Modelling Dining Philosopher Problem in DOSTE and EDEN

0955502

Abstract

The Dining Philosopher Problem is a classical example illustrating "mutual exclusion" in concurrent system[1]. In this paper, the Empirical Modelling (EM) approaches will be used to analyze and model the dependency between chopsticks and the philosophers, as well as some more new features, e.g. chopstick machines. The objective for this project is to model the same problem using two different EM languages: in DOSTE and in EDEN, as well as to show how to link DOSTE to EDEN GUI. Finally, this paper will show some advantages and limitations of EM languages.

1 Introduction

The Dining Philosopher Problem has always been illustrated as an example for the concurrency concept, e.g. "deadlock", "critical section" and "mutual exclusion". In this paper, a model has been built based on the "waiter solution", which has already solved the deadlock problem by using the "eating permission" from the waiter. However, all the philosophers can not have dinner together at the same time, as the number of the chopsticks is exactly the same as the number of the philosophers, whereas one philosopher needs two chopsticks to eat. To solve this problem, a revised dining philosopher problem model has been built by adding the chopstick machines which could provide chopsticks whenever requested.

The aims of this model:

First, modelling the Dining Philosopher Problem using Doste and Eden respectively, analyzing the differences.

Second, showing how to link Doste to Eden GUI.

Finally, compared with the implementing process in other advanced language, and show some advantages and limitations about EM and its languages.

2 Model specification

2.1 Layout Specification

As can be seen from Figure 1

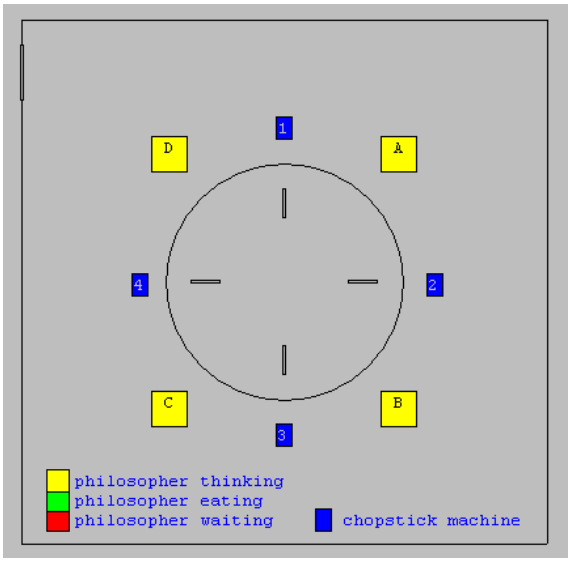


Figure 1: Dining Philosopher Layout

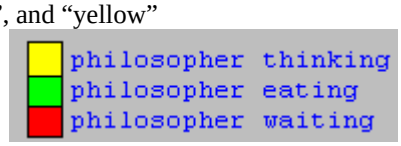
There will be 4 philosophers sitting around a dinning table, and their positions are: philosopher A at Northeast (NE), philosopher B at Southeast (SE), philosopher C at Southwest (SW), philosopher D at Northwest (NW).

There will be 4 chopsticks, each is on the table between each 2 philosophers, and their positions are: chopstick 1 at North, 2 at East, 3 at South, 4 at West.

There will be 4 chopstick machines; positions are the same as the chopsticks, but outside the table.
There will be a door and it is an indicator which can tell whether all the 4 philosophers have the same status.

2.2 Functional Specification and Assumption:

Each philosopher has 3 statuses: “eating”, “waiting” and “thinking”. And the 3 statuses will be represented by 3 colours respectively: “green”, “red”, and “yellow”



The precondition for the philosopher’s success in eating is that there has to be 2 chopsticks at his left and right hand sides, as well as eating action has been taken at the same time.

Each philosopher shall be in the “waiting” status autonomously if there are less than 2 chopsticks from his left and right hand sides.

Each philosopher shall be in the “thinking” status autonomously if there are 2 chopsticks from his left and right hand sides, but no eating action has been taken.

Eating action can only be taken when the current status is “thinking”. When eating action has been taken, the philosopher’s status will be “eating” and the 2 chopsticks will disappear from the table correspondingly.

Stopping action can only be taken when the current status is “eating”. When stopping action has been taken, the 2 chopsticks will be put back to the table by the philosopher correspondingly.

Each chopstick machine can provide 1 chopstick when there is no corresponding chopstick on the table and requesting action has been taken.

Each chopstick machine will keep the chopstick number at most 1 at one corresponding position on the table by collecting the extra one autonomously.

The door shall be closed if and only if the 4 philosophers are in the same status.

