Semantics in Space Systems Architectures

Alessandro Gerlinger Romero*
Brazilian National Institute for Space Research, São José dos Campos, Brazil
Klaus Schneider
Department of Computer Science, University of Kaiserslautern, Kaiserslautern, Germany
Maurício Gonçalves Vieira Ferreira†
Brazilian National Institute for Space Research, São José dos Campos, Brazil

Costs, life cycles, technologies and agreements between stakeholders and organizations make space systems unique with respect to complexity. A commonly accepted technique to address part of this complexity is to model and to maintain space systems architectures through the life cycle of their space programs. The benefits may range from supporting consistent model definitions and maintenance up to supporting analysis and verification. Space systems architectures have been modeled using UPDM (Unified Profile for DoDAF And MODAF). In fact, UPDM argues that it provides clearer understanding of the semantics behind specific views and viewpoints. Nonetheless, while UML defines its semantics imprecisely using plain text and variation points, UPDM does not define any semantics. In this paper, we evaluate an extension of fUML (semantics of a foundational subset for executable UML models) as a semantic foundation for space systems architectures. The extension of fUML as a synchronous language provides a limited, but formally precise and deterministic, form to describe structure and behavior in UML. Through the combination of this semantics foundation with UPDM, a precise language supporting a standardized meta-model emerges for the definition of space systems architectures. At the end, a simplified case study covering the operational view (OV-*) is presented. Our initial results show that synchronous fUML is able to offer semantics for UPDM.

I. Introduction

Costs, life cycles, technologies and agreements between stakeholders and organizations make space systems unique with respect to complexity. A commonly accepted technique to address part of this complexity is to model and to maintain space systems architectures through the life cycle of their space programs. There is a large number of research describing the benefits of this technique, which ranges from supporting consistent model definitions and maintenance up to supporting analysis and verification.[7, 15, 20]

Towards space systems architectures the major influences have come from RM-ODP (Reference Model of Open Distributed Processing)[8] and Systems Engineering, which led to definition of the following significative standards: space system model in ECSS-E-ST-70-31C[5] and Reference Architecture for Space Data Systems (RASDS) by CCSDS 311.0-M-1.[21]

In particular, RASDS recognizes in its Annex B that UML and SysML (Systems Modeling Language)[12] are feasible alternatives to represent the concepts of RASDS and then enable UML models definition in accordance with the meta-model prescribed by RASDS. Two important clarifications are provided in this annex: (1) the UML diagrams are just a visual notation for the model and (2) the diagrams must be adapted with a thorough semantics in order to adequately represent stakeholder viewpoints (pp. B-2[21]). Furthermore, the Annex C from RASDS defines an initial mapping between the RASDS meta-model and DoDAF (USA Department of Defense Architecture Framework) meta-model.

*This work was supported by Brazilian Coordination for Enhancement of Higher Education Personnel (CAPES).
†This work was supported by São Paulo Research Foudation (FAPESP).

One can interpret that these two annexes from RASDS converge to UPDM (Unified profile for DoDAF and MODAF),[13] which is an UML profile designed to enable practitioners to express DoDAF and MODAF(UK Ministry of Defence Architecture Framework) architectures using UML or SysML.[13] In fact, space systems architectures have been modeled using UPDM (Unified Profile For DoDAF And MODAF).[7]

UPDM defines an abstract syntax and a concrete syntax, however, there is no definition about semantics.[13] Indeed, one strength of RM-ODP is its architectural semantics (ISO/IEC 10746-4),[9] which is achieved by interpreting each concept of its reference model in terms of the constructs of the different formal techniques (for example, Z notation). As UPDM is based on UML, a natural candidate to provide semantics is fUML (semantics of a foundational subset for executable UML models),[11] which provides semantics for a small subset of UML. Nonetheless, research recognized that this semantics often exhibit non-determinism.[4,18]

In this paper, we evaluate an extension of fUML[11] as a semantic foundation for space systems architectures. The extension of fUML as a synchronous language provides a limited, but formally precise and deterministic, form to describe structure and behavior in UML. Through the combination of this semantics foundation with UPDM, a precise language supporting a standardized meta-model emerges for the definition of space systems architectures. The proposed combination is under evaluation for space systems architectures because it is precise allowing formal analysis but it can demand special considerations and more effort (detailed definition that can have good acceptance in space community due to complexity of the space systems).

The remainder of this paper is organized as follows: in Section II, related works are explored; in Section III, the background is briefly reviewed; in Section IV, we present the main concepts that support the evaluation; in Section V, we explore the operational viewpoint of a simplified case study from the National Institute for Space Research(INPE); finally, conclusions are shared in the last section.

II. Related Works

Shames and Skipper[19] proposed an extended version of RASDS, specifically, they proposed a physical viewpoint, in order to support, e.g., structural view, thermal view and power view.

Poupart and Charmeau[15] proposed a UML profile based on SysML. The profile provides a DSML (Domain-Specific Modeling Language) for the products of a space system architecture as an alternative to provide a uniform space system view shared by all the different actors designing or operating the system.

UPDM, SysML and space systems architectures. Shames et al.[20] chose SysML to model a space system architecture[20] because UPDM had limitations for modeling detailed system and software views. Nevertheless, it used RASDS to guide the development of suitable viewpoint specifications for the SysML models. The work concluded that using modeling tools, with the possibility of consistency checking, composability, and reuse, brought advantages. Hayden and Jeffries[7] chose UPDM to model a space system architecture because it was well suited to describing architecture in high detail. The model was defined by taking into account the UPDM viewpoints. In particular, in the operational view, the OV-5 (Operational activity model) and OV-6c (Operational event trace description, used to analyze message timing) became the core of the concept of operations (ConOps).

Semantics of fUML. Benyahia et. al.[4] recognized that fUML was not directly feasible to safety-critical systems because the model of computation(MoC) defined in the fUML execution model was nondeterministic and sequential. Romero et al.[16] concluded that the fUML's MoC was nondeterministic because it allowed more than one active object to write in a unique event pool of another active object. Afterwards, it explored additional roots of this nondeterminism, grouping them as follows: (1) structural features manipulation - e.g., to assign a value to a property of an object; (2) conditions - fUML conditional clauses; (3) token flow semantics - how tokens were offered, and, consequently, in which sequence nodes were fired; and (4) event dispatching - how signals in the event pool were dispatched to *AcceptEventActions*. The research concluded that the groups (3) and (4) definitively compromised the capacity of the standard fUML's MoC to give a meaningful semantics for deterministic models. Romero et. al.[17] proposed a complementary meta-model for fUML covering a subset of UML's composite structures. The subset was defined in such a way that ports (from UML) were changed to compute the required and provided features based on abstract classes instead of interfaces (excluded from fUML[11]). Moreover, the paper used the embedding technique to extend the base semantics including relations about the composite structures, and then formal rules (using first-order logic) for the static semantics were defined using those newly relations.

