在本系列的 第 1 部分 中，我们获得了 UAV 地面控制器的系统设计，我们使用 IBM Rational Harmony 系统工程作为一个流程，指引我们了解子系统和逻辑接口。不过，分布式系统的设计往往以数据为中心，而数据实体在系统设计中又占据最重要的位置。因此，很显然，我们只好稍微调整一下 Rational Harmony 系统工程流程，让设计流程把重点放在数据实体上，同时继续将 Rational Harmony 系统工程等成熟的 MBSE 流程的优势融入设计中。

在分布式系统设计中，使用一个先进的接口语言来定义这些数据交互是有必要的，这样做不仅可以在整个交互过程中确保各子系统的一致性，还可以捕获设置在语言本身中的数据的交互目的和行为。在不断变化的接口规范语言中，类似的步骤是通过 OMG 数据分发服务 (Data Distribution Service, DDS) 规范（参阅参考资料）实现。在派生的逻辑接口中的子系统之间弹出操作性 ICD（界面控制文件） 时，标准的 Rational Harmony 系统工程流程结束时的切换 （参阅参考资料）已经足够用，但是，在利用数据分发服务 (DDS) 将这些逻辑接口映射到信息交换结构时，可能并不简单。

在本文中， 我们将尝试调整标准的 Rational Harmony 系统工程流程的工作流， 让它支持分布式不协调性，而不是支持 Rational Harmony 。首先，我们将介绍 DDS 规范和 Problem-frame Analysis 的结构（请参阅参考资料）。然后，我们遵循修改过的 MBSE 流程中所涉及的步骤，这些步骤及时采用了 DDS ，并在整个分布式系统的分析和设计过程中体现它。最后，您应该能够通过使用与本文 第 1 部分 中相同的案例研究来运行这些步骤。

了解 DDS 和问题框架分析

OMG 数据分布服务 (Data Distribution Service, DDS) 规范被划分为两个架构层次。下层是以数据为中心的发布和订阅 (Data Centric Publish and Subscribe, DCPS) 层， 其中包含了发布和订阅通信机制的类型安全的接口。上层是数据本地重构层 (Data Local Reconstruction Layer, DLRL) ，它使应用程序开发人员能够在 DCPS 层上构建本地对象模型，以屏蔽应用程序对 DCPS 的感知。本文的内容仅限于 DCPS 的一些特定结构。

回页首

以数据为中心的发布和订阅

DCPS 层将数据从发布者传播到感兴趣的订阅者。它的实现所使用的概念是， 发布者 和数据编写器 和在发送端，而 订阅者 和数据读取器 在接收端。 DCPS 层由一个或多个数据域组成，其中每个域都包含通过 DDS 进行通信的发布者和订阅者。每对发布者和订阅者都从属于一个域。在所有数据域中，都是根据 主题 来识别数据，主题是一个类型特定的域段，使发布者和订阅者能够明确地指定数据。

在一个域中，每个主题都将惟一的 主题名称、数据类型 和一组 服务质量 (QoS) 策略与数据相关联。每个主题都与一个数据类型相关联， 但多个不同主题可以发布相同的数据类型。 发布者的行为由与发布者、 数据创建者和特定数据源的主题元素关联的 QoS 策略决定。 同样， 订阅者的行为由与订阅者、 数据读取器和特定数据接收器的主题元素关联的 QoS 策略决定。 可以在语言中

指定并在案例研究中使用的一些 QoS 策略和操作，如表 1 和表 2 所示。QoS 策略和操作如下所示。

表 1. 记录相关的 DDS QoS 策略

QoS	描述
Liveliness	验证，确保系统中的预期实体仍然活着
Reliability	确定样本交付所需的可靠性水平。
History	如果实例的值在与订阅者通信之前发生变化，则控制对该实例的处理
Lifespan	避免将 “过时” 的数据提供给应用程序。
Deadline	确定主题预计在期限内定期更新每个实例。

表 2. 记录相关的 DDS 操作

操作	描述
Read	数据读取器对数据值集合的访问。
Take	从数据读取器删除一个样本，这样就不能对其执行读或获取操作。
Wait set & Listener	使应用程序了解 DCPS 通信状态的变化。
Content filter	基于属性筛选传入的主题样本。
Data_Available	状态变化标志，指示在读取器中的数据可用性。
Read with condition	对符合在条件中所指定标准的样本具有 “读” 访问权限。条件可以是只读条件或查询条件。

回页首

问题框架分析

Problem Frames Approach （问题框架方法）是需求分析的方法。它使您能够将系统需求归类为一组预定义的问题，类似于解决方案范畴的设计模式。在将问题归类后，就可以通过解答与每个问题框架相关的一套标准题目来轻松地解释这些问题。图 1 显示了本文如何使用该技术来记录案例研究的构件。

