

Tasks

Tasks are specific goals which we want to assign to a robot to complete. Tasks can require certain capabilities to be completed, such as a specific sensor. Tasks can also have information, such as the location where it should be performed. Here are some examples of some classes of tasks, or Task Classes:




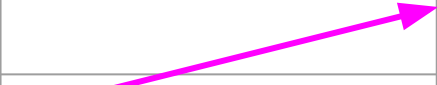
- A “temperature” task requires a temperature sensor capability and has a location value where the temperature related task should be taken
- A “camera” task requires a camera capability and has a location value describing where the camera image should be taken from

Similarly, each robot has a list of capabilities. Robots may have different capabilities depending on their sensor payload and other hardware.

Task allocation is the process of assigning a set of tasks to a set of robots. Ideally, all task capabilities are satisfied, and the tasks will be performed in an efficient manner. For efficient execution, task allocation may consider other information, such as:



- The closest robot to the task may already be assigned to a task
- Some tasks are more important than other tasks

Here is an example of allocating 3 camera tasks and 1 temperature task to a team of 3 robots. Each robot's capabilities are listed next to its name.

Camera task 1		Robot1 (temperature sensor and camera)
Camera task 2		Robot2 (camera)
Camera task 3		Robot3 (temperature sensor)
Temperature task 1		


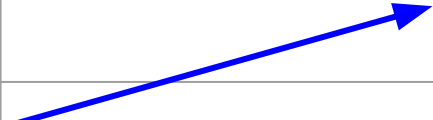
Here we have allocated a task to each robot, leaving one task unallocated as there are no more robots. As these robots finish their tasks, we can repeat this process to try and allocate Camera task 3.

Consider what will happen if Robot3 finishes its task, Temperature task 1. Robot 1 and 2 are still working on their previously assigned task. We will attempt to allocate Camera task 3.

Camera task 1		Robot1 (temperature sensor and camera)
Camera task 2		Robot2 (camera)
Camera task 3		Robot3 (temperature sensor)
Temperature task 4		

The only free robot is Robot 3, but it does not have a camera, so it cannot be allocated to Camera task 3.

Now Robot2 has finished its task, Camera task 2. Robot1 is working on its previously assigned task and Robot3 is doing nothing. We will attempt to allocate Camera task 3.

Camera task 1		Robot1 (temperature sensor and camera)
Camera task 2		Robot2 (camera)
Camera task 3		Robot3 (temperature sensor)
Temperature task 4		

Robot2 is available and has a camera, so it is assigned the task.

Similar to how we use tokens to represent specific robots, we will also use tokens to represent specific tasks, such as Camera task 3.

When a task token is used to activate an output event, such as →GroupGotoLocation, the SPN looks up which robot is allocated to the task and sends the command to that robot.

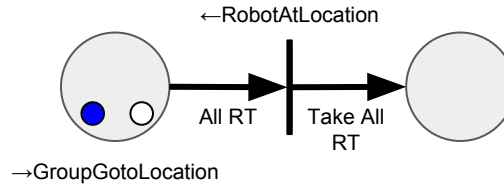
In the previous example where Camera task 3 is allocated to Robot2, when the Camera task 3 token is moved into the place with →GroupGotoLocation, Robot2 will receive that command.

We will also use the relevant token concept to move task tokens around in the same way we move robot tokens around.

When the robot finishes moving to the location, it will generate a \leftarrow RobotAtLocation input event as usual, but will list the task token as the relevant token instead of its robot token.

In the previous example, when Robot2 finishes moving to the location it was told to move to for Camera task 3, it will generate a \leftarrow RobotAtLocation input event with Camera task 3 as the relevant token.

QUIZ 9-1: What are the relevant tokens when Blue robot arrives at its location?



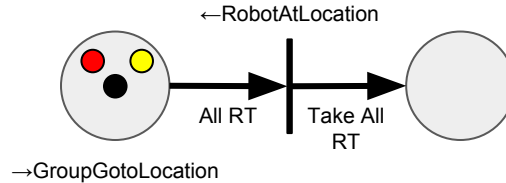
- Blue token is Blue robot's token
- White token is White robot's token

Quiz 9-1 Solution

The relevant tokens are:

- Blue robot's token

QUIZ 9-2: What are the relevant tokens when Yellow robot arrives at its location?



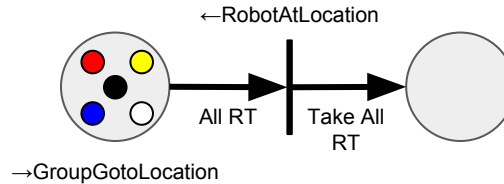
- Red token is a Temperature task 1's token, which is assigned to Red Robot
- Yellow token is a Camera task 1's token, which is assigned to Yellow Robot
- Black token is a generic token

Quiz 9-2 Solution

The relevant tokens are:

- Camera task 1's token

QUIZ 9-3: What are the relevant tokens when Red robot arrives at its location?



- Red token is a Temperature task 1's token, which is assigned to Red Robot
- Yellow token is a Camera task 1's token, which is assigned to Yellow Robot
- Blue token is Blue robot's token
- White token is White robot's token
- Black token is a generic token

Quiz 9-3 Solution

The relevant tokens are:

- Temperature task 1's token

Let's design a new plan using tasks to have a team of robotic boats take some both temperature measurements and panorama images at several locations in a pond. Here is some information about the team of boats:

- Some boats only have a temperature sensor
- Some boats only have a camera

This means the locations will sometimes need to be visited twice, once by a boat with a temperature sensors, and a second time with a boat with a camera.