In conclusion, we have not found evaluations of semantics in previous studies concerning space systems

architectures defined by RASDS or UPDM. Moreover, the semantics of models is a prerequisite for their analysis and verification.

III. Languages and Formalisms

In this section, languages and formalisms used in the present paper are reviewed.

A. UPDM

UPDM profile enables practitioners to express DoDAF and MODAF model elements and organize them in a set of specified viewpoints and views that support the needs of stakeholders.[13] It defines a language architecture reusing UML, furthermore, it provides an abstract syntax through the meta-model of the profile and a concrete syntax, the UML diagrams. There are two compliance levels in UPDM: Level 0 - that includes UML 2 and a subset of SoaML, and Level 1 - that extends the level 0 including SysML (meta-elements and diagrams).

The viewpoints defined by UPDM are: AcV-* - Acquisition View, AV-* - All View, CV-* - Capability View, DIV-* - Data and Information Views, OV-* - Operational View, PV-* - Project View, SOV-* - Service Oriented View, StdV-* - Standards View, STV-* - Strategic View, SV-* - System View, SvcV-* - Service View and TV-* - Technical View. In order to support each viewpoint, a series of stereotypes are defined, e.g., *OperationalActivity* is a stereotype for UML *Activities* to be used in the operational view (OV-*), while *Function* is a stereotype for UML *Activities* to be used in the system view (SV-*). Moreover, each view is composed of products, e.g., the OV-* is composed of nine products, which are: OV-1 (High-level operational concept), OV-2 (Operational flow description), OV-3 (Operational resource flow matrix), OV-4 (Organization relationship chart), OV-5 (Operational activity model), OV-6a (Operational rule model), OV-6b (Operational state transition description), OV-6c (Operational event-trace description) and OV-7 (Operational information model).

UPDM does not provide a geo-spatial-temporal modeling (4D) for all UPDM elements (pp.39),[13] in particular, *interaction points* (an action performed at some 3D point at a given instant) is not supported in the operational view (OV-*). Finally, UPDM is designed pursuing that teams concentrate on architecture issues rather than documentation, and consistency is automatically maintained by a tool (pp.49[13]).

B. fUML

Although UML 2 defined the action semantics, in which a set of actions are the fundamental units of behavior, the lack of precise semantics was still an issue.[14] This lack of a precise semantics in the OMG specifications has been manifested by a large number of proposals for semantics of UML.

The size and complexity of a language's syntax can have direct consequences on the size and complexity of its semantics. Aware of this, OMG defines a semantics for a foundational subset of UML (fUML), as an attempt to answer the need for a precise semantics for UML.[11] Thus, fUML selects part of actions defined in UML to model behavior, and part of expressiveness of classes to model structure. The specification does not define a concrete syntax so the only notation available to define user's models is the graphical notation provided by UML, namely activity diagrams and class diagrams

fUML defines four elements for the language: (1) abstract syntax, (2) model library, (3) execution model and (4) base semantics.[11] The abstract syntax (1) is a subset of UML with complementary constraints. The model library (2) defines primitive operations, e.g., *add* for real numbers. The execution model (3) is an interpreter written in fUML (circular definition), which is defined using core elements from fUML that together form the base UML (bUML). Base semantics (4) breaks the circular definition of fUML providing a set of axioms and inference rules, described in first-order logic, that constrains the allowable executions of interpreters (allowing formal evaluation of interpreters). fUML does not define a concrete syntax, therefore, the sole notation available for defining UML models is the graphical notation provided by UML, namely activity diagrams and class diagrams.[11]

Concerning fUML, one basic premise from this modeling language is that all behaviors are ultimately caused by actions executed by instances of active classes, a special type of class that has its own thread.[11]

Remark - asynchronous versus synchronous communication. The *CallBehaviorAction* in UML and UPDM defines that a call can be sequential, guarded or concurrent. fUML constrains these calls to the sequential type. As a result, the sole mechanism for asynchronous invocation in fUML is sending signals

(*SendSignalAction*) to another active objects.[11] Here, the asynchronous term is interpreted as defined by UML "the caller proceeds immediately and does not expect a return value"(pp. 250[14]). The term asynchronous, in this sense, does not comprehend any definition about the relationship between signals emitted, only about the invocation from the caller to the callee.

C. Synchronous languages

Synchronous languages have been established as a technology of choice for specifying, modeling, and verifying real-time applications.[2] Synchronous languages share the constructive semantics, which defines that the status of each signal in a macro-step is established and uniquely defined prior to being tested and used, which enforces a deterministic behavior provided that the model is constructive (nonconstructive models are rejected). A property of the constructive semantics is that the results do not depend on the macro-step execution strategy for the actions.[3] Moreover, the focus of synchronous languages is to allow modeling and programming of systems in which ***cycle*** (computation step) precision is a requirement. These cycles, a rigid division of time, force the modeler to be well aware of them so as not to miss important signals. In particular, it has been argued that synchronous languages are well-suited for programming reactive real-time systems, while complex systems generally require the combination of asynchronous and synchronous modules.

These languages define that the most of the actions of a given language are executed in zero physical time (at least in the idealized model). Synchronous computations consist of a possibly infinite sequence of atomic actions that are triggered by a global logical clock. In each reaction, all inputs are read and all outputs are computed by all components in parallel. In the synchronous-reactive MoC, the communication and computation are done in zero physical time. Consumption of global logical time must be explicitly defined with special statements (in imperative synchronous languages), e.g., the *pause* statement in Esterel.[2]

Indeed, the synchronous languages rely on an abstract notion of time: the notion of physical time is replaced by an order among events, for which the relevant relationships are coincidence and precedence. Physical time does not play a special role because it is handled as an external event, as any other event coming from the environment. This is called the *multiform notion of time*: one can express delays in "centimeters" or in "seconds" counting their occurrences.[1] The duration of such occurrences as well as their starting physical time are not considered and remain abstract.[6]

IV. UPDM, a Language for Architecture Descriptions

UPDM is a language for architecture descriptions that has a well-defined concrete and abstract syntax. Nevertheless, a fundamental component for a language is its semantics. Semantics of a language is usually described by two components: static semantics and dynamic semantics.

