UDC: 351.817:004.043 Info M str. 4-13

PYTHON BIBLIOTEKE ZA RAD SA PRAVILIMA
PYTHON LIBRARIES FOR WORKING WITH RULES

Dragica Ljubisavljević

REZIME: U ovom radu se daje kratak prikaz oblasti sistema zasnovanih na pravilima kao podoblasti veštačke inteligencije, sa posebnim naglaskom na alate potrebne za njihov razvoj. Glavni cilj je pružiti analizu i pregled popularnih Python biblioteka koje se koriste za razvoj ovakvih sistema. Rad počinje teorijskim osvrtom na osnovne koncepte koji se koriste prilikom razvoja sistema zasnovanih na pravilima. Posebna pažnja je posvećena analizi tri Python biblioteke: Pyke, Experta i Durable rules. Praktični deo rada ilustruje se kroz demonstracioni primer - razvoj sistema za unapređenje operativnih procesa u baletskom studiju. Ovaj primer demonstrira kako se pravila mogu implementirati korišćenjem svake od navedenih biblioteka. Na kraju, izvršena je uporedna analiza ovih biblioteka sa istaknutim prednostima i manama svake od njih.

KLJUČNE REČI: Sistemi zasnovani na pravilima, ekspertni sistemi, Python, Pyke, Experta, Durable rules.

ABSTRACT: This paper presents a short review of rule-based systems (an artificial intelligence subfield), with a special emphasis on the tools necessary for their development. The main goal is to provide an analysis and overview of popular Python libraries used for developing such systems. The paper begins with a theoretical overview of the basic concepts used in the development of rule-based systems. Special attention is given to the analysis of three Python libraries: Pyke, Experta, and Durable rules. The practical part of the paper is illustrated through a demonstration example - the development of a system to improve operational processes in a ballet studio. This example demonstrates how rules can be implemented using each of these libraries. Finally, a comparative analysis of these libraries is conducted with each one's pros and cons highlighted.

KEY WORDS: Rule-based systems, expert systems, Python, Pyke, Experta, Durable rules.

1. UVOD

Istraživanje u oblasti ekspertnih sistema je jedno od najduže trajućih i najuspešnijih područja stalnih istraživanja unutar polja veštačke inteligencije. Od 1980-ih, objavljeno je mnogo studija slučaja primene ekspertnih sistema koje pokrivaju širok spektar funkcionalnih oblasti i domena primene [15]. Tokom ovog perioda razvijen je veliki broj tehnologija i alata za razvoj ekspertnih sistema. Konstantno usavršavanje i inovacije u ovim alatima dovele su do povećanja njihove kompleksnosti a posredno i proširivanje domena primene ekspertnih sistema.

Programski jezik Python [12], istaknut zbog svoje intuitivne sintakse, efikasnosti i širokog spektra primene, postao je nezaobilazan alat u akademskim i inženjerskim disciplinama. Prema najnovijim podacima sa TIOBE indeksa [16], Python zauzima prvo mesto na listi najpopularnijih programskih jezika. Zahvaljujući velikom skupu svojih biblioteka, Python omogućava razvoj širokog spektra aplikacija.

Ovaj rad ima za cilj da pruži detaljan pregled i analizu popularnih Python biblioteka specijalizovanih za razvoj sistema zasnovanih na pravilima. Analizirane i upoređene su tri istaknute biblioteke: Pyke [11], Experta [6] i Durable rules [5].

Rad je organizovan na sledeći način. Na samom početku, dato je teorijsko razmatranje ekspertnih sistema. U okviru pregleda oblasti objašnjene su neke od ključnih komponenti svakog ekspertnog sistema poput baze znanja, radne memorije i mehanizma za zaključivanje. U trećem poglavlju opisano je zašto su baš biblioteke Pyke, Experta i Durable rules odabrane da budu opisane u ovom radu. Takođe, dat je osvrt na radove u kojima su ove biblioteke korišćene kao alati za razvoj sistema. Četvrto poglavlje predstavlja demonstracioni primer kako bi se ilustrovala praktična primena ovih biblioteka. Na primeru razvoja sistema za obračun članarina baletskih igrača u baletskom studiju prikazano je kako svaka od ovih biblioteka može pomoći prilikom implementacije pravila. U petom poglavlju data je uporedna analiza ističući njihove jedinstvene karakteristike i razlike u pristupu. Šesto poglavlje je zaključak.

2. PREGLED OBLASTI

Ekspertni sistemi (ES) su razvijeni od strane zajednice veštačke inteligencije sredinom 1960-ih godina. Ovi sistemi zasnivaju se na prenošenju znanja iz neke određene oblasti sa čoveka na računar. Znanje omogućava računarskim programima da rešavaju probleme koji zahtevaju ljudsku stručnost. ES rešavaju probleme na sličan način na koji to čini čovek. Posebno se ističu ***ES zasnovani na pravilima***, koji sadrže znanja u obliku pravila. Oni omogućavaju računaru da izvodi zaključke i donosi odluke bazirane na ovim pravilima [8].

Tokom vremena, ES su doživeli značajan razvoj, usled čega su se njihove primene i tehnološke karakteristike širile. Ovo je dovelo do toga da ES danas možemo pronaći pod različitim nazivima poput: 'Sistemi poslovnih pravila' [2] i 'Sistemi za podršku odlučivanju' [4].

Svaki ES sastoji se od ključnih komponenti i neke od njih opisane su u nastavku.

Baza znanja/Predstavljanje znanja(*knowledge representation*) - U sistemima zasnovanim na pravilima, znanje je predstavljeno u obliku 'IF-THEN' pravila, što predstavlja jednu od ključnih tehnika za predstavljanje znanja u okviru ES. Prvi deo pravila IF (ako) sastoji se od jednog ili više iskaza (premisa) na osnovu kojih se izvode zaključci. Ukoliko pravilo sadrži više premisa one su povezane logičkim operatorima AND, OR i NOT. Drugi deo pravila THEN (tada) predstavlja posledice pravila, odnosno ono što će se desiti ako su ispunjene premise navedene u IF delu. Iskazi navedeni u okviru ovog

drugog dela pravila nazivaju se zaključci. Dakle, svako pravilo definiše skup radnji koje treba izvršiti kada se ispune određeni uslovi. Ovako definisana pravila usmeravaju proces donošenja zaključaka i prikazuju kako eksperti koriste svoje znanje i logiku prilikom procesa zaključivanja. [13]

Radna memorija *(working memory*) - Ova komponenta u ekspertnim sistemima služi kao skup svih trenutno relevantnih činjenica i podataka o određenom domenu problema. Ona predstavlja je mesto gde se čuvaju podaci koji se koriste u pravilima.

Mehanizam za zaključivanje (*inference engine*) - Mehanizam za zaključivanje pokušava da izvede nove informacije o datom problemu koristeći pravila iz baze znanja i specifične činjenice koje se nalazi u radnoj memoriji. On kombinuje pravila i činjenice i ovo omogućava sistemima zasnovanim na pravilima da stvaraju nove zaključke koji su potrebni za rešavanje postavljenog problema. Izvođenje zaključaka i rezonovanje vrši se na način na koji bi to eksperti uradili.