In the end, we want the operator to

- Select the locations for temperature measurements and panoramas
- Select which computed task allocations to use

This is a lot of information to handle, so we have some new output events to help us out. Below is a list of the output events we will cover in this lesson:

- OperatorSpecifyLocations: Tells the operator to specify a list of locations on the GUI's map.
- GenerateTasks: Tells the system to create tasks of a certain type at specific locations in the world.
- TaskAllocationRequest: Tells the AI to generate task allocations for a list of Tasks
- OperatorChooseAllocation: Tells the operator to select from a list of task allocation choices or reject them all
- ApplyAllocation: Tells the system to apply a task allocation
- MeasureTemperature: Tells the robot to measure temperature at its current location, if possible
- TakePanorama: Tells the robot to take a panorama at its current location, if possible
- TaskComplete: Tells the system that a Robot has finished its assigned task

We also have some new input events we will cover this lesson:

- ←OperatorSpecifiedLocations: Returns the list of locations the operator choose in response to →OperatorSpecifyLocations
- ←TokensReceived: Returns a set of tokens representing tasks which were created in response to →GenerateTasks
- ←TaskAllocationResponse: Returns a number of task allocations in response to →TaskAllocationRequest
- ←TaskAllocationFailure: Indicates no task allocation was possible in response to →TaskAllocationRequest
- ←AllocationAccepted: Returns the task allocation the operator selected in response to OperatorChooseAllocation
- ←AllocationsRejected: Indicates all task allocation options were rejected by the operator in response to →OperatorChooseAllocation
- ←TaskAssigned: Indicates a specific task was assigned as a result of →ApplyAllocation
- ←TaskUnassigned: Indicates a specific task was not assigned as a result of →ApplyAllocation
- ←MeasurementDone: Indicates the robot is done taking a measurement as a result of →MeasureTemperature or →TakePanorama
- ←TaskCompleted: Indicates a task was marked as being complete via →TaskComplete

Here is a table showing the relationship between these new events

Output Event	Input Event(s) Generated
→OperatorSpecifyLocations	←OperatorSpecifiedLocations
→GenerateTasks	←TokensReceived
→TaskAllocationRequest	←TaskAllocationResponse or ←TaskAllocationFailure
→OperatorChooseAllocation	←AllocationAccepted or ←AllocationsRejected
→ApplyAllocation	←TaskAssigned and/or ←TaskUnassigned
→MeasureTemperature	←MeasurementDone
→TakePanorama	←MeasurementDone
→TaskComplete	

As a reminder each of these output and input events will have some number of fields. Each field has a type, name, and assignable value.

For output events, the values of their fields are used to make the request to the operator, robot, or AI.

For input events, the values of their fields are assigned by the operator, robot, or AI when the input event is generated.

Let's begin constructing this plan piece by piece:

- 1- Get the locations
- 2- Create the task tokens
- 3- Get task allocation options
- 4- Choose a task allocation
- 5- Apply the chosen task allocation
- 6- Perform and complete the assigned tasks
- 7- End the plan when all tasks have been completed

Let's construct this plan piece by piece:

1- Get the locations

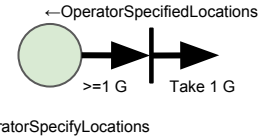
Output events to use:

- OperatorSpecifyLocations: Tells the operator to specify a list of locations on the GUI's map.
 - No fields

Input events to use:

- ←OperatorSpecifiedLocations: Returns the list of locations the operator choose in response to
 - OperatorSpecifyLocations
 - Field 1:
 - Name: List of locations.
 - Type: List of Locations
 - Value: These are the locations provided by the operator
 - No Relevant Tokens

Here we request and receive the list of locations from the operator.



Let's construct this plan piece by piece:

1- Get the locations

2- Create the task tokens

Output events to use:

→GenerateTasks: Tells the system to create tasks at specific locations in the world.

- Field 1
 - Name: Class of tasks to create?
 - Type: Task Class
 - Value: The type of task to generate at each location, such as Temperature or Camera
- Field 2
 - Name: Locations to generate tasks from?
 - Type: List of Locations
 - Value: A list of locations where tasks should be performed. We will use the list returned by
←OperatorSpecifiedLocations.

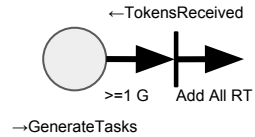
Input events to use:

←TokensReceived: Returns a set of task tokens representing the tasks created in response to

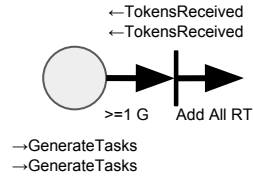
→GenerateTasks: one task token of the provided task class for each location in the provided list

- No fields
- Relevant Tokens: The Task tokens that were created

Below is the section of a SPN where task tokens are created for one task class (temperature or camera).



We add the output and input events again, this time for the other task class. Now if we activate the two \rightarrow GenerateTasks, using a list of 3 locations specified by the operator, 2 \leftarrow TokensReceived input events will be generated. The first will have 3 relevant tokens: Camera task 1, Camera task 2, and Camera task 3. The second will also have 3 relevant tokens: Temperature task 1, Temperature task 2, and Temperature task 3



Let's construct this plan piece by piece:

- 1- Get the locations
- 2- Create the task tokens
- 3- Get task allocation options

Output events to use:

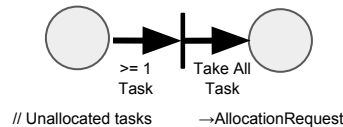
- TaskAllocationRequest: Tells the AI to generate task allocations for a list of Tasks
 - No fields

Input events to use:

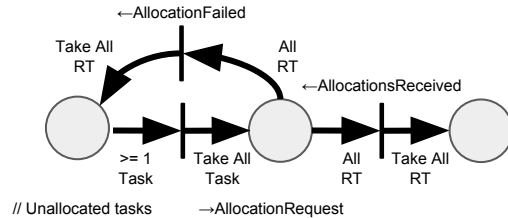
- ←TaskAllocationResponse: Returns a number of task allocations in response to →TaskAllocationRequest
 - Field 1
 - Name: Returned task allocation options.
 - Type: List of task allocations
 - Value: Each item in the list is a potential allocation
 - Relevant Tokens: The Task tokens for the tasks considered in the task allocation request
- ←TaskAllocationFailure: Indicates no task allocation was possible in response to →TaskAllocationRequest
 - No fields
 - Relevant Tokens: The Task tokens for the tasks considered in the task allocation request

Here we start with all unallocated task tokens in the “// Unallocated tasks” place.
The “//” just means that “Unallocated tasks” is simply a label on the place and not an output event.

