UNIVERZITET U BEOGRAD
ELEKTROTEHNIČKI FAKULTET

Projekat za domaći rad

Programiranje u realnom vremenu

Teodora Aleksić, 2016/3144,
at163144m@student.etf.bg.ac.rs,
Beograd, januar 2018.

Sadržaj

Uvod

Projekat opisan u daljem tekstu je projekat za domaći zadatak na predmetu Programiranje u realnom vremenu, na Elektrotehničkom fakultetu, na Univerzitetu u Beogradu. Projekat predstavlja simulaciju sistema za nadzor rudnika. Postavka projektnog zadatka se može naći na zvaničnom sajtu predmeta [1].

U ovom dokumentu su opisani alati korišćeni za realizaciju projekta. Potom je opisana implementacija projekta uz priložene dijagrame. Na kraju je opisan proračun rasporedivosti realizovanog sistema u realnom vremenu.

1. Korišćeni alati

Projekat je razvijan i testiran na platformi Windows 10 64-bit. Za realizaciju projekta su korišćeni programski jezici C++17 i Python 3.6. Sav izvorni kod i dokumentacija su dostupni na GitHub repozitorijumu [2].

Sistem za nadzor rudnika je razvijen u programskom jeziku C++17 koristeći C++ standardnu biblioteku za ostvarivanje višenitne obrade i konkurentnosti [3] [4]. GUI je razvijen u jeziku Python 3.6 korišćenjem modula Tkinter [5] iz standardne Python 3.6 instalacije. Korišćeno razvojno okruženje je Visual Studio 2017 Community.

Pored standardnih biblioteka, korišćene su i dve open-source C++ biblioteke: *spdlog* i *fmt*. *Spdlog* je biblioteka za brzo i jednostavno logovanje [6], a *fmt* biblioteka za formatiranje stringova [7].

C++ deo projekta se uvozi u Python deo projekta u vidu Python modula korišćenjem Python C API funckija i varijabli [8].

Radi pokretanja programa, potrebno je smestiti sve C++ h i cpp fajlove i Python skripte u isti direktorijum. Potrebno je skinuti zaglavlja biblioteka *spdlog* i *fmt* sa njihovih GitHub repozitorijuma i smestiti ih u isti direktorijum. Instaliranje C++ fajlova kao Python modula se vrši sledećom komandom iz trenutnog direktorijuma:

pip3.6 install .

Program se potom pokreće sledećom komandom:

python3.6 real_time_mining.py

2. Implementacija

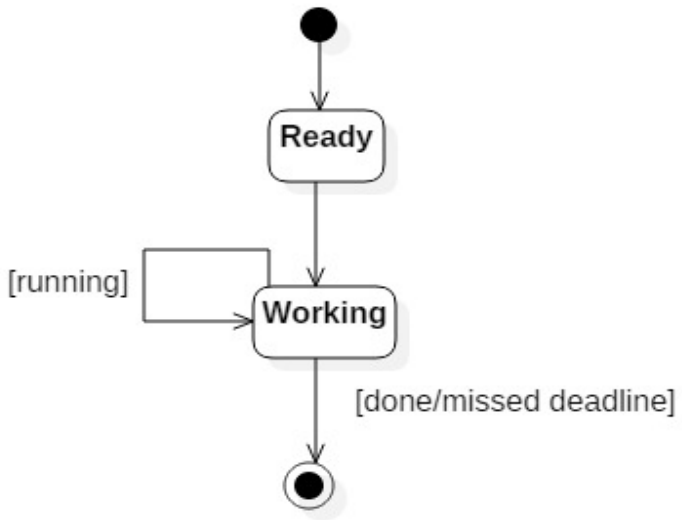
Za izradu UML modela korišćen je program StarUML 2 [9].

Detaljan dijagram klasa se usled svoje veličine može naći u dokumentaciji priloženoj uz izvorni kod. Opis relevantnih klasa je u nastavku:

- **Loggable**: osnovna klasa za logovanje. Predstavlja omotač oko loger objekta iz *spdlog* biblioteke. Loguje u fajl ili na standardni izlaz i na konzolu koja je deo korisničke aplikacije.
- **CustomSink**: klasa za logovanje za korisničku aplikaciju. Tipa je *Singleton*. Izvedena je iz klase *spdlog::sinks::sink* iz biblioteke *spdlog*.
- **Sleepable**: osnovna klasa za objekte koji mogu da se blokiraju do određenog vremenskog trenutka. Koristi mehanizme iz *chrono* zaglavlja iz standardne C++ bibloteke za merenje vreme. Sat realnog vremena koji koristi je *std::chrono::steady_clock*.
- **Runnable**: osnovna klasa za aktivne niti. Sadrži objekat *std::thread* iz standardne C++ biblioteke koji se pokreće za virtuelnu metodu *run* klase **Runnable**. Sadrži metode *start* i *stop* za započinjanje, tj. zaustavljanje niti.
- **Sensor**: klasa za senzor koji očitava vrednosti iz sredine. Izvedena je iz klasa **Loggable**, **Sleepable** i **Runnable**. Sadrži kontrolni, statusni i registar podataka. Statusni registar je tipa enumeracije **SensorStatus** i može imati vrednost *NONE*, *OK* i *ERROR* koje redom označavaju da senzor nije izračunao vrednost, da je senzor uspešno izračunao vrednost i da se pri izračunavanju vrednosti desila greška.
- **EnvironmentMonitor**: klasa koja kontroliše i očitava sve sonzore u sistemu. Izvedena je iz klasa **Loggable**, **Sleepable** i **Runnable**. Konstruiše i kontroliše senzore za CH4, CO i protok vazduha.
- **WaterTank**: klasa koja simulira cisternu vode. Izvedena je iz klasa **Loggable**, **Sleepable** i **Runnable**. Simulira kretanje vode u cisterni i signalizira sistem ako je došlo do prekoračenja gornje ili donje granice vode u cisterni.
- **WaterLevelMonitor**: klasa koja reaguje na nivo vode u cisterni. Izvedena je iz klasa **Loggable** i **Runnable**. Reaguje na događaj prekoračenja nivoa vode u cisterni i pali, tj. gasi pumpu.
- **PumpControl**: klasa za kontrolu i nadgledanje rada pumpe. Izvedena je iz klasa **Loggable**, **Sleepable** i **Runnable**. Zadužena je za paljenje, tj. gašenje pumpe pri manuelnoj komandi, događaju prekoračenja nivoa vode u cisterni i paljenja alarma za CH4. Nadgleda greške pri radu pumpe i ispravlja ih.

- **Simulation**: klasa zadužena za pokretanje i zaustavljanje simulacije. Izvedena je iz klasa **Loggable**, **Sleepable** i **Runnable**. Pokreće i zaustavlja objekte klasa koji učestvuju u simulaciji. Sadrži omotače za javne metode klasa koje učestvuju u simulaciji.
- **GlobalEvent**: klasa koja beleži globalne događaje u sistemu. Tipa je *Singleton*. Beleži bačen izuzetak u nekoj od niti zbog koga se zaustavlja simulacija. Beleži događaj prekoračenja nivoa vode u cisterni.

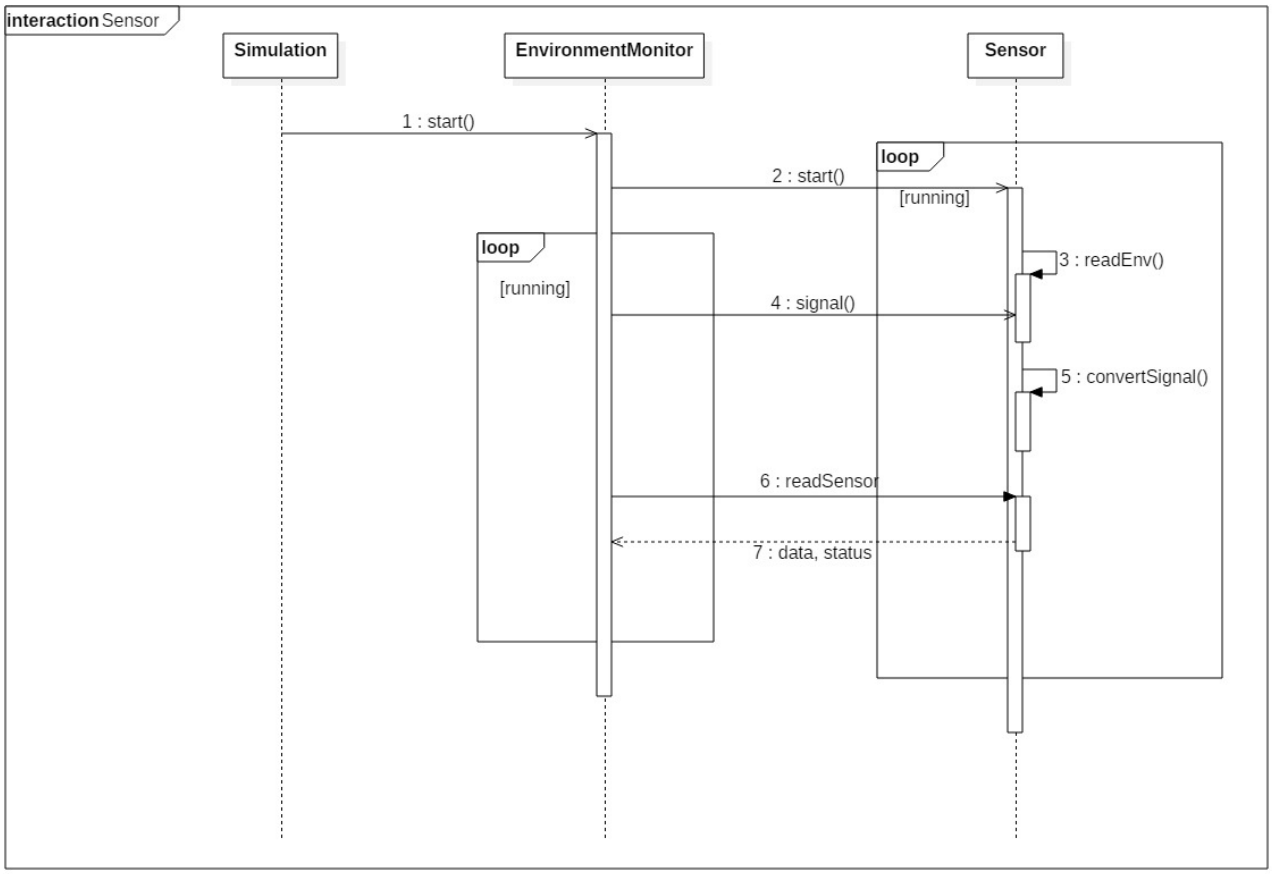
Objekti klasa **EnvironmentMonitor**, **WaterTank**, i **PumpControl** funkcionišu po principu periodičnih procesa. Počinju u inicijalnom **Ready** stanju i ostaju u stanju **Working** dok god se nit ne signalizira da se zaustavi ili dođe do prekoračenja roka. Njihov dijagram stanja se može videti u nastavku.



