Reconfigurable Storing, Scanning and Moving Installation

SSAV Mini Project — Group 04

All six group members participated in the presentation and the report.

1st Frederik Alexander Hounsvad
University of Southern Denmark
Odense, Denmark
frhou18

2nd István Nagy
University of Southern Denmark
Odense, Denmark
isnag22

3rd Nikolai Emil Damm
University of Southern Denmark
Odense, Denmark
nidam16

4th Oliver Lind Nordestgaard
University of Southern Denmark
Odense, Denmark
olnor18

5th Peter Andreas Brændgaard
University of Southern Denmark
Odense, Denmark
pebra18

6th Troels Zink Kristensen
University of Southern Denmark
Odense, Denmark
tkris17

***Abstract*—This paper describes how the group used UPPAAL to model a reconfigurable storing, scanning and moving installation that simulates an assembly line.**
The paper highlights how UPPAAL has been used to design the system and document non-functional and functional requirements to ensure the installation functions reliably and safely according to specified requirements and safety standards on machine stop functions in (DS/EN 60204-1).

***Index Terms*—uppaal, model-checking, timed automata, iot, assembly-line**

I. Introduction

The group's reconfigurable storing, scanning, and moving installation that simulates an assembly line is a complex system that requires a lot of time and effort to design and build such that it is safe and reliable.

The system consists of one or more disks, cameras, and cranes. Both the number of devices, their properties, and how the assembly line should operate are configured with a domain-specific language (DSL). Due to this flexibility, the paper will focus on an arbitrary number of devices, that being one object of each device, and a predefined sequence of operations.

A collection of disks, cameras, and cranes will operate asynchronously in the assembly line, each part of a more extensive manufacturing process. However, each disk, camera, and crane set will work synchronously. For a group of devices, the sequence of operations is as follows.

1) An incoming item is placed on a disk
2) The disk rotates the item to a camera
3) The camera scans the item's colour and adds an artificial timer representing some processing operation that takes time.
4) The finished item is rotated to a crane.
5) The crane picks up the item.
6) The crane rotates to a storage container corresponding to the item's colour.
7) The crane drops the item.

The system's safety is critical as it is expected to be around personnel. The personnel might be working near or at the simulated assembly line. If the system is not safe, it could cause harm to the personnel by, e.g. the crane dropping items or swinging its jib.

II. Problem Definition

How can UPPAAL be used to analyze and document the group's simulated assembly line such that it is reliable and safe to operate?

III. Requirements

To accurately design and analyze the group's simulated assembly line, functional and non-functional requirements must be met. These requirements are described in detail in Appendix A.

In particular, the devices in the assembly line must be able to operate reliably and safely. These are critical non-functional requirements for the installation.

A set of functional requirements have been defined based on what necessary functionality is required to satisfy the non-functional requirements.

For the disk to operate reliably, it is necessary to rotate items from one location to another, utilizing the shortest path without tangling wires. Furthermore, the disk must be configurable with slots and positions to support various operations at different locations. A slot is a movable location on the disk where an item can be placed, and a position is a

predefined location where the disk can rotate a slot to support some operation on the item.

Much like the disk, the crane must also be able to rotate its jib utilizing the shortest path and avoiding tangling wires. Furthermore, it must be able to raise and lower its hoist to pick up and drop items.

For both the disk and the crane, the devices must respond to an emergency stop request by temporarily halting all operations if the system is unsafe. A safe stop function is also a safety standard described in *DS/EN 60204-1*[1] safety standard 9.2.2. Specifically, the system must have a stop function that conforms to either of the following stop categories:

- Stop category 0: stopping by immediately removing power to the machine actuators (i.e., an uncontrolled stop.)
- Stop category 1: a controlled stop with power available to the machine actuators to achieve the stop and then remove power when the stop is achieved.
- Stop category 2: a controlled stop with power remaining available to the machine actuators.[1]

In this case, the disk and crane will have a stop function that conforms to category 2, as the devices would pose greater risk if the actuators were powered off. It is vital that the crane continues receiving power, as the pickup mechanism uses an electromagnet. If power were to be cut, the crane would therefore drop any item it was carrying. Dropping items is not safe, as it could potentially damage equipment or personnel. Furthermore, all other devices have a stop function that conforms to category 0, as no threat to safety would be posed by a hard shutdown of these devices.

Adhering to all standards that apply to our installation would give the system a CE-label, that is, a certification that the system is satisfying the directives and is considered safe according to Danish law. This however is out of scope for our project. As such we have chosen to stick to the standards for a safe stop.

The camera is not a safety-critical device, and as such, it is only required to operate reliably. It means it must be able to scan the colors of the items. However, as the system is very dependent on the camera to scan the colors of items correctly, it is critical that the camera can retry scanning if it fails.

For all the devices, they must be able to communicate with a backend orchestrator to respond to commands promptly. Not responding to commands could cause the system to get out of sync, which could affect the reliability and safety of the system. The emergency stop function is implemented directly on each safety-critical device, such that it can stop in unsafe situations immediately. If the stop function is implemented on the orchestrator, any connectivity problems could cause the devices to be dangerous. The communication requirements must be accommodated by each device having WiFi and being able to subscribe to MQTT topics that contain messages that tell the devices what they must do.

IV. The UPPAAL Model

The UPPAAL Model created consists of 14 processes. In UPPAAL, a process is represented by a template, which is an extended timed automaton as described in *UPPAAL Templates* [2]. Below each template is described in detail. The templates are visible Appendix C, as well as in the attached UPPAAL project.

A. Templates

1) The Master Controller template:
The master controller template is responsible for the core logic. It has two sequences that it runs through;

- The first sequence checks if any items are finished. If so, the finished item is moved to the crane. The crane then moves the item to the container corresponding to the colour,
- The other sequence adds new items to the disk if it has any empty slots. The new items are then moved to the scanner position to be scanned.

As it communicates with all of the other templates, it has a lot of intermediate locations with syncs where the master controller emits, the other templates then transition state or return information, before the master controller continues to the next location.

It starts in the idle location, and has two edges from there. The first edge transitions to the `getempty` location, and emits to the `getemptySlot` channel. It then has two edges from there. One goes back to idle, but has a guard requiring the the `GlobalTimer` to be greater than 2. This ensures that if there are no empty slots, the master cointroller does not get stuck waiting for a sync that never comes. It simply returns to its previous state and is ready to do something else. The second has a sync for the `foundemptySlot` channel. The `getempty` location has an invariant, ensuring that `GlobalTimer` is less than 10. This verifies the time bounded-liveness property. The second location connected to the idle location is `getcompleteSlot`, which works similarly as the above-mentioned `getempty` location. These two edges are then the start of the two previously mentioned sequences. The master controller thereby chooses which sequence to execute based on if there are any empty or complete slots.

From there the master controller uses a pattern of transitioning to a location and emitting to a channel, where it then gets an instant response. Different templates react to the master controller's communication to execute the desired system flow.

2) The Disk template:
The disk template tracks the current position of the disk. It consists of a state for each position of the disk. The disk can change position from any position to any other position. All

these transitions contain syncs that allow the master controller to communicate which position it wants the disk to have. Each position state also has a transition to itself to allow the master controller to request the disk to move to the same position. Without such a transition, the model would deadlock due to the master controller not being able to use a transition with an emitter without a corresponding receiver.

The disk template can also add or remove an item from the slot at its current position. The feature allows the master controller to command an item to be added or removed without knowing in which slot because the disk keeps knowledge of its current position. These requests are all done through committed locations to ensure that any deadlocks are discovered, i.e. the template is not waiting for a sync that never comes.

Finally, the disk can go to the stopped location from any of its position, if an emergency stop where to happen.

3) The Disk Slot template:
The disk slot template consists of ten locations, with the initial location being when the disk slot is empty.

The locations cover the states of a slot:

- If it is full or empty.
- If it is getting a colored item
- If it has a colored item
- If it fails to remove or add an item.

The template relies on a parameter to be used for all slots. The parameter is a type that has been declared in the global scope. It is a simple integer array. The template is then copied for each index in the array. The same applies to the synchronizations that operate given an index for a specific slot. In practice, this makes sure that each disk slot has its own state at any given time.

