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: MicroKnowledge 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 specifified, 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 - MicroKnowledgeSystem

Service Name:	MicroKnowledgeSystem
Service ID:	SI1
Description:	
Operation:	• logIn • searchUserByName • listMicroknowledgeOfUser • searchMicroknowledge • viewMicroknowledge • writeComment • starMicroknowledge • listStaredMicroknowledge
Temporary Variable	**Variable Description**
CurrentUser	CurrentUser is a object of User
CurrentMicroknowledge	CurrentMicroknowledge is a object of Microknowledge

SI2 - ManageUserService

Service Name:	ManageUserService
Service ID:	SI2
Description:	
Operation:	• createUser

SI3 - ManageMicroknowledgeService

Service Name:	ManageMicroknowledgeService
Service ID:	SI3
Description:	
Operation:	• createMicroknowledge • modifyMicroknowledge

SI4 - SearchMicroknowledgeService

Service Name:	SearchMicroknowledgeService
Service ID:	SI4
Description:	
Operation:	• searchMicroknowledge • viewMicroknowledge

SI5 - ListMicroknowledgeOfUserService

Service Name:	ListMicroknowledgeOfUserService
Service ID:	SI5
Description:	
Operation:	• searchUserByName • listMicroknowledgeOfUser

SI6 - ThirdPartyServices

Service Name:	ThirdPartyServices
Service ID:	SI6
Description:	
Operation:	

1.3.2 Product functions

Use Case Diagram

 Use Case Diagram

ID	Use Case Name	Use Case Description	Subfunction
UC1	searchMicroknowledge		searchMicroknowledge viewMicroknowledge
UC2	starMicroknowledge		
UC3	writeComment		
UC4	listMicroknowledgeOfUser		searchUserByName listMicroknowledgeOfUser
UC5	listStaredMicroknowledge		
UC6	manageMicroknowledge		createMicroknowledge modifyMicroknowledge
UC7	manageUser		createUser
UC8	logIn		

1.3.3 User characteristics

ID	Actor	Description	Super Actor
A1	Uploader		User
A2	Reader		User
A3	Administrator		
A4	User		

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.

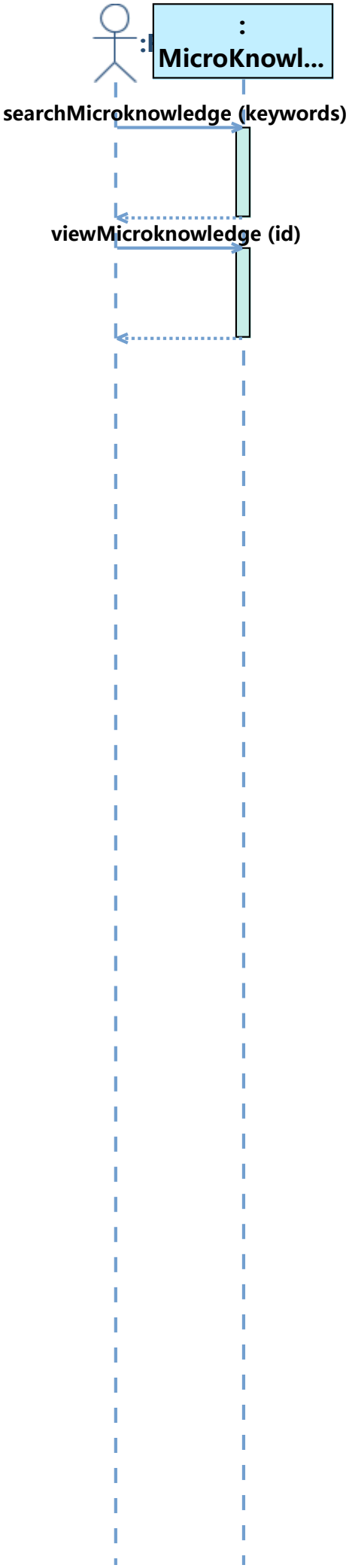
3 Requirements

3.1 Functions

3.1.1 Use Case

UC1 - searchMicroknowledge

System Sequence Diagram:


:
MicroKnowl...
searchMicroknowledge (keywords)
viewMicroknowledge (id)

Use Case Description:

UseCase Name:	searchMicroknowledge
UseCase ID:	UC1
Brief Description:	
Involved Actor:	Reader
Preconditions:	
Postconditions:	
Basic Path:	1. Reader clicks to execute the operation searchMicroknowledge, with entering keywords 2. Reader clicks to execute the operation viewMicroknowledge, with entering id
Alternative Path:	

UC2 - starMicroknowledge

System Sequence Diagram:


:
MicroKnowl...
viewMicroknowledge (id)
starMicroknowledge ()

Use Case Description:

UseCase Name:	starMicroknowledge
UseCase ID:	UC2
Brief Description:	
Involved Actor:	Reader
Preconditions:	
Postconditions:	
Basic Path:	1. Reader clicks to execute the operation viewMicroknowledge, with entering id 2. Reader clicks to execute the operation starMicroknowledge
Alternative Path:	

UC3 - writeComment

System Sequence Diagram:

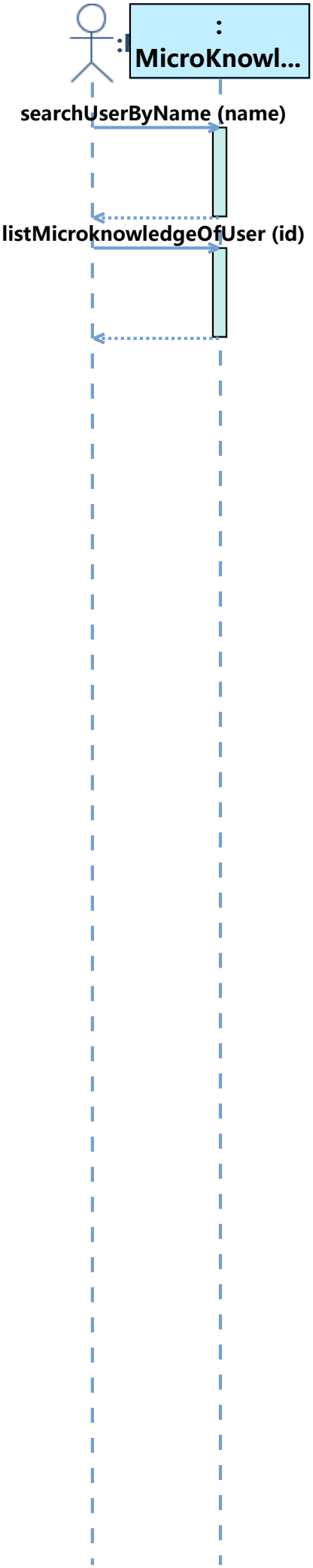

:
MicroKnowl...
viewMicroknowledge (id)
writeComment (content)

Use Case Description:

UseCase Name:	writeComment
UseCase ID:	UC3
Brief Description:	
Involved Actor:	Reader
Preconditions:	
Postconditions:	
Basic Path:	1. Reader clicks to execute the operation viewMicroknowledge, with entering id 2. Reader clicks to execute the operation writeComment, with entering content
Alternative Path:	

UC4 - listMicroknowledgeOfUser

System Sequence Diagram:


:
MicroKnowl...
searchUserByName (name)
listMicroknowledgeOfUser (id)

Use Case Description:

UseCase Name:	listMicroknowledgeOfUser
UseCase ID:	UC4
Brief Description:	
Involved Actor:	Reader
Preconditions:	
Postconditions:	
Basic Path:	1. Reader clicks to execute the operation searchUserByName, with entering name 2. Reader clicks to execute the operation listMicroknowledgeOfUser, with entering id
Alternative Path:	

UC5 - listStaredMicroknowledge

Use Case Description:

UseCase Name:	listStaredMicroknowledge
UseCase ID:	UC5
Brief Description:	
Involved Actor:	Reader
Preconditions:	
Postconditions:	
Basic Path:	
Alternative Path:	

UC6 - manageMicroknowledge

Use Case Description:

UseCase Name:	manageMicroknowledge
UseCase ID:	UC6
Brief Description:	
Involved Actor:	Uploader
Preconditions:	
Postconditions:	
Basic Path:	
Alternative Path:	

UC7 - manageUser

Use Case Description:

UseCase Name:	manageUser
UseCase ID:	UC7
Brief Description:	
Involved Actor:	Administrator
Preconditions:	
Postconditions:	
Basic Path:	
Alternative Path:	

UC8 - logIn

Use Case Description:

UseCase Name:	logIn
UseCase ID:	UC8
Brief Description:	
Involved Actor:	User
Preconditions:	
Postconditions:	
Basic Path:	
Alternative Path:	

3.1.2 System Operation

OP1 - createUser

Operation Name:	createUser
Operation ID:	OP1
Description:	
Service:	ManageUserService
Input:	1. name: *id*, type: String 2. name: *password*, type: String 3. name: *name*, type: String
Output Type:	Boolean
Definition:	*up* is the object *u* in the instance set of class User. *u* represents an object of class User, and *u* meets: The attribute *Id* of the object *u* is equal to *id*
Preconditions:	The object *up* doesn't exist
Postconditions:	1. *user* represented the object of class User 2. The object *user* was created 3. The attribute *Id* of the object *user* became *id* 4. The attribute *Password* of the object *user* became *password* 5. The attribute *Name* of the object *user* became *name* 6. The object *user* was put into the instance set of class User 7. The return value was **true**

Contract of **createUser**:

```
Contract ManageUserService::createUser(id : String, password : String, name :
String) : Boolean {
        definition:
            up:User = User.allInstance()->any(u:User | u.Id = id)
        precondition:
            up.oclIsUndefined() = true
        postcondition:
            let user:User in
            user.oclIsNew() and
            user.Id = id and
            user.Password = password and
            user.Name = name and
```

```
            User.allInstance()->includes(user) and
            result = true
}
```

OP2 - logIn

Operation Name:	logIn
Operation ID:	OP2
Description:	
Service:	MicroKnowledgeSystem
Input:	1. name: *userId*, type: String 2. name: *password*, type: String
Output Type:	Boolean
Definition:	*user* is the object *u* in the instance set of class User. *u* represents an object of class User, and *u* meets: The attribute *Id* of the object *u* is equal to *userId*
Preconditions:	1. The object *user* exists 2. The attribute *Password* of the object *user* is equal to *password*
Postconditions:	1. The object CurrentUser became *user* 2. The return value was **true**

Contract of **logIn**:

```
```

OP3 - createMicroknowledge

Operation Name:	createMicroknowledge
Operation ID:	OP3
Description:	
Service:	ManageMicroknowledgeService
Input:	1. name: *id*, type: String 2. name: *keywords*, type: String 3. name: *content*, type: String
Output Type:	Boolean
Definition:	*microknowledge* is the object *m* in the instance set of class Microknowledge. *m* represents an object of class Microknowledge, and *m* meets: The attribute *Id* of the object *m* is equal to *id*
Preconditions:	The object *microknowledge* doesn't exist
Postconditions:	1. *mk* represented the object of class Microknowledge 2. The object *mk* was created 3. The attribute *Id* of the object *mk* became *id* 4. The attribute *Keywords* of the object *mk* became *keywords* 5. The attribute *Content* of the object *mk* became *content* 6. The attribute *StarNumber* of the object *mk* became **0** 7. The attribute *LastEditTime* of the object *mk* was equal to *Now* 8. The object *CurrentUser* was linked to the object *mk* by *ContainedMicroknowledge* 9. The object *mk* was linked to the object *CurrentUser* by *BelongedUser* 10. The object *mk* was put into the instance set of class Microknowledge 11. The return value was **true**

Contract of **createMicroknowledge**:

```
Contract ManageMicroknowledgeService::createMicroknowledge(id : String, keywords
: String, content : String) : Boolean {
        definition:
            microknowledge:Microknowledge = Microknowledge.allInstance()-
>any(m:Microknowledge | m.Id = id)
        precondition:
            microknowledge.oclIsUndefined() = true
```

```
        postcondition:
            let mk:Microknowledge in
            mk.oclIsNew() and
            mk.Id = id and
            mk.Keywords = keywords and
            mk.Content = content and
            mk.StarNumber = 0 and
            mk.LastEditTime.isEqual(Now) and
            CurrentUser.ContainedMicroknowledge->includes(mk) and
            mk.BelongedUser = CurrentUser and
            Microknowledge.allInstance()->includes(mk) and
            result = true
}
```

