überTurnstile™

Example Requirement Specification Document

CS445/ECE451
cs445@student.uwaterloo.ca

September 13, 2001

Contents

List of Figures

List of Tables

1 Introduction

This document is the complete product requirement specification for the überTurnstile™. Henceforth, this is the only document that contains all information regarding the requirements placed on the überTurnstile by the customer, catalogued in an unambiguous fashion. Unless otherwise stated, this document, and any future revisions of this document, supersedes all other requirements documents that exist for the überTurnstile system.

1.1 Purpose

The purpose of this document is to detail the requirements placed on the überTurnstile and serves as a contract between the customer and the manufacturer as to what is expected of the überTurnstile, and how the components of the überTurnstile are to work with each other and with external systems.

1.2 Scope

This document covers the details of the überTurnstile system (*the system*), including the physical components of the system, and the behavioral, functional, and non-functional requirements. This document describes only the external systems and/or environments in which the überTurnstile system shall work, with enough detail to complete an implementation of the überTurnstile system.

1.3 Intended Audience

This documents is intended to be read by

- the customer and
- the manufacturer.

The reader is assumed to have basic knowledge on the basic theory and operation of a regular turnstile, and have some experience with the terminology used in the document. Also, experience in reading UML diagrams is required.

1.4 Definitions

[*This section lists any terminologies that need to be defined before the reader continues with the rest of the document.*]

For this document, there is no terminology requiring definitions.

1.5 Notational Conventions

[*This section details any notational conventions that this document follows. Essentially, your group needs to mention the notational language used in the diagrams shown in the document, and mention any notation/style that deviates from the standard notion used.*]

All diagrams in Section 3 conform to UML notation. For some reason, the encapsulated postscript exporter for the UML modelling software MagicDraw [1] converts all solid lines into dashed lines for the following:

- relationships in class diagrams, including aggregration, composition, and inheritance relationships, and
- state transitions in state diagrams.

For these cases, the reader should treat the dashed lines as solid lines.

In this document, the choice of font is meaningful:

- Narrative text is in seriffed Roman.
- *Introduced terms are in seriffed Italics.*
- *System components, states, and events are in slanted sans serif or bold upright sans serif.*

Indicative statements about the environment in which the system operates and assumptions that the system can rely on are in present tense or future tense.

Optative statements about what the system is supposed to do, i.e., its requirements, use "shall".

1.6 References

[*This section lists all documents and communications that are used as information sources. Examples of documents are project descriptions, web pages, and lecture notes; examples of communcations include customer sessions and electronic/verbal communication between your group and the customer. For this example, the author drew some ideas from the tools demo.*]

- CS445 Tools Demo: http://www.student.math.uwaterloo.ca/~cs445/demos/

[1] http://www.magicdraw.com

1.7 Overview

In the rest of the document, Section 2 gives a general description of the system, and identifies all assumptions and constraints placed upon the system. Section 3 exposes the specific reqirements of the system demanded by the customer, and their relationships via various class, state, and sequence diagrams. Finally, Appendix A presents the Data Dictionary, while Appendix B presents the Traceability Matrix.

2 General Product Description

Better Turnstiles LLC (**customers**), the makers of the Turnstile Example, wants to upgrade their flagship product, and the company has named it the überTurnstile system.

The überTurnstile is, above all, a turnstile (see Figure 1) – it consists of the following externally-visible components:

- the barrier bars (*barrier*), usually a triple of them arranged in a tripod fashion, joined and rotatable at their base to grant entry,
- the payment collector (*paybox*) is responsible for unlocking the barrier upon receiving the correct payment, and
- the main housing (*housing*) of the turnstile, which accommodates the aforementioned components of the turnstile.

Figure 1: An artist's impression of the überTurnstile

2.1 Product Perspective

The überTurnstile system, just like any turnstile, is used to control entry to a restricted area.

2.2 Product Functions

The main function of the überTurnstile system is to restrict entry into a specific area. A person who wishes to enter the area must pay a predetermined sum of money by feeding the payment into the paybox. Upon the payment of sufficient money, the barrier shall unlock and allow the paying person to enter.

2.3 Characteristics of Eventual Users

The *user* is anyone who wish to enter an area restricted by the überTurnstile system. The user is assumed to know how to insert payment into the paybox, and how to push the barrier to allow herself through the überTurnstile and into the restricted area.

The *operator* is the person that manages the überTurnstile system. She is the person that can decide, at any time, to turn on or off a particular überTurnstile system.

[1]Figure 1 courtesy of http://www.tomsed-turnstiles.com/page2.html

2.4 Constraints

None idenitified.

