C S 4 0 5

I N T R O D U C T I O N T O E M P I R I C A L M O D E L I N G

Name: Shahrokh Barati
University Number: 0960653
Professors: Dr Meurig Beynon & Dr Steve Russ

Table of Contents

Shahrokh Barati
0960653

Empirical Modelling
Baggage Management System

Introduction

This project is about simulating a baggage management system through Empirical Modelling techniques such as dependencies and instant variable updates using TkEden tools. The system is developed using Eden, Donald and Scout tools and simulates a simple 2D model of a baggage management system with 3 bags, taking into account priorities and parallel/sequential processing as well as user inputs. Furthermore, the advantages and drawbacks of traditional programming languages compared to Empirical Modelling are addressed and several recommendations on possible improvements are suggested.

Background

Recently, large airports have introduced complex baggage management systems where computerised simulations and models are used to represent the movement of suitcases and automatically determine the required pathways to insure correct arrivals and delivery. However time and time, these systems have failed to do their job successfully and suitcases have ended up in wrong locations which have led to large financial losses and customer dissatisfaction. In order to overcome such issues, simulations should be reliable, consistent and dynamic. Dynamic system would ultimately allow users to overcome problems caused by misclassification of suitcases and correctness of pathways, through manipulating the models and simulations with a fast processing and updating of the system. Empirical modelling techniques could therefore be useful in such cases due to their nature of updating variables instantly, and their effective usage of dependencies in order to ensure reliability.

Industry Usage of Simulation in Baggage Handling Systems

The airport technology website [4] contains numerous baggage handling systems and simulations readily available. The following images are examples of baggage handling systems developed by BCS Group and Babcock airports respectively, using 3D simulations to validate their systems and test scenarios to prove performance.

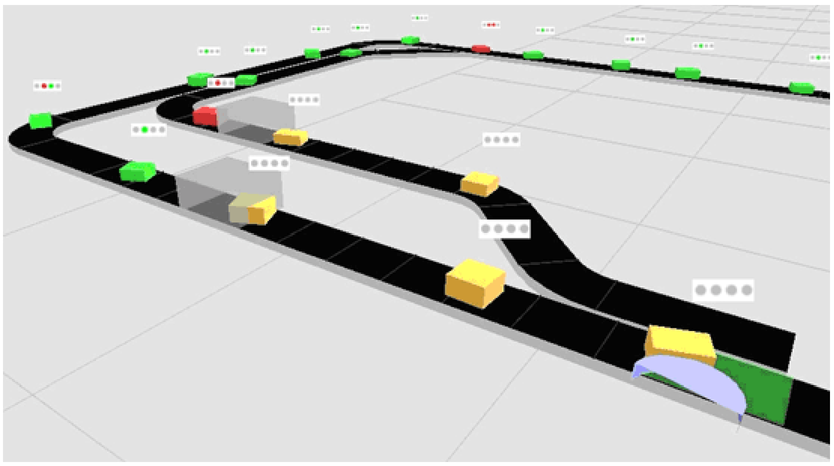


Image 1: Simulation of BCS Groups' Baggage Handling System using 3D CAT software [2]

