Paper 108

The Use of Behavioral Diagrams in SysML CSER 2007

Larry Zdanis	Robert Cloutier, Ph.D.
1526 Oxford Road, Wantagh, NY 11793	Stevens Institute of Technology
larryzdanis@yahoo.com	**Robert.cloutier@stevens.edu**

Abstract

The definition of behavior in Systems Modeling Language (SysML) presents special challenges to systems engineers, as overlapping functionality exists among SysML behavioral diagrams. This paper aims to guide the System Engineer (SE) through the challenges faced in defining a system's behavior via SysML by (1) identifying a set of purposes for behavior definition, (2) identifying criteria to help the SE decide which diagrams to use to satisfy these purposes, and (3) demonstrating realization of these purposes for a sample use case. The proposed purposes are as follows: defining activity flow, tracking system state, defining system control, defining interactions, and allocating responsibility.

Introduction

The Object Management Group (OMG) first introduced the Unified Modeling Language (UML) in 1997. UML models consist of diagrams which address static and dynamic aspects of a software design. The models aid in the design and evolution of the software. In 2003 Systems Modeling Language (SysML) Partners was formed to extend the OMG's UML to address system engineering concerns. SysML v. 1.0a specification was submitted in 2005 and adopted in May 2006. As an extension to UML, SysML tailors UML's basic constructs to facilitate use of the language for system modeling and for non-software system modeling.

As a language specification, the SysML Specification intentionally avoids defining a process for applying the language. However, a SE new to UML and SysML is likely to face confusion during the flexible process of defining system behavior for the following reasons: (1) a complete set of purposes for defining system behavior have not been defined, and (2) overlapping functionality exists among the behavioral diagrams.

This paper aims to guide the SE in fully defining a system's behavior, without compromising the flexibility of the process. It does so by (1) identifying a set of purposes for behavior definition, (2) identifying criteria to help the SE decide which diagrams to use to satisfy these purposes, and (3) demonstrating realization of these purposes for a sample use case.

First presented is a brief overview of the various SysML diagrams to show the context of Behavior Diagrams in SysML. An example system is characterized to equip the SE to make good modeling decisions. Six purposes of the behavioral diagrams are defined as follows: Narrow Problem Focus, Define Activity Flow, Track System State, Define System Control, Define Interactions, and Allocate Responsibility. Achievement of each of purposes is addressed in the remaining sections.

Context of Behavior Diagrams

The structure of SysML is shown in Figure 1.

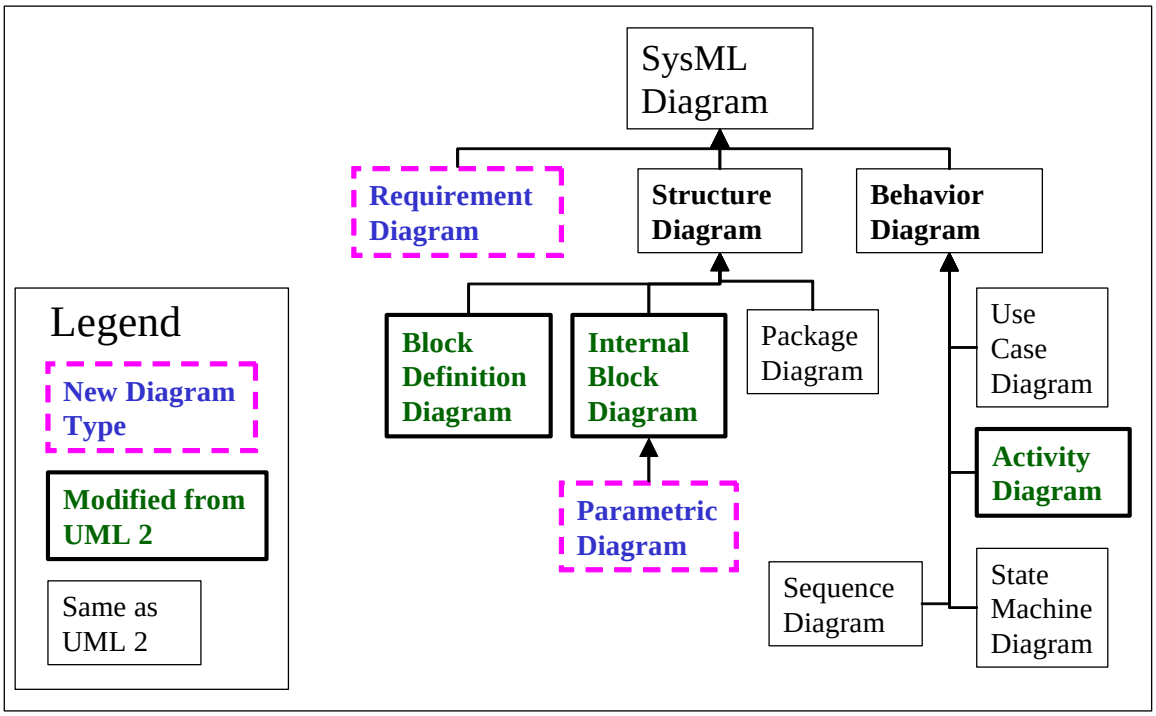

Figure 1. Structure of SysML (Reference: OMG 2006)

The usage of each diagram in Figure 1, along with its SysML abbreviation is shown in Table 1. The Requirement Diagram and Parametric Diagrams are new diagram types that don't exist in UML 2. The Activity Diagram, Block Definition Diagram, and Internal Block Diagrams have been modified from UML 2.

The Requirement Diagrams define "What" the system should do. The Structure and Behavior Diagrams communicate "How" the system will perform the "What". SysML diagrams provide for traceability of the visual "How" specification to the textual "What" specifications via relationship lines. Requirements Diagrams also express non-behavioral requirements, which are most likely to impact the Structure Diagrams.

Table 1. Usage of SysML Diagrams

Diagram	Abr	Usage
Requirements Pillar		
Requirement Diagram	req	Defines what the system should do. May include decomposition and traceability of requirements, and allocations of structure and tests to requirements.

Structure Pillar		
Block Definition Diagram	bdd	Defines structure of elements of various types, often in hierarchical form. Elements maybe physical or logical.
Internal Block Diagram	ibd	Defines the interfaces and item flows between elements.
Parametric Diagram	par	Defines equations which constrain system performance and links internal parameters to these equations.
Package Diagram	pkg	Provides a flexible method of grouping SysML diagrams.
Behavior Pillar		
Use Case Diagram	uc	Narrows the focus of system modeling to a single scenario at-a-time.
Activity Diagram	act	Defines the flow of activity through the system.
State Machine Diagram	smd	Defines transitions of the system or parts of the system through discrete states.
Sequence Diagram	sd	Defines a flow of messages between elements.

