Case study of model-based systems engineering (MBSE): Part 2. Develop data-focused processes for distributed systems analysis and design

Mohit Choudhary November 15, 2011

Distributed systems are inherently data-oriented, with data entities dictating the sub-system boundaries and specific data interaction that defines the dynamic characteristic of a system. The focus on data entities and their behavior in distributed environments cannot be undermined. Thus the derivation of ports and interfaces (data interactions and attributes) being a consequence of functional analysis in a typical MBSE workflow, such as the IBM® Rational® Harmony systems engineering process seems an oddity in such a case. In this article, we explore how to develop an MBSE process suited for analysis and design of distributed systems.

View more content in this series

In the part 1 of this series, we derived the system design of a UAV ground controller by using IBM Rational Harmony systems engineering as a process guiding us to emerge the sub-systems and the logical interfaces. However, the design of distributed systems is often data centric, where data entities take the place of first grade citizens in a system design. Thus it only seems obvious to tweak the Rational Harmony systems engineering process a bit to allow the design process to focus on the data entities and yet bring the goodness of a mature MBSE process like Rational Harmony systems engineering into the design.

In distributed systems design, it is necessary to define these data interactions through an evolved interface language that not only ensures the consistency of the various sub-systems throughout the interaction, but can also capture the interacted intent and behavior of the data set in the language itself. One such step in the evolving interface specification language is the OMG Data Distribution Service (DDS) specification (see Resources). While the hand-off at the end of a standard Rational Harmony systems engineering process (see Resources) is sufficient to spring operational ICD between sub-systems from the derived logical interfaces, mapping these logical interfaces to the information exchange constructs with data distribution service (DDS) might not be straightforward.

In this article, we attempt to tweak the standard Rational Harmony systems engineering process workflow so that it supports the distributed dissonance instead of Rational Harmony. First, you will see constructs of the DDS specification and Problem-frame Analysis (see Resources). Then we follow the steps involved in modified MBSE process that embraces DDS in time and spirit throughout the process of distributed systems analysis and design. Finally, you will be able to run the steps by using the same case study as in part 1 of the article.

Understand DDS and problem frame analysis

The OMG Data Distribution Service (DDS) specification is separated into architectural layers. The lower layer is the Data Centric Publish and Subscribe (DCPS) layer, which contains type-safe interfaces to a publish-and-subscribe communication mechanism. The upper layer is the Data Local Reconstruction Layer (DLRL), which enables application developers to construct a local object model on top of the DCPS layer to shield the application from DCPS knowledge. The context of this article is limited to a few specific constructs of DCPS.

Data-centric publish and subscribe

The DCPS layer disseminates data from publishers to interested subscribers. It is implemented using the concepts of publisher and data writer on the sending side and subscriber and data reader on the receiving side. The DCPS layer consists of one or more data domains, each of which contains publishers and subscribers that communicate via DDS. Each publisher and subscriber belongs to a domain. Within any data domain, data is identified by a topic, which is a type-specific domain segment that allows publishers and subscribers to refer to data unambiguously.

Within a domain, a topic associates a unique topic name, data type, and a set of quality of service (QoS) policies with the data itself. Each topic is associated with one data type, although many different topics can publish the same data type. The behavior of publishers is determined by the QoS policies associated with the publisher, data writer, and topic elements for a particular data source. Likewise, the behavior of subscribers is determined by the QoS policies associated with the subscriber, data reader, and topic elements for a particular data sink. A few of the QoS policies and operations specified in the language and used in the case study are as shown in Table1 and 2 respectively. The QoS policies and operations.

Table 1. Documenting relevant DDS QoS Policies

QoS	Description
Liveliness	Verifies that to make sure expected entities in the system are still alive.
Reliability	Determines the level of reliability required in delivery of the samples.
History	Controls what happens to an instance whose value changes before it is communicated to subscribers.
Lifespan	Avoids delivering "stale" data to the application.
Deadline	Establishes that the topic is expected to have each instance updated periodically within the deadline.