4) The Disk Get Empty / Complete / Free / In-progress Slot template:
The disk-get-empty-slot template emits and receives signals when an empty slot is needed for the system. When the master controller emits a request to get an empty slot, the disk-get-empty-slot template has a receiving edge for each disk slot. Each if these edges also have a guard to a boolean array with the index corresponding to the disk slot. This boolean array keeps track of which slots are empty. Any edge where the array indicates that the slot is empty is therefore a viable receiver and a random one of those is chosen. The current disk slot is then updated with the slot number and the master controller is notified of this. The same happens for the disk-get-complete-slot, disk-get-in-progress-slot and disk-free-progress slot, but for the other three disk-slot states.

5) The Disk Slot Variable - Complete / Free / In-Progress template:
All three templates regarding the slot variable (complete, free, and in-progress) are equivalent; the only difference is what it is checking for, that being the variable is complete, free, or in progress. Using the complete template as an example, starting from the `disk1_IsNotcomplete` state, the transition to `disk_Iscomplete` happens when the MasterController signals to change the state. Transistioning between these two states updates the underlying boolean array with either true or false depending on the direction.

6) The Crane template:
The crane template has nine locations. The first four locations define the positioning of the crane arm where the initial location is at intake over the disk. The other three are over the three storage containers based on the item's color expected to be delivered. The template also has four locations that define the state of the hoist in regards to the positioning of the crane arm. For example, the location `LoweredAtRed` means that the crane has lowered the magnet at the red storage container. The last location in the template is the stop location, which is reached when the emergency stop is triggered.

The four locations defining the positioning of the crane arm have bidirectional edges to their respective `LoweredAt*` locations, such that the crane can lower and raise the magnet. Furthermore, this ensures that the crane cannot change its positioning when the magnet is lowered.

Additionally, all locations have an edge to the stop location, as it is safety-critical that the crane can stop no matter its state. The stop location has no outgoing edges and is a deadlock.

All the edges have synchronizations to react to the master controller and respond with the updated state of the crane.

7) The Crane Magnet template:
The crane magnet template has four locations - for the magnet's two power states under normal conditions or during an emergency stop. The initial location is when the magnet is not powered.

The template has three edges. One edge allows the magnet to toggle its power state under normal conditions. The other two ensure the magnet state is unaltered during an emergency stop by transitioning it to deadlocked states where the power state is unaltered. Synchronizations are used to allow the master controller to toggle the state of the magnet.

8) The Camera template:
The camera template has four locations — one for when it is idle and three for the different colors it can scan.

There are bidirectional edges from the idle location to all the color locations.

The edges rely on updates and syncs to inform the master controller of the scanned color and the camera's current state. The color is chosen at random.

9) The Emergency Button template:
The emergency button consists of two locations and a single edge that transitions from a running location to a stopping location. The edge has a sync to inform the other templates of the current state of the emergency button.

B. Requirements specification

The queries created in UPPAAL (see Appendix B) are defined based on the functional requirements from Appendix A. It is not really feasible to map queries to non-functional requirements. However, the result of satisfying the functional requirements would, in this case, indicate that the system is both reliable and safe.

Table I visualizes which of the developed queries correspond to which requirements. Each implementation of requirements is described in detail below the table.

Requirements A.1.b, A.1.c, A.1.d, A.1.e, A.1.i, A.1.j, A.2.e, A.2.f, A.2.j, A.2.k, A.3.c and A.3.d are not tested; the reason why is discussed in section V.

Requirements	Queries
A.1.a	1, 2
A.1.f	4, 5, 6, 31, 32, 33
A.1.g	5, 6, 31, 32, 33
A.1.h	7, 8, 31, 32, 33
A.2.a	9, 10, 11, 16
A.2.b	12, 13, 14, 15, 16
A.2.c	14, 15, 16, 17, 18
A.2.d	23, 24, 25, 26, 27, 28, 29, 30
A.2.g	4, 5, 6
A.2.h	5, 6
A.2.i	19, 20
A.3.a	21
A.3.b	22
A.1.b, A.1.c, A.1.d, A.1.e, A.1.i, A.1.j, A.2.e, A.2.f, A.2.j, A.2.k, A.3.c, A.3.d	

TABLE I
TABLE OF MAPPED REQUIREMENTS.

Query 1 and 2: It should be possible to go from `disk1.Position1` to `disk1.Position2`. However, disk1 should eventually go to Position4 from Position2. This is a result of the disk could possibly be in an error state when starting at Position1.

Query 4, 5, 6, 31 and 32: The system should possibly end up in `EmergencyButton.Stopped`. Queries 5, 6, 31 and 32 checks if activating the emergency button causes crane1, disk1, and the magnet (in either off or on the state) to end up in their stopped state. This should be true both eventually and invariantly. Furthermore, there should either be no deadlock or the emergency button should have been activated.

Query 7, 8, 19, 20, and 33: Disk1 should eventually go from its stopped state to Position1. Furthermore, the stopped state should lead to Position1. However, it is expected that these queries will fail, as a result of a lack of preventing deadlock when the emergency button has been activated. Query 19 and 20 also checks if the crane can resume from an emergency stop. It is expected to fail.

Query 9, 10, and 11: These queries check whether the crane is able to rotate its jib from one location to another, which is done by going from `crane1.crane1_intake` to `crane1.crane1_outRed`, and the same for other locations.

Query 12, 13, 14, 15 and 16: Queries 12 and 13 check that the crane is able to move to different positions and lower its hoist; for example from `crane1.crane1_intake` to `crane1.crane1_LoweredAtintake` to `crane1.crane1_intake` again. Queries 14, 15 and 16 is doing almost the same as 12 and 13, but with the magnet on at the same time.

Query 17 and 18: Check that the magnet can turn on and off.

Query 21 and 22: Check that the camera is able to scan item colors, and that it can retry scanning. Query 22 is expected to fail.

Query 23, 24, 25, 26, 27, 28, 29, 30: These queries test that the magnet keeps its magnet state (on or off) while the crane is moving.

Query 34 tests that all disk slots are eventually empty.

V. VERIFICATION RESULTS

How the different functional requirements are satisfied or are yet to be satisfied by UPPAAL-queries are documented in Table II. A list of the verification results of each query can also be seen in Appendix D.

The requirements A.1.b, A.1.c, A.1.d, A.1.e, A.1.i, A.1.j, A.2.e, A.2.f, A.2.j, A.2.k, A.3.c and A.3.d where deemed out-of-scope because:

- Testing whether the disk or crane can rotate the shortest path and avoid tangling wires requires extensive changes to the model (A.1.b, A.1.c, A.1.e, and A.1.f).
- Checking whether the disk has configurable slots or positions are configured in the DSL, which does not apply to the scope of this paper (A.1.d, A.1.d).
- Checking WiFi and MQTT connectivity can only be a simulated check and would provide limited value (A.1.i, A.1.j, A.2.j, A.2.k, A.3.c, A.3.d).

Requirements	Queries	Satisfied
A.1.a	1, 2	yes
A.1.f	4, 5, 6, 31, 32, 33	partially
A.1.g	5, 6, 31, 32, 33	partially
A.1.h	7, 8, 31, 32, 33	partially
A.2.a	9, 10, 11, 16	yes
A.2.b	12, 13, 14, 15, 16	yes
A.2.c	14, 15, 16, 17, 18	yes
A.2.d	23, 24, 25, 26, 27, 28, 29, 30	yes
A.2.g	4, 5, 6	yes
A.2.h	5, 6	yes
A.2.i	19, 20	yes
A.3.a	21	yes
A.3.b	22	yes
A.1.b, A.1.c, A.1.d, A.1.e, A.1.i, A.1.j, A.2.e, A.2.f, A.2.j, A.2.k, A.3.c, A.3.d		no

TABLE II
TABLE OF SATISFIED REQUIREMENTS.

VI. Discussion

Using UPPAAL as a model-checker to check whether the group's assembly line is a reliable and safe system has been a challenge.

Overall the verification results show that our system works as intended but that our safety requirements are only partially satisfied (see Table II). Both the disk, the crane, and the camera can operate reliably. The crane can rotate, pick up, drop, and carry around items without dropping them prematurely. The disk can rotate, occupy slots, and rotate slots to positions. The system has a working emergency stop function; however, not all related queries satisfy our expectations. The system should end up in a deadlock when the emergency stop is triggered, and the queries show that this is not invariantly valid for the system. The group analyzed why and found that we have no deadlock state in the master controller, so when the emergency stop is triggered and all devices stop, the master controller will continue not knowing so. It is not critical for safety as the devices will not respond to MQTT topics when stopped. Fixing this would require adding a stopped state to the master controller triggered when the emergency stop is activated, but that would require all locations to have an edge to the stopped state, making the master controller template less readable. The master controller is already a big template, so this was decided against.