图 1. 通过问题框架进行记录

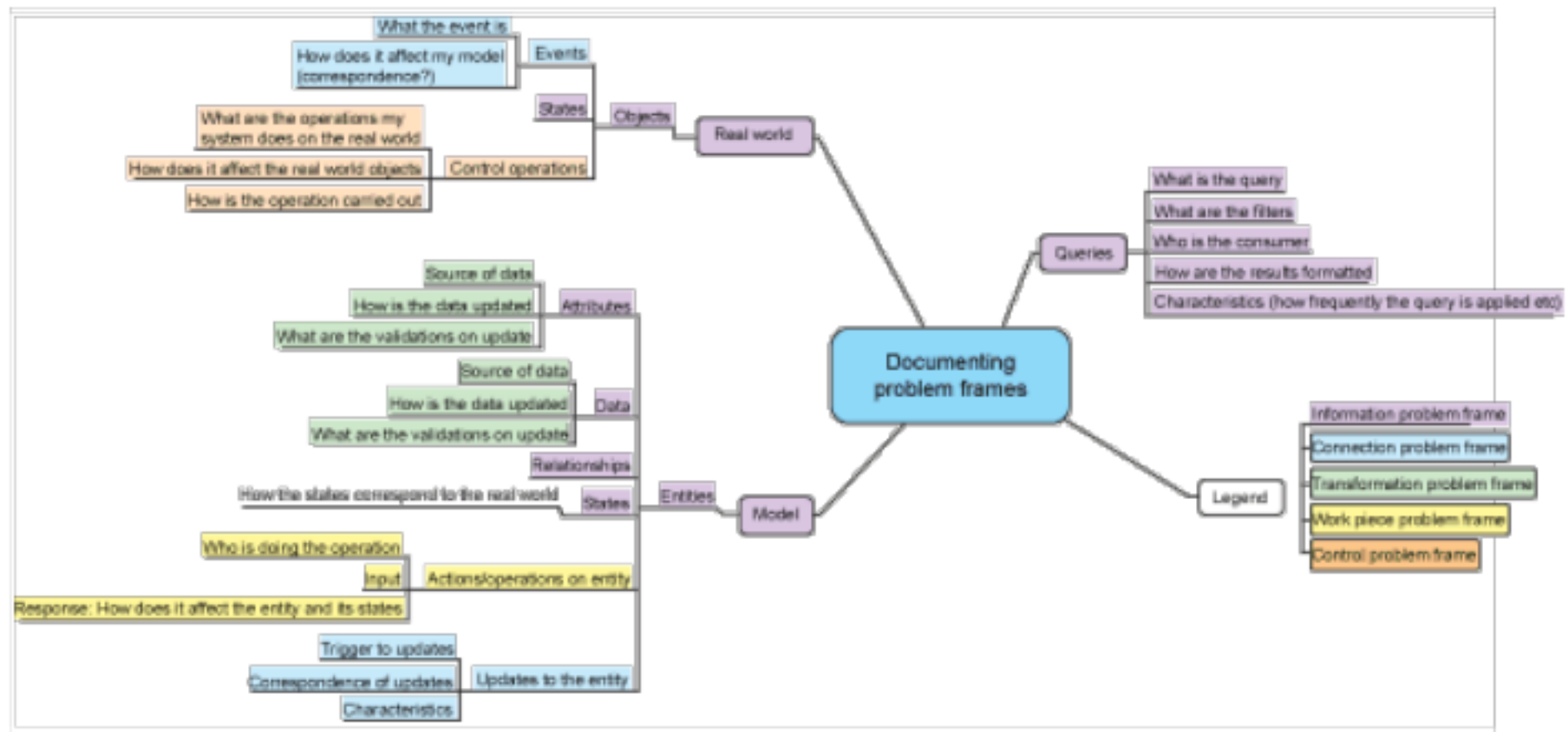


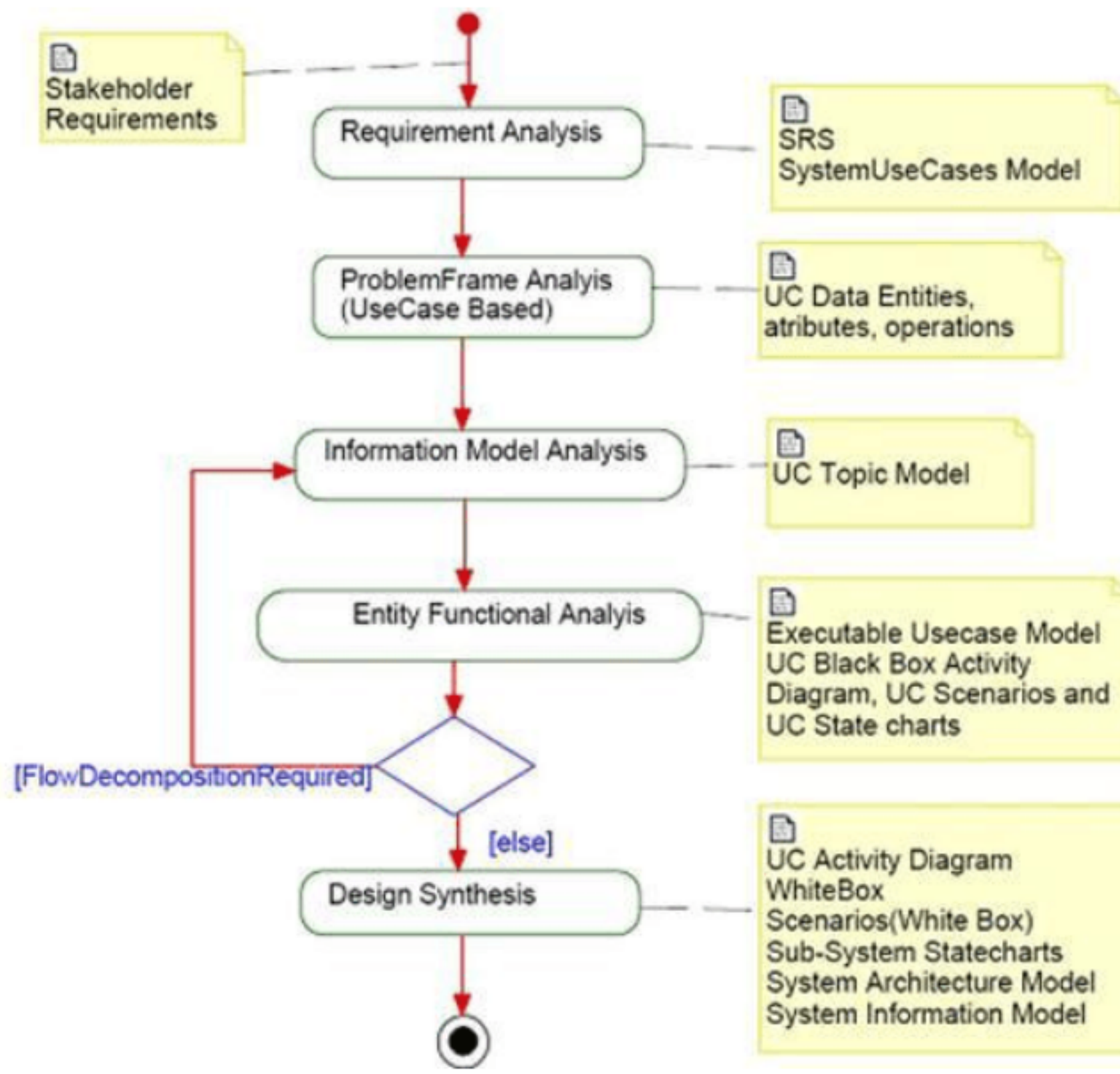
图 1 的大图

回页首

建议的工作流

建议的处理工作流如图 2 所示。

图 2. MBSE 流程的工作流



在需求分析阶段已定义了系统级用例，该流程旨在通过问题框架分析如何为每个系统用例定义数据实体、属性和操作。您可以使用这些信息问题框架来开发模型实体及其属性、连接和转换问题框架，然后定义这些实体的行为和工件问题框架，从而在已定义的实体上进行开发操作。

接下来，我们需要根据在问题框架分析阶段确定的实体来定义系统信息模型。通过分析所产生的构件是一个主题模型，它定义了已确定的 DDS 主题的名称、类型和 QoS 。因为我们已有了主题模型，所以在下一阶段的实体功能分析中，将重点介绍如何围绕已确定主题模型实体的生命周期操作来执行功能分析。您可以使用黑盒活动图来捕获每个已确定实体的生命周期并行流。此外，您必须通过组合一个或多个流来建立与序列图一致的真正功能流，从而生成黑盒用例场景。可以使用序列生成用例块的端口和接口。然后捕获用例的基于状态的行为，验证所生成的序列图，并将它们与黑盒场景序列图进行比较。