Mehanizmi za zaključivanje u sistemima zasnovanim na pravilima mogu koristiti različite strategije za izvođenje cilja (odnosno, novih činjenica). Najčešće strategije su: *ulančavanje unapred* (forward chaining) i *ulančavanje unazad* (backward chaining). Ulančavanje unapred podrazumeva proces gde sistem počinje sa poznatim činjenicama i primenjuje pravila kako bi se došlo do novih činjenica ili zaključaka. Ulančavanje unazad je pristup gde sistem počinje sa željenim zaključkom ili ciljem i radi unazad tražeći pravila i činjenice koje vode ka tom zaključku. Sistemi mogu koristiti ili jednu od ovih strategija ili kombinaciju obe. [13]

"Baza pravila i radna memorija predstavljaju ključne strukture podataka unutar sistema, dok se mehanizam za zaključivanje koristi kao osnovni programski alat za njihovu obradu i primenu." [13]

Pored ovih, jedna od ključnih komponenti ekspertnog sistema je **mehanizam za objašnjavanje (*explanation facility*)** koji omogućava korisniku da pita kako je sistem došao do određenog zaključka. On objašnjava korisniku razloge zaključivanja sistema. Postoji nekoliko često korišćenih tehnika za formiranje objašnjenja. Jedna od njih je tehnika učaurenog teksta (canned text) kojom se objašnjenja za korisnike definišu kao unapred pripremljeni tekstovi. Ovi unapred pripremljeni tekstovi mogu u sebi da sadrže i neke dinamičke delove. Druga često korišćena tehnika je praćenje toka izvršavanja pravila i po potrebi njihovo prevođenje u prirodni jezik, odnosno tekst. [13]

3. POPULARNE PYTHON BIBLIOTEKE ZA RAZVOJ SISTEMA ZASNOVANIH NA PRAVILIMA

Na početku ovog istraživanja bilo je neophodno identifikovati Python biblioteke koje se koriste za razvoj sistema zasnovanih na pravilima. Izvršena je pretraga na sajtovima PyPI i GitHub. Ključne reči koje su korišćene prilikom pretrage uključivale su kombinacije termina poput "Python rule engine" i "Python library for rule based system". Zbog velikog broja biblioteka koji se konstanto razvijaju pretraga je proširena na internet forume. Pronađeno je dosta članaka koji su u skladu sa temom istraživanja [17] [24] [25] [26]. Nakon toga, izvršena je pretraga na Google Schoolar-u kako bi se stekao uvid u to koje biblioteke autori primenjuju u svojim naučno-istraživačkim projektima.

Python biblioteke koje su na ovaj način bile identifikovane i razmatrane su: Intellect, PyRules, Rule engine, Business rules, PyCLIPS, PyKnow, Experta, Pyke i Durable rules. Biblioteke ***Intellect*** [30] i ***PyRules*** [23] nisu uzete u dalje razmatranje zbog njihove zastarelosti i ograničenja na starije Python verzije, kao i zbog nedostatka naučnih referenci o njihovoj primeni. Biblioteke ***Rule engine*** [21] i ***Business rules*** [29], iako novije, takođe su izostavljene zbog nedostatka naučnih referenci koje bi ukazale na popularnost ovih biblioteka. Biblioteka ***PyCLIPS*** [20] je isključena iz daljeg razmatranja jer predstavlja interfejs za CLIPS [18], tj. nije standardna Python biblioteka.

Biblioteka ***Pyke*** [11] pokazala se kao veoma popularna u raznim istraživačkim projektima. Pretragom na Google Scholar-u pronađeno je dosta naučnih referenci. To je bio razlog zašto je ova biblioteka odabrana da se obradi u ovom istraživanju. Nedostatak ove biblioteke je vezan za njenu nekompatiblinost sa novijim verzijama Pythona. Biblioteka ***PyKnow*** [19] takođe se pokazala kao veoma popularna i korišćena. Kada je ova biblioteka prestala da bude aktivno održavana razvijena je biblioteka ***Experta*** [9], kao njen naslednik. Iz ovog razloga, biblioteka PyKnow neće biti deo ovog istraživanja, već biblioteka Experta. Biblioteka ***Durable rules*** [5] je novija, kompatibilna sa novijim verzijama Pythona, korišćena u raznim projektima te je ona odabrana kao treća biblioteka u ovom radu. U produžetku radaje detaljan prikaz odabranih biblioteka.

Python Knowledge Engine (Pyke) je open-source biblioteka koja koristi logičko programiranje inspirisano Prologom. U istraživanjima fokusiranim na hibridne sisteme kontrole zgrada [9] i pristupima zasnovanim na znanju u mrežnim tehnologijama [7], biblioteka Pyke je bila ključna u razvoju ovih sistema. *Experta* je open-source biblioteka koja je inspirisana CLIPS-om. Kao što je spomenuto, razvijena je kao naslednik popularne biblioteke PyKnow, tako da je usklađena sa novijim verzijama Python-a. Razvoj sistema zasnovanih na pravilima prvobitno se oslanjao na biblioteku PyKnow, koja je bila izbor mnogih autora za projekte poput unapređenja farmaceutske prakse [1] i bezbednosti podataka na blockchain tehnologijama [3]. Sa pojavom Experte, zabeležena je smena u preferencijama, pri čemu je ova biblioteka postala popularnija u razvoju sistema, uključujući dijagnostiku korona virusa [10] i podršku odlučivanju u logističkim operacijama[14]. *Durable rules* je open-source biblioteka za analizu i koordinaciju događaja u realnom vremenu. Implementirana je u programskom jeziku C, što dovodi do brze evaluacije pravila i doprinosi performansama sistema. U sistemu za utovar paleta u vozila sa planiranjem ruta [22], biblioteka Durable rules pomogla je u kreiranju i korišćenju pravila za utovar i rutiranje, čineći raspodelu tereta bržom i lakšom.

4. DEMONSTRACIONI PRIMER

U nastavku poglavlja biće predstavljen *sistem dizajniran da unapredi obračun članarina u baletskom studiju*. Sistem je namenjen da olakša određivanje mesečnog iznosa članarine

za baletske igrače. *Ekspert* je zaposleni u baletskom studiju, a *korisnik* baletski igrač. Sistem menja eksperta, odnosno zaposlenog u baletskom studiju. Baletski igrači unose osnovne informacije o sebi i svojim aktivnostima unutar studija, nakon čega sistem automatski izračunava iznos članarine koji je potrebno uplatiti za tekući mesec. Cilj sistema je da pruži precizan i efikasan način upravljanja finansijskim obavezama članova studija.

Pravila na kojima će se zasnivati sistem nalaze se u nastavku:

1. Osnovna cena članarine se izračunava na osnovu starosti baletskog igrača. Za baletske igrače do 12 godina osnovna članarina iznosi 2,000 dinara.
2. Za baletske igrače od 12 do 18 godina osnovna članarina iznosi 3,000 dinara.
3. Za baletske igrače starije od 18 godina, osnovna članarina iznosi 3,500 dinara.
4. Svaki baletski igrač pri upisu dobija porodični indeks koji označava redosled upisa dece iz iste porodice u baletski studio. Porodice koje imaju upisano više dece u baletski studio ostvaruju pravo na popuste u zavisnosti od porodičnog indeksa deteta. Baletski igrači sa porodičnim indeksom 2 ostvaruju popust od 20% na članarinu.
5. Za baletske igrače sa porodičnim indeksom 3 ili većim, članarina se ne naplaćuje.
6. Cena individualnog časa za baletske igrače je 500 dinara. U zavisnosti od broja održanih individualnih časova članarina se povećava.
7. Baletski igrači koji treniraju više od 8 godina ostvaruju popust od 50% na članarinu.
8. Baletski igrači sa iskustvom dužim od 12 godina oslobođeni su plaćanja članarine.
9. Plaćanjem članarine unapred za celu godinu, baletski igrači ostvaruju 30% popusta.
10. Baletski igrači koji su volonteri (volontiraju na događajima studija) ostvaruju popust od 50% na članarinu.

4.1. *Implementacija u PYKE biblioteci*

Pyke ima nekoliko različitih vrsta izvornih datoteka. Sva pravila se nalaze u datoteci clanarina_pravila**.krb** (knowledge rule base - krb). Pravila se nalaze pojedinačno, ona nisu ugnježdena ili eksplicitno povezana jedna sa drugima. Ova biblioteka podržava dva *mehanizma zaključivanja*: ulančavanje unapred i ulančavanje unazad, svaki sa svojom jedinstvenom sintaksom. Kada se implementiraju pravila za ulančavanje unapred, često se dešava da različita pravila istovremeno zadovoljavaju uslove za primenu. Ovakvi *konflikti* među pravilima rešavaju se strategijom koja se bazira na redosledu pravila unutar baze pravila. To znači da pravila definisana ranije imaju veći prioritet u primeni u odnosu na ona koja su definisana kasnije. Ova biblioteka ne sadrži *mehanizme za objašnjavanje*.

Jedna .krb datoteka, odnosno baza pravila, može da sadrži i pravila ulančavanja unapred i pravila ulančavanja unazad. U nastavku je data implementacija pravila, deo primera je odrađen ulančavanjem unapred, a deo ulančavanjem unazad. Zbog problema kompatibilnosti ove biblioteke sa novijim verzijama Python-a, implementacija pravila nije u potpunosti dovršena.