Static semantics, also called *context-sensitive constraints*, defines well-formed rules considering the context in which concepts from the abstract syntax are used. One can define them writing a set of function specifications that defines the conditions under a given instance of the abstract syntax is declared well-formed. These functions can be Boolean-valued functions, and can express ideas like "all classes in the same package have unique names". In UPDM, likewise UML, these rules are defined by OCL (Object Contraint Language) in each element of the language.

The dynamic semantics of a language, $L_{semantics}$, can be defined as a double $L_{semantics} = (L_{semanticDomain}, L_{semanticMapping})$, where:

- $L_{semanticDomain}$ defines the universe of discourse of the meanings, in programming languages, it defines which types an execution manipulates, e.g., in an object-oriented language, *Objects* are part of the semantic domain;
- $L_{semanticMapping} : L_{abstractSyntax} \rightarrow L_{semanticDomain}$ maps syntactical elements defined by the abstract syntax into the semantic domain, therefore, it provides meaning for syntactical elements. Meaning covers structural and behavioral aspects so, for example: a class (syntax, structural) A defines a subset of the *Objects* in the semantic domain, an action *CreateObjectAction from class A* (syntax, behavioral) creates a new *Object* in the subset related with the class A in the semantic domain.

Problem statement: *Although some structural consistencies can be defined without semantics (e.g., in AV-2 architecture dictionary), a semantics is mandatory for advanced structural consistencies (e.g., in*

OV-2 operational flow description defined by UML composite structures or SysML internal block diagrams, see discussion in Romero et. al[17]) likewise for behavior consistencies. Moreover, the diagrams must be interpreted with a thorough semantics in order to adequately represent stakeholder viewpoints (pp. B-2[21]).

Following the same idea of architectural semantics (ISO/IEC 10746-4)[9] from ODP, the semantics can take the form of an interpretation of the basic modeling and specification concepts of the architecture language using the various features of different formal languages.

At this point, we take for granted synchronous fUML as a well-defined language with a formal semantics. Synchronous fUML is chosen instead of fUML because it can support deterministic models considering time (the multiform of time). Therefore, it is possible the definition and the proper verification of a geo-spatial-temporal modeling (4D) even in views that define abstract behavior, e.g., OV-* operational view. As a drawback, the modeler must be aware of the cycles, the rigid division of time.

Then subsection presents synchronous fUML, afterwards, synchronous fUML is used in the architectural semantics following the same idea from ODP[9] but with a smaller scope.

A. Synchronous fUML in a nutshell

Synchronous fUML is a selection of the minimal elements from fUML that can support definition of synchronous behavior. In synchronous fUML, the basic notion for the behavioral semantics is that of an action which transforms the state of a world. A world is modeled using objects in synchronous fUML. Furthermore, the semantics of fUML can be understood as a labeled transition systems (LTS) in which actions are transitions between states (disregarding control flow).

1. Syntactics

The synchronous fUML covers the following elements supporting structural modeling: *Class*, *PrimitiveType*, *DataType*, *ValueSpecification*, *Property*, *Reception*, *Signal*, *SignalEvent* and *Trigger*. Note *Association* and *Generalization* are not part of the abstract syntax so they can be used in the diagrams but the semantics does not cover them. Moreover, *Association ends* not owned by the *Associations* are *Properties* of a *Class*.

The abstract syntax from synchronous fUML supports composite structures with the following constraints:

- Constraint 1 - One active object cannot access data that is managed by another active object (shared data between processes are forbidden). The reason for this constraint is that shared data can easily make systems inconsistent, and pose challenges to composability.[17]
- Constraint 2 - The communication between objects cannot be bi-directional. The reason for this constraint is that the communication is best understood when the channel is uni-directional. This simplifies the static, and behavioral analyses, and there is no expressivity loss because a bi-directional channel can be replaced by two uni-directional channels.[17]
- Constraint 3 - Active objects (processes) are solely objects that can exchange messages asynchronously through signals.[11]
- Constraint 4 - Connectors have two end points because connectors with more than two end points are rarely used, they introduce unnecessary complexity in the semantics and there is no expressivity loss (a connector with three endpoints or more can be replaced by two or more connectors with two endpoints[17]).

Still regarding composite structures, the required and provided features of a *port* is defined by abstract classes and the attribute *isConjugated*. For example: a *port* that has type *AbstractClassX* and attribute *isConjugated* equals to *false* means that the *port* receives the signals defined by the abstract class *AbstractClassX* (an input port), whereas if the attribute *isConjugated* is equal to *true*, the port emits the signals (an output port). Finally, it is possible to define structure and content of pre-defined runtime instances using: *InstanceSpecification* and *Slot*.

Regarding behavioral modeling, synchronous fUML as well as fUML only support user-defined behaviors described by *Activities*. Table 1 lists the activities covered by the abstract syntax from synchronous fUML,

Table 1. Activities provided by synchronous fUML and available stereotypes.

node	Synchronous fUML	Available stereotypes in synchronous fUML
Intermediate Activities		
ControlFlow	✓	
DecisionNode	✓	*Pausable*
FlowFinalNode	✓	*Pausable*
ForkNode	✓	*Pausable*
InitialNode	✓	*Pausable*
MergeNode	✓	*Pausable*
ObjectFlow	✓	

and the available stereotypes in synchronous fUML. Every *ControlNode* can be stereotyped with *Pausable*, which means that it demarcates the end/begin of macro-steps.

Concerning the actions provided by synchronous fUML, Table 2 shows the actions and the available stereotypes. *CallOperationAction* is not supported by synchronous fUML because it does not support object-orientation. Furthermote, *AcceptEventAction* is one of the key elements for the definition of the model of computation. Regarding the synchronous languages, three stereotypes are available in synchronous fUML for the *AcceptEventAction*: *NonBlockable* - it enables the reaction to absence, i.e., in every macro-step the *AcceptEventAction* stereotyped with *NonBlockable* returns a value independently of the presence or absence of an event, in the case of presence, the signal that caused the event is returned, in the case of absence a "null" is returned (in the user's models, there is no representation for absence of values so the action simply returns "null"); *PrecededBy* defines that at first tick of the event's clock a statically defined signal is returned; and *Previous* enables memory and constructiveness (in closed-loops) establishing that the value returned is the value of the signal that cause the event in the previous macro-step, besides, it requires an initial value returned in the first macro-step.

