AATOM - An Agent-based Airport Terminal Operations Model

Stef Janssen, Anne-Nynke Blok, Arthur Knol

March 19, 2018

Contents

Introduction

In this work, *AATOM*, a microscopic agent-based model that simulates movement and operations in the airport terminal is presented. Specifically, the model includes the main handling processes required for outbound passengers namely, check-in, security and border control. Furthermore, basic facilities for discretionary activities are modelled namely, bathrooms, restaurants and shops. The model has an accompanying architecture, the AATOM architecture that is described in this work as well. The objective of the model is to serve as a basis for several studies in the area of airport terminal operations. It will be used to investigate properties in the fields of security, efficiency, resilience and possibly other fields like safety, and it will be used to investigate the relationship between any of these areas.

AATOM is developed following an agent-based modelling approach. First, the language and architecture of this work are introduced in Chapter 1. Then, a baseline model is introduced in Chapter 2. The environment of the model is discussed in Section 2.1. The agents are the entities that function autonomously in this environment, and they are discussed in Section 2.2. Interactions between the different aspects of the model are described in Section 2.3. Finally, the input parameters necessary for this model are described in Chapter 3 and a list of assumptions is presented in Chapter 4.

Chapter 1

Language & Architecture

This chapter describes the modelling language and architecture of AATOM. The order-sorted predicate logic-based language called LEADSTO is used [2]. This language allows both discrete and continuous modelling of a system at different aggregation levels. Furthermore, one can express both qualitative and quantitative aspects of a system using LEADSTO. The LEADSTO language is outlined in Section 1.1. The AATOM architecture specifies different modules that are responsible for the functioning of the human agents. These are explained in more detail in Section 1.2.

1.1 The LEADSTO Language

Dynamics in LEADSTO are represented as evolution of states over time. A state is characterized by a set of properties that do or do not hold at a certain point in time. To specify state properties for system components, ontologies are used that are defined by a number of sorts, sorted constants, variables, functions and predicates (i.e., a signature). For every system component A, a number of ontologies can be distinguished: the ontologies $IntOnt(A)$, $InOnt(A)$, $OutOnt(A)$, and $ExtOnt(A)$ are used to express respectively internal, input, output and external state properties of the component A. For a given ontology Ont, the propositional language signature consisting of all state ground atoms based on Ont is denoted by $APROP(Ont)$. State properties are specified based on such ontology by propositions. Propositions are formed, combining ground atoms by logical operators such as conjunction, negation, disjunction, and implication. Input ontologies contain elements for describing perceptions of an agent from the external world, such as the observed function *obs*: $IntOnt(A) \rightarrow APROP(IntOnt(A))$. Output ontologies describe actions and communications of agents. To this end, the function *performed*: $ACTION \rightarrow APROP(OutOnt(A))$ is introduced. Then, a state S is an indication of which atomic state properties are true and which are false: S: $APROP(Ont) \rightarrow \{true, false\}$.

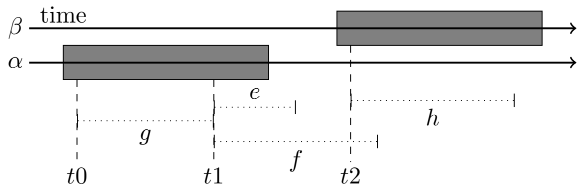

Figure 1.1: Timing relationships for LEADSTO expressions.

LEADSTO enables modeling of direct temporal dependencies between two state properties in successive states, also called dynamic properties. A specification of dynamic properties in LEADSTO is executable and can be depicted graphically. The format is defined as follows. Let $\alpha 1$ and $\alpha 2$ be state properties of the form conjunction of atoms or negations of atoms, and e, f, g, h non-negative real numbers. In the LEADSTO language the notation $\alpha 1 \twoheadrightarrow_{e,f,g,h} \alpha 2$ means: if state property $\alpha 1$ holds for a certain time interval with duration g, then after some delay (between e and f) state property $\alpha 2$ will hold for a certain time interval of length h (Fig. 1.1). To indicate the type of a state property in a LEADSTO property we shall use prefixes *internal*(c), *input*(c), *output*(c) and *external*(c), where c is the name of a component. Consider an example dynamic property:

$$\begin{gathered} \mathit{input}(\mathbf{A})|\mathit{obs}(\mathit{arrest\_fail}) \twoheadrightarrow_{0,0,1,1} \\ \mathit{output}(\mathbf{A})|\mathit{performed}(\mathit{detonate}()) \end{gathered}$$

Informally, this example expresses that if agent A observes a failed arrest during some time unit, then A will detonate an IED in the following time unit. Next, a *trace* or *trajectory* γ over a state ontology *Ont* is a time-indexed sequence of states over *Ont* (where the time frame is formalized by real numbers). A LEADSTO expression $\alpha 1 \twoheadrightarrow_{e,f,g,h} \alpha 2$, holds for a trace γ if:

$$\begin{aligned} &\forall t1[\forall t[t1 - g \leq t < t1 \Rightarrow \alpha 1 \text{ holds in } \gamma \text{ at time } t] \\ &\Rightarrow \exists d[e \leq d \leq f \& \forall t'[t1 + d \leq t' \leq t1 + d + h \\ &\Rightarrow \alpha 2 \text{ holds in } \gamma \text{ at time } t']] \end{aligned}$$

More details on the semantics of the LEADSTO language can be found in [2].

1.2 AATOM Architecture

As human agents are central in an airport terminal, an architecture used to specify agents is specified in this work. The AATOM architecture is structured in three layers, based on the work of Blumberg [1], Hoogendoorn [6] and Reynolds [7]. In this architecture three levels of abstraction are distinguished namely, the operational layer, the tactical layer and the strategical layer. An overview of the architecture is presented in Figure 1.2.

The top layer, called the *strategic layer*, is responsible for determination of goals, for updating the belief module and for reasoning. Reasoning includes analysis, planning and decision-making. The middle layer, called the *tactical layer*, is responsible for actuation of activities. It is also responsible for route navigation in the environment, the interpretation of observations and maintaining a lower level belief. The belief modules in the strategic and tactical layer exchange information with each other. The *operational layer* is responsible for interaction with the environment and other agents. Here, input is generated by sensory information an agent can observe. The output is defined as actions an agent can execute. The actions are generated based on input from the tactical layer.

Each of the different layers contain several modules that are responsible for specific tasks. In the architecture, the following main sequence of operations is followed: *observation* $\rightarrow$ *perception* $\rightarrow$ *interpretation* $\rightarrow$ *reasoning* $\rightarrow$ *activity control* $\rightarrow$ *actuation* $\rightarrow$ *action*. This reflects many of the processes discussed by Sun in his extensive analysis on cognitive architectures [8]. The different layers and modules are explained in the remainder of this chapter.

1.2.1 Operational Layer

The operational layer is the lowest layer of the architecture and is responsible for low level interactions with the environment and other agents. It contains two modules: the perception module and the actuation module. Both modules are discussed in more detail below.

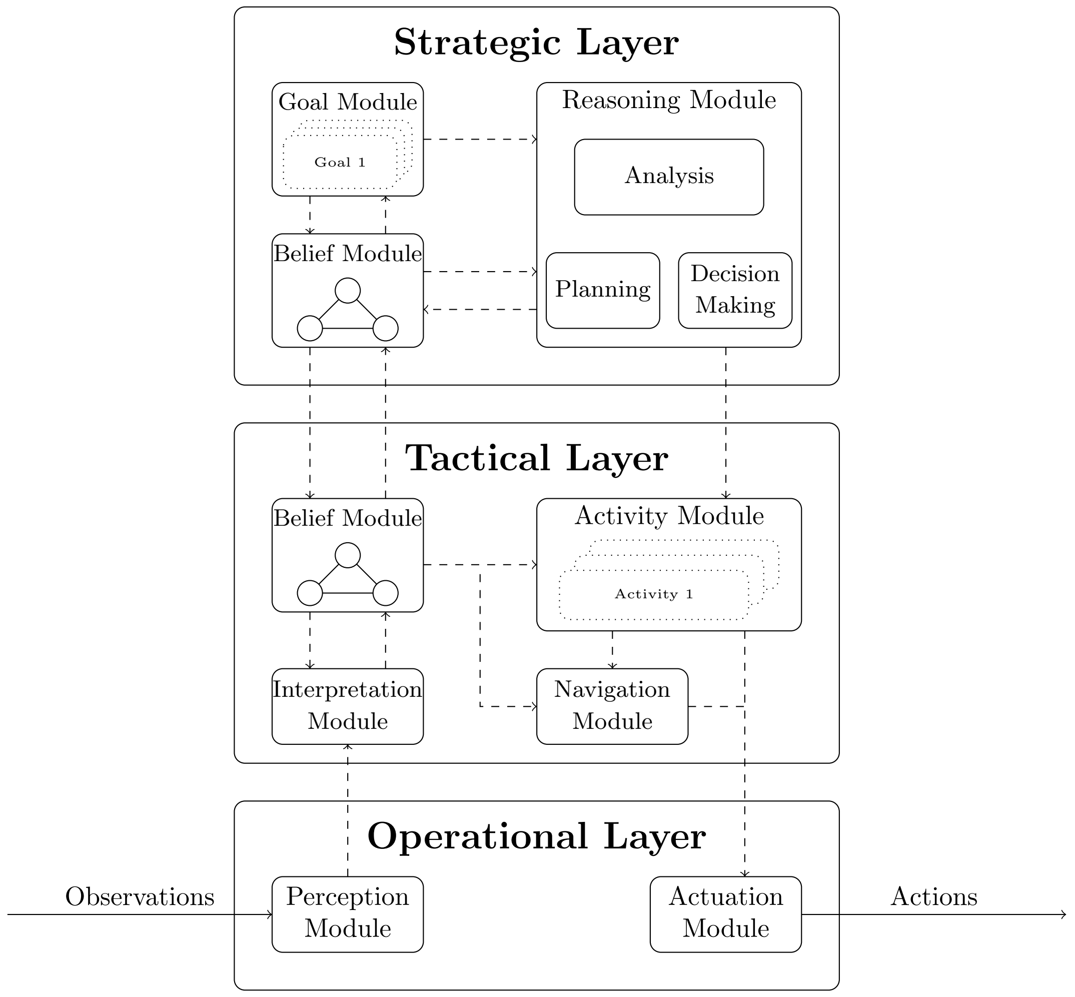

Figure 1.2: The AATOM architecture.

Perception Module

The perception module is responsible for perceiving information from the environment or other agents. Observations of this module are further processed by the interpretation and belief module, after which other modules can access the observations. Observations for all agents are described using the following function.

$$input(\mathbf{A})|obs(\mathbf{observation\_object})$$

Where *observation_object* represents the object that an agent observed, and *obs* is short for *observed*. The function maps specific *observation_object*s to a Boolean value that indicates that the specific object has been observed or not. The observations that the agents can perform are specific to agent types and are discussed in the baseline model description in Chapter 2.