3 Modelling

3.1 Architecture

As can be seen from Figure2, the correlated experiments and observations are involving Construal and Referent [2]. Chapter 2 can be regarded as the domain referent specification. To get the expected output, a system to make sense of the input is required. Figure3 shows the architectures in Doste Version and in Eden Version. Apart from sharing the same layout file, they are different systems. Chapter 3.3 and chapter 3.5 will give more details.

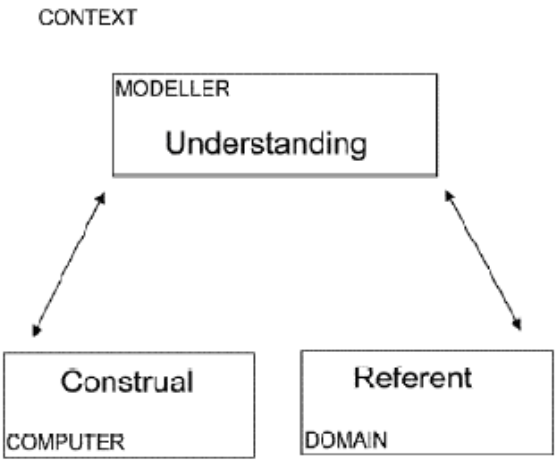


Figure2: Construal and Referent [2]

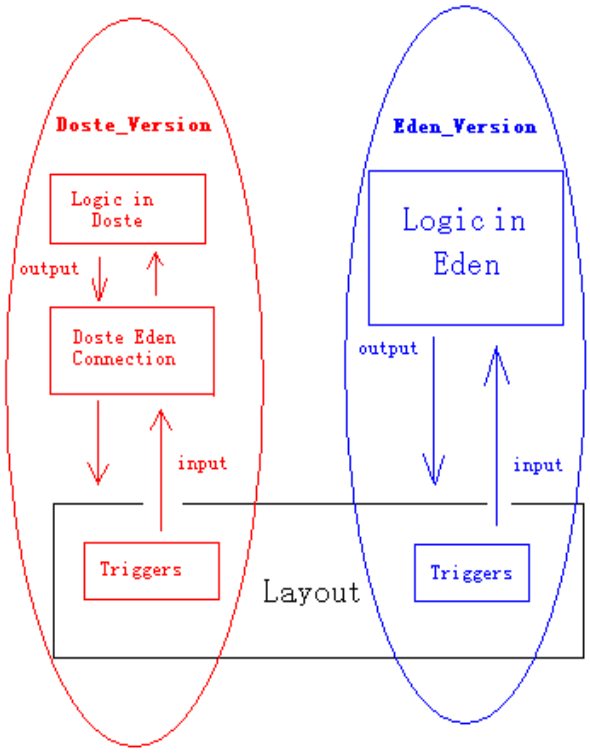


Figure 3: Architectures in Doste and Eden

3.2 Modelling using Doste

The Modelling process using Doste is quite straight forward, in this model, there are agents like: philosopher A B C D, chopstick 1,2,3,4, chopstickMachine1,2,3,4 and door. Each agent has its own observables, and the basic idea by using Doste is just like most of the declarative language, express it, tell the system “what is it”, rather than “how to do it” in Eden.

The keyword "is" in Doste is understood as "become", telling what the state of the observable is going to be in the next instant. This is a powerful feature, as all the observables can autonomously change it state depends on other observables' states.

However, when there are a large amount of observables, the dependency logic can be quite complicated, as there are no functions in Doste.

Unlike the procedures or functions in Eden or other languages which can change more than one observables at one execution. Take the observables chop1, chop2 in the Eden Version for instance: as can be seen from below, if the procedure has been invoked, the values of observables chop1, chop2 can be changed together.

```
proc statusIsEating_A:stopA,tryA
   {
     if(stopA==0){
             if(tryA==1)
             {statusIsEatingA=1;
              chop1=0;
              chop2=0;
              }
      }else {statusIsEatingA=0;
             chop1=1;
             chop2=1;
             }
   }
```

While In Doste, the value changing of the observable is not on "triggered procedure" but on the dynamic binding among observables, which is mainly expressed in the form of

"a is {if (true) b else c}"

As can be seen from below, in the Doste_Version, modelling the chop1 is totally different: it depends on other observables. When there are a large number of observables, this modelling process can be quite complicated. Additionally, chop2 can't change value together with chop1 just like in the above Eden_Version, it has to be modelled again.

```
chop_1 is {
    if(.Machine 1 not
       and (.isMachineOn_1 not)){
     if(.w_1
        and (.watcher_1 not)) false
     else true)
    else true
}

chop1 is {if(.chop_1)1 else 0}
```

3.3 Linking Doste and Eden

Doste is quite a good prototype language, but in some situations, using functions can be a better choice. The lack of function restricts its strength under some cases. However, now Doste can link with Eden, which can offer good functions and flexible GUI by using Scout and Donald. It also gives an alternative for the Doste visualization.

