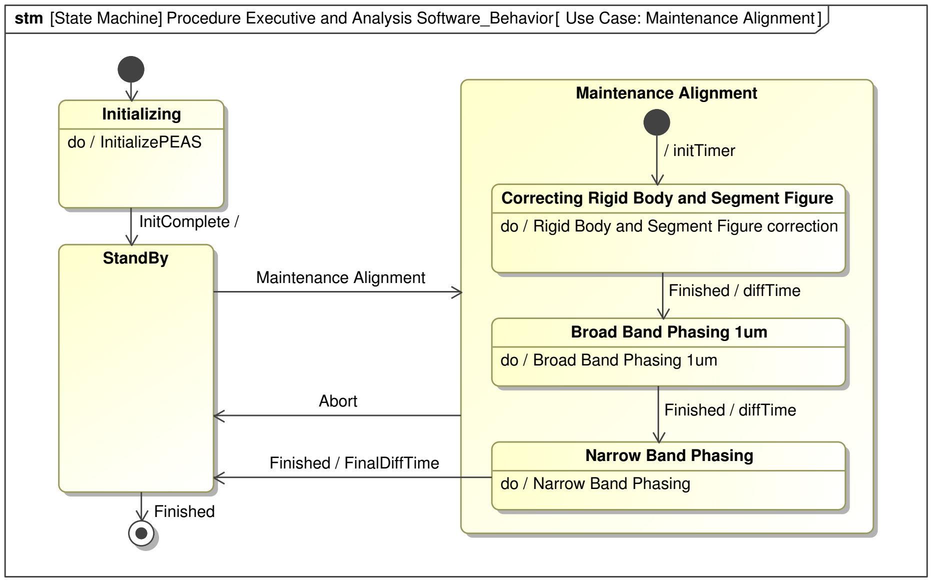


Figure 5. Excerpt of the specification of the *Procedure Executive Analysis Software* (PEAS) behavior as a state machine.

3.1 Modeling Subsystem Interactions using Signals

As stated in section 2, the specification of the operational behavior of APS and its components is driven by operational scenarios.[3] Many of these operational scenarios define interactions among subsystems of TMT and their components. For instance, mirror alignments performed by APS may require the primary, secondary, or tertiary mirror to be moved. The movement of the mirrors is controlled by other subsystems of TMT outside the authoritative domain of APS. Interfaces are negotiated between components to allow for commands to be invoked, or telemetry data to be received from other subsystems. Throughout the TMT / APS system model, such interactions among components and subsystems are modeled using a message passing mechanism (using SysML^{TM} Signals). It is these interactions that we use as a basis for deriving interface specifications.

To illustrate this message-passing-based interaction mechanism, consider interactions between internal components of APS (see Figure 4). In most cases, the *Procedure Executive Analysis Software* (PEAS) plays a central

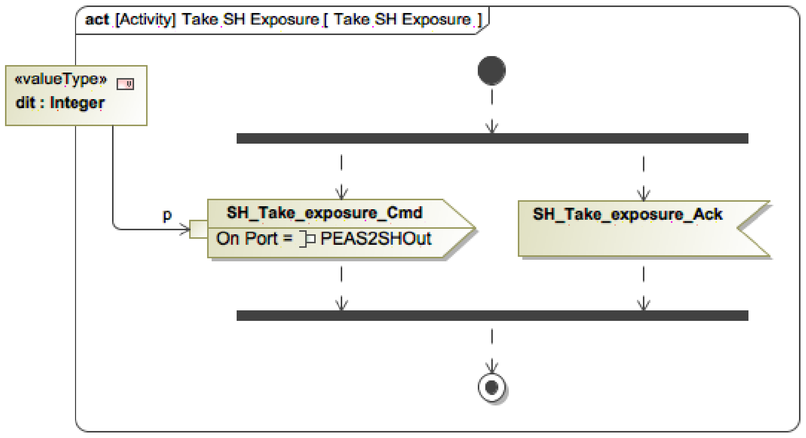


Figure 6. Interaction with Shack-Hartmann Camera: *Take SH Exposure* activity.