2.1. Dijagram stanja periodičnih procesa.

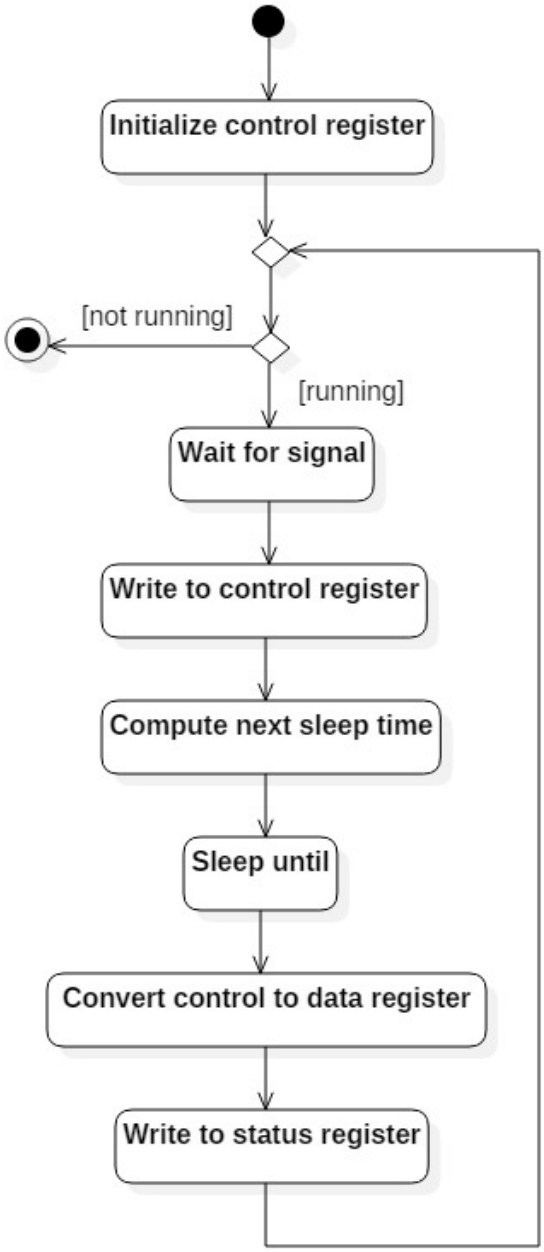
Objekat klase **WaterLevelMonitor** funkcioniše kao sporadični proces i aktivira se pri događaju prekoračenja nivoa vode u cisterni.

Objekti klase **Sensor** funkcionišu po principu pomeranja periode, tj. izvršavaju po jednu svoju iteraciju pri svakom signalu objekta klase **EnvironmentMonitor**. Objekat klase **EnvironmentMonitor**, na početku svog izvršavanja, započinje sve senzore i daje im signal da izvrše jednu svoju iteraciju. Potom očitava njihove rezultate i opet ih signalizira. Dijagram sekvence opisanog ponašanja se može videti ispod.



