Formal Specification

KSU Student Portal

Version 1.0

Submitted in partial fulfillment of the requirements of the degree of MSE

Javier Ramos Rodríguez
CIS 895 – MSE Project
Kansas State University
# Table of Contents

1. *Introduction*  
   1.1. Alloy  
   1.2. USE  
2. *Alloy Model*  
   2.1. Description  
   2.2. Model  
3. *Use Model*  
   3.1. Description  
   3.2. Model  
   3.3. Class Diagram  
   3.4. Object Diagram
1. Introduction

The goal of this document is to formally specify the application in order to find possible errors in early stages of development. The application of formal methods in the development of the system will help to assure a good quality level in the final product verifying that all possible system states are reached and the desirable behavior is executed.

1.1. Alloy

The **Alloy** Constraint Language will be used in phase two to formal specify the overall system without given much details about how is going to be implemented. Alloy is a non-deterministic language giving the possibility to find unstable states that we may forget about using a deterministic approach. Alloy only works with **Sets** and its relations. First, we write the Alloy model and the tool will generate instances of it checking all possible invariants.

We define the following Items:

- **Domains**: Here we define the possible values that the sets can take.
- **Sets**: Unordered collections of objects that take values from the domains.
- **Relations**: Relationships between sets
- **Multiplicities**: Constrains on the number of objects that participate in a given relation.
- **Invariants**: Constrains that the model should hold.
- **Assertions**: Are useful to check if a specific invariant holds. If the Alloy Constraint Analyzer finds a counterexample means that the invariant is incorrect.
- **Operations**: Functions that the system can execute. We define pre, post and frame conditions.
1.2. USE

USE will be used to formally check the UML diagrams. Concretely the class diagrams. We will use the **OCL** language to specify constraints in the **UML** classes and its relations.

USE uses a deterministic approach and it's used in later stages of development because we need to have the complete UML model and then check it with USE. So, in this phase since we have little information about the system and the UML documentation is not done, only a small part of the system will be checked using USE.
2. Alloy Model

2.1. Description

We are going to model the main functionalities of the system and its relations. We will model the User, Blog, Blog Entries, Messages, Events, Articles, Links, Profile and other attributes.

The user will be the main item and we are going to model a couple of his/her operations to check for possible unstable states.
2.2. Model

model MSEProject {

domain { User, fixed Blog, BlogEntry, Profile, Message, Link, Article, Event, Filter, fixed Country, fixed Language, fixed Course }

state {

//1) Users can have only one Blog and a Blog corresponds to exactly one user
user_blog(~blog_user): static User! -> static Blog!

//2) A blog can have several entries and one entry belongs to one blog
blog_entries(~entry_blog): static Blog! -> BlogEntry

//3) A User will have associated a profile and a profile belong to one user
user_profile (~profile_user): static User! -> Profile!

//4) A Article will have associated a filter and a fileter belong to one Article
article_filter (~filter_article): static Article! -> static Filter!

//5) A Event will have associated a filter and a filter belong to one Event
event_filter (~filter_event): static Event! -> static Filter!

//6) A Filter will have associated one course and one course can belong to many filters
filter_course(~course_filter): Filter! -> Course?

//7) A Filter will have associated one country and one country can belong to many filters
}
filter_country(~country_filter): Filter! -> Country?

//8) A Filter will have associated one language and one language can belong to many filters

filter_language(~language_filter): Filter! -> Language?

//9) A Profile will have several Courses associated and a Course can be associated to many Profiles

profile_course(~course_profile): Profile! -> Course

//10) A Profile will have one Country associated and a Country can be associated to many Profiles

profile_country(~country_profile): Profile -> Country!

//11) A Profile will have several Languages associated and a Language can be associated to many Profiles

profile_language(~language_profile): Profile -> Language

//12) An User can send a message to many users and one message belongs to one user

author_message(~message_author): User! -> Message

//13) An User can receive several messages and one message can be sent to many users

message_to(~message_dest): Message! -> User

//14) An User can publish many events and one event is published by one User

publish_event(~event_published): User! -> Event?

//15) An User can publish many articles and one article is published by one User
publish_article(~article_published): User! -> Article?

//16) An User can publish many links and one link is published by one User

publish_link(~link_published): User! -> Link?

inv message {
all u1:User, u2:User | u1.author_message.message_to=u2 -> u1 in u2.message_dest.message_author
}

assert message_a {
all u1:User, u2:User | u1.author_message.message_to=u2 -> u1 in u2.message_dest.message_author
}

inv realism {
    some u : User | some (User - u)
    some p : Profile | some (Profile - p)
}

op addBEntry(be:BlogEntry'!,b:Blog!){
be !in BlogEntry
BlogEntry'=BlogEntry+be
b.blog_entries'=b.blog_entries+be
Message'=Message
User'=User
Event'=Event
Article'=Article
Link'=Link
Profile'=Profile
Filter'=Filter
}
op sendMessage(m:Message!, to:User!, from:User!){
  m !in Message
  Message'=Message+m
  m.message_author'= m.message_author+from
  m.message_to'= m.message_to+to
  BlogEntry'=BlogEntry
  User'=User
  Event'=Event
  Article'=Article
  Link'=Link
  Profile'=Profile
  Filter'=Filter
}

op createEvent(e:Event!, author:User!, f:Filter!){
  e !in Event
  f !in Filter

  Filter'=Filter+f
  Event'=Event+e
  e.event_filter' = f
  e.event_published' = author

  Message'=Message
  User'=User
  Profile'=Profile
}
}
3. Use Model

3.1. Description

In the USE model we only check the user and his/her relations with the blog and the messages. We’ll check also the relation between the user and the events.

3.2. Model

```ruby
model MSEProject

class User
  attributes
  login:String
end

class Profile
  attributes
  p_id:Integer
end

class Blog
  attributes
  b_id: Integer
end

class BlogEntry
  attributes
  be_id: Integer
end

class Message
  attributes
  m_id: Integer
end

class Event
  attributes
  e_id: Integer
end

class Filter
  attributes
  f_id: Integer
end
```
association UserBlog between
  User[1] role blogUser
  Blog[1] role userBlog
end

association UserProfile between
  User[1] role profileUser
  Profile[1] role userProfile
end

association BlogEntry between
  Blog[1] role entryBlog
  BlogEntry[*] role blogEntry
end

association MessageAuthor between
  User[1] role writtenby
  Message[*] role writes
end

association MessageDest between
  Message[1] role message
  User[*] role dest
end

association PublishEvent between
  User[1] role eventPublished
  Event[*] role publishEvent
end

association EventFilter between
  Event[1] role filterEvent
  Filter[1] role eventFilter
End
3.3 Class Diagram
3.4 Object Diagram