Canal Monitoring and Control System (CMCS)

1. Overview

The Canal Monitoring and Control System (CMCS) is used to automate canal locks in response to watercraft that want to pass through a canal system. The CMCS can be installed in any network of canals. Each canal is divided up into segments. Each segment has two ends which can either be a bend, a lock gate, a low bridge, or a system entry/exit point (where boats can move to or from other waterways such as rivers and lakes). A special kind of canal segment is a lock. The water level in a lock can be raised or lowered so its height matches the height of an adjacent segment. A series of locks may exist, but there is always a regular segment in-between two locks. A low bridge is a location where the bridge needs raising for a boat to pass. A bend is simply a place where the canal changes direction - keeping track of these helps simplify mapping of the system.

The CMCS keeps track of the GPS co-ordinates of each segment end, the height of the water above sea level in each segment (which can change in locks), and optionally a name given to a series of locks or segments.

Each watercraft using the CMCS must be equipped with a transponder that includes a GPS unit and transceiver. The transponder transmits a unique ID for the transponder as well as the watercraft’s destination and dimensions when requested. When the captain of a watercraft wants to travel through the canal system, he enters his planned destination in the transponder. The size of the boat is also tracked. The GPS unit regularly transmits the location of the watercraft to the CMCS so the CMCS can monitor traffic.

When a watercraft reaches a gate, the CMCS knows it is there and needs to pass through since it has received the data about the desired destination. The CMCS takes care of opening the gate, raising or lowering of the water level, and then opening the next gate, etc. When a boat reaches a low bridge, a similar procedure takes place, except that raising or lowering the water does not happen.

When there are a lot of boats, the CMCS makes several of them queue up so they can pass through a lock or low bridge at the same time. The CMCS operates only on a first-come-first-serve approach – more sophisticated handling of the traffic flow of watercrafts on the canals is out of scope. The CMCS, however, has to ensure that only the right number and size of boats are put into a lock at once, so the lock does not become overly full (the CMCS knows the size of each lock). Complicating matters is the fact that the network of canals can be navigated in both directions – downstream or upstream. Each entry point to the canal is hence also an exit point.

The maximum depth and height of a watercraft as well as the canal including its low bridges and locks is not considered by the system. It is assumed that signage along the canal as well as maps published about the canal provide this information to the captain of the watercraft. Similarly, it is assumed that no watercraft attempts to navigate the network of canals that is wider or longer than the smallest lock. Again, signage and maps are assumed to exist.
2. Stakeholders
While there are many more stakeholders, this document restricts itself to a brief description of three stakeholders. The captain of a watercraft wants to navigate the canal system as quickly and safely as possible. The system administrator wants a minimal amount of work to be required for maintaining the system. Finally, the funding agency wants to minimize the initial infrastructure costs of the CMCS as well as ongoing costs.

3. Use Cases

![Use Case Diagram]

Figure 1 - Use Case Diagram

3.1 Travel to Destination
Actor: Captain
Goal: To reach the destination entry/exit point.
Precondition: The watercraft is at a start entry/exit point of the canal.
Postcondition: The watercraft has reached the destination entry/exit point.
Main Flow
1) The captain enters the destination and dimensions into the transponder. The destination is one of the entry/exit points of the canal (most likely entered as GPS coordinates).
2) The system sends requests to the transponder every few seconds.
3) The transponder replies to these requests by sending the transponder ID, destination, dimensions, and GPS coordinates of the watercraft.
4) The captain maneuvers the watercraft through the network of canals.
5) If the watercraft does not arrive at a low bridge, continue with step 13.
6) If the watercraft arrives at a low bridge, the system senses the arrival of the watercraft.
7) If the low bridge is lowered (the traffic light of the low bridge is red), the system fully raises the low bridge, taking the traffic on the bridge into account.
8) The system switches the traffic light of the low bridge to green.
9) The captain maneuvers the watercraft past the bridge.
10) The system switches the traffic light of the low bridge to red.
11) The system fully lowers the low bridge.
12) Continue with step 4.
13) If the watercraft does not arrive at a first lock gate, continue with step 28.
14) If the watercraft arrives at a first lock gate, the system senses the arrival of the watercraft.
15) If the first lock gate is closed (the traffic light of the first lock gate is red), the system, taking the watercraft traffic on both sides of the lock into account, evens out the water level on both sides of the first lock gate.

16) The system fully opens the first lock gate.

17) The system switches the traffic light of the first lock gate to green.

18) The captain maneuvers the watercraft past the first lock gate.

