TITAN

Turnaround **I**ntegration in **T**rajectory **A**nd **N**etwork

Project Number: 233690

Single Aircraft Turnaround Model Software Design Document

CLASSIFICATION: PU **ISSUE:** 1.0 **DATE:** 15/04/2012

DOCUMENT REFERENCE				
Project	**Work Package**	**Partner**	**Nature**	**Number**
TITAN	WP 2	CRI	DEL	02

Project co-funded by the European Commission and TITAN consortium.

DOCUMENT CONTROL			
Responsible	**Organisation**	**Name**	**Date**
Author	CRI	Andrea Villa/ Eva Puntero/ Ian Crook	23/11/2011
Partners involved	ISA	Tarja Kettunen/ Matthew Bray/ Ian Crook	30/11/2011
Reviewer	CRI	Nicolás Suárez	23/11/2011
	INE	Sara Luis	23/01/2012
	JEP	Alicia Grech	23/01/2012
	BLU	Steve Zerkowitz	23/01/2012
	RWT	Athanasios Katsaros	23/01/2012
Approver	INE	Laura Serrano	01/03/2012

DOCUMENT CHANGE LOG			
Issue	**Date**	**Author**	**Affected Sections / Comments**
0.1	23/11/2011	Andrea Villa/ Eva Puntero	All
0.2	30/11/2011	Ian Crook / Tarja Kettunen / Mat Bray	All
0.3	17/01/2012	María Ruiz/ Andrea Villa/ Eva Puntero	All/Partners review
0.4	01/03/2012	Eva Puntero	All/Update according to Laura Serrano comments
1.0	15/04/2012	Ana Sáez	Final Version after EC Approval

DOCUMENT DISTRIBUTION		
To/Cc	**Organisation**	**Name**
To	EC	Remy Denos
To	INECO	Laura Serrano
To	CRIDA	Nicolas Suarez
To	CRIDA	Eva Puntero
To	CRIDA	María Ruiz
To	CRIDA	Irene Navarro
To	INECO	Ana C. Sáez
To	ISA SW	Ian Crook
To	ISA SW	Matthew Bray
To	ISA SW	Zak Tibichte
To	ISA SW	Tarja Kettunen
To	Jeppesen	Alicia Grech
To	Jeppesen	John Butcher
To	Blusky Services	Steve Zerkowitz
To	RWTH Aachen University	Sebastian Kellner
To	RWTH Aachen University	Athanasios Katsaros
To	Slot Consulting	Roland Gurály
To	Slot Consulting	Noémi Král
To	Slot Consulting	Balazs Kerulo
Cc	SESAR JU	Paul Adamson
Cc	AENA/SESAR JU	Alejandro Egido
Cc	AENA/SESAR JU	Francisco Javier Fernández de Liger

TABLE OF CONTENTS

LIST OF FIGURES

EXECUTIVE SUMMARY

The present document contains information about how the software of the *Single Aircraft Turnaround Model* should be built. The Model Software development will use the information details provided on it to build the model architecture and its associated data structures. Within this Software Design Document, there are narrative and graphical information of the software design that comprises the data model structure and the class diagrams.

1. INTRODUCTION

1.1 Purpose

The purpose of this document is to provide a description of the design of the *Single Aircraft turnaround model*. This description will provide to the software development phase with an understanding of what is to be built and how it is expected to be built.

For this particular Software Design Document, the focus is placed on the high level logic of the model as well as on the data structures that the model will manage.

1.2 Document Structure

This document has four sections:

- The first section is this introduction which presents the purpose, structure, audience, references and abbreviations and acronyms of the document.
- The second section introduces the Single Aircraft turnaround model.
- The third section identifies and describes the data model structures.
- Last section deals with the model class diagrams.

1.3 Intended Audience

This document may be distributed freely within the TITAN consortium, both to those who are involved in the use of the model as well as stakeholders to check the consistency of the model.

1.4 Associated Documentation

[1] TITAN Annex I – “Description of Work”, INECO, Version 0.5, March 2011

1.5 Abbreviations and Acronyms

ATC	Air Traffic Controller
CDM	Collaborative Decision Making
EIBT	Estimated In Block Time
EOBT	Estimated Off Block Time
IBT	In Block Time
GPN	Graphical Process Network
OBT	Off Block Time
TITAN	Turnaround Integration in Trajectory and Network
V&V	Validation and Verification
WP	Work Package

2. SINGLE AIRCRAFT TURNAROUND MODEL

2.1 Design goals

The principal objective of the TITAN Single Aircraft Model software is to provide a flexible software model which can be used to emulate the processes and resources required to perform the turnaround of an aircraft at any given airport. This model will serve as a tool for the TITAN project WP3, where the TITAN operational concept will be validated.

In short, the definition of a single aircraft turnaround is *the sequence of operations that are required from the time that an aircraft is stationary at the airport with the wheel chocks in place, to the time that the chocks are removed and the aircraft is cleared to leave by ATC*.

In reality the turnaround process is made up of a complex set of often interdependent processes, many of which require the use of a limited set of shared resources, and which can vary widely in when and how the processes are carried out.

In the TITAN project, our objective is

> "*to build a model of a single aircraft turnaround that is flexible enough to be able to represent all of the variations, complexities and interdependencies that may exist during any given turnaround*". [1]

Additionally, a second objective is that

> "*the resulting 'single aircraft turnaround model' shall be able to be replicated and operate in parallel to represent the multiple turnarounds that are occurring every day at any given airport*". [1]

Finally, to allow us to model the additional concepts identified during the TITAN project,

> "*the resulting TITAN turnaround model should also include additional modelling features to represent TITAN information flow systems and allow users to evaluate benefits and performance indicators available through use of the additional TITAN services*" [1]

Given the large number of factors that characterize a turnaround, it is difficult and challenging to design a specific model which corresponds to each and every possible type of turnaround. The model will therefore be based on a generic solution that can be adapted quickly and easily by its users to allow it to represent many different types of turnaround.

There are a number of key tasks that make up a typical turnaround, including the unloading and loading of passengers and cargo, preparation of the aircraft for the next leg of its journey, pre and post flight safety and security checks, etc. However servicing arrangements and turnaround activities can vary considerably at different airports, in different parts of an airport or for different airline operators, based on their own policies or service level agreements and also depends on the type of flight (e.g. Schengen, Non Schengen). What is common to all turnaround operations, however, is the need for all of the processes and operators involved to work in a coordinated and efficient manner.

For modelling purposes and to cater for our high level design objectives stated earlier, we consider the turnaround from an abstract point of view, designing it as:

- a network of interdependent processes,
- to be carried out in a given sequence, or in parallel,
- that may be unpredictable in the execution time,

- can be subject to random interferences and,
- that may require access to limited shared support resources.

For this reason, the aim is to design a model that is made of a network of processes and which can achieve the objectives stated above, to provide a powerful and flexible capability through which to model any conceivable turnaround.

2.2 Model overview

2.2.1 Design concepts – flexible and extensible

To cater for the enormous variety and complex interdependencies that make up a turnaround, the Single Aircraft Turnaround Model has been specifically designed with flexibility and extensibility in mind.