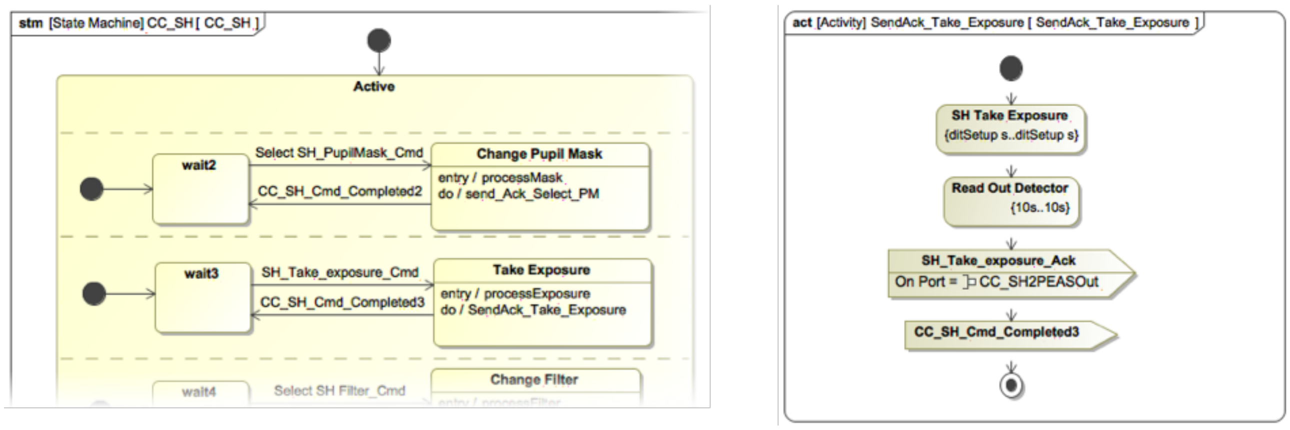


Figure 7. Specified behavior of SH filter (partial): (a) lifecycle behavior and (b) taking exposure.

role. Figure 5 illustrates the specified behavior of PEAS during the operational scenario *Maintenance Alignment*. After initialization, PEAS is in a `StandBy` state. An external signal (provided, perhaps, by an operator) named `Maintenance Alignment` triggers PEAS to transition to the *Maintenance Alignment* state. PEAS is specified to only be able to leave this state if an `Abort` signal is provided (internally or externally). Otherwise it proceeds with first correcting the rigid body and segment figure, then performs broad band phasing and finally narrow band phasing. The "do" behavior of each state specifies the behavior of PEAS when in that particular state. Once completed, PEAS returns to its `StandBy` state.

Figure 6 illustrates part of the "do" behavior of PEAS. In the behavior, a signal `SH_Take_exposure_Cmd` is specified to be sent over the port `PEAS2SHOut` of PEAS (see Figure 4). Payload data is sent along with the signal (here: *dit*, which is the exposure time). An excerpt of the state machine defining the lifecycle behavior of the Shack-Hartmann (SH) Camera is shown in Figure 7 (left). If the SH camera is currently not in a state in which it is taking an exposure, receiving the signal `SH_Take_exposure_Cmd` leads to a transition into the *Take Exposure* state. This, in turn, triggers the "do" behavior *SendAck_Take_Exposure* (see Figure 7 (right)), in which an exposure is taken, the detector is read out, and the results are sent back. PEAS is waiting for an acknowledgement by the SH Camera to have executed the command `SH_Take_exposure_Ack` successfully. As shown in Figure 7 (right), this acknowledgement is sent over the port `CC_SH2PEASOut` which connects the SH camera to PEAS.

