1 Introduction

1.1 Purpose

This subsection should

- a) Delineate the purpose of the SRS;
- b) Specify the intended audience for the SRS.

1.2 Scope

Name of software to be developed: Airport System

This subsection should

- b) Explain what the software product(s) will, and, if necessary, will not do;
- c) Describe the application of the software being specified, including relevant benefifits, objectives, and goals;
- d) Be consistent with similar statements in higher-level specififications (e.g., the system requirements specifification), if they exist.

1.3 Product Overview

1.3.1 Product perspective

This subsection of the SRS should put the product into perspective with other related products. If the product is independent and totally self-contained, it should be so stated here. If the SRS defines a product that is a component of a larger system, as frequently occurs, then this subsection should relate the requirements of that larger system to functionality of the software and should identify interfaces between that system and the software.

This subsection should also describe how the software operates inside various constraints. For example,
these constraints could include

- a) System interfaces;
- b) User interfaces;
- c) Hardware interfaces;
- d) Software interfaces;
- e) Communications interfaces;
- f) Memory;
- j) Operations;
- k) Site adaptation requirements.

1.3.1.1 System interfaces

SI1 - AirportSystem

Service Name:	AirportSystem
Service ID:	SI1
Description:	
Operation:	• createStaff • createDevice

SI2 - RepairService

Service Name:	RepairService
Service ID:	SI2
Description:	
Operation:	• approve • finishRepair • feedback

SI3 - ThirdPartyServices

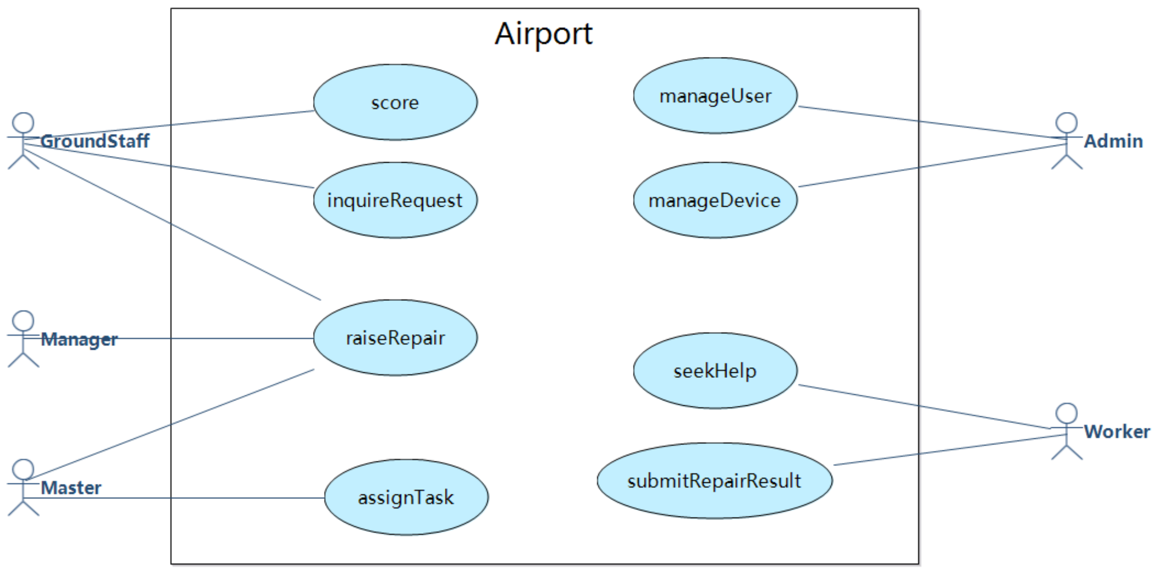
Service Name:	ThirdPartyServices
Service ID:	SI3
Description:	
Operation:	

SI4 - RaiseRepairService

Service Name:	RaiseRepairService
Service ID:	SI4
Description:	
Operation:	• submitRequest • managerApprove1 • masterApprove • managerReject • managerApprove2 • masterReject

1.3.2 Product functions

Use Case Diagram



ID	Use Case Name	Use Case Description	Subfunction
UC1	manageUser		
UC2	manageDevice		
UC3	assignTask		
UC4	submitRepairResult		
UC5	seekHelp		
UC6	inquireRequest		
UC7	score		
UC8	raiseRepair		submitRequest managerApprove1 masterApprove managerReject managerApprove2 masterReject