In general, the size of the UPPAAL-project made it extremely difficult to keep track of the flow between processes and maintain a consistent and readable layout. Lack thereof made it cumbersome to work with, and it increased the time it took to fix and finalise the model. The complexity is primarily due to the UPPAAL-project being entirely generated by the group's DSL-generator. Neither the generator nor UPPAAL supports automating a proper layout, and the generator is not concerned with the complexity of processes. Furthermore, as the generator is a work in progress, it is not capable of generating all concepts of UPPAAL, which might have introduced errors or bad practices.

As a result, the UPPAAL-model is not directly translatable to the existing system. The model has not been able to check that all requirements are satisfied. Even though the UPPAAL-model does an excellent job documenting the internal processes in the system, it fails to properly model the difficulties of wireless communication or the physical layout of the devices.

VII. Conclusion

In conclusion, it is safe to say that using UPPAAL as a model-checker can be an extensive, but rewarding task. The tool is handy to check that requirements are satisfied or not if a correct representation of the actual system can be modelled. As this paper uses UPPAAL as a model-checker on a code-generated UPPAAL project, the resulting models have at times been difficult to understand and debug. However when this was overcome, the combination of code generation and UPPAAL model checking provides for easy verification of any system implemented with the DSL.

The verification results from the generated UPPAAL-project indicate that the reconfigurable storing, scanning, and moving installation representing an assembly line mostly conforms to non-functional requirements of being reliable and safe.

In summary, the devices in the system operate according to requirements, but the safety features are lacking. According to requirements, the system should end up in a deadlocked state when an emergency function is triggered. Still, the verification results show that this is not invariantly true for the system.

Overall this paper demonstrates that UPPAAL can be a crucial tool for verifying systems. It is, however, a time intensive task that has to fit with the requirements and scope of the desired project. When working with UPPAAL, it is important to start simple with many abstractions rather than trying to model everything right from the beginning. Additional implementations can then be added iteratively. The paper, however, also proves that generating UPPAAL-models is very doable, and with enough time, it could provide even better results than manually creating UPPAAL-projects.

References

[1] DS, *DS/EN 60204-1*, 2018.

[2] UPPAAL, *Uppaal templates*, https://docs.uppaal.org/language-reference/system-description/templates/, May 2022.

Acronyms

DSL domain-specific language. 1, 4, 5

MQTT Message Queuing Telemetry Transport. 2

APPENDIX A
FUNCTIONAL AND NON-FUNCTIONAL REQUIREMENTS

A. Functional requirements

1) Functional requirements for the disk.
   a) The disk must be able to rotate items from one location to another.
   b) The disk must be able to rotate the shortest path.
   c) The disk must be able to avoid tangling wires while rotating.
   d) The disk must have configurable slots for items.
   e) The disk must have configurable zones for operations, e.g., scanning, crane pick up, intake.
   f) The disk must have a stop function that conforms to danish machine safety standards.
   g) The disk's stop function must halt all movement if the system is in an unsafe state.
   h) The disk's stop function must be able to resume operations when the unsafe condition is resolved.
   i) The disk must have WiFi connectivity.
   j) The disk must be controllable with MQTT.
2) Functional requirements for the crane.
   a) The crane must be able to rotate its jib from one location to another.
   b) The crane must be able to raise and lower with its hoist.
   c) The crane must be able to pick up and deliver items.
   d) The crane is not allowed to drop items before it reaches its target location.
   e) The crane must be able to rotate the shortest path.
   f) The crane must be able to avoid tangling wires while rotating.
   g) The crane must have a stop function that conforms to danish machine safety standards.
   h) The crane's stop function must halt all movement if the system is in an unsafe state.
   i) The crane's stop function must be able to resume operations when the unsafe condition is resolved.
   j) The crane must have WiFi connectivity.
   k) The crane must be controllable with MQTT.
3) Functional requirements for the camera.
   a) The camera must be able scan colors of items.
   b) The camera must be able to retry scanning if it fails.
   c) The camera must have WiFi connectivity.
   d) The camera must be controllable with MQTT.

B. Non-functional requirements

1) Non-functional requirements for the disk.
   a) The disk must operate reliably.
   b) The disk must operate safely and not cause harm to people.
2) Non-functional requirements for the crane.
   a) The crane must operate reliably.
   b) The crane must operate safely and not cause harm to people.
3) Non-functional requirements for the camera.
   a) The camera must operate reliably.

APPENDIX B
QUERIES

1) E<> disk1.Position1 imply disk1.Position2
2) A<> disk1.Position2 imply disk1.Position4
3) A<> disk1.AddItem1Req imply (disk1_DiscSlot(1).disk1_SlotFull and forall(i : int[2,8]) disk1_DiscSlot(i).disk1_SlotEmpty)
4) E<> EmergencyButton.Stopped
5) A<> EmergencyButton.Stopped imply (crane1.crane1_Stopped and disk1.Stopped and (crane1_CraneMagnet.crane1_StoppedMagnetOff or crane1_CraneMagnet.crane1_StoppedMagnetOn))
6) A[] EmergencyButton.Stopped imply (crane1.crane1_Stopped and disk1.Stopped and (crane1_CraneMagnet.crane1_StoppedMagnetOff or crane1_CraneMagnet.crane1_StoppedMagnetOn))
7) A<> disk1.Stopped imply disk1.Position1
8) disk1.Stopped $\rightarrow$ disk1.Position1
9) E<> crane1.crane1_intake imply crane1.crane1_outRed
10) E<> crane1.crane1_outRed imply crane1.crane1_outBlue
11) E<> crane1.crane1_outBlue imply crane1.crane1_intake
12) E<> crane1.crane1_intake imply crane1.crane1_LoweredAtintake imply crane1.crane1_intake
13) E<> crane1.crane1_outGreen imply crane1.crane1_LoweredAtoutGreen imply crane1.crane1_outGreen
14) E<> (crane1.crane1_LoweredAtintake and crane1_CraneMagnet.crane1_MagnetOn) imply (crane1.crane1_intake and crane1_CraneMagnet.crane1_MagnetOn)
15) E<> (crane1.crane1_outRed and crane1_CraneMagnet.crane1_MagnetOn) imply (crane1.crane1_LoweredAtoutRed and crane1_CraneMagnet.crane1_MagnetOn)

16) E<> (crane1.crane1_intake and crane1_CraneMagnet.crane1_MagnetOn) imply (crane1.crane1_outRed and crane1_CraneMagnet.crane1_MagnetOn)

17) A<> crane1_CraneMagnet.crane1_MagnetOn imply crane1_CraneMagnet.crane1_MagnetOff

18) E<> (crane1.crane1_LoweredAtoutRed and crane1_CraneMagnet.crane1_MagnetOn) imply (crane1.crane1_LoweredAtoutRed and crane1_CraneMagnet.crane1_MagnetOff)

19) A<> crane1.crane1_Stopped imply crane1.crane1_intake

20) crane1.crane1_Stopped $\rightarrow$ crane1.crane1intake

21) E<> (camera1.camera1_Green or camera1.camera1_Blue or camera1.camera1_Red) and (colour $<=$ 3 or colour $>$ 0)

22) MasterController.camera1_scanItem_14 $\rightarrow$ MasterController.camera1_scanItem_14

23) A[] (crane1.crane1_intake and crane1_CraneMagnet.crane1_MagnetOff) imply !(crane1.crane1_intake and crane1_CraneMagnet.crane1_MagnetOn)

24) A[] (crane1.crane1_intake and crane1_CraneMagnet.crane1_MagnetOn) imply !(crane1.crane1_intake and crane1_CraneMagnet.crane1_MagnetOff)

25) A[] (crane1.crane1_outRed and crane1_CraneMagnet.crane1_MagnetOff) imply !(crane1.crane1_outRed and crane1_CraneMagnet.crane1_MagnetOn)

26) A[] (crane1.crane1_outRed and crane1_CraneMagnet.crane1_MagnetOn) imply !(crane1.crane1_outRed and crane1_CraneMagnet.crane1_MagnetOff)

27) A[] (crane1.crane1_outGreen and crane1_CraneMagnet.crane1_MagnetOff) imply !(crane1.crane1_outGreen and crane1_CraneMagnet.crane1_MagnetOn)

28) A[] (crane1.crane1_outGreen and crane1_CraneMagnet.crane1_MagnetOn) imply !(crane1.crane1_outGreen and crane1_CraneMagnet.crane1_MagnetOff)

