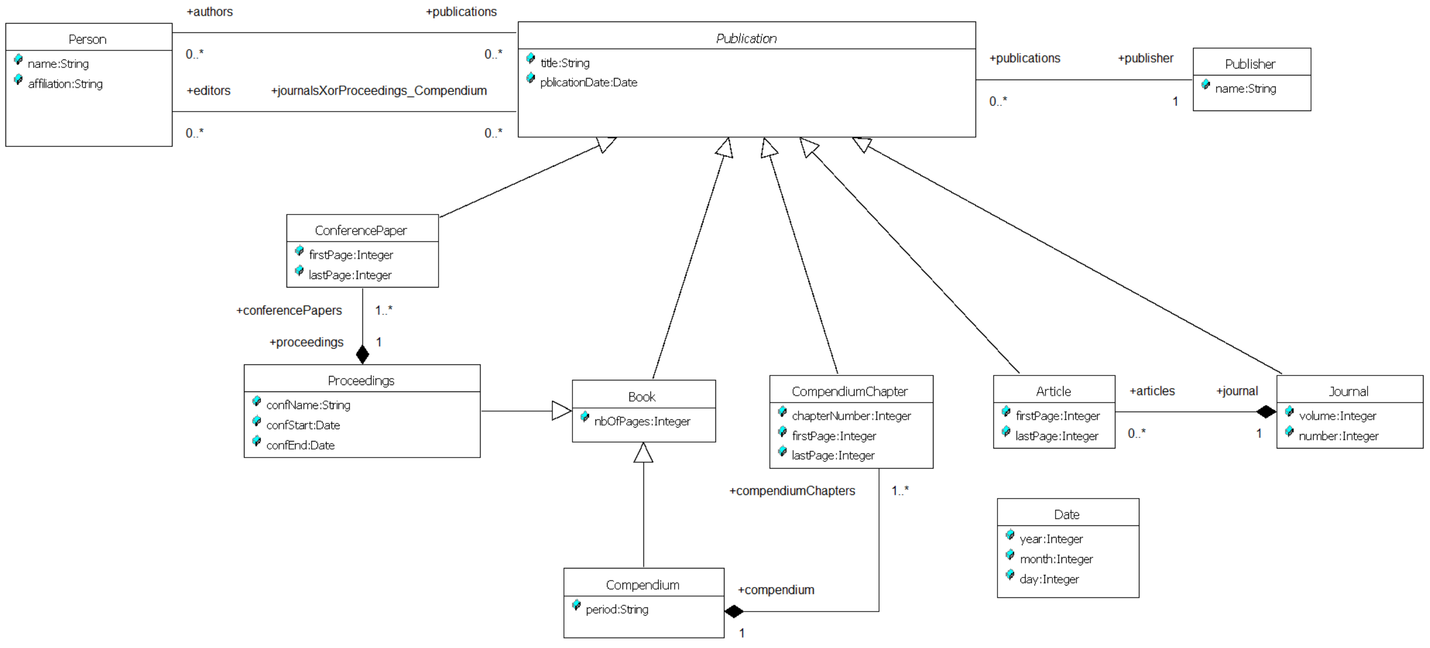
SE 11-07-2019

I. Persons may have as author or editor different kind of publications – (both roles are excluded). Each publication has a title, a publication date and is associated with a publisher which has a name. Persons are characterized by name and affiliation. There are different kind of publications: article, journal, conference paper, book, proceedings, compendium and compendium chapter. Articles written by author(s), are included in a journal and are characterized by their first and last page. Journals are edited by editors and are characterized by volume and number. A book is written by author(s) and is featured by the number of pages. Compendium and proceedings are two special kind of books. A compendium contains a set of compendium chapters, each featured by a chapter number, a first and last page. By consequence, each compendium is edited (by editors) and each compendium chapter is authored. Proceedings contain conference papers accepted and presented to a conference. Proceedings are edited (by editors) and are featured by the conference name, conference's start and end date. Each conference paper is characterized by first and last page.