The model is created as a network of the tasks and processes that will be required to implement the desired turnaround. Generic entities which can be characterized to represent real world elements that are involved in a turnaround process, such as passengers or bags, are fed into the model and travel through the network of nodes and links as the turnaround processes become active. Once all the processes have been completed for the defined set of entities, the turnaround operation is complete.

The model has been conceived from the outset to provide highly flexible, generic representations of both the tasks to be fulfilled and the elements that are involved in the turnaround – in this way the model can be configured by experienced users to represent any turnaround type in a huge range of definition levels.

For example, in a simple form, a turnaround could be described thus:

- Aircraft arrives in block
- Unload arriving passengers and cargo
- Prepare aircraft for next flight leg
- Load departing passengers and cargo
- Issue ATC clearance to depart
- Aircraft pushes back and leaves

Entities that are involved in the turnaround might be:

- Passengers
- Bags
- Other cargo
- Fuel
- Etc

Along with these, some shared resources may be required, for example:

- Boarding jetway / steps / bus etc.
- Tow Truck

- Fuel Truck
- Etc

This high level definition describes a simple turnaround process based on 6 high level processes, each of which will require some finite time to complete. The complexity of this turnaround can be gradually increased by adding new processes or sub-processes (e.g. re-fuelling or de-icing); by combining different types of entities (e.g. On line checked-in, special or VIP passenger); increasing the complexity of the shared resources definition (e.g. time consumed depending on the number of passengers or type of aircraft) or considering more shared resources (e.g. boarding agents, special assistance).

Even without formally linking them, it is easy to identify an implicit sequence for some of the processes – for example, departing passengers and cargo cannot normally be loaded until arrival passengers and cargo have been unloaded.

Finally, some of the processes could be executed in parallel. For example the aircraft could be prepared for the next flight leg at the same time that the departing passengers are boarding and the cargo is being loaded.

2.2.2 **Functional Elements**

The important point to draw from the example in section 2.2.1 is that even for such a simplified turnaround schema, there are a number of different ways that can be identified for the manner in which that turnaround is executed (two of which will be illustrated later in this section). To design a model which can capture every possible variation would be difficult, so the Single Aircraft Turnaround Model is based on a combination of techniques:

- ***Actors and resources –*** to define the actors/resources that may be involved in the process and their characteristics, the time they consume and how they behave in the processes.
- ***Process Mapping*** – to define the different processes in each turnaround, how they are carried out (characteristics), and which are the conditions required for this process to happen. Also duration, actors involved and how they are involved can be defined.
- ***Discrete event simulation*** – to permit the time during the execution of the simulation to change as the simulation progresses (see section 2.2.5).
- ***Activity based costing*** – to allocate specific costs (usually in terms of a process execution time) for objects (activities, services, projects or products) in the network of processes, and if necessary, to require that those activities use resources which may come from a pool of shared resources.

The model software is designed as follows:

First of all; task, processes, actors and resources are defined in the model.

Task and processes are defined including their times and required resources, if needed, to be carried out, including how many they need, when and how they need them to behave.

Actors and resources are also defined, how many they are and which is their capacity.

Afterwards relationships between tasks and processes are defined, giving their inputs, triggers and outputs. Triggers can be times or start/end of other tasks/processes. This defines the turnaround sequence, however, depending on the type of flight (e.g. Schengen, Non Schengen) different turnarounds processes are defined.

Combining these features, an interactive ***Graphical Process Network definition (GPN)***, can be built to create models of the aircraft turnaround process with different levels of detail as illustrated in figures 1 & 2 below:

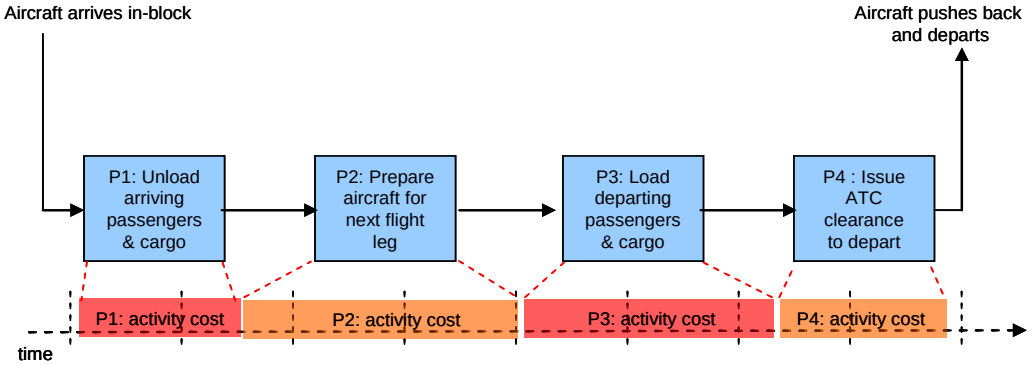


Figure 1: Sequential process network for simplified turnaround model

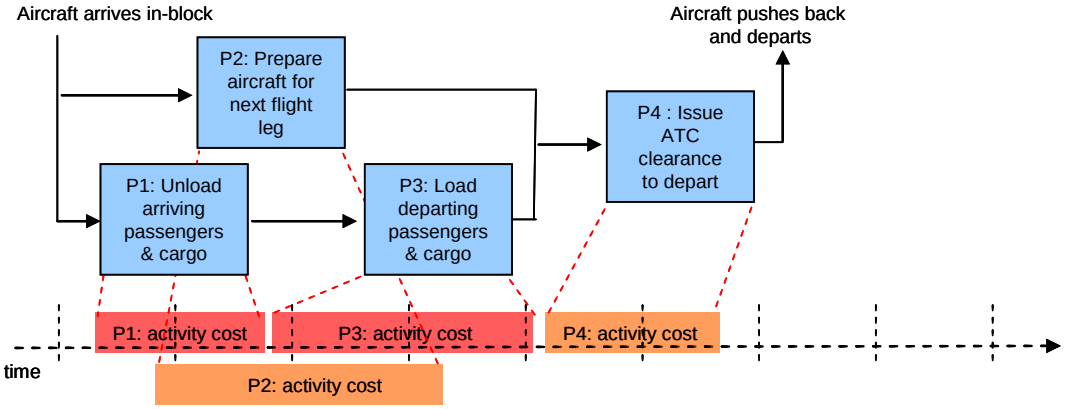


Figure 2: Parallel process network for simplified turnaround model

Using the GPN definition, each process in the turnaround can be defined in detail by a series of links and nodes which create a process network. Generic entities (data elements) such as passengers, bags, cargo, etc. can flow through that network during the turnaround process.

By design, the level of detail for each process network is entirely flexible: users can define processes from a very abstract level, perhaps simply requiring an execution time that is randomly sampled from a given time distribution to a high detail. For example, a process such as the P1 (Unload arriving passengers & cargo) in the Figure 1 and the Figure 2 could be defined in greater detail, specifying each step that is necessary to carry out the process and capturing a particular airline company operating policy.