A linking dependency can be made to link them together, in this project, as can be seen from below:

```
@philosopher clickTry_A is
{if(@root eden base actionTry_A > 0)
    true
 else false};
```

The actionTry_A is a observable in Eden, and there is a dependency between it and the clickTry_A in Doste, in this project, actionTry_A represents the mouse action, and through the above dependency, the input from the Eden GUI can be parsed to Doste Unit, which will compute the logic in this project.

```
@root eden base chop1 is
   {@philosopher chop1};
```

After computing the logic, the result will be parsed back to Eden GUI through such dependency as above, so the result can be shown.

In this way, the Doste Unit can have all the inputs from Eden GUI, and after computing, it will output the states of all the observables to Eden GUI. See Figure 3.

3.4 Problems in tkeden 2.10 in supporting Doste

There will be some failed loading messages if open the .dasm file directly when the lines of code are too long. To solve this problem, thanks to Nick Pope, we have to enter a command line to load the dasm file:

```
%dasm
@root notations dasm run=
 (new type=LocalFile filename="  ");
```

3.5 Modelling using Eden

Modelling this problem in Eden is quite easy to implement, after getting the dependency between the observables and the scenarios from Doste_Version, (which implies that Doste can be used as a good prototype language from this point of view), using procedures and triggers instead of "is" in Doste to express the dependency in this model.

Compared with Doste, the procedures, to some extent, can be helpful, however, as “is” in Eden is not as powerful as in Doste, there going to be a large amount of procedures and triggers to support the dependency in this case. Details can be found from the source code.

4. about the model

Generally, the model turns out well, and illustrates the dependency clearly by using Doste and Eden respectively. Additionally, this model could help people to understand the concept of “critical section” and “mutual exclusion”, which typically exists in the concurrent system by using the classical philosopher dining problem.

4.1 Implementing in JAVA

Since there is no dependency syntax in JAVA language, it can model this philosopher dining problem using synchronization of JAVA threads concept, which typically solve the concurrent system mutual exclusion problem.
Each philosopher can be modelled as a thread, and JAVA can monitor the critical section by using the keyword “synchronize”, so that only one thread can enter the critical section one time (the eating philosopher will lock the two chopsticks in this case), after that thread finishes running (philosopher stops eating), the lock will be released (chopsticks will be put back), and other threads will be notified to enter (other philosophers can be ready to get the chopsticks to eat).

4.2 Compared with JAVA

Compared with the synchronization of JAVA threads concept, EM language is more straightforward in expressing and modelling. Additionally, it is easy to model through experiment and observation, as the EM not just allow the user free to intervene the model on the fly, but also offers strong mechanisms in querying the observables. E.g. in Doste the“ %list” command can check all the observables; while in Eden “?observable;” can check the observable and dependency.

In JAVA , the type checking is quite strict, although this may lose the flexibility to some extent, it is precise in syntax. However in Doste and Eden, the lack of type checking sometime can give rise to some problems.

5 About EM

The dependency and agent-orient feature of EM is useful in some cases; however, it still has some limitations. For instance, the differences of the languages make it less convenient in linking together, e.g. keyword “is” has the different meanings in Doste and Eden; there is no Boolean types in Eden so sometimes translating is a must before linking to Doste which has the Boolean type…..apart from the language limitation, when it comes to large and critical system, using experiment and observation way may be not precise enough. At this time, Formal Methods can be a good alternative state-based modelling language for its mathematics-based nature.
However, all these above aren’t to say EM is not good, although it requires further developments, EM has its own advantages:
EM is good if using as a prototype language as it can easily capture the requirements, and produce corresponding models;
EM is easy in building model by experiment and observation, so it is more suitable in education whenever requires experiment and observation.

6. Conclusion

This paper reviews how to build an EM model in Philosopher Dining Problem using Doste and using Eden respectively, as well as argues the features of each languages and shows the way to link them together. Finally, this paper compares the EM languages with JAVA language in term of implementing this specific model, and some advantages and limitations of EM have also been given.

Acknowledgements

This paper would not have been possible without the time and effort of Meurig Beynon in explaining the Concept of EM and linking Doste and Eden.
Also I would like to thank Nick Pope for his work on Doste and Lin-Feng Lee for her bright ideas.

References

[1]: http://en.wikipedia.org/wiki/Dining_philosophers_problem

[2] Lecture notes http://www2.warwick.ac.uk/fac/sci/dcs/research/em/teaching/cs405/defnmodesdosteeden.pdf