1.3.3 User characteristics

ID	Actor	Description	Super Actor
A1	GroundStaff		
A2	Manager		
A3	Master		
A4	Worker		
A5	Admin		

1.3.4 Limitations

This subsection of the SRS should provide a general description of any other items that will limit the developer’s options. These include

- a) Regulatory policies;
- b) Hardware limitations (e.g., signal timing requirements);
- c) Interfaces to other applications;
- d) Parallel operation;
- e) Audit functions;
- f) Control functions;
- g) Higher-order language requirements;
- h) Signal handshake protocols (e.g., XON-XOFF, ACK-NACK);
- i) Reliability requirements;
- j) Criticality of the application;
- k) Safety and security considerations.
- l) physical/mental considerations; and
- m) limitations that are sourced from other systems, including real-time requirements from the controlled system through interfaces.

1.4 Definitions

This subsection should provide the defifinitions of all terms required to properly interpret the SRS. This information may be provided by reference to one or more appendixes in the SRS or by reference to other documents.

2 References

This subsection should

- a) Provide a complete list of all documents referenced elsewhere in the SRS;
- b) Identify each document by title, report number (if applicable), date, and publishing organization;
- c) Specify the sources from which the references can be obtained.

This information may be provided by reference to an appendix or to another document.

3 Requirements

3.1 Functions

3.1.1 Use Case

UC1 - manageUser

Use Case Description:

UseCase Name:	manageUser
UseCase ID:	UC1
Brief Description:	
Involved Actor:	Admin
Preconditions:	
Postconditions:	
Basic Path:	
Alternative Path:	

UC2 - manageDevice

Use Case Description:

UseCase Name:	manageDevice
UseCase ID:	UC2
Brief Description:	
Involved Actor:	Admin
Preconditions:	
Postconditions:	
Basic Path:	
Alternative Path:	

UC3 - assignTask

Use Case Description:

UseCase Name:	assignTask
UseCase ID:	UC3
Brief Description:	
Involved Actor:	Master
Preconditions:	
Postconditions:	
Basic Path:	
Alternative Path:	

UC4 - submitRepairResult

Use Case Description:

UseCase Name:	submitRepairResult
UseCase ID:	UC4
Brief Description:	
Involved Actor:	Worker
Preconditions:	
Postconditions:	
Basic Path:	
Alternative Path:	

UC5 - seekHelp

Use Case Description:

UseCase Name:	seekHelp
UseCase ID:	UC5
Brief Description:	
Involved Actor:	Worker
Preconditions:	
Postconditions:	
Basic Path:	
Alternative Path:	

UC6 - inquireRequest

Use Case Description:

UseCase Name:	inquireRequest
UseCase ID:	UC6
Brief Description:	
Involved Actor:	GroundStaff
Preconditions:	
Postconditions:	
Basic Path:	
Alternative Path:	

