File komega-v08a.py

Author: G.Doeben-Henisch
First date: September 4, 2020
Last change: 16.September 2020

##################################
Execution Environment of my local machine:
(venv) gerd@gerd-ub2:~/env/komega/tst$ python3 komega-v01d.py
#

####################################
GITHUB
#
We use a github repository at:
https://github.com/szmt/komega.git
#
Im working from a unix-shell using the following github-commands:
https://git-scm.com/docs/git

####################################
BACKGROUND THEORY
#
This code is a translation of a theory described in the blog
https://www.uffmm.org
#
Last document for the specification of this code:
#
'''
https://www.uffmm.org/wp-content/uploads/2020/09/requirements-no4-v5-13Sept2020.pdf'''
##
HMI - CLASS PUBLISH
#
'''
The intended interaction of the user with the system will be realized through an interactive web page. In this experimental program there is no web page but a normal console. Therefore (proposal from Tobias Schmitt) we have a special class 'Publish' which handles all console input and output and the other classes interact with this class Publish. In the context of the web server we can then replace the class Publish by appropriate libraries for HTML web pages.

The same holds for the class 'Storage' which is a placeholder for more advanced data base operations in the future. Now the python class 'shelve' is used.
'''
####################################
ACTOR STORY
#
In the specifications an actor story [AS] has been specified. This AS requires # some basic states which are dedicated for certain tasks to do:

'''
ACTOR STORY

```
S1: START
S2: EDIT P(roblem description)
S3: EDIT S (actual state)
S4: EDIT X (change rules)
S5: SIMULATION (Applying X to S)
S6: EVALUATION (After the simulation)
S7: STOP
'''

# MAIN IDEA
'''
According to the above mentioned actor story the user will be sitting in front of a system interface
[SI] which works first only as a console.
In the beginning the user is placed in a start state S1 showing all options available.
The user can select one of these options and can from start state S1 reach all other states S2-S7.
'''

############################
# IMPORTS

############################
# SUPPORTING FUNCTION
#

# No funtions yet

# CLASSES
#
'''
For every state there exists one working class to do the job.
The special class 'Publish' in this code exists only because the interaction of the user with the
system will happen with an interactive website which uses HTML and javascript. Here in this
experimental environment a simple unix-console is used.
'''

import kcv8a as kc      #The theory-related classes

#########################
# Main Programm
#


###############
# Start main loop
#
# The loop will work as long as the value of the variable 'loop' is different to 'N'

loop='Y'
while loop=='Y':

##############################
# STATE 1 : START
```

```
# Show available options
# Get feedback for selection
# Confirm the selection
# Distribute to different states

        kc.ast.menushow()

# Ask back for selection number
        message='Enter a Number [1-7] for Menu Option \n'
        kc.pub.userinput(message)

# Evaluate the selection

        opt=kc.pub.opt
        kc.ast.badoption(opt)

#########################################################################
# Call to a class instance
#
#########################################################################
# Call to state Edit Problem P
################################
# STATE 2 : EDIT P
# Ask Questions related to P
# Collect all answers into one problem document
#

        if opt=='2':
                # Where You are
                kc.pub.show(kc.ap)

                #Interaction with Problem Class
                message='Enter your problem as it is now given in plain text\n'
                kc.pub.userinput(message)
                inp=kc.pub.opt
                kc.app.getproblem(inp)

                message='Enter your vision of a better state in the future in plain text\n'
                kc.pub.userinput(message)
                inp=kc.pub.opt
                kc.app.getvision(inp)

                message='Enter the name of the city you are in\n'
                kc.pub.userinput(message)
                inp=kc.pub.opt
                kc.app.getregion(inp)

                message='Time model [From, Until,Cycleunit [Y or M or D or H]]: '
                kc.pub.userinput(message)
                inp=kc.pub.opt
                kc.app.gettime(inp)
```

```
                message='Which kinds of persons are important? Write a list, comma separated
please : '
                kc.pub.userinput(message)
                inp=kc.pub.opt
                kc.app.getperson(inp)

                kc.app.problemTotal()

                ########################################
                # Put the information from problem into the document 'docp'

                message='Your Problem document is now :\n'
                kc.pub.useroutput(message)
                docp=kc.app.problemAll
                kc.pub.useroutput(docp)

                ##############################################
                # STORE DOCUMENT PERMANENTLY

                kc.st.storedata('P-',docp)



#
#########################################################################
# Call to state Edit Actual State S
################################
# STATE 3 : EDIT S
# Collect single expressions
# Collect all expressions into one  document describing S
# The document S is organized as a set of expressions!
#

        elif opt=='3':
                # Where You are
                kc.pub.show(kc.ass)

                # Set document S to zero
                kc.aas.emptydocs()

                # Ask for a document S to be loaded
                message="Do You want to load a document S? [Y,N]\n"
                kc.pub.userinput(message)
                inp=kc.pub.opt
                if inp == 'Y':
                        kc.st.loaddata('S')
                        message='Your State Description document is as follows :\n'
                        kc.pub.useroutput(message)
                        docs=kc.aas.stateAll
                        kc.pub.useroutput(docs)
```

```
                Sloop='Y'
                while Sloop=='Y':
                        # Interaction with actual state S class
                   message='Enter an expression for your state description  in plain text : \n'
                   kc.pub.userinput(message)
                   inp=kc.pub.opt
                   kc.aas.getexpression(inp)
                   message="STOP Editing S  != Y', CONTINUE = 'Y'  \n"
                   kc.pub.userinput(message)
                   inp=kc.pub.opt
                   Sloop=inp

                ########################################################
                # Keeping the document

                message='Your State Description document is now :\n'
                kc.pub.useroutput(message)
                docs=kc.aas.stateAll
                kc.pub.useroutput(docs)

                ##############################################
                # STORE DOCUMENT PERMANENTLY

                kc.st.storedata('S-',docs)

#
#########################################################################