Actuation Module

The actuation module is responsible performing actions of an agent in the environment. It receives instructions from the activity module and the navigation module in the tactical layer. Within the actuation module, for instance the walking behaviour of human agents is modelled. Other actions, like communication with other agents, are also executed in the actuation module. Actions can be in any of the following three states: *not_started*, *in_progress* and *has_finished*, denoted as follows.

$$output(\mathbf{A})|action\_state(\mathbf{action\_type()})$$

Where *action_type*() represents the action that an agent can perform. Actions that have been performed (i.e. are in the *has_finished* state) can also be indicated using the following function.

$$output(\mathbf{A})|performed(\mathbf{action\_type()})$$

Where $performed$ states that the action has been performed. The function maps specific actions to a Boolean value that indicates that the specific action has been performed or not. The actions that the agents can perform are specific to agent types.

All human agents can communicate using a communication action. A communication is directed to another agent, has a specific type, *comm_type*, and a related parameter, *comm_parameter*. This is denoted as follows.

$$communicate(\mathbf{agent}, \mathbf{comm\_type}, \mathbf{comm\_parameter})$$

1.2.2 Tactical Layer

The tactical layer is responsible for the interpretation of observations which will update the belief of the agent. Furthermore, the tactical layer is responsible for preparation of actions by executing activities. The tactical layer also determines collision-free routes to target locations. The interpretation module, activity module and navigation module are described in detail below.

Interpretation Module

The interpretation module is responsible for the interpretation of observations made by the agent. It receives its input from the perception module and in cooperation with the belief module an interpretation of the observation is generated. When the interpretation is made the belief module is updated. An interpretation in this work follows the structure described below.

$$internal(\mathbf{A})|int(\mathbf{interpretation\_object})$$

Where *interpretation_object* is the type of interpretation that an agent has made, and *int* is short for *interpreted*. Often, the interpretation module is just an intermediate module that transfers information from the perception module to the belief module. In other cases, lower level observations and information in the belief module are combined into more advanced interpretations. An example of such an interpretation is the *stuck* interpretation, in which an agent determines if it is stuck in a location or not.

Belief Module

The belief module is responsible for maintaining the belief of the agent. Belief is represented as a 4-tuple, outlined below.

$$\text{Belief} = \begin{pmatrix} \text{Characteristics} \\ \text{Interpretations} \\ \text{Activity \& Action State} \\ \text{Plan} \end{pmatrix}$$

Characteristics of an agent are saved in the belief module. The interpretations refer to the interpretations the agent made. When an agent interprets something different from its current belief, the belief is updated. The activity & action state is a vector of activity-state pairs and action-state pairs, containing all activities and actions the agent can execute and their corresponding states. When the status of any activity or action changes, the activity-state pair or action-state pair is updated. Finally, the plan is updated in the belief module whenever the plan is generated and/or changed in the planning module. The belief module in the tactical layer automatically exchanges its belief with the belief module in the strategic layer.

Other modules present in the architecture, like navigation and reasoning, can access the belief maintained in the belief module such that they can function properly.

Activity Module

The activity module is responsible for executing activities based on input from the reasoning module in the the strategic layer and the belief module in the tactical layer. Activities are defined to be executable by agents and form a central concept in this architecture. An activity has an *activity_area* that corresponds to an area in which the activity is executed. In these areas the corresponding activities can be executed. The activity area is denoted as follows.

$$internal(\mathbf{A})|activity\_area(\mathbf{activity})$$

Similar to actions, activities can be in any of the following three states: *not_started*, *in_progress* and *has_finished*, denoted as follows.

$$internal(\mathbf{A})|activity\_state(\mathbf{activity})$$

The activity state is saved in the agent's belief module and can be accessed at any moment in time.

Navigation Module

Navigation of a human agent is performed in the navigation module. The navigation module determines a collision free path towards some area. A collision free path is defined to be a path that avoids all physical objects that have the blocking property. Other passengers are not defined as physical objects and are not taken into account when calculating the path. A path is represented as a sequence of location points and saved in the *path* variable. The function $end\_location(path)$ returns the last location point in the path.

The navigation requires input from the belief module (see Section 2.2.5) to access observations in the environment and the activity module (see Section 2.2.4) to determine the target area. Algorithms like the A* path finding algorithm [3] and the Jump Point Search algorithm [4] can be used for calculating the path.

1.2.3 Strategic Layer

The strategic layer is responsible for both the determination of goals, reasoning and maintaining and updating a belief.

Belief Module

The belief module in the strategic layer works the same as the belief module in the tactical layer. It exchanges information with the reasoning module and the goal module and exchanges its belief with the belief module in the tactical layer.

Goal Module

The goals module is responsible for the managing of the agent's goals. A goal can be in the following states: *not_completed*, *failed* and *succeeded*, denoted as follows.

$$internal(\mathbf{A})|goal\_state(\mathbf{goal})$$

All goals start in the *not_completed* state. If at any time t all conditions of a goal are met, the goal state changes to *succeeded.* If at any time t the goal conditions cannot be met anymore, the goal states changes to *failed.* The two types of transitions of goal states from the *not_completed* state to the *succeeded* state are specified below. The transitions to the *failed* state follow a similar pattern, but occur when the *succeeded* state cannot be reached anymore.

$$internal(\mathbf{A})|goal\_state(\mathbf{goal}) = \mathbf{not\_completed} \wedge activity\_state(\boldsymbol{some\_activity}) = \boldsymbol{has\_finished}$$
$$\&\ external(\mathbf{A})|\mathbf{t} < \mathbf{t}_{goal} \twoheadrightarrow_{0,0,1,1} internal(\mathbf{A})|goal\_state(\mathbf{goal}) = \mathbf{succeeded}$$

$$internal(\mathbf{A})|goal\_state(\mathbf{goal}) = \mathbf{not\_completed} \wedge activity\_state(\boldsymbol{some\_activity}) = \boldsymbol{has\_finished}$$
$$\wedge\ activity\_state(\boldsymbol{other\_activity}) \neq\ \boldsymbol{has\_finished} \twoheadrightarrow_{0,0,1,1} internal(\mathbf{A})|goal\_state(\mathbf{goal}) = \mathbf{succeeded}$$

These goal conditions of an agent represent that the wants to finish a specific activity before a certain time or before another activity has finished. For instance, a departing passenger wants to finish its gate activity before the time of its flight. A departing passenger also wants to finish its check-in activity before its security activity is finished.

The goals of an agent can depend on the *needs*, *values* and *motives* that are specific for an agent type.

Reasoning Module

The reasoning module is responsible for the reasoning processes of an agent. It contains three modules: planning, analysis and decision making. The analysis module is responsible for the analysis of the current state of the agent. If this module determines the need for an updated plan or a decision, it communicates this to the respective modules.

Planning Module In the planning module, a plan is generated based on the goals of an agent. A plan is an ordered sequence of activities. The planning module aims to define a plan that satisfies as many of the goals of the agent as possible. A plan is generated at the moment there are no more activities left in the plan.

$$internal(\mathbf{A})|empty(\boldsymbol{plan}) \twoheadrightarrow_{0,0,1,1} internal(\mathbf{A})|\neg empty(\boldsymbol{plan})$$

Where the *empty* function is a Boolean function that determines if a sequence contains information or not. While currently no times are related to each element of a plan are present, this can easily be extended in future versions of *AATOM.* The *next_activity* function returns if a specified activity is the next activity in the plan, denoted as follows.

$$internal(\mathbf{A})|next\_activity(\boldsymbol{activity})$$

Decision Making Module The decision making module is used to make decisions that are not covered by the other modules. For instance, while the planning module determines which activities need to be performed at what time, there can still some freedom in choosing the location. An airport can have two security areas and multiple queue areas within a check-in area. The decision making module determines which area is chosen to execute the activity.

Chapter 2

AATOM - Baseline Model

The AATOM baseline model is presented in this chapter. The chapter consists of three parts: environment, agents and interactions. First, the different environment objects are described in Section 2.1. Then, the agents and their characteristics are described in Section 2.2. Finally, the interactions betweeen agents and between agents and the environment are discussed in Section 2.3.

2.1 Environment

The environment of this model is specified to be an airport terminal. In this chapter, the set of environment objects that form the environment is described. Three base environment objects are defined: Area, Flight and Physical Object. Some of these base components have a set of subcomponents, more specific instances of the corresponding base component. A hierarchical visualization of the environment is shown in Figure 2.1. The different components of the environment are specified below.

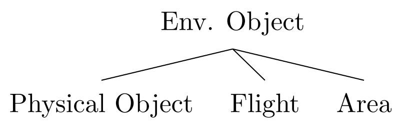

Figure 2.1: The hierarchy of environment objects.

An important concept used throughout the work, is that of a *location point*. A location point is defined to be a 2 dimensional vector that contains a real valued x and y value. Location points are used throughout the model to specify locations of agents and environment objects.

2.1.1 Area

An area is defined to be a shape that is accessible to all (human) agents. An area is a two dimensional polygon that is bounded by a sequence of location points C. Different types of areas can overlap or even completely be contained in other areas. A total of 10 different areas are defined. Each of these different areas form a direct subcomponent of the component area. A Boolean function *in_area* is defined to determine if a location point is in an area.

$$in\_area(\textbf{location\_point}, \textbf{area})$$

Some areas are defined to be within another area. This spatial relation between area objects can be found in Figure 2.2. Each of the different types of areas that are defined are discussed below.

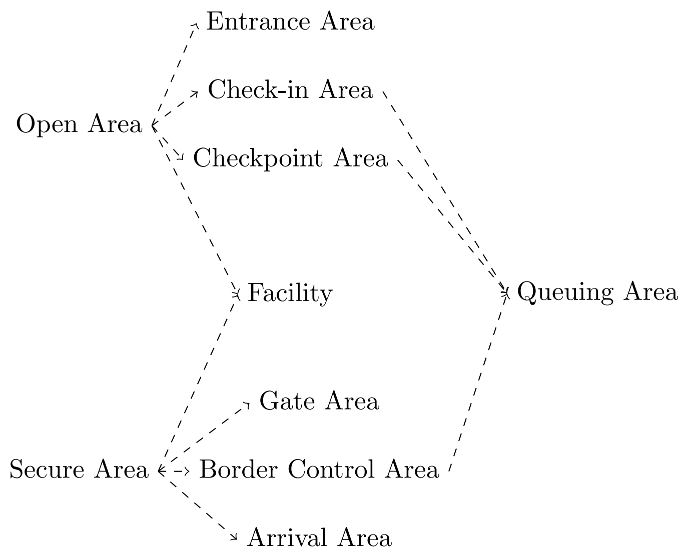

Figure 2.2: The different area objects and their relative locations. An arrow pointing from one area to another implies that the second area is located within the first area. If multiple arrows point to an area, this means that that area is located in any of the areas that have an arrow pointing towards that area.

Open Area

The open area is accessible for all (human) agents. This area is sometimes referred to as the non-sterile area or landside area. The open area contains physical objects and other areas. It contains at least one entrance area and a check-in area. It further contains at least one checkpoint area and can contain facility areas.

Entrance Area

The entrance area is the area in which passengers enter or leave the airport terminal. In this area, passenger agents with a departure flight are generated and passenger agents with an arrival flight are removed. It is a representation of the connection between the outside world and the open area of the airport terminal.