设计合成从执行系统结构分解开始，其依据不仅是关键功能，还有模型实体本身。在下一步中，在描述白盒活动图的同时，还需要将 DDS-Data-space 作为其中一个子系统组件包含在内，以修补独立的功能流。先将 DDS-Data-space 定义为一个子系统组件，然后使白盒状态机作为一个可执行模型来运行，同时保留在使用 DDS 过程中所要求的时间和空间的分离。现在，通过比较我们在这里生成的序列图与白盒场景所产生的序列图，可以验证这个可执行模型。最后，您需要生成系统内部结构框图 (IBD) ，它将产生白盒端口和接口。毫不奇怪，在本例中的接口与信息模型中的主题是一一对应的，这些主题已经充分定义了属性及其行为。

回页首

问题框架分析

该分析的范围由 Perform Area Search 用例定义，它检测飞行中的 UAV ，将搜索区域分配给 UAV ，从传感器获取追踪信息数据，并将该信息存储在信息模型中。所执行的分析如表 3 和表 4 所示。

表 3. 信息和连接问题框架分析

实体	属性和描述
UAVInfo：无人驾驶飞机。	1. 真实世界：对飞行中的无人机的特点进行建模 a. 对象 i. UAV b. 在信息模型中，对象的事件和反应

实体	属性和描述
	i. 无法联系 UAV A. 在限期内没有提供 UAV 位置更新。 B. 丢失的更新都将丢失。 2. 属性 a. Identification i. Vehicle id （飞机 id），由 UAV 发送 ii. 没有其他身份属性， 没有系统生成的 ID b. Time information （时间信息） i. Update time（更新时间） – 是时间和位置更新的信息 ii. Available flight time （可用飞行时间） – 是在飞行中剩余时间的信息 c. UAVstate （UAV 状态） i. Search assigned （已分配的搜索） ii. Search unassigned （未分

实体	属性和描述
	配的搜索） d. Sensor information（传感器信息） i. 可用传感器及其属性的清单 e. Own vehicle data （自己的飞机数据） i. Position data （位置数据） ii. Motion data （移动数据） 3. 数据 a. 没有更新历史。只有瞬时值。
Sensor（传感器）：由 UAV 用于检测追踪信息。	1. 传感器类型 a. SAR b. FLIR c. OPTICAL 2. 传感器属性 a. Sensor start time （传感器启动时间） – 用于确定传感器是否活动。 b. State（状态） i. Active （活动） ii. Inactive （不活动）

实体	属性和描述
Sensor tracks （传感器追踪信息）：一组来自特定目标的测量值，可以使用它们来计算目标的位置和运动。 追踪信息作为独立信息结构存在，除非在针对该追踪信息而给出的样本中没有提供追踪信息级别所需的其他数据。	1. 真实世界：对传感器发出的传感器追踪信息进行建模。 a. 对象 i. 由传感器检测到发射器 /触点 ii. 由传感器发送的测量值。 有一个与测量值相关的周期。 b. 在信息模型中，对象的事件和反应 i. 无法联系传感器 A. 在限期内无法提供追踪信息测量值。 B. 丢失的追踪信息测量值都将丢失，从我们再次获得连接开始，传感器追踪信息继续发送测量值。 ii. 传感器无法再获得追踪信息 - 在限期内无法提供追踪信息测量值。 iii. 传感器重新获得追踪信息 - 追踪信息测量值再次可用。 iv. 传感器无法获得追踪信息

实体	属性和描述
2.	与无法联系传感器之间的区别？ A. 传感器活动状态的可用性 属性 a. Identification （身份） i. ID，由外部传感器发送 – 由以下属性组成 ii. Sensor ID （传感器 ID） iii. Track-ID （追踪信息 ID），数字 1 至 50。 iv. 没有其他身份属性， 没有系统生成的 ID b. Time information （时间信息） - < 用每次转换时的时间戳存储每个状态转换的历史记录。 > i. Created time （创建时间） – 是来自传感器测量时间的信息，指追踪信息的第一次测量时间。 ii. Track age （追踪时间长度） - 是自上次测量值更新所经过的时间。 c. Track state （追踪状态） - 取决于

实体	属性和描述
	相关测量值的期限标志 i. Active（活动） ii. Lost（丢失） d. Source sensor（源传感器）– 来自任何测量值的传感器 ID，通常根据第一个样本进行设置 3. 数据 a. 由一个或多个追踪信息测量值组成 b. 存储超过 30 分钟窗口的测量值历史 c. 清除早于该时间段的测量值？ 4. 传感器特定的属性 a. 这些属性的基本信息：几乎所有这些数据都直接从传入的传感器数据中取出，并用于向用户显示。
4. Sensor Track Measures（传感器追踪信息测量值）	1. 真实世界：对传感器发送的单一数据样本进行建模 2. 属性 a. Identification（身份）– 由传感器

实体	属性和描述
	发送 i. Sensor ID （传感器 ID） ii. Track ID （追踪信息 ID） – 由传感器发送 b. Measure ID （测量值 ID）- 序列号 c. Time of the sample （采样的时间） – 由传感器发送，必须保存 d. 有效或无效的数据（好 /坏/延迟） - 需要根据数据范围验证来转换， 即基于传感器进行转换。系统会拒绝无效的测量值。 e. Data（数据） – 一个样本可以包含一个或多个以下属性： i. Position （位置） –Latitude （纬度）和 Longitude （经度） ii. Projection used （使用的投影） iii. Speed （速度） 3. 特征 a. 传感器以 1 Hz 的频率发送 Track Measures （追踪信息测量值）

表 4. 工件问题框架分析

实体	分析
Static information （静态信息）	1. 有否任何静态信息与传感器追踪信息相关？ a. 传感器的特性 i. 来自传感器的表示错误等信息的 RMS值，一般是一个表， 在系统中用于计算
工件出问题时的 Sensor Tracks （传感器追踪信息）<在已实现的实体上的操作>	1. 操作员驱动的操作 a. 创建 i. 从传感器测量值直接创建 A. 从上次测量值更新计算的Track Age （追踪时间长度） B. 用于获取追踪信息 ID 的First Measure （首次测量） b. 选择一个要转换为系统追踪信息的传感器追踪信息 2. 生命周期相关的操作 a. 根据追踪信息新的可用更新（测量值）更新传感器追踪信息 b. 自动清除在 30 分钟历史记录内没有任何测量值的传感器追踪信息。

实体	分析
	c. 归档 i. 无需求
工件出问题时的 Track Measures（传感器追踪信息测量值） Sensor	1. 操作员驱动的操作 a. 无 – 由传感器拥有 2. 生命周期相关的操作 a. 显示错过期限的更新 b. 自动清除超过 30 分钟历史的测量值。 c. 归档 i. 无需求

回页首

信息模型分析

在这一步中，您需要使用前一阶段的信息和连接问题框架分析来执行信息模型分析。这一阶段的目标是，确定代表数据实体及其行为的 DDS 主题。所有这些主题共同构成 DDS 环境中的交互单元，其行为的正确表示可以大大减轻子系统在维护和通用管理方面的责任。案例研究的一个有代表性的信息模型子集如图 2 所示。