The Structure Diagrams communicate a framework which can execute the system's intended behavior. The Structure of a system is defined by its parts, the interfaces between its parts, and the parametrics which constrain the system's performance. The Behavior Diagrams communicate the behavior of the system. Behavior Diagrams demonstrates how the parts of the structure works together to satisfy behavioral requirements.

Characterizing System Behavior

A wide variety of behavior exists among systems that can be modelled with SysML. Understanding these differences will assist the SE in choosing SysML diagrams and mechanisms that clearly communicate behavior. As a first step, the SE should identify the essential behavioral characteristics of the system. At a minimum, the SE should consider the following:

- Is the system behavior **continuous** or **discrete**?
- Will operation of the system involve significant **human interaction**?
- Does the system include **controllers**? If so, are the controllers continuous or discrete?
- Does the system involve **concurrency**?
- May the system or parts of the system be characterized as exhibiting **state behavior**?

Providing examples for all system types is beyond the scope of this paper. Therefore, a single example is used to provide continuity and to demonstrate the integration of multiple diagrams. This example is a typical Fuel Pump (FP), operating at retail Gasoline Stations around the world. Using the considerations presented above, some key FP system characteristics are as follows:

- The FP system is primarily a sequence of **discrete** activities such as Swiping Card, Selecting Fuel Grade, and Activating the Pump. Fuel flow during pumping is **continuous**.

- The FP system requires significant **human interaction**.

- The FP system will require **continuous** and **discrete** controllers. Decreasing fuel flow upon nearing prepaid amount is assumed to utilize continuous control. Computer activation of remote pump utilizes discrete control.

- The FP system requires straightforward **concurrent** activities. The following three activities start and stop together: Squeezing Nozzle, Flow of Fuel, Metering of Fuel

- The following **states** will exist in the FP system: Idle, Authorized, Primed, Pumping, and Charging.

The following section explains which purposes of behavioral definition are relevant, depending on the system's characteristics.

Identifying the Purposes of the Behavioral Model

A complete and accurate system behavioral model will address all of a system's essential behavioral characteristics. The purposes of the behavioral model are displayed in Table 2. Tracking System State and Defining System Control may be viewed as part of Defining Activity Flow. They are shown as distinct purposes because they are relevant only if the system has states and/or controllers, and because the existence of states or controllers adds an extra dimension to activity flow definition.

The manner in which each purpose is achieved will vary with the system's characteristics and SE preferences. The remainder of the paper will suggest some guidelines in deciding how to utilize the available diagrams to accomplish these purposes.

Table 2. Purposes of Behavior Model

	Purpose Title	Purpose
1	Narrow Problem Focus	To manage complexity by focusing on a single scenario at-a-time.
2	Define Activity Flow	To define what activities are performed by the system in what order.
*2	Track System State	To track the state of the system.
*2	Define System Control	To define how a system is controlled.
3	Define Interactions	To define the flow of items between parts of the system.
4	Allocate Responsibility	To define the parts of the structure responsible for performing each activity and interaction.

Notes:
* These purposes maybe viewed as part of Defining Activity Flow, but are only relevant if the system has states and/or controllers.

Narrowing Problem Focus

Narrowing problem focus is first accomplished using Use Case Diagrams. A Use Case (uc) is a "focused" scenario through which the system's behavior may be modeled. The Use Case Diagram begins the process of activity decomposition at a high level. A sample Use Case Diagram for a typical Gasoline Station Fuel Pump is shown in Figure 2. A SE is only required to include Use Cases relevant to their immediate purpose (Booch, 1999). In Figure 2, several Use Cases that would normally exist for a Fuel Pump, such as "Regulatory Inspection", have been purposely omitted. Similarly, only Actors relevant to the Use Cases being addressed are included in the uc diagram.

The system of interest, 'Fuel Pump', is enclosed by the subject box. The entities which interact with the system are outside this box, connected to each uc in which they

participate by a solid line. The primary 'Get Fuel' uc is extended by three modular Use Cases using the <<include>> extension. These are 'Obtain Authorization', 'Pump Fuel', and 'Pay for Fuel'. At this point we have provided a context for defining activity flow for the 'Get Fuel' uc.