Check-in Area

The check-in area is the area in which the process of checking in takes place. The area contains desks (see Section 2.1.3) and can contain a queue area.

Queue Area

A queue area is a designated area for queuing. It is separated by queue separators and is used as an area in which passengers wait to be served elsewhere. It can be contained in check-in, checkpoint and border control areas.

Facility Areas

There are multiple types of facility areas and they can be located throughout the airport, both in the open and in the secure area. The type of facility areas that are distinguished are shops, restaurants and bathrooms.

Checkpoint Area

The checkpoint area is the area that forms the boundary between the open and secure area. It consists of a queue area and checkpoint lanes.

Checkpoint lanes contain the physical infrastructure to scan passengers. One checkpoint lane consist of an X-Ray sensor that is surrounded by belts that move luggage of passengers. Furthermore, a Walk Through Metal Detector (WTMD) is present at the checkpoint lanes. Finally, an Explosive Trace Detector (ETD) is present as well. Each of these sensors is discussed in more detail in Section 2.1.3.

Secure Area

The secure area (sometimes airside area or sterile area) is only accessible for human agents that have passed the checkpoint area without being stopped by any of the security operators. The secure area contains different areas: border control area, gate area and arrival gate area. These are described in more detail below. The secure area can also contain facility areas.

Border Control Area

A border control area is the area in which border control takes place. The border control area is located in front of a gate area, or after the arrival area. This area consists of desks that are manned by security operators.

Gate Area

A gate area is an area in which passengers will be removed from the airport terminal at the departure time of the flight (see Section 2.1.2). A gate area can contain seats (see Section 2.1.3).

Arrival Area

The arrival area is located in the secure area. It is the area in which passenger with arrival flights are generated based on the arrival time of their flight.

2.1.2 Flight

A flight is defined to be an abstract concept with the following property vector.

$$\text{Property Vector Flight} = \begin{pmatrix} \text{Flight type} & i \\ \\ \text{Time} & t \end{pmatrix}$$

- *Flight type* (i)
  - A flight is either an *arrival* or *departure* flight, represented as a Boolean variable.
- *Time* (t)
  - The time the flight arrives or departs.

Apart from the properties described above, a flight can also exhibit the following relations.

- $has\_gate(flight, gate\_area)$
  - The gate area that is associated with the flight. Only one gate area can be assigned to a flight.
- $has\_desk(flight, desk)$
  - The check-in desks that are associated with the flight. A flight can have any number of desks.
- $is\_flown\_by(flight, passenger)$
  - A flight can be flown by a passenger. A flight can have any number of relations of this type.
- $has\_checked\_in(flight, passenger)$.
  - A flight can have a passenger that is checked in. This relation can only hold if $is\_flown\_by(flight, passenger)$ also holds. A flight can have at most as many relations of this type as the number of relations that exist for $is\_flown\_by(flight, passenger)$ defined above.
- $is\_managed\_by(flight, operator)$.
  - A flight is managed by some check-in operator. A flight can have any number of relations of this type.

2.1.3 Physical Object

A physical object is an environment object that has a physical representation in the environment. A physical object has a sequence of location points that bounds its location. A physical object is characterized by three properties.

$$\text{Property Vector Physical Object} = \begin{pmatrix} \text{Transparency} & p \\ \text{Blocking} & b \\ \text{Sensor} & s \end{pmatrix}$$

- *Transparency* (p)
  - Boolean value that indicates if the physical object blocks the *observation* of a sensor (both the sensor of an agent and environment sensors).
- *Blocking* (b)
  - Boolean value that indicates if the physical object allows the *position* of any other blocking object or agent to be the same.
- *Sensor* (s)
  - Boolean value that indicates if the physical object is a sensor. Sensors are described in more detail in Section 2.1.3.

The different types of physical objects that are defined in the model are discussed below. If a physical object property is not explicitly mentioned, it is assumed it does not possess that property.

Luggage

Luggage is a transparent and non-blocking object that is described by the following property vector.

$$\text{Property Vector Luggage} = \begin{pmatrix} \text{Luggage type} & u \\ \text{Complexity} & x \\ \text{Threat level} & e \end{pmatrix}$$

The property vector contains three properties:

- *Luggage type* (l)
  - Luggage is either *checked* luggage or *carry-on* luggage.
- *Complexity* (x)
  - Value that indicates the complexity of the luggage. It is a real value between 0 and 1.
- *Threat level* (e)
  - Value that indicates the threat level of the luggage. It is a real value between 0 and 1.

Apart from the properties defined above, luggage is also defined by the following relations.

- $is\_owned\_by(luggage, passenger)$
  - Luggage is owned by a passenger. Luggage has exactly one relation of this type.

Seats

Seats can be present in gate areas and are used by passengers to wait for their flights to depart. Only one passenger can sit on one seat at the same time. Seats are non-blocking and transparent.

Queue separator

Queue separators are used to form queue areas for passengers. They are located in queue areas, are blocking and transparent.

Belt

A belt is present in the checkpoint area. It is used to transport luggage. A belt is blocking and transparent.

Desk

A desk is used in the check-in area and the border control area. If they are located in the check-in area, it is referred to as a check-in desk. Check-in operators and security operators are located behind a desk when they perform their activity. A desk is blocking and transparent.

Wall

Walls are used to separate the airport terminal from the outside world, but also to separate other areas from each other. Walls are blocking and non-transparent.

Sensor

A sensor possesses the functionality that enables agents to sense using a mechanic object. A sensor is able to sense specific elements of the environment. An observation is conceptualized as some representation of the environment. It can be of any form- a real valued number, a Boolean value or a more complex structure. It is denoted as follows.

$$observation\_of(\mathbf{sensor})$$

An observation by a sensor causes a state change of the sensor, observable by agents. This means that a sensor can be in two states: *observed* and *idle*, denoted as follows.

$$sensor\_state(\mathbf{sensor})$$

A sensor can only do an observation in its idle state. When the sensor is in its observed state, after some time r_{sensor} or when an agent accessed the observation, the sensor resets its state to idle. This is formalized as follows.

$$sensor\_state(\mathbf{sensor}) = \mathbf{observed} \twoheadrightarrow_{0,r_{sensor},1,1} sensor\_state(\mathbf{sensor}) = \mathbf{idle}$$

$$\begin{aligned}&input(A)|obs(\mathbf{sensor})\ \&\ sensor\_state(\mathbf{sensor}) = \mathbf{observed}\\&\twoheadrightarrow_{0,0,1,1} sensor\_state(\mathbf{sensor}) = \mathbf{idle}\end{aligned}$$

Three types of sensors are defined, WTMD sensor, X-Ray sensor and ETD sensor. Each of the sensors is non-blocking and transparent.

WTMD sensor The Walk Through Metal Detector (WTMD) is a sensor that detects metal. The WTMD also randomly does an observation with a probability of p_{wtmd}. Observations of the WTMD sensor are represented as either 0, 1 or 2, where 0 is an observation of nothing, 1 is a metal observation and 2 is a random observation. A random observation can be used to start an ETD check.

X-Ray sensor The X-Ray sensor is used to observe properties of luggage. Specifically, the X-Ray sensor observes the threat level of the luggage it is observing. It is represented as a real value between 0 and 1. The threat level is compared to a threshold value to evaluate whether the threat level of the luggage is too high, and a luggage check is required. The X-Ray sensor has a processing time t_{X-Ray}. The X-Ray sensor is operated by a security employee that is assigned to perform the activity X-Ray handling.

ETD sensor The Explosive Trace Detector (ETD) is a sensor that is used to detect the presence of explosive traces. The observation of the ETD sensor is represented by a Boolean value. It can be operated by security employees with the capability to perform the activity ETD checking. The ETD has a processing time t_{etd}.

2.2 Agents

This section describes the agent types present in *AATOM* and how they operate. They follow the architecture that is presented in Section 1.2. In Section 2.2.1 the agent types are outlined, and in Section 2.2.2 the human agents characteristics are described. Finally, Sections 2.2.3 - 2.2.5 specify how the different modules of the AATOM architecture are used by the different agents.

2.2.1 Agent Types

AATOM consists of three agent types. Namely, passengers, operators and orchestration agents. Figure 2.3 presents the structure of the agent types in the environment.

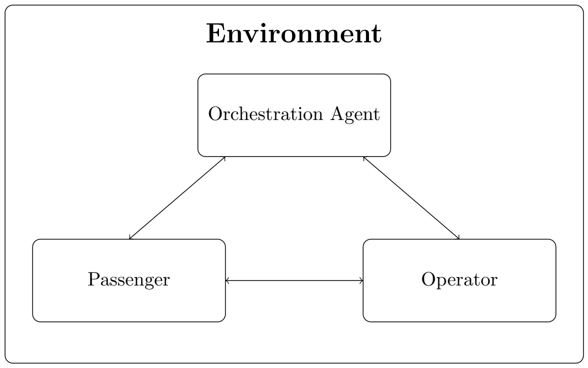

Figure 2.3: The different types of agents and their interactions in AATOM.

The passenger and operator agent types represent the users and operators of the airport terminal. Additional to the passenger and operator agents, orchestration agents can be included to monitor overall performance of the airport terminal and to coordinate terminal operations. An orchestration agent can be implemented in the model and can be defined in different forms. The agent can be represented as a human agent following the three layer architecture, or as a non-human agent. Orchestration agents can be designed as such that it is able to steer operator agents and/or passengers, and make adjustments in the environment, to achieve specific system goals.

The remainder of this document describes the operator and passenger agent types in detail, and will not specify details of the orchestration agent. The definition of this agent is left general on purpose, allowing more specific versions for subsequent studies.

2.2.2 Agents Characteristics

In this section the characteristics of the human agent types, *passenger* and *operator*, are described. We distinguish *departing passengers* and *arriving passengers*, as well as *check-in operators* and *security operators*. The specification of agent characteristics is presented by the agent's *attribute vector*, containing the agent's inherent properties, and by dynamic *relations* that hold for the specific agent type.

Human Agent

All human agents have a set of shared characteristics. They have a location point, denoted as follows.

- $location\_of(agent)$
  - This relation specifies the location point of the human agent. A human agent has exactly one location point.

A human agent is represented as a circle with radius r. Other human agents and other blocking physical objects cannot be within this radius r from the location of the agent. The radius of the human agent is specified in the attribute vector below.

$$\text{Attribute Vector Human Agent} = \begin{pmatrix} \text{Radius} & r \end{pmatrix}$$

The passengers and operators defined below all have the characteristics of the human agents as well. The specific characteristics of these agents are defined in the subsequent sections.

Passenger

Passengers that fly with a departure flight are referred to as *departing passengers*, while passengers that fly with an arrival flight are referred to as *arriving passengers.* To describe departing, the following attribute vectors are defined. For arriving passengers, no attributes are defined.

$$\text{Attribute Vector Departing Passenger} = \begin{pmatrix} \text{Arrival Time} & a \\ \\ \text{Checked-in} & c \end{pmatrix}$$

The properties are described as follows:

- *Arrival time* (a)
  - The arrival time for a passenger is the time it arrives at the airport.
- *Checked-in* (c)
  - This Boolean property defines whether a passenger is already checked-in upon arrival at the airport, or not.