3.2 Deriving Software Interface through Static Analysis of System Behavior

In the TMT / APS system model, we formally declare the existence of interactions among components or subsystems (modeled using *Blocks*) using *Ports* and *Connectors*. In SysML^{TM}, *Signals* can be broadcast or sent over specific ports, and can have payload data attached to them. Actions with the semantics of sending (e.g., *Send Signal Actions*) and waiting for the receipt of signals (e.g., *Triggers*) are native to SysML^{TM}, allowing for modeling of the invocation of remote behavior, and the communication between components through the sending and receiving of messages.

The sending of a Signal in one component, and its reception in another, can, therefore, be interpreted as the invocation of a command or sending of telemetry data as part of the specified behavior. From this, one can derive that this interaction is intended to be part of the interface description. Repeating this process by parsing all specified behavior in the system model yields the desired interface information. Subsystems / assemblies / components interacting with one another are simply Blocks in the SysML^{TM} model. Interfaces are declared using Ports and Connectors. Commands and Events correspond to Signals, and any of their attributes and parameters to Properties owned by the Signals. Types are simply SysML^{TM} Value Types. The protocol is apparent from the specified behavior. Table 1 summarizes these correspondences between ICD concepts and SysML^{TM} constructs.

While seemingly simple, there are a number of challenges associated with implementing this procedure. Firstly, Signals may be exchanged between components of a Subsystem A and another Subsystem B. This

Table 1. Mappings between elements specified in an ICD and their implementing SysML constructs.

ICD Concept	**SysML Construct**
Subsystem / Assembly / Component	Block
Interface Declaration	Port, Connector
Protocol	State Machine, Activity
Command	Signal
Subscribe Event	Signal
Publish Event	Signal
Parameter / Attribute	Property (of Signal)
Returned Data	Property (of Signal)
Data Type	Value Type, Block

requires tracing the signal from the Port that it was sent over to all possible recipients: for instance, PEAS is a component of APS, which can send a Signal over one of its ports to M1CS or TCS (which are other subsystems of TMT). This requires the signal to be traced from PEAS, to a port of APS, and to the receiving subsystem. In some cases, Signals may be sent to more than one component if a port connects multiple other ports. Secondly, ports may be inherited by super-classes: such is the case with APS, which defines a "Black Box" system, and sub-classes "Conceptual" and "Realization", each of which inherit the interfaces defined by the "Black Box" system. A third challenge may be model organization: in the TMT / APS model, not all behavior intended to be part of that of a component is necessarily *owned* by the component. This may require tracing the execution context through Call Behavior Actions defined in the context of owned behavior of the component. Algorithm 1 summarizes the procedure for deriving software interface information from behavioral specifications in the system model.

```
input  : System model $\mathcal{S}$
         Invoking source component $C_s$
         Target component $C_t$
output: A list sentCommands containing commands invoked by $C_s$ in $C_t$
portsOnSource = Ports on $C_s$ and any of its parts ;
for Port $p$ in portsOnSource do
    if $C_t$ transitively reachable via connectors on $p$ then
        add $p$ to candidatePorts list;
    end
end
sendSignalActionsInScope = SendSignalActions in $\mathcal{S}$ that send signals over a port in candidatePorts ;
for SendSignalAction ssa in sendSignalActionsInScope do
    signalSent = Signal sent by ssa ;
    if $C_t$ defines trigger in a reachable part that reacts to signalSent then
        add signalSent to sentCommands list
    end
end
```

Algorithm 1: Procedure for extracting commands invoked by a component C_s in a component C_t.

Since command invocations, and publishing and subscribing to telemetry events is modeled using the same mechanisms (namely, the sending and receiving of Signals), it is non-trivial, if not impossible, to distinguish between commands and events. This is a current limitation of the algorithmic procedure. Manual post-processing, or more sophisticated analysis of dynamic behavior (e.g., execution traces), may be required if distinguishing between commands and events is important.