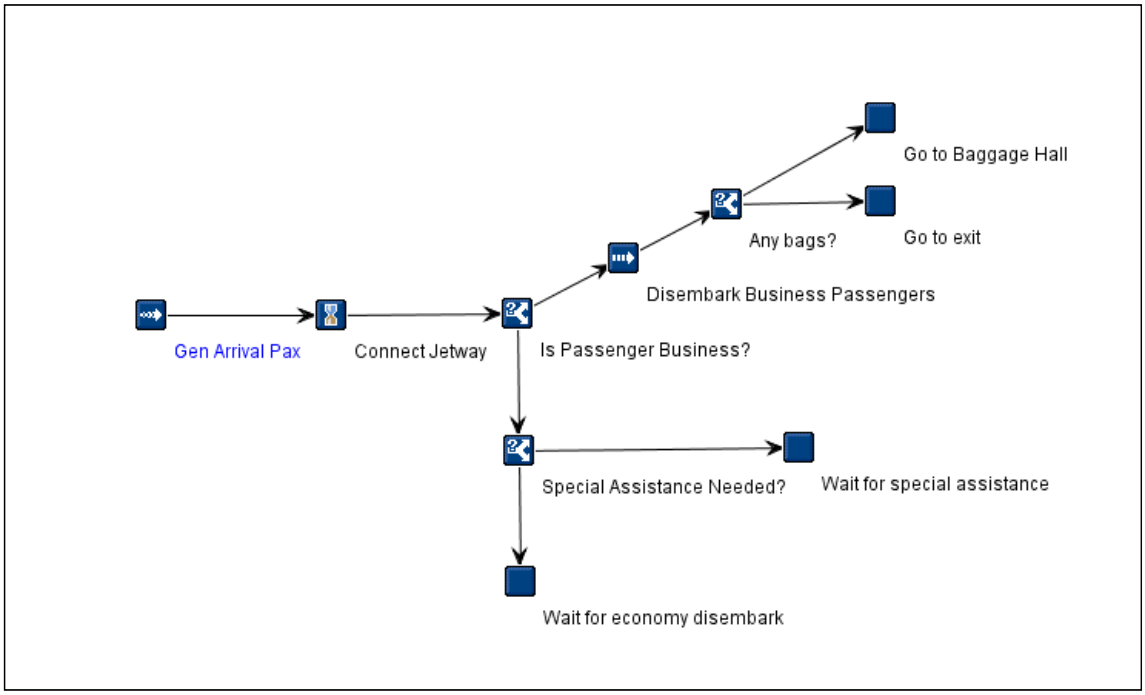


Figure 3: Detailed graphical network for a sub process P1: Unload Arriving Passengers

Figure 3 shows an example where passengers are allowed to disembark at different times depending on the type of passenger.

Data entities that will traverse the specified process network are generated by the generator node (see section 2.2.3 for nodes description) as 'passengers' based on the arrival manifest data that is specified for the flight undergoing the turnaround.

According to Figure 3, arrival passengers are not able to leave the aircraft until the jetway is connected which requires a finite time. Once it is connected, the policy of this airline it that passengers disembark at different times, starting with the business passengers. To achieve this behaviour in the model, the jetway is represented as a delay node ("connect jetway" node) where the delay node parameter is set so that it is only applied for the first entity arriving at the node. This way, only the first departing passenger produced by the generator node will be delayed (until the jetway is connected). All subsequent entities arriving at the delay node do not receive any further delay. Thus the delay node is able to model the actual time that is needed to physically move and connect the jetway.

Once the jetway is connected, arrival passengers begin to flow through the network and branch down different legs according to data characteristic of the arrival passenger entity (e.g. Business class, Economy class, Special Assistance etc).

At the end of the process, the arrival passenger entity is transferred to one of four possible follow on processes (Go to Baggage Hall, Go to exit, Wait for special assistance or Wait for economy disembark).

To capture the specific airline operating policy, i.e. allow business passengers to disembark first, while economy class passengers are required to wait until all business passengers have left the aircraft and special assistance passengers leave the aircraft last, inter-dependencies can be defined between subsequent processes.

For example the process "Wait for economy class disembark" would not be allowed to start until the process shown in Figure 3 is complete. Arrival passenger entities of type 'economy class'

would still be transferred to that process but would be queued at the entry point until the process becomes active. Similarly, the 'Wait for special assistance' process would be dependent on the completion of the "Wait for economy disembark" process to prevent special assistance passengers from disembarking until all economy class passengers were off the aircraft.

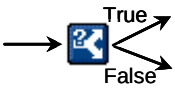
2.2.3 Graphical Process Network Links and Nodes

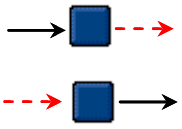
In the previous section we have been introduced to the concept of the "Graphical Process Network' (GPN) and seen an example how a complex set of tasks can be modelled using the features of those networks.

The key to achieving the underlying design objectives of flexibility and extensibility in the model comes from the GPN and the nodes that are available therein.

Each node provides functional behaviour that is applied when a data entity arrives at that node. Data entities are created and injected into the network using generator nodes. They will travel through the network along the links that connect successive nodes, can jump between successive process networks using the inter-process transfer (send and receive) nodes, and continue until they are disposed at the end of the path.

Eleven (11) different node types are supported in the model:

Node type	Symbol	Functionality
Batch		Consumes entities and adds them to a batch container which will only continue through the network when it satisfies one of the specified conditions, e.g. capacity achieved, last entity arrived, time offset etc.
Concurrency		Duplicates the incoming entity and sends one copy down each branch
Conditional		Tests a user specified logical condition and sends the incoming entity down one of the output paths depending whether condition is true or false
Delay		Consumes an entity and applies a specified time delay before releasing the entity to the output branch
Dispose		Consumes an entity and disposes it
Generate		Generates entities based on scenario data elements and sends entities down the output branch in a timed stream
Junction		Permits entities that have previously followed conditional branches to rejoin the main branch and continue
Queue		Joins input entities to a queue to be processed. At the head of the queue a user specified activity cost (time delay) is applied before the entity is released down the output branch
Resource		Holds the incoming entity until a specified resource is acquired from the resource pool, then sends the entity down the output branch.

Node type	Symbol	Functionality
Transfer		Transfers: 'Send' option transfers the incoming entity to a user specified receiver node with an optional user specified transfer delay. Transfer: 'Receive' option takes the entity that was transferred from a previous GPN and sends it into the new GPN process via the output branch, provided that the current process is active.
Transform		Transforms the incoming entity into a different outgoing entity and sends the new entity down the output branch. Also disposes the incoming entity for completeness.

2.2.4 Graphical Process Network Node Properties

The combination of process mapping, using GPN definitions to specify which tasks and activities should happen and in which order during the execution of a given process is further supported by the ability to customise each node of the network according to user defined parameters. Using these parameters provides the support to introduce different behaviour in the model for different types of turnaround operations – through activity based costing.

Each type of node has its own customisable parameters which can be defined through the user interface or input data files. The remainder of this section will specify which parameters exist for each type of node and how the parameters impact the functional behaviour as an entity traverses that node.

2.2.4.1 Batch Node

The batch node is designed to capture operations where a number of arriving entities are grouped together in a 'batch entity' (*container*) which once full, is transferred to a different part of the GPN. Examples of batching activities include loading passengers onto a bus, to transport them to a remote aircraft stand, or loading bags onto a baggage truck to be taken to the aircraft, and then be loaded into the cargo bay.

Each batch node is therefore specified with a type of entity which it will contain, along with the maximum capacity for the container. For example a bus will contain a set of passenger entities and might have a maximum capacity of 60 passengers.