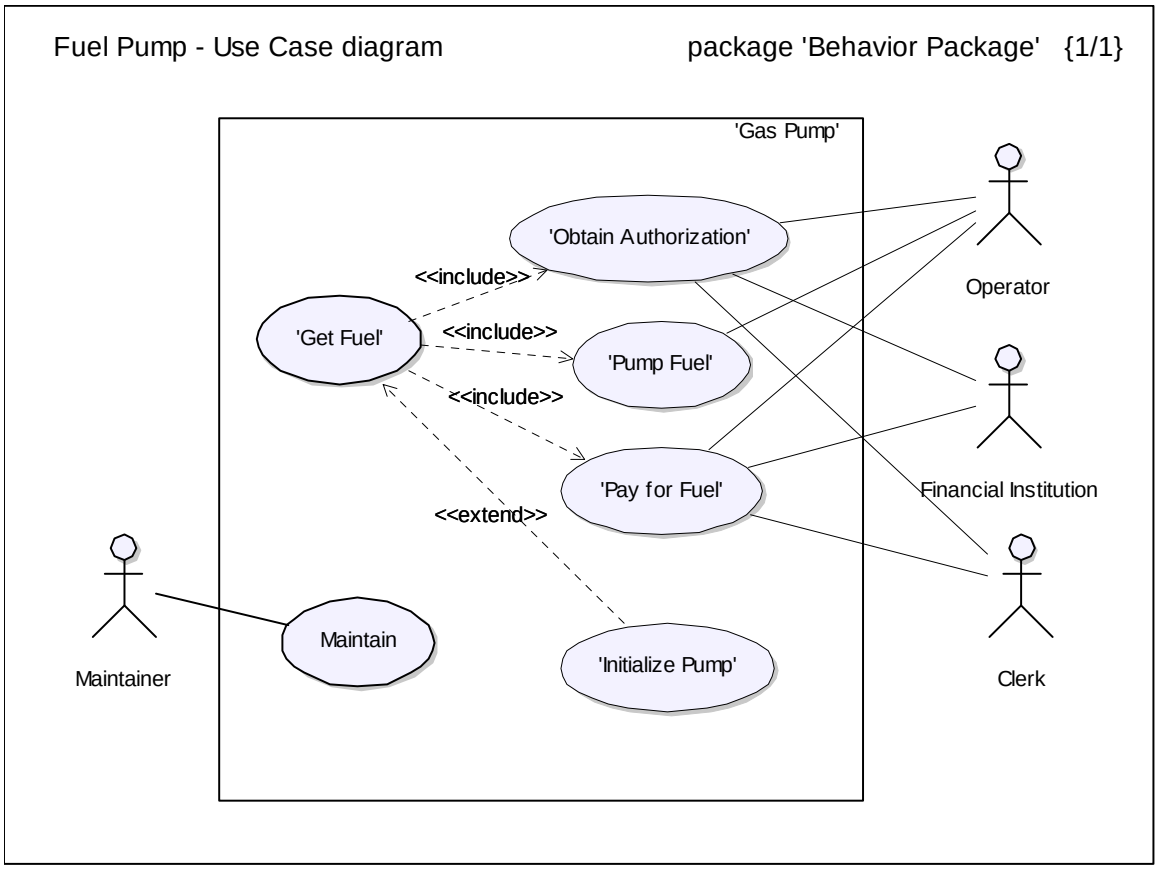

Figure 2. Fuel Pump Use Case Diagram

Starting Activity Flow Definition

Activity Diagrams, State Machine Diagrams, and Sequence Diagrams may all be used to define activity flow. The strengths and weaknesses of these diagrams for defining activity flow are shown in Table 3. As true in "Yin-Yang" theory, each of a diagram's strengths may cause weakness in other areas. This is why multiple diagrams are more powerful than one.

We'll now begin to define activity flow for the 'Get Fuel' uc of the 'Fuel Pump' system, making use of Table 3. We first aim to define a high level activity flow. The ad appears to be the best choice for this, as it provides a simplified depiction of intended flow, with auto-triggering of control upon completion of each activity. It also provides explicit visual depiction of complex flow, if needed. We avoid the state machine diagram, which may be ambiguous in demonstrating the desired sequence of flow.

Table 3. Strengths (S)and Weaknesses (W) of Diagrams for Defining Activity Flow

Activity Diagrams	
S	• Visually explicit depiction of complex flow. • Simplified depiction of intended flow, with auto-triggering of control upon completion of immediate activity (no event or message required) • Mechanism for a control operator available in SysML
W	o Generally shows only one of multiple possible sequences of state transitions at-a-time.
State Machine Diagrams	
S	• Visually explicit depiction of complex flow. • Clear indication of all possible transitions from each state of the system, subsystem, or element. • Exhibits same strengths in eliciting behavior of a controller.
W	o Standing alone, the desirable sequence of state transitions may be ambiguous.
Sequence Diagrams	
S	• Clear depiction of sequential flow.
W	o Less visually explicit depiction of complex flow. o Awkward for defining high level flow, as it requires allocation of activity to element

A high-level ad for the 'Get Fuel' uc is shown in Figure 3. The three <<include>> uc's from the Use Case Diagram in Figure 2 map to activities in this high-level diagram. The 'Select Fuel Grade' activity has not been designated a uc due to its simplicity.

There are different ways of viewing ad's. Establishing an ad viewpoint can assist the SE in effective use of the ad in a particular context. If an ad models a workflow, it maybe viewed as a simplified version of a statement machine diagram, in which each activity represents a different state, and the transition from one state to the next is triggered by completion of the immediate activity (Bennet 2001). If an ad models an operation, it may be viewed as a flowchart of the operation's actions.

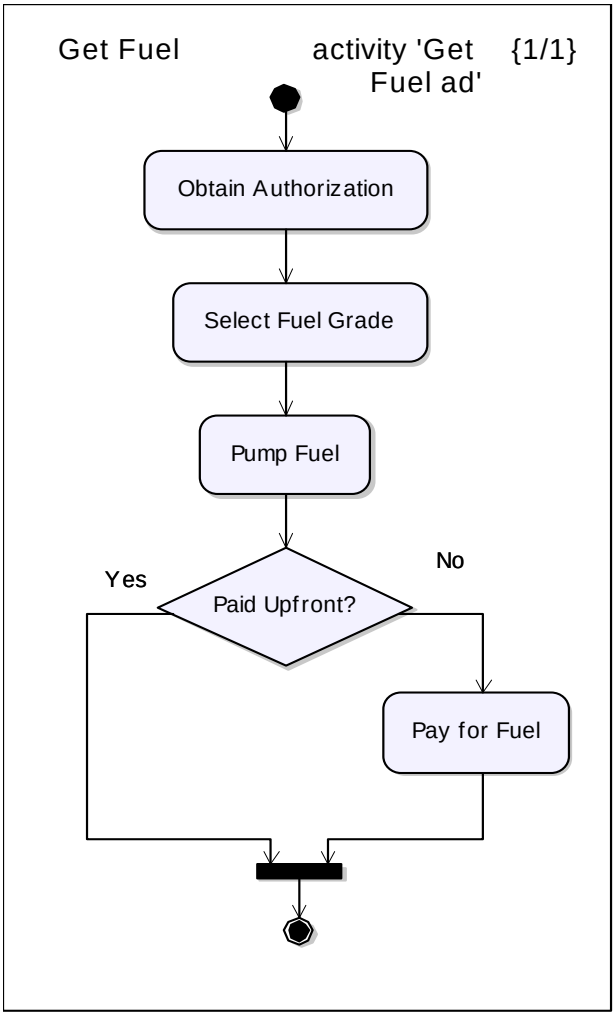

Figure 3. Get Fuel Activity Diagram

If an ad models the flow of activity through a series of states, it maybe viewed as the "inverse" of a smd, in which the ad emphasizes transitions between states, while the smd emphasizes states. In reality, the ad may elicit behavior within a state as well as behavior between states.

Upon examination of the Fuel Pump's characteristics and the ad, it appears most relevant to view this diagram as the inverse of a smd, which emphasizes the flow through a series of states. Recall that the Fuel Pump's behavior consists primarily of a sequence of discrete activities and exhibits discrete states. With this in mind, the next three sections will continue activity flow definition. Tracking System State will demonstrate a smd consistent with the ad above. Defining System Control will define control activity for the continuous controller part of the system. Finally, the sequence diagrams in Define Interactions will define activity flow between parts of the system.

Tracking System State

Tracking System State is important for all systems with states. If the SE is not aware of the current system state, the flow of activity is uncertain. States are best tracked through State Machine Diagrams (smd's). A smd may be drawn for the system being modeled or for any element in the system which exhibits various states. Figure 4 displays a smd for the Fuel Pump in the context of the 'Get Fuel' uc.