29) A[] (crane1.crane1_outBlue and crane1_CraneMagnet.crane1_MagnetOff) imply !(crane1.crane1_outBlue and crane1_CraneMagnet.crane1_MagnetOn)

30) A[] (crane1.crane1_outBlue and crane1_CraneMagnet.crane1_MagnetOn) imply !(crane1.crane1_outBlue and crane1_CraneMagnet.crane1_MagnetOff)

31) A[] (EmergencyButton.Stopped and crane1_CraneMagnet.crane1_MagnetOn) imply crane1_CraneMagnet.crane1_StoppedMagnetOn

32) A[] !deadlock or EmergencyButton.Stopped

33) A[] disk1.Stopped imply disk1.Position1

34) E<> forall (i : disk1_id_t) disk1_DiscSlot(i).disk1_SlotEmpty

APPENDIX C
TEMPLATES

A. The Master Controller template

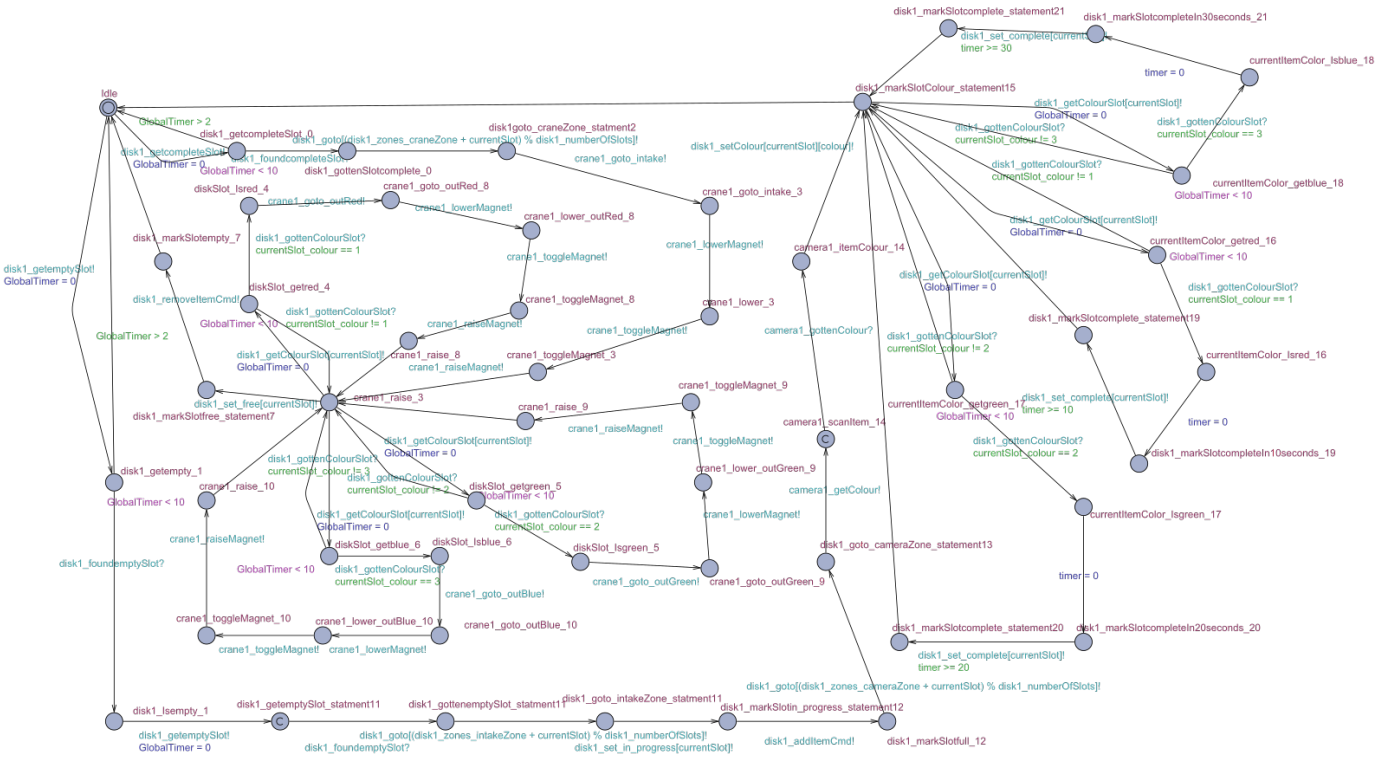

Fig. 1. Master Controller.

B. The Disk template

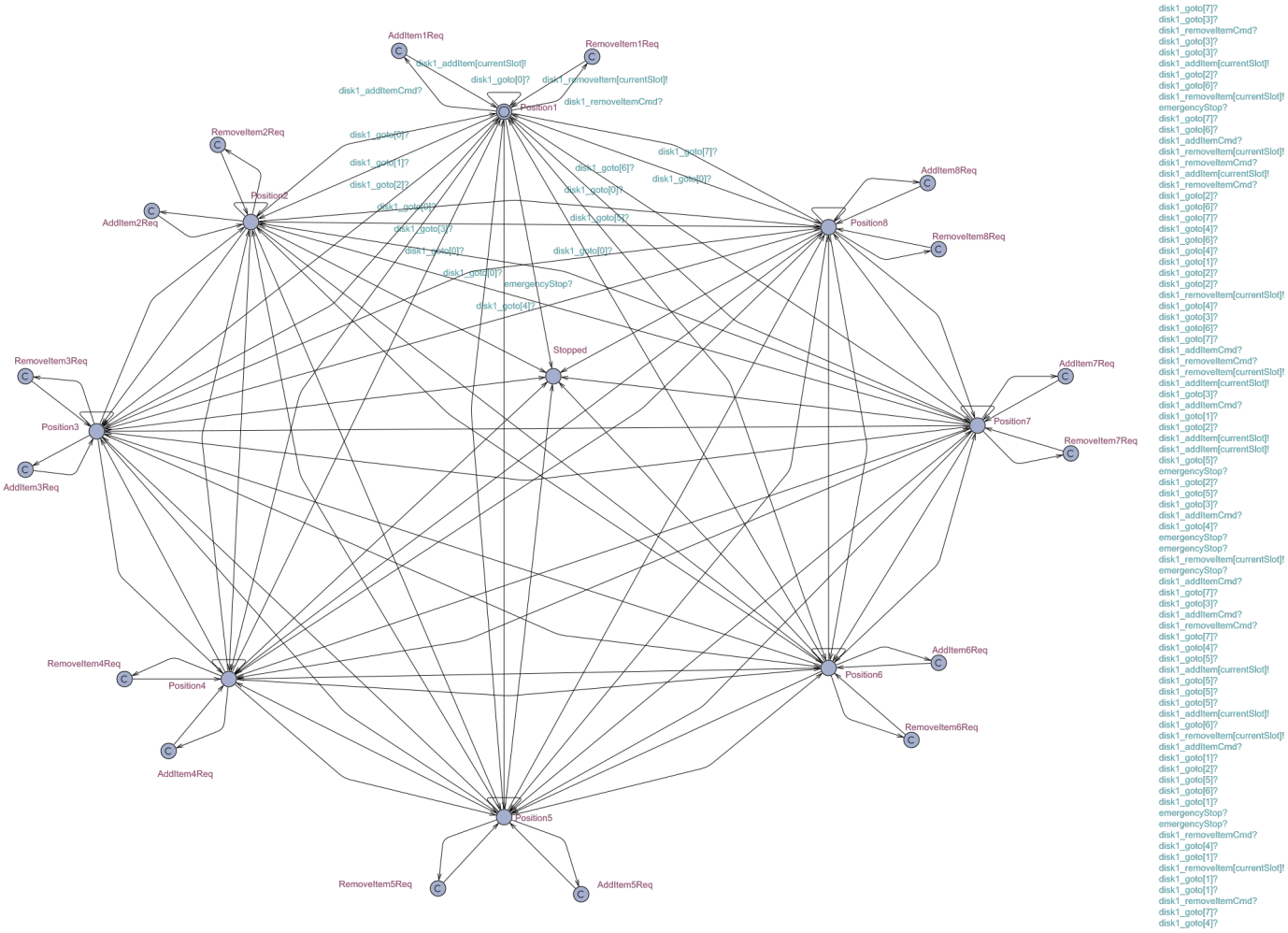

Fig. 2. Disk.