Apart from the attributes defined above, the following relations for passengers are specified.

- $flies\_with(passenger, flight)$
  - This relation specifies the flight that the passenger has. A passenger can only fly with exactly one flight. This is the inverse relation of $is\_flown\_by(flight, passenger)$.
- $is\_checked\_in(passenger, flight)$.
  - A passenger can be checked in into the flight it flies with. This relation can only hold if the passenger also flies with the same flight. The inverse relationship is $has\_checked\_in(flight, passenger)$. Only departing passengers exhibit this relation.
- $owns(passenger, luggage)$
  - This relation specifies the luggage that the passenger owns. A passenger can own any number of luggage. This is the inverse relationship of $is\_owned\_by(luggage, passenger)$.

Check-in Operator

A check-in operator is of the agent type *operator* and described by the following relation.

- $manages(operator, flight)$.
  - A check-in operator can manage a flight. The check-in operator can manage any positive number of flights. This is the inverse relationship of $is\_managed\_by(flight, operator)$.

Security Operator

The security operator is of the agent type *operator*. To describe the security operator, the following relation defined.

- $assigned\_to(operator, activity)$
  - The assignment of a security operator is defined as the activity the operator is currently performing. The activities the operator can execute are specified in Section 2.2.4.

2.2.3 Operational Layer

The operational layer.

Perception Module

Observations can be performed by all agents and are specified by the observation function (see Section 1.2.1). The different types of observations that can be made by passengers are outlined below.

- *obs*(**current_area**)
  - Passengers can observe in what area they currently are.
- *obs*(**physical_object**)
  - Passengers can observe physical objects in some radius r and angle ϕ.
- *obs*(**human_agent**)
  - Passengers can observe human agents in some radius r and angle ϕ.
- *obs*(**flight**)
  - A passenger can observe the flight f it flies with. The flight can be further used to observe relations like gate area and check-in desks.
- *obs*(**luggage**)
  - A passenger can observe the luggage it owns.
- *obs*(**wait_request**)
  - Passengers can be requested to wait by other agents.

Both check-in operators and security operators can do a set of observations. Check-in operators can do two types of observations, outlined below.

- *obs*(**passenger**)
  - The check-in operator can observe a passenger when it is at the check-in desk.
- *obs*(**flight**)
  - The operator agent can observe the flights (*f*) that it manages.

Security operators can do several types of observations, however not all observations are available for each of the security operators present within the model. Based on the assignment of the specific security operator, different types of observations are available.

- *obs*(**passenger**)

  - The security operators performing the luggage drop activity, the physical check activity, the ETD check activity, and the travel document check activity can observe a passenger at the corresponding activity area.

- *obs*(**luggage**)
  - This observation can be performed by a security operator performing the luggage check activity.
- *obs*(**x_ray_sensor**)
  - This observation can be performed by the security operator responsible for the X-Ray activity. The security operator observes an X-Ray sensor and can observe properties like the sensor state or observation it last performed.
- *obs*(**wtmd_sensor**)
  - This observation can be performed by the security operator responsible for the physical check activity. The security operator operator observes an WTMD sensor and can observe properties like the sensor state or observation it last performed.
- *obs*(**etd_sensor**)
  - This observation can be performed by the security operator responsible for the ETD check activity. The security operator observes an ETD sensor and can observe properties like the sensor state or observation it last performed.
- *obs*(**search_request**)
  - The search communication requests the operator to search the luggage that is closest.

Actuation Module

The actuation module is responsible performing the actions in the environment. Within the actuation module, the walking behaviour of human agents is modelled. Walking behaviour is used to transition between activities and to complete activities. The walking mechanism of a passenger is based on the Social Force Model developed by Helbing et al. [5]. The model combines physical and 'social' forces to calculate the velocity and velocity changes of a passenger. The velocity changes depend on observations of human agents and physical objects, within the passenger's proximity. The module requires input from the navigation module (see Section 2.2.4) and the interpretation & perception module (see Section 2.2.4). This walking action is denoted as follows.

*move(**location_point**)*

Passengers can execute the following actions.

- *wait*($\mathbf{t}_{wait}$)
  - Passengers can perform the wait act for t_{wait} seconds. This ensures that a passenger does not move for the specified time.
- *drop_luggage*(**luggage**)
  - Passengers can drop luggage, either at a desk or belt.
- *collect_luggage*(**luggage**)
  - Passengers can collect luggage at a belt.

Security operators can execute the following actions.

- *search*(**luggage**)
  - The search action can be performed by a security operator responsible for the luggage check activity.
- *check_document*(**passenger**)
  - The check document action can be performed by a security operator responsible for the travel document check activity.
- *check_physical*(**passenger**)
  - The check physical action can be performed by a security operator responsible for the physical check activity.
- *check_etd*(**passenger**)
  - The check ETD action can be performed by a security operator responsible for the ETD check activity.
- *check_x_ray*(**x_ray_sensor**)
  - The check X-Ray action can be performed by a security operator responsible for the X-Ray activity.

Finally, check-in operators can execute the following actions.

- *check_in*(**passenger**)
  - The search action can be performed by a security operator responsible for the luggage check activity.

2.2.4 Tactical Layer

Interpretation Module

The interpretation module in AATOM is mostly used for the simple transfer of observations of the perception module to the belief module. Furthermore, all agents can determine if they are stuck, based on current and historic observations. They do this by updating a stuck timer in the belief module based on the time spent at a certain location. This leads to the following state transition.

$$internal(\mathbf{A})|t_{stuck} > t_{threshold} \twoheadrightarrow_{0,0,1,1} internal(\mathbf{A})|int(\mathbf{stuck})$$

Passengers can also observe if an activity area is occupied or not. They do this by counting the number of passengers that they observed and comparing it to the number of available places.

$$\begin{aligned}&internal(\mathbf{A})|next\_activity(\mathbf{activity}) \wedge int(activity\_area(\mathbf{activity})) \wedge int(\mathbf{B}_1) \wedge \ldots \wedge int(\mathbf{B}_n)\\ &\&\ external(\mathbf{A})|in\_area(location\_of(\mathbf{B}_1), activity\_area(\mathbf{activity})) \wedge \ldots\\ &\wedge in\_area(location\_of(\mathbf{B}_n), activity\_area(\mathbf{activity})) \twoheadrightarrow_{0,0,1,1} internal(\mathbf{A})|int(\mathbf{activity\_area\_full})\end{aligned}$$

Where B_i is some observed passenger, and n is the maximum number of passengers that can execute an activity in the area.

Activity Module

The activity module is responsible for preparing the activities an agent will execute based on input from the reasoning module in the the strategic layer. The specification of the functioning of the activity module is specified in Section 1.2.2. How an agent chooses between the different activities is discussed in Section 2.2.5. Here, the different activities of both passengers and operators are discussed in detail.

Passengers Departing passengers can execute a total of six activities, while arriving passengers execute either one or two activities. Each of these activities are outlined below.

Queue Activity The queue activity is executed when other activities cannot be executed due to capacity limitations at the activity area. It can be performed by all passengers that want to start another activity and the queue activity is not bound to a specific area, but is often performed in a queue area. It is modelled as a (set of) waiting period(s) that end(s) when the passenger in front moves forward, or when the planned next activity has an available space again. Formally, the starting condition of the queue activity is defined as follows.

$$internal(\mathbf{A})|int(\textbf{activity\_area\_full}) \twoheadrightarrow_{0,0,1,1}$$
$$internal(\mathbf{A})|\textit{activity\_state}\boldsymbol{(queue\_activity)\ =\ in\_progress}$$

Once the activity started, the following is executed.

$$internal(\mathbf{A})|\textit{activity\_state}\boldsymbol{(queue\_activity)\ =\ in\_progress}$$
$$\twoheadrightarrow_{0,t_{wait},1,1} output(\mathbf{A})|performed(wait(\mathbf{t}_{wait}))$$

$$internal(\mathbf{A})|\textit{activity\_state}\boldsymbol{(queue\_activity)\ =\ in\_progress} \wedge \neg int(\textbf{activity\_area\_full})$$
$$\twoheadrightarrow_{0,0,1,1} internal(\mathbf{A})|\textit{activity\_state}\boldsymbol{(queue\_activity)\ =\ has\_finished}$$

Passenger Check-in Activity The passenger check-in activity is only executed by departing passengers that are not already checked-in and/or own checked luggage that needs to be checked-in. The check-in area forms the activity area of this activity. For the check-in activity, two actions are distinguished: check-in and luggage check-in. Depending on the attribute vector of the passenger, one of these actions or both are executed during the check-in activity. The check-in action is modelled as a passenger waiting in front of a check-in desk that is manned by a *check-in operator*. Both of these actions are part of an interaction, described in Section 2.3.2. More formally, this is modelled as follows.

$$internal(\mathbf{A})|\textit{int(activity\_area}\boldsymbol{(check\_in\_activity)}) \wedge \neg \textit{int}\boldsymbol{(activity\_area\_full)}$$
$$\wedge\ \textit{int}\boldsymbol{(flight)} \wedge \neg \textit{is\_checked\_in}\boldsymbol{(A,flight)} \wedge \textit{next\_activity}\boldsymbol{(check\_in\_activity)}$$
$$\twoheadrightarrow_{0,0,1,1} internal(\mathbf{A})|\textit{activity\_state}\boldsymbol{(check\_in\_activity)\ =\ in\_progress}$$

$$internal(\mathbf{A})|\textit{activity\_state}\boldsymbol{(check\_in\_activity)\ =\ in\_progress}$$
$$\wedge\ \textit{int}\boldsymbol{(wait\_request)} \twoheadrightarrow_{0,t_{wait},1,1} output(\mathbf{A})|performed(wait(\mathbf{t_{wait}}))$$

$$output(\mathbf{A})|performed(wait(\mathbf{t_{wait}}))\ \&\ internal(\mathbf{A})|int(\boldsymbol{flight}) \wedge is\_checked\_in(\boldsymbol{A},\boldsymbol{flight})$$
$$\twoheadrightarrow_{0,0,1,1}\ internal(\mathbf{A})|activity\_state(\boldsymbol{check\_in\_activity}) = \boldsymbol{has\_finished}$$

Checkpoint Activity The checkpoint activity is mandatory for all departing passengers. It is executed at a checkpoint area. The checkpoint activity consists of several actions, of which two mandatory.

1. *drop_luggage*(**luggage**)
2. *collect_luggage*(**luggage**)

Both of these actions can be preceded by an optional waiting activity, if the action cannot be executed at the moment. Furthermore, passengers perform a wait action, when communicated by security operators.