OP4 - modifyMicroknowledge

Operation Name:	modifyMicroknowledge
Operation ID:	OP4
Description:	
Service:	ManageMicroknowledgeService
Input:	1. name: *id*, type: String 2. name: *keywords*, type: String 3. name: *content*, type: String
Output Type:	Boolean
Definition:	*mk* is the object *m* in all objects which *CurrentUser* is linked to by *ContainedMicroknowledge*. *m* represents an object of class Microknowledge, and *m* meets: The attribute *Id* of the object *m* is equal to *id*
Preconditions:	The object *mk* exists
Postconditions:	1. The attribute *Id* of the object *mk* became *id* 2. The attribute *Keywords* of the object *mk* became *keywords* 3. The attribute *Content* of the object *mk* became *content* 4. The attribute *StarNumber* of the object *mk* became its previous value 5. The attribute *LastEditTime* of the object *mk* was equal to *Now* 6. The return value was **true**

Contract of **modifyMicroknowledge**:

```
Contract ManageMicroknowledgeService::modifyMicroknowledge(id : String, keywords
: String, content : String) : Boolean {
        definition:
            mk:Microknowledge = CurrentUser.ContainedMicroknowledge-
>any(m:Microknowledge | m.Id = id)
        precondition:
            mk.oclIsUndefined() = false
        postcondition:
            mk.Id = id and
            mk.Keywords = keywords and
            mk.Content = content and
            mk.StarNumber = mk.StarNumber@pre and
            mk.LastEditTime.isEqual(Now) and
            result = true
}
```

OP5 - searchUserByName

Operation Name:	searchUserByName
Operation ID:	OP5
Description:	
Service:	MicroKnowledgeSystem
Input:	name: *name*, type: String
Output Type:	Set of User
Preconditions:	The *name* is not equal to **null**
Postconditions:	The return value was the set of class User, including all *u* in the instance set of class User. *u* represented an object of class User, and *u* meet: The attribute *Name* of the object *u* was equal to *name*

Contract of **searchUserByName**:

```
Contract MicroKnowledgeSystem::searchUserByName(name : String) : Set(User) {
        precondition:
            name <> ""
        postcondition:
            result = User.allInstance()->select(u:User| u.Name = name)
}
```

OP6 - listMicroknowledgeOfUser

Operation Name:	listMicroknowledgeOfUser
Operation ID:	OP6
Description:	
Service:	MicroKnowledgeSystem
Input:	name: *id*, type: String
Output Type:	Set of Microknowledge
Definition:	*user* is the object *u* in the instance set of class User. *u* represents an object of class User, and *u* meets: The attribute *Id* of the object *u* is equal to *id*
Preconditions:	The object *user* exists
Postconditions:	The return value was the object which *user* was linked to by *ContainedMicroknowledge*

Contract of **listMicroknowledgeOfUser**:

```
Contract MicroKnowledgeSystem::listMicroknowledgeOfUser(id : String) :
Set(Microknowledge) {
        definition:
            user:User = User.allInstance()->any(u:User | u.Id = id)
        precondition:
            user.oclIsUndefined() = false
        postcondition:
            result = user.ContainedMicroknowledge
}
```

OP7 - searchMicroknowledge

Operation Name:	searchMicroknowledge
Operation ID:	OP7
Description:	
Service:	MicroKnowledgeSystem
Input:	name: *keywords*, type: String
Output Type:	Set of Microknowledge
Preconditions:	The *keywords* is not equal to **null**
Postconditions:	The return value was the set of class Microknowledge, including all *mk* in the instance set of class Microknowledge. *mk* represented an object of class Microknowledge, and *mk* meet: The attribute *Keywords* of the object *mk* was equal to *keywords*

Contract of **searchMicroknowledge**:

```
Contract MicroKnowledgeSystem::searchMicroknowledge(keywords:String) :
Set(Microknowledge) {
        precondition:
            keywords <> ""
        postcondition:
            result = Microknowledge.allInstance()->select(mk:Microknowledge|
mk.Keywords = keywords)
}
```