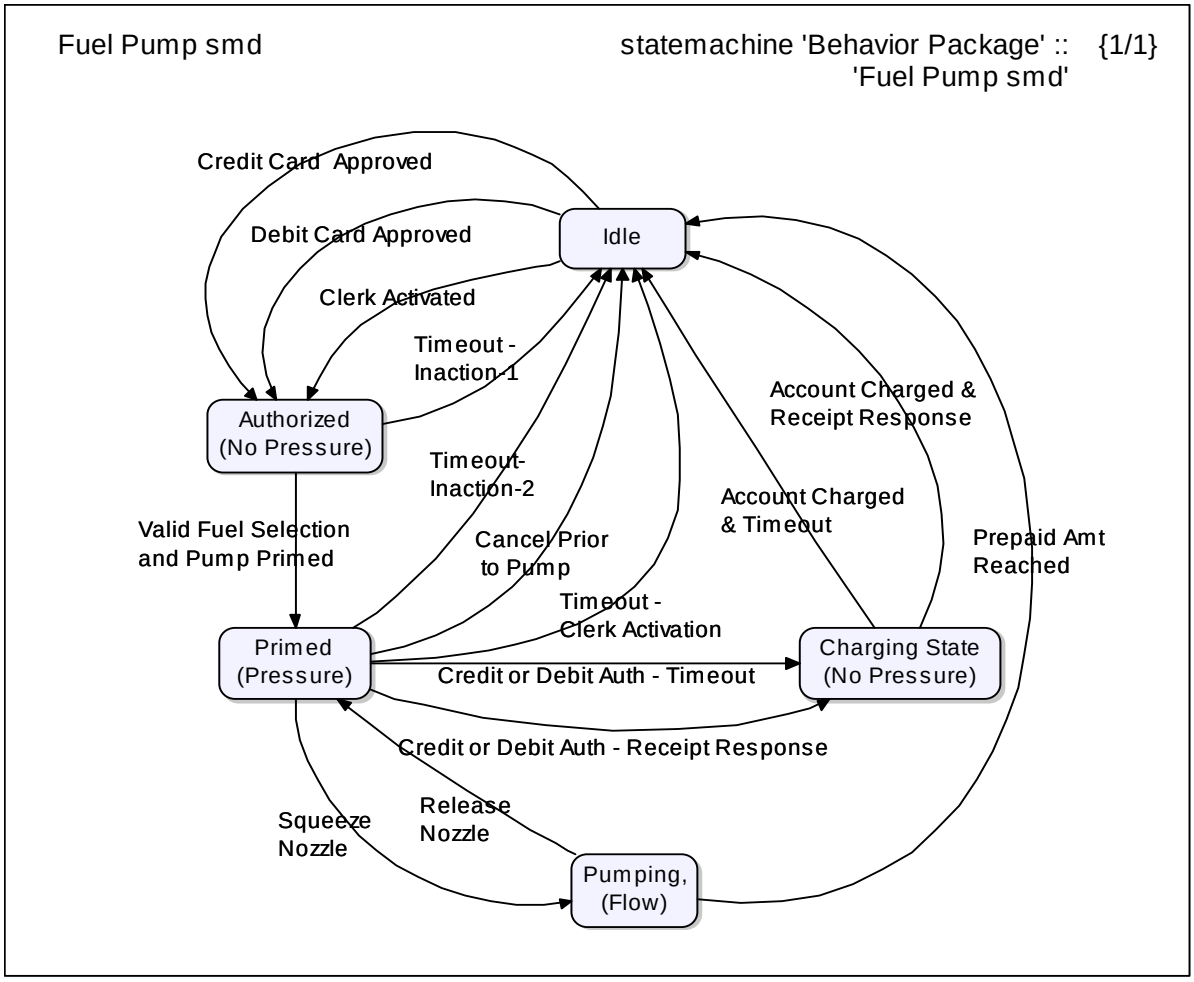

Figure 4. Fuel Pump State Machine Diagram

Figure 4 displays the following five states: Idle, Authorized, Primed, Pumping, and Charging. We arrived at these states by walking through a typical 'Get Fuel' uc scenario, and identifying discrete stages in which the Fuel Pump is expected to behave

differently from other stages. All of the possible transitions between states are shown and labeled. We arrived at the various transitions by considering all the possible operator, clerk, or internal events which may trigger a transition from each state.

The smd could potentially be used to independently elicit the flow of activity for a uc by showing only those transitions exercised for one instance of the use case. However, this would forfeit the power of the smd to clearly communicate a series of related information within the same visual space. Moreover, independent elicitation using a smd may contain ambiguities if the same state in the smd is reached more than once for a particular uc instance. By combining the natural ability of the ad or sd to show an intended flow with the smd's ability to depict potential transitions, an overall picture of activity flow has been obtained in an efficient manner.

Though Figure 3 and Figure 4 define the intended flow and all the possible flows at the system level, they don't dictate which flow variations will be realized for a particular uc instance. These flow variations depend on the actor's behavior at each point in the uc. This will be demonstrated using sequence diagrams in the "Defining Interactions" section.

Defining System Control

Activity flow definition would not be complete without addressing the activities performed by internal controllers. For the Fuel Pump, the discrete aspects of control (such as sending an activation signal to the remote fuel tank pump), are easily addressed as part of the lower-level sequence diagrams to follow. The continuous aspect of control, in which a metering valve is progressively closed upon nearing a prepaid amount, may be addressed by elaborating the "Pump" state with an activity diagram. Figure 5 shows an activity diagram for the "Pump" state.