1. *wait*($\mathbf{t_{wait}}$)

The checkpoint activity starts when the following conditions holds.

$$internal(\mathbf{A})|int(activity\_area(\boldsymbol{checkpoint\_activity})) \wedge \neg int(\boldsymbol{activity\_area\_full})$$
$$\wedge\ next\_activity(\boldsymbol{checkpoint\_activity})$$
$$\twoheadrightarrow_{0,0,1,1}\ internal(\mathbf{A})|activity\_state(\boldsymbol{checkpoint\_activity}) = \boldsymbol{in\_progress}$$

After the activity started, the luggage drop action is executed if the following conditions hold.

$$internal(\mathbf{A})|activity\_state(\boldsymbol{checkpoint\_activity}) = \boldsymbol{in\_progress}$$
$$\wedge\ int(\boldsymbol{luggage}) \wedge \neg action\_state(drop\_luggage(\boldsymbol{luggage})) = \boldsymbol{in\_progress}$$
$$\twoheadrightarrow_{0,t_{drop},1,1}\ output(\mathbf{A})|performed(drop\_luggage(\boldsymbol{luggage}))$$

When the drop luggage action is done, the passenger moves to the WTMD if it is not occupied.

$$output(\mathbf{A})|performed(drop\_luggage(\boldsymbol{luggage}))$$
$$\&\ internal(\mathbf{A})|int(\mathbf{wtmd}) \wedge location\_of(\boldsymbol{A}) \neq location\_of(\mathbf{wtmd})$$
$$\twoheadrightarrow_{0,t_{move},1,1}\ output(\mathbf{A})|performed(move(location\_of(\boldsymbol{wtmd})))$$

When that location is reached, the passenger moves to the luggage collect area.

$$output(\mathbf{A})|performed(move(location\_of(\boldsymbol{belt})))$$
$$\&\ internal(\mathbf{A})|int(\mathbf{belt}) \wedge \neg location\_of(\boldsymbol{A}) = location\_of(\mathbf{belt})$$
$$\twoheadrightarrow_{0,t_{move},1,1}\ output(\mathbf{A})|performed(move(location\_of(\boldsymbol{belt})))$$

Once the location of the belt has been reached, the passenger performs the luggage collect action.

$$output(\mathbf{A})|performed(drop\_luggage(\boldsymbol{luggage}))$$
$$\&\ internal(\mathbf{A})|int(\mathbf{belt}) \wedge \neg location\_of(\boldsymbol{A}) = location\_of(\mathbf{belt})$$
$$\wedge\ \neg action\_state(collect\_luggage(\boldsymbol{luggage})) = \boldsymbol{in\_progress}$$
$$\twoheadrightarrow_{0,t_{collect},1,1}\ output(\mathbf{A})|performed(collect\_luggage(\boldsymbol{luggage}))$$

If the agent receives a communication by another agent to wait, it will do that.

$$\begin{aligned}&\mathit{internal}(\mathbf{A})|\mathit{int(\textbf{wait\_request})}\\&\twoheadrightarrow_{0,t_{wait},1,1} \mathit{output}(\mathbf{A})|\mathit{performed(wait(}\mathbf{t}_{wait}))\end{aligned}$$

The activity is finished, when the luggage has been collected and the agent is not waiting.

$$\begin{aligned}&\mathit{internal}(\mathbf{A})|\mathit{activity\_state(\textbf{checkpoint\_activity})} \boldsymbol{=} \textbf{\textit{in\_progress}}\\&\&\ \mathit{output}(\mathbf{A})|\mathit{action\_state(wait(}\mathbf{t}_{wait})) \neq \textbf{in\_progress}\\&\twoheadrightarrow_{0,0,1,1} \mathit{internal}(\mathbf{A})|\mathit{activity\_state(\textbf{checkpoint\_activity})} \boldsymbol{=} \textbf{\textit{has\_finished}}\end{aligned}$$

Passenger Border Control Activity The passenger border control activity is performed by passengers and is modelled as a waiting period in a border control area. This activity is only performed by departing passengers with a flight that has a gate area behind a border control area. The border control activity also consists of an interaction with a security operator, described in Section 2.3. Formally, the border control activity is modelled as follows.

$$\begin{aligned}&\mathit{internal}(\mathbf{A})|\mathit{int(activity\_area(\textbf{passenger\_border\_control\_activity}))} \wedge \neg \mathit{int(\textbf{activity\_area\_full})}\\&\wedge\ \mathit{next\_activity(\textbf{passenger\_border\_control\_activity})}\\&\twoheadrightarrow_{0,0,1,1} \mathit{internal}(\mathbf{A})|\mathit{activity\_state(\textbf{passenger\_border\_control\_activity})} \boldsymbol{=} \textbf{\textit{in\_progress}}\end{aligned}$$

$$\begin{aligned}&\mathit{internal}(\mathbf{A})\mathit{activity\_state(\textbf{passenger\_border\_control\_activity})} \boldsymbol{=} \textbf{\textit{in\_progress}}\\&\wedge\ \mathit{int(\textbf{wait\_request})} \twoheadrightarrow_{0,t_{wait},1,1} \mathit{output}(\mathbf{A})|\mathit{performed(wait(}\mathbf{t}_{wait}))\end{aligned}$$

$$\begin{aligned}&\mathit{internal}(\mathbf{A})|\mathit{activity\_state(\textbf{passenger\_border\_control\_activity})} \boldsymbol{=} \textbf{\textit{in\_progress}}\\&\&\ \mathit{output}(\mathbf{A})|\mathit{action\_state(wait(}\mathbf{t}_{wait})) \neq \textbf{in\_progress}\\&\twoheadrightarrow_{0,0,1,1} \mathit{internal}(\mathbf{A})|\mathit{activity\_state(\textbf{passenger\_border\_control\_activity})} \boldsymbol{=} \textbf{\textit{has\_finished}}\end{aligned}$$

Gate Activity The gate activity is modelled as a waiting period in the gate area corresponding to the departing passenger's flight. Passengers seek for a free seat and sit until the flight departs. If no seat is available the queuing activity is executed until a seat becomes available. At the time point a flight departs, the passengers corresponding to this flight, disappear from the environment. Formally, the gate activity is modelled as follows.

$$\begin{aligned}&\mathit{internal}(\mathbf{A})|\mathit{int(activity\_area(\textbf{gate\_activity}))} \wedge \neg \mathit{int(\textbf{activity\_area\_full})}\\&\wedge\ \mathit{next\_activity(\textbf{gate\_activity})}\\&\twoheadrightarrow_{0,0,1,1} \mathit{internal}(\mathbf{A})|\mathit{activity\_state(\textbf{gate\_activity})} \boldsymbol{=} \textbf{\textit{in\_progress}}\end{aligned}$$

$$\begin{aligned}&\mathit{internal}(\mathbf{A})|\mathit{int}(\textbf{chair}) \wedge \mathit{in\_area}(\mathit{location\_of}(\textbf{chair}), \mathit{activity\_area}(\textbf{gate\_activity}))\\&\wedge\ \mathit{location\_of(\textbf{A})} \neq \mathit{location\_of}(\textbf{chair}) \wedge \mathit{activity\_state(\textbf{gate\_activity})} \boldsymbol{=} \textbf{\textit{in\_progress}}\\&\twoheadrightarrow_{0,t_{move},1,1} \mathit{output}(\mathbf{A})|\mathit{performed(move(location\_of(\textbf{chair})))}\end{aligned}$$

$$internal(\mathbf{A})|\textit{activity\_state}(\boldsymbol{gate\_activity}) = \boldsymbol{in\_progress} \wedge \textit{location\_of}(\boldsymbol{A}) = location\_of(\mathbf{chair})$$
$$\twoheadrightarrow_{0,0,1,1} internal(\mathbf{A})|\textit{activity\_state}(\boldsymbol{gate\_activity}) = \boldsymbol{has\_finished}$$

Facility Activity The facility activity is performed in a facility area. The facility activity consists of three types.

- Shop
- Restaurant
- Bathroom

Each of these types is associated with corresponding activity areas. The shop type is modelled as a random walk within the corresponding shop area, while the other two types are modelled as waiting periods in a corresponding facility area. This activity can be performed by both departing as arriving passengers. Formally, the facility activity is modelled as follows.

$$internal(\mathbf{A})|\textit{int}(\textit{activity\_area}(\boldsymbol{facility\_activity})) \wedge \neg \textit{int}(\boldsymbol{activity\_area\_full})$$
$$\wedge\ \textit{next\_activity}(\boldsymbol{facility\_activity})$$
$$\twoheadrightarrow_{0,0,1,1} internal(\mathbf{A})|\textit{activity\_state}(\boldsymbol{facility\_activity}) = \boldsymbol{in\_progress}$$

Both the restaurant and bathroom type are modelled as a waiting period at a random location in the related area. Below this is formalized for the restaurant type, while the bathroom type works the same.

$$internal(\mathbf{A})|\textit{activity\_state}(\boldsymbol{facility\_activity}) = \boldsymbol{in\_progress}$$
$$\wedge\ \textit{action\_state}(\textit{wait}(\mathbf{t}_{wait})) \neq \mathbf{in\_progress} \wedge \textit{action\_state}(\textit{move}(\mathbf{location})) \neq \mathbf{in\_progress}$$
$$\twoheadrightarrow_{0,t_{move},1,1} output(\mathbf{A})||\textit{performed}(\textit{move}(\textit{random\_location}(\textit{activity\_area}(\boldsymbol{facility\_activity}))))$$

Where the function $random\_location(area)$ returns a random location point in the area. If the location has been reached, the agent waits for a period of time.

$$internal(\mathbf{A})|\textit{activity\_state}(\boldsymbol{facility\_activity}) = \boldsymbol{in\_progress}$$
$$\&\ output(\mathbf{A})||\textit{performed}(\textit{move}(\textit{random\_location}(\textit{activity\_area}(\boldsymbol{facility\_activity}))))$$
$$\twoheadrightarrow_{0,t_{wait},1,1} output(\mathbf{A})|\textit{performed}(\textit{wait}(\mathbf{t}_{wait}))$$

After the waiting period is over, the activity is complete.

$$internal(\mathbf{A})|\textit{activity\_state}(\boldsymbol{facility\_activity}) = \boldsymbol{in\_progress}$$
$$\&\ output(\mathbf{A})|\textit{performed}(\textit{wait}(\mathbf{t}_{wait}))$$
$$\twoheadrightarrow_{0,0,1,1} internal(\mathbf{A})|\textit{activity\_state}(\boldsymbol{facility\_activity}) = \boldsymbol{has\_finished}$$

The shop type is modelled as a random walk within the shop area. This is modelled as a walk to N location points within the shop area.

$$internal(\mathbf{A})|\textit{activity\_state}(\boldsymbol{facility\_activity}) = \boldsymbol{in\_progress}$$
$$\wedge\ \textit{action\_state}(\textit{move}(\mathbf{location})) \neq \mathbf{in\_progress} \wedge number\_of\_locations(\mathbf{i}) < \mathbf{N}$$
$$\twoheadrightarrow_{0,t_{move},1,1} output(\mathbf{A})|\textit{performed}(\textit{move}(\textit{random\_location}(\textit{activity\_area}(\boldsymbol{facility\_activity}))))$$
$$\&\ internal(\mathbf{A})|number\_of\_locations(\mathbf{i{+}1})$$

Where *number_of_locations* is the number of locations that were visited within the shop already. The value i is initialized at 0. When N locations were reached, the activity is finished.

$$internal(\mathbf{A})|activity\_state(\textbf{\textit{facility\_activity}}) = \textbf{\textit{in\_progress}} \land number\_of\_points(\mathbf{i{+}1}) = \mathbf{N}$$
$$\twoheadrightarrow_{0,0,1,1} internal(\mathbf{A})|activity\_state(\textbf{\textit{facility\_activity}}) = \textbf{\textit{has\_finished}}$$

Exit Activity The exit activity is performed by arriving passengers and is executed at all times. The activity is modelled as a passenger arriving within the entrance area of the airport and disappear from the environment. Formally, the exit activity is modelled as follows.

$$internal(\mathbf{A})|int(activity\_area(\textbf{\textit{exit\_activity}})) \land \neg int(\textbf{\textit{activity\_area\_full}})$$
$$\land\ next\_activity(\textbf{\textit{exit\_activity}})$$
$$\twoheadrightarrow_{0,0,1,1} internal(\mathbf{A})|activity\_state(\textbf{\textit{exit\_activity}}) = \textbf{\textit{in\_progress}}$$

Check-in Operator The check-in operator has the capability to observe and to execute the handle check-in activity.