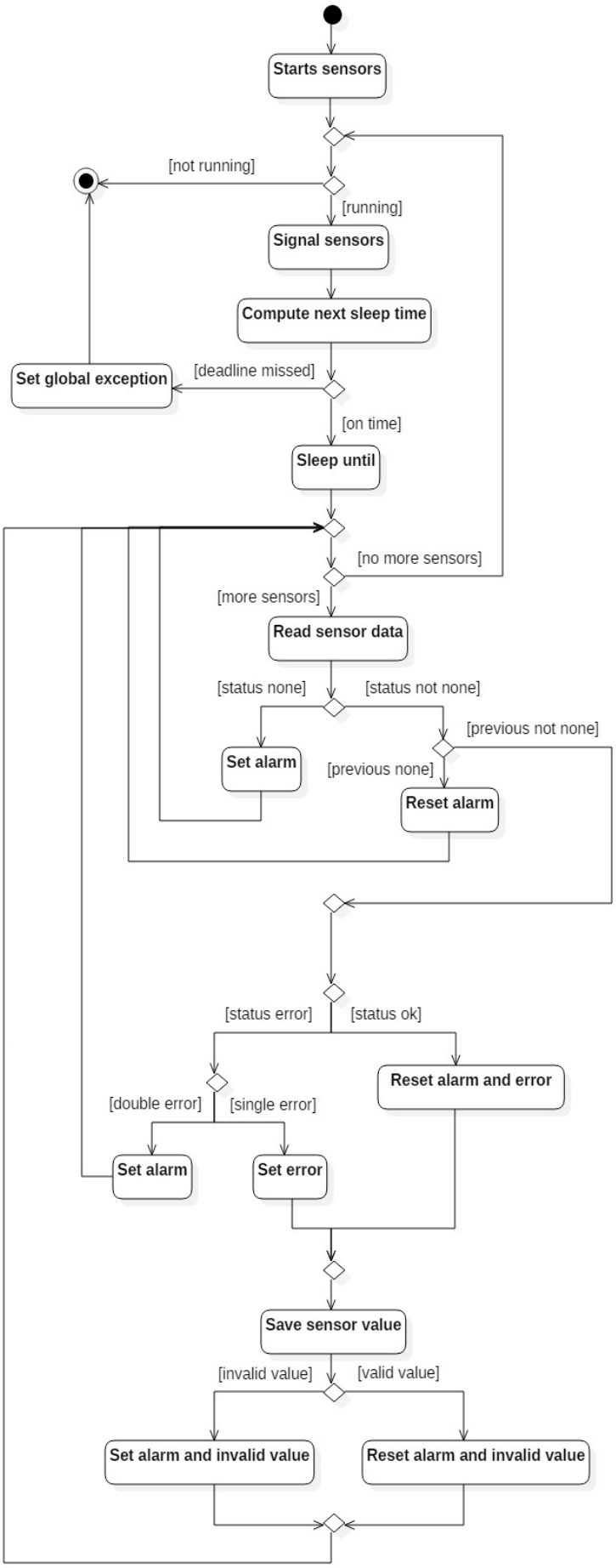
2.2. Dijagram sekvence senzora.

Ispod se može videti dijagram aktivnosti objekata klase **Sensor**. Po primanju signala objekta klase **EnvironmentMonitor**, očitava se vrednost kontrolnog registra. Potom je nit neaktivna 50ms, kojima se simulira A/D konverzija. Potom se upisuju odgovarajuće vrednosti u statusni i registar podataka senzora.



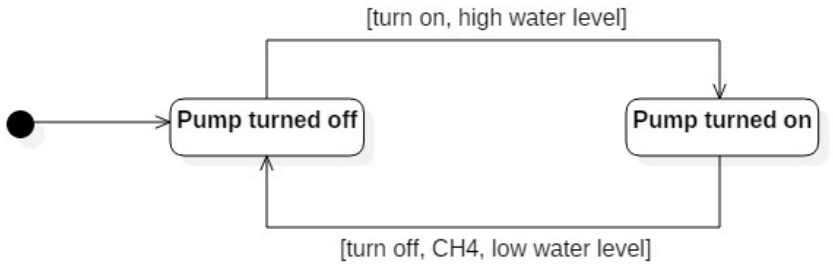
2.3. Dijagram aktivnosti objekta klase **Sensor**.

Ispod se može videti dijagram aktivnosti objekta klase **EnvironmentMonitor**. Na početku svog izvršavanja, započinje i signalizira sve senzore. Potom nit izračunava vreme do kog će biti neaktivna i do tada se blokira. Po aktivaciji, iterira po senzorima i čita njihove registre. Ako senzor nije uspeo da izračuna vrednost do tog trenutka ili je senzor zabeležio grešku pri izračunavanju ili je vrednost senzora prekoračila graničnu, postavlja se alarm tog senzora. Inače se proces ponavlja. Pri prekoračenju roka, postavlja se globalni izuzetak i simulacija se zaustavlja.