Table 2. Documenting relevant DDS operations

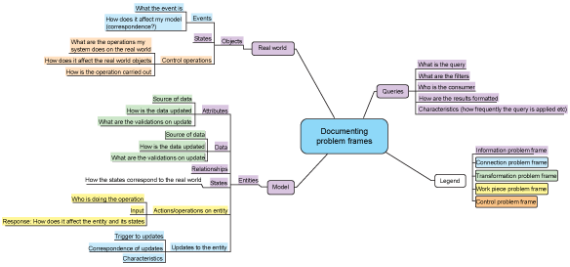
Operation	Description
Read	Accesses a collection of data values from the data reader.
Take	Removes a sample from the data reader so that the read or take operations cannot be performed on it.
Wait set & Listener	Makes the application aware of changes in the DCPS communication status.
Content filter	Filters incoming topic samples based on attributes.
Data_Available	Status change flag that indicates availability of data at the reader.

Read with condition	Has "read" access to the samples that match the criteria specified in the condition. The condition can be a read condition or query condition.

Problem-frame analysis

Problem Frames Approach is an approach to requirements analysis. It enables you to categorize the system requirements as a set of pre-defined problems analogous to design patterns in the solution space. Once categorized, the problems can be easily explained by answering a standard set of questions associated with each problem frame. Figure 1 shows how the technique has been used in this article to document the artifacts of the case study.

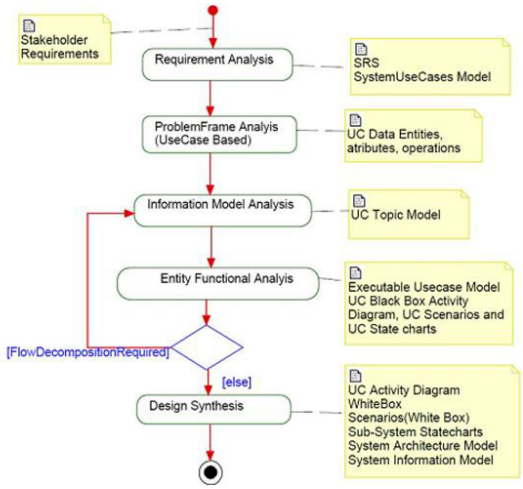
Figure 1. Documenting through problem frames



Proposed work flow

The proposed process workflow is shown in Figure 2.

Figure 2. Workflow for MBSE process



Having defined the system-level use cases in the requirement analysis phase, the process aims at defining the data entities, attributes and operations for each of the system use cases through problem frame analysis. You can use the information problem frame to evolve the model entities and their attributes, the connection and transformation problem frames for defining the behavior of these entities and the work-piece problem frame to evolve the operations on the defined entities.

Next, we need to define the system information model based in the entities that we identified in the problem frame analysis phase. The artifact that you produce through this analysis is the topic model that defines the name, type and QoS of identified DDS topics. Since we have the topic model, in the next phase of entity functional analysis you will focus on performing functional analysis around the life cycle operations of

identified topic model entities. You can use the black box activity diagram to capture the life cycle parallel flows for each of the identified entities. Further, you have to generate the black box use-case scenarios by combining one or more flows to establish the real functional flows as sequence diagrams. Use the sequence to generate the ports and interfaces of the use-case block. Then capture the state based behavior of the use case and verify the generated sequence diagrams and compare them to the black box scenario sequence diagrams.

The design synthesis starts by performing the structural decomposition of the system not only based on the key functions, but also on the model entities themselves. In the next step while describing the white box activity diagram, you need to include DDS-Data-space as one of the sub-system components to patch the independent functional flows. Defining the DDS-Data-space as a sub-system component enables the white box state machine to run as an executable model while preserving the decoupling in space and time desired through the use of DDS. You can now verify this executable model by comparing the sequence diagrams that we generate here against those produced as white box scenarios. Finally you need to generate the system Internal block diagram (IBD) that brings out the white box ports and interfaces. Not surprisingly, the interfaces in this case map one to one to the topics in the information model that has already adequately defined the attributes and their behavior.

Problem frame analysis

The scope of this analysis defined by the Perform Area Search use case is detecting UAVs in flight, assigning search areas to the UAVs, acquiring track data from sensors, and storing this in an information model. The analysis performed is shown in Tables 3 and 4.