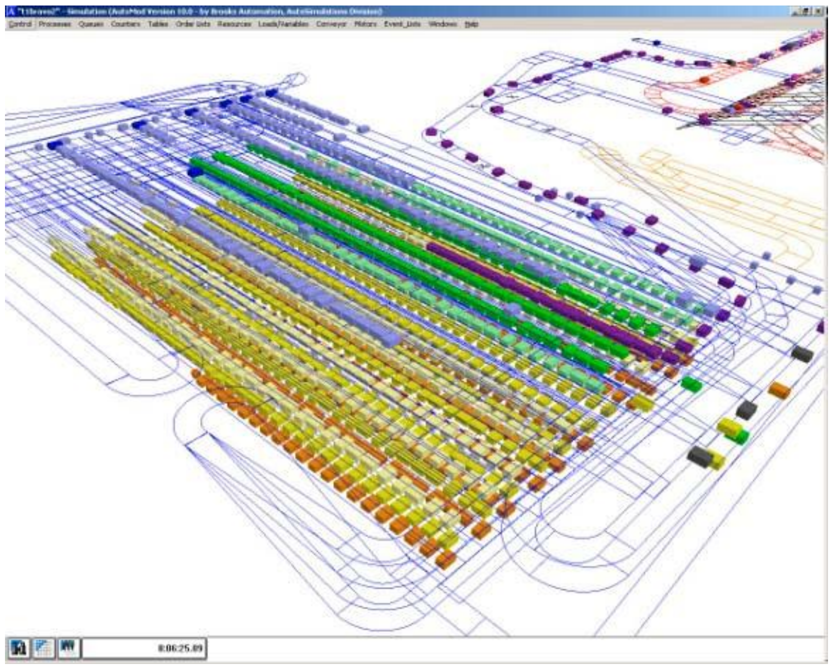
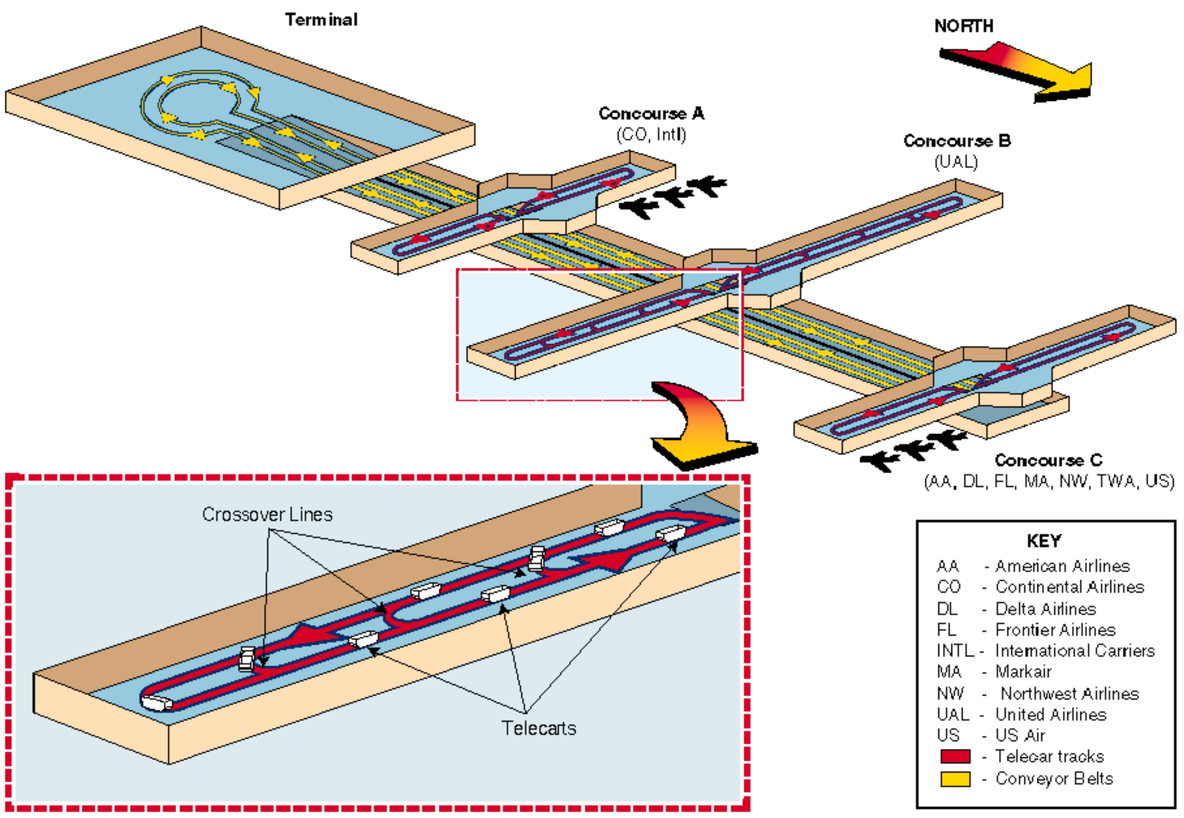


Image 2: Shows a simulation of a Baggage Handling System developed by Babcock airports [3].