[*That there are no constraints identified is very rare and exceptional. You should make a serious effort to identify them when your group writes the real SRS.*]

2.5 Assumptions and Dependencies

- The paybox has infinite capacity – it can never be full.
- All payboxes always perform correctly, i.e. when the user inserts payment, the paybox is always able to accept the payment, and it correctly identifies the amount of the payment.
- The barrier rotates only one way. Users do not attempt to push the barrier the other way.
- The barrier, when unlocked and pushed by the user, rotates without fail until the user has successfully passed through the überTurnstile and gained entry into the restricted area.

3 Specific Requirements

3.1 External Interface Requirements

3.1.1 Operator Interface

The operator's interface is shown in Figure 2.

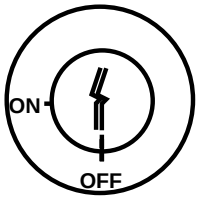

Figure 2: Operator On/Off Switch

The operator switches the überTurnstile system on or off by using this key-operated interface. The interface is shown in the "off" setting. To turn the system on, the operator inserts the key, and turns the key clockwise until the notch lines up with the "on" markings; to turn the system off, the key should be rotated counter-clockwise until the notch aligns with the "off" markings, as shown. This interface generates the *turnOn* and *turnOff* events.

3.1.2 User Interface

The users's interface is shown in Figure 3.

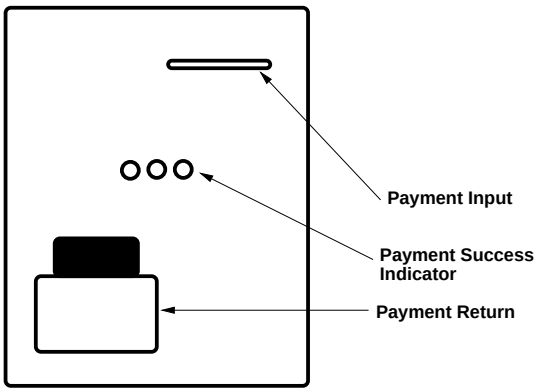

Figure 3: Paybox Exterior

This interface allows the user to insert payment for entry. Starting from the top:

Payment Input is a slot for inserting the payment. This component generates the *insertPayment()* event.

Payment Success Indicator is used to indicate if the payment was accepted or rejected to the user. This component receives the *paymentAccepted* and *paymentRejected* events and displays indications of them to the user.

Payment Return is where the payment is returned to the user if the payment was rejected. If this component receives the *paymentRejected* event, it returns the current payment to the user.

3.1.3 Communications Interface

There are no other external systems that the überTurnstile is designed to communciated with.

