Testing AGVs in Dynamic Warehouse Environments

Alexander Helleboogh, Tom Holvoet and Yolande Berbers

AgentWise, DistriNet, Department of Computer Science K.U.Leuven University, Belgium
{Alexander.Helleboogh,Tom.Holvoet,Yolande.Berbers}@cs.kuleuven.be

Abstract. Automatic Guided Vehicles (AGVs) are unmanned vehicles that can transport loads in a warehouse. AGVs are instructed by on-board AGV control software. As multiple AGVs operate in a decentralized manner in the warehouse environment, conflicts may arise. Consequently, it is crucial to test thoroughly whether the AGV control software actually handles the potential conflicts in the appropriate way.
In this paper, we employ a simulated warehouse environment to test the AGV control software. The AGV control software is embedded and activated in the simulated warehouse environment. The simulated warehouse environment provides support for testing by means of (1) representing dynamism in the warehouse environment in an explicit manner, and (2) detecting conflicts of dynamism in an automated way. The approach is illustrated for the case of testing collision avoidance.

1 Introduction

Since March 2004 the AgentWise research group is involved a joint R&D project, called *Egemin Modular Controls Concept* (EMC^2) in cooperation with Egemin, an industrial expert in automating warehouse transportation systems [1]. An AGV transportation system is an industrial transport system using several automatic guided vehicles (AGVs). Typical applications are repackaging and distributing incoming goods to various branches, or distributing manufactured products to storage locations. An AGV is an unmanned, computer-controlled transportation vehicle using a battery as its energy source. AGVs have to perform transports. A transport consists of picking up a load at a particular spot in the warehouse and bringing it to its destination. Transports are generated by client systems, for example business management programs, particular machines, employees or service operators.

Traditionally, AGVs in a warehouse are directly controlled by a central server. AGVs have limited autonomy: the server plans the schedule for the system as a whole, dispatches commands to the AGVs and continually polls their status. This system architecture has successfully been deployed in numerous practical installations. The centralized server architecture has two main benefits. The control software can be customized easily to the needs of a particular project, since the server is a central configuration point. This allows for specific per-project optimizations. A second benefit is that the system is deterministic and predictable.

In the EMC^2 project, we are investigating the feasibility of a decentralized system architecture [2,3] to improve the flexibility of the system. We use concepts from *situated*

multi-agent systems (situated MAS) [4]. In our approach, each AGV is controlled by a situated agent. The agents cooperate to ensure the functionality of the system. In contrast to the centralized server architecture, each agent of the situated MAS takes decisions based on local information only. Situated agents deal with opportunities and tackle problems in a decentralized manner.

The warehouse environment the agents are situated in, is inherently dynamic. It contains different AGVs that are constantly driving around, sending messages and manipulating loads. Consequently, conflicts may arise locally between different AGVs each acting autonomously. Examples of conflicts are collisions between AGVs and communication loss because of congestion of the communication channel.

It is evident to test the situated agents thoroughly before they are deployed on real AGVs. In decentralized systems, testing is necessary to determine whether the situated agents actually handle potential conflicts in the appropriate way. Formal approaches are practically infeasible to verify the behavior of decentralized systems [5], such as a situated MAS. This emphasizes the importance of simulation as a means to verify the behavior of decentralized systems [6,7].

In this paper, we employ a *simulated warehouse environment* to test the situated agents that control the AGVs. The simulated warehouse environment is a model of the real warehouse environment, and contains simulated AGVs. The agents are tested by deploying and activating them in the simulated environment [8]. The simulated warehouse environment facilitates testing by offering a means to (1) represent dynamism in the warehouse environment in an explicit manner, and (2) detect conflicts of dynamism in an automated way. The approach is illustrated for the case of testing collision avoidance between AGVs in the presence of unreliable communication.

The remainder of this paper is structured as follows. In Sect. 2, we elaborate on the real warehouse environment. In Sect. 3, we describe the model of the simulated warehouse environment that was developed to represent the real warehouse environment. In Sect. 4, we explain how the simulated environment supports testing collision avoidance. We evaluate the approach in Sect. 5 and draw conclusions in Sect. 6

2 The Real Warehouse Environment

We focus on two parts of the warehouse environment: the warehouse layout and the AGVs. The warehouse layout is discussed in Sect. 2.1. In Sect. 2.2, we focus on the architecture of an AGV. Section 2.3 analyzes how collisions can occur in the warehouse environment. The requirements for avoiding collisions are specified in Sect. 2.4.

2.1 The Warehouse Layout

The warehouse layout typically contains various loads positioned at various locations in the warehouse. Loads are typically stored in racks. Racks are used to hold loads and are positioned across the warehouse layout, usually according a geometrical pattern that combines easy accessibility of the loads, as well as efficient use of the available room for storage purposes. Typically, also one or several battery chargers for the AGVs are positioned at particular locations on the warehouse layout.

To support AGVs, the warehouse layout is usually customized. This typically involves a custom configuration of the racks. In addition, a complex layout of magnet strips is built into the warehouse floor to guide the AGVs to move from one spot in the warehouse to another. This *magnet track* allows AGVs to maneuver in an accurate manner according to predefined pathways. Moreover, as magnets are inexpensive and can be installed easily, magnet guided navigation is relatively cost-effective.

2.2 Architecture of an AGV

In Fig. 1, the architecture of an AGV is depicted. Each AGV consists of both hardware and software. The hardware of an AGV comprises a number of hardware modules. *AGV sensor modules* represent the sensors to detect the position and battery level of the AGV. *AGV actuator modules* represent the various actuators, such as the engines to move and the lift to pick loads. The *AGV WiFi module* represents the wireless communication infrastructure to send and receive messages. The software of an AGV comprises two main modules: the *AGV controller* and the *AGV agent*.

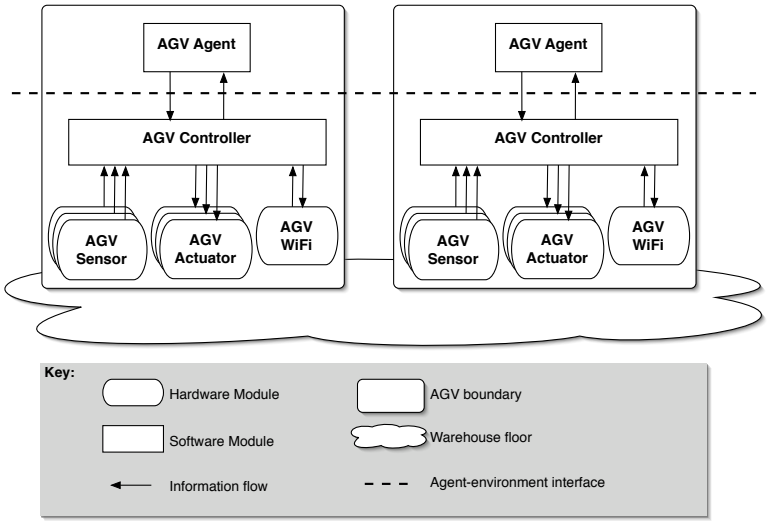

Fig. 1. The architecture of AGVs.