C. The Disk Slot template

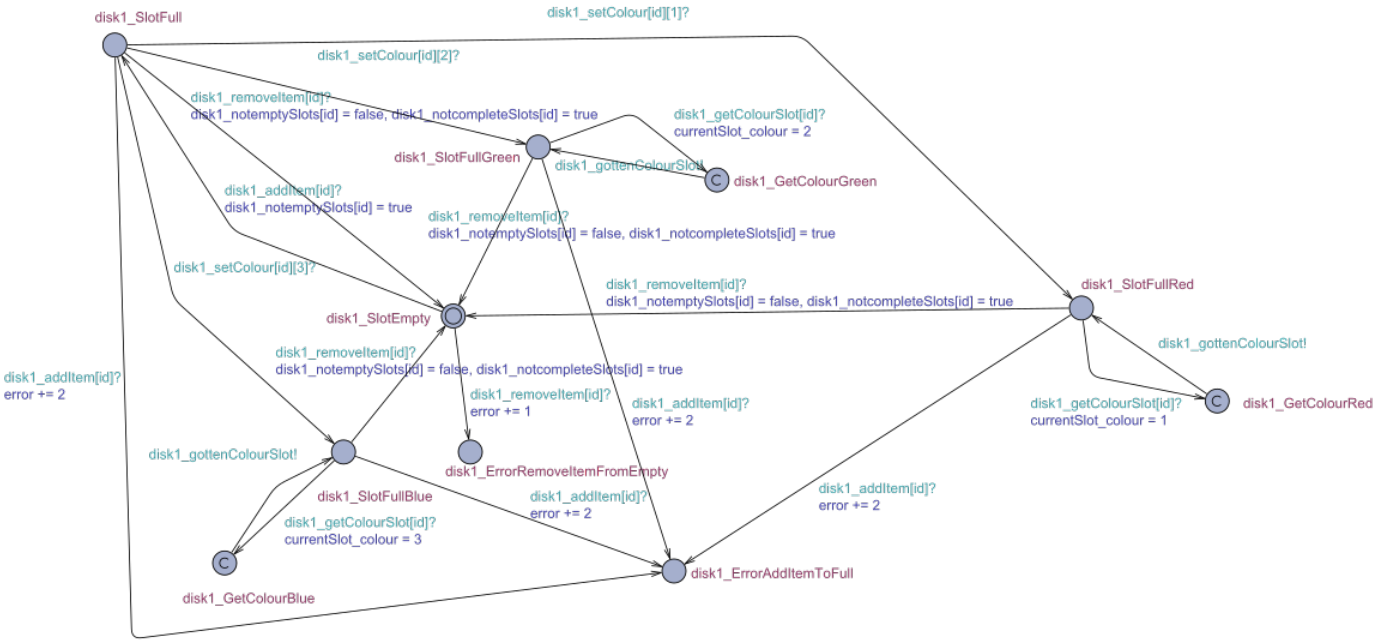

Fig. 3. Disk Slot.

D. The Get Empty Slot template

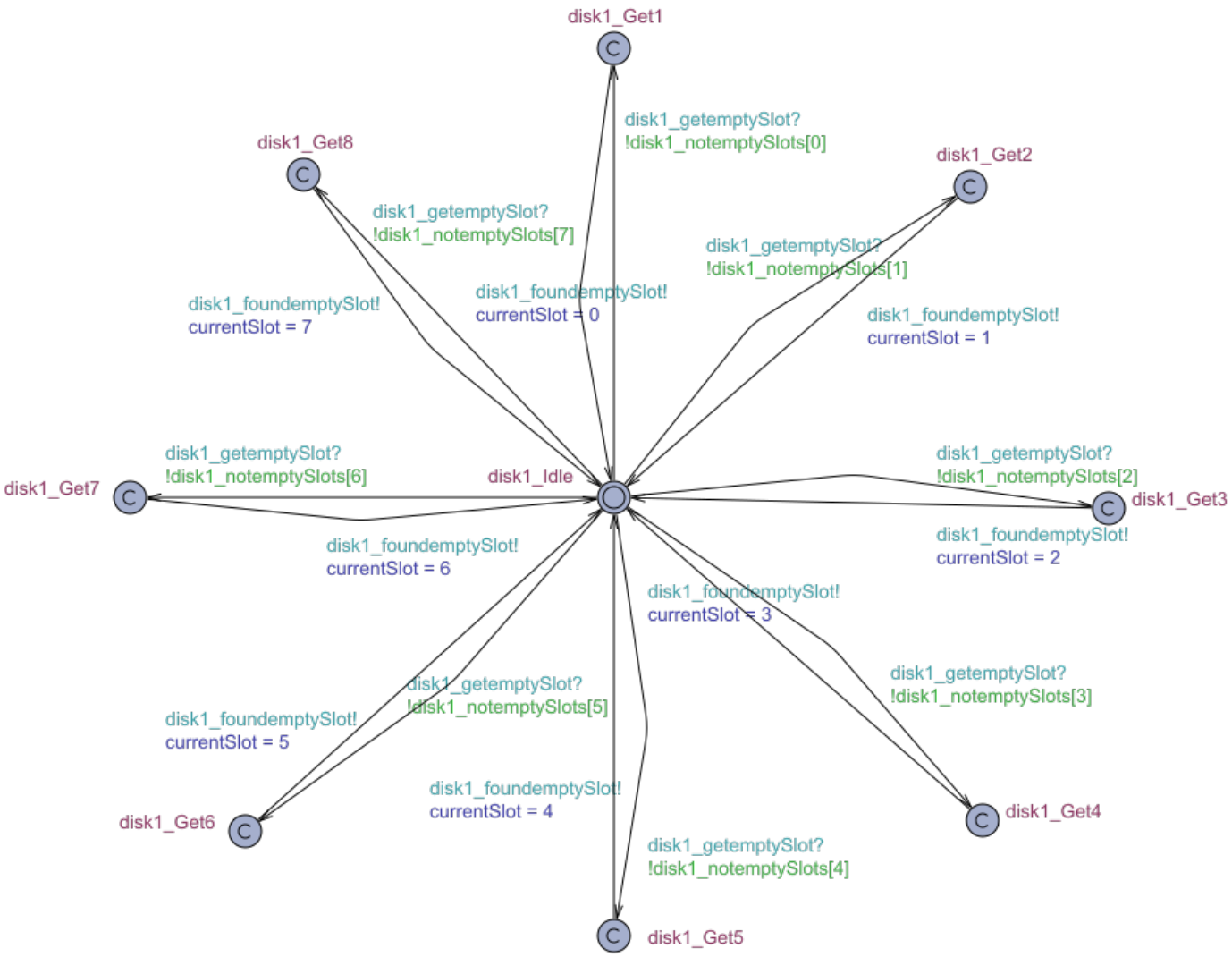

Fig. 4. Disk - Get Empty Slot.