在为主题 “SensorTrackMeasure” 定义的行为中，可以看到这种责任减少的示例。在这个主题上定义的键描述（其值构成键的数据字段列表）是一种复合结构，由 SensorID 和 TrackID 组成。具有相同键值的不同数据值代表相同实例的连续值，而具有不同键值的不同数据值则代表不同的实例。此外，主题的 HistoryQosPolicy 被定义为 KEEP_ALL，其深度为 1800，这表示在数据空间中为每个这样的实例最多保存 1800 个样本（按照 @ 1Hz 的频率更新 30 分钟时间）。最后，持续时间对应为 30 分钟的 LifespanQosPolicy 指定数据样本在 DDS 空间的最长有效时间，这段时间过去后，系统会自动处理该样本。有关 SensorTrackMeasure 实体这种行为的定义，明确地定义了 DDS 服务接管实体的历史管理责任。现在，在用例中对该功能进行建模会是重复操作。

图 3. 主题模型表示

主题名称（描述）	主题定义	QoS策略	QoS理论值

<<UC01_04 Sensor track measure>>

本主题将发布传感器追踪信息测量值信息	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 SensorTrackMeasureulSourseID
		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

主题名称（描述）	主题定义	QoS 策略	QoS 理论值
		Durability service	Default

备注：

图 2 中有代表性的主题模型描述了与主题 “SensorTrackMeasure” 相关的名称、类型和 QoS 策略。我们使用一个类似的练习在用例的其余部分中描述以下主题：

- UAVInfo
- SensorTrack
- SensorTrackMeasure
- Command

这里需要重点注意的是，并非所有在问题框架分析阶段中确定的模型实体都与主题模型存在一一对应关系。

实体功能分析

实体功能分析阶段的输入是主题模型和工件问题框架分析模型。这个阶段的重点是围绕已确定主题的生命周期操作来执行功能分析。在此步骤中产生的构件是用例的黑盒活动图、场景和状态图。用例的黑盒活动图如图 4 所示，而有代表性的场景和状态图分别如图 5 和图 6 所示。黑盒活动图表示基于 read、write、content_filter 或 read_with_query_condition 等 DDS 结构的操作。这些结构是简化功能流的一种途径。可以将这样的绑定带入活动流程，这被认为是通过使用 DDS 实现功能效率的必要条件。另一方面，可以创建黑盒场景，用它们代表现实世界的场景，将所产生的流的不同序列引入到代表该场景的主序列中。为了确保能够满足需求分析阶段所了解的需求，这一步非常重要。反过来，这将有助于我们对详细主题及围绕这些主题的操作进行效能分析。

如果出现不匹配的情况，则有必要回头修改信息模型，直到实现所需的现实世界功能序列。此外，获得用例的状态机也很简单。用例的状态机由多个 'AND' 状态组成，每个 AND 状态分别代表活动图中的独立功能流的状态行为。虽然每个流都由一个 AND 状态表示，并因此可以独立执行，但这些流都通过从一个 AND 状态到其他等待子状态的事件流被捆绑在一起，以便支持作为一个整体状态机来执行。通过模型执行，参照之前生成的黑盒场景来验证自动生成的序列，这样做是有必要的。