Operator Check-in Activity Each check-in operator performs the passenger check-in activity. During the check-in activity the check-in operator interacts with the passenger and the flight (f) corresponding to the passenger. These interactions are described in Sections 2.3.1 and 2.3.2.

$$internal(\mathbf{A})|int(\textbf{\textit{passenger}}) \land manages(\textbf{\textit{A}},\textbf{\textit{flight}})\ \&\ external(\mathbf{A})|flies\_with(\textbf{\textit{passenger}},\textbf{\textit{flight}})$$
$$\land \neg is\_checked\_in(\textbf{\textit{passenger}},\textbf{\textit{flight}}) \twoheadrightarrow_{0,0,1,1} internal(\mathbf{A})|activity\_state(\textbf{\textit{check\_in\_activity}}) = \textbf{\textit{in\_progress}}$$

When the activity is in progress, and the passenger is not checked-in yet, the operator communicates a wait order to the passenger.

$$internal(\mathbf{A})|activity\_state(\textbf{\textit{check\_in\_activity}}) = \textbf{\textit{in\_progress}}$$
$$\&\ external(\mathbf{A})|flies\_with(\textbf{\textit{passenger}},\textbf{\textit{flight}}) \land \neg is\_checked\_in(\textbf{\textit{passenger}},\textbf{\textit{flight}})$$
$$\twoheadrightarrow_{0,0,1,1} external(\mathbf{A})|is\_checked\_in(\textbf{\textit{passenger}},\textbf{\textit{flight}})$$
$$\&\ output(\mathbf{A})|performed(communicate(\mathbf{passenger}, \mathbf{wait}, \mathbf{t}_{wait}))$$

After the passenger waited, the activity is over.

$$internal(\mathbf{A})|activity\_state(\textbf{\textit{check\_in\_activity}}) = \textbf{\textit{in\_progress}}$$
$$\&\ output(\mathbf{A})|performed(communicate(\mathbf{passenger}, \mathbf{wait}, \mathbf{t}_{wait}))$$
$$\twoheadrightarrow_{0,t_{wait},1,1} internal(\mathbf{A})|activity\_state(\textbf{\textit{check\_in\_activity}}) = \textbf{\textit{has\_finished}}$$

Security Operator As defined in Section 2.2.2, a security operator has an assignment. The assignment defines which activity the security operator is currently assigned to. Furthermore, a security operator can do observations. All activities and observations that can be executed by security operators are defined below.

Travel Document Check Activity Travel document checking is modelled to be an interaction with passengers. The activity starts when a passenger is observed.

$$internal(\mathbf{A})|activity\_state(\textbf{\textit{tdc\_activity}}) \neq \textbf{in\_progress} \land int(\textbf{\textit{passenger}})$$
$$\twoheadrightarrow_{0,0,1,1} internal(\mathbf{A})|activity\_state(\textbf{\textit{tdc\_activity}}) = \textbf{\textit{in\_progress}}$$

When the activity is in progress, and the operator did not communicate with the passenger yet, the operator communicates a wait order to the passenger.

$$\begin{aligned}&internal(\mathbf{A})|activity\_state(\boldsymbol{tdc\_activity}) = \boldsymbol{in\_progress}\\&\&\ output(\mathbf{A})|\neg performed(communicate(\mathbf{passenger}, \mathbf{wait}, \mathbf{t}_{wait}))\\&\twoheadrightarrow_{0,0,1,1} output(\mathbf{A})|performed(communicate(\mathbf{passenger}, \mathbf{wait}, \mathbf{t}_{wait}))\end{aligned}$$

After the passenger waited, the activity is over.

$$\begin{aligned}&internal(\mathbf{A})|activity\_state(\boldsymbol{tdc\_activity}) = \boldsymbol{in\_progress}\\&\&\ output(\mathbf{A})|performed(communicate(\mathbf{passenger}, \mathbf{wait}, \mathbf{t}_{wait}))\\&\twoheadrightarrow_{0,t_{wait},1,1} internal(\mathbf{A})|activity\_state(\boldsymbol{tdc\_activity}) = \boldsymbol{has\_finished}\end{aligned}$$

Luggage Drop Activity The luggage drop activity is currently modelled as a holder activity. As passengers currently drop their luggage on their own (within the checkpoint activity), no action is needed from the security operator performing the luggage drop activity. In future versions of AATOM, the luggage drop activity can be modelled as an an interaction with passengers that do not execute their *drop_luggage* action properly.

X-Ray Activity Luggage is checked by going through the X-Ray sensor. This process is executed by an security operator and defined as the X-ray activity. The X-Ray activity starts when the security operator observes that the X-Ray sensor is in the observed state. The operator observes the X-Ray sensor state and the observed the output of the X-Ray sensor, which is a real number representing the threat level of the luggage.

$$\begin{aligned}&internal(\mathbf{A})|int(\boldsymbol{x\_ray\_sensor}) \wedge sensor\_state(\boldsymbol{x\_ray\_sensor}) = \mathbf{observed}\\&\twoheadrightarrow_{0,0,1,1} internal(\mathbf{A})|activity\_state(\boldsymbol{checkpoint\_activity}) = \boldsymbol{in\_progress}\end{aligned}$$

If the output of the X-Ray sensor exceeds a threshold h, the X-Ray activity includes an interaction with the operator for the luggage checking.

$$\begin{aligned}&internal(\mathbf{A})|activity\_state(\boldsymbol{checkpoint\_activity}) = \boldsymbol{in\_progress}\\&\wedge int(\boldsymbol{x\_ray\_sensor}) \wedge observation\_of(\boldsymbol{x\_ray\_sensor}) > \mathbf{h}\\&\twoheadrightarrow_{0,0,1,1} internal(\mathbf{A})|activity\_state(\boldsymbol{checkpoint\_activity}) = \boldsymbol{has\_finished}\end{aligned}$$

If the observed X-Ray sensor observation is below the threshold h, the activity is completed without any further action. The luggage will proceed to the luggage collect area.

$$\begin{aligned}&internal(\mathbf{A})|activity\_state(\boldsymbol{checkpoint\_activity}) = \boldsymbol{in\_progress} \wedge int(\boldsymbol{x\_ray\_sensor})\\&\wedge observation\_of(\boldsymbol{x\_ray\_sensor}) < \mathbf{h} \wedge int(\mathbf{operator}) \wedge int(\mathbf{luggage})\\&\twoheadrightarrow_{0,0,1,1} internal(\mathbf{A})|activity\_state(\boldsymbol{checkpoint\_activity}) = \boldsymbol{has\_finished}\\&\&\ output(\mathbf{A})|performed(communicate(\mathbf{operator}, \mathbf{search}, \mathbf{luggage}))\end{aligned}$$

Luggage Check Activity If the security operator performing the luggage check activity observes the search communication, the luggage check activity is started. This activity is modelled as an interaction between the security operator responsible for the luggage check and the passenger. This interaction is described in Section 2.3.2.

$$\begin{aligned}&internal(\mathbf{A})|activity\_state(\boldsymbol{luggage\_check\_activity}) \neq \mathbf{in\_progress} \wedge int(\boldsymbol{search\_request})\\&\wedge int(\boldsymbol{luggage}) \twoheadrightarrow_{0,0,1,1} internal(\mathbf{A})|activity\_state(\boldsymbol{luggage\_check\_activity}) = \boldsymbol{in\_progress}\end{aligned}$$

$$\begin{aligned}&internal(\mathbf{A})|activity\_state(\boldsymbol{luggage\_check\_activity}) = \boldsymbol{in\_progress}\\ &\wedge int(\mathbf{luggage}) \wedge int(owner\_of(\mathbf{luggage}))\\ &\twoheadrightarrow_{0,0,1,1} output(\mathbf{A})|performed(communicate(owner\_of(\mathbf{luggage}), \mathbf{wait}, \mathbf{t}_{wait}))\end{aligned}$$

$$\begin{aligned}&internal(\mathbf{A})|activity\_state(\boldsymbol{luggage\_check\_activity}) = \boldsymbol{in\_progress}\\ &\&\ output(\mathbf{A})|performed(communicate(owner\_of(\mathbf{luggage}), \mathbf{wait}, \mathbf{t}_{wait}))\\ &\twoheadrightarrow_{0,t_{wait},1,1} internal(\mathbf{A})|activity\_state(\boldsymbol{luggage\_check\_activity}) = \boldsymbol{has\_finished}\end{aligned}$$

Physical Check Activity The security operator performing the physical checking activity is responsible for detecting illegal items on the body of a passenger. It performs a check based on observations of the WTMD, the check is initiated when the WTMD sensor output value is 1. The process is modelled as an interaction with a passenger (see Section 2.3.2).

$$\begin{aligned}&internal(\mathbf{A})|int(\boldsymbol{wtmd}) \wedge sensor\_state(\boldsymbol{wtmd}) = \mathbf{observed}\\ &\twoheadrightarrow_{0,0,1,1} internal(\mathbf{A})|activity\_state(\boldsymbol{physical\_check\_activity}) = \boldsymbol{in\_progress}\end{aligned}$$

$$\begin{aligned}&internal(\mathbf{A})|activity\_state(\boldsymbol{physical\_check\_activity}) = \boldsymbol{in\_progress} \wedge int(\boldsymbol{wtmd})\\ &\wedge sensor\_state(\boldsymbol{wtmd}) = \mathbf{observed} \wedge observation\_of(\boldsymbol{wtmd}) = \mathbf{1}\\ &\twoheadrightarrow_{0,0,1,1} output(\mathbf{A})|performed(communicate(\mathbf{passenger}, \mathbf{wait}, \mathbf{t}_{wait}))\end{aligned}$$

$$\begin{aligned}&internal(\mathbf{A})|activity\_state(\boldsymbol{physical\_check\_activity}) = \boldsymbol{in\_progress}\\ &\&\ output(\mathbf{A})|performed(communicate(\mathbf{passenger}, \mathbf{wait}, \mathbf{t}_{wait}))\\ &\twoheadrightarrow_{0,t_{wait},1,1} internal(\mathbf{A})|activity\_state(\boldsymbol{physical\_check\_activity}) = \boldsymbol{has\_finished}\end{aligned}$$