2. Semantics

The basic building block for concurrency in fUML is an active class. A class becomes an active class when the modeler assigns the value *true* to the attribute *isActive* of the class. Moreover, every active class must have an activity that defines its behavior, called *classifier behavior*. Both definitions are made during the modeling. One can create an *object* of an active class using the action *CreateObjectAction*, however, the creation does not start the classifier behavior. It is needed to use the action *StartObjectBehavior* passing as parameter an active object to start the classifier behavior. Therefore, the existence of an active object does not mean that it is running. This thesis uses the term "alive" or "dead" for active objects, meaning that their classifier behavior are running or not, respectively.

Non-terminating loops must be non-instantaneous, otherwise the system is not constructive. Active classes have an infinity loop that must be non-instantaneous meaning that once an active object is started, it runs forever. An infinity loop is not mandatory in every classifier behavior, in fact, a classifier behavior can terminate. If there is no active object alive, nothing is computed because the premise of UML states that *all behavior in a modeled system is ultimately caused by actions executed by the so-called active objects*.[11]

Indeed, synchronous fUML is a synchronous language because it has the essential and sufficient features for a synchronous language,[3] namely:

1. *Programs progress via an infinite sequence of macro-steps* - the semantics of synchronous fUML defines the semantics for a macro-step;

2. *In a macro-step, decisions can be taken on the basis of the absence of signals* - as presented above, the action *AcceptEventAction* stereotyped with *Nonblockable* enables the reaction to absence, absence is indicated by the returned value "null";

Table 2. Actions provided by synchronous fUML and available stereotypes.

node	Synchronous fUML	Available stereotypes in synchronous fUML
Basic Actions		
CallBehaviorAction	✓	
CallOperationAction	×	
InputPin	✓	
OutputPin	✓	
SendSignalAction	✓	
Intermediate Actions		
AddStructuralFeatureValueAction	✓	
ClearStructuralFeatureAction	✓	
CreateObjectAction	✓	
ReadSelfAction	✓	
ReadStructuralFeatureValueAction	✓	
RemoveStructuralFeatureValueAction	✓	
ValueSpecificationAction	✓	
Complete Actions		
AcceptEventAction	✓	*NonBlockable,PrecededBy,Previous*
StartObjectBehaviorAction	✓	

3. *Communication is performed via instantaneous broadcast* - the signals sent to a port (an active object) that it is not alive are instantaneously broadcasted to all objects connected (if the active object is alive, it defines a different behavior, in this case, the broadcast is not done by the semantics);

Likewise, a synchronous language, parallel composition of active objects is well-behaved and deterministic for constructive systems. Synchronous fUML deals with computation and communication as different phenomena. Computation is performed internally to active objects and it allows more than one value for a given variable at a given macro-step. The sequence of values for the variable is determined by the data flow and control flow dependencies. Communication is only allowed using signals exchanged between active objects and each of these signals assumes only one value at a given macro-step.

Lastly, the semantic mapping for the most basic syntactical elements is the following: a *Class* maps onto an *Object* in the semantic domain, a *Property* maps onto a *FeatureValue* from a given *CompoundValue*, a *DataType* maps onto a *DataValue*, a *Signal* maps onto a *SignalInstance* and an *Activity* maps onto an *ActivityExecution*. In addition, *CompoundValue* in the semantic domain is a generalization of *Object*, *DataValue* and *SignalInstance*, and *Object* in the semantic domain is a generalization of *ActivityExecution*. Note actions have no representation in the semantic domain from synchronous fUML but they change the state of the system.

B. Architectural semantics in Synchronous fUML

The semantics takes the form of an interpretation of the basic modeling and specification concepts of UPDM using the features of synchronous fUML. UPDM and synchronous fUML share the UML as basis so every meta-class used by UPDM that is part of synchronous fUML has a straightforward semantics. For the meta-classes used in UPDM that are not part of synchronous fUML, there are, at least, three options: (1) to extend synchronous fUML to support the meta-classes, (2) to define the semantics by a translation from the unsupported meta-classes into supported meta-classes in synchronous fUML, or (3) to extend UPDM in order to support the meta-classes from fUML.

Althougn the package *UPDM L0::Core::ServiceElements* should be the initial candidate to evaluate the interpretation of UPDM using synchronous fUML because it suppors *Signals*, the only mechanism in

synchronous fUML that provides inter-communication between active objects, we chose the package *UPDM L0::Core::OperationalElements* to evaluate the semantics. The reason is that we aim to investigate how an executable representation can impact the usual approach to deal with an essentially non-computational view.

In order to support the operational view, selected elements from the packages *UPDM L0::Core::AllElements* and *UPDM L0::Core::OperationalElements* have their interpretations defined using synchronous fUML. As discussed above, there are operational elements defined in UPDM not covered by synchronous fUML, and then, we chose to extend UPDM in order to make them interpretable using synchronous fUML. Next subsections declare the interpretation from selected UPDM elements for each selected package, and the extensions are grouped in a new package called *UPDM L0::Core::OperationalElements::Extended*.

1. UPDM L0::Core::AllElements::Behavior

Activity from UPDM is represented by an *Object* in the semantic domain from synchronous fUML.

2. UPDM L0::Core::AllElements::Environment

Environment from the UPDM abstract syntax is a specialization of *DataType* from UML and it is represented by a *DataValue* in the semantic domain from synchronous fUML.
EnvironmentProperty from the UPDM abstract syntax is a specialization of *Property* from UML and it is represented by a *FeatureValue* that is part of a *DataValue*.

3. UPDM L0::Core::AllElements::Structure

ExchangeElement from the UPDM abstract syntax is represented by a *CompoundValue*.
Participant from the UPDM abstract syntax is represented by an alive active *Object*, i.e. it has an *ActivityExecution* for its classifier behavior.