3.2 Behaviour Requirements

3.2.1 Use Case Diagram

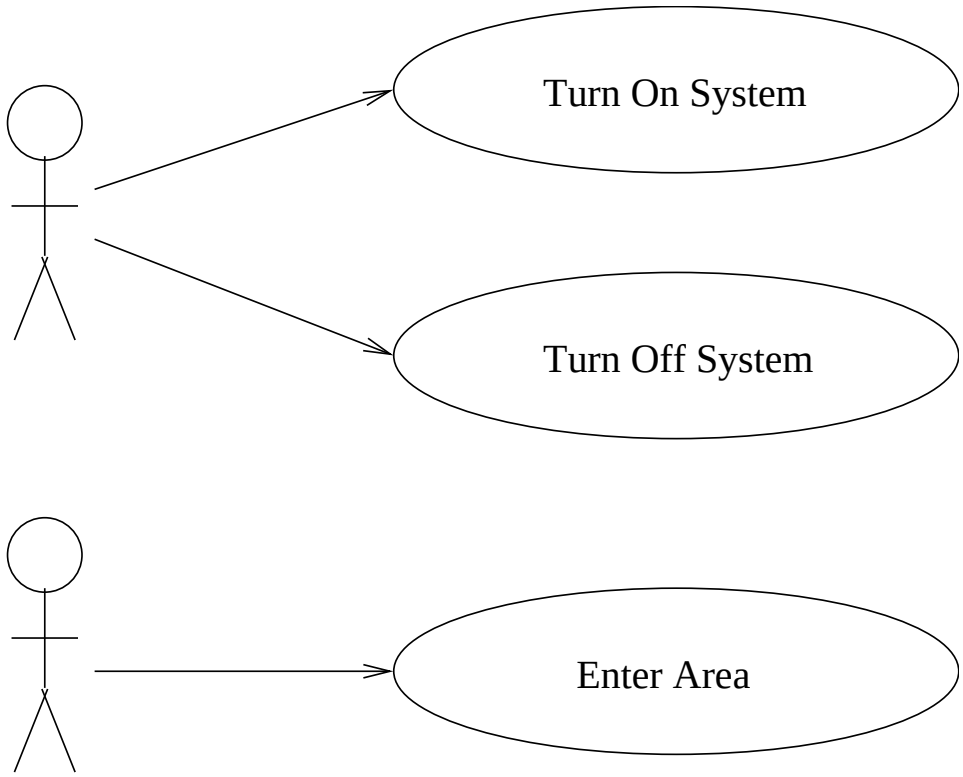

Figure 4: Use Cases Diagram

3.2.2 Use Case Descriptions

Table 1: Use case: Turn Off System

Name: Turn Off System	**ID:** 1
Event: Operator turns off überTurnstile system **System:** überTurnstile **Actors:** Operator(Initiator) **Overview:** Description of internal behaviour after the operator turns the überTurnstile off. **References:** N3 **Related Use Cases:** UC2	
Typical process description:	
Initiator Actions	*System Response*
1. Operator turns off überTurnstile	
	2. resets counters for available entries and visitors
	3. locks barrier
	4. system stops responding to events except "turn on", shall return payment immedately if it was made at this point

Table 2: Use case: Turn On System

Name: Turn On System	**ID:** 2
Event: Operator turns on überTurnstile system **System:** überTurnstile **Actors:** Operator(Initiator) **Overview:** Description of internal behaviour after the operator turns the überTurnstile on. **References:** N3 **Related Use Cases:** UC1	
Typical process description:	
Initiator Actions	*System Response*
1. Operator turns on überTurnstile	
	2. system (again) accepts external events

Table 3: Use case: User Enter Through Turnstile

Name: User Enter Through Turnstile	**ID:** 3
Event: User attempt to enter restricted area through the überTurnstile **System:** überTurnstile **Actors:** User(Initiator) **Overview:** Description of internal behaviour while user makes payment and then pushes barrier in attempt to enter. **References:** **Related Use Cases:**	
Typical process description:	
Initiator Actions	*System Response*
1. User inserts payment	
	2. concludes payment can purchase one entry
	3. update the number of available entries
	4. unlock barrier
5. User pushes barrier	
	6. barrier rotates
	7. visit is noted, visitor count incremented
	8. barrier locked
Alternative 1:	
5. user pushes barrier without payment	
	6. barrier remains locked, entry denied
Alternative 2:	
1. User inserts payment for 2 entries	
	2. paying user enters as per steps 2-7
	8. barrier remains unlocked
9. Another user pushes barrier	
	10. barrier rotates
	11. visit is noted, visitor count incremented
	12. barrier locked

3.2.3 Sequence Diagrams

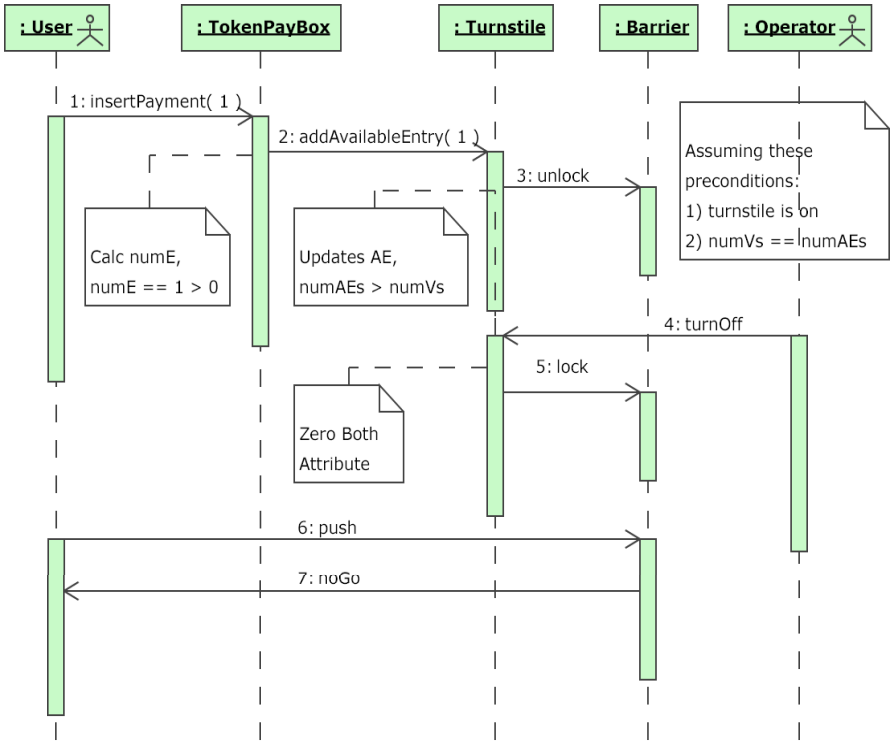

Figure 5: Sequence Diagram - Operator turns off the überTurnstile

This sequence diagram shows how the überTurnstile shall behave if the operator decides to turn off the turnstile after a user paid, but before she enters though the turnstile. After the turnstile is turned off, the barrier should be locked, preventing the paying user from entering.