If there are task tokens which are in the “// Unallocated tasks” they are moved to the place with \rightarrow AllocationRequest



The result of $\rightarrow \text{AllocationRequest}$ is either $\leftarrow \text{AllocationFailed}$ or $\leftarrow \text{AllocationsReceived}$. If the system fails to compute an allocation for a set of task tokens, we it to try again. If the system succeeds in computing some task allocations for some task tokens, we want to move to the next step.



Let's construct this plan piece by piece:

- 1- Get the locations
- 2- Create the task tokens
- 3- Get task allocation options
- 4- Choose an allocation

Output events to use:

→OperatorChooseAllocation: Tells the operator to select a task allocation choice or reject them all

- Field 1
 - Name: Resource allocation options to show to operator?
 - Type: List of Task Allocations
 - Value: The list of task allocation options the operator gets to choose from. We will use the list of task allocations computed by \leftarrow AllocationsReceived

Let's construct this plan piece by piece:

- 1- Get the locations
- 2- Create the task tokens
- 3- Get task allocation options
- 4- Choose an allocation

Input events to use:

←AllocationAccepted: Returns the task allocation the operator selected in response to

→OperatorChooseAllocation

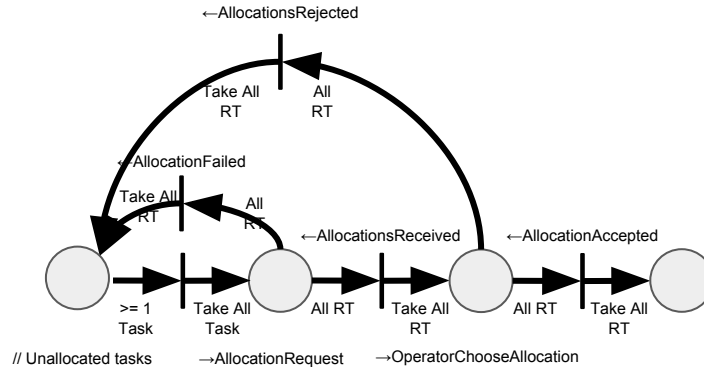
- Field 1
 - Name: Accepted resource allocation.
 - Type: Task Allocation
 - Value: The allocation selected by the operator
- Relevant Tokens: The Task tokens for the tasks considered in the task allocation

←AllocationsRejected: Indicates all task allocation options were rejected by the operator in response to

→OperatorChooseAllocation

- No fields
- Relevant Tokens: The Task tokens for the tasks considered in the rejected task allocations

If the operator rejects the task allocations, we want to return them to the “// Unallocated tasks” place, so we can attempt to allocate them again later. If the operator accepts a task allocation, we want to move forward in the SPN.



Let's construct this plan piece by piece:

- 1- Get the locations
- 2- Create the task tokens
- 3- Get task allocation options
- 4- Choose an allocation
- 5- Apply the chosen allocation

Output events to use:

→ApplyAllocation: Tells the system to apply a task allocation

- Field 1
 - Name: Resource allocation to apply?
 - Type: Task Allocation
 - Value: The task allocation which should be applied to the system. We will use the task allocation accepted by the operator.

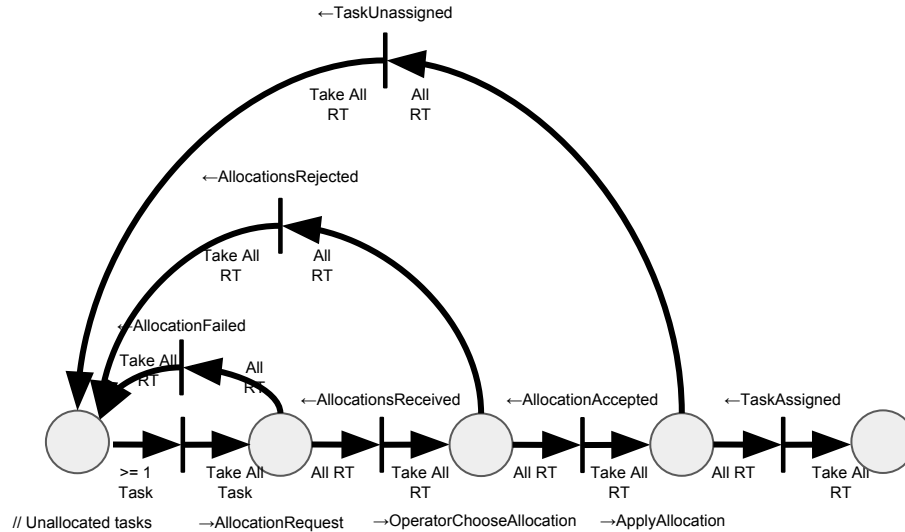
Input events to use:

←TaskAssigned: Indicates a task was assigned as a result of →ApplyAllocation

- No fields
- Relevant Tokens: The Task token for the task which was assigned

←TaskUnassigned: Indicates a task was not assigned as a result of →ApplyAllocation

- No fields
- Relevant Tokens: The Task token for the task which was unassigned



When we apply an allocation, some tasks will be allocated to a robot, while other may not, similar to our earlier example where we tried to allocate 4 tasks to 3 robots. For tasks that do not get allocated, we want to return those task tokens to the “`// Unallocated tasks`” place. For tasks that do get allocated, we want to move them forward in the SPN.