By default, once a batch node 'container' reaches its maximum capacity, the entire set of entities that have been added to the container are transferred down the output link as a single 'batch entity' (e.g. a bus containing 60 passengers).

However, there are occasions that a batch node might need to be released before it reaches its maximum capacity, for example if the last passenger is loaded onto a bus (and the total number of passengers is less than the bus capacity) or at a given time before the aircraft is due to leave (even if all the passengers are not yet on the bus).

The batch node therefore provides additional parameters to allow the user to specify different behaviour, in particular to support

- Release of the batch entity at a given offset time from a planned downstream event
- Release of the batch entity for a given input condition (e.g. last passenger loaded)

As the batch entity travels through the remainder of the GPN, it will be treated as a single entity (e.g. a bus, a baggage truck, etc.). On arriving at the target node, the user should specify how to manage the batch entity. For example, on arriving at the aircraft, the bus is not usually loaded onto the aircraft – rather the bus doors are opened, provided that there are steps available for passengers to board the aircraft, and the air crew & ground crew are available to supervise the boarding, then the passengers will leave the bus and board via the aircraft steps. However, bags heading to the aircraft may be loaded in different ways, once batched together. Those on a baggage truck might be taken off one by one and loaded into the cargo bay using a conveyor belt. Others might have been placed into a larger cargo container which is specifically designed to be loaded directly into the cargo bay, thereby not needing to unload and reload the contents one by one.

To allow the system to manage the different ways of treating a batch set of entities, the batch node is also designed to have a 'un-batching' feature. Un-batching therefore takes the entities contained in the current batch entity and releases them one by one into the output link of the batch node. (e.g. representing the passengers as they get off the bus).

2.2.4.2 Concurrency Node

The concurrency node is not designed with any user specific parameters. It has the sole function of replicating the incoming entity to produce two identical copies, one on each output link. The usual aim of the concurrency node is to allow compound entities to be separated to follow different paths in the GPN.

For example, a passenger arriving at check-in with 2 bags to check in would hand the bags to the check-in agent after which the bags would follow one part of the turnaround process (e.g. through baggage handling) while the passenger goes to another part (e.g. security). To model this, the GPN would use a concurrency node in the check-in process to replicate the passenger entity, followed by a transform node on one of the two branches to convert the passenger with 2 bags into new entities (bags) which can then be sent into the baggage handling process. The original passenger entity can be sent into the next downstream process (e.g. security) without its bags.

2.2.4.3 Conditional Node

The conditional node is designed to allow user to create logical branches in a GPN, where entities input to the node are subjected to user-defined logical tests (returning true or false) and branch according to the result of that test.

The logical tests associated with a conditional node can be programmed to use fully functional custom code written in the '*Groovy*' programming language (see http://groovy.codehaus.org/ for full documentation). These custom code snippets can access any of the available system entities and their attributes to perform the desired logical test.

2.2.4.4 Delay Node

The delay node is designed to allow a delay to be introduced within a GPN which is not linked to the execution of a specific task in the process (i.e. is in addition to the activity-based cost associated with the execution of a certain task in the network process, e.g. Check-in, Baggage-drop etc). Delay nodes are designed to capture the time needed to get between different parts of the airport, or to carry out specific actions which are not being modelled in detail and which do not require entities to be processed in a sequence, supported by a queue. For example, following check-in, there may be a finite distance between the current check-in desk and the entry to security which takes a finite time to travel for each passenger.

The delay node can be used to define a fixed or variable/random travel time between the two locations. Fixed delays will be applied to each entity entering the node before releasing them. Random/variable delays are taken from a user-defined distribution (either a boolean or a normal distribution) which are sampled each time that a new entity arrives at the node.

Additionally, as seen in the example elaborated in figure 3 previously, a delay can be included on a one-time basis. In this case, the delay will only be applied when the first entity arrives at the input branch of the node, and once the first delay has been applied, all other entities arriving at the node will continue unimpeded.

Custom code may also be used to include different amounts of delay according to characteristics of the entity or the system. For example, a passenger requiring special assistance may require more (or less) time to move from one part of the airport to another - to represent this situation a custom code snippet which interrogates the passenger special assistance attribute could be used to adapt the delay as required.

2.2.4.5 Dispose Node

The dispose node has no specific parameter associated with it. The node is designed to remove and delete entities from the GPN, and is used to dispose of those entities when they reach the end of the process network.

2.2.4.6 Generate Node

The generate node is used to create and introduce entities in the GPN. In the design of the single aircraft turnaround model, it was decided to limit which data and entities can be used by a generate node to passengers in the aircraft departure and/or arrival manifest, or the scheduled aircraft operation itself. In this way, the generate node is able to access the list of passengers which will travel through the system during the turnaround, and inject them into the network according to timing parameters specified in the turnaround control parameters. All subsequent processes or acquisition/release of any required shared resource in support of turnaround will be triggered as part of the discrete event simulation as time progresses and as each entity arrives at the different nodes included in the GPN.

The generate node is also designed to allow it to access the details and attributes of the aircraft involved in the current turnaround operation, for example the type of aircraft, its destination, Estimated Off Block Time etc.

2.2.4.7 Junction Node

The junction node is not designed with any customizable parameters and fulfils the role of recombining entities that have previous branched down different conditional paths to allow them to continue down a subsequent common path segment. For example, one passenger may have gone to either an automated check-in or to a desk staffed by a check-in agent. Once completed, the passengers would be joined back together in the same flow using a junction node, then proceed to the same security zone where they may queue one behind the other to be processed.

2.2.4.8 Queue Node

The queue node implements the GPN activity-based cost model. Each queue node represents a physical activity that needs to be completed as part of the turnaround process (e.g. check-in, get boarding card, pass security, load bag onto conveyor belt, etc.). The queue node has an associated cost which is specified as either a fixed or variable time that is needed to carry out the

activity (e.g. it requires a random time of t seconds, sampled from a normal distribution with mean of 5-minutes and variance of 1.4 minutes to complete the check-in activity). The activity cost is applied once the entity reached the head of the queue. If other entities are currently being processed or are queuing for the current activity, the incoming entity will join the back of the queue and remain until it reaches the front, where it will also be processed.

2.2.4.9 Resource Node

The resource node is designed to allow the current process to search for and acquire shared or mobile resources within the system being modelled.

Resources can be defined by the user and allocated to a global resource pool, from which they can be acquired to support a given process. Examples of a resource that might be shared can include airport resources such as buses, baggage trucks, aircraft steps etc, support staff such as ground agents, security agents, customs officers etc or any other resource that a user wishes to include in a turnaround modelling scenario.

When an entity arrives at a resource node that is included in the GPN, the system will try to obtain a resource of the defined type from the shared resource pool. Resources can be acquired for each and every entity passing the resource node or for a user defined period when an entity arrives at the node. In both cases, if the resource pool does not have any available resources of the requested type, the process will wait until one becomes available, then acquire it for the node to allow the entity to continue through the network. Each resource may have one or more travel times associated with it – representing a variable time that is required for the resource to move from a different location in the airport to the current place.