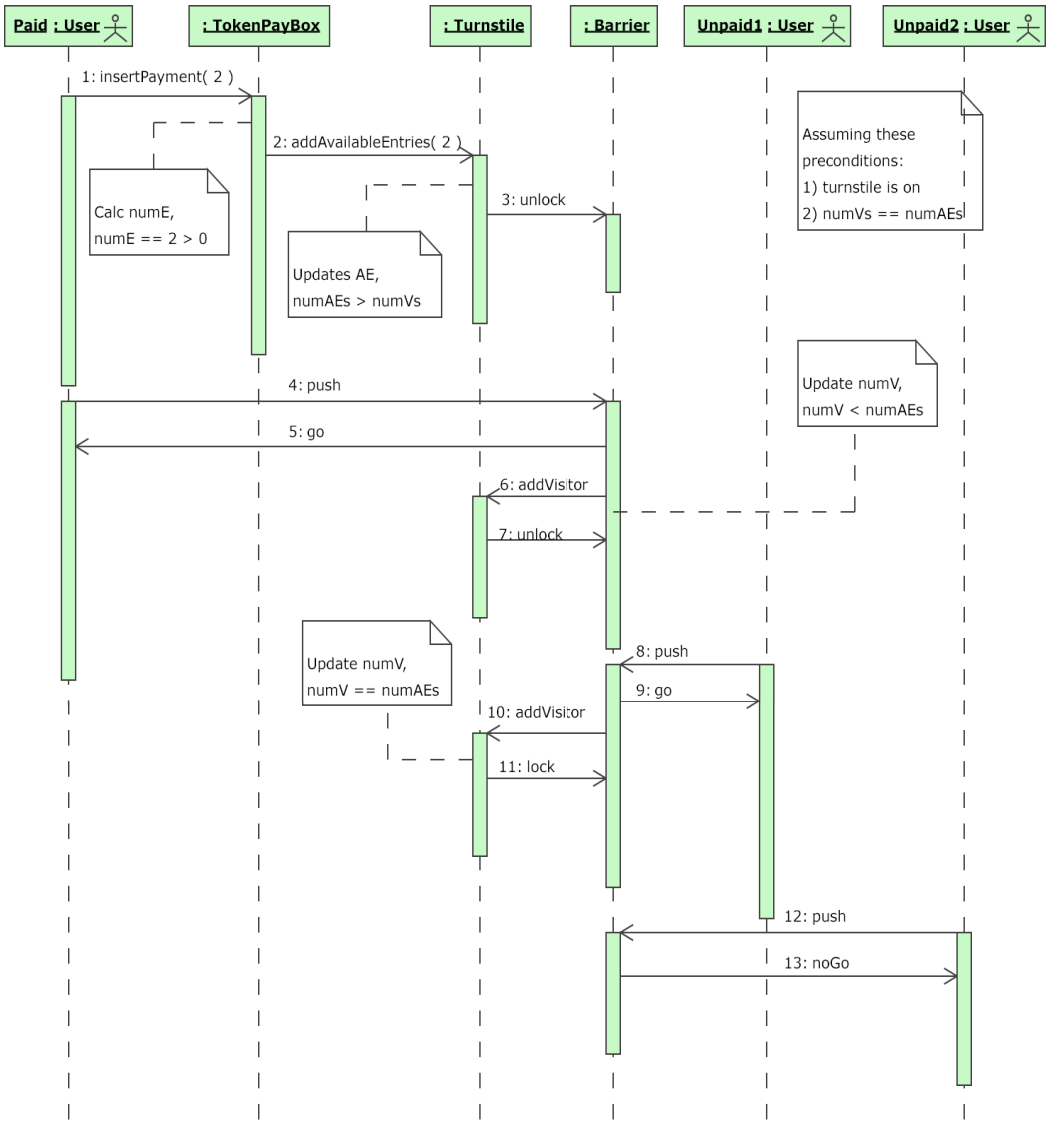

Figure 6: Sequence Diagram - Paying for two

This sequence diagram illustrates a more complex case, when one user pays enough for two people to enter through the überTurnstile system. Essentially, the turnstile should be able to recognize that the first user ("Paid") paid for two entries, and so when the second user ("Unpaid1") pushes the barrier, she is allowed to pass. Afterwards, a second unpaid user ("Unpaid2") shall fail to enter without paying.