The *AGV agent* encapsulates the logic to steer the AGV. The AGV agent uses the AGV controller to steer the AGV. The *AGV controller* takes care of all interfacing with the hardware, and determines the granularity of control that can be used to steer an AGV.

The granularity of control offered by the AGV controller is determined by a *logical map* that is a representation of the magnet track, but is expressed in the AGV controller in terms of road segments connected by stations:

- *Road segments*. A road segment corresponds to a particular part of the magnet track, and has a unique identifier. The granularity of road segments is typically chosen such that they represent a physical distance of three to five meters. Road segments can be unidirectional or bidirectional.
- *Stations.* Stations are the logical nodes at the beginning and end of road segments. A station corresponds to a particular spot on the magnet track. A station indicates a special-purpose location for AGVs. A particular station can offer a location for a subset of the following purposes:
  - *Routing.* A station can serve as a location that connects various road segments and allows AGVs to choose alternative routes.
  - *Storage.* A station can serve as a location where loads can be picked up or put down.
  - *Battery charging.* In case a station is positioned at the location of a battery charger, it can offer an AGV the possibility to charge its battery.
  - *Parking.* A station can serve as a location where AGVs can park temporarily, e.g. in case there are no more pending transportation tasks.

The AGV controller allows an AGV agent to steer the AGV *per segment*: the AGV can stop on every station, where it can be instructed to change direction. The AGV controller uses the low-level instructions of the hardware of an AGV in order to stay on the magnet track until the next station is reached. The AGV controller offers the following actions to steer the AGV:

- `move(segment)`: this instructs the AGV controller to drive the AGV over the given segment until the next station is reached.
- `pick(segment)`: instructs the AGV controller to drive the AGV over the given segment and pick up a load at the station at the end of it.
- `drop(segment)`: the same as pick, but drops a load the AGV is carrying.
- `park(segment)`: instructs the AGV controller to drive the AGV over the given segment and park at the station at the end of that segment.
- `charge(segment)`: instructs the AGV controller to drive the AGV over a given segment and start charging batteries at the station at the end of that segment.
- `sendBroadcast(message)`: instructs the AGV controller to broadcast a given message using onboard wireless communication infrastructure.
- `sendUnicast(message,receiver)`: instructs the AGV controller to send a given message to a given receiver.

Furhermore, the AGV controller can be used to perform the following perceptions to inspect the status of an AGV.

- `getPosition()`: instructs the AGV controller to determine the current position of the AGV. This is the position of a particular reference point situated on the robot.
- `getBatteryLevel()`: instructs the AGV controller to read out the remaining energy level of the battery of the AGV.
- `getMessage()`: instructs the AGV controller to return the next message in the inbox of the AGV, which contains all messages received.
- `isLoaded()`: instructs the AGV controller to check whether the AGV is currently holding a load.

– `getAction()`: instructs the AGV controller to inspect which action the AGV is currently performing, i.e. busy driving, picking, dropping or charging.

2.3 Collisions in the Warehouse Environment

The warehouse environment in which the agents are situated, is highly dynamic. A dynamic environment is an environment that changes in ways beyond an agent's control [9]. Each agent experiences dynamism in the environment, primarily originating from other AGVs that are constantly driving around, sending messages and manipulating loads. Consequently, there is the possibility that conflicts arise between AGVs in the warehouse environment. The conflicts of interest in this paper are collisions of AGVs.

The movement of an AGV on the warehouse floor over a road segment towards another station can be initiated using a `move`, `pick`, `drop`, `park` or `charge` action. In the dynamic warehouse environment, the movement of an AGV can cause collisions in the following ways:

– *With other AGVs.* Although the layout of road segments is static, the traffic load caused by other AGVs on the layout changes continuously. Consequently, the road segment over which an AGV is driving can become obstructed because of the movement of another AGV. This can lead to collisions.
– *With obstacles.* In a warehouse environment, all kinds of obstacles can appear on the road segments. Examples of obstacles are loads that fall off of AGVs, other AGVs that are out of order because they have collided, broken down or ran out of energy. AGVs can collide into obstacles, as obstacles typically hinder passage on particular road segments.

Agents typically rely on communication to anticipate collisions. However, the unreliability of the communication channel is an important factor that has to be taken into account by the agents. The transmission of communication messages can be initiated by using a `sendUnicast` or `sendBroadcast` action. In the dynamic warehouse environment, the transmission of messages is not reliable and can affected in the following ways:

– *Limited communication range.* The AGV's wireless on-board communication infrastructure can only send and receive messages within a limited range. Consequently, communication is affected as soon as AGVs are moving out of each other's range.
– *Interference of transmission.* The transmission of a communication message can be hindered or delayed by concurrent transmissions of other AGVs within range, or by external sources working on the same channel.

2.4 Requirements for Avoiding Collisions

For an AGV transportation system, the AGV control software has to adhere requirements that specify how AGVs should cope with potential conflicts. We focus on *collision avoidance* in the presence of *unreliable communication*.

The movements of AGVs are not allowed to cause conflicts in the warehouse environment. It is required that the situated agents prevent the AGVs from colliding with each other or with obstacles. The central concept in preventing collisions is the *minimal safety distance*. For each AGV, the agent has to maintain a minimal safety distance at all times with respect to obstacles or other AGVs. This minimal safety distance takes into account the maximum deviation of an AGV with respect to the path of the magnet strip it follows. For AGVs the minimal safety distance is typically about 10 centimeters[1]. It is physically possible that an AGV can pass at a distance smaller than the minimal safety distance. However, it is required that the minimal safety distance is respected in order to guarantee a safe passage *at all times*.

An important factor that has to be taken into account with respect to collision avoidance, is the unreliability of the communication between AGVs. AGVs are required to maintain the minimal safety distance, even in the presence of unreliable communication. A number of quality-of-service attributes specify the worst-case communication characteristics under which correct and safe functioning of the AGVs is required. For the AGVs, a worst-case quality is typically characterized by about 40 percent message loss and 2 seconds transmission delay. Safe and normal operation of the AGVs is required as long as the quality of service is better than the worst-case's. In case the wireless communication quality drops beneath the worst-case's, AGVs are allowed to go into a safe mode, typically suspending any further movements to prevent unsafe situations.

3 The Simulated Warehouse Environment

In this section, we describe the simulated warehouse environment we have developed. A simulated environment is a model of the real environment [10]. We first explain how the situated agents are embedded in the simulated warehouse environment in Sect. 3.1. Then we describe the model of the simulated warehouse environment. In the model of the simulated warehouse environment, we make a distinction between two concerns: state and dynamism. The state (see Sect. 3.2) is concerned with modeling a snapshot at a particular point in time of all parts that constitute the warehouse environment. Dynamism (see Sect. 3.3) is concerned with representing in an explicit manner the evolution of the simulated warehouse environment over time. In Sect. 4, we illustrate how the model is used to test collision avoidance.