Table 3. Information and connection problem frame analysis

Entity	Attributes and description
UAVInfo: Unmanned Ariel vehicle.	1. Real world: models the characteristics of UAVs in flight a. Objects i. UAV b. Events on the objects and reaction in Information model i. UAV not contactable A. UAV position update not available within a deadline period. B. Updates that are lost are lost. 2. Attributes a. Identification i. Vehicle id, sent by UAV ii. No other identification attributes, no system generated b. Time information i. Update time – is the information of time of position update ii. Available flight time – is the information of time left in flight c. UAVstate i. Search assigned ii. Search unassigned d. Sensor information i. List of available sensors with attributes e. Own vehicle data i. Position data ii. Motion data

	3. Data a. No history of updates. Only instantaneous value.
Sensor: Used by the UAV to detect tracks.	1. Types of sensors a. SAR b. FLIR c. OPTICAL 2. Sensor attributes a. Sensor start time – used to determine if the sensor is active. b. State i. Active ii. Inactive
Sensor tracks: A set of measures from a specific target by which the position and motion of the target can be computed. The track exists as a separate information structure only if there is additional data that is required at the level of a track that is not available in a given sample for that track.	1. Real world: models the sensor track information sent by sensor. a. Objects i. Emitters/contacts detected by a sensor ii. Measures sent by the sensor. There is a periodicity associated with the measures. b. Events on the objects and reaction in information model i. Sensor not contactable A. Track measure not available within a deadline period. B. Track measures that are lost are lost, the sensor track continues with measures from the point at which we get connected again. ii. Sensor cannot contact track any more – track measure not available within a deadline period. iii. Sensor regains a track – track measure available again. iv. Distinguish between sensors not able to track vs. sensor not contactable? A. Availability of liveliness status of the sensor 2. Attributes a. Identification i. ID, sent by external sensor – composite of ii. Sensor ID iii. Track-ID, numeric value from 1 to 50. iv. No other identification attributes, no system generated ID b. Time information - <store history of each state transition in terms of timestamp at which it occurred.> i. Created time – is the information from the sensor measure time, first measure for the track. ii. Track age – is the time elapsed since last measure update. c. Track state - depending upon the deadline indication of associated measures i. Active ii. Lost d. Source sensor– from sensor ID in any measure, normally set based on the first sample 3. Data a. Consists of one or more track measures b. Stores history of measures over a 30 minute window c. Measures earlier than this are purged? 4. Sensor specific attributes a. General information about these attributes: Almost all of this data is taken directly from the incoming sensor data and used to display to users.

4. Sensor Track Measures	1. Real world: models a single data sample sent by a sensor 2. Attributes a. Identification – is sent by sensor i. Sensor ID ii. Track ID – sent by sensor b. Measure ID - Sequence number c. Time of the sample – sent by sensor, has to be maintained d. Valid or invalid data (good/bad/delayed) – requires transformation based on data range validations, in turn based on sensor. Invalid measures are rejected by the system. e. Data – a sample can contain one or more of the following: i. Position – Latitude and Longitude ii. Projection used iii. Speed 3. Characteristics a. Track Measures are emitted by sensors at 1 Hz frequency

Table 4. Work-piece problem frame analysis

Entity	Analysis
Static information	1. Is there any static information related to sensor tracks? a. Characteristics of sensors i. The RMS values from sensors for errors etc., typically a table, used in the system for computation
Sensor Tracks as work-piece problems <operations on a realized entity>	1. Operator driven actions a. Creation i. Direct creation from sensor measures A. Track Age calculated from last measure update B. First Measure used to get track ID b. Select a sensor track to be made into a system track 2. Lifecycle related a. Updating a sensor track on new available update (measure) for the track. b. Automatic purging of sensor tracks with no measures in 30 min history. c. Archiving i. No requirement
Sensor Track Measures as work-piece problem	1. Operator driven actions a. Nil – owned by the sensor 2. Lifecycle related a. Indicating missed deadline of update b. Automatic purging of measures older than 30 min history. c. Archiving i. No requirements

Information model analysis

In this step, you need to perform the information model analysis using the information and connection problem frame analysis of the previous phase. The objective of this phase is to identify the DDS topics that represent the data entities and their behavior. Each topic forms the unit of interaction in a DDS environment. The correct representation of its behavior can vastly reduce the sub-systems responsibility of housekeeping and generic management. A representative information model subset of case study is as shown in Figure 2.