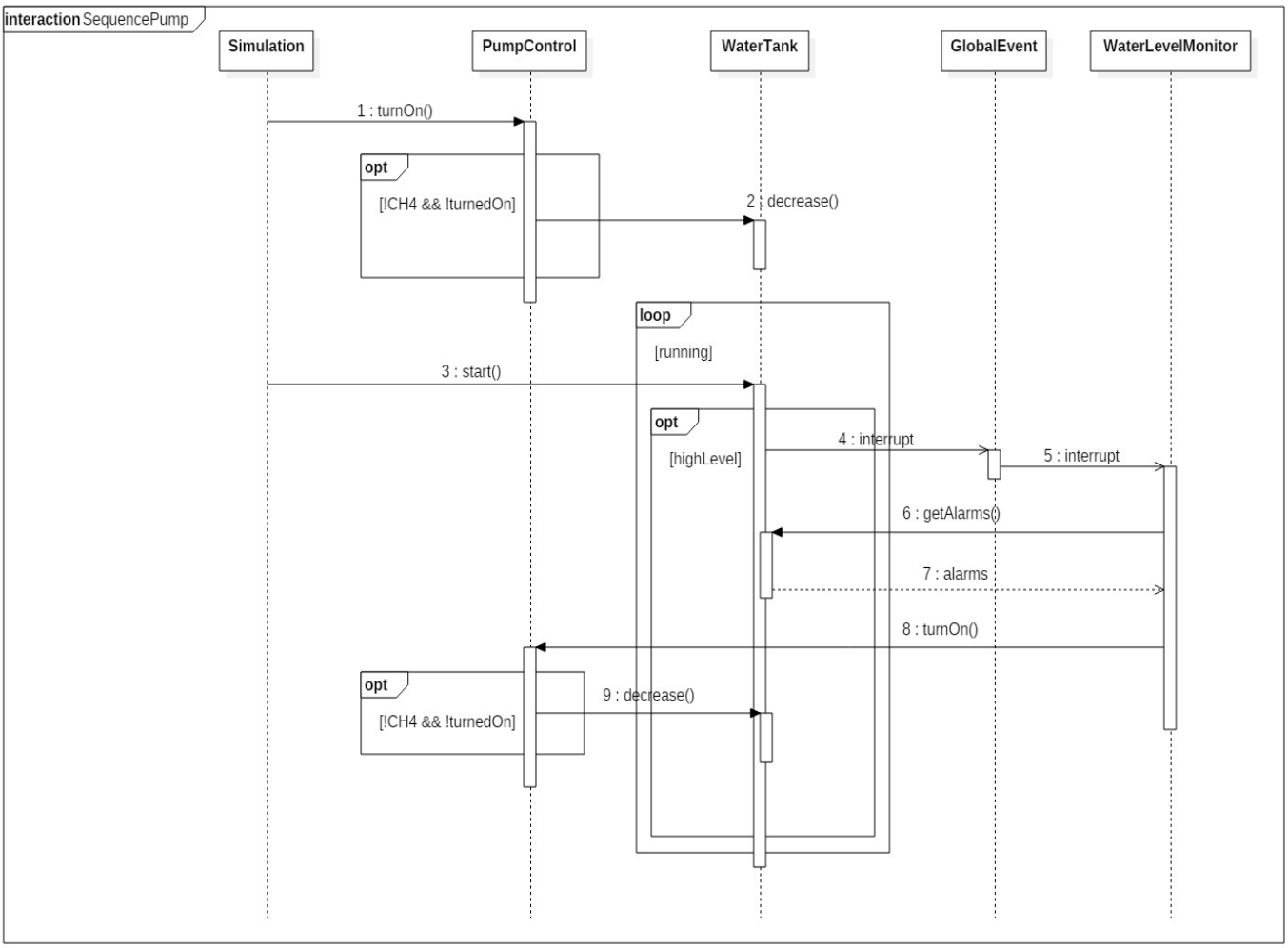
2.4. Dijagram aktivnosti objekta klase **EnvironmentMonitor**.

Pumpa se može naći u dva stanja: upaljenom i ugašenom. Paljenje pumpe se dešava pri visokom nivou vode u cisterni ili komandom operatera. Gašenje pumpe se dešava pri niskom nivou vode u cisterni, komandom operatera ili paljenjem CH4 alarma. Dijagram stanja pumpe se može videti u nastavku.



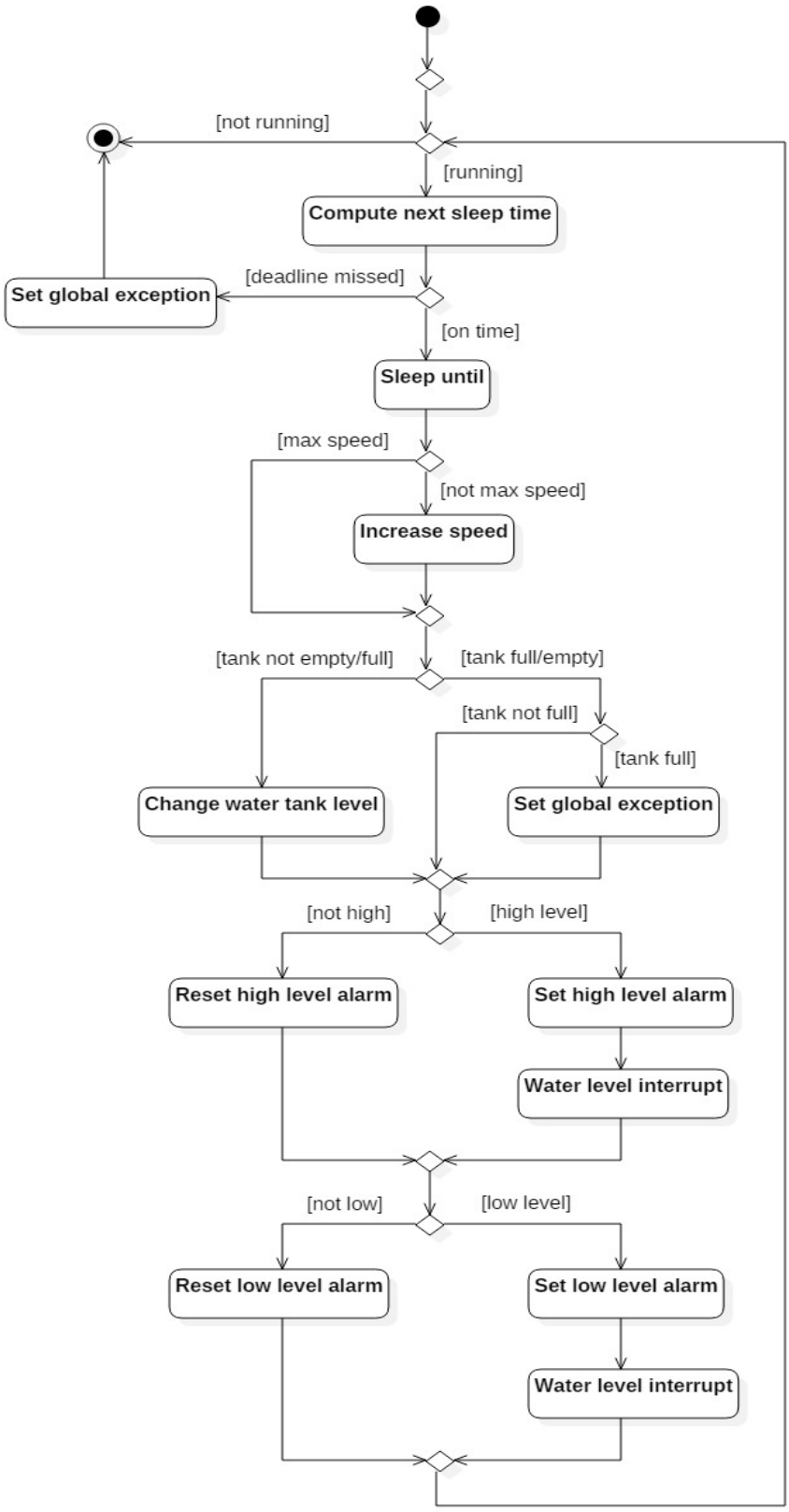
2.5. Dijagram stanja pumpe.