图 4. 黑盒活动图（面向数据）

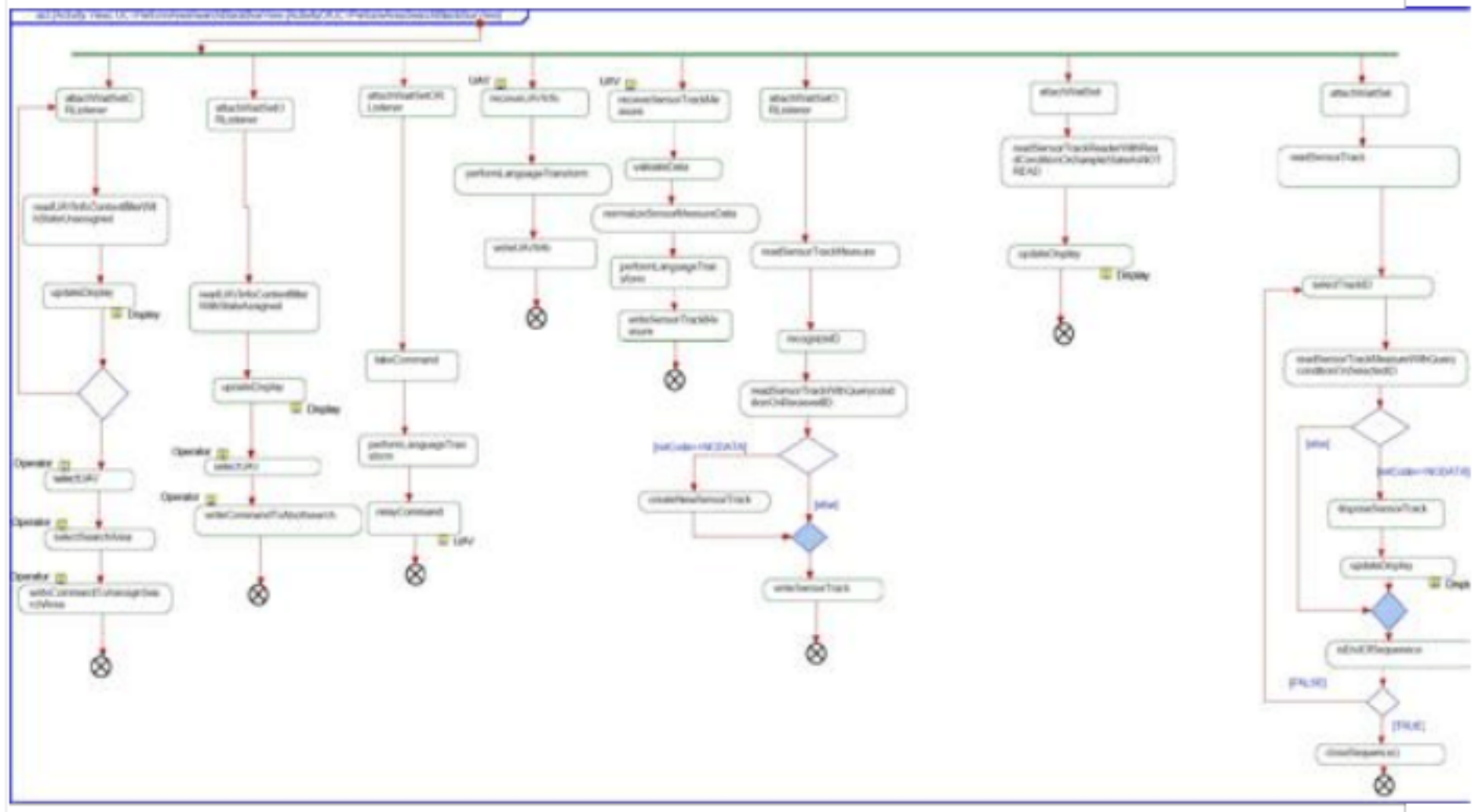


图 4 的大图

图 5. 黑盒场景（面向数据）

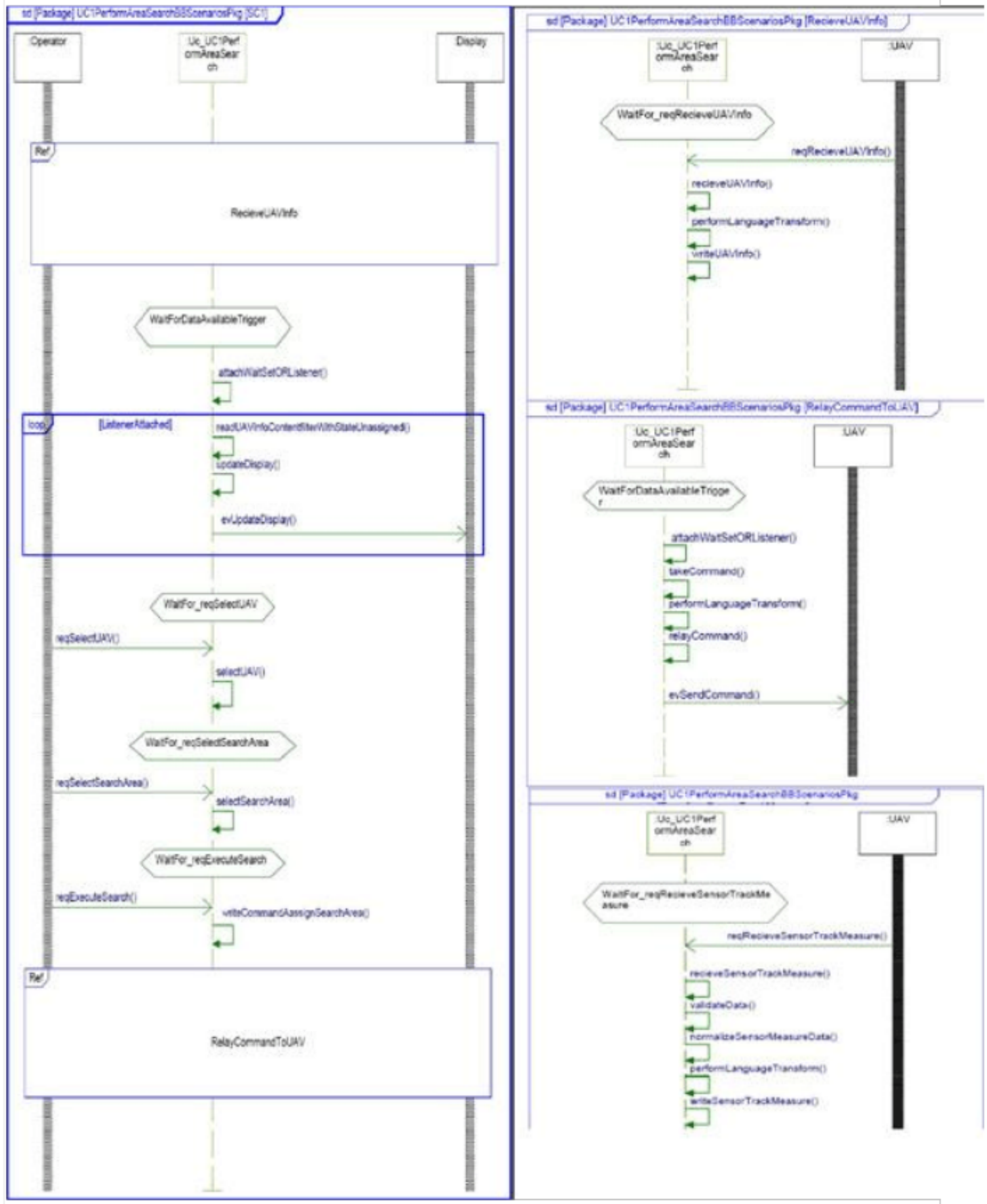


图 5 的大图

图 6. 黑盒状态图（面向数据）

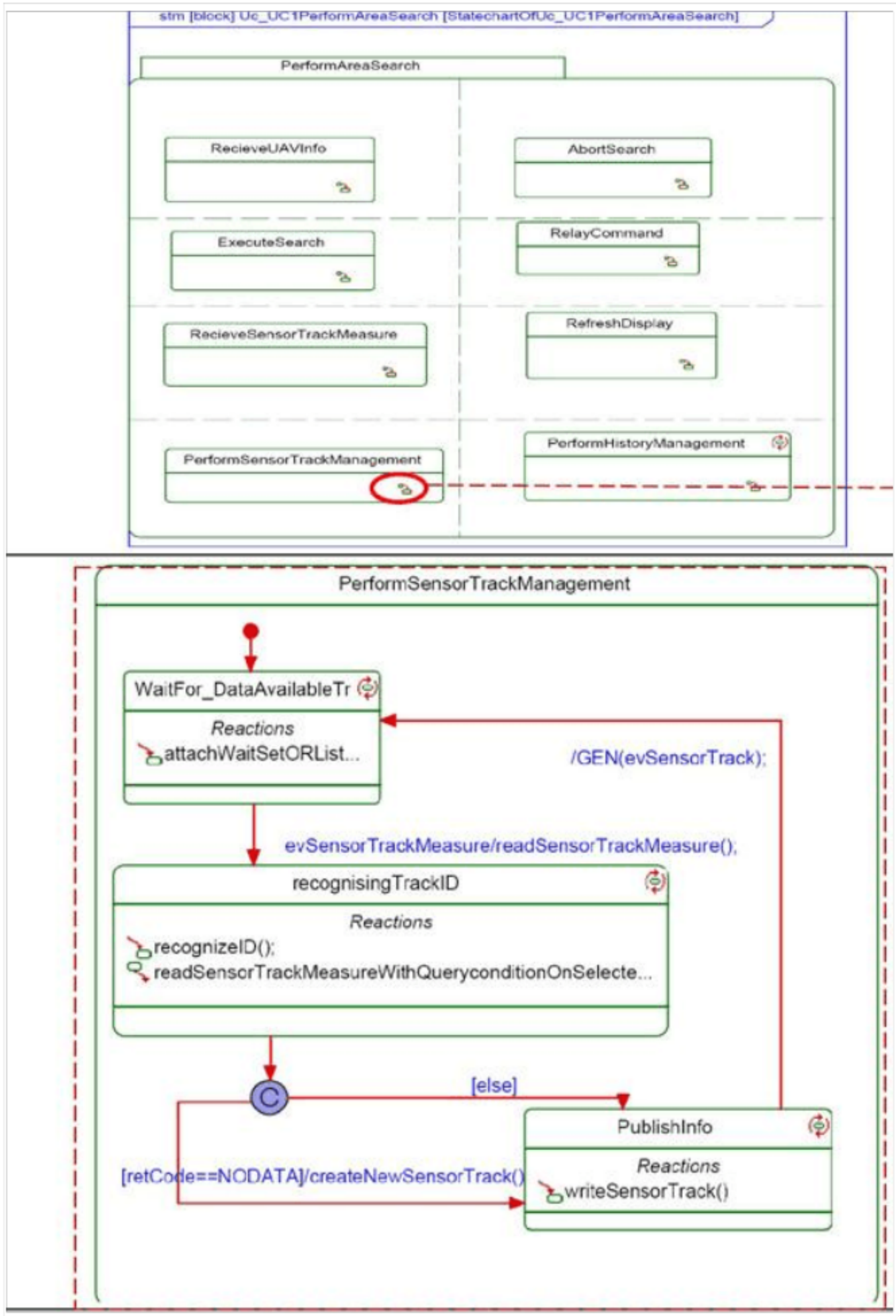

stm [block] Uc_UC1PerformAreaSearch [StatechartOfUc_UC1PerformAreaSearch]
PerformAreaSearch
RecieveUAVInfo
AbortSearch
ExecuteSearch
RelayCommand
RecieveSensorTrackMeasure
RefreshDisplay
PerformSensorTrackManagement
PerformHistoryManagement
PerformSensorTrackManagement
WaitFor_DataAvailableTr
Reactions
attachWaitSetORList...
/GEN(evSensorTrack);
evSensorTrackMeasure/readSensorTrackMeasure();
recognisingTrackID
Reactions
recognizeID();
readSensorTrackMeasureWithQueryconditionOnSelecte...