An example of this reduction can be seen in the behavior that is defined for the topic "SensorTrackMeasure". The key description (the list of data fields whose value forms the key) defined on this topic is a composite structure composed of SensorID and trackID. Different data values with the same key value represent successive values for the same instance, while different data values with different key values represent different instances. Further, the HistoryQosPolicy of the topic is defined as KEEP_ALL with a depth of 1800 to indicate that at most 1800 samples of each such instance is maintained in the data space (@ 1Hz update for 30 minutes). Finally, the LifespanQosPolicy with a duration corresponding to 30 minutes specifies the maximum duration of validity of the data sample in the DDS space, after which it shall be automatically disposed. Defining such behavior about the SensorTrackMeasure entity unambiguously defines to the DDS service to take over the responsibility of history management for the entity. It would now be redundant to model such functionality into the use case.

Figure 3. Topic model representation

Topic name (description)	Topic definition	QoS policies	QoS rationale	
<<UC01_04 Sensor track measure>>				

This topic shall publish sensor track measure information	**struct idendity
	{
	unsigned long ulsensorID ;
	unsigned long ultrackID;
	};
	typedef unsigned long measure_ID;
	struct SensorTrackMeasure
	{
	Identity ulSourceID; // owner of meassage
	measure_ID ulSeq_no;
	unsigned long ullSystemTimemilliSecs; // current time in milliseconds
	float fLatitudeDeg; // latitude
	float fLongitude Deg; // Longitude
	double dXSpeedMtrs; // X coordinate
	double dYSpeedMtrs; // Y coordinate
	};
	#pragma keylist SensorTrackMeasure ulSourseID**
		Deadline	1 sec	

		Destination order	BY_SOURCE_TIMESTAMP	
		Durability	Volatile	
		History	KEEP_ALL	
		History depth	30	
		Lifespan	1800000ms	
		Liveliness	AUTOMATIC	
		Latency budget	30ms	
		Ownership	SHARED	
		Reliability	BEST_EFFORT	
		Resource limit	Default max_samples, max_instances, max_samples_per_instance (all set to LENGTH_UNLIMITED)	
		Transport priority	1	
		Durability service	Default	
Remarks:				

The representative topic model at figure 2 describes the name, type and QoS policies that are related to the topic "SensorTrackMeasure". We use a similar exercise for the rest of the use case to describe the following topics:

- UAVInfo
- SensorTrack
- SensorTrackMeasure
- Command

It is important to note that not all model entities identified during the problem frame analysis phase have one-to-one correspondence to the topic model.

Entity functional analysis

The input to the entity functional analysis phase is the topic model and the work-piece problem frame analysis model. This phase focuses on performing functional analysis around the lifecycle operations of identified topics. The artifacts produced at this step are the black box activity diagram, scenarios and state charts of the use case. The black box activity diagram for the use case is shown in figure 4, while the representative scenarios and state charts are shown in figures 5 and 6 respectively.

The black box activity diagram represents actions based on the DDS constructs like read, write, content_filter, or read_with_query_condition. These constructs are means of simplifying the functional flows. Bringing such binding into the activity flow is considered essential for achieving functional efficiency through the use of DDS. On the other hand, the black box scenarios are created to represent the real world scenarios by referencing different sequence of generated flows into main sequence representing the scenario. This step is very important to ensure that the requirements as understood through the requirement analysis phase are satisfied. This in turn helps us to perform the sufficiency analysis of the detailed topics and the operations around them.

In case of any mismatch it is considered essential to go back and change the information model until the desired real world functional sequences are achieved. Further, deriving the state machine of the use case is straightforward. The use case state machine is composed of multiple ‘AND' states with each one representing the state behavior of the independent functional flows in the activity diagram. While each of the flows is represented by an AND state and thus independent in its execution, yet these are bound together through event flows from one AND state to other wait sub-states in order to enable the execution of the state machine as a whole. This is necessary to verify the auto-generated sequences through model execution against the black box scenarios produced earlier.