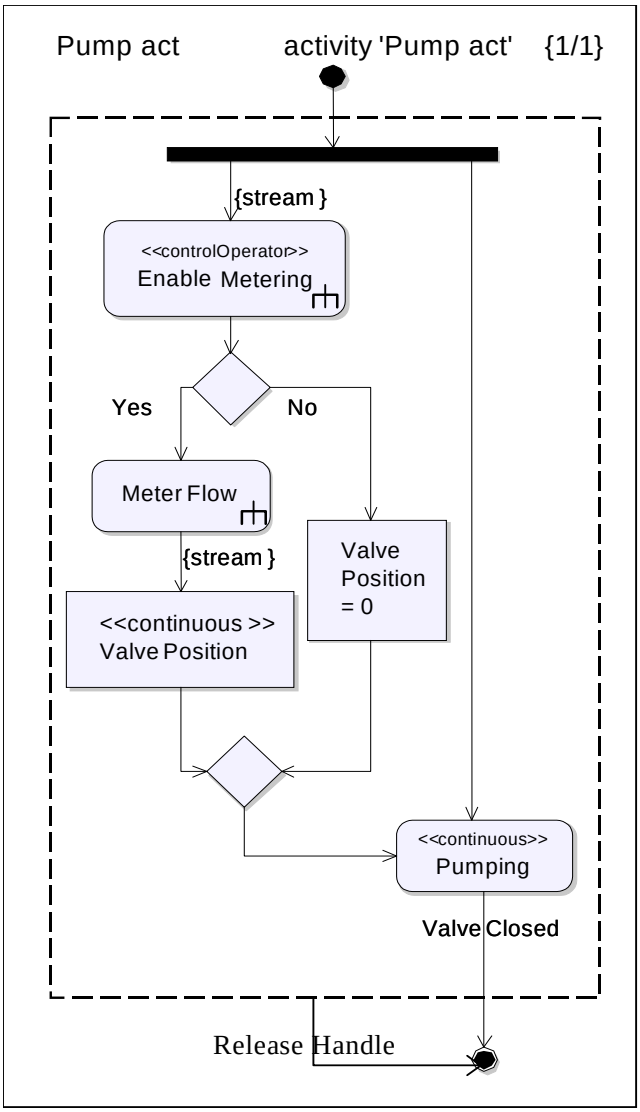

Figure 5. Pump Activity Diagram

The dashed line in Figure 5 represents an interruptable region; releasing the Fuel Nozzle causes immediate exit of the Pump state. SysML's rate stereotype is applied to indicate Pumping as a <<continuous>> activity. SysML's <<controlOperator>> stereotype is applied to the Enable Metering Activity, which determines whether or not metering of the flow is required. The <<controlOperator>> is continuous per SysML convention (OMG 2006). If metering is required, the Meter Flow activity calculates the appropriate Valve Position; otherwise Valve Position is set to 0, or wide open. The

Valve Position object is fed to the Pumping activity. If the Valve closes completely, the Pre-paid amount has been reached, causing a transition out of the Pump state.

Figure 6 reveals the details of the <<controlOperator>> Enable Metering. If the Prepaid Amount minus the Dollars Pumped is less than $1.00, the Control Value is set to enable; otherwise, it is disable. This causes the flow to be metered only when nearing the Prepaid Amount.

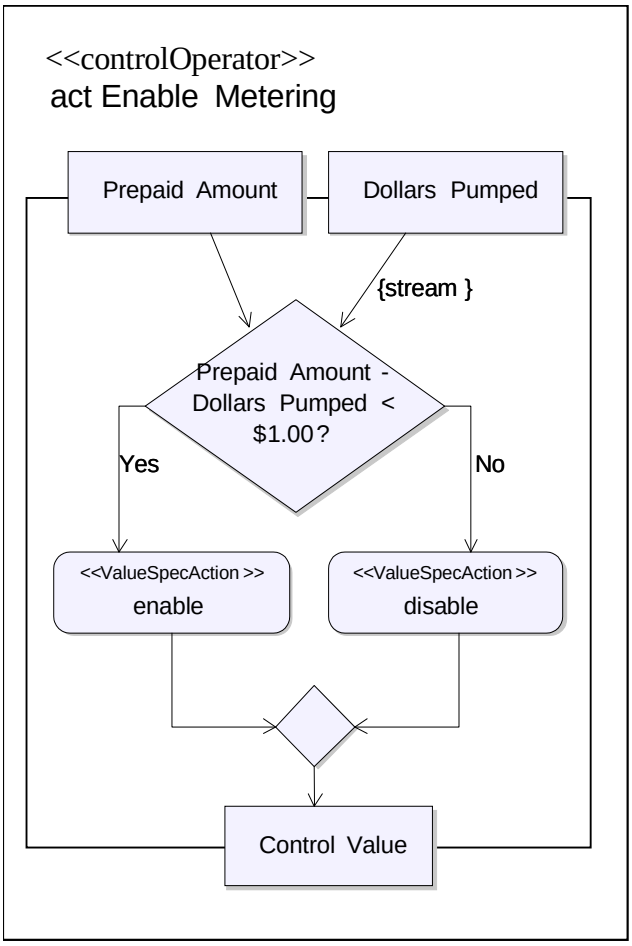

Figure 6. Enable Metering Activity Diagram

Figure 7 reveals the details of how the commanded Valve Position is calculated. The Flow Meter signal has been used to tally the Dollars Pumped. Valve Position = 1 - (Prepaid Amount - Dollars Pumped). This formula reveals that the Valve closes in a continuous linear fashion as remaining Dollars to be pumped falls from $1.00 to $0.00.

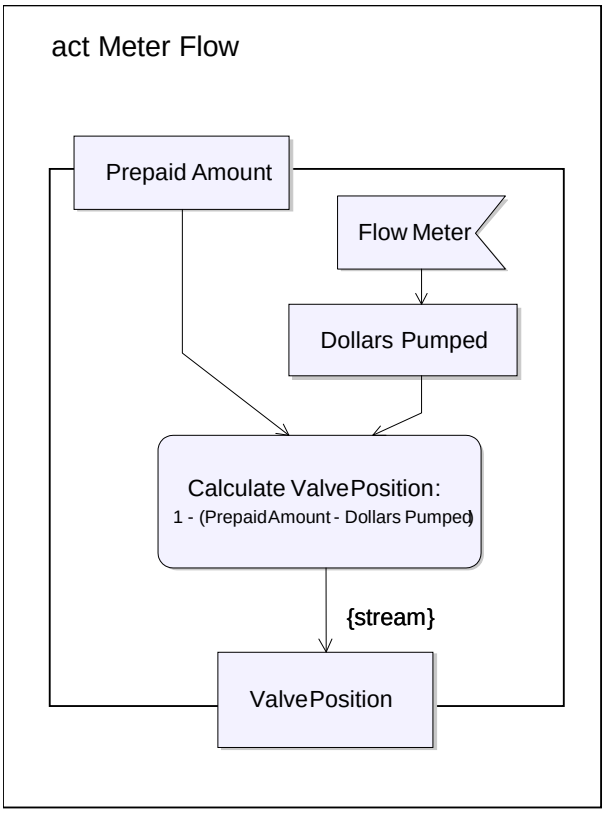

Figure 7. Meter Flow Activity Diagram

Defining Interactions

At this point, we have demonstrated high level activity flow definition, including tracking a system's state, and defining system control. For the high level activity flow already demonstrated, it remains to define the interactions between the actors and the system. It also remains to demonstrate interactions between parts of the system (Overgaard, 1999). Table 4 show the strengths and weaknesses of sequence and activity diagrams for defining interactions. Figure 8 shows a sequence diagram.

Table 4. Strengths (S) and Weaknesses (S) of Behavior Diagrams for Defining Interactions

Sequence Diagrams	
S	• Cleanly defines sequence of interactions as time marches vertically down the page. • Cleanly defines messages or operations along horizontal lines. • Allows for depiction of actions performed in-between interactions. • Allows for depiction of element states in-between interactions.
W	o Eliciting continuous interaction is awkward.
Activity Diagrams	
S	• Shows interactions in the context of activities being performed. • Continuous interaction maybe demonstrated using {stream} stereotype.
W	o Requires usage of swimlanes to identify source & recipient elements, which can make spatial arrangement of complex flow awkward. o Requires use of additional object blocks and lines to show item flow, which can clutter a diagram.