3.1 Embedding Situated Agents in the Simulated Warehouse Environment

From Fig.1, it is clear that agents are software modules that are embedded in physical AGVs. From the viewpoint of an agent program, all interaction with the environment is mediated by the interface provided by the AGV controller. Consequently, the simulated warehouse environment has to provide the same interface to the agents. However, instead of being wired to the AGV controller of a real AGV, the simulated warehouse environment redirects the agents' invocations on the interface into actions and perceptions performed by a simulated AGV.

[1] The order of magnitude of numerical data used throughout this paper is based on typical data from several industrial AGV projects.

Each agent autonomously decides at what time to invoke an action on the AGV controller. The amount of time it takes an agent to decide upon what to do, results in a delay for all its subsequent actions. To deduce the precise moment in time an action is invoked, we need to determine how long an agent has been thinking or waiting. We developed an approach to map the internal process of an agent to simulation time. The approach relies on aspect weaving to insert all code to maintain, update and synchronize the logical clocks in a transparent way [11].

3.2 Modeling State

The state of the simulated warehouse environment contains all information to describe the actual state of affairs in the real warehouse environment at a given point in time. A state description is always considered at a particular point in time [12]. The state of the simulated warehouse environment is comprised of *environmental entities* and *environmental relationships*. Figure 2 shows a simplified example of the state of the simulated warehouse environment at a particular point in time.

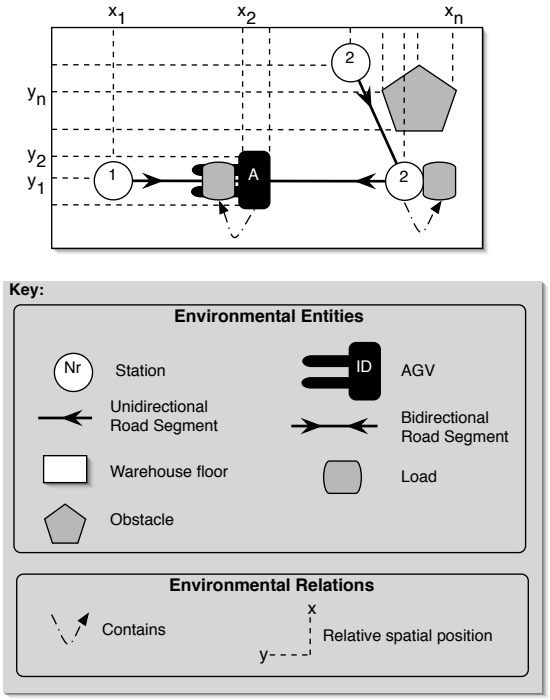

Fig. 2. The state of the simulated warehouse environment.

Environmental Entities. Environmental entities are characterized by their own, distinct existence in the environment. The following environmental entities of the warehouse environment are modeled in the simulated environment:

- *AGVs*. AGVs are characterized by a bounding volume to represent the physical size, a battery level, an inbox and an outbox for the wireless communication infrastructure.
- *Loads*. Loads are characterized by a bounding volume that represents the size of the load.
- *Obstacles*. Obstacles are characterized by a bounding volume that represents the size of the obstacle.
- *Warehouse floor*. The warehouse floor is characterized by a two-dimensional area with given size.
- *Road segments*. The magnet track is modeled in terms of the logical map representation that is employed by the AGVs, i.e. in terms of road segments and stations. A road segment can be unidirectional or bidirectional. In the simulated warehouse environment, each road segment is characterized by a direction and a length.
- *Stations*. Stations are locations that connect adjacent road segments. Each station is annotated with the purposes it can be used for, i.e. a location for routing, storage of loads, parking and/or battery charging.

Environmental Relations. An environmental relation is a particular relation between several environmental entities that expresses how these entities are related to each other at a given point in time.

- *Spatial relations.* All environmental entities are spatially related to each other [13]. The spatial relations of all entities are expressed relative to the warehouse floor, using a two-dimensional continuous coordinate system.
- *Containment relations.* Containment relations are used with respect to loads. A containment relation is used to indicate in an explicit manner whether a specific AGV is holding a particular load. For example, a containment relation indicates that a load that is still contained by a particular station and is not yet picked up by an AGV. As soon as an AGV picks up the load, the containment relation indicates the load as being contained by that AGV.

3.3 Dynamism

Until now, we focussed on the static description of the simulated environment. We now elaborate on the way the environment evolves *over time*.

Dynamism as Activities. Dynamism is the evolution of environmental entities and environmental relations over time. An example of dynamism in the warehouse environment is the movement of an AGV driving over a road segment. In the simulated warehouse environment, dynamism is represented as a first-order abstraction, by means of *activities* [14]. An activity represents a well-specified evolution of a particular environmental entity or relation, that happens over a specific time interval. Consequently, the description of an activity comprises the following:

- *A specification of the time interval.* Dynamism happens over time. The time interval of an activity specifies the point in time a particular activity starts and how much time it takes until the evolution completes. The time interval is custom for each activity and can be configured in correspondence to the characteristics measured in the real world.
- *The environmental entity or relation involved.* Dynamism is related to particular environmental entities or relations. Consequently, each activity incorporates a description of the part of the environment it describes the evolution of.
- *An evolution strategy.* Dynamism evolves in a particular, gradual way. Consequently, each activity incorporates a description of the specific evolution as a function of time within the time interval of the activity.

Initiation of Activities. Agents perform actions by invoking methods in the interface of the AGV controller. The invocations on the AGV controller typically initiate activities in the simulated warehouse environment. As such, the invocation of a method by the agent on the interface of the AGV controller, is decoupled from the activity that is initiated in the simulated warehouse environment as the result of the invocation. The time at which an agent triggers the AGV controller (see Sect. 3.1) corresponds to the start of the time interval of the activity it initiates.

Activities in the Simulated Warehouse Environment. In the simulated warehouse environment, activities are used to represent driving, sending messages, lifting and putting down loads, charging the battery, etc. We describe the activity that represents the driving of an AGV in detail:

- `DriveActivity`. A drive activity represents the driving of a particular AGV.
  - *Initiation.* A drive activity can be initiated in case the agent invokes a `move`, `pick`, `drop`, `park` and `charge` action on the AGV controller.
  - *Time Interval.* The time interval of the drive activity is calculated in terms of the physical performance of the AGV en the length of the road segment.
  - *The environmental entity or relation involved.* A drive activity describes the evolution of two parts of the state of the simulated environment: (1) the environmental relation that describes the position of the AGV on the warehouse floor and (2) the battery level of the simulated AGV.
  - *Evolution strategy.* The position of an AGV describes the path over the road segment. The position changes over time, approximated by a model of a constant velocity of 2 meters each second. However, during the first 4 seconds of each drive activity, the speed of the AGV increases linearly to represent its acceleration, whereas during the last 2 seconds, its speed decreases linearly to represent the AGV's deceleration. The evolution of the battery level of an AGV is approximated by a model describing a linear decrease according to the distance travelled.