ETD Check Activity The security operator performing the ETD checking activity is responsible for detecting explosive traces on a passenger. The check is based on indications of the WTMD, the check is initiated when the WTMD sensor output value is 2. The process is modelled as an interaction with a passenger and a waiting period of the passenger.

$$\begin{aligned}&internal(\mathbf{A})|int(\boldsymbol{wtmd}) \wedge sensor\_state(\boldsymbol{wtmd}) = \mathbf{observed}\\ &\twoheadrightarrow_{0,0,1,1} internal(\mathbf{A})|activity\_state(\boldsymbol{etd\_check\_activity}) = \boldsymbol{in\_progress}\end{aligned}$$

$$\begin{aligned}&internal(\mathbf{A})|activity\_state(\boldsymbol{etd\_check\_activity}) = \boldsymbol{in\_progress} \wedge int(\boldsymbol{wtmd})\\ &\wedge sensor\_state(\boldsymbol{wtmd}) = \mathbf{observed} \wedge observation\_of(\boldsymbol{wtmd}) = \mathbf{2}\\ &\twoheadrightarrow_{0,0,1,1} output(\mathbf{A})|performed(communicate(\mathbf{passenger}, \mathbf{wait}, \mathbf{t}_{wait}))\end{aligned}$$

$$\begin{aligned}&internal(\mathbf{A})|activity\_state(\boldsymbol{etd\_check\_activity}) = \boldsymbol{in\_progress}\\ &\&\ output(\mathbf{A})|performed(communicate(\mathbf{passenger}, \mathbf{wait}, \mathbf{t}_{wait}))\\ &\twoheadrightarrow_{0,t_{wait},1,1} internal(\mathbf{A})|activity\_state(\boldsymbol{etd\_check\_activity}) = \boldsymbol{has\_finished}\end{aligned}$$

Operator Border Control Activity The operator border control check activity is executed at the border control area, and is initiated when a passenger is at the border control desk. The activity is modelled at a waiting time enforced on the passenger.

$$\begin{aligned}&internal(\mathbf{A})|\textit{activity\_state}(\textbf{\textit{border\_control\_activity}}) \neq \textbf{in\_progress} \wedge int(\textbf{\textit{passenger}})\\&\twoheadrightarrow_{0,0,1,1} internal(\mathbf{A})|\textit{activity\_state}(\textbf{\textit{border\_control\_activity}}) = \textbf{\textit{in\_progress}}\end{aligned}$$

When the activity is in progress, and the operator did not communicate with the passenger yet, the operator communicates a wait order to the passenger.

$$\begin{aligned}&internal(\mathbf{A})|\textit{activity\_state}(\textbf{\textit{border\_control\_activity}}) = \textbf{\textit{in\_progress}}\\&\&\ output(\mathbf{A})|\neg performed(communicate(\mathbf{passenger}, \mathbf{wait}, \mathbf{t}_{wait}))\\&\twoheadrightarrow_{0,0,1,1} output(\mathbf{A})|performed(communicate(\mathbf{passenger}, \mathbf{wait}, \mathbf{t}_{wait}))\end{aligned}$$

After the passenger waited, the activity is over.

$$\begin{aligned}&internal(\mathbf{A})|\textit{activity\_state}(\textbf{\textit{border\_control\_activity}}) = \textbf{\textit{in\_progress}}\\&\&\ output(\mathbf{A})|performed(communicate(\mathbf{passenger}, \mathbf{wait}, \mathbf{t}_{wait}))\\&\twoheadrightarrow_{0,t_{wait},1,1} internal(\mathbf{A})|\textit{activity\_state}(\textbf{\textit{border\_control\_activity}}) = \textbf{\textit{has\_finished}}\end{aligned}$$

Navigation Module

While it is not required to find the shortest path towards a target area, in this work the shortest path finding algorithm Jump Point Search is used []. Each time the move action (performed by the actuation module) is in progress, and no path to the designated location is calculated yet, a path is calculated towards the designated location.

$$\begin{aligned}&internal(\mathbf{A})|\textit{end\_point}(\textbf{\textit{path}}) \neq \mathbf{location}\ \&\ output(\mathbf{A})|\textit{action\_state}(move(\mathbf{location})) = \mathbf{in\_progress}\\&\twoheadrightarrow_{0,0,1,1} internal(\mathbf{A})|\textit{performed}(\textit{update}(\textbf{\textit{path}}, \textbf{\textit{location}}))\end{aligned}$$

Agents also recalculate their paths when they are stuck. They do so by finding a new path to the activity area if the current path does not lead to the activity area anymore.

$$internal(\mathbf{A})|int(\mathbf{stuck}) \twoheadrightarrow_{0,0,1,1} internal(\mathbf{A})|\textit{performed}(\textit{update}(\textbf{\textit{path}}, \textit{end\_point}(\textbf{\textit{path}}))) \wedge \neg int(\mathbf{stuck})$$

2.2.5 Strategic Layer

Here, the strategic layer for departing and arriving passengers are described. As the behaviour of operator agents follows directly from the individual activity they perform, the plans and goals of these agents consists only of a single item (the activity they perform). Therefore, a strategic layer for these agents is not defined.

Goals Module

The goals module.

Departing Passenger A departing passenger has at most five goals that it wishes to achieve. These goals are related to the activities it can execute, outlined below.

$$\text{Activities} = \begin{pmatrix}\text{Passenger check-in activity}\\\text{Security activity}\\\text{Passenger border control activity}\\\text{Gate activity}\\\text{Facility activity}\end{pmatrix}$$

There are two goals that are the same for every departing passenger. The conditions for these goals are stated below.

$$internal(\mathbf{A})|activity\_state(\boldsymbol{gate\_activity}) = \boldsymbol{has\_finished} \,\&\, external(\mathbf{A})|\mathbf{t} < \mathbf{t}_{flight}$$

$$\begin{aligned}&internal(\mathbf{A})|activity\_state(\boldsymbol{checkpoint\_activity}) = \boldsymbol{has\_finished}\\&\wedge\, activity\_state(\boldsymbol{gate\_activity}) \neq \ \boldsymbol{has\_finished}\end{aligned}$$

The first goals states that the passenger wants to finish its gate activity before the time of its flight, while the second goal states that the security activity has to be finished before the gate activity.

Other goals of departing passengers are introduced under certain conditions. The passenger check-in activity will be executed if the checked-in feature of the passenger is false and/or owns checked luggage that needs to be checked-in. If these conditions are met, the goal is defined as follows.

$$\begin{aligned}&internal(\mathbf{A})|activity\_state(\boldsymbol{check\_in\_activity}) = \boldsymbol{has\_finished}\\&\wedge\, activity\_state(\boldsymbol{checkpoint\_activity}) \neq \ \boldsymbol{has\_finished}\end{aligned}$$

The passenger border control activity is executed by passengers with a flight that has a gate area behind a border control area. If this condition is met, the following goals are defined.

$$\begin{aligned}&internal(\mathbf{A})|activity\_state(\boldsymbol{border\_control\_activity}) = \boldsymbol{has\_finished}\\&\wedge\, activity\_state(\boldsymbol{gate\_activity}) \neq \ \boldsymbol{has\_finished}\end{aligned}$$

$$\begin{aligned}&internal(\mathbf{A})|activity\_state(\boldsymbol{security\_activity}) = \boldsymbol{has\_finished}\\&\wedge\, activity\_state(\boldsymbol{border\_control\_activity}) \neq \ \boldsymbol{has\_finished}\end{aligned}$$

Finally, a departing passenger can have a goal related to facility visits. This goal is added based on input by the user.

$$\begin{aligned}&internal(\mathbf{A})|activity\_state(\boldsymbol{facility\_activity}) = \boldsymbol{has\_finished}\\&\wedge\, activity\_state(\boldsymbol{gate\_activity}) \neq \ \boldsymbol{has\_finished}\end{aligned}$$

Arriving Passenger An arriving passenger has either one or two goals that it wishes to fulfill. These goals are related to the activities stated below.

$$\text{Activities} = \begin{pmatrix}\text{Facility activity}\\ \\ \text{Exit activity}\end{pmatrix}$$

The exit activity has to be executed by all arriving passengers. The goal that is related is stated below.

$$internal(\mathbf{A})|activity\_state(\boldsymbol{exit\_activity}) = \boldsymbol{has\_finished} \,\&\, external(\mathbf{A})|\mathbf{t} < \mathbf{t}_{exit}$$

This goal represents that the gate activity has to be finished as soon as possible. Further, based on user input, the following goal is added to the goals of the passenger.

$$\begin{aligned}&internal(\mathbf{A})|activity\_state(\boldsymbol{facility\_activity}) = \boldsymbol{has\_finished}\\&\wedge activity\_state(\boldsymbol{exit\_activity}) \neq \ \boldsymbol{has\_finished}\end{aligned}$$

Reasoning Module

The reasoning module is responsible for the reasoning processes of an agent. It contains three modules: planning, analysis and decision making. The analysis module is responsible for the analysis of the current state of the agent. If this module determines the need for an updated plan or a decision, it communicates this to the respective modules. In this model, the analysis module is not used and only serves as a placeholder. The other two models are discussed below.

Planning Module When a departing passenger arrives in the airport terminal (i.e. at $t = t_{arrival}$, it will generate a plan containing a complete set of activities it has to perform to fulfill its goals. A plan is defined by the set of selected activities and the order of the activities. The activities are selected from the following set.

Departing Passenger When a departing passenger arrives in the airport terminal, it will generate a plan containing a complete set of activities it has to perform to fulfill its goals. A plan is defined by the set of selected activities and the order of the activities. The activities are selected from the following set.

The activity order, for each passenger, is based on the goals that are defined in the goals module. All activities have a fixed position in the sequence of activities, apart from the facility activity. To determine the position of this activity in the plan, an integer value n is drawn randomly from the set $n \in N$ with $N = [1, M]$. Here, the value M represents the total number of activities present in the plan, not taking into account the facility activity. Then, the value of n represents on which position in the plan the facility activity is located. For instance, a passenger needs to check-in, go through security and to the gate, then $M = 3$. During the generation of the plan a random number is drawn from the set $N = [1, 3]$. If this number is 2, the facility activity will be executed as second activity, thus after the check-in activity is completed. If this position is infeasible (i.e. there are no facilities before or after the security to fulfill the activity), the position is removed from the set N, and the process is repeated.

Arriving Passenger When an arriving passenger arrives in the arrival area it will generate a plan which contains a complete set of activities it will perform in order to leave the airport. The plan is defined by the goals of the passenger.
The passenger generates the plan by evaluating its goals (see Section 2.2.2), directly after it is generated in the arrival area. As this contains only one or two goals, the plan is straightforward: it ends with the *exit_activity* and optionally starts with the *facility_activity*.