E. The Get Complete Slot template

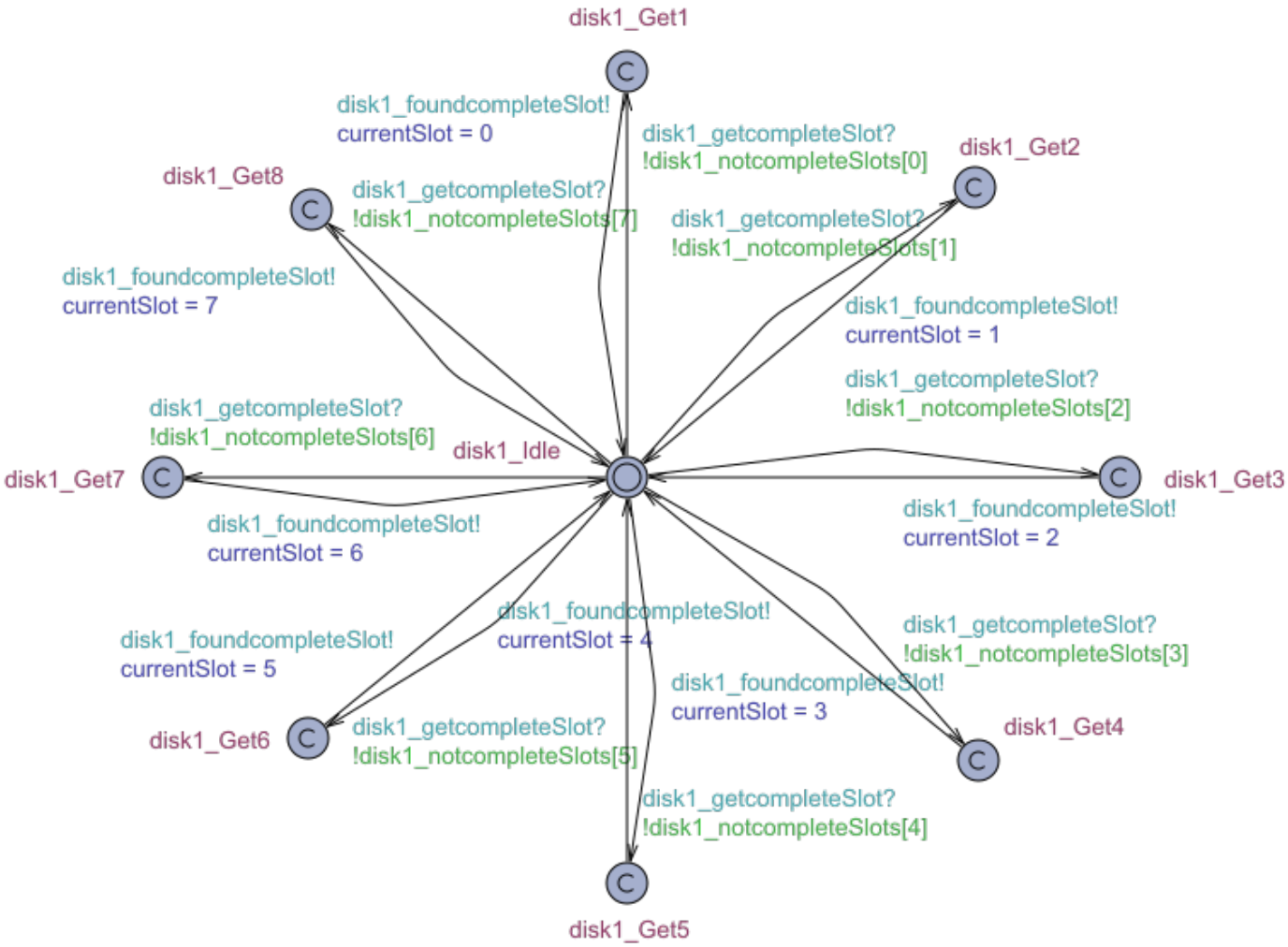

Fig. 5. Disk - Get Complete Slot

F. The Disk Get Free Slot template

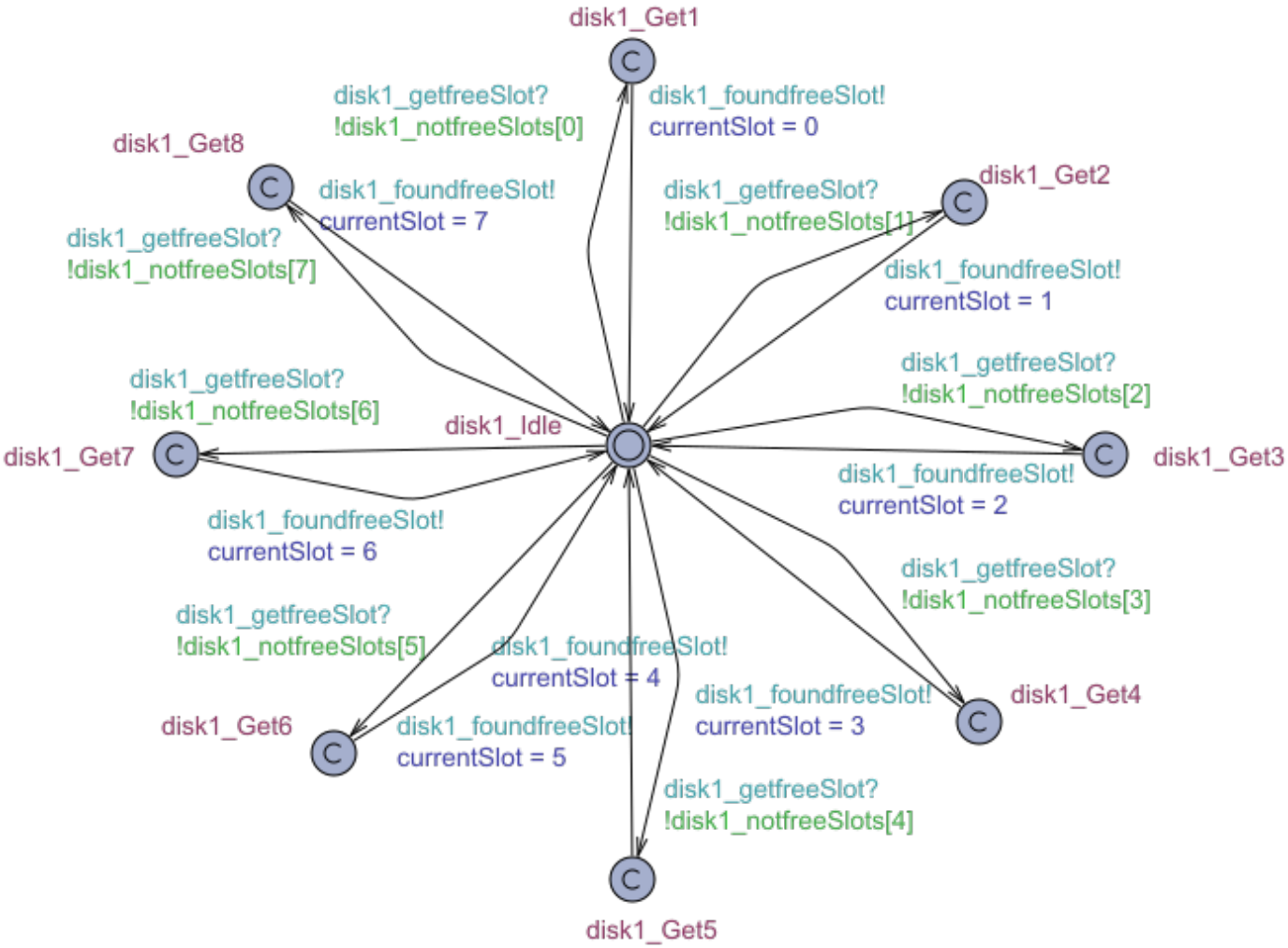

Fig. 6. Disk - Get Free Slot

G. The Get In Progress Slot template

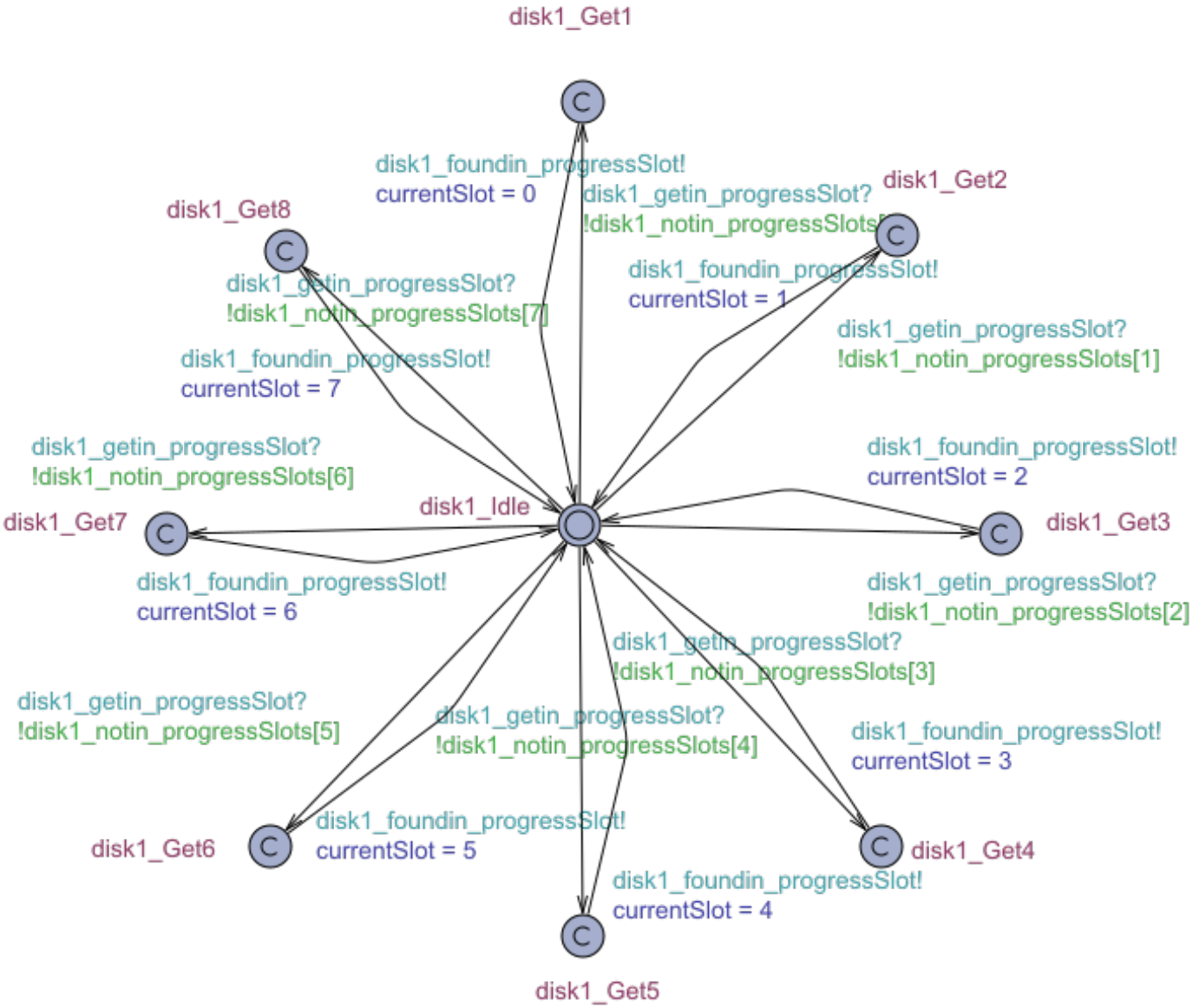

Fig. 7. Disk - Get In Progress Slot

H. The Slot Variable - Complete template