[*This example SRS includes only two sequence diagrams that illustrate more complex, exceptional scenarios, the kind that many students fail to find. Most of the other required scenarios are the straightforward, normal ones that, traditionally, pose no problems to students. Therefore, the sequence diagrams for these normal scenarios are left as an exercise for the student.*]

3.2.4 Class Diagram

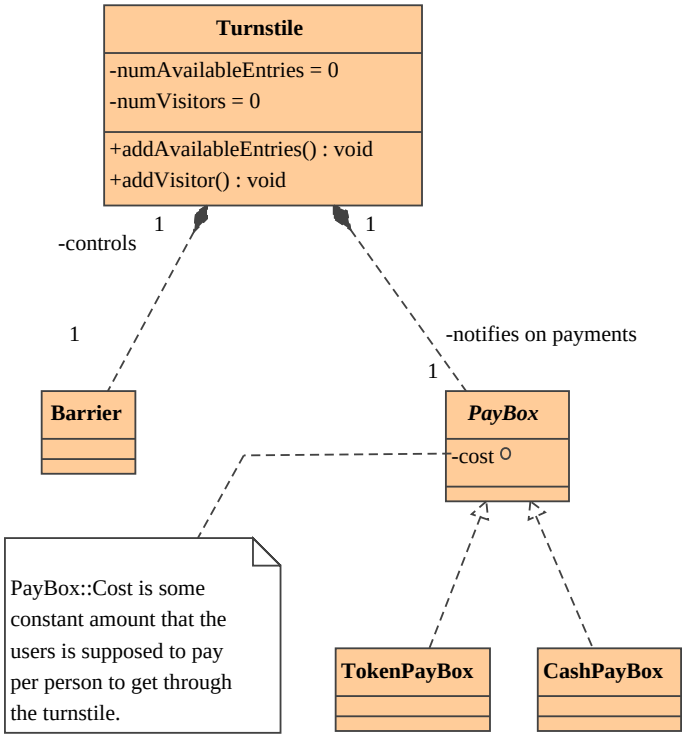

Figure 7: Class Diagram of überTurnstile

There are three main classes in the überTurnstile system.

Turnstile represents the housing. This class is the central class of the system representation, and is responsible for handling most of the logic of the system.

Barrier represents the physical barrier – and it also keeps track of users who pass through it, notifying the Turnstile class when each passage is complete.

PayBox is an abstract class that represents the family of subsystem that takes payment from users. It keeps track of the cost of each entry, and is able to calculate the number of "entries" that is bought by each payment it receives. The paybox communicates with the Turnstile each time sufficient payment is received.

TokenPayBox

CoinPayBox are the concrete classes of *PayBox*. They take tokens and coins (cash) as payment, respectively.