   a. By means of a UML class diagram, please represent the architecture of a model complying with the above-mentioned requirements. 2 pts
   b. In the context of Publication class, please specify an invariant stating that each publication may authored exclusive or edited, and in case in which is edited, the publication kind must be one of the: Journal, Compendium, Proceedings. 1 pt
   c. In the context of Person, please specify an observer returning the articles authored by the current person as unique author. 1 pt



```
context Publication
    inv editors:
        (not self.editors->isEmpty implies ((self.oclIsTypeOf(Journal) xor
self.oclIsTypeOf(Compendium)) xor self.oclIsTypeOf(Proceedings))) xor (not
self.authors->isEmpty)


context Person
    def pubWithUniqueAuth:
      let articlesWithUniqueAuth:Set(Article) = self.publications->select(p:Publication
         | p.authors->size = 1 and p.oclIsTypeOf(Article)).oclAsType(Article)->asSet
```

II. What do we mean by requirement elicitations? In this context, please shortly describe requirement elicitations activities. Please mention the information used in the template used to describe use cases. Name the UML view specified by means of use cases.

(0.5+ 0.5 + 1) pts

Requirements elicitation focuses on describing the purpose of the system. The client, the developers, and the users identify a problem area and define a system that addresses the problem. Such a definition is called a **requirements specification** and serves as a contract between the client and the developers.

Requirements elicitation includes the following activities:

- **Identifying actors**. During this activity, developers identify the different types of users the future system will support.

- **Identifying scenarios**. During this activity, developers observe users and develop a set of detailed scenarios for typical functionality provided by the future system. Scenarios are concrete examples of the future system in use. Developers use these scenarios to communicate with the user and deepen their understanding of the application domain.
- **Identifying use cases**. Once developers and users agree on a set of scenarios, developers derive from the scenarios a set of use cases that completely represent the future system. Whereas scenarios are concrete examples illustrating a single case, use cases are abstractions describing all possible cases. When describing use cases, developers determine the scope of the system.
- **Refining use cases**. During this activity, developers ensure that the requirements specification is complete by detailing each use case and describing the behavior of the system in the presence of errors and exceptional conditions.
- **Identifying relationships among use cases**. During this activity, developers identify dependencies among use cases. They also consolidate the use case model by factoring out common functionality. Between use cases we may have: include relationships, extend relationships and generalization/specialization relationships
- **Identifying nonfunctional requirements**. During this activity, developers, users, and clients agree on aspects that are visible to the user, but not directly related to functionality. These include constraints on the performance of the system, its documentation, the resources it consumes, its security, and its quality.

A use case specifies all possible scenarios for a given piece of functionality. A use case is initiated by an actor. After its initiation, a use case may interact with other actors, as well. A use case represents a complete flow of events through the system in the sense that it describes a series of related interactions that result from its initiation.

The template used to describe use cases contains:

- Use case name
- Participating actors
- Flow of events
  - Normal (Main) flow of events
  - Exceptional flow of events (described by means of extend relationships and by mentioning extension points)
- Entry conditions
- Exit conditions
- Quality requirements