19) The system switches the traffic light of the first lock gate to red.

20) The system fully closes the first lock gate.

21) The system evens out the water level on both sides of the second lock gate.

22) The system fully opens the second lock gate.

23) The system switches the traffic light of the second lock gate to green.

24) The captain maneuvers the watercraft past the second lock gate.

25) The system switches the traffic light of the second lock gate to red.

26) The system fully closes the second lock gate.

27) Continue with step 4.

28) If the watercraft has not arrived at its destination, continue with step 4.

29) The watercraft arrives at its destination and exits the network of canals.

Alternative Flows

7.a.1) If the low bridge is raised (the traffic light of the low bridge is green), continue with step 9.

7.b.1) The captain turns the watercraft around and heads in the opposite direction.

7.b.2) Continue with step 4.

10.a.1) The captain turns the watercraft around and heads in the opposite direction again passing the low bridge.

10.a.2) Continue with step 4.

15.a.1) If the lock gate is open (the traffic light of the lock gate is green), continue with step 18.

15.b.1) The captain turns the watercraft around and heads in the opposite direction.

15.b.2) Continue with step 4.

19.a.1) The captain turns the watercraft around and heads in the opposite direction again passing the lock gate.

19.a.2) Continue with step 4.

25.a.1) The captain turns the watercraft around and heads back into the lock.

25.a.2) Continue with step 24.

25.a.2.a.1) The system evens out the water level on both sides of the first lock gate.

25.a.2.a.2) The system fully opens the first lock gate.

25.a.2.a.3) The system switches the traffic light of the first lock gate to green.
25.a.2.a.4) The captain maneuvers the watercraft in opposite direction of original travel past the first lock gate.
25.a.2.a.5) Continue with step 4.

### 3.2 Manage Canals

**Actor:** System Administrator  
**Goal:** To keep the canal information up-to-date.  
**Precondition:** The information is out-of-date or has not yet been entered.  
**Postcondition:** The information is now up-to-date.  

**Main Flow**  
Create, read, update, and delete (CRUD) canal information...

### 4. Class Diagrams

![Main Class Diagram](image)

**Figure 2 – Main Class Diagram**

**Assumptions:**

A1) A lock always has two lock gates and each lock gate is part of only one lock (i.e., if there are two locks in a row, then there is a regular segment in-between the two locks).

A2) It is not necessary to keep track of the exact location of the watercraft as long as it is known to which barrier the watercraft currently is close to and as long as it is known that the watercraft is in a lock.
A3) The unit for length and width is centimeters.
A4) The unit for direction is a degree between 0 and 360.
A5) GPS Coordinates are assumed to be defined.
A6) The unit for the water levels is centimeters. The water level is measured from the top of the lock gate on each side of the lock gate.

OCL Constraints:

C1) Only locks have lock gates as both segment ends:
context Segment inv:
self.oclIsTypeOf(Lock) implies self.downstreamEnd.oclIsTypeOf(LockGate) && self.upstreamEnd.oclIsTypeOf(LockGate) && self = self.downstreamEnd.lock && self = self.upstreamEnd.lock

C2) Segments that are not locks do not have two lock gates as segment ends:
context Segment inv:
!self.oclIsTypeOf(Lock) implies !(self.downstreamEnd.oclIsTypeOf(LockGate) && self.upstreamEnd.oclIsTypeOf(LockGate))

C3) An entry/exit point is the segment end of exactly one segment:
context Canal inv:
self.segments->forAll(s1, s2 | s1.downstreamEnd.oclIsTypeOf(EntryExitPoint) implies s1.downstreamEnd <> s2.downstreamEnd && s1.downstreamEnd <> s2.upstreamEnd) &&
self.segments->forAll(s1, s2 | s1.upstreamEnd.oclIsTypeOf(EntryExitPoint) implies s1.upstreamEnd <> s2.downstreamEnd && s1.upstreamEnd <> s2.upstreamEnd)

C4) A segment end that is not an entry/exit point is the downstream segment end of exactly one segment and the upstream segment end of exactly one different segment:
context Canal inv:
self.segments->forAll(s1 | !s1.downstreamEnd.oclIsTypeOf(EntryExitPoint)}
implies self.segments->exists(s2 | s1 <> s2 && s1.downstreamEnd = s2.upstreamEnd)

C5) Possible values for Integer variables:
context Watercraft inv:
self.length > 0
self.width > 0

ccontext Lock inv:
self.length > 0
self.width > 0

ccontext Bend inv:
self.direction >= 0 && self.direction <= 360

ccontext LockGate inv:
self.waterLevelUpstream > 0
self.waterLevelDownstream > 0

5. State Machine Diagrams

5.1 Description of Low Bridge

Assumptions:

A7) Four zones (A, B, C, D) exist around a low bridge. The GPS/transponder
system detects whether a watercraft is in one of these four zones and invokes
leftBarrier, atBarrier(barrier, “upstream”), and atBarrier(barrier, “downstream”) on the Watercraft instance. Furthermore, the GPS/transponder system also
detects if the watercraft has reached its destination and invokes atDestination.
A8) Location information about the watercraft is updated every few seconds (i.e., the watercraft is detected in each zone).

### 5.2 Watercraft (for Low Bridge)

The state machine diagram for the watercraft in Figure 5 describes how the watercraft reacts to the calls by the GPS system.

- First, the watercraft is initialized with its destination and size. Its current barrier is set to null as initially the watercraft is not close to any barrier.

![Figure 5 – Watercraft passes a Low Bridge (Barrier)](image)

- When the watercraft is in the canal and receives the atDestination call, the final state is reached.
- When the watercraft is in the canal, it may approach a low bridge and therefore receive the atBarrier(barrier, zone) call. barrier identifies the approached low bridge and zone may either be “upstream” or “downstream”. The from attribute of watercraft is set to the zone and the current barrier is set to the low bridge as the watercraft is now close to the low bridge. The watercraft changes to the “needs to pass barrier” state. Furthermore, wcNeedsToPass() is invoked on the current barrier (i.e., a low bridge – see section 5.3 how this is handled by the low bridge).
- In the “needs to pass barrier” state, three situations may occur. 1) The watercraft may leave the low bridge area (leftBarrier). In this case, wcTurnedAround() is invoked on the current barrier and then the current barrier is set to null for the watercraft as the watercraft is not close to the barrier anymore.
• 2) The watercraft stays where it is and does not pass the low bridge. In this case, the atBarrier(barrier, zone) call is received and the zone is the same as the one stored in the from attribute. The watercraft does not need to react to this call.

• 3) The watercraft actually passes the low bridge. In this case, the same atBarrier(barrier, zone) is received. The zone, however, has now changed from “upstream” to “downstream” or vice versa. This is tested with !equalsFrom(zone). Only if the guard evaluates to true, did the watercraft actually pass the low bridge. Then, wcPassed() is invoked on the current barrier and the watercraft is now in the “passed barrier” state.

Finally, in the “passed barrier” state, three situations may occur. 1) The watercraft may leave the low bridge area (leftBarrier). In this case, the watercraft is again “in the canal” and the current barrier is set to null since the watercraft is not close to any barrier anymore.

• 2) The watercraft stays where it is. In this case, the atBarrier(barrier, zone) call is received, but the zone is the not same as the one stored in the from attribute. The watercraft does not need to react to this call.

• 3) The watercraft actually passes the low bridge again (i.e., it traveled in the opposite direction). In this case, atBarrier(barrier, zone) is also received. The zone, however, is now the same as the one stored in the from attribute. This is tested with equalsFrom(zone). Only if the guard evaluates to true, did the watercraft actually pass the low bridge again. In this case, the watercraft is back in the “needs to pass barrier” state and the entry action of this state ensures that wcNeedsToPass() is invoked on the current barrier.

5.3 Low Bridge

The state machine diagram for the low bridge in Figure 6 describes how the low bridge reacts to the calls by watercrafts and its motor and the passing of time. The low bridge keeps track of the number of watercraft wanting to pass the bridge (nrWC).

• Initially, nrWC is set to 0, the closedSince time is set, and the low bridge is in the Closed state. At any time, wcNeedsToPass and wcTurnedAround calls may increase and decrease nrWC, respectively.

• If there is at least one watercraft that wants to pass the low bridge and the minClosedTime has passed, the low bridge moves to the Opening state in which the motor opens the bridge.

• When the motor signals that the bridge has been opened (barrierOpened), the openedSince time is set and the traffic lights are switched to green. The low bridge is now in the Opened state.

• In this state, watercraft may pass (wcPassed), which decreases nrWC. If no watercrafts are waiting to pass the low bridge anymore or the maxOpenedTime has been reached, the traffic lights are switched to red and the motor closes the bridge. The low bridge is now in the Closing state.

• When the motor signals that the bridge has been closed (barrierClosed), the closedSince time is set and the low bridge is now again in the Closed state.
Assumptions:

A9) The bridge motor automatically stops at the highest/lowest point and acknowledges this with the barrierOpened and barrierClosed calls.