# Call to state Edit Change Rules X
################################
# STATE 4 : EDIT X
# Collect single expressions for change rules
# Collect all expressions into one  document describing X
#
################################################
# FORMAT OF A RULE WITHOUT ACTOR
#
# IF: CONDITION THEN: PROBABILITY - E-MINUS - E-PLUS
#################################################################
#V0:.1 Expression......[0,1].....1 Expression .1 Expression
#
# In the first version we assume the most simple case which is possible!

        elif opt=='4':
                # Where You are
                kc.pub.show(kc.ax)

                # Set document X to zero
                kc.axx.emptydocx()

                # Ask for a document X to be loaded
                message="Do You want to load a document X? [Y,N]\n"
                kc.pub.userinput(message)
```

```
        inp=kc.pub.opt
        if inp == 'Y':
                kc.st.loaddata('X')
                message='Your Rules  document is as follows :\n'
                kc.pub.useroutput(message)
                docx=kc.axx.rulesAll
                kc.pub.useroutput(docx)

        Xloop='Y'
        while Xloop=='Y':

                # Interaction with actual X class
                message="Enter a rule for your xchange rules in plain text with the parts
CONDITION, PROBABILITY, EFFEKT-, EFFEKT+ \n"
                kc.pub.useroutput(message)
                message="We will ask You for each category separatedly :\n"
                kc.pub.useroutput(message)

                message=kc.axx.rcat[0]+' : ' #Shows CONDITION
                kc.pub.userinput(message)
                kc.axx.getrule()

                message=kc.axx.rcat[1]+' : ' #Shows PROBABILITY
                kc.pub.userinput(message)
                kc.axx.getrule()

                message=kc.axx.rcat[2]+' : ' #Shows E-MINUS
                kc.pub.userinput(message)
                kc.axx.getrule()

                message=kc.axx.rcat[3]+' : ' #Shows E-PLUS
                kc.pub.userinput(message)
                kc.axx.getrule()

                kc.axx.rulesAll.append(kc.axx.rule)
                kc.axx.rule=[]
                kc.axx.rulessummary()

                message="CONTINUE  Editing X = 'Y', STOP != 'Y'  \n"
                kc.pub.userinput(message)
                inp=kc.pub.opt
                Xloop=inp

        ########################################################
        # Keeping the document

        message='Your Rules  document is now :\n'
        kc.pub.useroutput(message)
        docx=kc.axx.rulesAll
        kc.pub.useroutput(docx)

        kc.st.storedata('X-',docx)
```

```
#
#########################################################################
# Call to state  Simulation SIM
################################
# STATE 5 : Run the simulation
# PREPARATION
# (1) Take a state description S and an appropriate set of  change rules X
# (either actaully edited or from a file)
# SIMULATION CYCLE
# (2) Select all rules X* whose conditions are fulfilled by S.
# (3) For each rule x in X*:
# (3.1) Apply the E-Minus part and remove the E-Minus expression from S
# (3.2) Apply the E-Plus part and add th e-Plus expressions to S
# (3.3) Show the new version of S after applying X* to S
# (3.4) If no Stop then repeat from (2)

        elif opt=='5':
                kc.pub.show(kc.asim)

# PREPARATION
# (1) Take a state description S and an appropriate set of  change rules X
# from a file
# Ask for a document S to be loaded

                message="Do You want to load a document S? [Y,N]\n"
                kc.pub.userinput(message)
                inp=kc.pub.opt
                if inp == 'Y':
                        kc.st.loaddata('S')
                        message='Your State Description document is as follows :\n'
                        kc.pub.useroutput(message)
                        kc.assim.s=kc.aas.stateAll
                        kc.pub.useroutput(kc.assim.s)

# Ask for a document X to be loaded
                message="Do You want to load a document X? [Y,N]\n"
                kc.pub.userinput(message)
                inp=kc.pub.opt
                if inp == 'Y':
                        kc.st.loaddata('X')
                        message='Your Rules  document is as follows :\n'
                        kc.pub.useroutput(message)
                        kc.assim.x=kc.axx.rulesAll
                        kc.pub.useroutput(kc.assim.x)

# SIMULATION CYCLE

                s=kc.assim.s
                x=kc.assim.x
                kc.assim.xapply(s,x)
```

```
#
#####################################################################
# Call to state  Evaluation EV
#

        elif opt=='6':
                kc.pub.show(kc.aev)

#
#####################################################################
# Call to state Stop STP
#

        elif opt=='7':
                kc.pub.show(kc.astp)

#####################################################################
# End of Loop



# Clarify how to continue

        message="STOP MAIN LOOP != 'Y', CONTINUE = 'Y'  \n"
        kc.pub.userinput(message)
        inp=kc.pub.opt
        loop=inp




'''
TEST SIMULATION

(venv) gerd@gerd-ub2:~/env/komega/tst$ python3 komega-v08a.py
1 is START
2 is EDIT P
3 is EDIT S
4 is EDIT X
5 is SIMULATION
6 is EVALUATION
7 is STOP
Enter a Number [1-7] for Menu Option
5
!!You have selected the state :
SIMULATION
Here You can run a simulation SIM to check what happens with your initial state S when the change rules X will be applied repeatadly on the state S.
Do You want to load a document S? [Y,N]
Y
Here is the list of all stored documents so far :
```