4. VERIFYING INTERFACES & GENERATING INTERFACE CONTROL DOCUMENTS

Given the aforementioned procedures for extracting software-interface-related information from specified behavior, one can now transform and serialize this information into different representations. Here, we discuss representing the extracted information in a form suitable as input to the TMT ICD database. We then discuss how the extracted interface descriptions can be compared to change controlled and signed off ICDs for purposes of verification, and discuss two methods of how interface control documents can be generated.

4.1 Generating Input for the TMT ICD Management System

TMT's ICD management system [†] is a collection of software tools for validating, searching, and viewing subsystem APIs and ICDs.[1] Interface descriptions are written in HOCON (or JSON) and conform to TMT-specific JSON schemas (see Table 2 for an excerpt) [‡]. The schema is well-defined and publicly available.

In the TMT database, interface descriptions are stored for each subsystem. Each subsystem is divided into several components. There are five basic file types that make up a subsystem's interface description in the TMT ICD management system:[1]

- **subsystem-model.conf** : exactly one description of the subsystem that this ICD database entry is about (includes a name and description).
- **<component>/component-model.conf** : one or more files describing the components of the subsystem (each one is typically in a separate directory <component>).
- **<component>/command-model.conf** : description of the commands *received* and *sent* by the particular component. Received commands are *invokeable* by other subsystems. Sent commands are commands invoked by the subsystem in other subsystems and include (a) the name of the command, (b) the subsystem the command is invoked in, and (c) the component that the command is invoked in.
- **<component>/publish-model.conf** : a description of the events published by the particular component, including the name, publish frequency, and attributes.
- **<component>/subscribe-model.conf** : a description of the events subscribed to by the particular component, including the name of the event, the publishing subsystem, and the publishing component.

The information contained in these files directly corresponds to the information extracted from the SysML^{TM} model (see Table 1 for a list of correspondences between ICD concepts and elements in the SysML^{TM} model). The subsystems and subsystem-component structure can be directly reproduced from the containment relationships between Blocks in the system model (see Figure 1 and 4). Derived commands and events are traceable from the calling unit to the component the corresponding Signal is received by, thereby providing sufficient information for the *sent* and *received* commands and events to be reconstructed. Table 2 shows a partial example of a (generated) *command-model.conf* file.

4.2 Verifying Conformance of System Model to Software Interface Specification

We have chosen to produce TMT ICD database files from the derived interface information for two reasons: firstly, their well-defined schema makes the produced artifacts machine readable and convenient to process. Secondly, TMT uses the TMT ICD database as an authoritative source for interface descriptions, meaning that one can, in theory, algorithmically compare the derived ICD information to the authoritative (and agreed upon) ICD through comparison of the content of each of the files in an ICD database entry.

Performing a comparison of the generated interface descriptions to those found in the TMT ICD database assumes an equivalent decomposition of the subsystem into components both in the system model and in the TMT

[†]See `https://github.com/tmtsoftware/icd`

[‡]Complete examples available at `https://github.com/tmt-icd`

Table 2. Extract of the generated input for the TMT ICD Management System in HOCON format.

```
subsystem = M1ControlSystem
component = default

send = [
          {
          ...
          }
]

receive = [
          ...
          {
                    // Message / signal name: Set WH Strain Cmd
                    // Target state: Setting WH Strain
                    name = "Set WH Strain Cmd"
                    description = """"""
                    args = [
                        {
                         name = segment
                         description = """"""
                         type = integer
                        }
                        {
                         name = strains
                         description = """"""
                         type = array
                         dimensions: [21]
                         items = {
                                   type = float
                         }
                        }
                    ]
          }
          ...
]
```

ICD database. *Sent* and *Received* commands, and *Published* and *Subscribed* events can be directly compared through finding a corresponding entry (e.g., through name matching), and comparing parameters and attributes. Non-defining information, such as descriptions of commands, events, and other elements can be left out in the comparison. However, other information, such as types, should be included in the comparison.