In Fig. 3, an example of a drive activity of AGV A over time interval $(2 \rightarrow 7)$ is depicted. We depict each drive activity as a hull that wraps the intermediate positions that are taken by the AGV over time. In Fig. 3, the evolution represented by the drive

activity is illustrated using two snapshots of the state within time interval $(2 \rightarrow 7)$: at time $T = 3$ and $T = 5$, showing the instant position of the AGV.

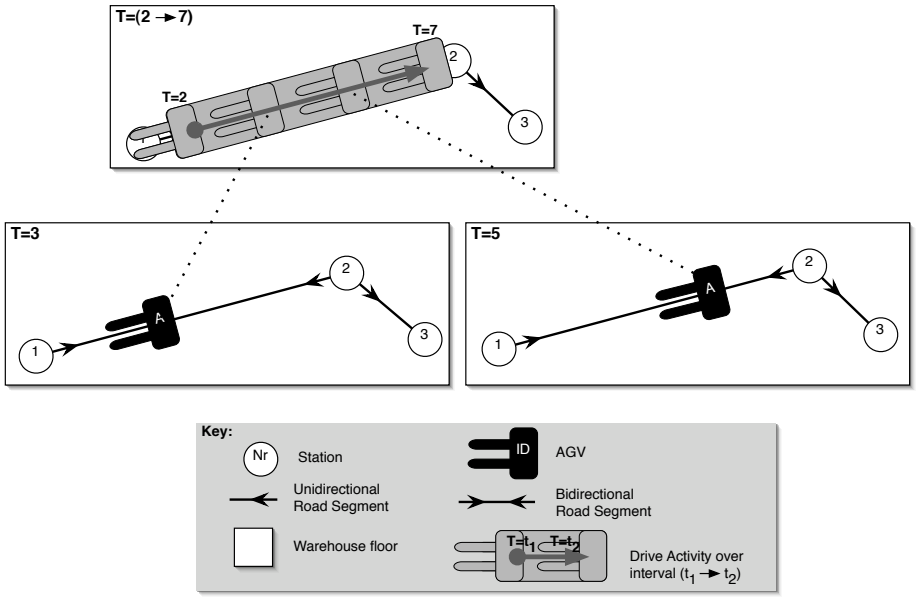

Fig. 3. Example of a drive activity in the simulated warehouse environment.

4 Testing Collision Avoidance

It is crucial to test whether the situated agents handle collision avoidance as required. This requires testing whether the AGVs maintain a minimal safety distance at all times, in the presence of unreliable communication (see Sect. 2.4). In Sect. 4.1, we describe an example scenario performed by a number of agents in a particular warehouse environment. In Sect. 4.2, we focus on how a collision can be detected in an automated way.

4.1 An Example Scenario

In Fig. 4, a fragment of an example layout of a warehouse is depicted. It consists of 10 stations and 12 road segments. Time is expressed in seconds. Initially, 5 AGVs are positioned as depicted on the upper part of of Fig. 4:

- `AGV A` is positioned at `station 10`.
- `AGV B` is positioned at `station 6`.

– `AGV C` is positioned at `station 2`.
– `AGV D` is positioned at `station 8`.
– `AGV E` is positioned at `station 4`.

The quality-of-service attributes of the communication are set to the worst-case of 40 percent message loss and 2 seconds transmission delay.

Starting from this initial setup, the agents of the AGVs are embedded in the simulated warehouse environment and activated. During 12 seconds, i.e. over time interval $(0 \rightarrow 12)$, agents are allowed to perform a number of actions. The actions performed by the agents result in activities depicted on the lower part of Fig. 4:

– At time $T = 2$, the agent of `AGV A` invokes a `move` action to drive from `station 10` to `station 9`. This results in a `drive activity` over time interval $(2 \rightarrow 10)$, representing the movement `AGV A` as a result of the action.
– At time $T = 4$, the agent of `AGV B` invokes a `move` action to drive from `station 6` to `station 1`. This results in a `drive activity` over time interval $(4 \rightarrow 9)$, representing the movement `AGV B` as a result of the action.
– At time $T = 7$, the agent of `AGV C` invokes a `move` action to drive from `station 2` to `station 3`. This results in a `drive activity` over time interval $(7 \rightarrow 12)$, representing the movement `AGV C` as a result of the action.
– At time $T = 6$, the agent of `AGV D` invokes a `move` action to drive from `station 8` to `station 7`. This results in a `drive activity` over time interval $(6 \rightarrow 9)$, representing the movement `AGV D` as a result of the action.
– At time $T = 2$, the agent of `AGV E` invokes a `move` action to drive from `station 4` to `station 5`. This results in a `drive activity` over time interval $(2 \rightarrow 5)$, representing the movement `AGV E` as a result of the action.

We now focus on detecting whether the minimal safety distance is violated in this scenario.

4.2 The Collision Detection Law

A *collision detection law* is a rule that checks whether drive activities proceed safely. For each drive activity, the collision detection law is able to detect two kinds of interference: violations of the safety distance by entities that are stationary, and violations of the safety distance by entities that are non-stationary.

Entities are stationary over a particular time interval in case they are not involved in any drive activity during that time interval. Otherwise, the entity is non-stationary over that time interval. Obstacles are always stationary. AGVs are stationary during the time intervals they are *not* involved in a drive activity.

A code-fragment of the collision detection law is given in Fig. 5, and will be explained next.

Detecting Interference with Stationary Entities We first focus on the case of detecting for a particular drive activity whether it violates the minimal safety distance with respect to entities that are stationary during the time interval of that activity. The *activity perimeter* is central in checking interference. The activity perimeter represents the

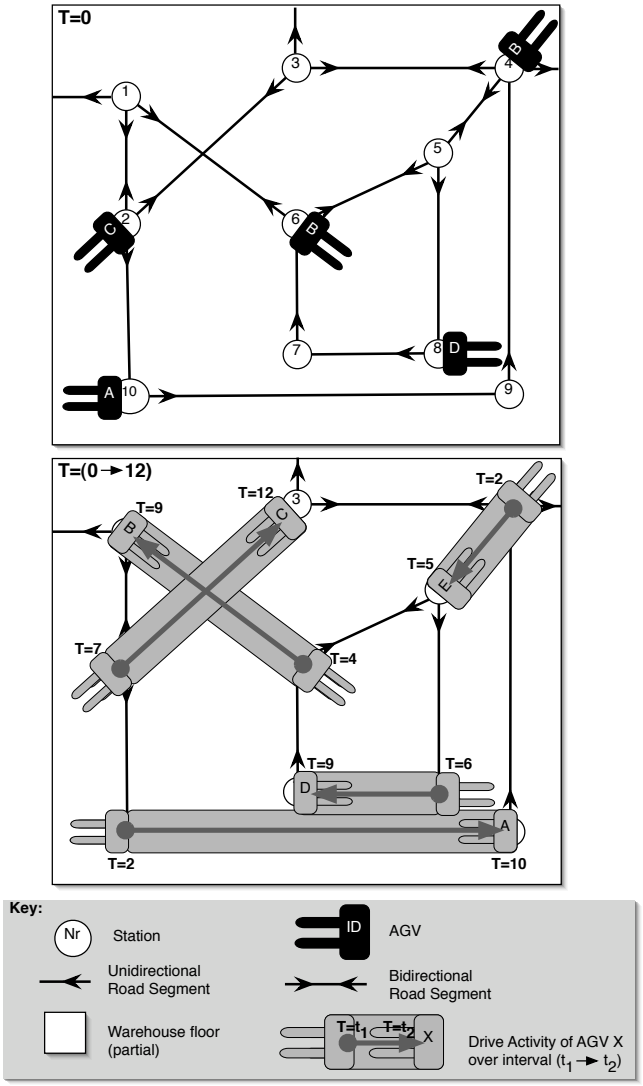

Fig. 4. An example scenario of five AGVs moving through a warehouse.