C
[else]
PublishInfo
Reactions
writeSensorTrack()
[retCode==NODATA]/createNewSensorTrack()

图 6 的大图

设计合成

在这种方法中的系统结构分解不仅基于关键系统功能的识别，还基于已获得的主题模型。此外，白盒活动图的功能分配的执行通过将黑盒活动图的完整流独立分配到一个泳道来完成。这种分配是可以实现的，因为通过 DDS 全球数据空间可以跨子系统边界访问所获得的主题实例和样本。为用例所生成的白盒活动图如图 7 所示。分配给代表 DDS 数据空间的泳道的功能，是通过发布/订阅模式将其余组件拼接在一起的功能。为了使子系统在模型级别共同参与现实世界的场景，并且使可执行白盒状态机生效，这种表示方式被认为是必要的。该流程的下一步是发展用例的白盒场景，代表黑盒子场景的白盒视图。有代表性的白盒场景如图 8 所示。

最后，我们从白盒场景推导出端口和接口，并实现子系统组件的白盒状态行为，以准备切换。有代表性的子系统的状态行为以及白盒端口和接口分别如图 9 和图 10 所示。

本例中的子系统状态图使用与相应黑盒状态图完全相同的模式。两者之间仅有的区别是从子系统等待状态过渡的触发事件，这些事件由组件 DDS-Data-space 产生。DDS-Data-space 的状态机是一种模拟表示形式，用于支持不同分离组件的执行。立即执行白盒状态机，以便比较生成的序列图与白盒场景，从而针对切换确定模型的基线。最后，明确映射到主题模型的子系统接口是根据确定基线的模型生成的，如表 5 所示。