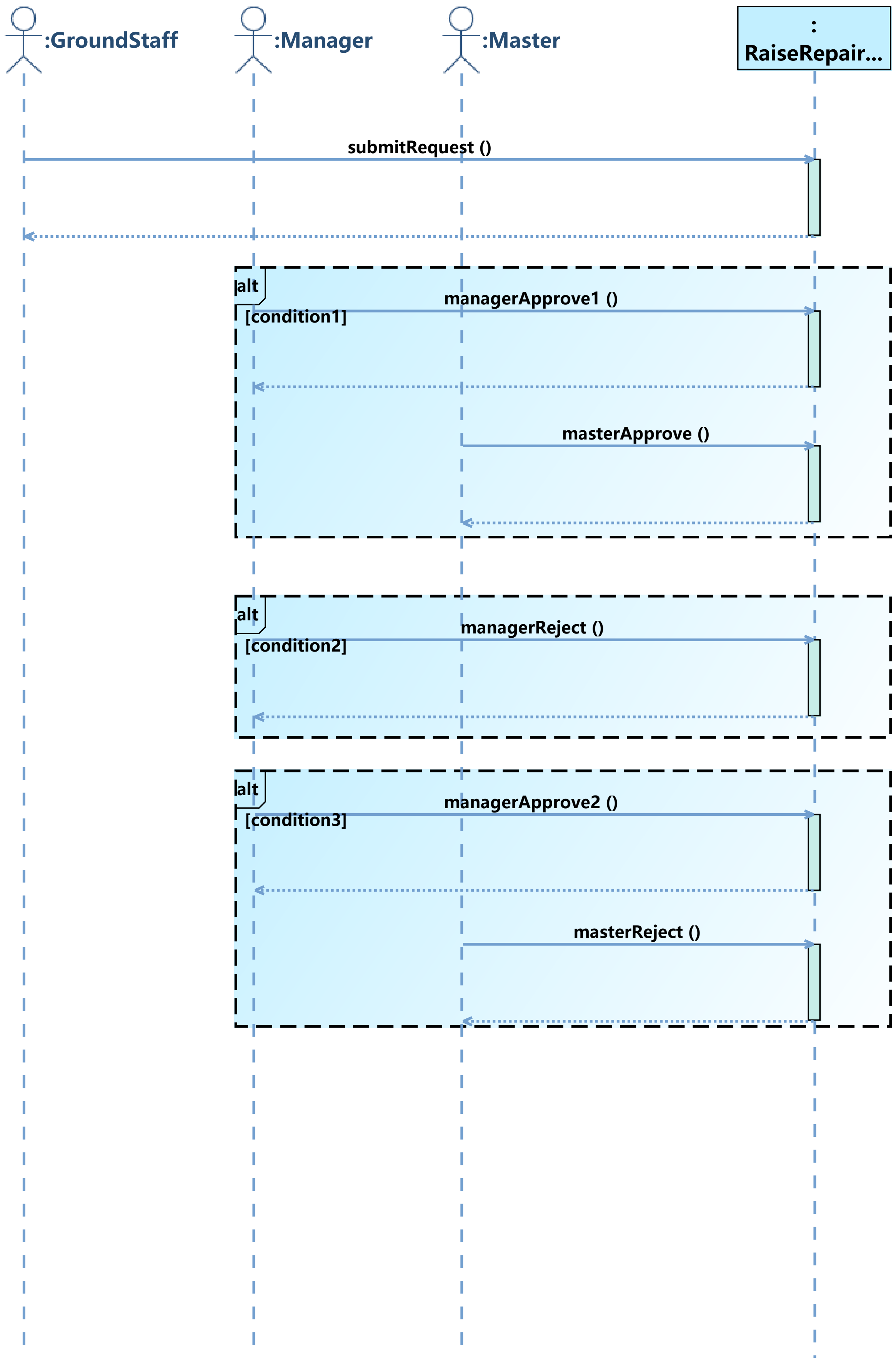
UC7 - score

Use Case Description:

UseCase Name:	score
UseCase ID:	UC7
Brief Description:	
Involved Actor:	GroundStaff
Preconditions:	
Postconditions:	
Basic Path:	
Alternative Path:	

UC8 - raiseRepair

System Sequence Diagram:



Use Case Description:

UseCase Name:	raiseRepair
UseCase ID:	UC8
Brief Description:	
Involved Actor:	GroundStaffManagerMaster
Preconditions:	
Postconditions:	
Basic Path:	1. GroundStaff clicks to execute the operation submitRequest 2. Execute combinedFragement1 Select condition1: Manager clicks to execute the operation managerApprove1 Master clicks to execute the operation masterApprove 3. Execute combinedFragement2 Select condition2: Manager clicks to execute the operation managerReject 4. Execute combinedFragement3 Select condition3: Manager clicks to execute the operation managerApprove2 Master clicks to execute the operation masterReject
Alternative Path:	