safety distance around the aggregate of all intermediate positions of an AGV during a drive activity as a whole. As an example, consider the drive activity of `AGV E` in Fig. 4. The activity perimeter for this drive activity is depicted in Fig. 6.

A necessary condition for a particular drive activity to be safe, is that all entities that are stationary over the time interval of that activity, have no overlap with the activity perimeter. This condition is checked in lines 15 to 20 in Fig. 5.

```
1   /**
2    * Check the collisions of a given drive activity
3    * @param act the drive activity
4    * @param entities the entities to check collisions with
5    * @return a vector containing the collisions
6    */
7   public Vector checkCollisions(DriveActivity act, Vector entities){
8      Vector result = new Vector();
9      TimeInterval interval = act.getTimeInterval();
10     BoundingBox perimeter = act.getActivityPerimeter();
11     //a loop to check each entity
12     for (int i=0; i < entities.size(); i++)
13     {
14        Entity ent = (Entity)entities.get(i);
15        if (ent.isStationaryDuring(interval))
16        {  //in case the entity is stationary:
17           //do one check for the perimeter of the whole activity
18           result.add(checkOverlap(perimeter,
19                 ent.getBoundingBox(interval.getBegin())));
20        }
21        else //in case the entitiy is non-stationary
22        {
23           //get all activities of the entity that happen during the interval
24           Vector activities = ent.getDriveActivities(interval);
25           //a loop for each activity of the entity
26           for(int j=0; j < activities.size(); j++)
27           {
28              DriveActivity otherAct = (DriveActivity)activities.get(i);
29              //test whether the activity perimeters of both activities overlap
30              if(checkOverlap(perimeter,otheract.getActivityPerimeter())!=null)
31              {  //in case the activity perimeters overlap
32                 //take snapshots in the common interval of both activities
33                 TimeInterval common = otherAct.getTimeInterval().getIntersection(interval);
34                 for(Time t=common.getBegin(); t.before(common.getEnd()); t.increment())
35                    //do the check for one snapshot
36                    result.add(checkOverlap(act.getEntityPerimeterAt(t),
37                          otherAct.getEntityBoundingBoxAt(t)));
38              }
39           }
40        }
41     }
42     return result;
43  }
```

Fig. 5. Code fragment of the collision detection law.

The collision detection law will detect whether stationary entities violate the activity perimeter. When considering the drive activity of `AGV E`, it is clear from Fig. 4 that `AGV C` and `AGV D` are the only stationary entities during the time interval $(2 \rightarrow 5)$. As `AGV C` and `AGV D` are both entirely outside the activity perimeter of the drive activity of `AGV E`, stationary entities do not compromise the safety of `AGV E`.

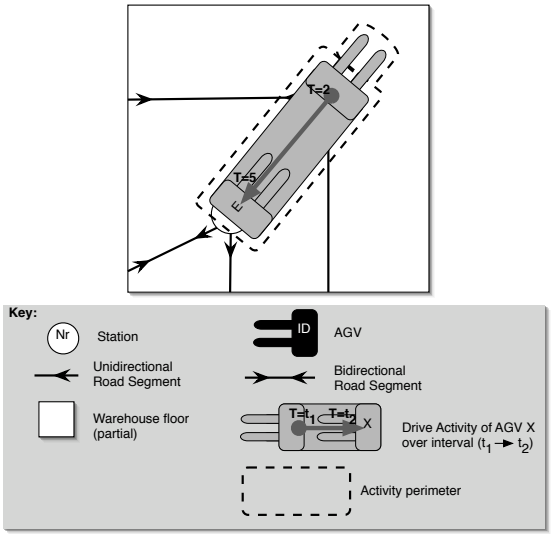

Fig. 6. The activity perimeter of the drive activity of `AGV E`.

Detecting Interference with Non-Stationary Entities Until now, we only considered entities that are stationary during a drive activity. We now focus on the case of detecting for a particular drive activity whether it violates the minimal safety distance with respect to entities that are non-stationary during the time interval of that activity. All other AGVs involved in drive activities during the time interval of the drive activity under investigation, are non-stationary.

Checking the non-stationary entities is performed in lines 21 to 37 of Fig. 5.

As an example, consider the drive activity of `AGV B` in Fig. 4. The activity perimeter to detect interference with stationary entities is depicted on the left hand side of Fig. 7. Note that no interference with stationary entities is detected, as no AGV is stationary during time interval $(4 \rightarrow 9)$.

To determine the safety of a particular drive activity, a detailed investigation of all non-stationary entities that cross the activity perimeter is needed. In the example of Fig. 7, the activity perimeter of `AGV B` overlaps with the drive activity of `AGV C`. In Fig. 5, this is checked in line 30. To determine the interference of a particular drive activity with another drive activity, we only consider the common time interval between both. In Fig. 5, the common time interval is determined in line 33. In our example of the drive activities of `AGV B` and `AGV C`, the common time interval is $(7 \rightarrow 9)$. This is depicted on the right hand side of Fig. 7.

Detecting interference between two drive activities over a common time interval is done by taking a number of state snapshots, see Fig. 8. For each state snapshot, it is checked wither the other AGVs is completely outside the safety perimeter of the former AGV. In case of Fig. 8 this is always the case, so `AGV C` does not compromise the

safety of the drive activity of AGV B. In Fig. 5, the snapshots over the common time interval are checked in lines 34 to 37.