3.2.5 State Diagrams

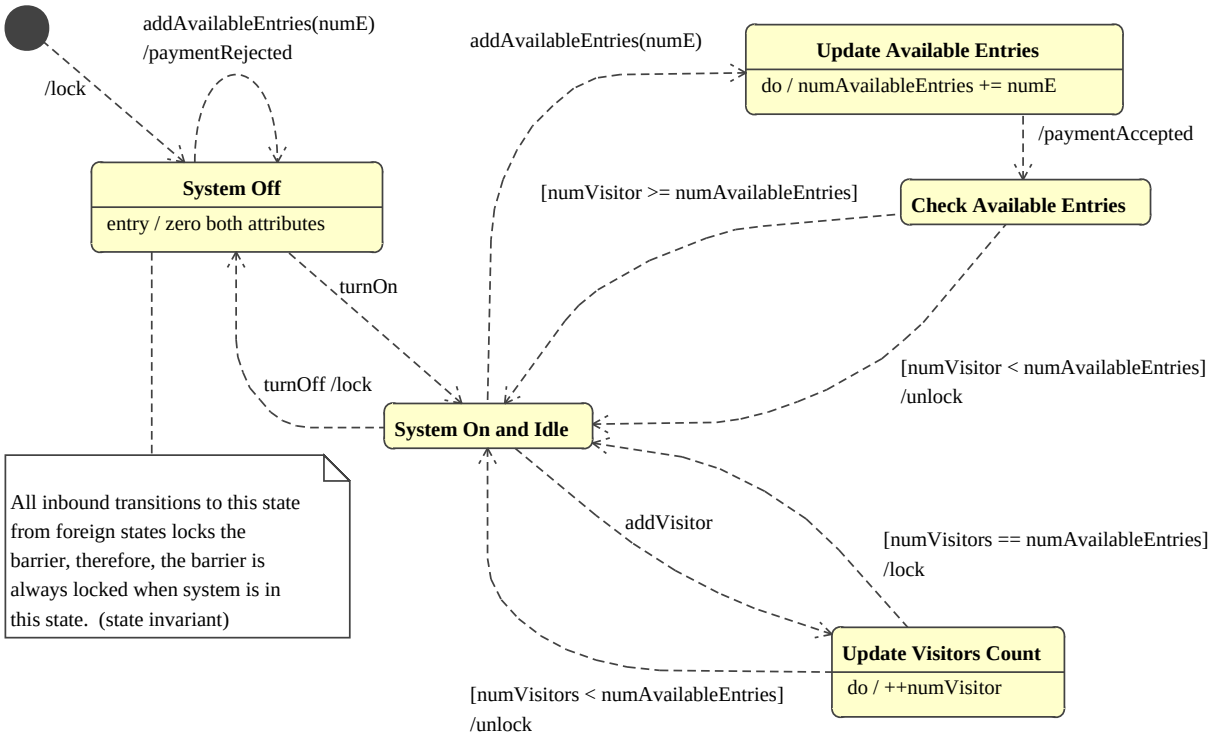

Figure 8: State Diagram - *Turnstile*

This state diagram can be partitioned into three major state groups:

System Off In this state the überTurnstile shall ignore all external events, except *turnOn*,

Update Visitor Count This state handles the logic needed for adding a visitor, a user that has successfully passed through the überTurnstile system,

Update Available Entries This state provides the logic to handle a user paying for one or more entries.

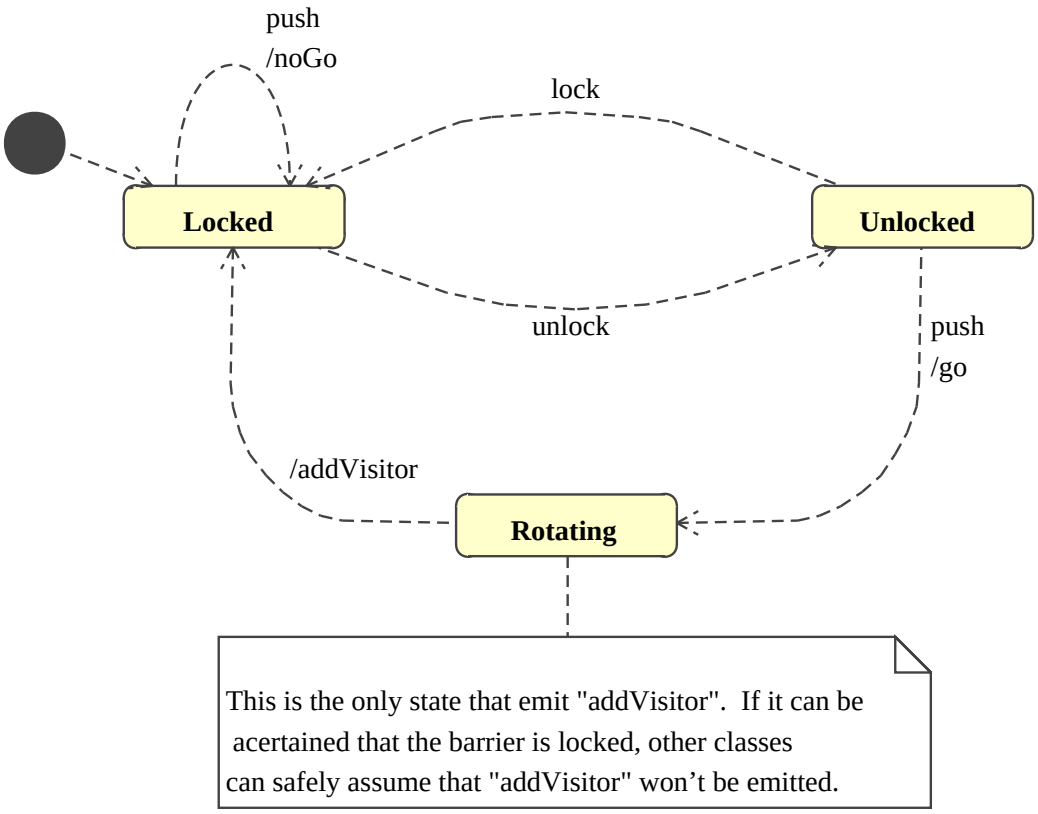

Figure 9: State Diagram - *Barrier*

The barrier can be in only three states,

- locked (the state that the barriers is in most of the time),
- unlocked (the state that allow a user to push and enter), and
- rotating (the transitional state that the barrier shall leave immediately after entry).

The barrier is mostly passive, reacting to push events sent by the user and events sent by the turnstile.