['X-Maryv1-16sept2020', 'S-Maryv1-16sept2020']
Enter a name for the document you want to load :
S-Maryv1-16sept2020
Your State Description document is as follows :

{'Mary needs a book'}
Do You want to load a document X? [Y,N]
Y
Here is the list of all stored documents so far :

['X-Maryv1-16sept2020', 'S-Maryv1-16sept2020']
Enter a name for the document you want to load :
X-Maryv1-16sept2020
Your Rules document is as follows :

[[{'Mary needs a book'}, {'1.0'}, {''}, {'Mary wents to the library'}], [{'Mary wents to the library'}, {'1.0'}, {'Mary wents to the library'}, {'Mary enters the library'}], [{'Mary enters the library'}, {'1.0'}, {'Mary enters the library'}, {'Mary is in the library'}]]
Set S given :
{'Mary needs a book'}
Actual rule :
[{'Mary needs a book'}, {'1.0'}, {''}, {'Mary wents to the library'}]
Set S after Remove :
{'Mary needs a book'}
Set S after Union :
{'Mary wents to the library', 'Mary needs a book'}
Set S given :
{'Mary wents to the library', 'Mary needs a book'}
Actual rule :
[{'Mary wents to the library'}, {'1.0'}, {'Mary wents to the library'}, {'Mary enters the library'}]
Set S after Remove :
{'Mary needs a book'}
Set S after Union :
{'Mary enters the library', 'Mary needs a book'}
Set S given :
{'Mary enters the library', 'Mary needs a book'}
Actual rule :
[{'Mary enters the library'}, {'1.0'}, {'Mary enters the library'}, {'Mary is in the library'}]
Set S after Remove :
{'Mary needs a book'}
Set S after Union :
{'Mary is in the library', 'Mary needs a book'}
STOP MAIN LOOP != 'Y', CONTINUE = 'Y'
N