3.1.2 System Operation

OP1 - createStaff

Operation Name:	createStaff
Operation ID:	OP1
Description:	
Service:	AirportSystem
Input:	1. name: *id*, type: Integer 2. name: *name*, type: String 3. name: *pswd*, type: String 4. name: *phone*, type: String 5. name: *role*, type: Integer 6. name: *bossid*, type: Integer
Output Type:	Boolean
Definition:	1. *sta* is the object *u* in the instance set of class Staff. *u* represents an object of class Staff, and *u* meets: The attribute *Id* of the object *u* is equal to *id* 2. *bo* is the object *uu* in the instance set of class Staff. *uu* represents an object of class Staff, and *uu* meets: The attribute *Id* of the object *uu* is equal to *bossid*
Preconditions:	The object *sta* doesn't exist
Postconditions:	1. *s* represented the object of class Staff 2. The object *s* was created 3. The attribute *Id* of the object *s* became *id* 4. The attribute *Name* of the object *s* became *name* 5. The attribute *Password* of the object *s* became *pswd* 6. The attribute *Phone* of the object *s* became *phone* 7. The attribute *Role* of the object *s* became *role* 8. If the object *bo* existed, take the following as postcondition(s): The object *s* was linked to the object *bo* by *Boss* 9. The object *s* was put into the instance set of class Staff 10. The return value was **true**

Contract of **createStaff**:

```
Contract AirportSystem::createStaff(id : Integer, name : String, pswd : String,
phone : String, role : Integer, bossid : Integer) : Boolean {
        definition:
            sta:Staff = Staff.allInstance()->any(u:Staff | u.Id = id),
            bo:Staff = Staff.allInstance()->any(uu:Staff | uu.Id = bossid)
        precondition:
            sta.oclIsUndefined() = true
        postcondition:
            let s:Staff in
            s.oclIsNew() and
            s.Id = id and
            s.Name = name and
            s.Password = pswd and
            s.Phone = phone and
            s.Role = role and
            if
                bo.oclIsUndefined() = false
            then
                s.Boss = bo
            endif and
            Staff.allInstance()->includes(s) and
            result = true
}
```

OP2 - createDevice

Operation Name:	createDevice
Operation ID:	OP2
Description:	
Service:	AirportSystem
Input:	1. name: *id*, type: Integer 2. name: *name*, type: String 3. name: *location*, type: String 4. name: *contactsid*, type: Integer
Output Type:	Boolean
Definition:	1. *dev* is the object *u* in the instance set of class Device. *u* represents an object of class Device, and *u* meets: The attribute *Id* of the object *u* is equal to *id* 2. *sta* is the object *uu* in the instance set of class Staff. *uu* represents an object of class Staff, and *uu* meets: The attribute *Id* of the object *uu* is equal to *contactsid*
Preconditions:	1. The object *dev* doesn't exist 2. The object *sta* exists
Postconditions:	1. *d* represented the object of class Device 2. The object *d* was created 3. The attribute *Id* of the object *d* became *id* 4. The attribute *Name* of the object *d* became *name* 5. The attribute *Location* of the object *d* became *location* 6. The object *d* was linked to the object *sta* by *Contacts* 7. The object *dev* was put into the instance set of class Device 8. The return value was **true**

Contract of **createDevice**:

```
Contract AirportSystem::createDevice(id : Integer, name : String, location :
String, contactsid : Integer) : Boolean {
        definition:
            dev:Device = Device.allInstance()->any(u:Device | u.Id = id),
            sta:Staff = Staff.allInstance()->any(uu:Staff | uu.Id = contactsid)
        precondition:
            dev.oclIsUndefined() = true and
            sta.oclIsUndefined() = false
        postcondition:
            let d:Device in
            d.oclIsNew() and
            d.Id = id and
            d.Name = name and
            d.Location = location and
            d.Contacts = sta and
            Device.allInstance()->includes(dev) and
            result = true
}
```

OP3 - approve

Operation Name:	approve
Operation ID:	OP3
Description:	
Service:	RepairService
Input:	1. name: *sid*, type: Integer 2. name: *rid*, type: Integer 3. name: *reject*, type: Boolean 4. name: *suggestion*, type: String
Output Type:	ApprovalHistory
Definition:	1. *rep* is the object *u* in the instance set of class Repair. *u* represents an object of class Repair, and *u* meets: The attribute *Id* of the object *u* is equal to *rid* 2. *sta* is the object *uu* in the instance set of class Staff. *uu* represents an object of class Staff, and *uu* meets: The attribute *Id* of the object *uu* is equal to *sid*
Preconditions:	1. The object *rep* exists 2. The object *sta* exists