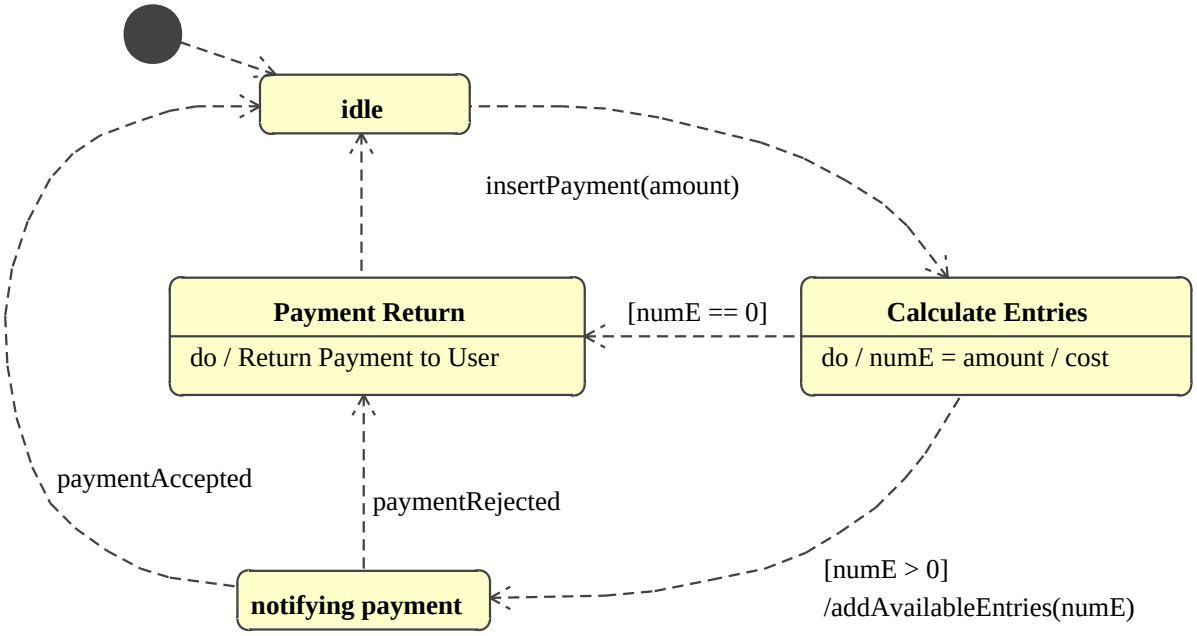

Figure 10: State Diagram - *PayBox*

Basically there are two steps that the paybox performs after receiving payment:

1. checks to see how many entries can be bought by the latest payment.
   - if insufficient for at least one entry, payment is returned to the user. Entry is disallowed.
   - no change shall be returned to the user – for example, if it costs $2 for each entry, inserting $5 shall grant 2 entries, with no change returned to the user. Entry is allowed.
2. check with turnstile to see if the payment is to be accepted.
   - if the payment is rejected, it is returned to the user.

[*The state diagrams for $TokenPayBox$ and $CashPayBox$ are very similar to that of the $PayBox$ and are left as exercise for the student*]

[*Notice that the description following each state diagram does not repeat the information presented in the diagrams themselves; instead, the description should provide additional information to aid reader understanding.*]

3.3 Functional Requirements

[*Traditionally, this section is for requirements that are not described adquately by the behavioural specifications given in the previsous section. This section was important back in the OMT days, when there were functional diagrams that could be inserted into this section. With the switch to UML, and the use of the traceability matrix (Appendix B), it is not clear that this section is still as useful.*]

Please refer to Appendix B for a complete list of requirements identifed by this document.

3.4 Non-Functional Requirements

[*The types of non-functional requirements include, but are not limited to, those listed in the following subsections. See the lecture notes for a complete list.*]

3.4.1 Scalability

- While a single überTurnstile unit processes only one user at a time, multiple überTurnstile units working together, independently, can process multiple users simultaineously.
- While this document lists two types of paybox, the specification does not prohibit new types of payboxes to be plugged into the überTurnstile system, so long as they obey the interface between payboxes and turnstile outlined in this specification, specifcally in Section 3.2.5.

3.4.2 Reliability

- The überTurnstile shall allow exactly the same number of users to enter as the number of entries that has been paid for.

3.4.3 Security

- To switch on and off the überTurnstile system, the operator must possess the operator's key, and can do so only through the operator's interface detailed in Section 3.1.2

3.4.4 Usability

- Users needs only the knowledge described in section 2.3 to operate the überTurnstile system.