A10) The bridge motor also automatically stops closing the bridge if something is underneath the bridge and continues only when it is clear again.

A11) Opening the bridge includes stopping all traffic on the bridge first. Traffic on the bridge is allowed again in the Closed state.

5.4 **Updates to the Class Diagrams for Low Bridge Behavior**

The initial class diagrams were derived from the problem description only. Now with the behavior of the low bridge specified in more detail, the class diagram needs to reflect the additional required attributes and operations as shown in this section.

Assumptions:

A12) The unit for maxOpenedTime and minClosedTime is seconds.

OCL Constraints:

C6) Possible values for new Integer variables and String variables:

context Watercraft inv:
self.from = “upstream” || self.from = “downstream”

context Barrier inv:
self.maxOpenedTime > 0

context LowBridge inv:
self.minClosedTime > 0
C7) Postconditions of the three wc… methods:
context Barrier::wcNeedsToPass()
post: nrWC = nrWC@pre + 1

class Barrier::wcPassed()
post: nrWC = nrWC@pre - 1

ccontext Barrier::wcTurnedAround()
post: nrWC = nrWC@pre – 1

Figure 7 – Additions to Class Diagram due to State Machines for Low Bridge

5.5 Description of Lock

Figure 8 – Lock Overview
Assumptions:
A13) Five zones (A, B, C, D, E) exist around a lock. The GPS/transponder system detects whether a watercraft is in one of these five zones and invokes leftBarrier, atBarrier(barrier, “upstream”), atBarrier(barrier, “downstream”), and inLock on the Watercraft instance.
A14) Location information about the watercraft is updated every few seconds (i.e., the watercraft is detected in each zone).
A15) Watercraft traffic going downstream has priority over traffic going upstream if watercrafts are waiting on either side of the lock (i.e., the upstream lock gate is opened first).
A16) Before a lock gate can be opened, the water level on each side of a lock gate must be evened out by opening a valve that connects the two sides.

5.6 Watercraft (for Lock)
The state machine diagram for the watercraft in Figure 9 describes how the watercraft reacts to the calls by the GPS system. Figure 9 is an addition to the state machine in Figure 5 as most of the behavior is exactly the same. The difference lies in that a watercraft may also be now in the lock. Note that the “needs to pass barrier” and “passed barrier” states including their entry actions are the same as in Figure 5.

• In the case of a lock instead of a low bridge, the “needs to pass barrier” state is left when the inLock call is received (since inLock will occur before the second atBarrier defined in Figure 5). The current barrier was already set to the lock gate in the portion of the state machine shown in Figure 5. When leaving the “needs to pass barrier” state, wcPassed is invoked on the current barrier (i.e., a lock gate – see section 5.8 how this is handled by the lock gate) and the watercraft is now in the “in lock” state. Upon entry into the “in lock” state, the watercraft is added to the list of watercraft currently in the lock.

• In the “in lock” state, three situations may occur. 1) The watercraft may stay in the lock. In this case, the inLock call is still received. The watercraft does not need to react to this call.

• 2) The watercraft passes the second lock gate. In this case, the atBarrier(barrier, zone) call is received. The zone, however, has now changed from “upstream” to “downstream” or vice versa. This is tested with !equalsFrom(zone). Only if the guard evaluates to true, did the watercraft actually pass the second lock gate. Note that the from attribute was set by the state machine defined in Figure 5. The watercraft is now in the “passed barrier” state. Upon exit of the “in lock” state, the watercraft is removed from the list of watercraft currently in the lock.

• 3) The watercraft actually passes the first lock gate again (i.e., it traveled in the opposite direction). In this case, atBarrier(barrier, zone) is also received. The zone, however, is now the same as the one stored in the from attribute. This is tested with equalsFrom(zone). Only if the guard evaluates to true, did the watercraft actually pass the first lock gate again. In this case, the watercraft is back in the “needs to pass barrier” state. Upon exit of the “in lock” state, the watercraft is removed from the list of watercraft currently in the lock. The entry action of the “needs to pass barrier” state ensures that wcNeedsToPass() is invoked on the current barrier.
Finally, once the watercraft is in the “passed barrier” state, the watercraft may travel back into the lock (again changing its original direction). In this case, inLock is received and the watercraft is now back in the “in lock” state.