Figure 4. Black box activity diagram (data-oriented)

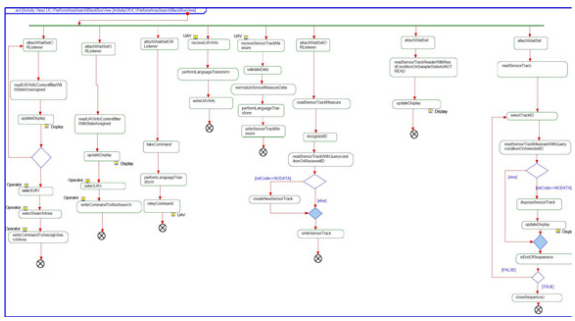


Figure 5. Black box scenarios (data-oriented)

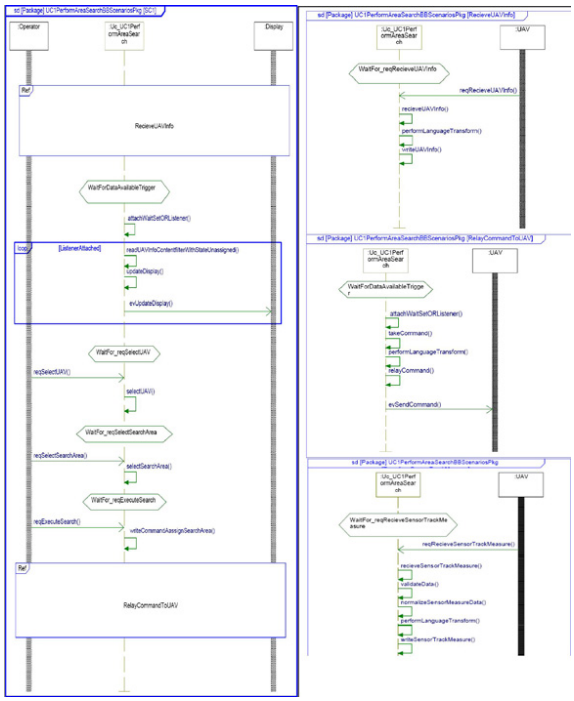
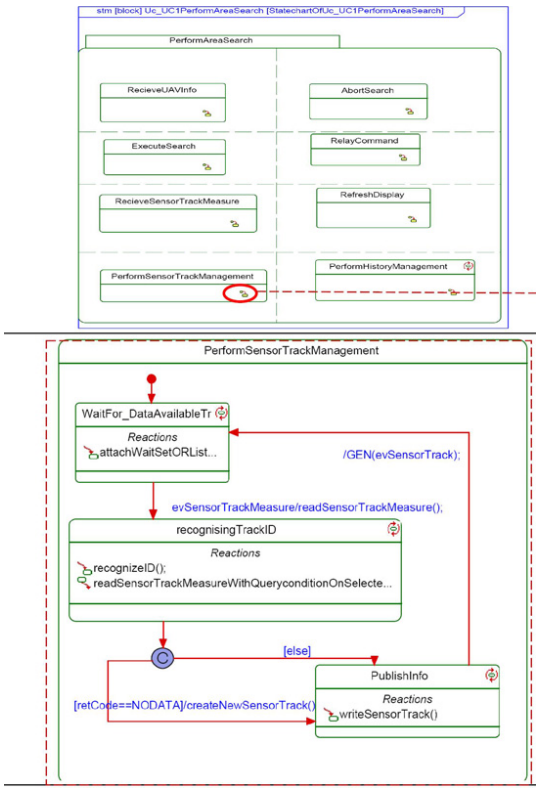


Figure 6. Black box state chart (data-oriented)



Design synthesis

The structural decomposition of the system in this approach is based not only on the identification of key system functions, but also on the derived topic model. Further, the allocation of functions from the white

box activity diagram is performed by allocating a complete flow from the black box activity diagram independently to a swim lane. Such an allocation is possible due to the accessibility of derived topic instances and samples across the sub-system boundaries by means of the DDS global data space.

The derived white box activity diagram for the use case is as shown in Figure 7. The functionality allocated to the swim lane representing DDS data space is that of stitching the rest of the components together through publish/ subscribe paradigm. This representation is considered necessary to bring the sub-systems together to participate in real world scenarios at the model level and also to bring life into the executable white box state machine. The next step in the process is to evolve the white box scenarios for the use case, representing the white box view of the black box scenarios. A representative white box scenario is as shown in Figure 8.