A Data Dictionary

A.1 Classes

Table 4: *Turnstile* class

Turnstile					
Purpose	Stores information collected during operation, and responsible for many of the logical operation of the system				
Attributes	*Name* : *Type*	*Purpose*			*Range*
	numAEs : int	# users allowed to enter			$0 \ldots \infty$
	numVs : int	# users who has entered			0...numAEs
Operations	*Name*	*Purpose*	*Parameters*	*Pre-Cond*	*Post-Cond*
	addAE	increment numAEs	numE:int		numAEs += numE
	addV	increment numVs			numVs += 1
[*Shorthand: AE = AvailableEntry, V = Visitor*]					

Table 5: *Barrier* class

Barrier	
Purpose	Responsible for storing the state of the physical barrier and its lock

Table 6: *PayBox* class

PayBox			
Purpose	Abstract class responsible for handling the logic of a generic paybox		
Attributes	*Name* : *Type*	*Purpose*	*Range*
	cost : int	cost per person to enter	$0 \ldots \infty$

Table 7: *TokenPayBox* class

TokenPayBox	
Purpose	Concrete subclass of PayBox – tokens as payment medium

Table 8: *CashPayBox* class

CashPayBox	
Purpose	Concrete subclass of PayBox – cash as payment medium

A.2 Events

Table 9: Events Table

Name	*Purpose*	*Parameter Name : Type*	*Source → Sink*
insertPayment	pay inserted	amount : int	user → Paybox
paymentAccepted	turnsile accepts pay	–	Turnstile → Paybox
paymentRejected	turnsile rejects pay	–	Turnstile → Paybox
push	user pushes barrier	–	user → Barrier
go	barrier is rotating	–	Barrier → user
noGo	barrier is locked	–	Barrier → user
lock	lock barrier	–	Turnstile → Barrier
unlock	unlock barrier	–	Turnstile → Barrier
turnOn	switch turnstile on	–	Operator → Turnstile
turnOff	switch turnstile off	–	Operator → Turnstile
addVisitor	signals end of barrier rotation	–	Barrier → Turnstile
addAvailableEntries	user paid for nunE entries	numE : int	Paybox → Turnstile

A.3 Activities

Table 10: Activities Table

Name	*Purpose*	*Parameter Name : Type*	*Initiator → Target*
Return payment to user	Refund payment if rejected	–	PayBox → User
Calc numE	numE = # people allowed in for payment	amount : int cost : int	PayBox
Zero both attrib	zeros numAEs, numVs	–	Turnstile
++V	inc. V by 1	–	Turnstile
AE+=numE	inc. AE by numE	numE : int	Turnstile
[*Shorthand: AE = numAvailableEntries, V = numVisitors*]			

B Traceability Matrix

[Usually there should also be a column for "source of requirement", but this example SRS has only one source, so this column is omitted.]

Table 11: Traceability Matrix

ID	Catagory	Name	Description	Details/Constraints	Related Reqs/UCs	Found in
F1	[I]	:Entry/Vistor Tracking	System keeps track of # visitors/entries	–	N2	§3.2.4
F2	[E]	:Operator Interface	Interface for operator to turn on or off system	–	N3, UC1,2	§3.1.2
F3	[E]	:User Interface	Interface for users to insert payment and receive return payment	–	N4, UC3	§3.1.2
F4	[E]	:Components	The turnstile consists of the barrier, the paybox, and the turnstile itself	These componetns and their cooperative relationship shoudl be represented	–	§2, §3.2.4
F5	[E]	:General Usage	The turnstile shall receive payment from the user, and then grant entry to the user(s)	The number of entries is based on the amount of payment and the predetermined entry cost	–	Fig6, §3.2.5
F6	[E]	:Barrier Default	The barrier will by default be locked	Barrier will be locked when system first turned on, when it is off	F7	Fig5, Fig8
F7	[E]	:Did not pay	A person that did not pay is not allowed through the turnstile,	because the barrier will remained locked until payment	F6	Fig6, Fig8, Fig9
F8	[E]	:Pay for multiple	A person that can pay for more than one entry	Users following the payment user can enter without payment	–	Fig6, Fig8, Fig9 Fig10
F9	[E]	:Not enough pay	A payment of insufficient amount for at least one entry will be returned	Entry denied, unless a previous user paid for multiple entries	–	Fig10
F10	[E]	:Pay when off	All payments shall be retuned when system is off	–	–	Fig10
⋮			⋮	⋮	⋮	⋮
N1	[M]	:Paybox types	Compatability with future payboxes types	new payboxes also conform to specs in §3.2.5	–	§3.4.1
N2	[M]	:Must pay	Every visitor that successfully enters through überTurnstile must have had paid	–	–	§3.4.2

Continued on next page...

Table 11: Traceability Matrix (*continued*)

ID	*Catagory*	*Name*	*Description*	*Details/Constraints*	*Related Reqs/UCs*	*Found in*
N3	[M]	:Turn on/off	Only the operator key, through the operator interface, can turn on or off the überTurnstile	–	UC1,2	§3.4.3
N4	[M]	:Ease of use	Users that know how to deposit payment into turnstile and push barrier can use überTurnstile	–	–	§3.4.4

C Post Customer-Interview Reports

[*If there were any post customer-interview reports, they would go here. Read* `http://www.student.math.uwaterloo.ca/~cs445/OfficeHours.shtml` *for instructions on how to write these reports.*]