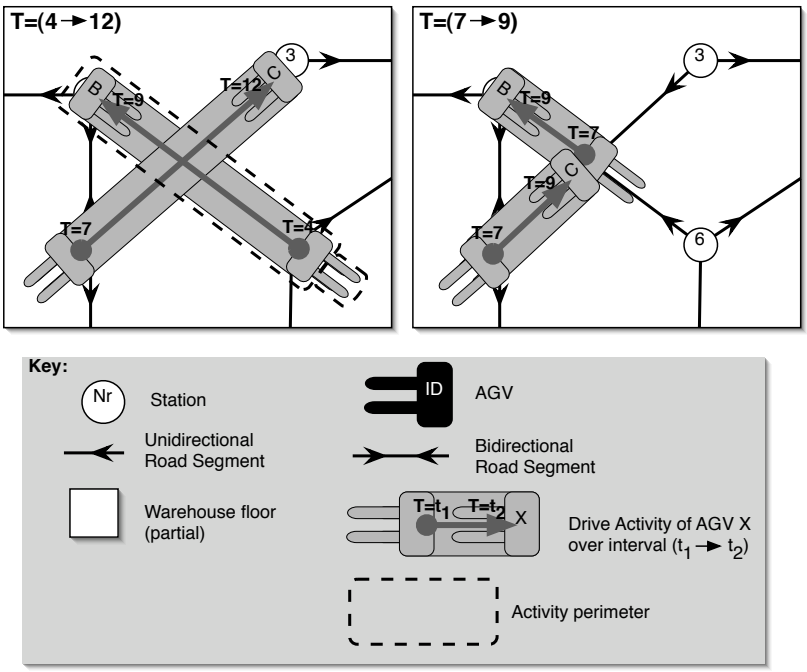

Fig. 7. The drive activity of AGV B. The activity perimeter is depicted on the left part hand side, the common time interval with the drive activity of AGV C is depicted on the right hand side.

A Collision Detection Example. Finally, consider the drive activity of AGV D in Fig. 4. The activity perimeter is depicted in Fig. 9.

- *Interference with stationary entities*. From the activity interference perimeter it is clear that the drive activity of AGV D does not interfere with any entities that are stationary.
- *Interference with non-stationary entities*. The drive activity of AGV A crosses the activity perimeter. Figure 10 illustrates the analysis of the drive activities of AGV D and AGV A over their common time interval $(6 \rightarrow 9)$. In Fig. 10 it is also illustrated how at time $T = 8$, AGV A violates the safety perimeter of AGV D.

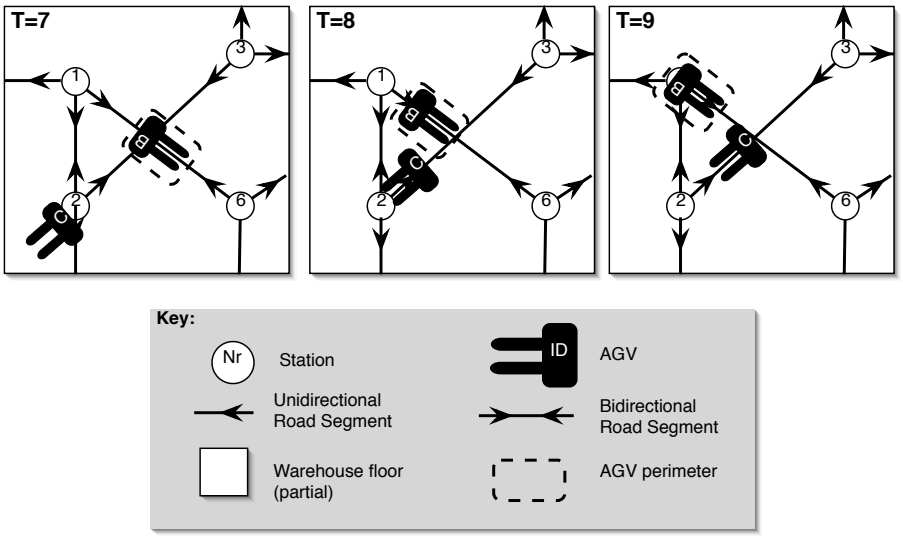

Fig. 8. Checking the drive activity of `AGV B`. The check for non-stationary entities over time interval $(7 \rightarrow 9)$, on the right of Fig. 7, is analyzed using three state snapshots at $T = 7$, $T = 8$ and $T = 9$.

5 Discussion and Evaluation

We elaborate on two important characteristics of the simulated warehouse environment we developed: modularity and performance.

5.1 Modularity

Modularity is applied extensively throughout the model of the simulated warehouse environment. Modularity is crucial as it allows *separation of concerns*, a ground rule for decent software engineering. At the highest level of abstraction, the simulated warehouse environment is decomposed in three modular parts: (1) a representation of the *state* of the simulated warehouse environment, (2) a representation of the *dynamism* in the simulated warehouse environment and (3) a representation of *detection laws* describing rules to detect when the consistency is broken in the simulated warehouse environment.

At a lower level of abstraction, each of the three modular parts is itself designed in a modular way.

Modularity of State. The state of the simulated warehouse environment is designed in a modular way. A distinction is made between environmental entities and environmental relations. For example, spatial relations are easy to manage as they are not scattered throughout the state of environmental entities. As spatial relations are modeled

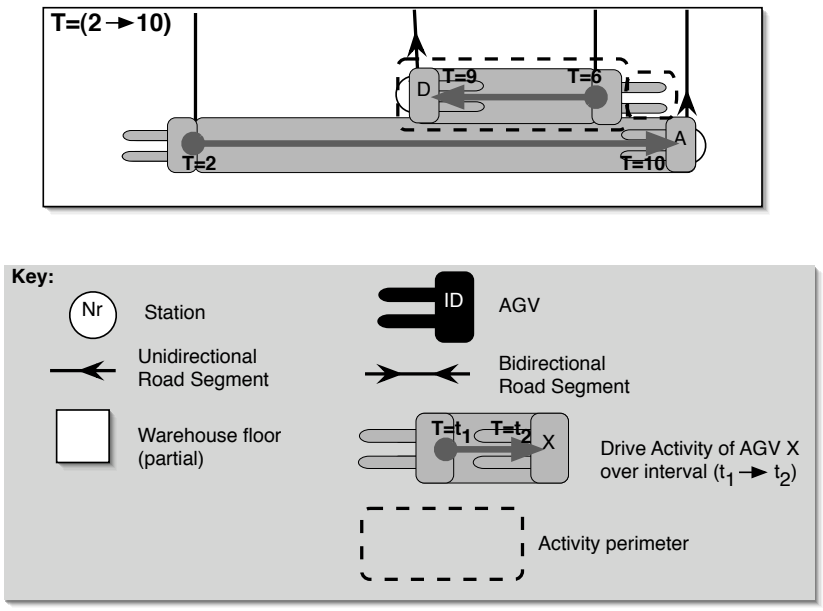

Fig. 9. The activity perimeter of the drive activity of `AGV D`.

separately, their representation can evolve without affecting the representation of the environmental entities.

Modularity of Dynamism. Dynamism is designed in a modular way, clearly separated from the state. Activities encapsulate all characteristics of a particular kind of dynamism happening in the warehouse environment. The characteristics of various activities can be adjusted in a modular way, to suit the characteristics in the real warehouse environment. For example, the acceleration and deceleration characteristics of an individual AGV can be adjusted in the evolution strategy, to accurately reflect the performance of the real AGVs.