Both sequence and activity diagrams can serve to define interaction as well as activity flow, but the sd defines interactions more naturally. Sequence Diagrams cleanly define a sequence of interactions as time marches vertically down the page. They also cleanly define messages or operations along horizontal lines. On the other hand, activity diagrams require Swimlanes (to allocate behavior) and Data Flow Objects (to show messages or operation calls) to define interactions (Long, 1995). Swimlanes in ad's interfere with the ability to cleanly split a flow for alternate or concurrent activity. If each of the split flow paths include activities performed by the same elements, spatial arrangement becomes awkward. Activity diagram Data Flow Objects may also clutter a diagram.

For these reasons, sd's have been chosen to define interactions for our example case. The sd's weakness in eliciting complex flow is a non-factor in this case, since the Fuel Pump state transitions involve non-complex sequences of activities. Figure 8 elicits the activity flow and interactions for each of the possible transitions between the Idle State and the Authorized State of the Fuel Pump system. The appearance of the <<subject>> system 'Fuel Pump' and actors as lifeline blocks indicates this sd elicits external interactions.

The "alt" construct is used to differentiate between the 'Clerk Activated' transition, and the 'Credit/Debit Card' transition. The similarity of the 'Credit Card' and 'Debit Card' transitions in the smd allow them to be addressed together here. The "alt" construct is also used to differentiate a successful versus unsuccessful Card Authorization, and two different Clerk Activation alternatives.

State Symbols are used on the sd to show the connection to the smd. As all the alternatives begin in the Idle State, the "Idle" state symbol is shown as the first state on the 'Gas Pump' lifeline. At each point in the flow where a change of state occurs, the new state is indicated on the lifeline. A successful instance of the 'Obtain Authorization' uc results in the 'Authorized' State.

The 'Credit/Debit Card' alternative begins with the 'Fuel Pump' in the Idle state sending a message to the Operator to swipe their credit or debit card. The operator transfers their card information to the FP by swiping the card, and the FP forwards the information to a Financial Institution. Depending on the status of the card account, the Financial institution returns an "approved" or "disapproved" message to the Pump. If "approved", the FP requests the Operator to

select a fuel grade and transitions the Pump to the 'Authorized' state; if "disapproved" the Pump sends a message to the Operator indicating an invalid account. If the card was "approved", the FP must store how authorization occurred in "memory", to properly control behavior later in the flow.

In the same manner, sd's may be created for the remainder of the transitions, completing the definition of flow activity and corresponding interactions at the system level for the 'Get Fuel' uc.

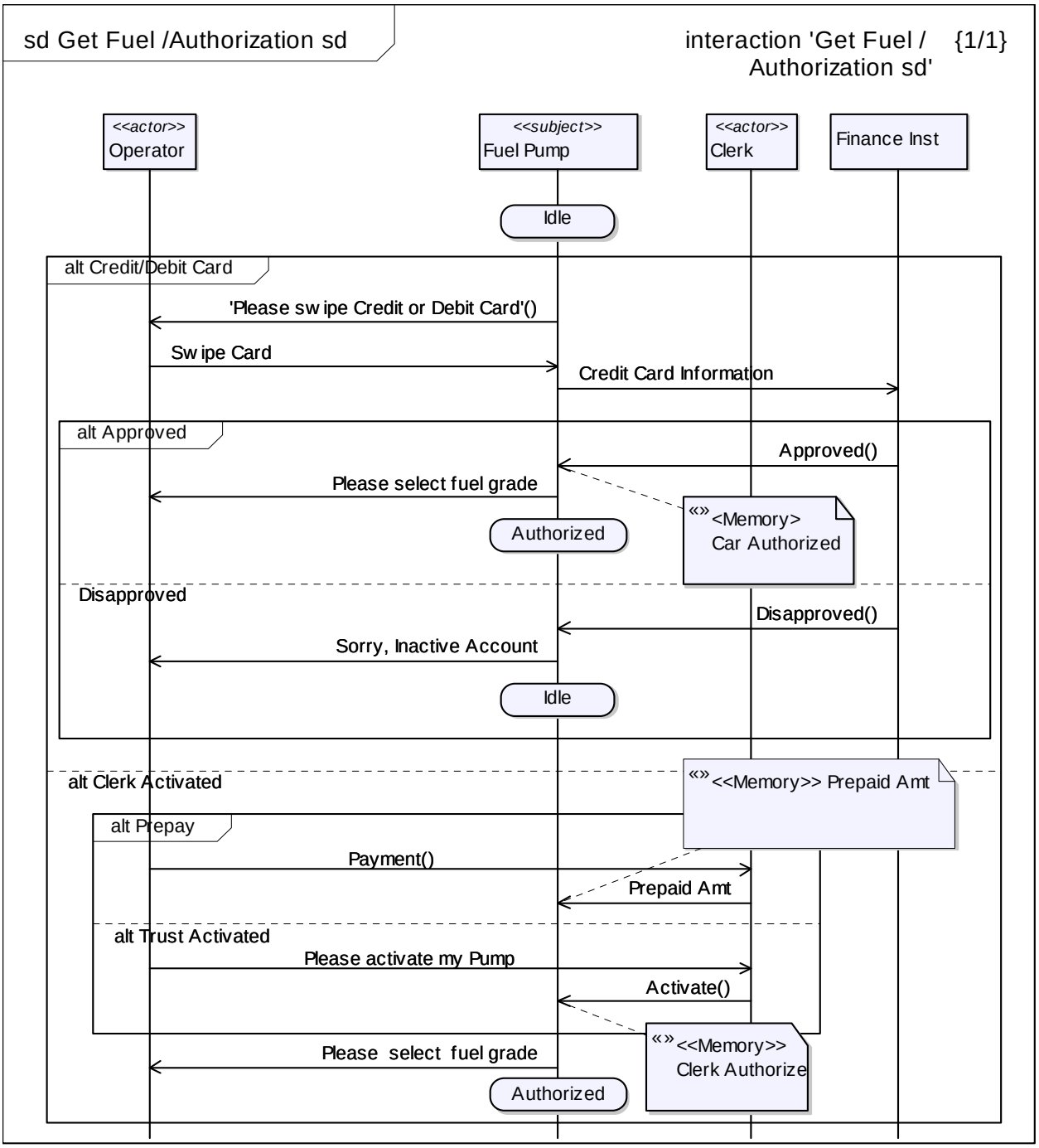

Figure 8. Authorization Sequence Diagram

Allocating Responsibility

Now that system level behavior has been defined, the next step is to formulate a structure for the system. The sequence diagrams provide a guide for modeling the structure. To start, we know that the system needs a component which is able to handle each interaction between itself and an actor (Friedenthal, 2006). Furthermore, the system needs the capacity to transform each interaction input into the proper output, whether it is an internal output to another part of the system, or an external output to another actor.