Postconditions:	1. *ah* represented the object of class ApprovalHistory 2. The object *ah* was created 3. The attribute *Reject* of the object *ah* became *reject* 4. The attribute *Suggestion* of the object *ah* became *suggestion* 5. The object *ah* was put into the instance set of class ApprovalHistory 6. The object *rep* was linked to the object *ah* by *History* 7. If the *reject* was not equal to **false**, take the following as postcondition(s): If the attribute *Process* of the object *rep* was equal to **0**, and the attribute *Role* of the object *sta* was equal to **1**, take the following as postcondition(s): The attribute *Process* of the object *rep* became **1** Otherwise, take the following as postcondition(s): If the attribute *Process* of the object *rep* was equal to **1**, and the attribute *Role* of the object *sta* was equal to **2**, take the following as postcondition(s): The attribute *Process* of the object *rep* became **2** Otherwise, take the following as postcondition(s): If the attribute *Process* of the object *rep* was equal to **2**, and the attribute *Role* of the object *sta* was equal to **3**, take the following as postcondition(s): The attribute *Process* of the object *rep* became **3** Otherwise, take the following as postcondition(s): The attribute *Process* of the object *rep* became **5** 8. The object *rep* was put into the instance set of class Repair 9. The return value was *ah*

Contract of **approve**:

```
Contract RepairService::approve(sid : Integer, rid : Integer, reject : Boolean,
suggestion : String) : ApprovalHistory {
        definition:
            rep:Repair = Repair.allInstance()->any(u:Repair | u.Id = rid),
            sta:Staff = Staff.allInstance()->any(uu:Staff | uu.Id = sid)
        precondition:
            rep.oclIsUndefined() = false and
            sta.oclIsUndefined() = false
        postcondition:
            let ah:ApprovalHistory in
            ah.oclIsNew() and
            ah.Reject = reject and
            ah.Suggestion = suggestion and
```

```
              ApprovalHistory.allInstance()->includes(ah) and
              rep.History->includes(ah) and
              if
                  reject <> false
              then
                  if
                      rep.Process = 0 and // STAFFREQUEST
                      sta.Role = 1 // MASTER
                  then
                      rep.Process = 1 // MASTERAPPROVE
                  else
                      if
                          rep.Process = 1 and // MASTERAPPROVE
                          sta.Role = 2 // MANAGER
                      then
                          rep.Process = 2 // MANAGERAPPROVE
                      else
                          if
                              rep.Process = 2 and // MANAGERAPPROVE
                              sta.Role = 3 // WORKER
                          then
                              rep.Process = 3 // WORKERAPPROVE
                          endif
                      endif
                  endif
              else
                  rep.Process = 5 // REJECT
              endif and
              Repair.allInstance()->includes(rep) and
              result = ah
}
```

OP4 - finishRepair

Operation Name:	finishRepair
Operation ID:	OP4
Description:	
Service:	RepairService
Input:	1. name: *id*, type: Integer 2. name: *sid*, type: Integer 3. name: *did*, type: Integer 4. name: *res*, type: String
Output Type:	Boolean
Definition:	1. *rep* is the object *u* in the instance set of class Repair. *u* represents an object of class Repair, and *u* meets: The attribute *Id* of the object *u* is equal to *id* 2. *sta* is the object *uu* in the instance set of class Staff. *uu* represents an object of class Staff, and *uu* meets: The attribute *Id* of the object *uu* is equal to *sid* 3. *dev* is the object *uuu* in the instance set of class Device. *uuu* represents an object of class Device, and *uuu* meets: The attribute *Id* of the object *uuu* is equal to *did*
Preconditions:	1. The object *dev* is linked to the object *sta* by *Contacts* 2. The attribute *Role* of the object *sta* is equal to **3**
Postconditions:	1. The attribute *Process* of the object *rep* became **7** 2. The return value was **true**

Contract of **finishRepair**:

```
Contract RepairService::finishRepair(id : Integer, sid : Integer, did : Integer,
res : String) : Boolean {
        definition:
            rep:Repair = Repair.allInstance()->any(u:Repair | u.Id = id),
            sta:Staff = Staff.allInstance()->any(uu:Staff | uu.Id = sid),
            dev:Device = Device.allInstance()->any(uuu:Device | uuu.Id = did)
        precondition:
            dev.Contacts = sta and
            sta.Role = 3
        postcondition:
            rep.Process = 7 and // FINISH
            result = true
}
```