Modularity of Detection. The detection laws that check for inconsistencies happening in the simulated environment are developed in a modular way. Detection laws avoid the use of a uniform, global granularity that crosscuts the whole simulation. Instead, each detection law employs its own granularity, customized according to the required accuracy to detect particular inconsistencies. For example, detecting collisions can be done by a collision detection law that uses snapshots with a granularity of 1 second to check interference with non-stationary entities. Obtaining a higher accuracy of detection involves an adjustment applied locally in the collision detection law, e.g. a change in the granularity from 1 second to 5 milliseconds.

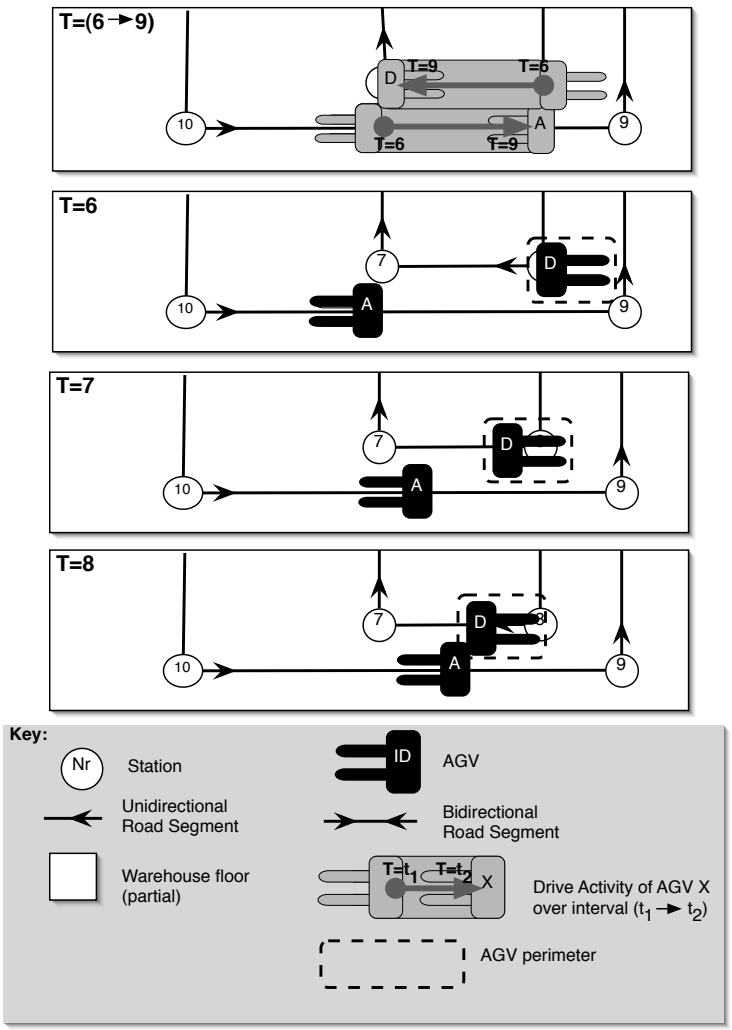

Fig. 10. The drive activity of `AGV D` and the drive activity of `AGV A` over the common time interval $(6 \rightarrow 9)$. The interference of both activities is detected using state snapshots at $T = 6$, $T = 7$ and $T = 8$, at which the safety perimeter of `AGV D` is violated.

5.2 Performance of Collision Detection

We now elaborate on the performance of collision detection based on our approach. We compare the performance of collision detection using a collision detection law (see Sect. 4.2) with the performance of detecting collisions using a global time step. As a performance measure, we employ the number of perimeter checks to detect violations of the safety perimeter.

We compare both approaches using the scenario of Fig. 4. Suppose the required accuracy to detect perimeter violations is 1 centimeter, and that the maximum velocity of an AGV is 2 meter per second.

Collision Detection Using a Global Time Step. In this approach, collision detection happens by evolving the simulation according to a common, system-wide time step. We first determine the step size to check for perimeter violations with the required accuracy. Driving at its maximum speed of 2 meters per second, it takes an AGV 5 milliseconds to move over 1 centimeter. As two AGVs can travel at top speed, their relative position changes at a maximum rate of 4 meters per second. Consequently, to detect collisions with an accuracy of 1 centimeter, a perimeter check must happen at least every 2.5 milliseconds. This means 400 perimeter checks are needed to check violations of the safety perimeter of an AGV driving during 1 second.

We now determine the number of perimeter checks for the scenario of Fig. 4:

- `AGV A`: to check the `drive activity` over time interval $(2 \rightarrow 10)$, $400 \times 8 = 3200$ checks are needed with *each* of the other four AGVs. This results in $3200 \times 4 = 12800$ checks.
- `AGV B`: to check the `drive activity` over time interval $(4 \rightarrow 9)$, $400 \times 5 = 2000$ checks are needed with each of the other four AGVs. This results in $2000 \times 4 = 8000$ checks.
- `AGV C`: to check the `drive activity` over time interval $(7 \rightarrow 12)$, $400 \times 5 = 2000$ checks are needed with each of the other four AGVs. This results in $2000 \times 4 = 8000$ checks.
- `AGV D`: to check the `drive activity` over time interval $(6 \rightarrow 9)$, $400 \times 3 = 1200$ checks are needed with each of the other four AGVs. This results in $1200 \times 4 = 4800$ checks.
- `AGV E`: to check the `drive activity` over time interval $(2 \rightarrow 5)$, $400 \times 3 = 1200$ checks are needed with each of the other four AGVs. This results in $1200 \times 4 = 4800$ checks.

This means a total of 38400 perimeter checks are needed for the scenario.

Collision Detection Using a Collision Detection Law. We now focus on the number of perimeter checks using the collision detection law that inspects drive activities. In analogy with the previous approach, 400 perimeter checks are needed to check violations of the safety perimeter of a single AGV driving during 1 second.

We now determine the number of perimeter checks for in case of Fig. 4. The number of checks needed for each activity is as follows:

- `AGV A`: checking the `drive activity` over time interval $(2 \rightarrow 10)$. There are no stationary entities during this time interval: all four other AGVs are non-stationary. Consequently, the activity perimeter check of line 30 in Fig. 5 is performed 4 times. However, only the activity perimeter of the drive activity of `AGV D` actually overlaps with the one of `AGV A`. As the common time interval of

both activities is $(6 \rightarrow 9)$, the check in the loop at line 36–37 in Fig. 5 is executed $400 \times 3 = 1200$ times. The total number of perimeter checks needed is $4 + 1200 = 1204$.