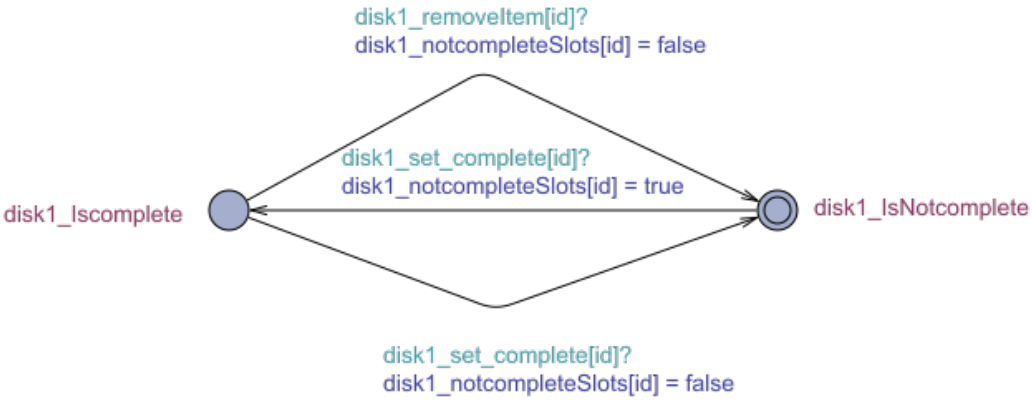

Fig. 8. Disk - Slot Variable Complete

I. The Slot Variable - Free template

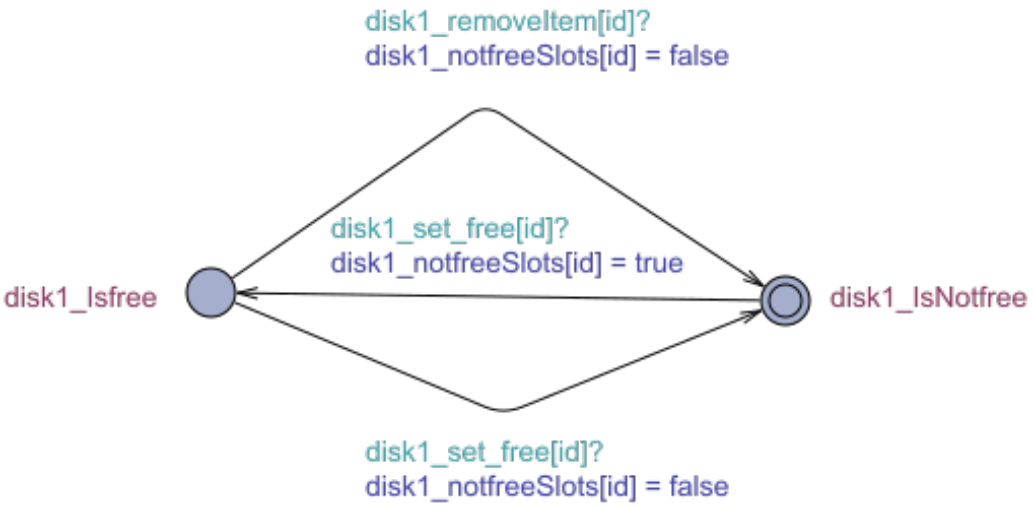

Fig. 9. Disk - Slot Variable Free

J. The Slot Variable - In-progress template

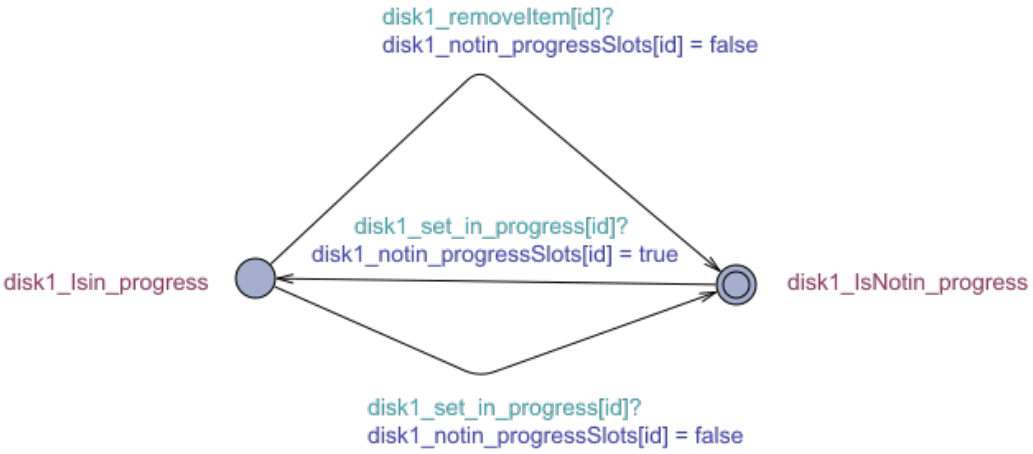

Fig. 10. Disk - Slot Variable In Progress

K. The Crane template

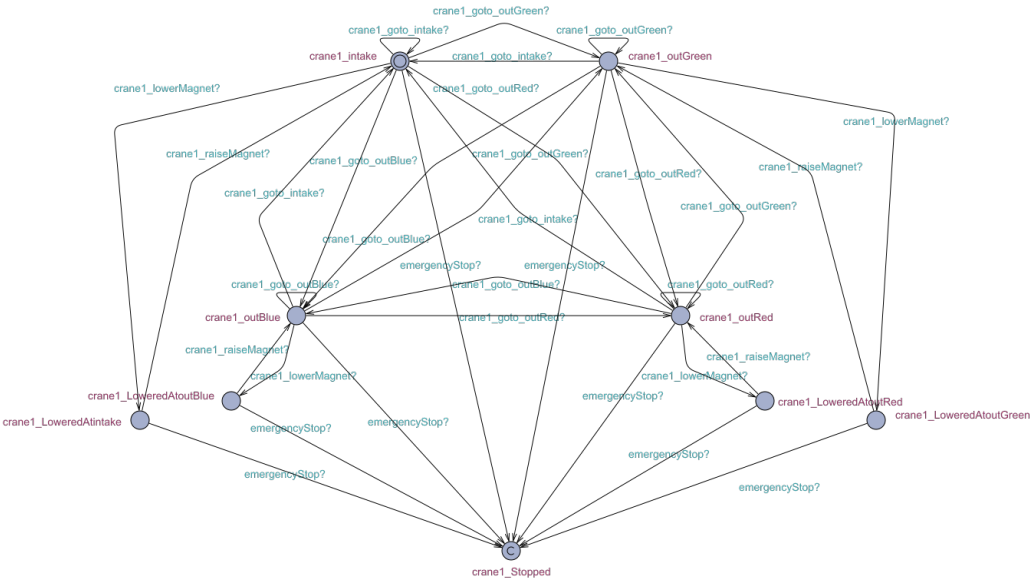

Fig. 11. Crane.