OP5 - feedback

Operation Name:	feedback
Operation ID:	OP5
Description:	
Service:	RepairService
Input:	1. name: *id*, type: Integer 2. name: *sid*, type: Integer 3. name: *score*, type: Integer 4. name: *des*, type: String
Output Type:	Boolean
Definition:	1. *rep* is the object *u* in the instance set of class Repair. *u* represents an object of class Repair, and *u* meets: The attribute *Id* of the object *u* is equal to *id* 2. *sta* is the object *uu* in the instance set of class Staff. *uu* represents an object of class Staff, and *uu* meets: The attribute *Id* of the object *uu* is equal to *sid*
Preconditions:	1. The object *rep* is linked to the object *sta* by *RaiseStaff* 2. The attribute *Role* of the object *sta* is equal to **0** 3. The attribute *Process* of the object *rep* is equal to **7**
Postconditions:	1. The attribute *Score* of the object *rep* became *score* 2. If the *score* was greater than or equal to **3**, take the following as postcondition(s): The attribute *Close* of the object *rep* became **true** Otherwise, take the following as postcondition(s): The attribute *Close* of the object *rep* became **false** The attribute *Description* of the object *rep* became *des* The attribute *Process* of the object *rep* became **0** 3. The object *rep* was put into the instance set of class Repair 4. The return value was **true**

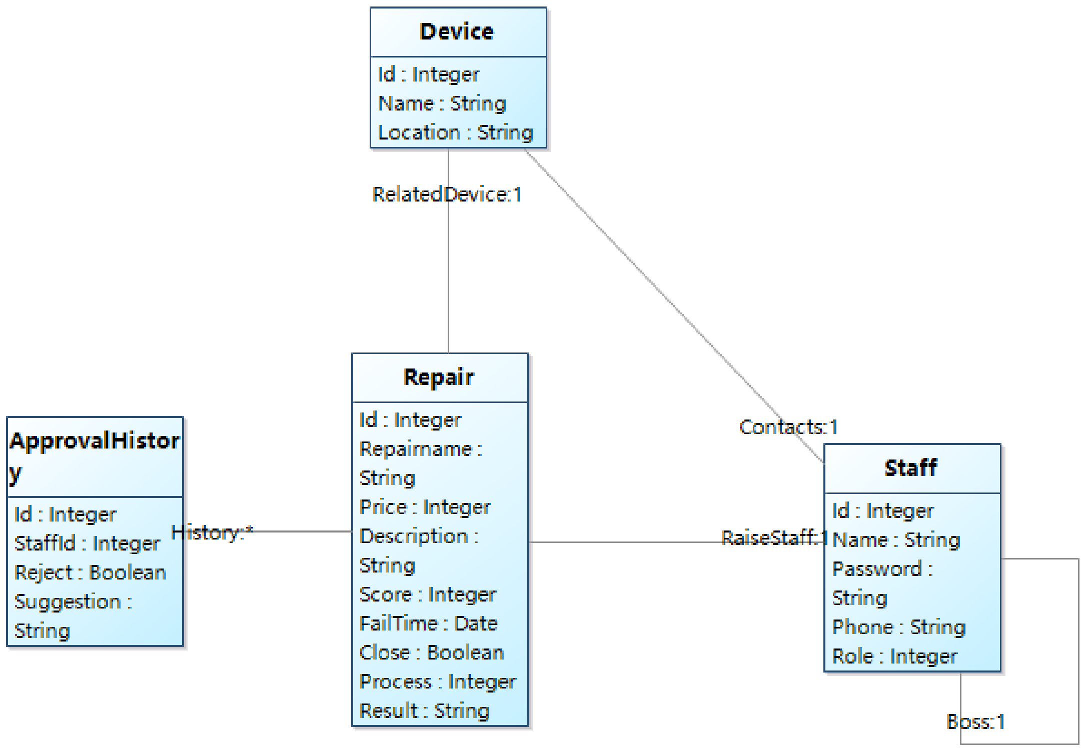
Contract of **feedback**:

```
Contract RepairService::feedback(id : Integer, sid : Integer, score : Integer,
des : String) : Boolean {
        definition:
            rep:Repair = Repair.allInstance()->any(u:Repair | u.Id = id),
            sta:Staff = Staff.allInstance()->any(uu:Staff | uu.Id = sid)
        precondition:
            rep.RaiseStaff = sta and
            sta.Role = 0 and
            rep.Process = 7 // FINISH
        postcondition:
            rep.Score = score and
            if
                score >= 3
            then
                rep.Close = true
            else
                rep.Close = false and
                rep.Description = des and
                rep.Process = 0 // 重新进入维修列表
            endif and
            Repair.allInstance()->includes(rep) and
            result = true
}
```