4. UPDM L0::Core::OperationalElements

OperationalActivity from the UPDM abstract syntax is a specialization of *Activity* from UML, which is part of synchronous fUML. It maps to an *ActivityExecution* for the specified *behavior*.
OperationalActivityAction from the UPDM abstract syntax is a specialization from the action *CallBehaviorAction*, which is part of synchronous fUML. The action has no counterpart in the semantic domain from synchronous fUML, however, it changes the state creating a new *ActivityExecution* for the specified *behavior*, hence, the activity execution is run.

5. UPDM L0::Core::OperationalElements::Extended

Fig. 1 shows the extensions in the UPDM profile in order to support a straightforward interpretation of *ExchangeElement* and *OperationalStateDescription*.

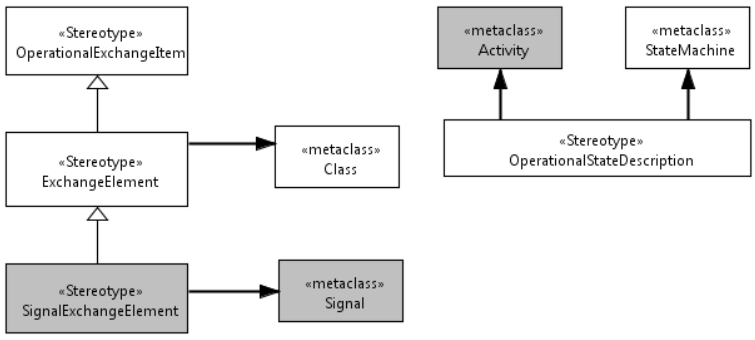

Figure 1. Package *UPDM L0::Core::OperationalElements::Extended* from the extended UPDM profile.

ExchangeElement is specialized by a new stereotype called *SignalExchangeElement*, therefore, exchanges in the operational view can be expressed by *Classes* or *Signals*. With the stereotype *SignalExchangeElement* in

a *Signal*, it is possible to express exchange of information elements in the operational view in accordance with UPDM. *SignalExchangeElement* from extended UPDM is represented by a *SignalInstance* in the semantic domain from synchronous fUML.
OperationalStateDescription is a specialization of *Activity* from UML, therefore, one can define the operational state description using state machines or activities. *OperationalStateDescription* described by an *Activity* is represented by an *ActivityExecution* for a classifier behavior from an active *Object* in the semantic domain from synchronous fUML. *OperationalStateDescription* described by state machines has no interpretation.

V. Case Study

In this simplified case study, the operational view is defined using the compliance level 0 from UPDM, i.e. based on UML.

An Operational View (OV) describes the activities, operational elements and information exchanges required to conduct operations. Moreover, as preconized by the UPDM, the emphasis is on the modeling and analysis of *Participants* (*Node*), their operational activities *OperationalActivity* and their communication. The computation is abstract, denoted by the operational activity actions *OperationalActivityActions*, which indeed are *CallBehaviorActions* to activities not necessarily detailed.

Next subsections explore the products produced for the description of the operational view. The products OV-4 organization relationship chart, OV-6a operational rule model and OV-7 operational information model were not defined.

A. OV-1b High-level Operational Concept Description

In the National Institute for Space Research (INPE), the satellite tracking and control center (CRC, *SatelliteTrackingAndControl*) is the department responsible for the activities of tracking and control of satellites. The CRC consists of the satellite control center (SCC, *SatelliteControlCenter*) in São José dos Campos and the tracking ground stations (*TrackingGroundStation*) of Cuiabá (CBA) and Alcantara (CLA). These three sites are interconnected by a private network, which is suppressed in the sequel models to keep them simple.

The communication of the CRC with the satellites is established by the tracking ground stations during the visibility window of the antennas. During these windows, the signals transmitted by a satellite are sent by its antenna providing a downlink communication. The signals contain the information of the satellite telemetry revealing its current state of operation. After the establishment of a downlink, the tracking ground station provides an uplink, which is used for sending telecommands. All control actions are planned, coordinated and executed from the CRC. During the windows of satellites' visibility, the CRC connects to a tracking ground station through the network, and then it is able to receive and send data from the visible satellite.

B. OV-2 Operational Flow Description

OV-2 illustrates the nodes in the *SatelliteTrackingAndControl* as well as the need to exchange information between the them. Fig. 2 shows the structural view (UML class diagram) of the nodes. The main points are:

- *SatelliteTrackingAndControl* defines the boundaries of the operational view with three ports to an outer system, *rawdataReceiver*, *rawdataEmitter* and *telecommandReceiver*. The ports *rawdataReceiver* and *rawdataEmitter* communicate with the space segment (beyond of the scope of this operational view), and the port *telecommandEmitterSystem* broadcast to an outer system the telecommands defined to be sent to the space segment. It has two parts: *TrackingGroundStation* and *satelliteControlCenter*. The multiplicity of the *TrackingGroundStation* is not defined as two (CBA and CLA) to maintain the operational view independent of the system view. Finally, *SatelliteTrackingAndControl* is modeled using an active class, denoted in the diagram by a class box with an additional vertical bar on either side.
- *TrackingGroundStation* is an active class with four ports. The ports are clearly shown in Fig. 3.
- *SatelliteControlCenter* is the last active class with two ports.

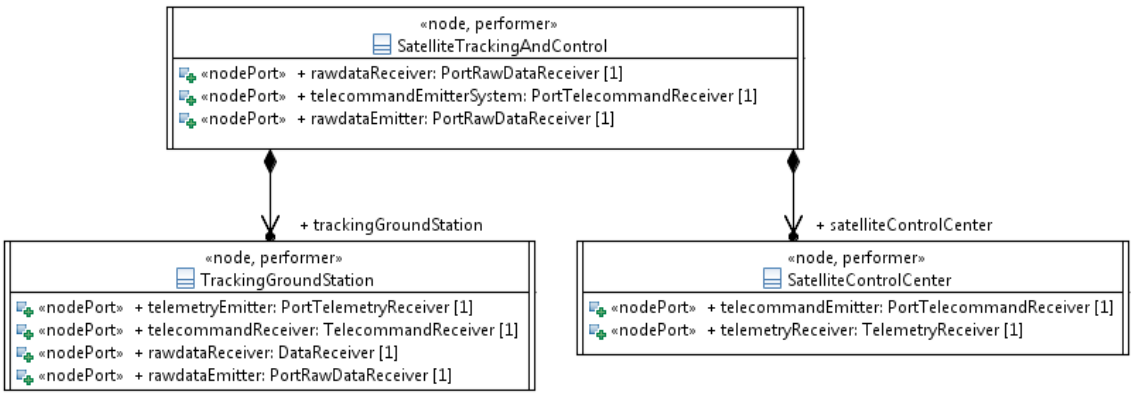

Figure 2. OV-2 - Operational flow description - UML class diagram.