Shahrokh Barati
0960653

Historical Failures of Baggage Management Systems

Automatic Baggage Management Systems do not always operate reliably and there have been several cases in the past where catastrophes were produced due to unreliable systems and unrealistic simulations resulting in huge financial losses and unnecessary pressure on airports and customers.
The most famous example of such failures is the Denver Airport system failure of 1994[1]. The following diagram is a representation of the Denver Airport system where bags were put in telecarts and with the use of simulations and sensors; they were supposed to "find" their target locations. The system failed due to slow simulations resulting in one of the biggest financial losses caused by IT system failures.



KEY	
AA	- American Airlines
CO	- Continental Airlines
DL	- Delta Airlines
FL	- Frontier Airlines
INTL	- International Carriers
MA	- Markair
NW	- Northwest Airlines
UAL	- United Airlines
US	- US Air
🟥	- Telecar tracks
🟨	- Conveyor Belts

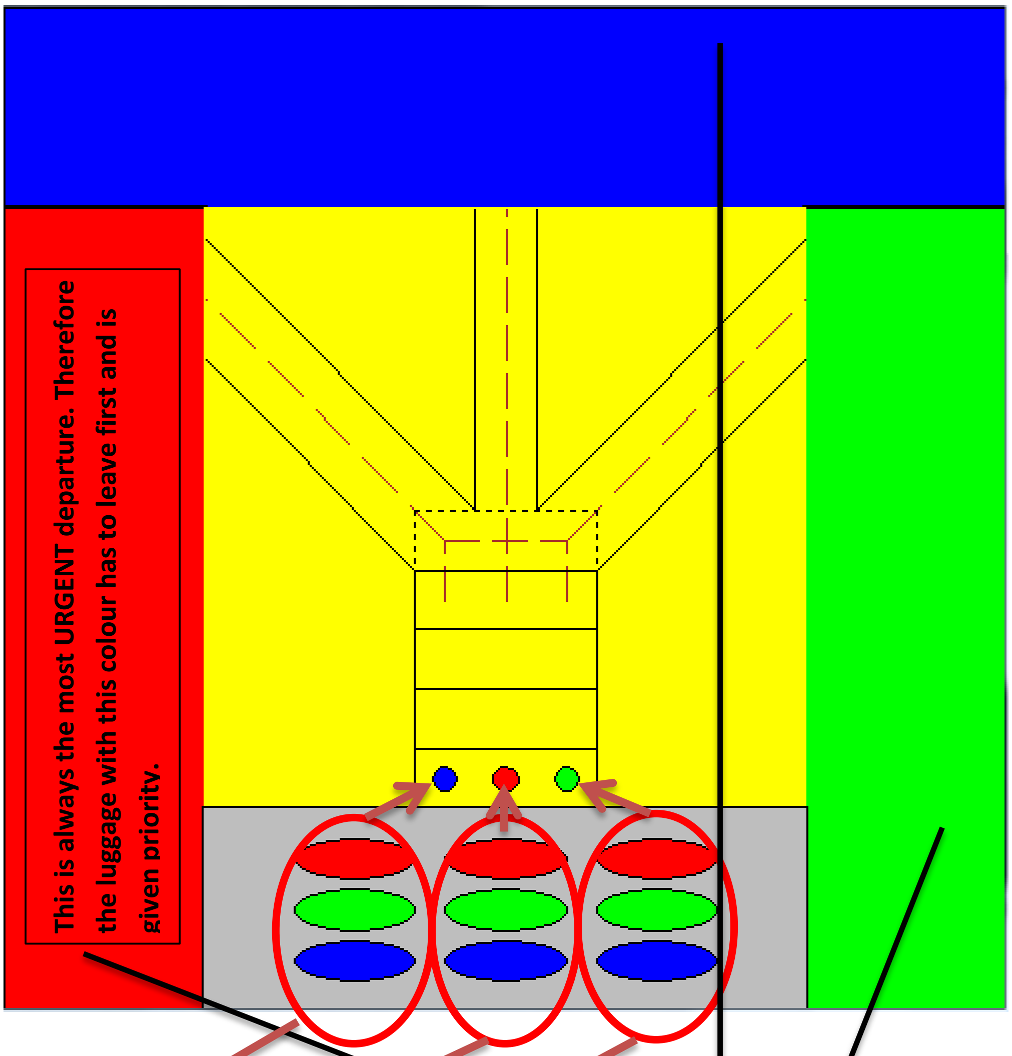
Image 3: The Denver airport baggage handling system 1997

Shahrokh Barati
0960653

The System

There are three airlines and three bags at a given time. Each airline is represented with a different colour. These colours represent the planes that are closest to taking off with the left one being the most urgent one. The user acts as the baggage check-in assistant and once three bags are checked (i.e. the colours are chosen by user); the system starts working and handling the pathways according to their colours and priorities. Image 4 shows the Graphical User Interface of the system:

Image 4:Showing a screenshot of the BHS system.