3.2 Database requirements

3.2.1 Entity Analysis

Conceptual Class Diagram



E1 - Repair

Entity Name:	Repair	
Entity ID:	E1	
Entity Description:		
Attribute Name	**Attribute Type**	**Attribute Description**
Id	Integer	The Id of Repair
Repairname	String	The Repairname of Repair
Price	Integer	The Price of Repair
Description	String	The Description of Repair
Score	Integer	The Score of Repair
FailTime	LocalDate	The FailTime of Repair
Close	Boolean	The Close of Repair
Process	Integer	The Process of Repair
Result	String	The Result of Repair
Relationship Name	**Related Entity**	**Relationship Type**
History	ApprovalHistory	Association: One-to-Many
RelatedDevice	Device	Association: One-to-One
RaiseStaff	Staff	Association: One-to-One

E2 - Staff

Entity Name:	Staff	
Entity ID:	E2	
Entity Description:		
Attribute Name	**Attribute Type**	**Attribute Description**
Id	Integer	The Id of Staff
Name	String	The Name of Staff
Password	String	The Password of Staff
Phone	String	The Phone of Staff
Role	Integer	The Role of Staff
Relationship Name	**Related Entity**	**Relationship Type**
Boss	Staff	Association: One-to-One

E3 - Device

Entity Name:	Device	
Entity ID:	E3	
Entity Description:		
Attribute Name	**Attribute Type**	**Attribute Description**
Id	Integer	The Id of Device
Name	String	The Name of Device
Location	String	The Location of Device
Relationship Name	**Related Entity**	**Relationship Type**
Contacts	Staff	Association: One-to-One

E4 - ApprovalHistory

Entity Name:	ApprovalHistory	
Entity ID:	E4	
Entity Description:		
Attribute Name	**Attribute Type**	**Attribute Description**
Id	Integer	The Id of ApprovalHistory
StaffId	Integer	The StaffId of ApprovalHistory
Reject	Boolean	The Reject of ApprovalHistory
Suggestion	String	The Suggestion of ApprovalHistory

3.2.2 Other database requirements

This should specify the logical requirements for any information that is to be placed into a database. This may include the following:

- a) Types of information used by various functions;
- b) Frequency of use;
- c) Accessing capabilities;
- d) Integrity constraints;
- e) Data retention requirements.

3.3 Performance requirements

3.3.1 Static numerical requirements

This subsection should specify both the static and the dynamic numerical requirements placed on the software or on human interaction with the software as a whole. Static numerical requirements may include the following:

- a) The number of terminals to be supported;
- b) The number of simultaneous users to be supported;

- c) Amount and type of information to be handled.

3.3.2 Dynamic numerical requirements

Dynamic numerical requirements may include, for example, the numbers of transactions and tasks and the amount of data to be processed within certain time periods for both normal and peak workload conditions.

All of these requirements should be stated in measurable terms.

For example,

- *95% of the transactions shall be processed in less than 1 s.*

rather than,

- *An operator shall not have to wait for the transaction to complete.*

NOTE:Numerical limits applied to one specifific function are normally specifified as part of the processing subparagraph description of that function.

3.4 Usability requirements

Define usability and quality in use requirements and objectives for the software system that can include measurable effectiveness, efficiency, satisfaction criteria and avoidance of harm that could arise from use in specific contexts of use.

3.5 Interface requirements

3.5.1 User interfaces

This should specify the following:

- a) The logical characteristics of each interface between the software product and its users. This includes those configuration characteristics (e.g., required screen formats, page or window layouts, content of any reports or menus, or availability of programmable function keys) necessary to accomplish the software requirements.
- b) All the aspects of optimizing the interface with the person who must use the system. This may simply comprise a list of do’s and don’ts on how the system will appear to the user. One example may be a requirement for the option of long or short error messages. Like all others, these requirements should be verifiable, e.g., “a clerk typist grade 4 can do function X in Z min after 1 h of training” rather than “a typist can do function X.” (This may also be specified in the Software System Attributes under a section titled Ease of Use.)