Za paljenje i gašenje pumpe je zadužen objekat klase **PumpControl**. Za stvarno menjanje nivoa vode u cisterni i detekciju prekoračenja nivoa vode u cisterni je zadužen objekat klase **WaterTank**. Za reagovanje na događaj visokog, tj. niskog nivoa vode u cisterni i paljenja, tj. gašenja pumpe u tom slučaju je zadužen objekat klase **WaterLevelMonitor**. Dijagram sekvence ovih klasa se može videti ispod.



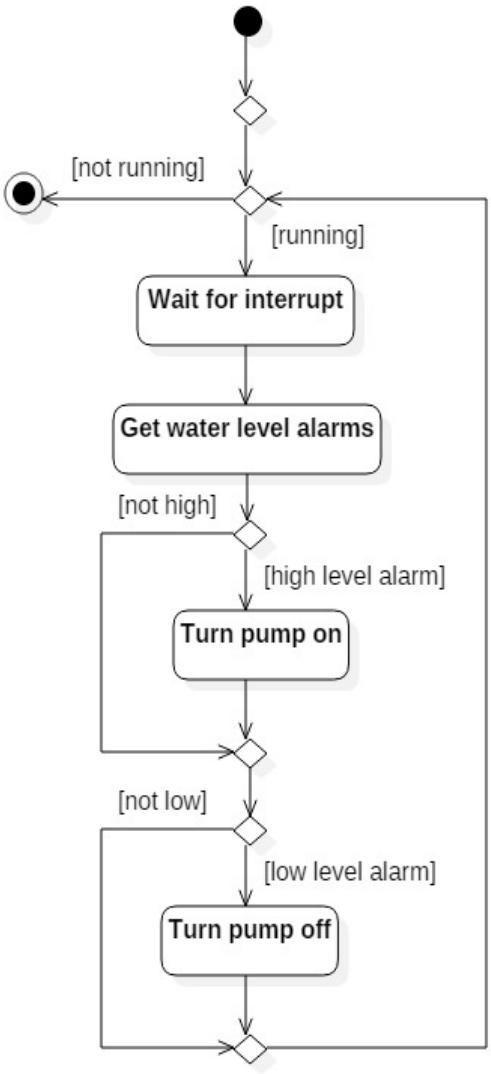
2.6. Dijagram sekvence pumpe.

Ispod se može videti dijagram aktivnosti objekta klase **WaterTank**. Na početku, nit izračunava vreme do kog će biti neaktivna i do tada se blokira. Po aktivaciji, menja nivo vode u cisterni i generiše globalni događaj u slučaju prekoračenja nivoa vode. Pri prekoračenju roka, postavlja se globalni izuzetak i simulacija se zaustavlja.