图 7. 白盒活动图（面向数据）

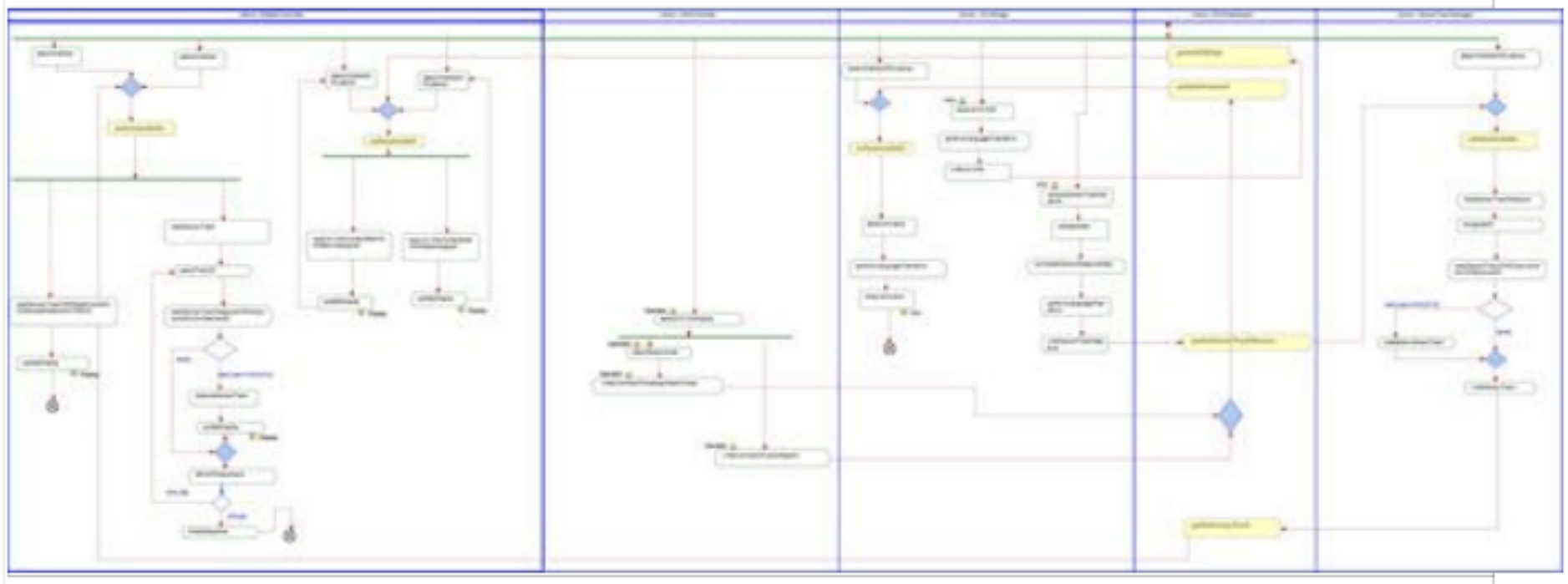


图 7 的大图

图 8. 白盒序列图（面向数据）

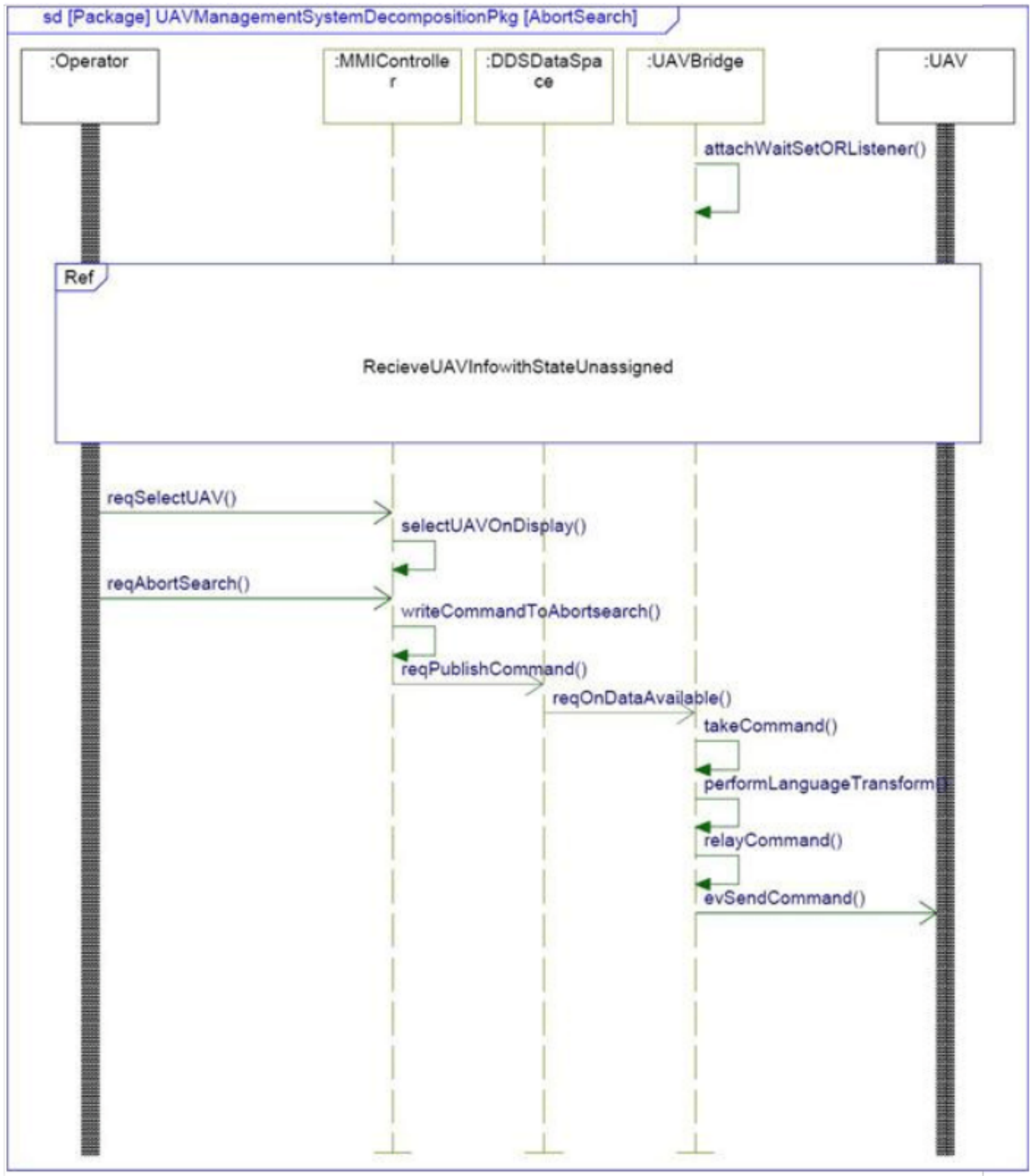


图 8 的大图

图 9. 白盒状态行为（面向数据）

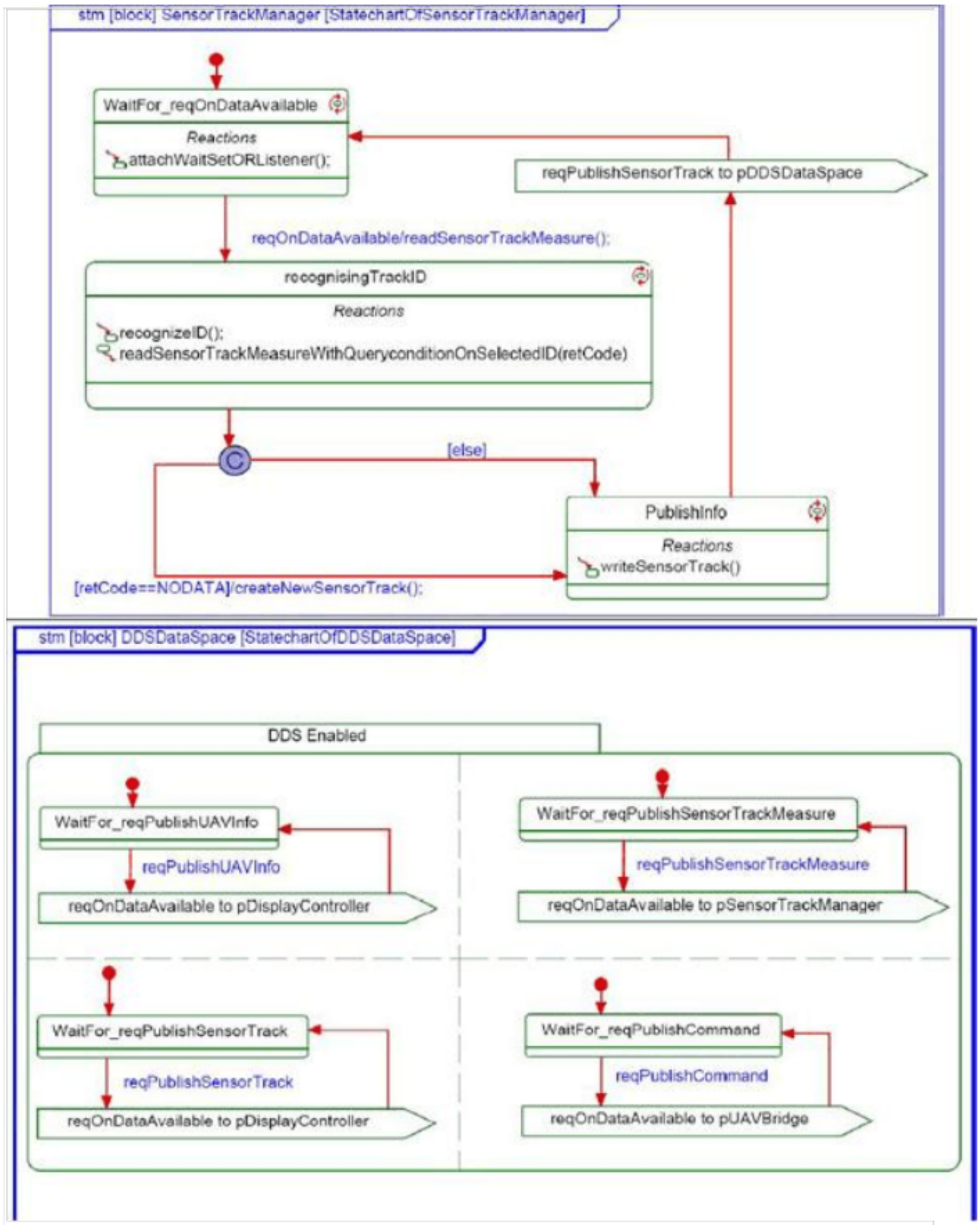


图 9 的大图

图 10. 白盒端口和接口（面向数据）

火龙果 • 整理
uml.org.cn

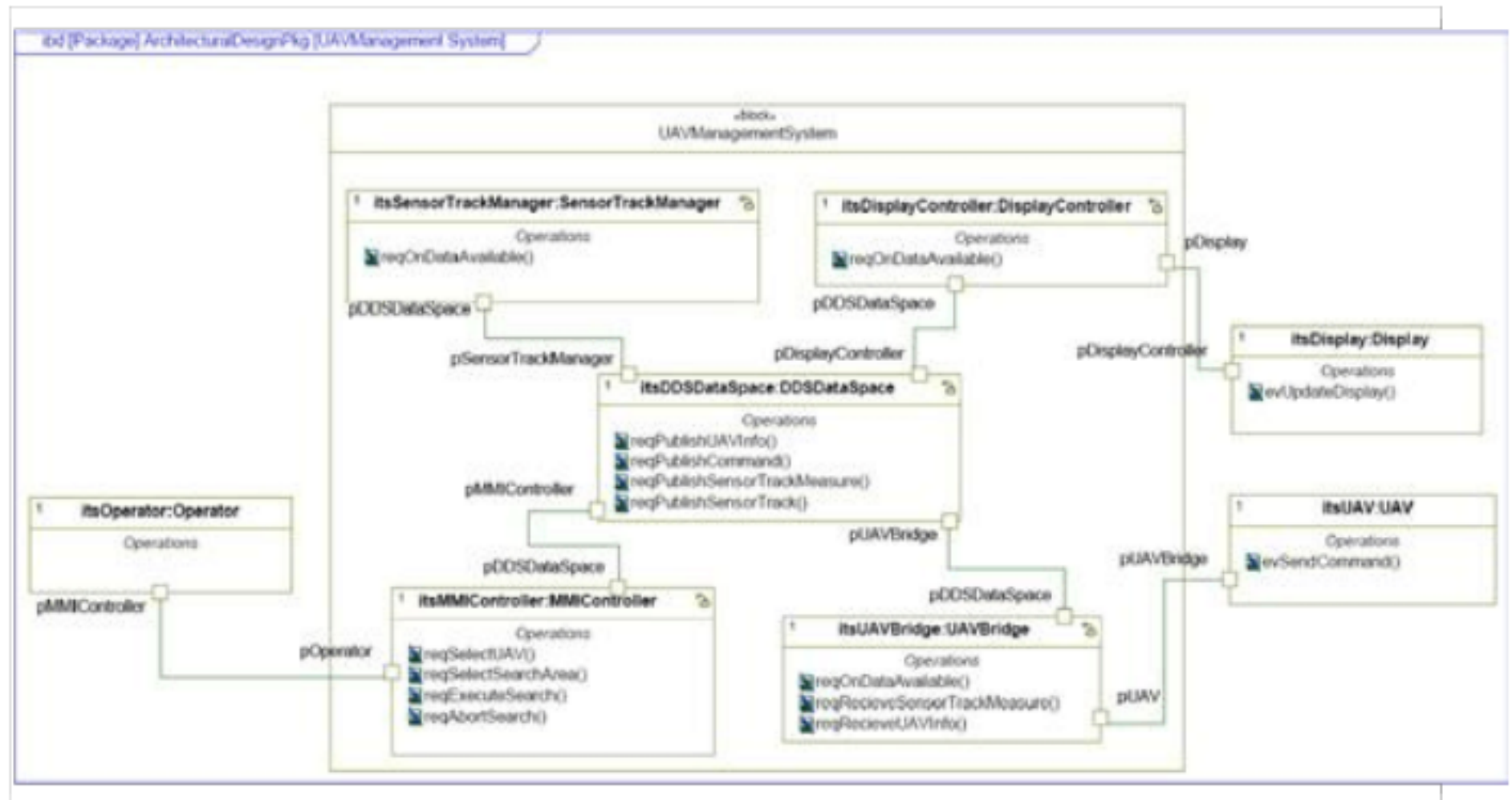


图 10 的大图

表 5. 白盒接口列表

块	端口名称	I/F 类型	接口事件	主题名称 / 描述
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
SensorTrack Manager	pDDSDataSpace	Provided	reqOnDataAvailable	SensorTrackMeasureTopic
		Required	reqPublishSensorTrack	SensorTrackTopic

块	端口名称	I/F 类型	接口事件	主题名称 / 描述
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
	pSensorTrack	Provid	reqPublishSensorTr	SensorTrackTopi

块	端口名称	I/F 类型	接口事件	主题名称 / 描述
	Manager	ed	ack	c
		Required	reqOnDataAvailable	SensorTrackMeasureTopic