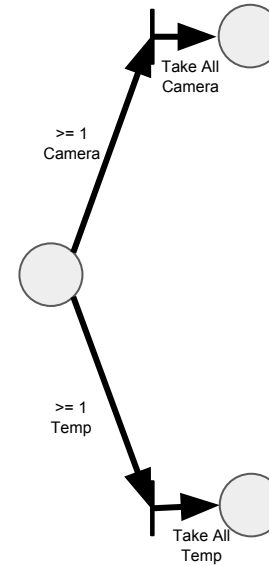
Let's construct this plan piece by piece:

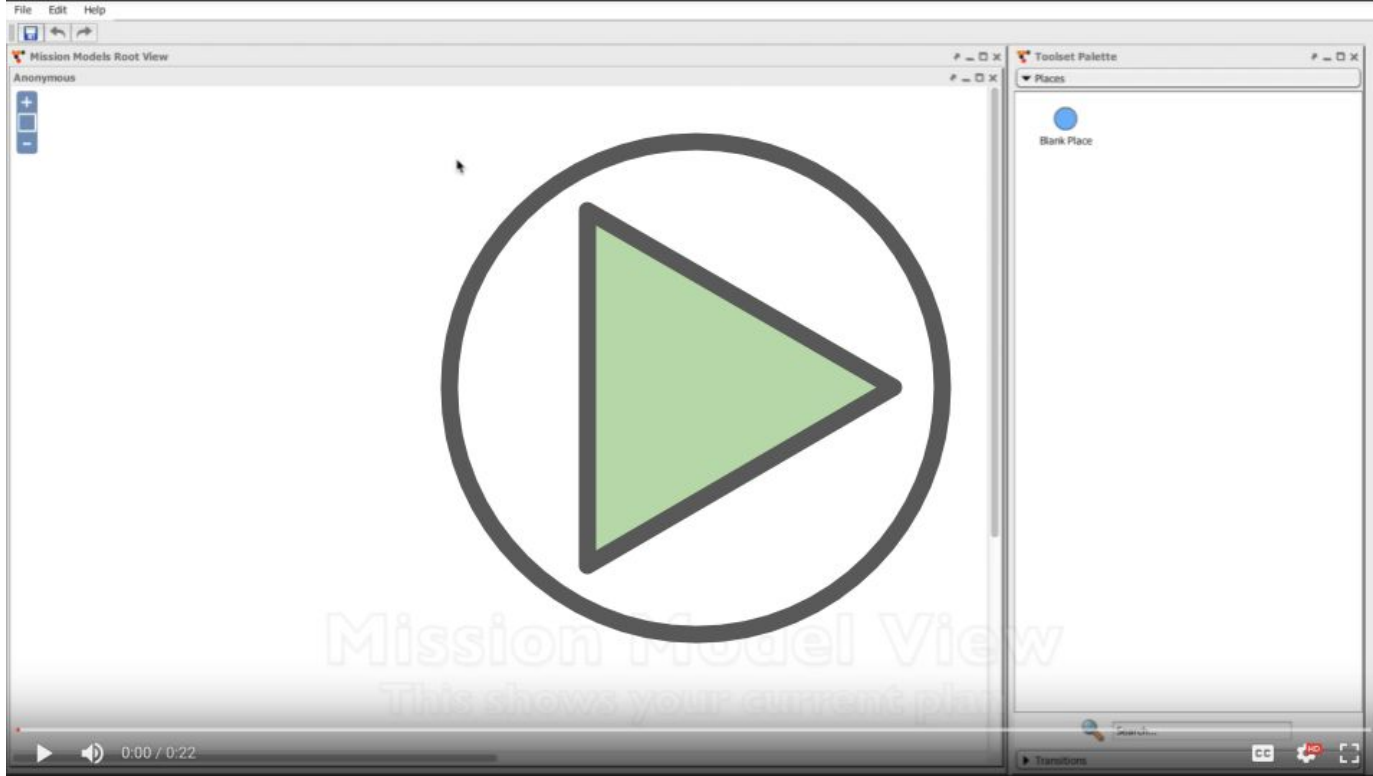
- 1- Get the locations
- 2- Create the task tokens
- 3- Get task allocation options
- 4- Choose an allocation
- 5- Apply the chosen allocation
- 6- Perform and complete the assigned tasks

The Camera and Temperature tasks will involve taking different actions, so we need to separate the Camera Task tokens from the Temperature Task tokens. We will do this using some new edge requirement capabilities:

- We can use a new in edge requirement to require a certain number of Task tokens of a certain Task Class (Camera, Temperature, etc) be present or absent
- We can use a new out edge requirement to add, take, or consume Task tokens of a certain Task Class

For the in and out edge requirements, we have shortened “Temperature” to “Temp”





Watch “Using Task Class Requirements”: This video will show you how to use in and out edge requirements which depend on Task tokens of a specific type

Now that we can split up the Task tokens by their type, we can have each task perform its specific responsibilities

For the camera task, we want the allocated robot to move to the location of the task, take the panorama, and then mark the task as complete.

For the temperature task, we want the allocated robot to move to the location of the task, measure the water temperature, and then mark the task as complete.

We already know how to move robots using \rightarrow RobotGotoLocation and \leftarrow RobotAtLocation.

Here are additional output events to use:

\rightarrow MeasureTemperature: Tells the robot to measure the water temperature at its current location

\rightarrow TakePanorama: Tells the robot to take a panorama at its current location

\rightarrow TaskComplete: Tells the system that a Robot has finished its assigned task