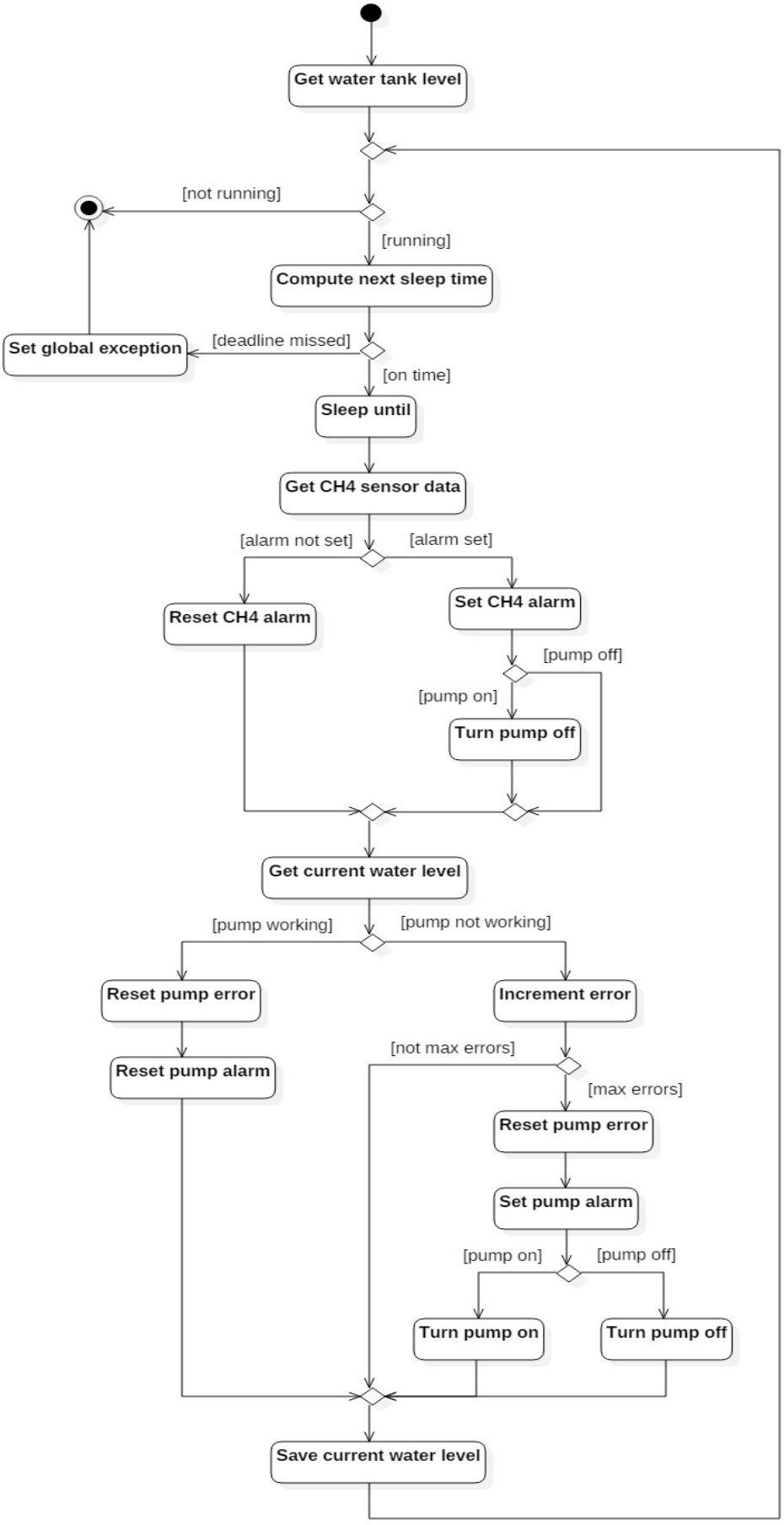
2.7. Dijagram aktivnosti objekta klase **WaterTank**.

Ispod se može videti dijagram aktivnosti objekta klase **WaterLevelMonitor**. Na početku izvršavanja, nit se blokira čekajući događaj prekoračenja nivoa vode u cisterni. Po dešavanju događaja, nit očitava alarme objekta klase **WaterTank**, prema njima zaključuje vrstu prekoračenja koja se desila i pali, tj. gasi pumpu.



2.8. Dijagram aktivnosti objekta klase **WaterLevelMonitor**.

Ispod se može videti dijagram aktivnosti objekta klase **PumpControl**. Na početku, nit izračunava vreme do kog će biti neaktivna i do tada se blokira. Po aktivaciji, proverava vrednost CH4 alarma. Ako je alarm uključen, gasi pumpu. Potom proverava stvarno stanje vode u cisterni i da li je došlo do greške u funckionisanju pumpe. Ako posle n=6 iteracija utvrdi da je došlo do greške, postavlja alarm pumpe i ponovo pali, tj. gasi pumpu. Pri prekoračenju roka, postavlja se globalni izuzetak i simulacija se zaustavlja.



2.9. Dijagram aktivnosti objekta klase **PumpControl**.

3. Proračun rasporedivosti

Projekat je razvijan i testiran na platformi Windows 10 64-bit. Na ovoj platformi, niti dobijaju prioritete prema zbiru prioriteta procesa kojem pripadaju i sopstvenog prioriteta kojeg imaju u okviru procesa [10]. Prema standardnoj konfiguraciji, pokrenut proces počinje sa prioritetom *NORMAL_PRIORITY_CLASS*, a niti u okviru njega sa prioritetom *THREAD_PRIORITY_NORMAL*. Taj zbir prioriteta je jednak 8. Sistem tretira niti istog prioriteta kao jednake i raspoređuje ih prema *round-robin* algoritmu. Prema tome, sve niti koje prokrene program započinju sa jednakim prioritetom 8.