OP8 - viewMicroknowledge

Operation Name:	viewMicroknowledge
Operation ID:	OP8
Description:	
Service:	MicroKnowledgeSystem
Input:	name: *id*, type: String
Output Type:	Microknowledge
Definition:	*mk* is the object *m* in the instance set of class Microknowledge. *m* represents an object of class Microknowledge, and *m* meets: The attribute *Id* of the object *m* is equal to *id*
Preconditions:	The object *mk* exists
Postconditions:	1. ERROR12 2. The return value was *mk*

Contract of **viewMicroknowledge**:

```
Contract MicroKnowledgeSystem::viewMicroknowledge(id : String) : Microknowledge
{
        definition:
            mk:Microknowledge = Microknowledge.allInstance()-
>any(m:Microknowledge | m.Id = id)
        precondition:
            mk.oclIsUndefined() = false
        postcondition:
            CurrentMicroknowledge = mk and
            result = mk
}
```

OP9 - starMicroknowledge

Operation Name:	starMicroknowledge
Operation ID:	OP9
Description:	
Service:	MicroKnowledgeSystem
Input:	None
Output Type:	Boolean
Preconditions:	1. The object *CurrentMicroknowledge* exists 2. The object *CurrentUser* exists
Postconditions:	1. The object *CurrentUser* was linked to the object *CurrentMicroknowledge* by *StaredMicroknowledge* 2. The attribute *StarNumber* of the object *CurrentMicroknowledge* became its previous value plus **1** 3. The return value was **true**

Contract of **starMicroknowledge**:

```
Contract MicroKnowledgeSystem::starMicroknowledge() : Boolean {
        precondition:
            CurrentMicroknowledge.oclIsUndefined() = false and
            CurrentUser.oclIsUndefined() = false
        postcondition:
            CurrentUser.StaredMicroknowledge->includes(CurrentMicroknowledge)
and
            CurrentMicroknowledge.StarNumber =
CurrentMicroknowledge.StarNumber@pre+1 and
            result = true
}
```

OP10 - writeComment

Operation Name:	writeComment
Operation ID:	OP10
Description:	
Service:	MicroKnowledgeSystem
Input:	name: *content*, type: String
Output Type:	Boolean
Preconditions:	1. The object *CurrentMicroknowledge* exists 2. The object *CurrentUser* exists
Postconditions:	1. *comment* represented the object of class Comment 2. The object *comment* was created 3. The attribute *Content* of the object *comment* became *content* 4. The attribute *WritingTime* of the object *comment* was equal to *Now* 5. The object *comment* was put into the instance set of class Comment 6. The object *CurrentMicroknowledge* was linked to the object *comment* by *ContainedComment* 7. The object *comment* was linked to the object *CurrentMicroknowledge* by *CommenttoMicroknowledge* 8. The object *CurrentUser* was linked to the object *comment* by *ReadertoComment* 9. The object *comment* was linked to the object *CurrentUser* by *CommenttoReader* 10. The return value was **true**

Contract of **writeComment**:

```
Contract MicroKnowledgeSystem::writeComment(content:String) : Boolean {
        precondition:
            CurrentMicroknowledge.oclIsUndefined() = false and
            CurrentUser.oclIsUndefined() = false
        postcondition:
            let comment:Comment in
            comment.oclIsNew() and
            comment.Content = content and
            comment.WritingTime.isEqual(Now) and
            Comment.allInstance()->includes(comment) and
            CurrentMicroknowledge.ContainedComment->includes(comment) and
            comment.CommenttoMicroknowledge = CurrentMicroknowledge and
            CurrentUser.ReadertoComment->includes(comment) and
            comment.CommenttoReader = CurrentUser and
```

```
            result = true
}
```

OP11 - listStaredMicroknowledge

Operation Name:	listStaredMicroknowledge
Operation ID:	OP11
Description:	
Service:	MicroKnowledgeSystem
Input:	None
Output Type:	Set of Microknowledge
Preconditions:	The object *CurrentUser* exists
Postconditions:	The return value was the object which *CurrentUser* was linked to by *StaredMicroknowledge*