Alternatively, a shared resource can be allocated and fixed to a given entity, for example, allocating a wheelchair for a passenger with reduced mobility. The resource is then transferred through the GPN until another resource node is traversed, somewhere downstream, which releases the shared resource from the entity.

Note that the design of resources and batch entities are complementary in nature, but have opposite logic – one or more resources can be acquired by an entity to be either used then released or held by the entity until a downstream point or time. A batch node gathers multiple entities and holds them as a group until either reaching its declared capacity or a given release time, at which the entire set of entities will move onward through the network until reaching an un-batch node where they are separated once more, or being disposed (as a group).

2.2.4.10 Transform Node

The transform node is a feature that is designed to receive an incoming entity and, using the entity properties and attributes can output a new entity. The objective is to allow the user to ‘separate’ parts of entities and send the resulting new entity onward through the network. For example a passenger entity used as input to the transform node, could be output as a baggage entity.

The transform node is designed to allow access to the incoming entity properties, which can be copied into the outgoing entity as a new attribute of that entity. For example, an incoming passenger with 2 bags and departing on flight DLH123 can be transformed into a baggage entity for passenger X comprising 2 bags and destined for flight DLH123.

Transform nodes also have access to the current turnaround operation giving additional access to the properties of the turnaround itself and the current flight which is undergoing the turnaround operation.

2.2.5 Discrete Event Simulation

The TITAN simulation engine is designed using a discrete event simulation engine. This is a type of simulation engine where events within the simulation are scheduled to occur at particular time in the future and each event is tied to a piece of code to be executed once that time arrives.

The engine is comprised of a *clock*, an *event list* and the *event scheduling and dispatch* system.

The *scheduling-dispatch* system acts upon the head of the *event list* which is always equal to the current time. Each piece of code that is tied to the event being processed is able to create new events on the schedule either at the current time, or at some time in the future. Once all events at the current time are exhausted the clock steps to the next discrete point in time (the next position in the list) and events that are scheduled at this new time in the simulation are executed.

A simulation continues to run until there are no events left in the system to dispatch, at which point the simulation is considered to be complete.

The simulation engine itself is not domain specific allowing development of different applications upon it.

The TITAN model will use the discrete event simulation engine to introduce and process events for operations (IBT, EOBT etc), processes (expected start, end time etc), nodes (generation, sending of entities, delays) and milestones (expected occurrence).

2.2.6 Software Design Approach

The model is designed for use on a standard desktop computer due to a large part of the project being design and validation of scenarios with a graphical editor and post-processing tools.

The Java programming language running on a Java 1.6 compatible Virtual Machine (JVM) is used to encode the model on the standard Windows operating system. It should, however, be portable to other systems with a supported 1.6 JVM (e.g. Linux, Mac OS X).

Additional scripting within the simulation engine (aka *custom* code) is possible within the Groovy programming language which also runs on the same version 1.6 compatible JVM.

There are three distinct parts to TITAN:

- Data - The TITAN data model and data format (see next section).

  TITAN has a text-based scenario format and facilities to load and save it. Internally TITAN maintains an object-oriented representation of the data. From this static data a simulation engine can be created and run.

- Simulation - The TITAN discrete event simulation model (described previously in this section).

  The simulation is generated from the static data set. It is 'bootstrapped' with initial timings generated from this data set and a set of random number generators are initialised. Once running the random generators will allow divergence between different scenario runs.

  The model contains three functional processes - the operation, the processes to be run for the process (which depend on turnaround type) and the nodes that are to be active in the process.

  During running decision making is divided between those decisions made automatically by pre-defined decision types within process nodes, decisions made through user designed scripts / custom code (developed in Groovy) and decisions that are presented to the user via interactive dialogues. User decision points can be also be configured to use default choices if required.

- Simulation observation - The listeners.

  The actions within a simulation can be observed through one of the many observation interfaces provided. Currently these are used only for two purposes - interaction and visualisation. It is foreseen that in future versions of the tool, additional observers will be written, for example to enable remote or portable devices to access status and milestone information and possibly interact with the simulation as it executes.

3. DATA MODELS

3.1 Introduction

Several data types are needed in order to define the scenarios to be simulated. The following paragraphs provide a description of the data structures that TITAN model will use or produce during or after the simulations.

Two different data structures may be defined according with the logical architecture of the software: those which will feed the simulation while it is running and those that will be produced as data output.

Figure 4 shows the logical architecture of the TITAN model. The inputs are introduced into the model through TITAN interface (display) or by editing the text-based data files. These inputs are used by the simulation manager. The Simulation Manager creates the TITAN Models and fixes the parameters of the simulation.

TITAN Models are composed by several definitions of services and interactions between them, and they model different turnaround operational concepts depending on the configured parameters. TITAN Models run producing new parameters or changing the ones previously fixed by the data input. Simulation Manager sends to the Reporting Engine the results of the simulation. Reporting Engine elaborates and produces the data output of the simulation.

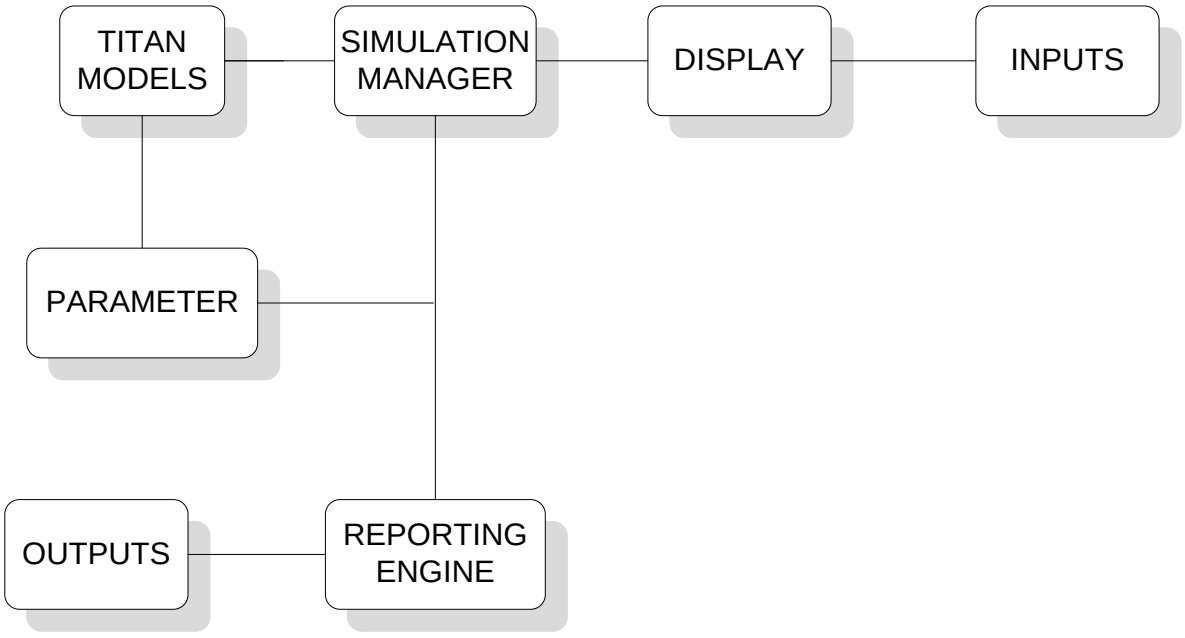


Figure 4: Simulation Logical Architecture.