- `AGV B`: checking the `drive activity` over time interval $(4 \rightarrow 9)$. There are no stationary entities during this time interval: all four other AGVs are non-stationary. Consequently, the activity perimeter check of line 30 in Fig. 5 is performed 4 times. However, only the activity perimeter of the drive activity of `AGV C` actually overlaps with the one of `AGV B`. As the common time interval of both activities is $(7 \rightarrow 9)$, the check in the loop at line 36–37 in Fig. 5 is executed $400 \times 2 = 800$ times. The total number of perimeter checks needed is $4 + 800 = 804$.
- `AGV C`: checking the `drive activity` over time interval $(7 \rightarrow 12)$. `AGV E` is stationary during this time interval, resulting in one check performed at line 18–19 in Fig. 5. All three other AGVs are non-stationary. Consequently, the activity perimeter check of line 30 in Fig. 5 is performed 3 times. However, only the activity perimeter of the drive activity of `AGV B` actually overlaps with the one of `AGV C`. As the common time interval of both activities is $(7 \rightarrow 9)$, the check in the loop at line 36–37 in Fig. 5 is executed $400 \times 2 = 800$ times. The total number of perimeter checks needed is $1 + 3 + 800 = 804$.
- `AGV D`: checking the `drive activity` over time interval $(6 \rightarrow 9)$. `AGV C` and `AGV E` are stationary during this time interval, resulting in 2 checks performed at line 18–19 in Fig. 5. The two other AGVs are non-stationary. Consequently, the activity perimeter check of line 30 in Fig. 5 is performed 2 times. However, only the activity perimeter of the drive activity of `AGV A` actually overlaps with the one of `AGV D`. As the common time interval of both activities is $(6 \rightarrow 9)$, the check in the loop at line 36–37 in Fig. 5 is executed $400 \times 3 = 1200$ times. The total number of perimeter checks needed is $2 + 2 + 1200 = 1204$.
- `AGV E`: checking the `drive activity` over time interval $(2 \rightarrow 5)$. `AGV C` and `AGV D` are stationary during this time interval, resulting in 2 checks performed at line 18–19 in Fig. 5. The two other AGVs are non-stationary. Consequently, the activity perimeter check of line 30 in Fig. 5 is performed 2 times. However, none of the activity perimeters of the activities of `AGV A` and `AGV B` overlap with the one of `AGV E`. Consequently, the loop at line 34–37 in Fig. 5 is not executed. The total number of perimeter checks needed is $2 + 2 = 4$.

This means a total of 4020 perimeter checks are needed for the scenario, which is only about 10% of the checks needed in the previous approach.

The number of checks needed by the collision detection law is highly dependent upon the density of AGVs in the warehouse environment. The layout fragment used in Fig. 4 is kept small for demonstration purposes, and hence the density of AGVs is high. The complete layout of a warehouse is much more expanded, and has a lower density of AGVs. For example, for 5 AGVs the number of stations and road segments in an industrial layout typically ranges from 50 to 500, instead of the 10 or 12 in our example. In a layout with a lower density of AGVs, it will be likely that the collision detection law needs to take into account non-stationary entities less often, reducing the number of perimeter checks. For the approach of detecting collisions based on a global

time-step, the number of perimeter checks always remains the same, irrespective of the density of AGVs.

6 Conclusion

In this paper, we described a simulated warehouse environment that can be used to test AGV control software. To support testing, the simulated warehouse environment is decomposed in three parts, each with their own responsibility:

- The *state* is responsible to represent snapshots of the warehouse environment at a particular moment in time.
- The model of *dynamism* represents in an explicit manner the evolution of the simulated warehouse environment over time.
- *Detection laws* are responsible to detect for the occurrence of conflicts.

We illustrated the use of the simulated warehouse environment to test collision avoidance. The approach employed a collision detection law that relies on inspection of drive activities to detect whether the minimal safety distance is maintained at all times.

We refer to [15] for further information on the AGV-simulator that was developed and supports the approach described in this paper.

References

1. Egemin International NV: (http://www.egemin.com/) Home page of Egemin International NV. Date of publication: 2002. Date retrieved: December 1, 2005. Date last modified: 2005.
2. Weyns, D., Schelfthout, K., Holvoet, T., Lefever, T., Wielemans, J.: Architecture-centric development of an AGV transportation system. In: Multi-Agent Systems and Applications IV. Volume 3690 of Lecture Notes in Computer Science., Springer Verlag Berlin Heidelberg New York (2005) 640–645
3. Weyns, D., Schelfthout, K., Holvoet, T., Lefever, T.: Decentralized control of E'GV transportation systems. In: Autonomous Agents and Multiagent Systems, Industry Track, University of Utrecht, ACM (2005) 67–74
4. Weyns, D., Holvoet, T.: A formal model for situated multi-agent systems. Fundamenta Informaticae **63** (2004) 125–158
5. Wegner, P.: Why Interaction is More Powerful than Algorithms. Communications of the ACM **40** (1997) 80–91
6. De Wolf, T., Samaey, G., Holvoet, T.: Engineering self-organising emergent systems with simulation-based scientific analysis. In: Proceedings of the Fourth International Workshop on Engineering Self-Organising Applications, Universiteit Utrecht (2005) 146–160
7. Uhrmacher, A.: Simulation for agent-oriented software engineering. In Lunceford, W., Page, E., eds.: First International Conference on Grand Challenges for Modeling and Simulation, SCS, San Diego (2002)
8. Uhrmacher, A.M., Kullick, B.G.: "Plug and test": software agents in virtual environments. In: WSC '00: Proceedings of the 32nd conference on Winter simulation, San Diego, CA, USA, Society for Computer Simulation International (2000) 1722–1729
9. Russell, S., Norvig, P.: Artificial Intelligence: A Modern Approach. Prentice-Hall, Englewood Cliffs, NJ (1995)

10. Klügl, F., Fehler, M., Herrler, R.: About the role of the environment in multi-agent simulations. In: Environments for multi-agent systems. Volume 3374 of Lecture Notes in Computer Science., Springer-Verlag (2005) 127–149
11. Helleboogh, A., Holvoet, T., Weyns, D., Berbers, Y.: Extending time management support for multi-agent systems. In: Multi-Agent and Multi-Agent-Based Simulation: Joint Workshop MABS 2004, New York, NY, USA, July 19, 2004, Revised Selected Papers. Volume 3415 / 2005 of Lecture Notes in Computer Science., Springer-Verlag, GmbH (2005) 37–48
12. Carson, J.S.: Introduction to simulation: introduction to modeling and simulation. In: Winter Simulation Conference. (2003) 7–13
13. Bandini, S., Manzoni, S., Simone, C.: Dealing with space in multi–agent systems: a model for situated mas. In: AAMAS '02: Proceedings of the first international joint conference on Autonomous agents and multiagent systems, New York, NY, USA, ACM Press (2002) 1183–1190
14. Helleboogh, A., Holvoet, T., Berbers, Y.: Simulating actions in dynamic environments. In Barros, F., Bruzzone, A., Frydman, C., Giambiasi, N., eds.: Conceptual Modeling and Simulation Conference, LSIS, Université Paul Cézanne Aix Marseille III (2005) 123–129
15. AgentWise Taskforce, KULeuven: (http://www.cs.kuleuven.ac.be/∼distrinet/taskforces/agentwise/agvsimulator/) Home page of the AGV Simulator. Date of publication: 2005. Date retrieved: December 1, 2005. Date last modified: 2005.