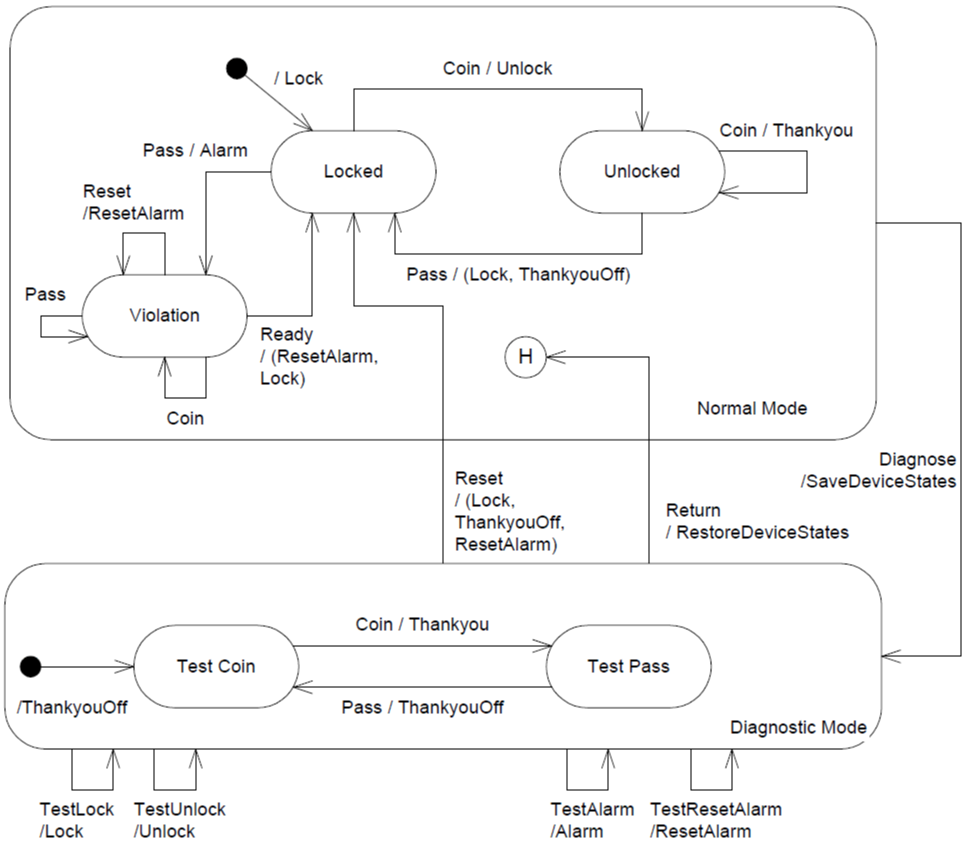
The view described by means of use cases if the functional view.

III. Please mention the techniques that can be used in order to deal with faults. 1 pt

The techniques used to deal with faults depends on the moment of system development and system usage we are. So:

- **Before the system release**, the techniques used for fault avoidance are:
    - The use of a methodology to reduce complexity,
    - The configuration management to prevent inconsistencies,
    - Doing verifications to prevent algorithmic faults,
    - Use Reviews
- **While system (a part of the system or the whole system) is running**, techniques supporting **Fault detection**:
    - **Testing**: Activities to provoke failures in a planned way,
    - **Debugging**: Finding and removing the cause (Faults) of an observed failure,
    - **Monitoring**: Delivering information about state => Used during debugging
- **Recover from failure after release**, techniques supporting **Fault tolerance**:
    - Exception handling,
    - Modular redundancy.

IV. Please name and describe shortly the UML concepts represented in the diagram represented on the next page and explain the functionality specified by the diagram. Also, please name the kind of UML diagram, and the UML view in which this diagram is used. (1 + 1) pts



The UML concepts represented in this diagram are:

1. **composed states**: NormalMode and DiagnosticMode
2. **concrete states**: Violation, Locked, Unlocked, TestCoin and TestPassReady
3. **pseudo states**:
    a. 2 **input states** (one in each composed state) and,
    b. a **history state** in NormalMode
4. **transitions**: between two different states or in the same state as those triggered by stimuli: Pass, Reset, Coin
5. **events** (stimuli): Pass, Reset, Coin, Diagnose, Return, TestLock, TestUnlock, TestAlarm and TestResetAlarm
6. **actions**: Alarm, Lock, Unlock, Thankyou a.s.o.

After powering on the system, the action Lock will be executed, and the system will be in the state Locked. In this state, the events that will trigger a transition are: Coin and Pass. Coin trigger the execution of Unlock action and the transition in the state Unlock. Pass will trigger the execution of the Alarm action and the transition in the state Violation. The behavior corresponding to the remained concrete states: Violation, Unlocked, TestCoin and TestPassReady is similar.
If the system is in the composed state NormalMode (that is in one of its inner concrete states) and a Diagnose event will appear, then, a transition will be executed: from the inner concrete state of NormalMode in the inner concrete state TestCoin of the composed state DiagnosticMode. The semantics of the auto-transitions specified for the composed state DiagnosticMode is that the system will remain in the concrete state in which it is and, the corresponding action of each event will be executed. If the system is in the composed state DiagnosticMode and the Return event will appear, then, the ReturnDeviceStates will be executed and the system will transit in the last concrete state of NormalMode in which the system was before to transit toward TestCoin. This happen due to the pseudostate History.

It is a State Transition Diagram (STD), which is used to specify the behavioral view of reactive objects.