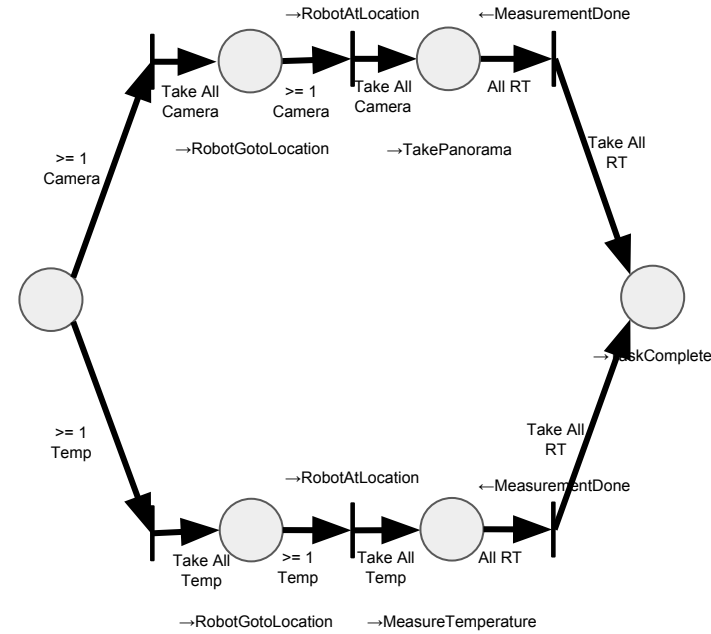
- No fields

Additional input events to use:

\leftarrow MeasurementDone: Indicates the robot is done taking a measurement as a result of

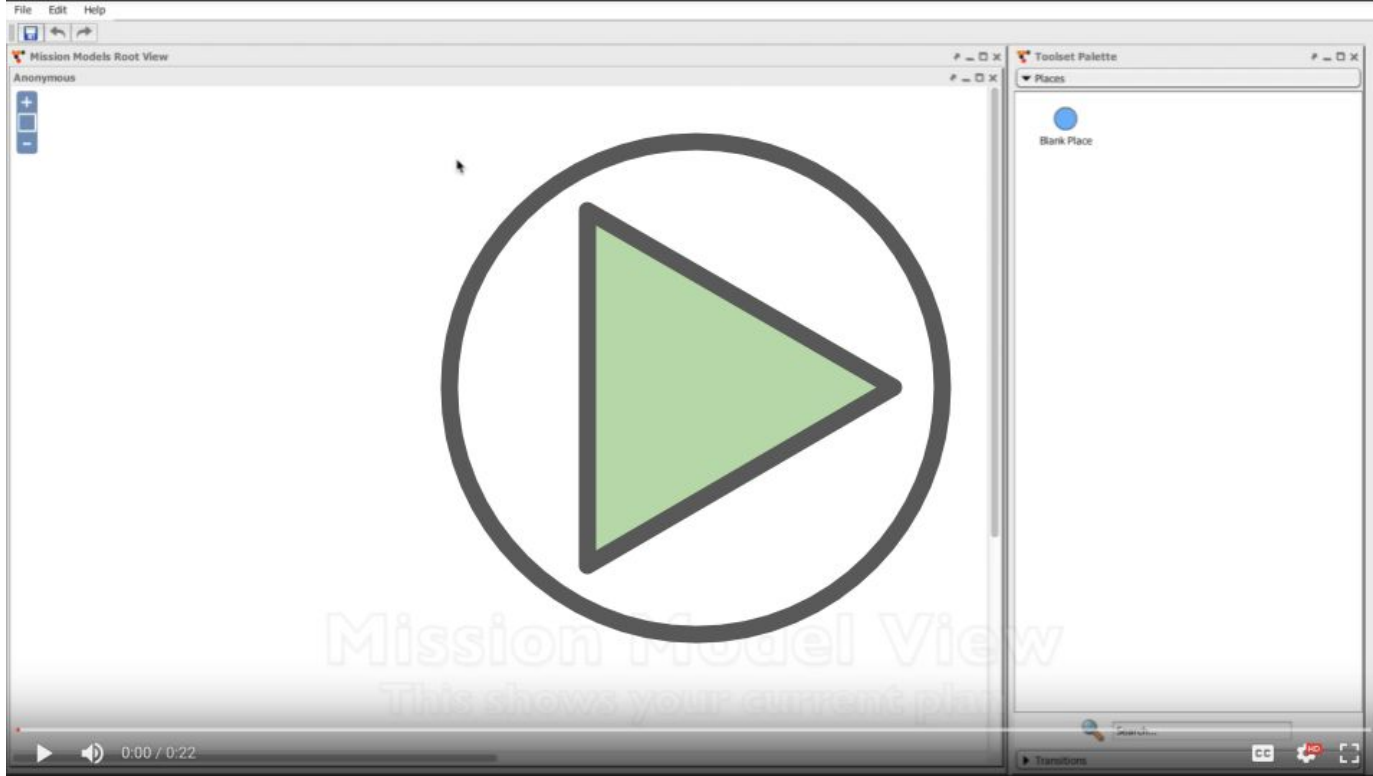
\rightarrow MeasureTemperature or \rightarrow TakePanorama

- No fields
- Relevant Tokens: If \rightarrow MeasureTemperature/ \rightarrow TakePanorama was activated with a Task token the RT is the task token; if it was activated by a Robot token the RT is the Robot token



When a Task token assigned to a robot activates the →RobotGotoLocation event, we have the problem of knowing where to send the robot. The task has an associated location, but the SPN needs to know to grab that location from the task.

To do this, for output events which are activated by a task token and have a Location field, we can list the task as the variable to read from for the Location field. This tells the system to use the Location associated with the task for the Location field.