Finally, we derive the ports and interfaces from the white box scenarios and arrive at the white box state behavior of the sub-system components in preparation for the hand-off. The representative state behavior and the white box ports and interfaces of the sub systems are as shown in Figure 9 and 10, respectively.

The sub-system state charts in this case use exactly the same pattern as the corresponding black box ones. The only differences between the two being the events triggering the transition from sub-system wait state are generated by the component DDS-Data-space. The state machine of the DDS-Data-space is a mock representation to enable execution of the different decoupled components. The white box state machines are now executed to compare the generated sequence diagrams against the white box scenarios in order to baseline the model for hand-off. Finally, the sub-system interfaces with unambiguous mapping to the topic model are generated from the base lined model as shown in Table 5.

Figure 7. White box activity diagram (data-oriented)

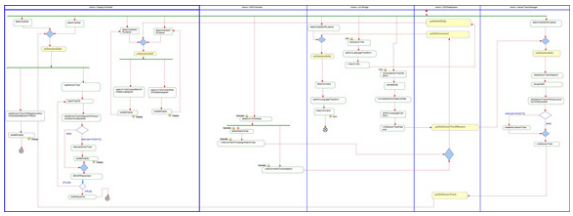


Figure 8. White box sequence diagram (data-oriented)

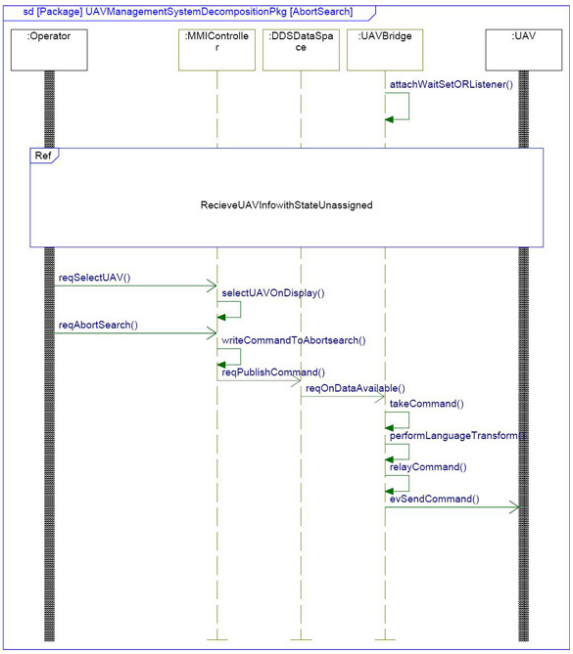


Figure 9. White box state behavior (data-oriented)

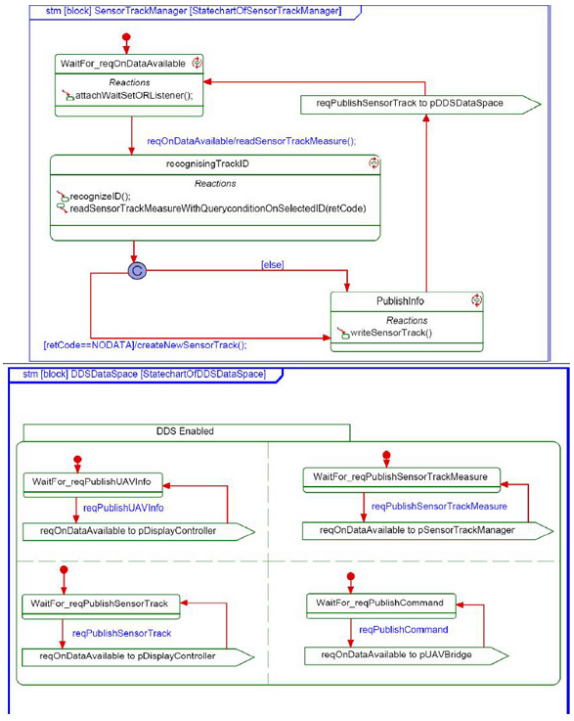


Figure 10. White box ports and interfaces (data-oriented)