This is always the most URGENT departure. Therefore the luggage with this colour has to leave first and is given priority.

Each row allows the user (i.e. check-in assistant) to choose the colour of the bag for that lane. Only once a colour from EACH lane is chosen the system starts to work and simulate the pathways and priorities.

Each colour represents a different airline with different departure times. The most urgent departure is the one on the left.

Shahrokh Barati
0960653

Scout

Scout is the tool that defines the attributes of window frames for the GUI. In this system there are 5 windows. The upper, left and right windows represent each departure with a colour (e.g. red, green, blue). The lower window is where the user is allowed to choose the colours of bags for each lane and the middle window is the main window where the simulation is displayed.

Advantages

Scout is a useful tool for specifying the general attributes of a window and it does it is quite reliably. One of the best attributes of Scout is the ability to set the mouse sensitivity for each window and also the fact that different windows are able to have different attributes and be differentiated quite easily. Moreover, the scout syntax is simple and easily comprehensible for non-experts.

Disadvantages & Recommendations

The attributes that are set in each window frame are quite limited, and it would be great if text could be inputted directly as one of the attributes and also if certain variables could be defined within scout.

Moreover, it is difficult to access variables that are set within Eden/Donald. For example if a variable is set in Eden as "`bagColor = "red";`", one has to also specify it as a string in scout to be able to call it.

The following snippet of code shows a window named bwin, with the bgcolor set to bagColor:

```
String bagCol
bwin = {
type :DONALD
box :[{0,0},{500,100}]
pict: "BLUEWIN"
xmin: 0
xmax: 1000
ymin: 0
ymax: 1000
bgcolor: bagCol
/*Does not work since only a string could be inputted, hence
 "string bagCol" should be defined at the beginning of scout.*/
sensitive: ON
};
```

Donald

This tool is used for drawing lines and 2D models. In this system all the points, lines, circles and rectangles are defined in Donald.

Advantages

It is easy to access Donald variables within Eden and vice-versa. In addition, usage of dependencies in Donald allows the system do be more dynamic when faced with on-the-fly variable adjustments.

Disadvantages & Recommendations

The limitation of shapes and more advanced functionalities are the main drawbacks of Donald. For example more advanced predefined shapes such as stars could be introduced.

Eden

Advantages

This is the main tool used for executing the simulations. It is very effective in combining dependencies and observables with procedural programming methods such as "execute" and "todo". The distinction between "is" and "=" is also very smartly done where if a variable is defined using "is", it allows updating of all the variables that are dependent on it accordingly. However "=" is an absolute definition that does not change not matter what.

Disadvantages & Recommendations

A main drawback of Eden is that Donald variables should be assigned to a new Eden variable for the system to be able to handle the adjustments effectively. For example:

```
pos is _pos;
pos2 is _pos2;
center is _center;
width is _width;
```

Therefore, Donald variables should be more accessible and more easily adjusted in Eden since redefining observables could be confusing at times.

System demonstration

The following sequence of screenshots shows how the system handles three bags of different colour with the middle one having priority to go first due to its colour (being the colour of the left window frame i.e. the URGENT frame):