Defining structure is outside the scope of this paper. However, a few parts will be defined to allow for demonstration of internal interactions and allocation. These parts include a Computer Motherboard (with a built-in Network Card and other I/O ports), a Transaction Display, and a Card Reader. These are the only Fuel Pump parts necessary to define the internal interactions for the Authorization activities for the given level.

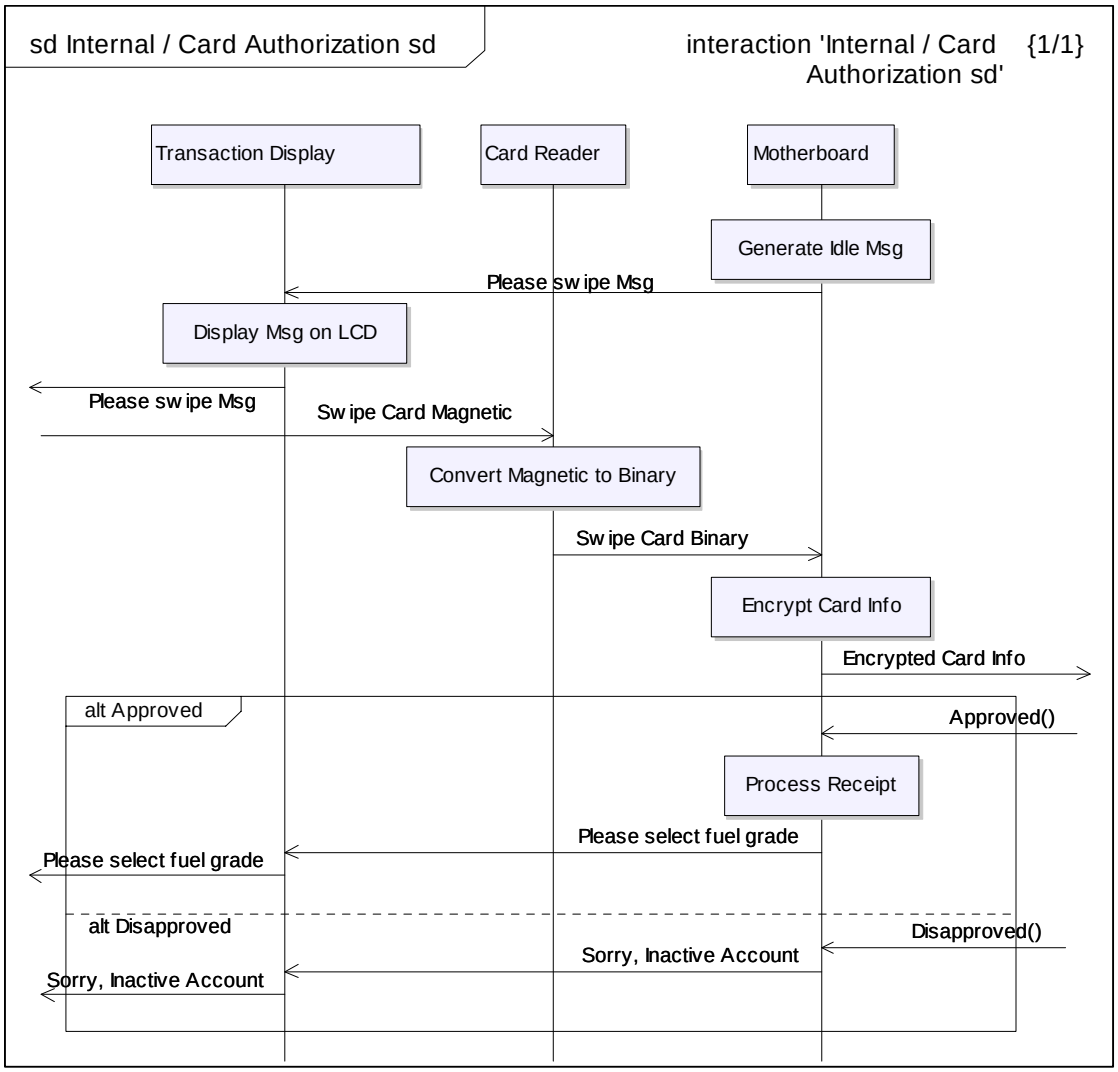

Figure 9. Card Authorization Sequence Diagram

The same diagrams used to define external interactions are used to define internal interactions. Figure 9 displays an "internal" sequence diagram which conforms to the external Authorization sd. This sd is simply a decomposition of the 'Fuel Pump' sd in Figure 8. The messages coming in-and-out of the Pump from Figure 8 should match the messages coming in-and-out of Figure 9.

Allocations are implicitly performed as part of the process of defining internal interactions. Notice that "Action Blocks" such as 'Generate Idle Message' and 'Convert Magnetic to Binary' have been included on some of the lifelines. Those actions which don't map directly to system functional requirements would be classified as Derived Requirements, as they are lower-level functions required for the system to accomplish its system-level behavior.