Kroz objašnjenje implementacije prvog pravila iz demonstracionog primera biće ilustrovana upotreba mehanizma zaključivanja unapred. Pravilo se sastoji iz dva dela, ali se umesto ključnih reči *if* i *then* koriste *foreach* i *assert*. Upotreba *foreach* i *assert* identifikuje pravilo kao pravilo ulančavanja unapred. Klauzula *foreach* nam omogućava da prođemo kroz sve dostupne činjenice o baletskim igračima. Koristeći ključnu reč *check*, za svakog baletskog igrača proveravamo da li je mlađi od 12 godina. Ukoliko igrač ispunjava ovaj uslov, pomoću klauzule *assert* u radnu memoriju unosi se nova činjenica. Ta činjenica odnosi se na to da je za baletskog igrača, identifikovanog preko njegovog jedinstvenog identifikatora, obračunata članarina u iznosu od 2000 dinara.

```
clanarina_ispod_12_godina
    foreach
    podaci.baletski_igrac($igrac_id, $godine, $poro-
dica_indeks, $broj_casova, $iskustvo, $placanje_una-
pred, $volonter)
       check $godine <= 12
    assert
        podaci.clanarina($igrac_id, 2000)
```

Pravila prikazana u nastavku odrađena su po sličnom principu, mehanizmom ulančavanja unapred.

```
clanarina_od_12_do_18
    foreach
   podaci.baletski_igrac($igrac_id, $godine, $poro-
dica_indeks, $broj_casova, $iskustvo, $placanje_una-
pred, $volonter)
        check $godine > 12
        check $godine <= 18
    assert
        podaci.clanarina($igrac_id, 3000)
```

```
clanarina_iznad_12_godina
    foreach
 podaci.baletski_igrac($igrac_id, $godine, $porodi-
ca_indeks, $broj_casova, $iskustvo, $placanje_una-
pred, $volonter)
        check $godine > 18
    assert
        podaci.clanarina($igrac_id, 3500)
```

```
drugo_dete_iz_porodice
    foreach
podaci.baletski_igrac($igrac_id, $godine, $porodica_
indeks, $broj_casova, $iskustvo, $placanje_unapred,
$volonter)
        check $porodica_indeks == 2
        podaci.clanarina($igrac_id, $clanarina)
    retract
        podaci.clanarina($igrac_id, $clanarina)
    assert
       podaci.clanarina($igrac_id, $clanarina * 0.8)
```

```
placanje_unapred
    foreach
```

```
podaci.baletski_igrac($igrac_id, $godine, $porodica_
indeks, $broj_casova, $iskustvo, $placanje_unapred,
$volonter)
        check $placanje_unapred == True
        podaci.clanarina($igrac_id, $clanarina)
    retract
        podaci.clanarina($igrac_id, $clanarina)
    assert
     podaci.clanarina($igrac_id, $clanarina * 0.7)
```

Kroz objašnjenje implementacije pravila koja se odnose na ostvarivanje besplatne članarine u baletskom studiju (peto i osmo pravilo) biće ilustrovana upotreba mehanizma zaključivanja unazad. Pravilo se takođe sastoji iz dva dela, ali se koriste ključne reči *use* i *when*. Upotreba *use* i *when* identifikuje pravilo kao pravilo ulančavanja unazad. Cilj koji želimo da dokažemo nalazi se u okviru *use* klauzule. U okviru *when* klauzule definišu se uslovi koje je potrebno ispuniti kako bi cilj bio dokazan. U našem slučaju cilj, odnosno besplatna članarina, biće ostvaren ako baletski igrač ima iskustvo veće od 12 godina ili porodični indeks veći od 3.

```
besplatna_clanarina
   use clanarina($igrac_id, 0)
   when
       podaci.baletski_igrac($igrac_id,    $godine,
$porodica_indeks, $broj_casova, $iskustvo, $placa-
nje_unapred, $volonter)
check $iskustvo > 12 or $porodica_indeks >= 3
```

U datoteci podac**i.kfb** (knowledge fact base - kfb), koja služi kao osnova za čuvanje činjenica u Pyke, nalaze se podaci o baletskim igračima. Ove činjenice obuhvataju informacije kao što su jedinstveni identifikator igrača, njihova starost, pripadnost porodici, broj individualnih časova koje su pohađali, iskustvo u baletu, informacije o plaćanju unapred, kao i status volontera. Specifičnost za činjenice u Pyke je da su nepromenjive, znači kada se jednom dodaju ne mogu se ažurirati.

```
baletski_igrac(1, 10, 1, 0, 2, False, False)
baletski_igrac(2, 15, 1, 3, 2, False, False)
baletski_igrac(3, 18, 1, 0, 6, False, True)
baletski_igrac(4, 9, 2, 0, 0, False, False)
baletski_igrac(5, 10, 1, 0, 3, True, False)
baletski_igrac(6, 16, 1, 2, 5, False, False)
```

Nakon kreiranja baze pravila i baze činjenica potrebno je pokrenuti program.

Prvi korak je *kreiranje engine objekta*. Engine objekat predstavlja centralnu tačku za upravljanje pravilima, činjenicama i procesom zaključivanja. Kada se kreira engine objekat, Pyke počinje sa procesom učitavanja i kompajliranja pravila i činjenica koja su definisana u Pyke izvornim datotekama. To znači da se izvorne datoteke pretvaraju u Python kod. Engine objekat se kreira na sledeći način:

```
from pyke import knowledge_engine
my_engine = knowledge_engine.engine(__file__)
```

Pravila ulančavanja unapred se automatski pokreću u trenutku kada se aktivira odgovarajuća baza pravila. To znači da se sva pravila koja su unapred definisana u aktiviranoj bazi izvršavaju, uzimajući u obzir dostupne činjenice u tom trenutku. U nastavku je prikazan poziv metode *activate* nad engine objektom.

```
my_engine.activate('clanarina_pravila')
```

Pre ili posle aktiviranja pravila, činjenice se mogu dinamički dodati u radnu memoriju. To se može uraditi pozivom metode *assert* nad engine objektom.

```
my_engine.assert_('podaci',  'baletski_igrac',  (8,
10, 1, 0, 2, False, False))
```

Za razliku od pravila ulančavanja unapred, koja se automatski aktiviraju, za pravila ulančavanja unazad mora se eksplicitno zahtevati aktiviranje. Potrebno je zatražiti od Pyke engine-a da dokaže određeni cilj. To se postiže upotrebom metode *prove_1_goal*, kojom se navodi cilj koji program treba da dokaže.

```
my_engine.prove_1_goal('clanarina_pravila.besplat-
na_clanarina(3, 0)')
```

4.2. *Implementacija u EXPERTA biblioteci*

U nastavku je data implementacija pravila korišćenjem biblioteke Experta. U biblioteci Experta činjenice su instance klasa koje nasleđuju klasu **Fact.** Činjenice mogu imati svoja polja definisana sa **Field**. Polja se mogu proglasiti obaveznim (*mandatory*) ili podrazumevanim (*default*). U našem primeru pomoću klasa BaletskiIgrač, Članarina i ObračunPopusta definišemo strukturu činjenica, a instance ovih klasa će biti konkretni podaci baletskih igrača. Klasa BaletskiIgrač sadrži osnovne podatke, kao što su jedinstveni identifikator igrača, starost, porodični indeks, broj individualnih časova koje su pohađali, iskustvo u baletu, informacije o plaćanju unapred, kao i status volontera. Klasa Članarina sadrži podatak o iznosu članarine za određenog baletskog igrača. Klasa ObračunPopusta prati informacije o tome koji baletski igrač je ostvario određeni tip popusta. Ova klasa omogućava da se spreči ponovno aktiviranje pravila sa istim tipom popusta za istog igrača ukoliko bi se ponovo evaluiralo.

```
class BaletskiIgrac(Fact):
    igrac_id = Field(int, default=0)
    godine = Field(int, default=0)
    porodica_indeks = Field(int, default=0)
    broj_casova = Field(int, default=0)
    iskustvo = Field(int, default=0)
    placanje_unapred = Field(bool, default=False)
    volonter = Field(bool, default=False)
```

```
class Clanarina(Fact):
    igrac_id = Field(int, default=0)
    iznos=Field(int, default=0)
```

```
class ObracunPopusta(Fact):
    igrac_id = Field(int, default=0)
    tip_popusta = Field(str)
```