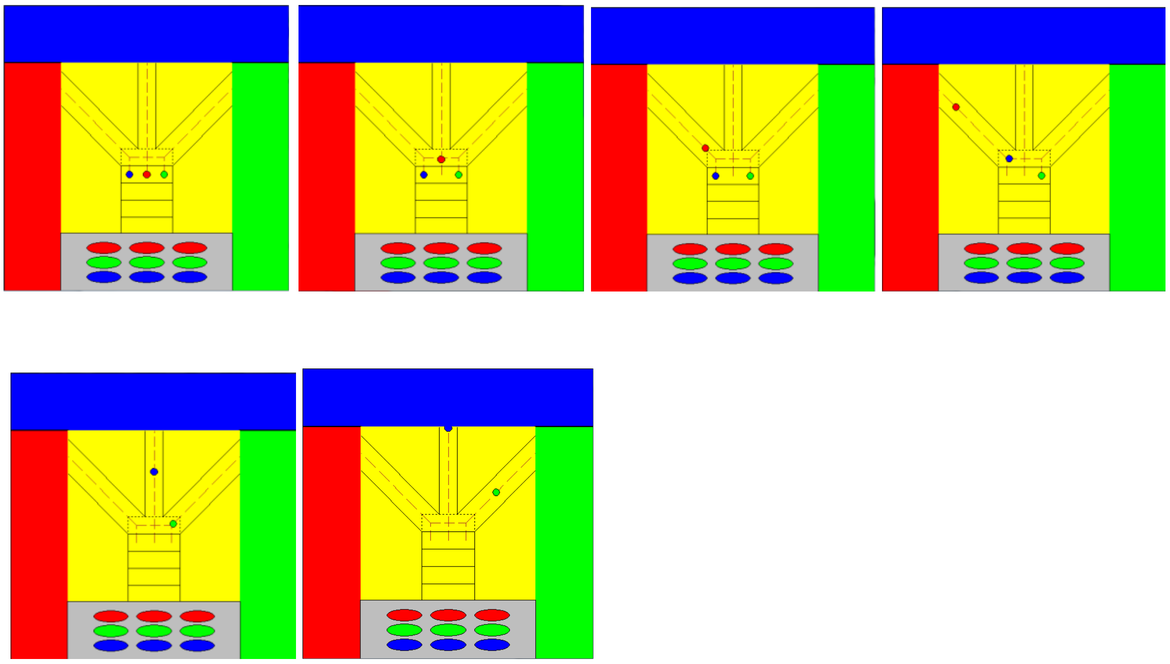


Image 5: Showing a sequence of screenshots illustrating the handling of bags.

If there are no urgent bags, the priorities are handled from left to right meaning the left bag will be handled first and the right one last. However if there is a bag having the same colour as the left window frame (red in the examples) it will be handled before the other no matter its position.
The colour of each bag is defined as follows:

cCol = The colour of bag on left lane
c1Col = The colour of bag on middle lane
c2Col = The colour of bag on right lane

leftC = the colour of the left frame – i.e. the urgent frame

There are 7 main conditions for the priorities which are the following:

Condition 1:
cCol = leftC & c1Col = leftC
sort c→c1→c2

Condition 2:
cCol = leftC & c1Col ≠leftC & c2Col = leftC
sort c →c2→ c1

Condition 3:
cCol = leftC & c1Col ≠leftC & c2Col ≠ leftC
sort c →c1→ c2

Condition 4:
cCol ≠leftC & c1Col = leftC & c2Col ≠ leftC
sort c1→c→c2

Condition 5:
cCol ≠leftC & c1Col = leftC & c2Col = leftC
sort c1→c2→c

Condition 6:
cCol ≠leftC & c1Col ≠ leftC & c2Col = leftC
sort c2→c→c1

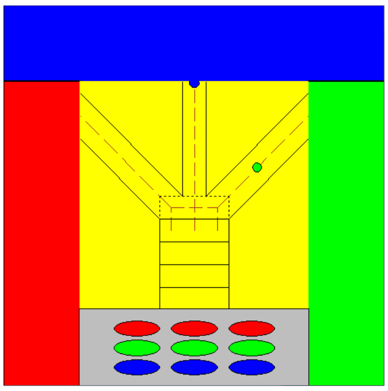
Condition 7:
cCol ≠leftC & c1Col ≠ leftC & c2Col ≠ leftC
sort c→c1→c2

Furthermore, if the user presses on the upper window frame the simulation is paused until the mouse click is released.