4.3 Generating ICDs

Representing the interface descriptions derived from the system model in the TMT-specific format allows for TMT ICD management tools to be reused. As described in the previous sections, these tools include document generation tools. Therefore, given the generated TMT ICD database entries, both API documentation and ICD documents can be generated.

In our work, we have also investigated the use of OpenMBEE [§] components, specifically View Editor and

[§]See http://www.openmbee.org

Table 3. Commands sent by APS to M1CS as specified in the APS-M1CS ICD (left column) compared to those extracted from the system model (right column).

APS-M1CS ICD Command	Extracted Command
setCalibCoeff	-
getCalibCoeff	-
offsetSegment	Move Segment PTT Cmd
saveCalibCoeff	-
offloadSensorOffsets	offloadSensorOffsets Cmd
saveSensorReadings	Take Snapshot Cmd
genCalibCoeff	-
calibrateWarpingHarness	Calibrate Warping Harness Cmd
readWHStrain	-
setWHStrain	Set WH Strain Cmd
offsetWHStrain	-
setWHPosition	Move Segment WH Cmd
offsetWHPosition	-
-	Turn WH On Cmd
-	Turn WH Off Cmd

DocGen, for purposes of generating ICDs. To enable this, we have realized the algorithmic procedure described in section 3.2 through a series of OCL (Object Constraint Language[10]) queries, and have used DocGen-specific actions to prototype the generation of a web-based interface description document.

5. APPLICATION TO ALIGNMENT AND PHASING SYSTEM

The procedure for extracting interface descriptions from the system model, as well as procedures for generating TMT ICD database content were implemented using Java. In this section, we discuss the results from applying the aforementioned procedures to verify and generate two ICDs: the interface between APS and the *Primary Mirror Control System* (M1CS), and the interface between APS and the *Telescope Control System* (TCS).

5.1 Application to APS-M1CS Software Interface

Tables 3 and 4 list the commands and events that were extracted from analyzing the specified interactions between APS and M1CS in the system model. As noted previously, a limitation of the algorithmic extraction procedure is its inability to distinguish between commands, and published or subscribed events. The separation was performed manually in a post-processing step.

In the tables, notice the discrepancies between the extracted names of the commands and events (right column) to those specified in the ICD (left column). These discrepancies have made a computational comparison practically impossible, requiring significant post-processing. While some names are similar (e.g., *offloadSensorOffsets* and *offloadSensorOffsets Cmd*), others use synonyms, or are so different that it becomes difficult even for a human to compare without additional knowledge (e.g., *saveSensorReadings* vs. *Take Snapshot Cmd*). In some cases, the extracted name suggests a command invokation, while in actual fact the ICD specifies a telemetry event that is published or subscribed to (e.g., *m1cs.sensorHeights Cmd*).

Interesting to note is also that Table 3 contains two extracted commands (specified to be invoked by APS in M1CS) that the ICD does not contain. Further analysis revealed that these commands (*Turn WH On Cmd* and *Turn WH Off Cmd*) were deprecated in a previous iteration of the APS-M1CS ICD, since it was decided

Table 4. Events published by M1CS and subscribed to by APS as specified in the APS-M1CS ICD (left column) and those extracted from the system model (right column).

M1CS to APS ICD Pub/Sub Event	**Extracted Pub/Sub Event**
m1cs.health	-
m1cs.alarm	-
m1cs.status	-
m1cs.actuatorPositions	-
m1cs.sensorHeights	m1cs.sensorHeights Cmd
m1cs.sensorGaps	-
m1cs.pistonTipTilt	Get Segment WH Pos Cmd
m1cs.servoErrors	-
m1cs.pistonTipTiltTarget	-
m1cs.outerLoopCtrlCmds	-
m1cs.segmentStatus	Get installed_Segment_Query
m1cs.warpingHarnessStrain	Get Segment WH Pos Ack
m1cs.warpingHarnessStatus	-
m1cs.purgeSystemStatus	-
m1cs.ctrlNetworkStatus	-

Table 5. Comparison of parameters specified in the APS-M1CS ICD to those extracted from the system model. Only events / commands that have parameters specified are included. Angled brackets denote vectors (e.g., int [21] is a vector of 21 integers).