Pravila se u biblioteci Experta definišu kao metode klase koja nasleđuje klasu **KnowledgeEngine**. Svaka metoda koja predstavlja pravilo označena je dekoratorom **@Rule.** Tu se definišu uslovi koji definišu kada se pravilo primenjuje, a u okviru tela metode se definiše šta pravilo treba da uradi. Ova biblioteka ne podržava mehanizam zaključivanja ulančavanje unazad, već samo ulančavanje unapred. Kroz objašnjenje implementacije prvog pravila iz demonstracionog primera biće ilustrovana upotreba ovog mehanizma zaključivanja. Ovo pravilo se aktivira ukoliko su ispunjeni određeni uslovi navedeni unutar zagrade nakon dekoratora @Rule. Prvo se proverava da li je činjenica koja je aktivirala pravilo tipa BaletskiIgrac. Za baletskog igrača se proverava da su njegove godine manje ili jednake 12 pomoću ***"P" (Predicate Field Constraint)***. Ako je uslov ispunjen, tj. ako je baletski igrač mlađi od 12 godina, pravilo se aktivira. Tada se koristi ***MATCH objekat*** da se dobije identifikator baletskog igrača koji je zadovoljio uslov. Zatim se koristi metoda ***declare*** kako bi se dodala nova činjenica tipa Clanarina. Ova nova činjenica sadrži identifikator igrača i iznos članarine postavljen na 2000.

```
class ObracunClanarina(KnowledgeEngine):

@Rule(BaletskiIgrac(godine=P(lambda godine: godine <= 12), igrac_id=MATCH.igrac_id))
def clanarina_ispod_12_godina(self, igrac_id):
        self.declare(Clanarina(igrac_id=igrac_id, iznos=2000))

@Rule(BaletskiIgrac(godine=P(lambda godine: 13 <= godine <= 18), igrac_id=MATCH.igrac_id))
 def clanarina_od_12_do_18(self, igrac_id):
        self.declare(Clanarina(igrac_id=igrac_id, iznos=3000))

@Rule(BaletskiIgrac(godine=P(lambda godine: godine > 18), igrac_id=MATCH.igrac_id))
def clanarina_iznad_18_godina(self, igrac_id):
        self.declare(Clanarina(igrac_id=igrac_id, iznos=3500))
```

Pravila mogu da sadrže i više uslova koji se kombinuju pomoću logičkih operatora AND, OR i NOT. Ako nije eksplicitno naveden ni jedan logički operator, već su činjenice samo odvojene zarezima, podrazumeva se da je operator AND. To je prikazano u narednom pravilu, koje će se aktivirati ako baletski igrač ima porodični indeks 2 i ako prethodno nije primenjen popust za drugo dete za tog baletskog igrača. ***AS objekat*** se koristi da bi se dobila referenca na činjenicu koja predstavlja iznos članarine za odgovarajućeg baletskog igrača. Kada se pravilo aktivira, koristi se metoda ***modify*** za postavljanje novog iznosa članarine (20% popusta) za tog baletskog igrača. Takođe, pravilo dodaje činjenicu o obračunu ovog tipa popusta za tog igrača kako bi se sprečilo ponovno aktiviranje pravila za istog igrača. Pored metoda declare i modify, u biblioteci Experta za rad sa činjenicama mogu se koristiti i metode *retract* (uklanja postojeću činjenicu) i *duplicate* (dodaje novu činjenicu koristeći postojeću kao šablon i dodajući neke izmene).

```
@Rule(BaletskiIgrac(porodica_indeks=2, igrac_id=MATCH.igrac_id),
        NOT(ObracunPopusta(igrac_id=MATCH.igrac_id, tip_popusta="drugo_dete")),
        AS.r_clanarina << Clanarina(igrac_id=MATCH.igrac_id, iznos=MATCH.iznos))
def drugo_dete_iz_porodice(self, igrac_id, iznos, r_clanarina):
self.modify(r_clanarina, iznos=int(iznos * 0.8))
self.declare(ObracunPopusta(igrac_id=igrac_id, tip_popusta="drugo_dete"))
```

```
@Rule(BaletskiIgrac(porodica_indeks=P(lambda porodica_indeks: porodica_indeks >= 3), igrac_id=MATCH.igrac_id),
        NOT(ObracunPopusta(igrac_id=MATCH.igrac_id, tip_popusta="trece_dete")),
        AS.r_clanarina << Clanarina(igrac_id=MATCH.igrac_id, iznos=MATCH.iznos))
def trece_dete_iz_porodice(self, igrac_id, iznos, r_clanarina):
self.modify(r_clanarina, iznos=0)
self.declare(ObracunPopusta(igrac_id=igrac_id, tip_popusta="trece_dete"))
```

```
@Rule(BaletskiIgrac(iskustvo=P(lambda iskustvo: iskustvo > 8), igrac_id=MATCH.igrac_id),
        NOT(ObracunPopusta(igrac_id=MATCH.igrac_id, tip_popusta="iskustvo_preko_12")),
       AS.r_clanarina << Clanarina(igrac_id=MATCH.igrac_id, iznos=MATCH.iznos))
def iskustvo_preko_12_godina(self, igrac_id, iznos, r_clanarina):
self.modify(r_clanarina, iznos=0)
self.declare(ObracunPopusta(igrac_id=igrac_id, tip_popusta="iskustvo_preko_12"))
```

```
@Rule(BaletskiIgrac(placanje_unapred=True,igrac_id=MATCH.igrac_id,
    NOT(ObracunPopusta(igrac_id=MATCH.igrac_id, tip_popusta="placanje_unapred")),
        AS.r_clanarina << Clanarina(igrac_id=MATCH.igrac_id, iznos=MATCH.iznos))
def placanje_unapred(self, igrac_id, iznos, r_clanarina):
self.modify(r_clanarina, iznos=int(iznos * 0.7))
self.declare(ObracunPopusta(igrac_id=igrac_id, tip_popusta="placanje_unapred"))
```

Kao i kod Pyke biblioteke i ovde se pravila izvršavaju redosledom kojim su definisana u okviru baze pravila. U ovom demonstracionom primeru neophodno je da se pre bilo kakvih obračuna popusta odredi osnovna cena članarine (prvo, drugo i treće pravilo). Kada bi se ova tri pravila prebacila na kraj baze pravila program ne bi funkcionisao dobro. Ova biblioteka ima još jednu strategiju za rešavanje konflikta koji mogu nastati, a to je definisanje prioriteta pravila preko ***salience*** *vrednosti*.

Ova vrednost određuje prioritet, pa time i redosled kojim će se pravila izvršavati. Pravila koja imaju veću vrednost za salience izvršavaće se pre pravila sa manjom vrednošću za salience. Ukoliko ne piše ni jedna vrednost, kao što je to bio slučaj u prethodno definisanim pravilima, podrazumevana vrednost je nula. U nastavku je dat prikaz kako bi se salience mogao primeniti. Pravilo za određivanje osnovne cene članarine aktiviraće se pre pravila za obračun popusta, jer ima veću vrednost za salience.

```
@Rule(salience=1,BaletskiIgrac(porodica_indeks=2,igrac_id=MATCH.igrac_id),
        NOT(ObracunPopusta(igrac_id=MATCH.igrac_id, tip_popusta=”drugo_dete”)),
        AS.r_clanarina << Clanarina(igrac_id=MATCH.igrac_id, iznos=MATCH.iznos))
def drugo_dete_iz_porodice(self, igrac_id, iznos, r_clanarina):
self.modify(r_clanarina, iznos=int(iznos * 0.8))
self.declare(ObracunPopusta(igrac_id=igrac_id, tip_popusta=”drugo_dete”))
```

```
@Rule(salience=2,BaletskiIgrac(godine=P(lambda godine: godine <= 12), igrac_id=MATCH.igrac_id))
def clanarina_ispod_12_godina(self, igrac_id):
          self.declare(Clanarina(igrac_id=igrac_id, iznos=2000))
```