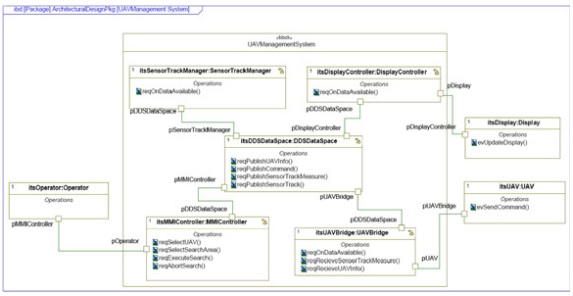


Table 5. White box interface list

Block	Port Name	I/F Type	Interface Event	Topic Name / Description
MMIController	pOperator	Provided	reqSelectUAV	Operator selection through h/w interrupt
			reqAbortSearch	Operator selection through h/w interrupt
			reqSelectSearchArea	Operator selection through h/w interrupt
			reqExecuteSearch	Operator selection through h/w interrupt
	pDDSDataSpace	Required	reqPublishCommand	CommandTopic
SensorTrackManager	pDDSDataSpace	Provided	reqOnDataAvailable	SensorTrackMeasureTopic
		Required	reqPublishSensorTrack	SensorTrackTopic
UAVBridge	pDDSDataSpace	Provided	reqOnDataAvailable	CommandTopic
		Required	reqPublishUAVInfo	UAVInfoTopic
			reqPublishSensorTrackMeasure	SensorTrackMeasureTopic
	pUAV	Provided	reqRecieveUAVInfo	UAVInfo message on UAV link
			reqRecieveSensorTrackMeasure	SensorTrack Measure msg on UAV link
		Required	evSendCommand	Command message on UAV link
DisplayController	pDDSDataSpace	Provided	reqOnDataAvailable	SensorTrackTopic
			reqOnDataAvailable	UAVInfoTopic
	pDisplay	Required	evUpdateDisplay	Interface on display link
DDSDataSpace	pMMIController	Provided	reqPublishCommand	CommandTopic
	pUAVBridge	Provided	reqPublishUAVInfo	UAVInfoTopic
			reqPublishSensorTrackMeasure	SensorTrackMeasureTopic
		Required	reqOnDataAvailable	CommandTopic
	pDisplayController	Required	reqOnDataAvailable	SensorTrackTopic
			reqOnDataAvailable	UAVInfoTopic
	pSensorTrackManager	Provided	reqPublishSensorTrack	SensorTrackTopic
		Required	reqOnDataAvailable	SensorTrackMeasureTopic

Conclusion

The main concern while putting together the architecture of a distributed component-based system is to be able to unambiguously define business functions and their interfaces across sub-systems. We can adequately address these concerns if the system embraces the following Open Architecture principles of service-oriented architecture (SOA) (see Resources):

- Modularity: This implies architecture that has carefully partitioned business and technical functions in a way that allows them to be independently accessed with minimal need to maintain state between interactions. The use of publish and subscribe paradigm in the design of a distributed system inherently promotes use of stateless nature of sub-system design. As is evident from the case study in the paper, each of the independent activity flows naturally represent an AND state of the component, while the only predominant state in each AND state being the wait state for each flow. The behavior is largely stateless after that resulting in the creation or update of a defined entity which itself is exposed to rest of the system through a write operation and never preserved. Such a design naturally exhibits the properties of a modular system that preserves minimal state between interactions.

- Open standards: The use of open standards most affects the distributed system design where those standards have to do with service description, discovery, and access of functionality. SysML is a modeling language based on open standard and so is DDS specification. SysML is the language for unambiguous definition of system or sub-system functionality. While, DDS is a language for unambiguously defining sub-system interfaces it also magically encapsulates within itself the mechanisms of discovery and access.

- Interoperability: In distributed systems, interoperability relies on well-defined interface syntax and semantics. DDS as an interface language not only clearly exposes the interface capability in what it does, but also the relevant semantics and behavior of the data model unambiguously. This leaves no scope of misinterpretation of data by applying the wrong context to its interpretation.

Thus getting the two enablers viz. DDS and SysML together in a MBSE process seems like a definite success story for distributed systems analysis and design, one that closely follows the path defined by a mature process like Rational Harmony systems engineering that itself is based on systems engineering best practices.