Figure 5 shows the global data model structuring data into two groups; data inputs and data outputs.

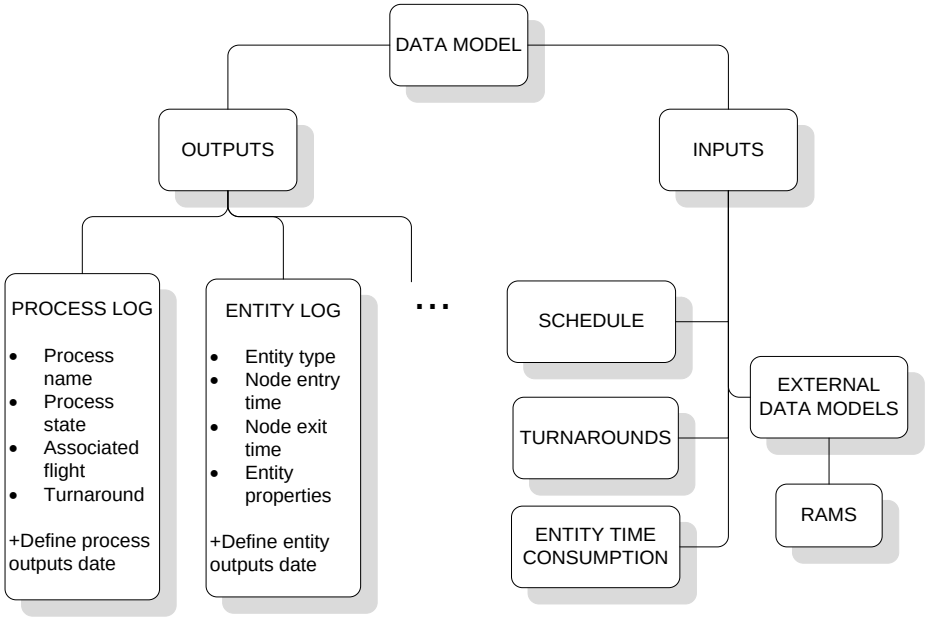


Figure 5: Data structure attending to the model architecture

Some data inputs have to be defined in order to generate a scenario. These data inputs are related to traffic data, processes description, turnaround models and airport resources. They are used by the simulation to model a context of turnaround operations in an airport.

Data outputs are required to provide awareness about the development of the processes during the simulation, as well as their performances, resources used and time consumption. These data outputs will serve to assess the validation objectives.

3.2 Data Inputs

Regarding to data input types, the following classification can be done:

- Data related to schedule. These data will contain information in order to generate a traffic sample. Different type of entities will be created with certain properties.
- Turnaround data. This data will allow defining several types of turnaround models and associated processes. Also several milestones will be fixed to monitor operations and control points will be selected.
- Data related to the time consumption of the entities. This data will allow modelling each entity or process time flow. Several constraints between some of them would be determined.

3.2.1 **Schedule data**

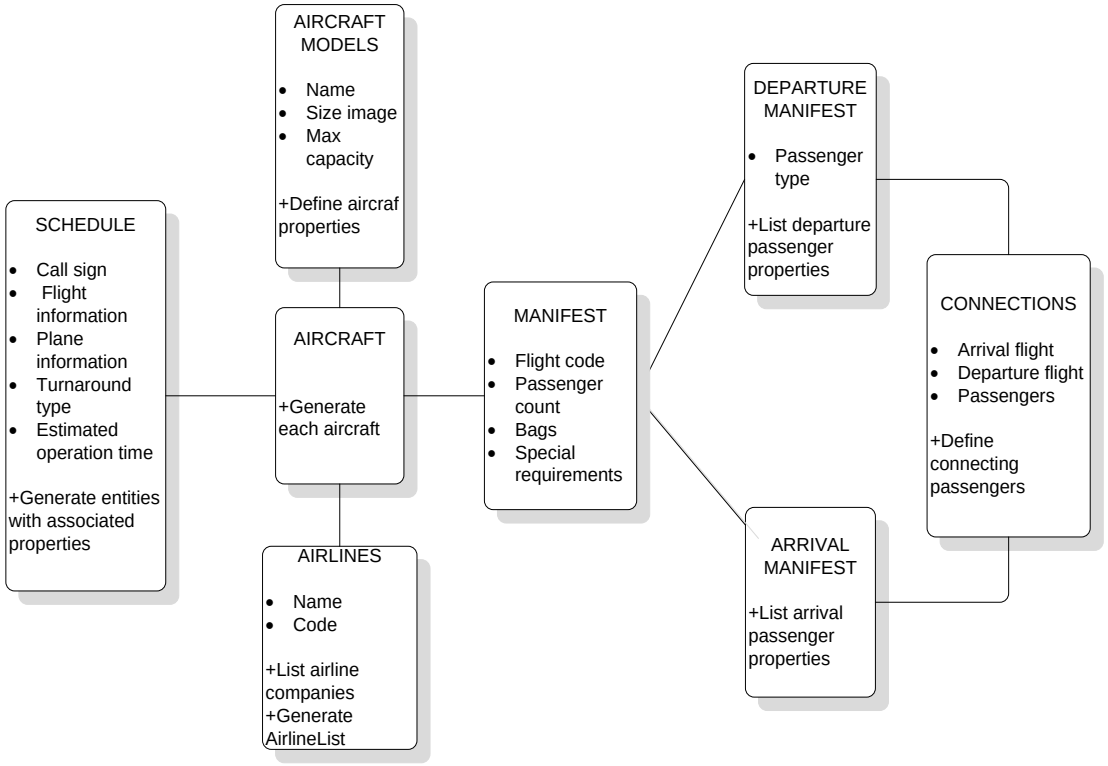


Figure 6: Schedule data input structure

In order to generate each flight the following data items are needed: Aircraft model, Airline, Arrival and Departure manifests and Connections.

Airlines data inputs allow the user to define each of the airline companies present at the airport which is being modelled. Aircraft models data input simply provides some basic information that are required by the model. They are used, for example, to represent the aircraft icons for different aircraft sizes in the model interface.

The arrival manifest defines a passenger list along with passenger properties for each arrival operation found in the traffic schedule. It offers a number of different mechanisms to define the passenger count, number of bags and other special requirements that each passenger may require. The departure manifest is similar to the arrival manifest (in addition to data similar to arrival manifest, it contains information about the amount of passenger doing online check-in). It is a passenger list with passenger properties for each departure operation found in the traffic schedule. Finally connections data input defines the passengers that are connecting from one arrival flight to another departure flight.