Event or Command from ICD	**Parameters in ICD**		**Extracted Parameters**	
	Name	Type	Name	Type
offsetSegment	actuatorOffset	float [492x3]	-	-
saveSensorReadings	type	[ALIGNED, DIAGNOSTIC]	-	-
	metadata	string	-	-
offloadSensorOffsets	segmentLocation	integer	segmentLocation	integer
calibrateWarpingHarness	motor number	integer	motorID	integer
	segment	integer	segment	integer
setWHStrain	segment	integer	segment	integer
	strains	float [21]	strains	float [21]
setWHPosition	segment	integer	p	float
	position	integer [21]	-	-
m1cs.sensorHeights	heights	float [2272]	-	-
m1cs.warpingHarnessStrain	strain	float [492x21]	-	-
m1cs.pistonTipTilt	pistonTipTilt	float [3]	-	-
m1cs.warpingHarnessStrain	strain	float [492x21]	p	float

Table 6. Comparison of commands sent by APS to TCS specified in the APS-TCS ICD to those extracted from the behavioral specification in the system model.

APS-TCS Command from ICD	**Extracted Command**
offsetM2Position	Move M2 PTT Cmd
saveM2Position	Take M2 Snapshot Cmd
loadM2Position	-
offsetM3Position	M3Offset Cmd
offsetPupilLocationWithM3	Move Pupil with M3 Cmd
offsetImageLocationWithM3	ADApparent Offset Cmd
saveM3Position	Take M3 Snapshot Cmd
loadM3Position	-
offsetTelescopePosAzEl	-
offsetTelescopePosAptPixels	Update APT Pixel Offset Cmd
-	StopGuiding Cmd

that controlling the On/Off state of the Warping Harness is to be done automatically as part of executing other Warping Harness related commands. These commands were deeply hidden in the behavior specification, and would have been very difficult to detect otherwise. This alone clearly demonstrated the utility of the approach.

Finally, notice that there are numerous events and commands that are specified in the change controlled ICD, but to which no reference could be found in the system model. This can be an indicator of two things: either the model is incomplete, suggesting that the coverage of the operational scenarios analyzed is not sufficiently complete to cover the complete behavior of APS, or the ICD is over-specified. A third, perhaps harder to diagnose cause, could be that the seemingly unused commands or events are, in fact, referred to in the behavior description, but no message passing mechanism was implemented. This could be a result of a miscommunication, or an oversight - possibly also an abstraction that was deemed appropriate at least at the time of introducing it. Nonetheless, these findings warrant careful review of the specified behavior of APS and corrective actions to align with, and possibly update the current ICDs.

Table 7. Comparison of TCS to APS events specified in the APS-TCS ICD to those extracted from the behavioral specification in the system model.

TCS to APS ICD Pub/Sub Event	**Extracted Pub/Sub Event**
TCS Health Event	-
TCS Alarm Events	-
TCS Status Event	-
TCS Guiding Residual Event	-
M2 Position Event	M2_Status_Position Query
M3 Position Event	-
M2 Status Event	-
M3 Status Event	-
TCS Telescope Position Event	-
TCS Current Star Event	-

Table 8. Comparison of APS to TCS events specified in the APS-TCS ICD to those extracted from the behavioral specification in the system model.