5.7 Lock

The state machine diagram for the watercraft in Figure 10 describes how the lock coordinates the opening and closing of its two lock gates. cond1 in the figure is “upstreamEnd.getNrWC() > 0 && isRoomInLock(upstreamEnd.watercrafts)” and cond2 is “downstreamEnd.getNrWC() > 0 && isRoomInLock(downstreamEnd.watercrafts)”.

- cond1 ensures that the upstream gate is only opened if there are watercraft waiting and there is enough room in the lock. isRoomInLock considers the watercraft already in the lock (Lock.inLock) and the watercrafts passed in as a parameter. Only the watercrafts in the “needs to pass barrier” state are considered.
- The downstream gate is only opened if the upstream gate is not to be opened, there are watercraft waiting, and there is enough room in the lock.
- In both cases, the lock gate is activated by opening its valve (see 5.8 how this is handled by the lock gate). When the lock gate is closed again, the lock gate invokes gateInactive on the lock.
- This causes either the other lock gate to be activated (if the lock is not empty) or the lock returns to the “lock empty” state.
Assumptions:

A17) An undesired behavior could occur for the lock in the following situation. If a lock gate is opened but none of the watercraft enters the lock, the lock gate will be closed after a timeout. In this case, the lock returns to the “lock empty” state and then will activate the same lock gate (since the conditions have not changed as the watercrafts remained stationary). This could go on indefinitely. However, it is assumed that watercrafts will in almost all cases follow the normal behavior of entering the lock when given an opportunity. A more sophisticated way of selecting a lock gate to be opened could replace cond1 and cond2 in the state machine for the lock in order to address such exceptional cases.

5.8 Lock Gate

The state machine diagram for the lock gate in Figure 11 describes how the lock gate reacts to the calls by the lock, watercrafts, and its motor and the passing of time. The lock gate keeps track of the number of watercraft wanting to pass the lock gate (nrWC).

- Initially, nrWC is set to 0 and the lock gate is in the Closed state. At any time, wcNeedsToPass and wcTurnedAround calls may increase and decrease nrWC, respectively.
- When the lock gate receives the openValve call from its lock, the lock gate actually opens the valve and is now in the Valve Opened state. In this state, the water levels are read.
- When the water level is even out, the valve is closed and the motor starts opening the gate. The lock gate is now in the Opening state.
• When the motor signals that the lock gate has been opened (barrierOpened), the openedSince time is set and the traffic lights are switched to green. The lock gate is now in the Opened state.

![Figure 11 – Lock Gate Behavior](image)

• In this state, watercraft may pass (wcPassed), which decreases nrWC. If no watercrafts are waiting to pass the lock gate anymore or there is no room left in the lock or the maxOpenedTime has been reached, the traffic lights are switched to red and the motor closes the lock gate. The lock gate is now in the Closing state.

• When the motor signals that the bridge has been closed (barrierClosed), the lock gate invokes gateInactive on its lock and the lock gate is now again in the Closed state.

Assumptions:
A18) The gate motor automatically stops when opening/closing the gate and acknowledges this with the barrierOpened and barrierClosed calls.
A19) The gate motor also automatically stops closing the gate if something is blocking the gate and continues only when it is clear again.

### 5.9 Updates to the Class Diagrams for Lock Behavior

Similar to before, the behavior of the lock gate is now specified in more detail and the class diagram needs to reflect the additional required attributes and operations as shown in this section. Note that only inLock() was added to Watercraft.
6. Component Diagram

The component diagram in Figure 13 shows the major building blocks of the canal control system and their required and provided interfaces. The Canal Controller is the heart of the system as it implements the main application logic. It provides an interface to the GPS/Transponder System to update the watercraft position and information. The GPS/Transponder System requires the CanalInformation interface to access the location information of the segment ends of the canal to determine the position of watercrafts given the signals from the watercraft transponders. The Canal Controller provides the same CanalInformation interface to the UI component to access and update application data. The Canal Controller also uses the Persistence interface of the database to store application data. Finally, four devices (Motor, Valve, TrafficLight, and WaterLevelSensor) are controlled by the Canal Controller via their Control interfaces. In addition, the Motor component requires the MotorFeedback interface to indicate the successful completion of motor control tasks. This interface is also provided by the Canal Controller.
7. Deployment Diagram

The deployment diagram in Figure 14 shows how the artifacts corresponding to the components from section 6 are deployed to physical computational resources, i.e., nodes. It also shows the communication paths between such nodes. The system consists of five types of nodes: clients for the system administrator, the central server, the GPS/Transponder server, the low bridge controllers, and the lock gate controllers.

Figure 14 – Deployment Diagram