U nastavku je prikazan kod koji je potreban da bi se program pokrenuo, kao i rezultat koji se dobija. Prvo se instancira engine, nakon čega se poziva *reset* metoda kako bi se pripremila radna memorija. Zatim su dodati podaci o nekoliko baletskih igrača. Nakon toga, izvršava se metoda *run* kako bi se pokrenulo zaključivanje na osnovu unetih podataka. Na kraju, prolazi se kroz sve činjenice koje su dodate u radnu memoriju i proverava da li su instance klase Clanarina. Ako jeste, ispisuje se identifikator baletskog igrača i iznos njegove članarine.

```
>>> engine = ObracunClanarina()
>>> engine.reset()
>>>  engine.declare(BaletskiIgrac(igrac_id=1, godine=10,
porodica_indeks=0,broj_casova=3,iskustvo=2,placanje_unapred=False,
volonter=False))
>>>  engine.declare(BaletskiIgrac(igrac_id=2, godine=8,
porodica_indeks=3,broj_casova=0,iskustvo=6,placanje_unapred=False, volonter=False))
>>>  engine.declare(BaletskiIgrac(igrac_id=3, godine=18,
porodica_indeks=2,broj_casova=0,iskustvo=10,placanje_unapred=False,volonter=False)
>>>  engine.declare(BaletskiIgrac(igrac_id=4, godine=11,
porodica_indeks=0,broj_casova=0,iskustvo=5,placanje_unapred=False, volonter=True))
>>> engine.run()
>>> for fact in engine.facts.values():
       if isinstance(fact, Clanarina):
print(“Članarina za baletskog igrača ID {0}: {1} RSD”.format(fact[‘igrac_id’], fact[‘iznos’]))
```

```
Članarina za baletskog igrača ID 1: 3500 RSD
Članarina za baletskog igrača ID 2: 0 RSD
Članarina za baletskog igrača ID 3: 1200 RSD
Članarina za baletskog igrača ID 4: 1000 RSD
```

4.3. *Implementacija u DURABLE RULES biblioteci*

U nastavku je data implementacija pravila korišćenjem biblioteke Durable rules. Korišćenjem ove biblioteke pravila se grupišu u skupove pravila, poznate kao ***rulesets***. Svaki skup pravila dobija svoje ime, u ovom slučaju ***obracun_clanarina***. Za definisanje kada će se pravilo aktivirati, koristi se dekorator **@when_all**. Ova biblioteka ne podržava mehanizam ulančavanja unazad, već samo unapred. Dekorator @when_all prihvata jedan ili više uslova koji moraju biti ispunjeni da bi se pravilo izvršilo. Kada se koristi deo **c.igrac <<** zapravo se hvata i čuva činjenica koja zadovoljava određene uslove u promenljivu nazvanu igrac. To omogućava da činjenica bude dostupna za korišćenje unutar funkcije koja implementira pravilo. Simbol **m** omogućava da se direktno pristupi činjenici koja je aktivirala pravilo. Dakle, prvo pravilo iz demonstracionog primera će se aktivirati ako je igrač mlađi od 12 godina i ako prethodno za njega nije postavljena članarina (članarina je nula, ali ne zbog ostvarivanja nekog od popusta za besplatnu članarinu). Metodom ***retract_fact*** činjenica se uklanja iz radne memorije. Metodom ***assert_fact*** nova činjenica se dodaje u radnu memoriju sa ažuriranim atributima. Kod nove činjenice izmenjen je atribut članarina, koji je sada 2000, dok su svi ostali isti (prenose se iz originalne činjenice igrač koja je aktivirala pravilo).

```
from durable.lang import *

with ruleset(‘obracun_clanarina’):

@when_all(c.igrac << (m.godine <= 12) & (m.clanarina == 0)  & (m.popust_indeks_3 == False) & (m.popust_iskustvo_12 == False))
def clanarina_ispod_12_godina(c):
       c.retract_fact(c.igrac)
c.assert_fact({‘igrac_id’:c.igrac.igrac_id,’godine’:c.igrac.godine,’porodica_indeks’:c.igrac.porodica_indeks,’broj_casova’:c.igrac.broj_casova,’iskustvo’:c.igrac.iskustvo,’placanje_unapred’:c.igrac.placanje_unapred,’volonter’:c.igrac.volonter,’clanarina’:2000,’popust_indeks_2’:c.igrac.popust_indeks_2,’popust_indeks_3’:c.igrac.popust_indeks_3,’ind_cas’:c.igrac.ind_cas,’popust_iskustvo_8’:c.igrac.popust_iskustvo_8,’popust_iskustvo_12’:c.igrac.popust_iskustvo_12,’popust_placanje_unapred’:c.igrac.popust_placanje_unapred,’popust_volonter’:c.igrac.popust_volonter})
```

Ostala pravila implementirana su na sličan način. Svako od pravila aktiviraće se ako je ispunjen odgovarajući uslov za popust i ako prethodno nije primenjen taj tip popusta za tog baletskog igrača.

```
@when_all(c.igrac << (m.godine >= 13) & (m.godine <= 18)) & (m.clanarina == 0)  & (m.popust_indeks_3 == False) & (m.popust_iskustvo_12 == False))
```

```
    def clanarina_od_12_do_18(c):
        c.retract_fact(c.igrac)
c.assert_fact({'igrac_id':c.igrac.igrac_id,'godine':c.igrac.godine,'porodica_indeks':c.igrac.porodica_indeks,'broj_casova':c.igrac.broj_casova,'iskustvo':c.igrac.iskustvo,'placanje_unapred':c.igrac.placanje_unapred,'volonter':c.igrac.volonter,'clanarina':3000,'popust_indeks_2':c.igrac.popust_indeks_2,'popust_indeks_3':c.igrac.popust_indeks_3,'ind_cas':c.igrac.ind_cas,'popust_iskustvo_8':c.igrac.popust_iskustvo_8,'popust_iskustvo_12':c.igrac.popust_iskustvo_12,'popust_placanje_unapred':c.igrac.popust_placanje_unapred,'popust_volonter': c.igrac.popust_volonter})
```

```
@when_all(c.igrac << (m.godine > 18) & (m.clanarina == 0)  & (m.popust_indeks_3 == False) & (m.popust_iskustvo_12 == False))
def clanarina_iznad_18_godina(c):
        c.retract_fact(c.igrac)
c.assert_fact({'igrac_id':c.igrac.igrac_id,'godine':c.igrac.godine,'porodica_indeks':c.igrac.porodica_indeks,'broj_casova':c.igrac.broj_casova,'iskustvo':c.igrac.iskustvo,'placanje_unapred':c.igrac.placanje_unapred,'volonter':c.igrac.volonter,'clanarina':3500,'popust_indeks_2':c.igrac.popust_indeks_2,'popust_indeks_3':c.igrac.popust_indeks_3,'ind_cas':c.igrac.ind_cas,'popust_iskustvo_8':c.igrac.popust_iskustvo_8,'popust_iskustvo_12':c.igrac.popust_iskustvo_12,'popust_placanje_unapred':c.igrac.popust_placanje_unapred,'popust_volonter': c.igrac.popust_volonter})
```

```
@when_all(c.igrac << (m.porodica_indeks == 2) & (m.popust_indeks_2 == False))
def drugo_dete_iz_porodice(c):
        c.retract_fact(c.igrac)
c.assert_fact({'igrac_id':c.igrac.igrac_id,'godine':c.igrac.godine,'porodica_indeks':c.igrac.porodica_indeks,'broj_casova':c.igrac.broj_casova,'iskustvo':c.igrac.iskustvo,'placanje_unapred':c.igrac.placanje_unapred,'volonter':c.igrac.volonter,'clanarina':
c.igrac.clanarina*0.8,'popust_indeks_2':True,'popust_indeks_3':c.igrac.popust_indeks_3,'ind_cas':c.igrac.ind_cas,'popust_iskustvo_8':c.igrac.popust_iskustvo_8,'popust_iskustvo_12':c.igrac.popust_iskustvo_12,'popust_placanje_unapred':c.igrac.popust_placanje_unapred,'popust_volonter': c.igrac.popust_volonter})
```

```
@when_all(c.igrac << (m.porodica_indeks >= 3) & (m.popust_indeks_3 == False))
def trece_dete_iz_porodice(c):
        c.retract_fact(c.igrac)
c.assert_fact({'igrac_id':c.igrac.igrac_id,'godine':c.igrac.godine,'porodica_indeks':c.igrac.porodica_indeks,'broj_casova':c.igrac.broj_casova,'iskustvo':c.igrac.iskustvo,'placanje_unapred':c.igrac.placanje_unapred,'volonter':c.igrac.volonter,'clanarina':0,'popust_indeks_2':c.igrac.popust_indeks_2,'popust_indeks_3':True,'ind_cas':c.igrac.ind_cas,'popust_iskustvo_8':c.igrac.popust_iskustvo_8,'popust_iskustvo_12':c.igrac.popust_iskustvo_12,'popust_placanje_unapred':c.igrac.popust_placanje_unapred,'popust_volonter': c.igrac.popust_volonter})
```

```
@when_all(c.igrac << (m.iskustvo > 12) & (m.popust_iskustvo_12 == False))
def iskustvo_preko_12_godina(c):
        c.retract_fact(c.igrac)
c.assert_fact({'igrac_id':c.igrac.igrac_id,'godine':c.igrac.godine,'porodica_indeks':c.igrac.porodica_indeks,'broj_casova':c.igrac.broj_casova,'iskustvo':c.igrac.iskustvo,'placanje_unapred':c.igrac.placanje_unapred,'volonter':c.igrac.volonter,'clanarina':0,'popust_indeks_2':c.igrac.popust_indeks_2,'popust_indeks_3':c.igrac.popust_indeks_3,'ind_cas':c.igrac.ind_cas,'popust_iskustvo_8':c.igrac.popust_iskustvo_8,
'popust_iskustvo_12':True,'popust_placanje_unapred':c.igrac.popust_placanje_unapred,'popust_volonter': c.igrac.popust_volonter})
```

```
@when_all(c.igrac << (m.placanje_unapred == True) & (m.popust_placanje_unapred == False))
def placanje_unapred(c):
        c.retract_fact(c.igrac)
c.assert_fact({'igrac_id':c.igrac.igrac_id,'godine':c.igrac.godine,'porodica_indeks':c.igrac.porodica_indeks,'broj_casova':c.igrac.broj_casova,'iskustvo':c.igrac.iskustvo,'placanje_unapred':c.igrac.placanje_unapred,'volonter':c.igrac.volonter,'clanarina':
c.igrac.clanarina*0.7,'popust_indeks_2':c.igrac.popust_indeks_2,'popust_indeks_3':c.igrac.popust_indeks_3,'ind_cas':c.igrac.ind_cas,'popust_iskustvo_8':c.igrac.popust_iskustvo_8,'popust_iskustvo_12':c.igrac.popust_iskustvo_12,'popust_placanje_unapred':True,'popust_volonter': c.igrac.popust_volonter})
```