3.5.2 Hardware interfaces

This should specify the logical characteristics of each interface between the software product and the hardware components of the system. This includes configuration characteristics (number of ports, instruction sets, etc.). It also covers such matters as what devices are to be supported, how they are to be supported, and protocols. For example, terminal support may specify full-screen support as opposed to line-by-line support.

3.5.3 Software interfaces

This should specify the use of other required software products (e.g., a data management system, an operating system, or a mathematical package), and interfaces with other application systems (e.g., the linkage between an accounts receivable system and a general ledger system). For each required software product, the following should be provided:

- a) Name;
- b) Mnemonic;
- c) Specification number;
- d) Version number;
- e) Source.

For each interface, the following should be provided:

- a) Discussion of the purpose of the interfacing software as related to this software product.
- b) Definition of the interface in terms of message content and format. It is not necessary to detail any well-documented interface, but a reference to the document defining the interface is required.

3.5.4 Communications interfaces

This should specify the various interfaces to communications such as local network protocols, etc.

3.6 Design constraints

Specify constraints on the system design imposed by external standards, regulatory requirements or project limitations.

3.6.1 Standards compliance

This subsection should specify the requirements derived from existing standards or regulations. They may include the following:

- a) Report format;
- b) Data naming;
- c) Accounting procedures;
- d) Audit tracing.

For example, this could specify the requirement for software to trace processing activity. Such traces are needed for some applications to meet minimum regulatory or financial standards. An audit trace requirement may, for example, state that all changes to a payroll database must be recorded in a trace file with before and after values.

3.7 Software system attributes

3.7.1 Reliability

This should specify the factors required to establish the required reliability of the software system at time of delivery.

3.7.2 Availability

This should specify the factors required to guarantee a defined availability level for the entire system such as checkpoint, recovery, and restart.

3.7.3 Security

This should specify the factors that protect the software from accidental or malicious access, use, modification, destruction, or disclosure. Specific requirements in this area could include the need to

- a) Utilize certain cryptographical techniques;
- b) Keep specific log or history data sets;
- c) Assign certain functions to different modules;
- d) Restrict communications between some areas of the program;
- e) Check data integrity for critical variables.

3.7.4 Maintainability

This should specify attributes of software that relate to the ease of maintenance of the software itself. There may be some requirement for certain modularity, interfaces, complexity, etc. Requirements should not be placed here just because they are thought to be good design practices.

3.7.5 Portability

This should specify attributes of software that relate to the ease of porting the software to other host machines and/or operating systems. This may include the following:

- a) Percentage of components with host-dependent code;
- b) Percentage of code that is host dependent;
- c) Use of a proven portable language;
- d) Use of a particular compiler or language subset;
- e) Use of a particular operating system.

3.8 Supporting information

Additional supporting information to be considered includes:

- a) sample input/output formats, descriptions of cost analysis studies or results of user surveys;
- b) supporting or background information that can help the readers of the SRS;
- c) a description of the problems to be solved by the software; and
- d) special packaging instructions for the code and the media to meet security, export, initial loading or other requirements.

The SRS should explicitly state whether or not these information items are to be considered part of the requirements.

4 Verification

Provide the verification approaches and methods planned to qualify the software. The information items for verification are recommended to be given in a parallel manner with the information items in Section 3.

5 Appendices

5.1 Assumptions and dependencies

This subsection of the SRS should list each of the factors that affect the requirements stated in the SRS. These factors are not design constraints on the software but are, rather, any changes to them that can affect the requirements in the SRS. For example, an assumption may be that a specific operating system will be available on the hardware designated for the software product. If, in fact, the operating system is not available, the SRS would then have to change accordingly.

5.2 Apportioning of requirements

Apportion the software requirements to software elements. For requirements that will require implementation over multiple software elements, or when allocation to a software element is initially undefined, this should be so stated. A cross-reference table by function and software element should be used to summarize the apportionments.

Identify requirements that may be delayed until future versions of the system (e.g., blocks and/or increments).

5.3 Acronyms and abbreviations

This subsection should provide the acronyms and abbreviations required to properly interpret the SRS. This information may be provided by reference to one or more appendixes in the SRS or by reference to other documents.