3.2.2 Turnaround data inputs

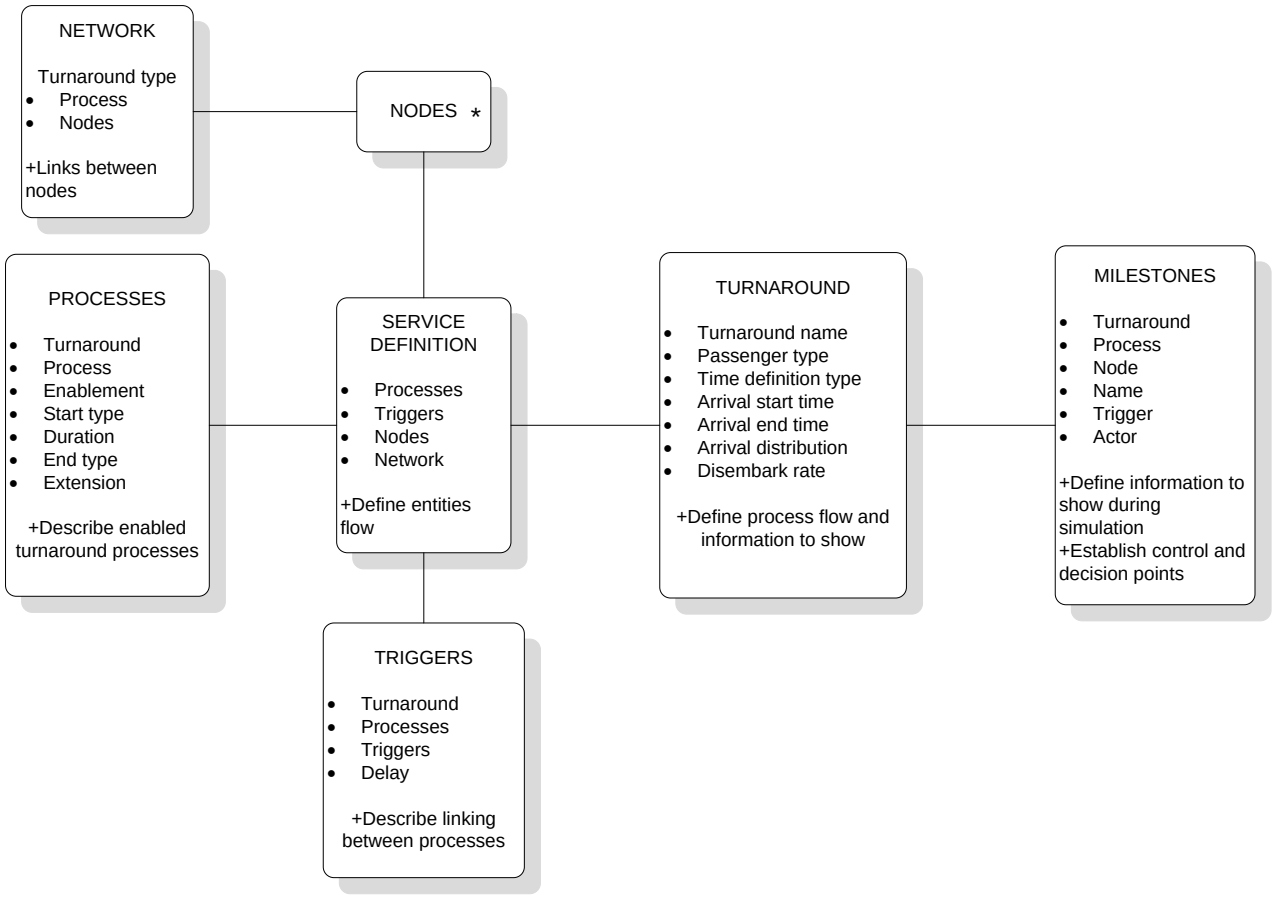


Figure 7: Turnaround data input structure

The turnaround data input allows defining several types of turnarounds available to be allocated to each flight in the model. Available fields are: Turnaround name, standard or online passenger arrival start time type, arrival start time, standard or online passenger arrival end time type, arrival end time, arrival mode, arrival distribution or disembarkment rate.

Each turnaround is composed by several services provided to the aircraft or to the passengers. Turnaround services are those provided between In Block Time (IBT) to Off Block Time (OBT). However the model takes into account also those airport services that have an impact on the turnaround which may begin outside of that time window. The goal of all of them is to prepare aircraft for next flight leg.

Some of these processes are directly related to the aircraft (e.g. catering or re-fueling). Others ones involve entities flow (passengers or baggage) in order to be completed (e.g. boarding is finished when all passengers are at aircraft).

Services modelled are composed by one or several processes. The Process input file describes each standard turnaround process in terms of whether the process is enabled for a given turnaround type, together with start and end duration times (e.g. If catering is enabled it can be defined to start an offset from EOBT or EIBT and to finish after a fixed duration).

A process is defined into the model by a network of nodes. Nodes and links connecting them determine entities flow within each process.

While some processes can be run in parallel, dependencies between processes' start ups and shutdowns may be necessary so specific relationships between processes can be specified using triggers (e.g. baggage loading process starts when baggage unloading process has finished).

In addition, each turnaround type can have several model milestones associated to its processes; milestones in the model are notifications displayed when a process starts or has been completed. Milestones data input also contains the actor who will be notified of each milestone. Associated to milestones, some control and decision points (interaction points) can be defined in order to decide which processes will continue and in what manner.