General Advantages

The incorporation of dependencies and on-the-fly variable updates are very useful in dynamic systems where if one variable is changed others are adjusted accordingly. In this system, for example once the code is executed if a user inputs "center=800.0" the variable center is adjusted and since all the lines and points are dependent on "center", they all change accordingly and the system remains "unharmed". The following screenshots illustrate this point:

```
center  = 500.0
```



```
center = 800.0
```

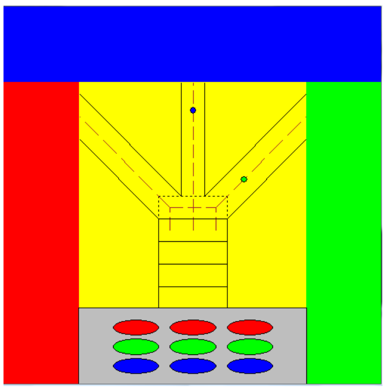
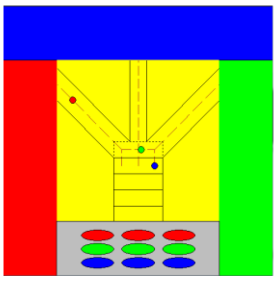


Image 6: The system before and after the variable "center" is changed

The user is also able to change the colours of the window frames and everything else will change accordingly and the system continues to work. The variable leftC, midC and rightC are the colours of each window frame and if the user inputs a new colour for either one the colours of the bags will also change accordingly. The following screenshots illustrate this effect:

?leftC= "red";
?midC= "blue";
?rightC= "green";



?leftC= "white";
?midC= "black" ;
?rightC= "orange";

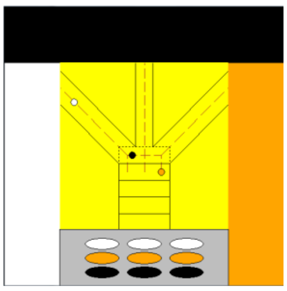


Image 7: The system before and after the colour variables have been adjusted.

General Disadvantages & Recommendations

- It would be better if the syntax of these tools were more similar to one another to avoid confusion. For example in scout integers are specified by “integer” whereas in Donald they are specified by “int”.
- Moreover, the introduction of some object oriented programming techniques could be introduced in order to create more elaborate models such as the introduction of buttons and drop-down lists for better GUIs.
- Furthermore adding the ability to pass parameters within functions could greatly benefit sequential processes in terms of timing and priority.
- Lastly, the variables/observables within each tool should be made more readily accessible within other tools.

Comparison with procedural and object oriented programming

Empirical Modelling techniques such as on-the-fly variable updates and dependencies are very useful in dynamic simulations such as baggage management systems. The ability to update variables instantly and adjust all variables that are affected by the update, allows systems to be more dynamic and reliable resulting in effective and sustainable models.
In procedural programming, tasks are broken down into a collection of variables, data structures, and subroutines. In object oriented programming (OOP) tasks are broken into objects encapsulating their own data and methods. On the other hand in Empirical Modelling, ideally, tasks are broken into a set of interdependent definitions and observables. This allows definitions to be reset on-the-fly very easily enabling more sustainable programs compared to OOP. In addition, another advantage of EM is that it can combine procedural programming methods with definitions and dependencies to obtain better results.

Conclusion

TkEden has a set of very useful tools that use several high level functions combined with lower level programming methods to model artefacts and create simulations. Empirical modelling concepts such as dependency and on-the-fly variable adjustments are easily performed through these tools. These concepts could greatly benefit “live” systems such as baggage handling where instant updates and prioritizations are important for the efficiency and effectiveness of the system. In this project, a simple baggage handling system was developed using TkEden tools and several advantages such as use of dependency and on-the-fly updates and disadvantages such as syntax confusion and accessibility of variables within different tools were addressed.

Bibliography

[1] John Swartz, *Simulating the Denver Airport Automated Baggage System*, January 1997, http://www.drdobbs.com/184410112

[2] BCS Group, *Airport Baggage Handling Systems*,
http://www.airport-technology.com/contractors/baggage/bcs-group/

[3] Babcock Airports, *Baggage Handling Systems*,
http://www.airport-technology.com/contractors/baggage/alstec/

[4] Airport Technology, *Baggage and Cargo Handling Systems and Equipment*,
http://www.airport-technology.com/contractors/baggage/

[5] Roxana Grigoras and Cornelis Hoede, *Design of a Baggage Handling System*, April 2007, University of Twente,
http://doc.utwente.nl/67072/1/memo1835.pdf