Belief Module

The belief module is responsible for maintaining the belief of the agent. The belief is represented as a triple, outlined below.

$$\text{Belief} = \begin{pmatrix} \text{Interpretations} \\ \text{Activity State} \\ \text{Plan} \end{pmatrix}$$

The interpretations refer to the interpretations the agent made. When an agent observes something different from its current belief, the belief is updated. The activity state is a vector of activity-state pairs, containing all activities the agent can execute and their corresponding states . When the status of any activity changes, the activity-state pair is updated. Finally, the plan is updated in the belief module whenever the plan is generated and/or changed in the planning module.

Other modules present in the model, like navigation and reasoning, can access the belief maintained in the belief module such that they can function properly.

2.3 Interactions

Additional to the autonomous behaviour described in Chapter 2.2, agents interact with each other and the environment. This chapter describes the interactions between agents and the environment in Section 2.3.1, and the interactions between agents in Section 2.3.2. Furthermore, in Section 2.3.3 it is outlined how coordination can be implemented in the model.

2.3.1 Environment

This section describes the interactions defined between agents and the environment defined for this model.

Check-in Operator Interaction with Flight

Check-in operators can observe the flights that they have a relation with. They can also edit the flight property *set of checked-in passengers*, by adding observed passengers to this set. This interaction belongs to the check-in activity described in Section 2.2.4.

$$\begin{aligned}&\mathit{internal}(\mathbf{A})|\mathit{int}(\boldsymbol{passenger}) \land \mathit{manages}(\boldsymbol{A,flight})\ \&\ \mathit{external}(\mathbf{A})|\mathit{flies\_with}(\boldsymbol{passenger,flight})\\ &\land \neg \mathit{is\_checked\_in}(\boldsymbol{passenger,flight}) \twoheadrightarrow_{0,0,1,1} \mathit{external}(\mathbf{A})|\mathit{is\_checked\_in}(\boldsymbol{passenger,flight})\end{aligned}$$

Security Operator interaction with Sensors

As was described in Section 2.1.3, sensors can be in two states: *observed* and *idle*. When a security operator accesses the observation, the sensor resets its state to idle. This interaction occurs between the three sensor types (WTMD sensor, X-Ray sensor and ETD sensor) and their operators.

$$\begin{aligned}&\mathit{input}(\mathbf{A})|\mathit{obs}(\boldsymbol{sensor})\ \&\ \mathit{external}(\mathbf{A})|\mathit{sensor\_state}(\boldsymbol{sensor}) = \mathbf{observed} \land \mathit{observation\_of}(\boldsymbol{sensor}) \neq \mathbf{null}\\ &\twoheadrightarrow_{0,0,1,1} \mathit{external}(\mathbf{A})|\mathit{sensor\_state}(\boldsymbol{sensor}) = \mathbf{idle} \land \mathit{observation\_of}(\boldsymbol{sensor}) = \mathbf{null}\end{aligned}$$

2.3.2 Agents

This section describes the interactions defined between agents for this model. Two types of agent interactions are defined. Interactions between operators and passengers, and interactions between security operators. These are discussed in the following sections.

Operator - Passenger Interaction

All interactions between passengers and operators follow the same structure: the passenger is instructed to wait for a specified time period by the operator (performing a specific activity) under a certain condition.

The main structure of an interaction between an operator and a passenger in this work is outlined below.

$$internal(\mathbf{operator})|activity\_state(\mathbf{current\_activity}) = \mathbf{in\_progress} \wedge int(\mathbf{passenger})$$
$$\twoheadrightarrow_{t_{wait},t_{wait},1,1}\ output(\mathbf{passenger})|performed(wait(\mathbf{t}_{wait}))$$

In the outline above, when some operator is performing activity and observes a passenger in its activity area, it communicates a waiting instruction to the observed passenger. This waiting instruction consists of a waiting time t_{wait}, which is followed by the passenger.

The following interactions between operators and passengers are defined. The activity that operator is performing is indicated between brackets.

- Check-in Operator (Check-in) with Passenger
- Security Operator (Travel Document Check) with Passenger
- Security Operator (Luggage Drop) with Passenger
- Security Operator (Luggage Check) with Passenger
- Security Operator (Physical Check) with Passenger
- Security Operator (ETD Check) with Passenger
- Security Operator (Border Control) with Passenger

Operator - Operator Interaction

One interaction between operators is defined in the model. The interaction is between the security operator responsible for X-Ray scanning and the security operator responsible for luggage checking. When the security operator responsible for X-Ray scanning observes a threat level above h, it communicates a search command to the security operator responsible for luggage checking. This operator then starts its luggage check activity.

$$internal(\mathbf{A})|activity\_state(\mathbf{x\_ray\_activity}) = \mathbf{in\_progress} \wedge int(\mathbf{x\_ray})$$
$$external(\mathbf{A})|sensor\_state(\boldsymbol{x\_ray}) = \mathbf{observed} \wedge observation\_of(\boldsymbol{x\_ray}) > \mathbf{h}$$
$$\twoheadrightarrow_{0,0,1,1}\ output(\mathbf{B})|performed(search(\mathbf{luggage}))$$

2.3.3 Coordination

As described in Section 2.2.1, a global control agent can be implemented for coordination of terminal operations. In order to allow for coordination, the global control agent can be designed to interact with operators, passengers and the environment. Furthermore, coordination can be implemented by allowing specific low level interactions between passengers and operators. Then coordination can occur in a cooperative setting or in a competitive setting. In a cooperative setting, agents try to combine their efforts to accomplish as a group what they could not accomplish as individual agents. In a competitive setting, agents try to get what only some of them can have; competitive agents try to maximize their own benefit at the expense of other agents.

Chapter 3

Input Parameters

The input parameters of the model are described in this chapter. Two types of input parameters are distinguished: environment parameters and agent parameters. These parameters are used to design the system for a specific instance.

3.1 Environment Parameters

Three types of environment parameters are distinguished: map layout parameters, airport parameters and sensor parameters. Each of these parameter types are discussed in more detail below.

3.1.1 Map Layout

Two map layout parameters are distinguished and listed below.

- Area locations
- Physical object locations

The area locations refers to the layout of all areas that are present without the model. The physical object locations refer to all the physical objects that are defined and placed in the model.

3.1.2 Airport Parameters

A single airport parameter is identified and showed below.

- Flight set
  - Type i
  - Time t
  - Set of checked-in passengers C
  - Gate area g
  - Set of check-in desks D

The flight set is the set of flights that are present within the model. For each flight in the set, the type, time, gate area and set of check-in desks needs to be determined.

3.1.3 Sensor Parameters

Three sensor parameters are identified and listed below.

- WTMD sensor
  - Recovery time r_{wtmd}
  - Random observation probability p_{wtmd}
  - Processing time t_{wtmd}
- X-Ray sensor
  - Recovery time r_{X-Ray}
  - Processing time t_{X-Ray}
- ETD sensor
  - Recovery time r_{etd}
  - Processing time t_{etd}

As there are three types of sensors defined in the model, three types of parameters need to be determined. For each of the sensors, a recovery time r_{sensor} (the time the sensor takes to return to its idle state) and the processing time t_{sensor} is identified. For the WTMD an additional parameter, referring to the probability of random observations of the WTMD.

3.2 Agent Parameters

As two types of agents are defined in the model, two types of agent parameters are distinguished as well: passenger parameters and operator parameters. Both of these parameter types are discussed below.

3.2.1 Passenger Parameters

Seven passenger parameters are identified and are listed below.

- Arrival time a
- Flight f
- Luggage set L
  - Type l
  - Complexity x
  - Threat level e
- Checked-in c
- Facility visitor y
- Observation radius r
- Observation angle θ

For each passenger that is generated, seven parameters need to be determined. The first parameter, flight refers to a flight from the flight set identified in the airport parameters above. The second parameter, arrival time, refers to the time the agent arrives at the airport, and has to be before the flight time for a departing passenger and after the flight time for an arriving passenger. Then, the luggage set is defined for the passenger. For all luggage, the type and threat level needs to be set as well. Two Boolean parameters need to be set for the passenger: checked-in (referring to the passenger being checked-in or not) and facility visitor (referring to the passenger visiting facilities). Finally, two observations parameters need to be set: observation radius and angle. These refer to the radius and angle the passenger can observe in.

3.2.2 Operator Parameters

As there exist two operator types, two operator parameter types are distinguished: check-in operator parameters and security operator parameters. These are discussed below.

Check-in operator

A check-in operator needs a single parameter to be set, listed below.

- Flight set F

The flight set refers to the set of flights the operator can interact with. It is a subset of the flight set defined in the airport parameters.

Security operator

Two parameters are identified for the security operator. These are listed below.

- Activity assignment s
- Threat level threshold l_{threat}

The activity assignment needs to be determined for each security operator in the model. It refers to the specific activity it is performing. Further, for the security operator that executes the X-Ray activity, a threat level threshold has to be set. This threshold is used to determine if the luggage under investigation needs to be checked.

Chapter 4

Assumptions

This section contains a list of assumptions in this model. It contains the assumptions for each of the elements in the model. While this list is aimed to be exhaustive, it might not be complete.

- Flights have a fixed time, gate and set of check-in desks that does not change over time.
- Luggage has a complexity and threat level.
- Sensors are 100% accurate at all times.
- Sensors are always working.
- Passengers knows in what area it is at all times.
- Passengers always travel alone.
- Passengers always follow the shortest path towards their goals.
- Security operators always performs an extra check on luggage if the threat level is above a threshold.
- Security operators never deny passengers entrance to the secure area.
- All passengers pass through the WTMD.
- The goal of an agent is represented as finishing activities before a certain time.
- Passengers plans all their activities on arrival.
- Check-in operators can check-in passengers at all times.

Bibliography

[1] Bruce M Blumberg and Tinsley A Galyean. Multi-level direction of autonomous creatures for real-time virtual environments. In *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pages 47–54. ACM, 1995.

[2] Tibor Bosse, Catholijn M Jonker, Lourens Van Der Meij, and Jan Treur. A language and environment for analysis of dynamics by simulation. *International Journal on Artificial Intelligence Tools*, 16(03):435–464, 2007.

[3] Xiao Cui and Hao Shi. A*-based pathfinding in modern computer games. *International Journal of Computer Science and Network Security*, 11(1):125–130, 2011.

[4] Daniel Damir Harabor, Alban Grastien, et al. Online graph pruning for pathfinding on grid maps. In *AAAI*, 2011.

[5] Dirk Helbing, Illies Farkas, and Tamas Vicsek. Simulating dynamical features of escape panic. *Nature*, 407(6803):487–490, 2000.

[6] Serge P Hoogendoorn and Piet HL Bovy. Pedestrian route-choice and activity scheduling theory and models. *Transportation Research Part B: Methodological*, 38(2):169–190, 2004.

[7] Craig W Reynolds. Steering behaviors for autonomous characters. In *Game developers conference*, volume 1999, pages 763–782, 1999.

[8] Ron Sun. The importance of cognitive architectures: An analysis based on clarion. *Journal of Experimental & Theoretical Artificial Intelligence*, 19(2):159–193, 2007.