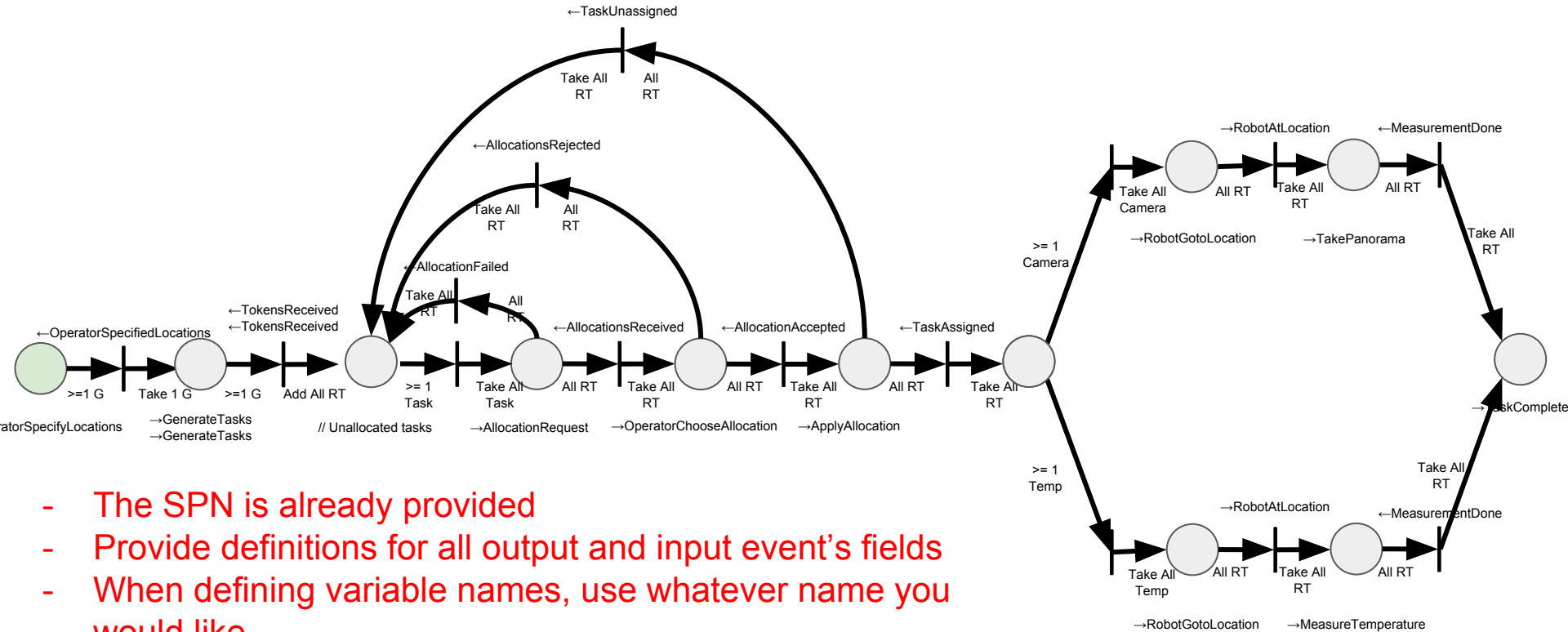


Watch “Using Task Definition for Field”: This video will show you how to have
→RobotGotoLocation’s Location destination field use the Location associated with
the Task token that activates it

Note that if a Robot token, not a Task token, activates a \rightarrow RobotGotoLocation with the Location field set to use “task” there will be no destination, so it will immediately assume it is at the destination.

Let's combine all the pieces

Job 9-1: Provide field definitions in DREAMM

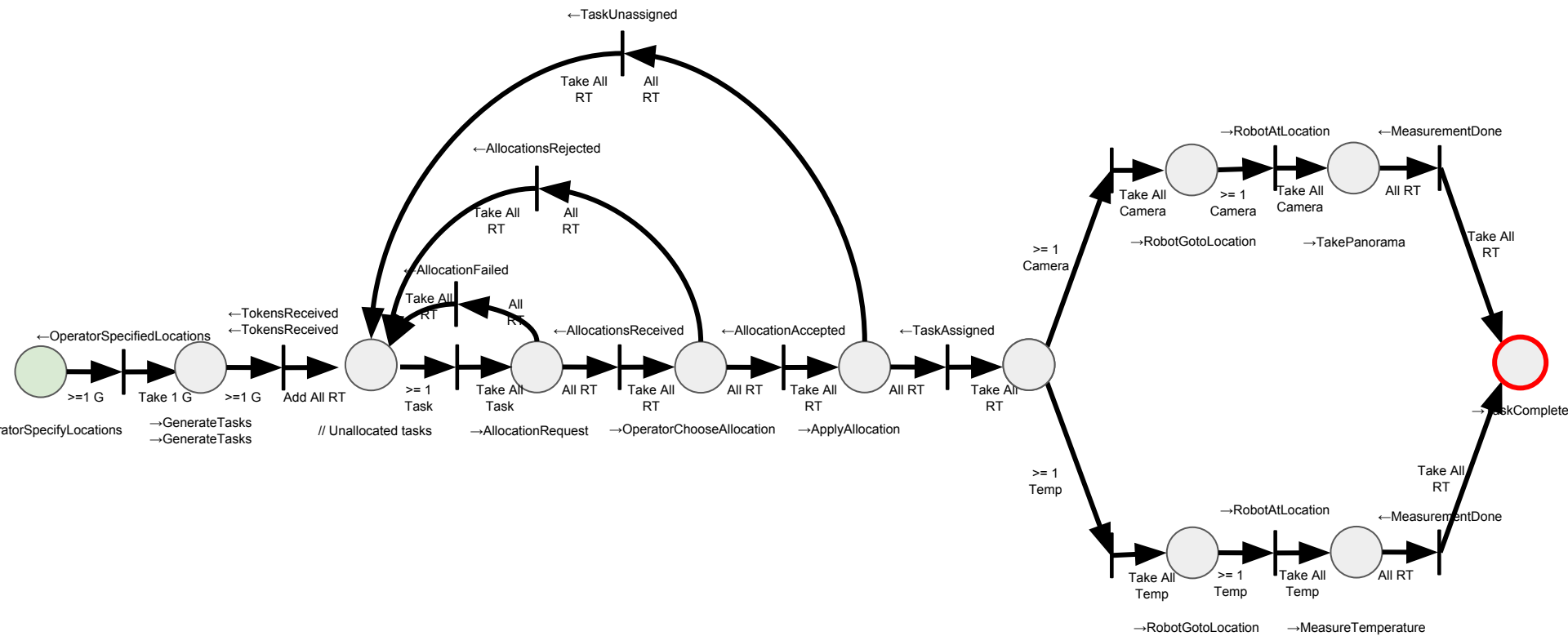


- The SPN is already provided
- Provide definitions for all output and input event's fields
- When defining variable names, use whatever name you would like

The final step in completing the SPN is setting an end place.