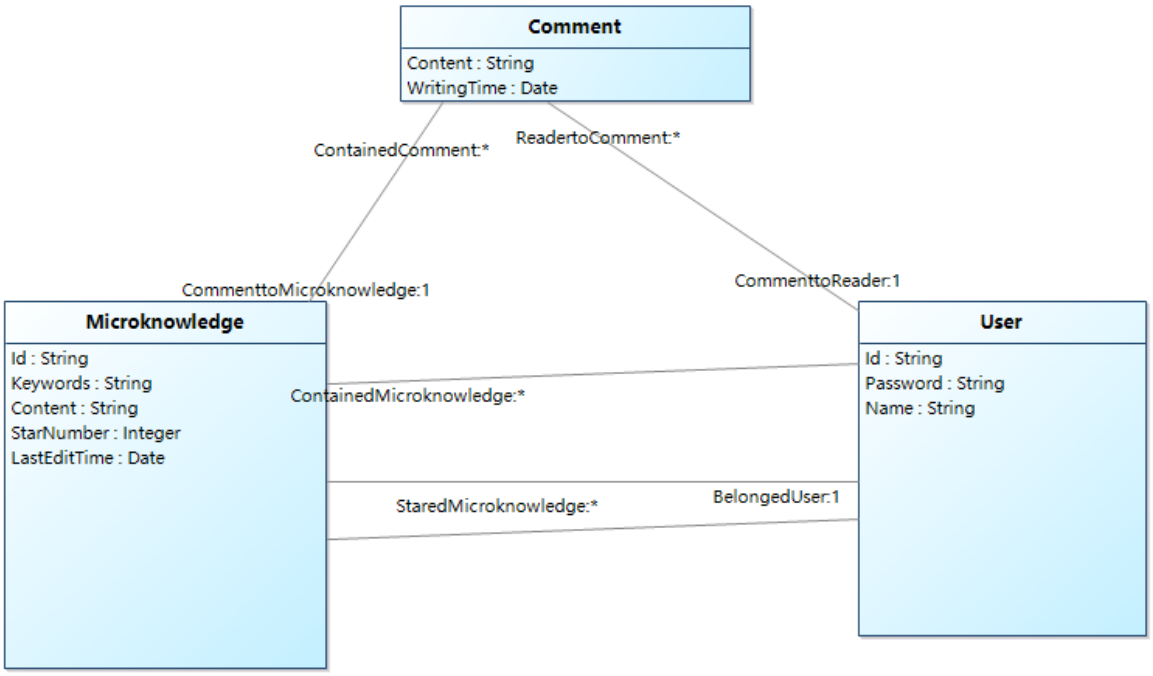
Contract of **listStaredMicroknowledge**:

```
Contract MicroKnowledgeSystem::listStaredMicroknowledge() : Set(Microknowledge)
{
        precondition:
            CurrentUser.oclIsUndefined() = false
        postcondition:
            result = CurrentUser.StaredMicroknowledge
}
```

3.2 Database requirements

3.2.1 Entity Analysis

Conceptual Class Diagram



E1 - User

Entity Name:	User	
Entity ID:	E1	
Entity Description:		
Attribute Name	**Attribute Type**	**Attribute Description**
Id	String	The Id of User
Password	String	The Password of User
Name	String	The Name of User
Relationship Name	**Related Entity**	**Relationship Type**
StaredMicroknowledge	Microknowledge	Association: One-to-Many
ReadertoComment	Comment	Association: One-to-Many
ContainedMicroknowledge	Microknowledge	Association: One-to-Many

E2 - Microknowledge

Entity Name:	Microknowledge	
Entity ID:	E2	
Entity Description:		
Attribute Name	**Attribute Type**	**Attribute Description**
Id	String	The Id of Microknowledge
Keywords	String	The Keywords of Microknowledge
Content	String	The Content of Microknowledge
StarNumber	Integer	The StarNumber of Microknowledge
LastEditTime	LocalDate	The LastEditTime of Microknowledge
Relationship Name	**Related Entity**	**Relationship Type**
BelongedUser	User	Association: One-to-One
ContainedComment	Comment	Association: One-to-Many

E3 - Comment

Entity Name:	Comment	
Entity ID:	E3	
Entity Description:		
Attribute Name	**Attribute Type**	**Attribute Description**
Content	String	The Content of Comment
WritingTime	LocalDate	The WritingTime of Comment
Relationship Name	**Related Entity**	**Relationship Type**
CommenttoReader	User	Association: One-to-One
CommenttoMicroknowledge	Microknowledge	Association: One-to-One

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.