APS to TCS ICD Pub/Sub Event	Extracted Pub/Sub Event
APT Frame Event	GCFrame

Last, but not least, the parameters and attributes specified for each command and event in the APS-M1CS ICD were compared to those extracted from the system model. Recall that parameters and attributes are mapped to Properties of Signals, and their values are considered payload of the messages sent between components. Results from the comparison are illustrated in Table 5. Only commands and events are included where a correspondence between an extracted and a specified command or event was identified, and that have parameters or attributes. The observations that can be made are similar to those made when comparing commands and events. In some cases, the parameters match well, although the naming may slightly differ (e.g., *motor number* vs. *motorID*). There are also a number of instances where attributes and parameters have not been modeled. It should be noted that this was often a conscious decision to reduce complexity of the model, since their inclusion would not have had an effect on the behavior and, hence, system performance. However, for completeness, these parameters should be included. In other cases, the parameters do not match. For instance, consider the command *setWHPosition*: a single parameter named *p* (a floating point number) was extracted from the model, while the ICD specifies two parameters, one of which is an integer, and the other a vector of integers.

5.2 Application to APS-TCS Software Interface

Results from applying the approach to the APS-TCS software interface are captured in Tables 6, 7, 8 and 9. Table 6 lists commands invoked in TCS by APS. Table 7 lists the extracted telemetry events published by TCS and received by APS, and Table 8 lists those events published by APS that are received by TCS. Finally, Table 9 compares the attributes and parameters of commands and events.

The types of observations made for APS-TCS are practically identical to those made for APS-M1CS. The system model specifies APS to invoke a command *StopGuiding Cmd* in TCS that is not specified in the APS-TCS ICD. Names of commands and events are, to a large degree, very different. There are only a few commands that are specified in the ICD, but unused in the model. However, almost none of the telemetry events published by TCS and, according to the ICD, subscribed to by APS, are used in the behavioral specification. This may, again, be an indication of the coverage of use cases not being complete (e.g., *TCS Alarm Events* may only appear in exception handling cases, very few of which have been modeled to date), or an indication of the ICD being over-specified. Different to the APS-M1CS case is that there is a telemetry event published by APS that is received by TCS.

Comparing the attributes and parameters or commands and events lead to similar observations as before (see Table 9). In many cases, either no parameter is specified, or the set of parameters / attributes is incomplete. In some cases, a single parameter appears to lump together two other parameters (e.g., *offset* appears to lump together *az* and *el* in *offsetImageLocationWithM3*. A similar observation can be made about the parameter *windowCoordinates* of the *APT Frame Event*. Finally, note how the coverage of parameters in *APT Frame Event* is fairly complete - yet, the data types of the parameters are not very well aligned.

6. CONCLUSIONS

In this paper, an approach to deriving software interface descriptions from the behavioral specifications in a system model in SysML^{TM} is introduced. The approach is based on a static analysis of the specified behavior, and tracing Signal-based interactions among components and subsystems. In the paper, we show how, using this information, one can verify whether the modeled interactions conform to controlled software interfaces (i.e., those described in ICDs). We also describe how (updated) ICDs can be generated from the derived interface descriptions.

The application to two case studies - namely, the software interfaces between APS and M1CS, and APS and TCS - has revealed a number of discrepancies between the controlled software interfaces and the specified

Table 9. Comparison of parameters specified in the APS-TCS ICD to those extracted from the system model. Only events / commands are included that have parameters specified.

Event or Command from ICD	**Parameters in ICD**		**Extracted Parameters**	
	Name	Type	Name	Type
offsetM2Position	piston	float	-	-
	tip	float	-	-
	tilt	float	-	-
	decenter-x	float	-	-
	decenter-y	float	-	-
offsetM3Position	tip	float	-	-
	tilt	float	-	-
offsetPupilLocationWithM3	x	float	-	-
	y	float	-	-
	correctPointing	boolean	correction	boolean
offsetImageLocationWithM3	az	float	offset	float [2]
	el	float	-	-
	correctPointing	boolean	-	-
offsetTelescopePosAptPixels	x	float	offset	float [2]
	y	float	-	-
	plateScale	float	-	-
APT Frame Event	frameWidth	int	windowCoordinates	float
	frameHeight	int	-	-
	plateScale	float	plateScale	float
	setPoint	int [2]	desiredSetpoint	float
	roiOffset	int [2]	orientation	float
	frameData	int [frameWidth *frameHeight]	pixelData	float

behavior. Most are minor, in that only the names of commands or events are different. Other cases hint towards incomplete coverage of the operational scenarios accounted for in the APS system model, or an over-specification of the controlled software interfaces. In yet other cases it was found that APS was specified to invoke commands in other subsystems that these subsystems do not support (at least according to the interface control documents). We were able to clearly demonstrate the utility of the approach through uncovering these discrepancies, some of which would have been difficult to detect without the help of computational analysis. Detecting these problems early allows for relatively inexpensive corrective action to be taken.