- 1- Get the locations
- 2- Create the task tokens
- 3- Get task allocation options
- 4- Choose an allocation
- 5- Apply the chosen allocation
- 6- Perform and complete the assigned tasks
- 7- End the plan when all tasks have been completed

QUIZ 9-4: What would happen if this was the end place?



Quiz 9-4 Solution

The plan would end as soon as the first task was completed

As a reminder, we tell the system when a single task is completed by activating a \rightarrow TaskComplete output event with that task token:

\rightarrow TaskComplete: Tells the system that a Robot has finished its assigned task

- No fields

However, we don't have a way of knowing when all tasks have been completed.

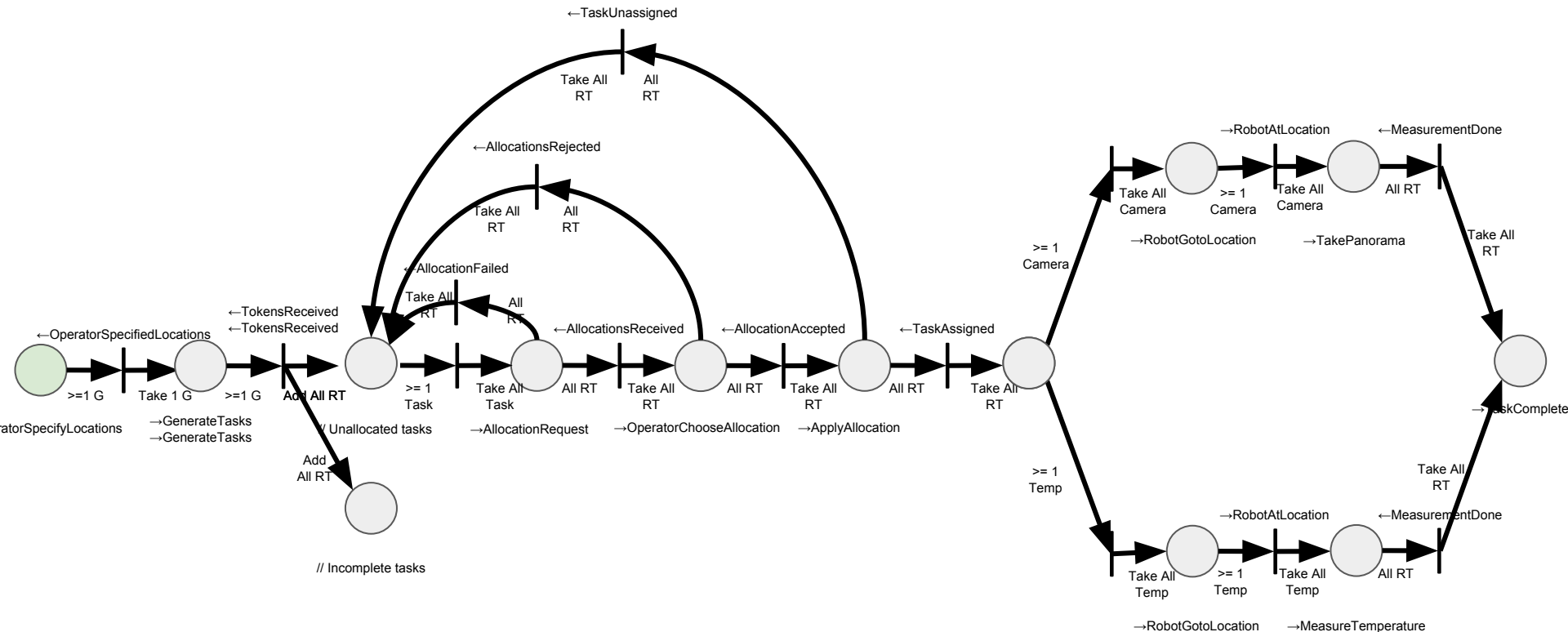
One way of doing this is to have a place that receives one copy of each task token as they are created. When that task has been completed, it consumes or takes that task token out of the place. When the place has no task tokens left, we know all tasks have been completed.

To do this, we will need a new input event:

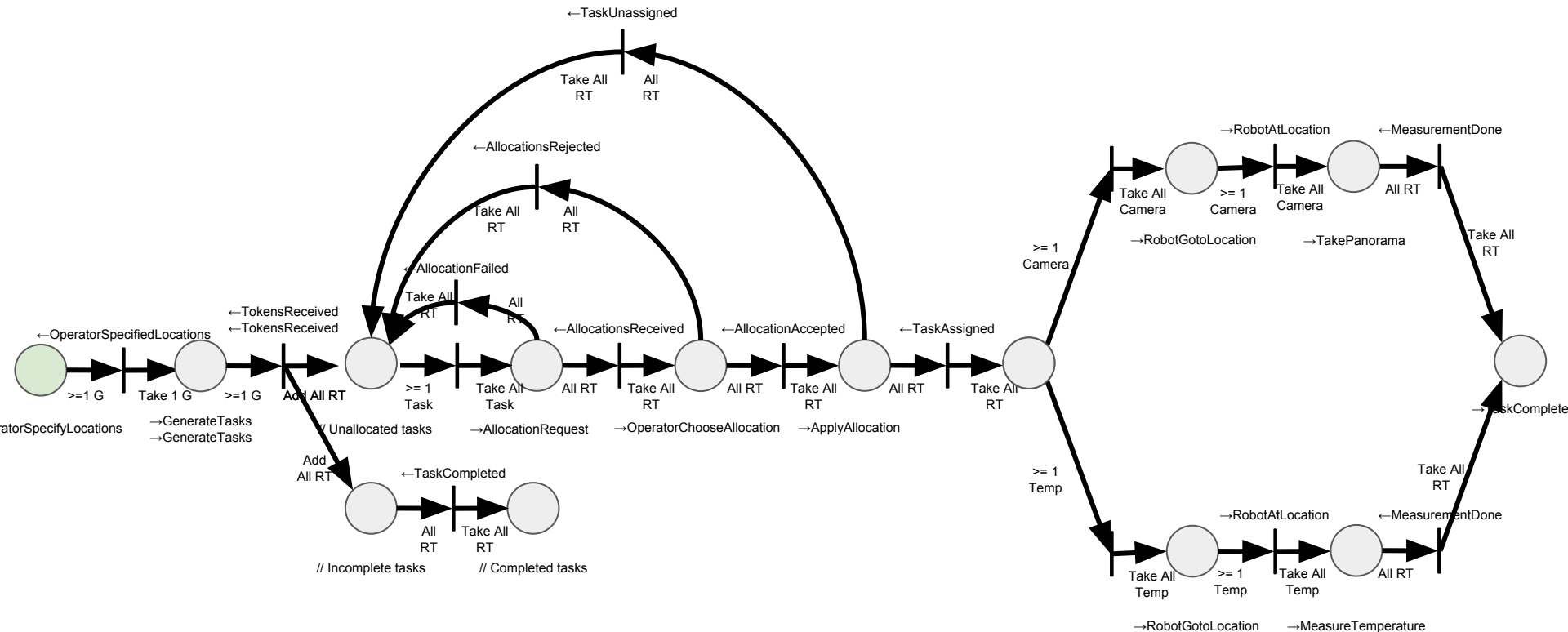
\leftarrow TaskCompleted: Indicates a task was marked as being complete via \rightarrow TaskComplete