'''

```
# File kcv8a.py

# Author: G.Doeben-Henisch
# First date: September 6, 2020
# Last date: September  16, 2020

################################
# CLASS DEFINITIONS

class Start:
        def __init__(self):
                self.menulist = ['START','EDIT P','EDIT S', 'EDIT
X','SIMULATION','EVALUATION','STOP']

        def menushow(self):
                i=0  # Counter for menu-loop
                for state in self.menulist:
                        i=i+1
                        message="str(i)+' is '+state"
                        pub.useroutput(eval(message))

        def badoption(self,opt):
                if int(opt)<1 or int(opt)>7:
                        message='!!You have selected a bad option'
                        pub.useroutput(message)

                if int(opt)>0 and int(opt)<8:
                        message='!!You have selected the state :\n'+self.menulist[int(opt)-1]
                        pub.useroutput(message)

#################################################################

class Actor:
        def __init__(self,inp):
                self.message=inp

#################################################################

class Publish():

        def show(self,other):
                print(other.message)

        def useroutput(self,message):
                print(message)

        def userinput(self,message):
                self.opt=input(message)
```

```
#####################################################################
# CLASS PROBLEM
'''
MAIN IDEA

A main window W1 with a menu showing all possible questions to be
answered.

(a) Describe the problem P: What is given and what is the intended future state?
(b) Describe the intended real part of the world (space).
(c) Describe the time model T : which time period, which cycles.
(d) Which kinds of actors are seen as being important for the problem and its  future?
(e) Some other assumptions.
'''


class Problem(Actor):

        def getproblem(self,inp):
                self.problemNow = inp
                message='Feedback Problem Now :\n'+self.problemNow
                pub.useroutput(message)

        def getvision(self,inp):
                self.problemFuture = inp
                message='Feedback Problem Future :\n'+self.problemFuture
                pub.useroutput(message)

        def getregion(self,inp):
                self.problemRegion = inp
                message='Feedback Problem Region :\n'+self.problemRegion
                pub.useroutput(message)

        def gettime(self,inp):
                self.problemTime = inp
                self.problemTM = self.problemTime.split(',')
                message='Feedback Problem TimeModel :\n'+str(self.problemTM)
                pub.useroutput(message)

        def getperson(self,inp):
                self.problemPerson = inp
                self.problemPRS = self.problemPerson.split(',')
                message='Feedback Problem Persons :\n'+str(self.problemPRS)
                pub.useroutput(message)

        def problemTotal(self):
                self.problemAll =[]
                self.problemAll.append(self.problemNow)
                self.problemAll.append(self.problemFuture)
                self.problemAll.append(self.problemRegion)
                self.problemAll.append(self.problemTime)
                self.problemAll.append(self.problemPerson)
```

```
        def problemnow(self):
                message="The actual problem document is the following one :\
n"+str(self.problemAll)
                pub.useroutput(message)

#####################################################################
# CLASS S(tate Description)
'''
IDEA:
This state should allow in the final version the editing of the texts S and X in parallel. Additionally
one should be able to call from within this state(s) the simulation mode to test whether the actual
texts are working.

FOR NOW:
In this first experimental version one has to work either with the stae S or with the state X
separatedly. Simulation would be a follow up state.

TASK:
Input all data which are necessary for the S-state (including sectioning
and extended texts with details)

ACTORS:
Human experts.

SYSTEM INTERFACE:
A main window W1 offering the editing of a text consisting of individual
statements. Every statement can be edited separately and repeatedly.

ACTIONS:
Select either a given statement for editing or edit a new statement or
stop.

IMPLEMENTATION:
Using the set class of python to collect expressions. Set operations are very convenient e.g. to unify
different sets to one set, and more.

'''
class AState(Actor):

        def __init__(self):
                self.stateAll = set()

        def emptydocs(self):
                self.stateAll = set()

        # Load a document S from file
        def newdocs(self,filen):
                self.stateAll=filen


        def getexpression(self,inp):
```

```
                self.expression = inp
                self.stateAll.add(inp)
                message='Feedback Your last expression :\n'+str(self.expression)
                pub.useroutput(message)
                message='Feedback Your document S so far :\n'+str(self.stateAll)
                pub.useroutput(message)

################################################################################
# CLASS X (Change Rules)
'''
IDEA:
The change rules X are described in the requirements paper cited in the beginning of the main
program text. The principal idea of the change rules X is to allow changes to an actual state S if
certain conditions are fulfilled (satisfied). These changes will be executed during the state called
simulation.

FOR NOW:
Because a complete implementation of the theoretically possible change rules is nearly an infinite
task this version of the change rules X called X01 is limited to the simplest possible case. This
contains the following simple structure:

IF Condition C THEN with Probability Pr realize the Effect E- and E+.

While this is already the case without any actor all the parts (C,E+,E+) are additionally limited to
one expression each. Thus we start with the format:

IF Condition C(1) THEN with Probability Pr [0,1] realize the Effect E-(1) and E+(1).

The strategy is to extend all these limits stepwise in the next versions.

TASK:
Input all rules   for the X-state

ACTORS:
Human experts.

SYSTEM INTERFACE:
A main window W1 offering the editing of a text consisting of individual
rules. Every statement can be edited separately and repeatedly.

ACTIONS:
Select either a given statement for editing or edit a new statement or
stop.

IMPLEMENTATION:
Using the list-construct of python to collect expressions, because lists are ordered and mutable and
allow many intersting operations.
'''

class Xrules(Actor):

        def __init__(self):
```

```
                self.rulesAll = []
                self.rule=[]
                self.rcat=['CONDITION', 'PROBABILITY','EFFECT-', 'EFFECT+']
                self.cond=set()          #Needs a set

        def greeting(self):
            message="Your rule set at start :"+str(self.rulesAll)
            pub.useroutput(message)
            message="Your single rule buffer at start :"+str(self.rule)
            pub.useroutput(message)

        def emptydocx(self):
            self.rulesAll = []

        def getrule(self):
                self.inp="
                self.inp = pub.opt
                self.cond=set()
                self.cond.add(self.inp)
                self.rule.append(self.cond) #Appends a set
                message="Your single rule buffer : "+str(self.rule)
                pub.useroutput(message)

        def rulessummary(self):
                message='Feedback Your rules  so far :\n'+str(self.rulesAll)
                pub.useroutput(message)

################################################################################
# CLASS SIMULATOR
############################
# FIRST SIMPLE SIMULATOR
'''
As a first simple simulator cycle the following schema has been proposed:
1. If there already exists a state file S and a rule file X load such files, otherwise edit two
new files.
2. The simulator works in cycles.
3. Every cycle CYC the actual version of a state description S as well a rule set X will be
loaded into the simulator.
4. If the set of applicable rules is not empty ¬∅ then applying satisfying change rules X∗ to the
actual state Snow will change the state according to the schema: Si+1=Snow−E− ∪E+.
5. After the simulation has finished the set S can have been changed. S can be stored permanently.
'''
'''
Using set operations:
https://docs.python.org/3.1/library/stdtypes.html#set
'''
################################################
# FORMAT OF A RULE WITHOUT ACTOR
#
# IF: CONDITION   THEN: PROBABILITY   - E-MINUS -     E-PLUS
#############################################################
#V0:. Expressions.....[0,1].....Expressions...Expressions
```

```
#

class Simulation():
	def __init__(self):
		self.s={}

	def xapply(self,s,x):
		for i in range(len(x)):
			message='Set S given : \n'+str(s)
			pub.useroutput(message)
			r=x[i]
			message="Actual rule :\n"+str(r)
			pub.useroutput(message)
			if r[0].issubset(s):
				delset=r[2]
				for j in range(len(delset)):
					d=delset.pop()
					if d in s:
						s.remove(d)
					else:
						pass
					message='Set S after Remove : \n'+str(s)
					pub.useroutput(message)
					s=s.union(r[3])
					message='Set S after Union : \n'+str(s)
					pub.useroutput(message)


##########################################################################
# CLASS DATA STORAGE
#
'''
All the files which will be created during a session can at the end of the session be stored at will in a shelve. These can also be loaded in the beginning to continue editing. Every data set is marked by an individual free index.
'''
import shelve  #To store and reload data
	#See: https://docs.python.org/3.1/library/shelve.html

class Storage():
	def __init__(self,storageID):
		self.storename=storageID

	def openstorage(self):
		self.d=shelve.open(self.storename,writeback=True)

	def closestorage(self):
		self.d.close()

	def showkeys(self):
		keylist=list(self.d.keys())
		pub.useroutput(keylist)
```

```
        def storedata(self,doctype,docc):
                message="Do You want to store your document? [Y,N] \n"
                pub.userinput(message)
                inp=pub.opt
                if inp == 'Y':
                        self.openstorage()
                        message="Here is the list of all stored documents so far :\n"
                        pub.useroutput(message)
                        klist=list(self.d.keys())
                        pub.useroutput(klist)
                        message="Enter a name for the document to be stored. We prefix it
automatically with 'P/S/X-'"
                        pub.userinput(message)
                        fname=doctype+pub.opt
                        self.d[fname]=docc
                        self.closestorage()

        def loaddata(self,doctype):
                        self.openstorage()
                        message="Here is the list of all stored documents so far :\n"
                        pub.useroutput(message)
                        klist=list(self.d.keys())
                        pub.useroutput(klist)
                        message="Enter a name for the document you want to load : \n"
                        pub.userinput(message)
                        fname=pub.opt
                        docL=self.d[fname]
                        self.closestorage()
                        if doctype == 'P':
                                app.problemAll=docL
                        elif doctype == 'S':
                                aas.stateAll=docL
                        elif doctype == 'X':
                                axx.rulesAll=docL
                        else: pass


######################################################################
# CLASS INSTANCES

ast=Start()

ap=Actor("Here you can describe your problem with regard to different questions.")
app=Problem("Here you can describe your problem with regard to different questions.")

ass=Actor("Here You can describe an actual state S related to your problem.")
aas=AState()
ax=Actor("Here You can edit some change rules X to apply to an actual state S.")
axx=Xrules()
asim=Actor("Here You can run a simulation SIM to check what happens with your initial state S
when the change rules X will be applied repeatadly on the state S.")
```

```
assim=Simulation()

aev=Actor("Here some advice will be given how to organize an evaluation EVAL of a realized simulation SIM.")
astp=Actor("This will stop the whole program.")

pub=Publish()
st=Storage('GDH2')
```