Fig. 3 shows the communication and collaboration between the nodes defining the need to exchange information. It is a UML composite structure diagram, in which the white color in ports means that they are not conjugated so they are input ports, and the gray color in ports means that they are conjugates, therefore, they are output ports.

The communication and collaboration in the system can be explained as follows. *Rawdata* coming from an outer system is received by the part *trackingGroundStation*, the data is transformed in *Telemetry* and sent to the part *SatelliteControlCenter*. The part *satelliteControlCenter* sends *Telecommands* to the *trackingGroundStation* as well as to an outer system. Finally, the part *trackingGroundStaticion* may sent *Rawdata* to an outer system.

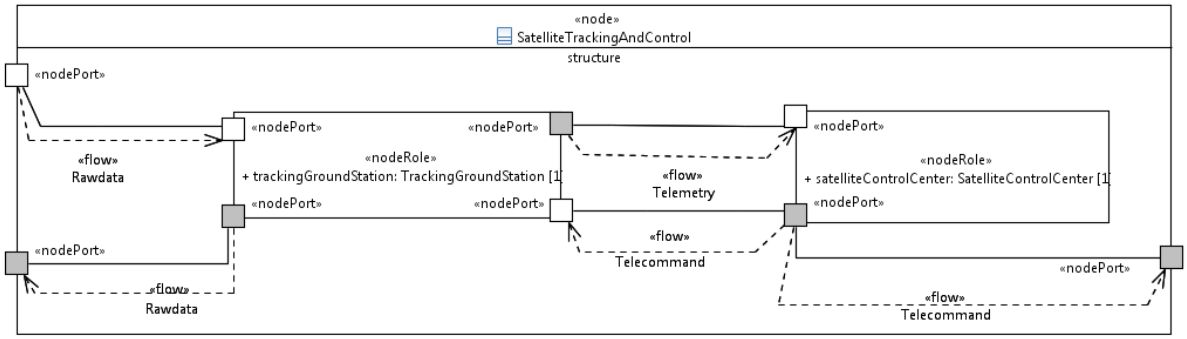

Figure 3. OV-2 - Operational flow description - UML composite structure diagram.

Note, generally, it is necessary to type a connector with an association. However, system engineers do not use associations to support connectors because associations are viewed as a software-level concept with weak semantics and not suitable for system-level modeling (pp.259[10]). Therefore, the structure of the operational view shown in Fig. 2 does not have associations to support the connectors.

C. OV-3 Operational Resource Flow Matrix

Fig. 4 is the result of a query in the model focused on the UML composite structure shown in Fig. 3. It shows in a tabular form the *operationalExchanges* with their transported information *conveyed*, their source, their target and the *connector*(stereotyped with *Needline*) that realize them. These tables are a key element in the definition of interface requirement documents.[20]

D. OV-5 Operational Activity Model

The operational activity model describes the operations that are conducted in the course of achieving a mission. It describes the activities hierarchy and the nodes that performs each activity. Fig. 5 shows the

	[Label]	conveyed	informationSource	realizingConnector	informationTarget
1	«OperationalExchange» rawdataIn	«SignalExchangeElement» Rawdata	«NodePort» rawdataReceiver	«Needline» rawdataIn	«NodePort» rawdataReceiver
2	«OperationalExchange» rawdataOut	«SignalExchangeElement» Rawdata	«NodePort» rawdataEmitter	«Needline» rawdataOut	«NodePort» rawdataEmitter
3	«OperationalExchange» telemetryOut	«SignalExchangeElement» Telemetry	«NodePort» telemetryEmitter	«Needline» telemetryOut	«NodePort» telemetryReceiver
4	«OperationalExchange» telecommand	«SignalExchangeElement» Telecommand	«NodePort» telecommandEmitter	«Needline» telecommandOut	«NodePort» telecommandReceiver
5	«OperationalExchange» telecommandOut	«SignalExchangeElement» Telecommand	«NodePort» telecommandEmitter	«Needline» telecommandOutSystem	«NodePort» telecommandEmitterSystem

Figure 4. OV-3 - Operational resource flow matrix.

identified hierarchical decomposition of the operational activities. It uses a class diagram to show how the behavior is structured, nevertheless, it is common to use activity diagrams with *swimlanes* for the product OV-5b. However, *swimlanes* are notational features in UML (pp.352[14]) so they have no semantics in synchronous fUML, and then, they could be used only for visualization.

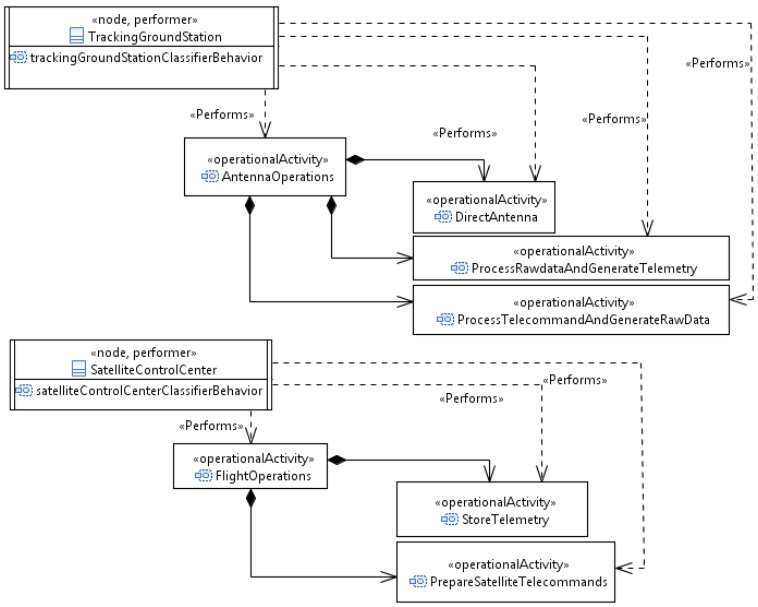

Figure 5. OV-5a Operational activity model - UML class diagram.

One important point is that each *node* has an owned activity to express its classifier behavior, therefore, the relationship *performs* is implicit in this case.