Naredno pravilo će se pokrenuti poslednje i na ekranu ispisati iznose članarina za sve igrače. Ti iznosi su obračunati na osnovu prethodnih pravila. ***Simbol*** **+** ispred m.clanarina omogućava da se funkcija rezultat pozove kada postoji činjenica sa definisanim poljem clanarina, bez obzira na njegovu vrednost.

```
    @when_all(+m.clanarina)
    def rezultat(c):
print("Članarina za baletskog igrača ID {0}: {1} RSD".format(c.m.igrac_id, c.m.clanarina))
```

Kao i kod prethodne dve biblioteke pravila se izvršavaju redosledom kojim su definisana u skupu pravila obracun_clanarina. U poglavlju koje se odnosi na implementaciju pravila primenom biblioteke Experta objašnjen je i pristup rešavanju konflikta definisanjem prioriteta pravila preko salience vrednosti. Biblioteka Durable rules takođe pruža ovu mogućnost. To je omogućeno korišćenjem funkcije ***pri***. Ovoj funkciji se kao parametar prosleđuje vrednost za salience. Suprotno od biblioteke Experta, ovde se prvo izvršavaju pravila koja imaju nižu vrednost za salience prosleđenu funkciji pri. U nastavku je prikazano korišćenje ove funkcije na istom primeru kao i kod biblioteke Experta. Pravilo za određivanje osnovne cene članarine aktiviraće se pre pravila za obračun popusta, jer je funkciji pri prosleđena niža vrednost za salience.

```
@when_all(pri(2), c.igrac << (m.porodica_indeks ==
2) & (m.popust_indeks_2 == False))
def drugo_dete_iz_porodice(c):
        c.retract_fact(c.igrac)
c.assert_fact({'igrac_id':c.igrac.igrac_id,'go-
dine':c.igrac.godine,'porodica_indeks':c.igrac.
porodica_indeks,'broj_casova':c.igrac.broj_ca-
sova,'iskustvo':c.igrac.iskustvo,'placanje_una-
pred':c.igrac.placanje_unapred,'volonter':c.igrac.
volonter,'clanarina':
c.igrac.clanarina*0.8,'popust_indeks_2':True,'po-
pust_indeks_3':c.igrac.popust_indeks_3,'ind_cas':c.
igrac.ind_cas,'popust_iskustvo_8':c.igrac.po-
pust_iskustvo_8,'popust_iskustvo_12':c.igrac.po-
pust_iskustvo_12,'popust_placanje_unapred':c.igrac.
popust_placanje_unapred,'popust_volonter':  c.igrac.
popust_volonter})
```

```
        @when_all(pri(1), c.igrac << (m.godine <= 12)
& (m.clanarina == 0)  & (m.popust_indeks_3 == False)
& (m.popust_iskustvo_12 == False))
def clanarina_ispod_12_godina(c):
        c.retract_fact(c.igrac)
c.assert_fact({'igrac_id':c.igrac.igrac_id,'go-
dine':c.igrac.godine,'porodica_indeks':c.igrac.
porodica_indeks,'broj_casova':c.igrac.broj_ca-
sova,'iskustvo':c.igrac.iskustvo,'placanje_una-
pred':c.igrac.placanje_unapred,'volonter':c.igrac.
volonter,'clanarina':2000,'popust_indeks_2':c.
igrac.popust_indeks_2,'popust_indeks_3':c.igrac.
popust_indeks_3,'ind_cas':c.igrac.ind_cas,'popust_
iskustvo_8':c.igrac.popust_iskustvo_8,'popust_isku-
stvo_12':c.igrac.popust_iskustvo_12,'popust_placa-
nje_unapred':c.igrac.popust_placanje_unapred,'popust_volonter':
c.igrac.popust_volonter})
```

U biblioteci Durable rules činjenice se definišu kao ***rečnici***, gde svaki par ključ-vrednost opisuje jedan atribut činjenice i njegovu vrednost. Definisanje činjenica na ovaj način može izazvati odgovarajuća ograničenja u pogledu modelovanja domena problema. Sa druge strane, korišćenje rečnika može biti veoma pogodno ako se javi potreba za rad sa JSON formatom, jer omogućava direktnu i intuitivnu konverziju podataka. U nastavku je prikazano dodavanje činjenica u definisani skup pravila korišćenjem metode *assert_fact*.

```
>>>assert_fact('obracun_clanarina',{'igrac_
id':1,'godine':10,'porodica_indeks':0,'broj_caso-
va':3,'iskustvo':2,'placanje_unapred':False,'vo-
lonter':False,'clanarina':0,'popust_indeks_2':Fal-
se,'popust_indeks_3':False,'ind_cas':False,'ind_
cas':False,'popust_iskustvo_8:False,'popust_isku-
stvo_12':False,'popust_placanje_unapred':False,
'popust_volonter': False})
>>>assert_fact('obracun_clanarina',{'igrac_
id':2,'godine':8,'porodica_indeks':3,'broj_caso-
va':0,'iskustvo':6,'placanje_unapred':False,'vo-
lonter':False,'clanarina':0,'popust_indeks_2':Fal-
se,'popust_indeks_3':False,'ind_cas':False,'ind_
cas':False,'popust_iskustvo_8:False,'popust_isku-
stvo_12':False,'popust_placanje_unapred':False,
'popust_volonter': False})
>>>assert_fact('obracun_clanarina',{'igrac_
id':3,'godine':18,'porodica_indeks':2,'broj_caso-
va':0,'iskustvo':10,'placanje_unapred':False,'vo-
lonter':False,'clanarina':0,'popust_indeks_2':Fal-
se,'popust_indeks_3':False,'ind_cas':False,'ind_
cas':False,'popust_iskustvo_8:False,'popust_isku-
stvo_12':False,'popust_placanje_unapred':False,'po-
pust_volonter': False})
>>>assert_fact('obracun_clanarina',{'igrac_
id':4,'godine':11,'porodica_indeks':0,'broj_caso-
va':0,'iskustvo':5,'placanje_unapred':False,'vo-
lonter':True,'clanarina':0,'popust_indeks_2':Fal-
se,'popust_indeks_3':False,'ind_cas':False,'ind_
cas':False,'popust_iskustvo_8:False,'popust_isku-
stvo_12':False,'popust_placanje_unapred':False,
'popust_volonter': False})
```