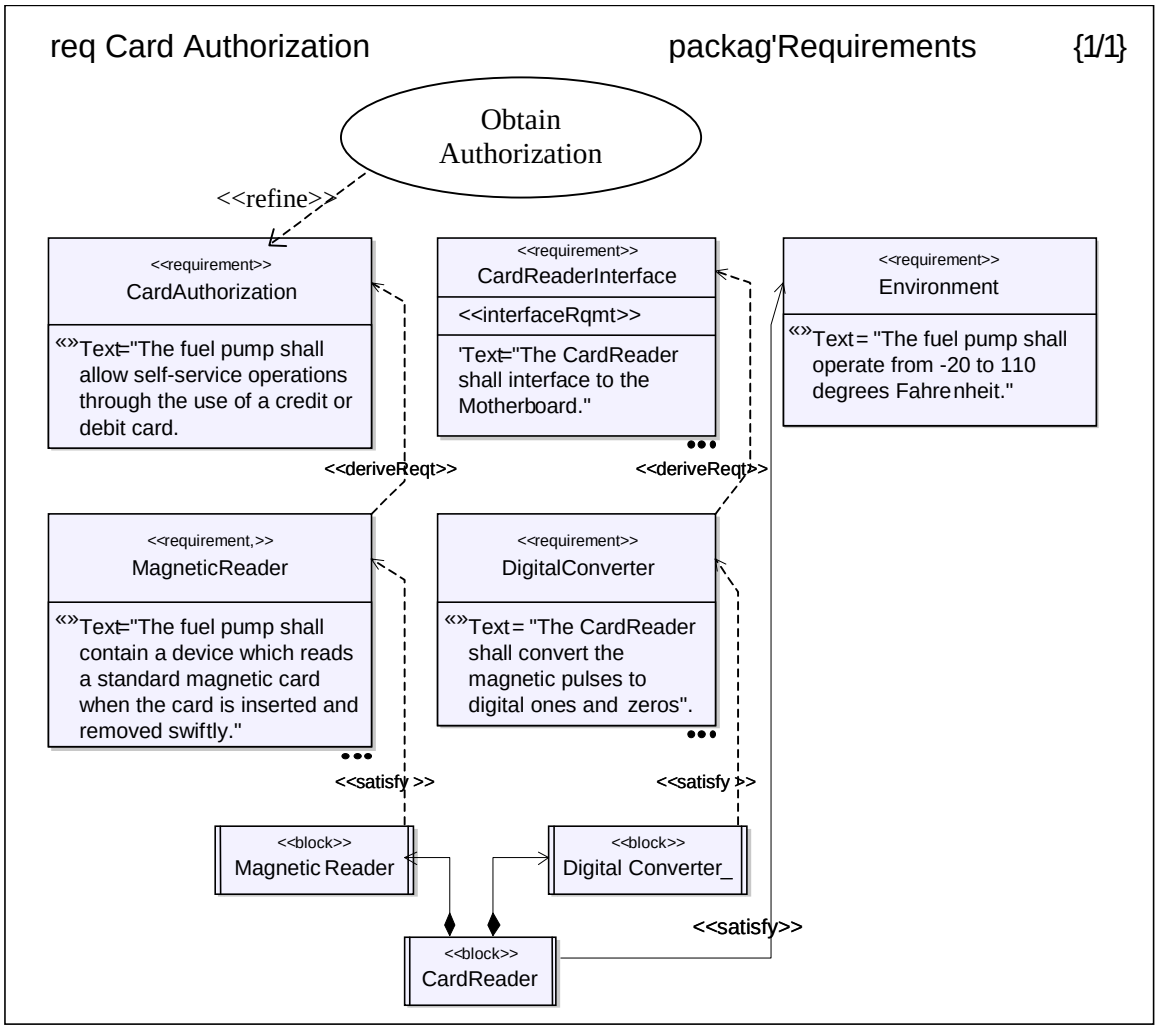

Figure 10. Card Authorization Requirements Diagram

Along with allocation of actions to structure, interactions must be allocated to structure. Interactions are allocated to interfaces, until each message or flow between components is realized by an interface specification. By tracing through a complete

set of interaction diagrams, and ensuring that each flow input and output is allocated to an interface, the SE is assured to capture all the interfaces.

Figure 10 shows a Requirements Diagram which demonstrates the allocation of actions and interactions to structural elements using the <<satisfy>> construct. Also shown is traceability of derived requirements to system requirements. The "Obtain Authorization" uc has been used to <<refine>> the Card Authorization requirement. Modeling of the uc resulted in the MagneticReader <<derived>> requirement, and a CardReader <<block>> which <<satisfies>> this derived requirement. Generation of the Card Reader and Motherboard components led to the creation of the CardReaderInterface <<interfaceSpecification>>. This led to the DigitalConverter derived requirement, and a DigitalConverter component of the Card Reader to satisfy the new requirement. The Card Reader block is also shown as needing to satisfy a system-wide environmental requirement.

Conclusions

Behavioral Diagrams are a key Pillar of the SysML language. Utilizing common visual constructs, they provide a means of visualizing behavior from an actor's perspective down through a detailed design perspective. They lay the framework for defining the structure of the system which can execute the intended behavior. Due to the wide variety of systems which may be addressed using SysML, no fixed methodology of applying behavioral diagrams is likely to work for all systems. An understanding of the strengths and weakness of each diagrams should guide the SE in achieving the purposes of the Behavioral Pillar.

The Use Case Diagram is primarily useful in managing complexity by focusing the SE on one scenario at-a-time. Activity flow definition may be accomplished through various combinations of act's, smd's, and sd's. Act's and smd's are better than sd's for defining complex flows. For systems with states, smd's play a vital role. Ad's are preferred for defining continuous control activity. Sd's are preferred for defining interactions with actors and internal components. Sd's cleanly allocate behavior to structural parts and identify messages flow, setting the stage for structural block diagrams, internal block diagrams, and interface specifications.

References

Bennett, Simon; Skelton, John; Lunn, Ken, *Schaum's Outlines UML Second Edition.* McGraw-Hill Intl, 2001.

Booch, Grady; Rumbaugh, James; Jacobson, Ivar, *The Unified Modeling Language User Guide*. Addison-Wesley, 1999.

Friedenthal, Sanford; Moore, Alan; Steiner, Rick, "OMG Systems Modeling Language Tutorial", Object Management Group, July 11, 2006.

Long, James E., "Relationships Between Common Graphical Representations Used in Systems Engineering", NCOSE Symposium, 1995.

OMG SysML Specification, Object Management Group, May 4, 2006

Overgaard, Gunnar; Bran, Selic; Bock, Conrad, "Object Modeling with UML: Behavioral Modeling Powerpoint Tutorial", Object Management Group, 1999.

Biography

Larry Zdanis currently works at Northrop Grumman AEW/EW in Bethpage New York as a Systems Software Engineer. He has completed the requirements for a M.S. in Systems Engineering from Stevens Institute of Technology, Hoboken, NJ. In 1997, he

obtained a M.S. in Mechanical/Aerospace Engineering at the Joint Institute for the Advancement of Flight Sciences (JIAFS), a collaborative between George Washington University and NASA Langley.

Robert Cloutier received his Ph.D. in Systems Engineering from Stevens Institute of Technology. He currently is a Principal Systems Engineer in Moorestown, NJ. He is responsible for developing and modeling architectures for complex systems, and has over 20 years experience in systems engineering, software engineering, and project management in both commercial and defense industries.

Rob also has an M.B.A. from Eastern College, and a B.S. from the United States Naval Academy. He is an Industry Fellow at Stevens Institute of Technology and an Adjunct Professor for Eastern University. He is a member of the International Council on Systems Engineering (INCOSE) and active in the Delaware Valley Chapter, an active member of the Association of Enterprise Architects, and is a member of IEEE. Rob also chairs the Rowan University Electrical and Computer Engineering Department Industry Advisory Board.