E. OV-6b Operational State Transition Description

The operational state transition description is a graphical method of describing how an operational node responds to various events by changing its state. The diagram represents the sets of events to which the node will respond (by taking an action to move to a new state) as a function of its current state.[13]

As defined by UPDM, the *OperationalStateDescription* should be defined by a state machine visualized by a state machine diagram, however, the package *UPDM L0::Core::OperationalElements::Extended* introduces the possibility to define its behavior using activities, and then, interpret according to the architectural semantics. Indeed, environments of synchronous languages offer tools to visualize the resulting automata from a given action-oriented description avoiding the explicit enumeration of states. Therefore, Fig. 6 and Fig. 7 show the state transition description for the operational nodes using activities. Fig. 6 can be roughly explained as follows. In every reaction, the antenna is directed (described by the *OperationalActivityAction DirrectAntenna*), then *Rawdata* is received, concurrently, *telecommands* are received. Afterwards, *RawData* is (ideally) processed by an *operationalActivity* and *Telecommand* is (ideally) processed by another *operationalActivity* concurrently. Finally, the results are sent to the respective target, and the reaction ends.

Fig. 7 can be roughly explained as follows. In every reaction, the *Telemetry* is received, then it is stored and used to prepare the telecommands. Finally, the telecommands are sent to the port *telecommandEmitter* and the reaction ends.

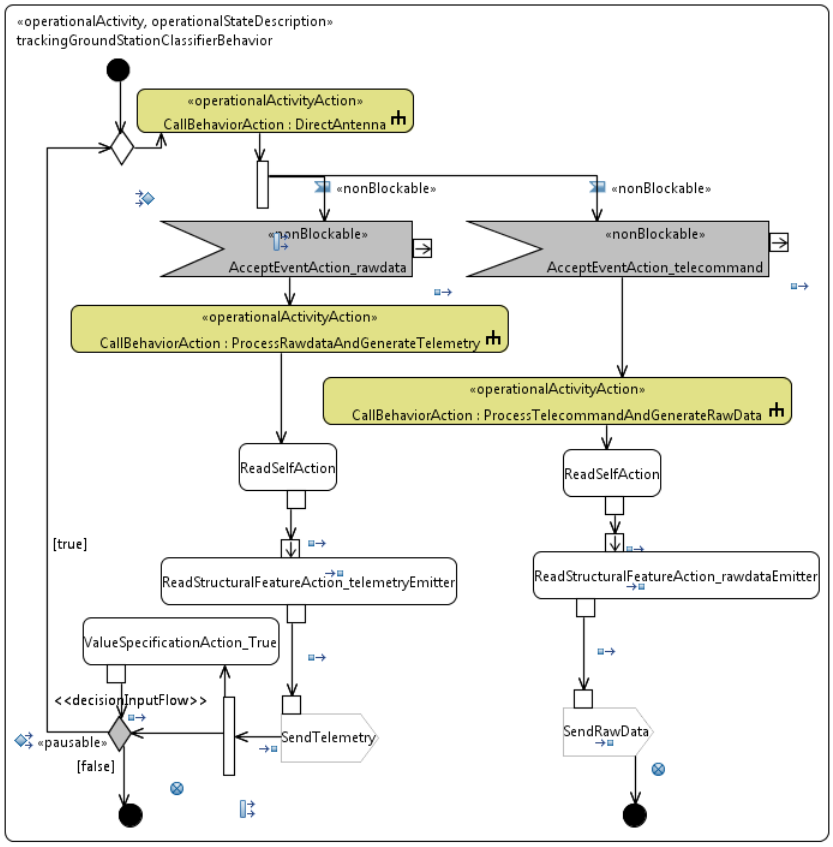

Figure 6. OV-6b Operational state transition description - *trackingGroundStationClassifierBehavior*.

Note the sterotypes *Pausable*, *Nonblockable*, *Previous* are used to define a reaction that is constructive, therefore, it is possible to execute the behaviors with guarantees of determinism.

F. OV-6c Operational Event-Trace Description

OV-6c is used to define time-based behavioral scenarios between the operational elements. Using the architectural semantics, the previously defined diagrams can automatically generate this product running the model. It follows an excerpt from the synchronous fUML simulator.

reactionClk	status	signal	source	target
1	PRESENT	*Telecommand*	*SatelliteControlCenter*	*PortTelecommandReceiver*
1	PRESENT	*Rawdata*	*TrackingGroundStation*	*PortRawDataReceiver*
1	PRESENT	*Telemetry*	*TrackingGroundStation*	*PortTelemetryReceiver*
...	...	...	...	...

VI. Conclusion

The emergence for semantics of architecture languages are not new, indeed, in 1998 ODP defined its architectural semantics[9] and RASDS, ten years later, recognized that diagrams must be adapted with a thorough semantics in order to adequately represent stakeholder viewpoints (pp. B-2[21]). While UML defines its semantics imprecisely using plain text and variation points, UPDM does not define semantics.

The standard fUML is a natural candidate to provide semantics for UPDM, however, fUML does not cover basic demands of system engineering as UML composite structures,[10] furthermore, it is easy to observe nondeterminism in its interpretations of models. In this context, synchronous fUML is a suitable candidate

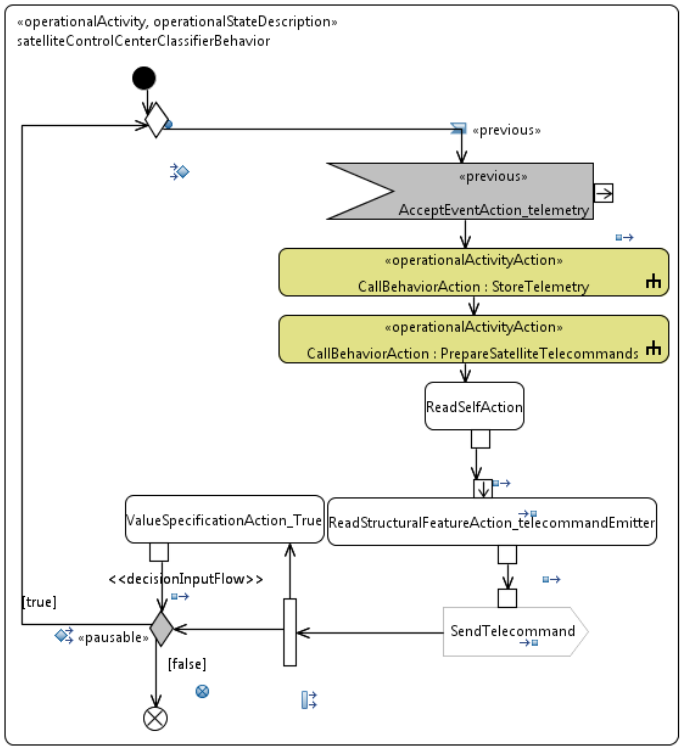

Figure 7. OV-6b Operational state transition description - *satelliteControlCenterClassifierBehavior*.