U biblioteci Durable rules, pored činjenica, postoji i ***koncept događaja***. Događaji, slično kao i činjenice, mogu se dodati u radnu memoriju i aktivirati neko od pravila iz skupa pravila. Ali, za razliku od činjenica, događaji su kao nešto što ne traje dugo, čim ih sistem iskoristi za pokretanje nekog pravila, on ih odmah briše. To znači da se svaki događaj koristi samo jednom - kad ga sistem uoči, odmah pokreće neku akciju kao odgovor, i nakon toga, taj događaj više nije dostupan za korišćenje. Način na koji se podaci dodaju u radnu memoriju određuje da li će se oni smatrati činjenicama ili događajima. Ako se podatak doda metodom *assert_fact* on će predstavljati činjenicu kao što je prikazano u prethodnom kodu. Sa druge strane, ako se podatak doda metodom *post* on će predstavljati *događaj*. U nastavku je prikazano dodavanje događaja u radnu memoriju.

```
>>>post('obracun_clanarina',{'igrac_id':1,'godi-
ne':10,'porodica_indeks':0,'broj_casova':3,'isku-
stvo':2,'placanje_unapred':False,'volonter':Fal-
se,'clanarina':0,'popust_indeks_2':False,'po-
pust_indeks_3':False,'ind_cas':False,'ind_
cas':False,'popust_iskustvo_8:False,'popust_isku-
stvo_12':False,'popust_placanje_unapred':False,
'popust_volonter': False})
>>>post('obracun_clanarina',{'igrac_id':2,'godi-
ne':8,'porodica_indeks':3,'broj_casova':0,'isku-
stvo':6,'placanje_unapred':False,'volonter':Fal-
se,'clanarina':0,'popust_indeks_2':False,'po-
pust_indeks_3':False,'ind_cas':False,'ind_
cas':False,'popust_iskustvo_8:False,'popust_isku-
stvo_12':False,'popust_placanje_unapred':False,
'popust_volonter': False})
>>>post('obracun_clanarina',{'igrac_id':3,'godi-
ne':18,'porodica_indeks':2,'broj_casova':0,'isku-
stvo':10,'placanje_unapred':False,'volonter':Fal-
se,'clanarina':0,'popust_indeks_2':False,'po-
pust_indeks_3':False,'ind_cas':False,'ind_
cas':False,'popust_iskustvo_8:False,'popust_isku-
stvo_12':False,'popust_placanje_unapred':False,'po-
pust_volonter': False})
>>>post('obracun_clanarina',{'igrac_id':4,'godi-
ne':11,'porodica_indeks':0,'broj_casova':0,'isku-
stvo':5,'placanje_unapred':False,'volon-
ter':True,'clanarina':0,'popust_indeks_2':Fal-
se,'popust_indeks_3':False,'ind_cas':False,'ind_
cas':False,'popust_iskustvo_8:False,'popust_isku-
stvo_12':False,'popust_placanje_unapred':False,
'popust_volonter': False})
```

Nakon kreiranja seta pravila i činjenica ili događaja, nema potrebe za pravljenjem engine objekta, već je *dovoljno samo pokrenuti program*. Na taj način automatski će se pokrenuti evaluacija pravila na osnovu podataka iz radne memorije i doneće se određeni zaključci. Rezultat koji će biti ispisan na ekranu prikazan je ispod.

```
Članarina za baletskog igrača ID 1: 3500 RSD
Članarina za baletskog igrača ID 2: 0 RSD
Članarina za baletskog igrača ID 3: 1200 RSD
Članarina za baletskog igrača ID 4: 1000 RSD
```

5. POREĐENJE BIBLIOTEKA

Uporedni prikaz biblioteka je dat u Tabeli 1, a objašnjenje poređenja po datim kriterijumima u pasusima koji slede.

U biblioteci Pyke, činjenice se predstavljaju kao izjave unutar baze znanja, koristeći sintaksu u kojoj se navodi naziv činjenice i vrednosti njenih atributa. Biblioteka Experta koristi objektno orijentisani pristup i predstavlja činjenice kao Python objekte. Ovo omogućava jednostavniji razvoj sistema koji imaju složene konceptualne modele. U biblioteci Durable rules činjenice se definišu kao rečnici. Iako ovaj način predstavljanja činjenica umanjuje jasnoću i struktuiranost koju pruža model zasnovan na klasama, on nudi svoje prednosti zbog intuitivne konverzije u JSON format. To može biti korisno u razvoju sistema koji se oslanjaju na web servise. Za razliku od biblioteka Pyke i Durable rules, biblioteka Experta dozvoljava da činjenice budu promenjive, odnosno da se mogu menjati tokom izvršavanja programa. To je korisno kod sistema gde se stanje činjenica menja i neophodno je prilagoditi pravila u skladu sa tim, kao što je to bio slučaj u prikazanom demonstracionom primeru. Kod biblioteka Pyke i Durable rules promene činjenica moraju se uraditi ručno, povlačenjem postojeće činjenice i kreiranjem nove ažurirane činjenice. Biblioteka Durable rules se izdvaja kroz koncept prolaznih događaja, koji nakon aktiviranja pravila nestaju, čime je omogućena trenutna i jednokratna reakcija sistema na specifične promene.

Biblioteka Pyke podržava dva ***mehanizma zaključivanja***: ulančavanje unapred i ulančavanje unazad pružajući korisnicima mogućnost da odaberu pristup koji odgovara njihovim potrebama. To je čini dobrim izborom za različite tipove sistema zasnovanim na pravilima, jer se može prilagoditi različitim scenarijima rešavanja problema. Biblioteke Experta i Durable rules podržavaju samo mehanizam ulančavanja unapred, što ih čini manje fleksibilnim u tom domenu.

Korišćenjem biblioteke Pyke ***konflikti*** koji mogu nastati kada različita pravila istovremeno zadovoljavaju uslove za primenu rešavaju se na osnovu redosleda pravila unutar baze znanja. To može otežati razvoj i održavanje sistema koji imaju veliki broj pravila. Sa druge strane biblioteke Experta i Durable rules nude naprednije mehanizme za rešavanje konflikata definisanjem vrednosti koja eksplicitno definiše prioritet pravila. Ispravno definisanje redosleda izvršavanja pravila je veoma bitno, jer u mnogim situacijama upravo od redosleda zavisi izlaz iz sistema.

Ni jedna od ove tri biblioteke koje su opisane u radu ne sadrži ***mehanizme za objašnjavanje***.

Biblioteka Pyke već dugo nije ažurirana što dovodi do nekompatibilnosti sa novijim verzijama Pythona. Sa druge strane biblioteke Experta i Durable rules su novije i kompatibilne sa novijim verzijama Pythona. To ih čini boljim izborom za novije projekte.

Biblioteke Pyke i Experta napisane su u Python-u. Ključni deo biblioteke Durable rules implementiran je u C-u što omogućava visoke performanse u evaluaciji pravila i obradi događaja.

Sve tri biblioteke su ***open source*** i potpuno su ***besplatne*** za korišćenje. Biblioteke Pyke i Durable rules se distribuiraju pod MIT licencom [27], dok se biblioteka Experta distribuira pod LGPLv3 licencom [28].

Dokumentacija za biblioteku Pyke je detaljna, ali je kao i sama biblioteka neažurirana i zastarela. Biblioteke Experta i Durable rules imaju noviju dokumentaciju, ali se dokumentacija biblioteke Experta izdvaja kao najbolje i najdetaljnije napisana.