回页首

结束语

在把基于分布式组件的系统架构放在一起时，主要需要考虑的事项是，能够明确界定业务功能及其跨子系统的接口。如果系统遵循下面这些面向服务的架构的 Open Architecture 原则（请参阅参考资料），那么我们完全可以解决上述这些问题：

- 模块化：这意味着，架构已仔细划分业务和技术功能，使用户能够单独访问它们，并在状态之间保持最少量的交互需求。在分布式系统的设计中使用发布和订阅模式，这从本质上促进了子系统设计的无状态性质的使用。从本文的案例研究中，显然可以得出以下结论：每个独立的活动流应当代表组件的一个 AND 状态，而每个 AND 状态中惟一的主导状态是每个流的等待状态。行为主要是无状态的，因此，已定义实体的创建或更新本身就是通过写操作暴露给系统的其余部分，并且从不保存。这样的设计自然地呈现模块化系统的属性，即在状态之间保持最少交互。
- 开放式标准：使用开放式标准对分布式系统的设计造成的最大影响最大是，这些标准必须在何处完成服务描述、发现和功能访问。SysML 是基于开放式标准的建模语言，所以也是 DDS 规范。SysML 是用于明确定义系统或子系统功能的语言。而 DDS 是用于明确定义子系统接口的语言，它也在其自身中封装了发现和访问机制。
- 互操作性：在分布式系统中，互操作性依赖于定义良好的接口语法和语义。作为接口语言的 DDS 不仅清楚地暴露了接口功能，还明确暴露了数据模型的相关语义和行为。这避免了因为对解析采用了错误的上下文而造成数据的误解。

因此，我们考虑在 MBSE 流程中将两个驱动因素（即 DDS 和 SysML ）结合在一起，这看起来似乎是成功的分布式系统分析和设计，该分析和设计严格采用了 Rational Harmony 系统工程等成熟流程定义的路径，而这些流程本身就以系统工程的最佳实践为基础。