Python interpreter je *single-threaded*, tj. omogućava izvršavanje samo jedne niti u jednom trenutku. Python GIL (*Global Interpreter Lock*), koji je zadužen za sekvencijalizovanje izvršavanja niti, se oslanja na implementaciju operativnog sistema za njihovo raspoređivanje. Prema tome, na izvršavanje programa će se gledati pojednostavljeno, kao da se izvršava na jednoprocesorskom sistemu, gde niti imaju isti prioritet i raspoređuju se prema *round-robin* algoritmu.

Pri merenju performansi realnog sistema, utvrđeno je da je vreme izvršavanja jedne iteracije niti u rangu mikrosekundi. Prema tome, u kritičnom trenutku, i uzevši u obzir način raspoređivanja niti na ciljanom sistemu, zaključuje se da su sve niti rasporedive u vremenskom roku bilo koje niti.

Posmatraju se niti koje izvršavaju telo *run* metoda **Sensor** i **EnvironmentMonitor** klasa. Perioda **EnvironmentMonitor** niti je 150ms. Ona je zadužena za očitavanje vrednosti senzora i signaliziranje senzora da počne sa izračunavanjem sledeće vrednosti. Nit koja izvršava **Sensor** prema tome ima istu periodu kao i **EnvironmentMonitor** nit koja je signalizira. Perioda **Sensor** niti je ujedno i njen rok. Ako **Sensor** nit prekorači svoj rok, to se obrađuje postavljanjem alarma tog senzora.

Perioda i ujedno i rok niti koja izvršava *run* metodu **WaterTank** klase je 150ms. U trenutku kada se u jednoj iteraciji izvršavanja ove niti promeni nivo vode u cisterni, takođe će se proveriti da li je došlo do prekoračenja nivoa vode. U tom trenutku će se signalizirati globalni događaj na koji treba da reaguje nit **WaterLevelMonitor**. Uzevši u obzir merenje performansi realnog sistema i način raspoređivanja niti na ciljanom sistemu, podrazumeva se da će se na događaj prekoračenja nivoa vode u cisterni reagovati u roku od 200ms.

Po standardnim podešavanjima, maskimalna brzina promene nivoa vode u cisterni je 0.08 po 150ms. Da bi se osiguralo da do prekoračenja nivoa vode u cisterni dođe najviše jednom u 5s, primenjuje se sledeća računica. Po standardnim podešavanjima, granica gornjeg prekoračenja nivoa vode u cisterni je 80.0, a granica donjeg prekoračenja nivoa vode u cisterni je 20.0. Ako neprekinuta manuelnim paljenjem, tj. gašenjem pumpe ili CH4 alarmom, nivo vode treba da se promeni za (80.0 - 20.0) = 60.0 da bi se desila dva ovakva događaja. Brzina kojom nivo vode u cisterni mora da se menja da bi se ovo desilo je 150ms * 60.0 / 5s = 1.8, što je daleko veća vrednost od 0.08 prema standardnim podešavanjima.

Perioda i ujedno i rok niti koja izvršava *run* metodu **PumpControl** klase je 150ms. Nit koja kontroliše pumpu reguliše i njen korektan rad. Ako se posle n=6 iteracija ustanovi da pumpa ne radi korektno, u istoj iteraciji se rad pumpe koriguje. Time se kvar pumpe ustanovljava posle 6 * 150ms = 900ms. Pošto se na grešku pri funkcionisanju pumpe reaguje odmah, osigurava se i da će se na njen kvar reagovati u roku od 300ms.

Rok za reagovonje na CH4 alarm ili nevalidnu vrednost je 400ms. Ako **EnvironmentMonitor** nit signalizira CH4 senzor i on odmah po dobijenom signalu izračuna vrednost koja je nevalidna ili prijavi grešku pri izračunavanju, **EnvironmentMonitor** nit će toga biti svesna tek posle pune periode od T = 150ms. Ako je **PumpControl** nit završila sa izvršavanjem u tom trenutku, ona će CH4 alarma biti svesna tek za svoju periodu od T = 150ms. To znači da od izračunavanja nevalidne vrednosti ili prijave greške do gašenja pumpe ima 2 * T = 300ms što je manje od roka od 400ms. Ako **EnvironmentMonitor** nit ili **PumpControl** nit prekorači svoj rok, podrazumeva se nemogućnost reagovanja na CH4 alarm u predefinisanom roku i pogram se zaustavlja.

Reference

[1] http://prv.etf.rs/prv/projekat/Projektni%20zadatak.pdf

[2] https://github.com/TeodoraAleksic/real-time-mining

[3] http://en.cppreference.com/w/cpp/thread

[4] http://en.cppreference.com/w/cpp/atomic

[5] https://wiki.python.org/moin/TkInter

[6] https://github.com/gabime/spdlog

[7] https://github.com/fmtlib/fmt

[8] https://docs.python.org/3.6/extending/extending.html

[9] http://staruml.io/

[10] https://msdn.microsoft.com/en-us/library/windows/desktop/ms685100(v=vs.85).aspx