L. The Crane Magnet template

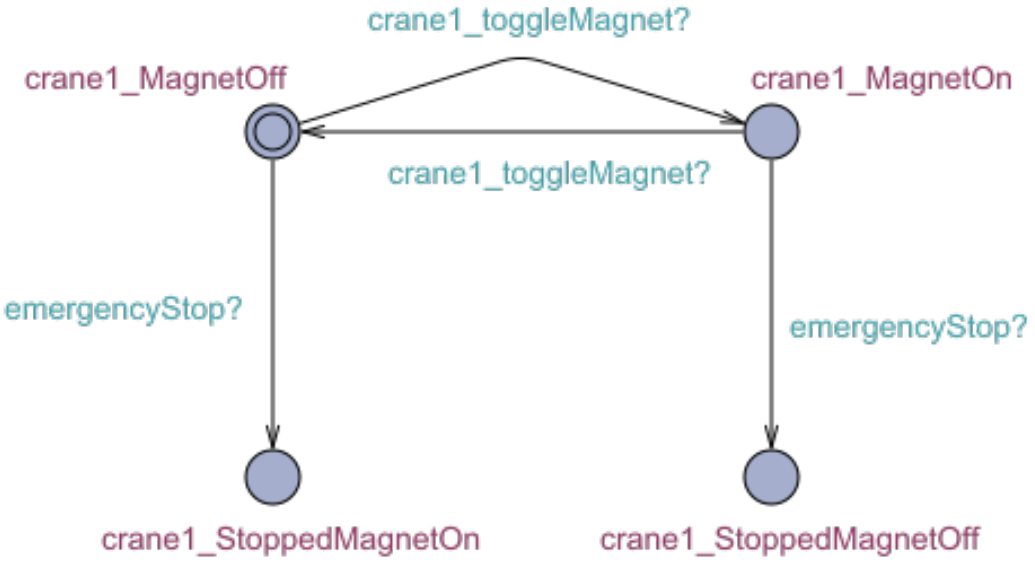

Fig. 12. Crane Magnet.

M. The Camera template

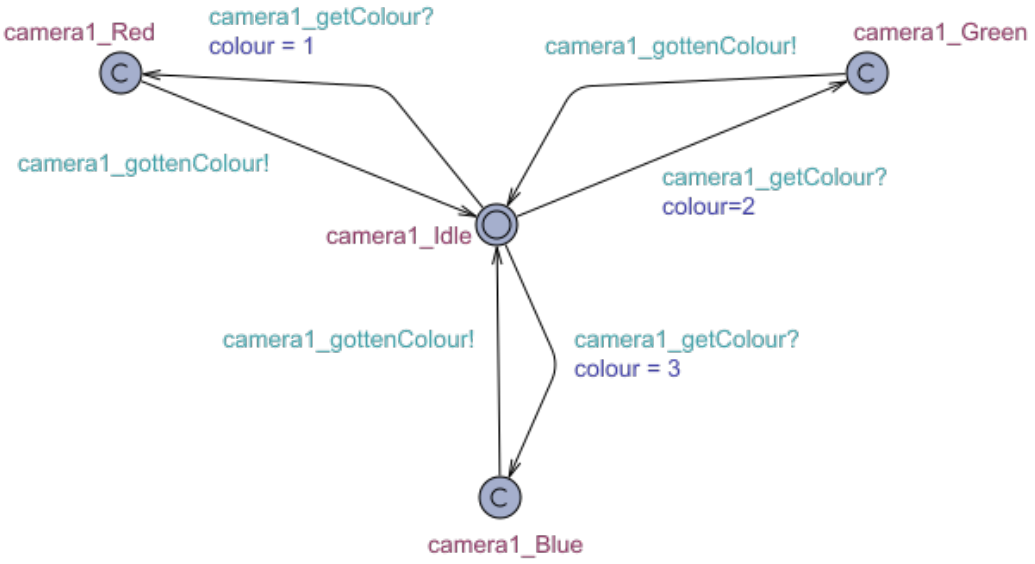

Fig. 13. Camera.

N. The Emergency Button template

Fig. 14. Emergency Button.

APPENDIX D
QUERY RESULTS

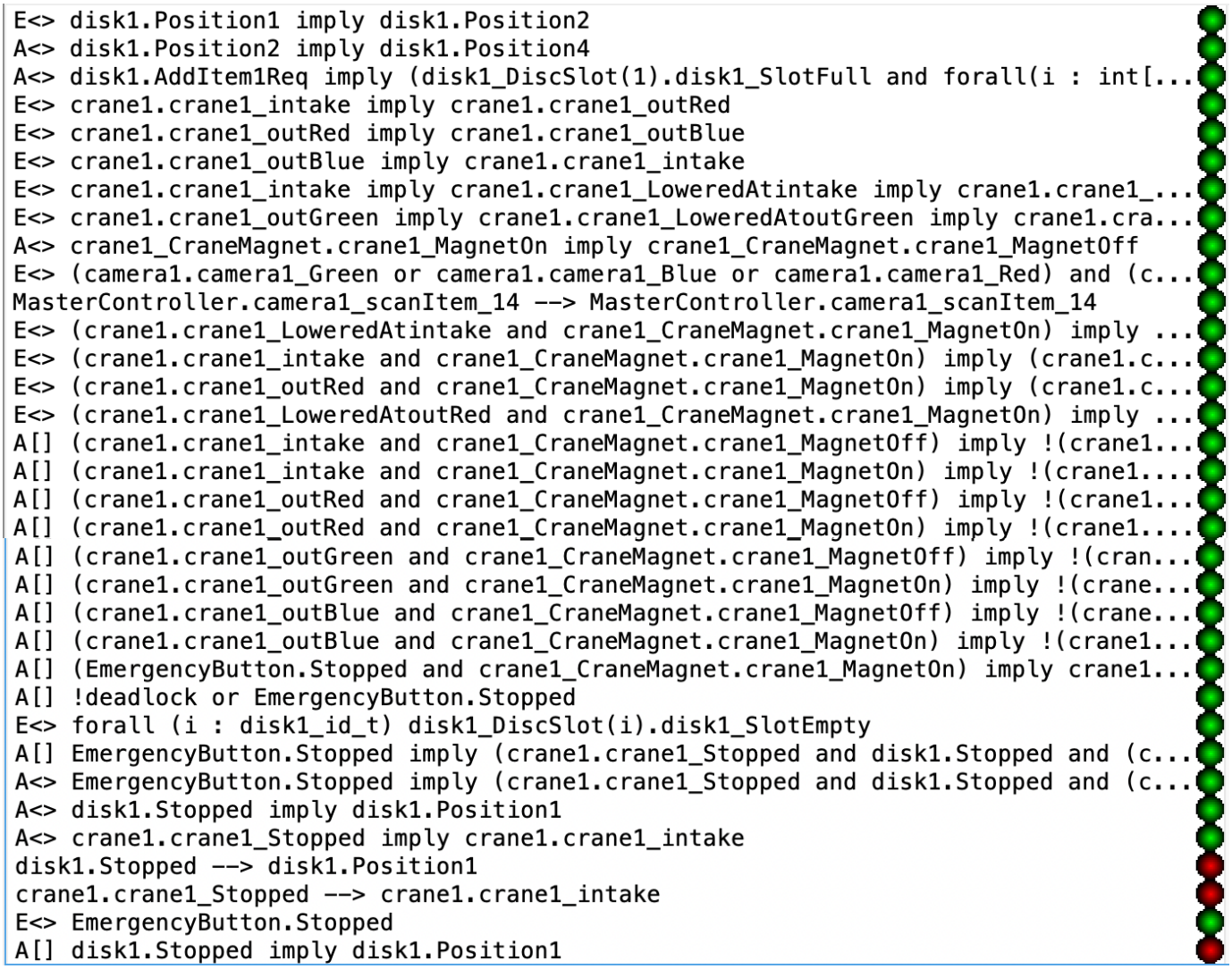

Fig. 15. The status of queries after a full simulation run

APPENDIX E
CONTRIBUTIONS

This appendix describes briefly the contribution for each member in this report.

- Introduction (István)
- Problem Definition (István)
- Requirements (Frederik and Peter)
- The UPPAAL Model
  - Templates (Everyone)
  - Requirements specification (Troels and Nikolai)
- Verification Results (Oliver)
- Discussion (Nikolai)
- Conclusion (Frederik and Peter)
- Appendices (Troels)