providing semantics for composite structures (including broadcast of information elements) likewise precision and determinism. In the non-computational viewpoints, it has been evaluated for precise definition of information's communication as well as for precise temporal behavior disregarding its capacity for the computation (which is important for system and service viewpoints). Even in non-computational viewpoints, it enables the automatic generation of OV-5a, OV-5b and OV-6c simply running the view (here, not considering the necessity for analysis, e.g., OV-5a may be defined and analyzed before OV-6b). The representation as a labeled transition system with a well-defined time concept enables advanced analysis, e.g., model checking. Nonetheless, it introduces the dimension of time since the beginning of modeling and analysis, for which it is early to draw a conclusion about costs and benefits. Considering flows of energy and material, we have been evaluating hybrid fUML, which is an extension of synchronous fUML supporting DAEs (differential algebraic equations).

In conclusion, we are sure that fUML is the candidate for semantics of UPDM supporting space systems architectures, moreover, we think synchronous fUML may raise the accuracy level of the space systems architectures radically changing the distribution of efforts during modeling and analysis of space systems architectures.

References

[1]Andre, C., Mallet, F., Peraldi-Frati, M. A. (2007) A multiform time approach to real-time system modeling. In: Inter. Symposium on Industrial Embedded Systems, 2007.

[2]Benveniste, A., Caspi, P., Edwards, S., Halbwachs, N., Le Guernic, P.,de Simone, R. (2003) The Synchronous Languages Twelve Years Later. In: Proceedings of the IEEE, vol. 91, number 1, pages 64-83.

[3]Benveniste, A., Caillaud, P., Le Guernic, P. (2000) Compositionality in Dataflow Synchronous Languages: Specification and Distributed Code Generation. In: Inf. Comput, vol. 163, pages 125-171.

[4]Benyahia, A., Cuccuru, A., Taha, S., Terrier, F., Boulanger, F., Gerard, S. (2010) Extending the Standard Execution Model of UML for Real-Time Systems. In: IFIP Advances in Information and Communication Technology, pp. 43–54, Australia, 2010.

[5]European Cooperation for Space Standardization (ECSS). (2008). Space Engineering Ground systems and operations monitoring data definition, ECSS-E-ST-70-31C, ESAESTEC The Netherlands, 2008.

[6]Forget, J., Boniol, F., Lesens, D., Pagetti, C., Pouzet, M. (2008) Programming Languages For Hard Real-Time Embedded Systems. In: Embedded Real Time Software, France, 2008.

[7]Hayden, L. J., Jeffries, A. (2012). On Using SysML, DoDAF 2.0 and UPDM to Model the Architecture for the NOAAs Joint Polar Satellite System (JPSS) Ground System (GS). In: SpaceOps 2012, 2012, Stockholm. 12th International Conference on Space Operations. Stockholm: AIAA, 2012.

[8]International Organization for Standardization (ISO). (1998). Information technology Open Distributed Processing Reference model: Overview. ISO/IEC 10746-1.

[9]International Organization for Standardization (ISO). (1998). Information technology Open Distributed Processing - Reference Model: Architectural semantics. ISO/IEC 10746-4.

[10]Ober, I., Ober, I., Dragomir, I., Aboussoror, E. (2011) UML/SysML semantic tunings. Journal Innovations in Systems and Software Engineering, p 257-264, Springer-Verlag.

[11]Object Management Group (OMG). Semantics of a Foundational Subset for Executable UML Models: Version 1.1 RTF Beta1 (2012) http://www.omg.org/spec/FUML/

[12]Object Management Group (OMG). Systems Modeling Language: Version: 1.3 (2012) http://www.omgsysml.org/

[13]Object Management Group (OMG). Unified Profile For DoDAF And MODAF (UPDM): Version: 2.1 RTF Beta (2013) http://www.omg.org/spec/UPDM/2.1/

[14]Object Management Group (OMG): Unified Modeling Language Superstructure: Version: 2.4.1. (2011) USA: OMG, 2011. http://www.omg.org/spec/UML/2.4.1/

[15]Poupart, E., Charmeau, M. C. (2012). Modeling Space System to provide global coherency from design to operation phases. In: SpaceOps 2012, 2012, Stockholm. 12th International Conference on Space Operations. Stockholm: AIAA, 2012.

[16]Romero, A. G., Schneider, K., Ferreira, M. G. V. (2013). Towards the Applicability of Alf to Model Cyber-Physical Systems. In: International Workshop on Cyber-Physical Systems (IWCPS), 2013, Krakow, Poland.

[17]Romero, A. G., Schneider, K., Ferreira, M. G. V. (2014). Integrating UML Composite Structures and fUML. In: International Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM), 2014, High Tatras, Slovakia.

[18]Romero A. G., Schneider K., and Ferreira M. G. V. (2014). Using the Base Semantics given by fUML for Verification. In: International Conference on Model-Driven Engineering and Software Development (MODELSWARD), Lisbon, Portugal, 2014.

[19]Shames, P., Skipper, J. (2006). Toward a framework for modeling space systems architectures. In: SpaceOps 2006, 2006, Rome. 9th International Conference on Space Operations. Rome: AIAA, 2006.

[20]Shames, P., Anderson, M. L., Kowal, S., Levesque, M., Sindiy, O. V., Donahue, K. M., Barnes, P. D. (2012). NASA Integrated Network Monitor and Control Software Architecture. In: SpaceOps 2012, 2012, Stockholm. 12th International Conference on Space Operations. Stockholm: AIAA, 2012.

[21]The Consultative Committee for Space Data Systems (CCSDS). (2008). Reference Architecture for Space Data Systems (RASDS), CCSDS 311.0-M-1, CCSDS, Sept 2008.