3.2.3 Entity time consumption data

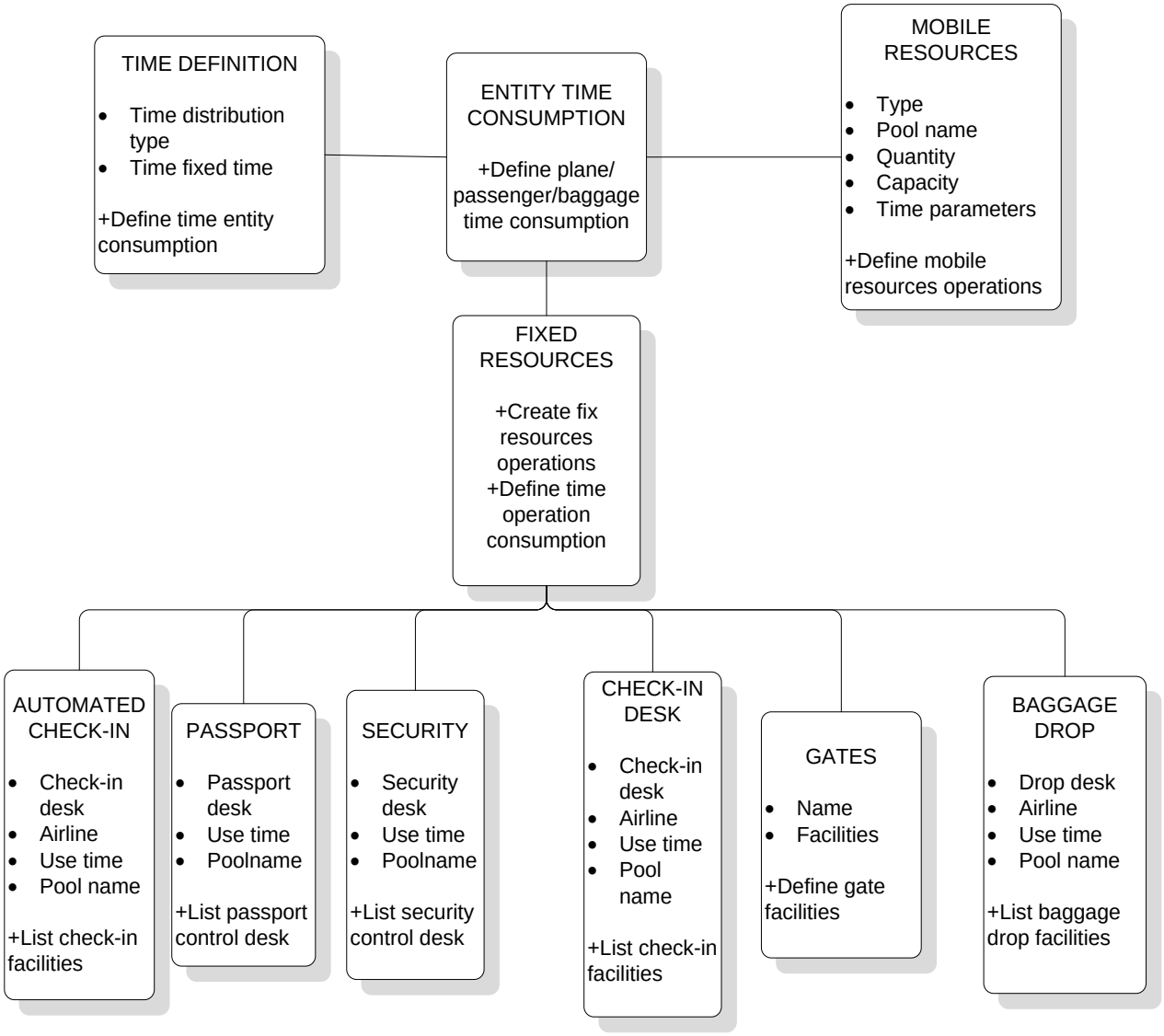


Figure 8: Entity type consumption data input Structure.

Each aircraft, passenger or baggage consume a certain time in the flow of processes. This time can be defined by a fixed time or by a certain distribution.

The operations of a number of different mobile resources are defined in the mobile resources input file. The travel time are intended to model situations where the mobile resources are not always at the gate, but in a random location at the airport. This way, some time will be consumed to get to the required location. Another input file exists to define non-mobile resources. Finally, some fixed airport infrastructure resources can be defined.

3.3 Data Outputs

Data Outputs are provided by means of text files. These files register different events that happen during the simulation in chronological order. There are several files which register different kinds of information:

- Entity log: Makes a register of each entity movement through the nodes.
- Error log: Reports errors happened during the simulation (e.g. wrong definitions of data input)
- Milestone log: Reports the status (passed or missed) of the milestones associated to each flight and the moment they were achieved.
- Node log: Reports the status of the nodes during the simulation for each flight (initialising, running, completed)
- Stats log: This file provides some statistics about the use of the resources such as the number of entities that have passed through a node; average waiting time; minimum and maximum wait time and queue length.
- Operation log: This file reports the status of each flight operation (initialised, on gate, off gate)
- Process log: This file reports the status of each flight operation process (initialising, running, completed)
- Resource log: This file reports the status of the resources used during the simulation (acquired, released)
- Trigger log: Reports the triggers launched during the simulation.

4. SINGLE AIRCRAFT TURNAROUND MODEL DIAGRAM

A global model class diagram is shown in the following figure.

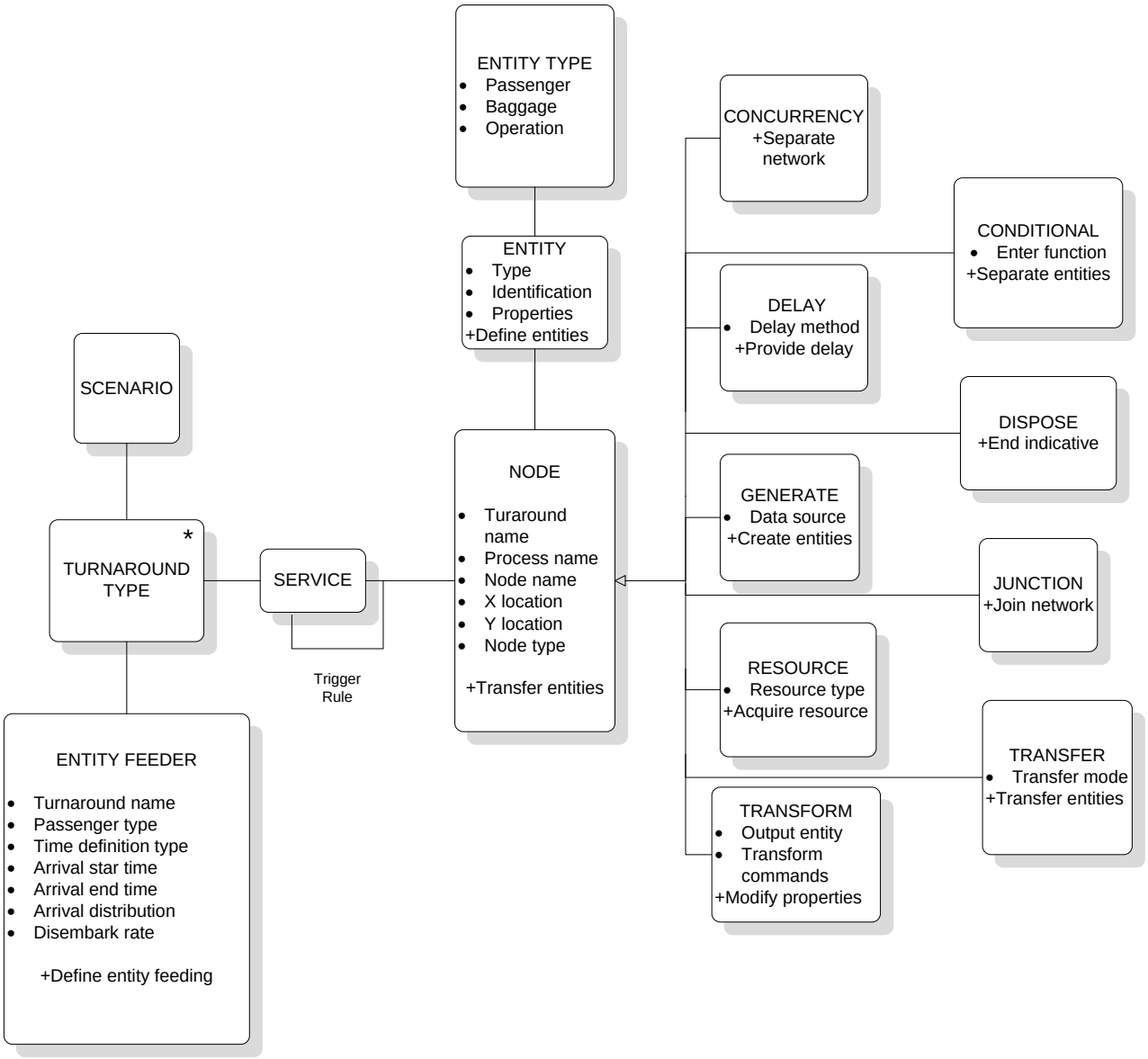


Figure 9: Single Aircraft model diagram

Nodes are central elements in the model class diagram. All of them have several common properties and are used to define the entity flow. Different node types are shown in the figure. Entities use services whose start and end time will be determined by triggers or rules. Finally, each turnaround will have several defined processes that allows to provide a service. Entity feeder will provide all necessary parameters to define the turnaround process.