Tabela 1 Poređenje biblioteka Pyke, Experta i Durable rules

Kriterijum	Pyke	Experta	Durable rules
Predstavljanje činjenica	Izjave sa atributima unutar baze znanja	Python objekti	Pyhton rečnici
Mehanizam zaključivanja	Ulančavanje unapred i ulančavanje unazad	Ulančavanje unapred	Ulančavanje unapred
Strategija za rešavanje konflikata	Na osnovu redosleda pravila	Definisanje prioriteta pravila (salience vrednost)	Definisanje prioriteta pravila (funkcija pri)
Mehanizam za objašnjavanje	Ne sadrži	Ne sadrži	Ne sadrži
Kompatibilnost sa Python-om	Neažuriran, nije kompatibilna sa novijim verzijama	Kompatibilna sa novijim verzijama	Kompatibilna sa novijim verzijama
Implementacija	Python	Python	Ključni deo implementiran u C-u
Licenca	MIT licenca	LGPLv3 licenca	MIT licenca
Dokumentacija	Detaljna ali zastarela	Detaljna	Detaljna

6. ZAKLJUČAK

Ovo istraživanje potvrđuje značajnu ulogu Python-a i njegovih biblioteka u razvoju sistema zasnovanih na pravilima. Pretragom kroz PyPI, GitHub, i različite internet forume identifikovano je dosta Python biblioteka koje se mogu koristiti za razvoj sistema zasnovanih na pravilima. U ovom radu fokus je stavljen na tri odabrane biblioteke: Pyke, Experta i Durable rules. Kroz demonstracioni primer sistema dizajniranog da unapredi obračun članarina u baletskom studiju prikazana je praktična primena ovih biblioteka. Pravila se odnose na određivanje visine članarine u baletskom studiju, uzimajući u obzir različite faktore poput starosti baletskog igrača, broj dece iz iste porodice upisane u studio, broja individualnih časova,

godine iskustva u baletu, unapred plaćene članarine i učešće u volonterskim aktivnostima. Implementacija deset ključnih pravila kroz sve tri biblioteke pokazuje njihove specifične mogućnosti i različite pristupe u implementaciji.

Svaka biblioteka nudi svoje prednosti koje ih čine boljim izborom za razvoj određenih vrsta sistema zasnovanih na pravilima. Biblioteka Experta se ističe svojim objektno orijentisanim pristupom, što je čini veoma intuitivnom i jednostavnom za korišćenje za programere koji su navikli na objektno orijentisanu paradigmu. Biblioteka Pyke se ističe podrškom za oba mehanizma zaključivanja: ulančavanje unapred i ulančavanje unazad, što programerima pruža mogućnost izbora strategije zaključivanja. Biblioteka Durable rules se ističe svojom jednostavnošću i performansama, što je čini dobrim izborom za sisteme koji zahtevaju brzo reagovanje. Izbor neke od ove tri biblioteke zavisio bi od specifičnih zahteva sistema koji se razvija.

LITERATURA

[1] Acolatse, D. K. (2018). Web-based pharmacist expert system for supporting community pharmacy practice–premier point chemists (Doctoral dissertation, Ashesi University).

[2] Bajec, M., & Krisper, M. (2005). A methodology and tool support for managing business rules in organisations. Information Systems, 30(6), 423-443.

[3] Bhattacharya, M. P., Zavarsky, P., & Butakov, S. (2020, October). Enhancing the security and privacy of self-sovereign identities on hyperledger indy blockchain. In 2020 International Symposium on Networks, Computers and Communications (ISNCC) (pp. 1-7). IEEE.

[4] Chung, K., Boutaba, R., & Hariri, S. (2016). Knowledge based decision support system. Information Technology and Management, 17, 1-3.

[5] Durable rules. [Online]. Available: https://pypi.org/project/durable-rules/

[6] Experta. [Online]. Available: https://experta.readthedocs.io/

[7] Houidi, Z. B. (2016, November). A knowledge-based systems approach to reason about networking. In Proceedings of the 15th ACM Workshop on Hot Topics in Networks (pp. 22-28).

[8] Liao, S. H. (2005). Expert system methodologies and applications—a decade review from 1995 to 2004. Expert systems with applications, 28(1), 93-103.

[9] Maryasin, O. Y., Ogarkhov, A. A., & Kolodkina, A. S. (2018, September). Rule-based hybrid building control system. In 2018 International Russian Automation Conference (RusAutoCon) (pp. 1-6). IEEE.

[10] Mouti, S. Expert system for corona diagnosis (ESCD). IOSR J. Comput. Eng.(IOSR-JCE), 23(1), 2278-0661.

[11] Python Knowledge Engine (Pyke). [Online]. Available: http://pyke.sourceforge.net/.

[12] Python Software Foundation. Python.org. Available: https://www.python.org/

[13] Ramani, M. S. S. (2007). A practical introduction to rule based expert systems.

[14] Vitković, N., Marinković, D., Stan, S. D., Simonović, M., Miltenović, A., Tomić, M., & Barać, M. (2024). Decision Support System for Managing Marshalling Yard Deviations. Acta Polytechnica Hungarica, 21(1).

[15] Wagner, W. P. (2017). Trends in expert system development: A longitudinal content analysis of over thirty years of expert system case studies. Expert systems with applications, 76, 85-96.

[16] TIOBE Index. TIOBE - The Software Quality Company. Available: https://www.tiobe.com/tiobe-index/

[17] Datathrillz. (2021, September 29). Considerations for building a rules engine in Python.https://datathrillz.com/considerations-for-building-a-rules-engine-in-python/

[18] CLIPS: A Tool for Building Expert Systems. Retrieved January 31, 2024, from https://www.clipsrules.net/

[19]·buguroo/pyknow. GitHub. Retrieved January 31, 2024, from https://github.com/buguroo/pyknow

[20] Garosi, F. pyclips: Python CLIPS interface (1.0.7.343) [Computer software]. Retrieved January 31, 2024, from http://pyclips.sourceforge.net

[21] McIntyre, S. (n.d.). rule-engine: A lightweight, optionally typed expression language with a custom grammar for matching arbitrary Python objects. (4.3.1) [Python; OS Independent]. Retrieved January 31, 2024, from https://github.com/zeroSteiner/rule-engine

[22] Nacakli, I., Guzel, S., & Zontul, M. (2022). Web Service-Based Two-Dimensional Vehicle Pallet Loading with Routing for a Real-World Problem. 2022 International Conference on Theoretical and Applied Computer Science and Engineering (ICTASCE), 153–156.https://doi.org/10.1109/ICTACSE50438.2022.10009852

[23] Petrilli, C. pyrules: A Python rules engine inspired by OPS5. (0.1.0) [Python]. Retrieved January 31, 2024, from https://github.com/petrilli/pyrules

[24] Python Rule Engine: Logic Automation & Examples | Nected Blogs. Retrieved January 31, 2024, from https://www.nected.ai/blog/python-rule-engines-automate-and-enforce-with-python

[25] The 7 families of artificial intelligence and their implementation in Python. Retrieved January 31, 2024, from https://abilian.com/en/news/the-7-families-of-artificial-intelligence-and-their-implementation-in-python/

[26] Stack Overflow. (2018, November 24). Python Rule Based Engine [Forum post]. https://stackoverflow.com/q/53421492

[27] Software Package Data Exchange (SPDX). The MIT License. Retrieved March 19, 2024, from https://spdx.org/licenses/MIT.html

[28] Software Package Data Exchange (SPDX). GNU Lesser General Public License v3.0 or later. Retrieved March 19, 2024, from https://spdx.org/licenses/LGPL-3.0-or-later.html

[29] Venmo. (n.d.). business-rules: Python DSL for setting up business intelligence rules that can be configured without code (1.1.1) [Python]. Retrieved January 31, 2024, from https://github.com/venmo/business-rules

[30] Walsh, M. J. . Intellect: A Domain-specific language and Rules Engine for Python (1.4.9) [Python; OS Independent]. Retrieved January 31, 2024, from http://github.com/nemonik/Intellect

Dragica Ljubisavljević, Fakultet organizacionih nauka, Katedra za softversko inženjerstvo, Univerzitet u Beogradu
Kontakt: dragica.ljubisavljevic@fon.bg.ac.rs
Oblasti interesovanja: softversko inženjerstvo i veštačka inteligencija, sa posebnim fokusom na procesiranju prirodnog jezika