The introduced algorithmic procedure for extracting software interfaces from the behavioral specifications of components in the system model has the caveat that it cannot distinguish between telemetry events and commands. This is primarily due to the practically identical mechanism used in SysML^{TM} to model either mode of communication: Signals are sent and received by different components. This limitation requires manual post-processing of the extracted interface information. Adding additional information to the model (e.g., through an extended vocabulary using stereotypes), or deeper analysis of the behavior (e.g., the dynamic behavior through tracing of the execution) may aid in overcoming this limitation.

Future work should include refining the algorithmic procedure for extracting interface descriptions by investigating possible ways of differentiating between events and commands - for instance, through analyzing the dynamic behavior through following execution traces. Also investigated should be the possibility of extracting the protocol underlying the interactions between two subsystems.

ACKNOWLEDGMENTS

This research was carried out in part at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration. The authors gratefully acknowledge the support of the TMT collaborating institutions. These are the California Institute of Technology, the University of California, the National Astronomical Observatory of Japan, the National Astronomical Observatories of China and their consortium partners, the Department of Science and Technology of India and their supported institutes, and the National Research Council of Canada. This work was also supported in part by the Gordon and Betty Moore Foundation, the Canada Foundation for Innovation, the Ontario Ministry of Research and Innovation, the Natural Sciences and Engineering Research Council of Canada, the British Columbia Knowledge Development Fund, the Association of Canadian Universities for Research in Astronomy (ACURA), the Association of Universities for Research in Astronomy (AURA), the U.S. National Science Foundation, the National Institutes of Natural Sciences of Japan, and the Department of Atomic Energy of India.

REFERENCES

[1] Gillies, K., Roberts, S., Brighton, A., and Rogers, J., "A database for tmt interface control documents," in [*Modeling, Systems Engineering, and Project Management for Astronomy VI*], **9911**, 99112Q, International Society for Optics and Photonics (2016).

[2] Thirty Meter Telescope (TMT), "Overview," (2018).

[3] Herzig, S. J. I., Karban, R., Trancho, G., Dekens, F. G., Jankevičius, N., and Troy, M., "Analyzing the operational behavior of the alignment and phasing system of the thirty meter telescope using sysml," in [*Adaptive Optics for Extremely Large Telescopes (AO4ELT)*], (2017).

[4] Karban, R., Dekens, F. G., Herzig, S., Elaasar, M., and Jankevicius, N., "Creating system engineering products with executable models in a model based engineering environment," *Modeling, Systems Engineering, and Project Management for Astronomy VI, SPIE, Edinburgh, UK* (2016).

[5] Karban, R., Jankevičius, N., and Elaasar, M., "Esem: Automated systems analysis using executable sysml modeling patterns," in [*INCOSE International Symposium*], **26**(1), 1–24, Wiley Online Library (2016).

[6] NoMagic, "Magicdraw," (2018).

[7] Object Management Group, "The Unified Modeling Language Specification Version 2.4.1." Online (July 2011).

[8] Object Management Group, "The Unified Modeling Language Specification Version 2.5.1." Online (Dec. 2017).

[9] Friedenthal, S., Moore, A., and Steiner, R., [*A practical guide to SysML: the systems modeling language*], Morgan Kaufmann (2014).

[10] Object Management Group, "Object Constraint Language Version 2.4." Online (Jan. 2014).