- No fields
- Relevant Tokens: The token for the task which was completed

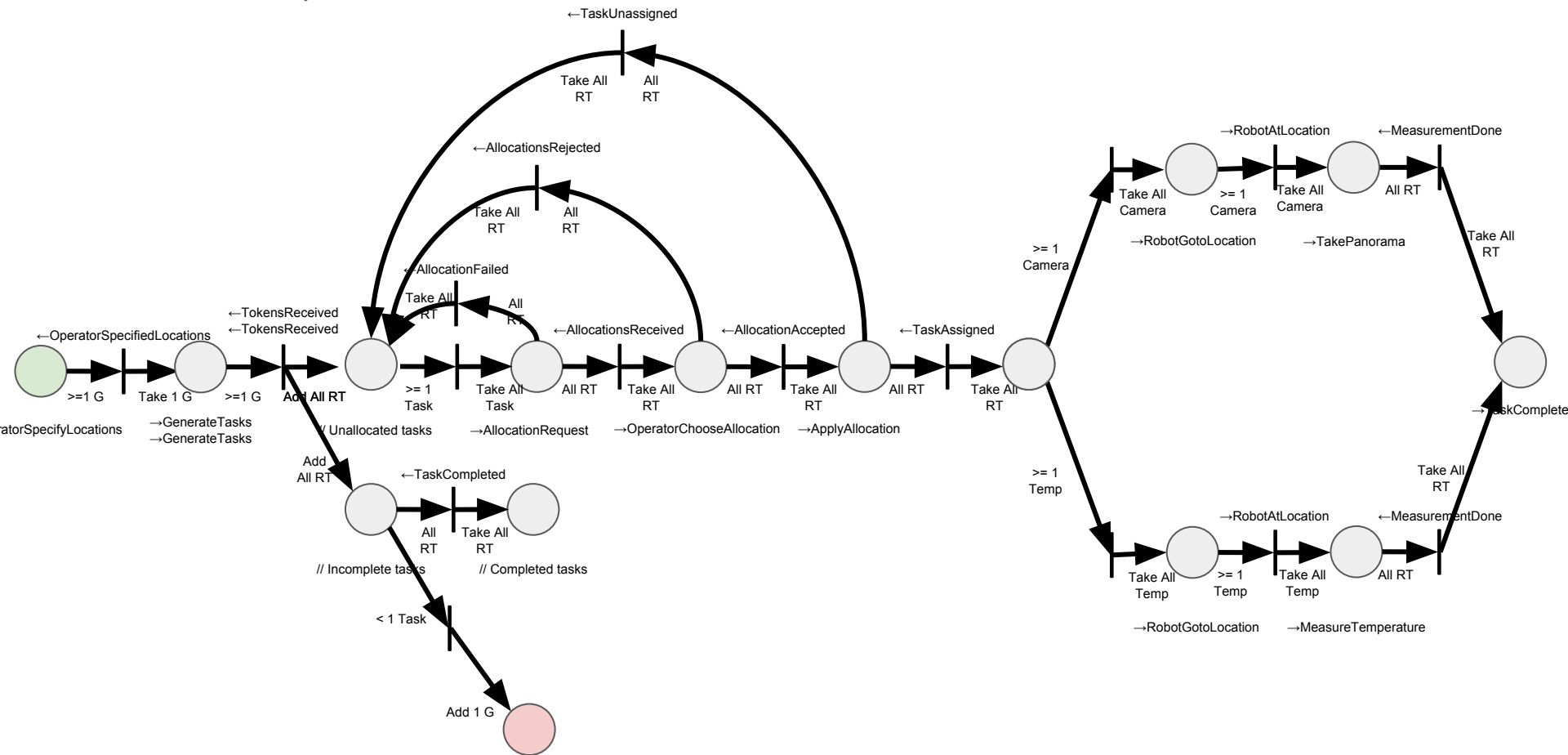
Here we make a new place labelled “// Incomplete tasks,” which also gets each of the
←TokensReceived input events’ relevant tokens added to it, just like the “// Unallocated tasks” place.



When we receive a `←TaskCompleted` input event, we move the relevant token (the completed task's task token) into the “// Completed tasks” place. Alternatively, we could consume the token.



When the “// Incomplete tasks” place has no more task tokens, we are done and can add a Generic token to an end place to end the SPN.



Job 9-2: Update the SPN in DREAMMM

