UDC: 004.45 INFO M: str. 19-23

PROBLEM INTEGRACIJE VISOKO DOSTUPNIH KOMPLEKSNIH DISTRIBUIRANIH SOFTVERSKIH SISTEMA
AN INTEGRATION PROBLEM OF HIGHLY AVAILABLE COMPLEX DISTRIBUTED SOFTWARE SYSTEMS

Nebojša Trninić - Schneider Electric DMS NS,
Lazar Stričević - FTN Novi Sad,
Miroslav Hajduković - FTN Novi Sad

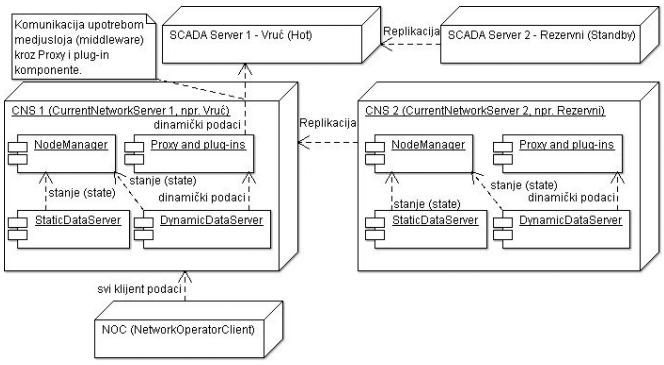
REZIME: Kompleksni računarski sistemi često obuhvataju kompleksnu koordinaciju između distribuiranih komponenti, koje su podložne otkazima i rekonfiguraciji u toku redovnog rada. Ovaj rad se bavi problemom nefunkcionisanja komunikacije između delova kompleksnog sistema, odnosno slučajevima kada trajanje komunikacije prevazilazi zadati vremenski okvir. Dato je jedno rešenje uvođenjem softverskog međusloja. U slučaju predugog trajanja komunikacije, ovaj međusloj omogućava aplikaciji da nastavi izvršavanje i reaguje na ovako nastalu grešku. Opisan je dizajn međusloja i na primeru je testirana funkcionalnost ovakvog pristupa. Analizom dobijenih rezultata je pokazano koje su granice ovakvog pristupa i u kojim pravcima je moguće tražiti poboljšanja.
KLJUČNE REČI: distribuirani sistemi, dostupnost, srednji sloj,prekoračenje vremenskog okvira
ABSTRACT: Complex computer systems often imply complex coordination of distributed components that are prone to failures and reconfiguration in the course of their regular operation. This paper describes a communication problem, when communication takes more time than allowed. One solution to the problem is given by introducing an additional middleware layer. In the case of communication timeout, this middleware allows the application to continue its operation and promptly react to this type of error. Middleware design is presented and in a test example functionality of this approach is tested. The analysis of the results shows the limits of this approach and the directions for further research.
KEY WORDS: distributed systems, availability, middleware, timeout.

1. UVOD

Kompleksni sistemi menjaju tradicionalnu sliku homogenih distribuiranih sistema, gde se aplikacije specifične za datu oblast posebno projektuju i razvijaju na specifičnim platformama i međusloju. Primer za ovo su grid aplikacije, mobilne mrežne aplikacije, poslovni sistemi ili senzorske mreže. Delovi ovakvih sistema su dinamički povezani kako bi se stvorile kompleksne međusobno povezane strukture, često nazivane "sistemi sistema". Oni se često izvode u obliku delova softvera na umreženim računarima, koji komuniciraju i koordinišu svoje aktivnosti razmenom poruka [1].

Iako postoje mnogobrojni izazovi u projektovanju kompleksnih distribuiranih sistema, ovaj rad se bavi problemima integracije njegovih delova, odnosno, sposobnošću komponenti sistema da se povežu i međusobno razmenjuju podatke.

Problem na koji će se posebno obratiti pažnja je slučaj kada komunikacija između dva sistema traje duže nego što dopuštaju zadata ograničenja. Kompleksni sistem koji je uzet kao primer za demonstraciju ovog problema i njegovog rešenja je sistem za upravljanje distribucijom (*Distribution Management System* - DMS) električne energije [2], koji je prikazan na slici 1. Ovaj sistem se sastoji od dela za nadzor i prikupljanje podataka (*Supervisory Control and Data Acquisition* - *SCADA*), sistema servera za automatsko upravljanje distribucijom (*Current Network Server* - *CNS*) i sistema klijenata za podršku nodzornom osoblju, tj. dispečerima (*Network Operator Client* - *NOC*).



Slika 1. *Primer kompleksnog visoko dostupnog distribuiranog sistema: DMS*

Tokom normalnog rada, SCADA prikuplja podatke iz elektroenergetskog sistema (EES) određenom dinamikom. Vrednosti merenja koje SCADA prikuplja se, zajedno sa ostalim promenama stanja EES, nazivaju promenama dinamičkih podataka. Ove promene registruje SCADA, a zatim se prenose u CNS (automatski nadzorni deo DMS-a) da bi bile upotrebljene u automatskom nadzoru. Ove podatke takođe može da vidi i nadzorno osoblje (dispečeri) pomoću NOC klijenata i eventualno na njih reaguje. Potrebno je da promene dinamičkih podataka budu prosleđene CNS-u u zadatom vremenskom okviru, da bi DMS proračuni radili sa ažurnim ulaznim podacima, odnosno da bi dobijeni upravljački signali DMS-a bili primenljivi na aktuelno stanje elektroenergetskog sistema.

Regulacija elektroenergetskog sistema je vitalna za njegovo funkcionisanje i zbog toga je neophodno da se obezbedi da DMS bude visoko dostupan, drugim rečima da u definisanom vremenskom intervalu njegov period nefunkcionisanja bude manji od neke unapred zadate vrednosti. Ovo se između ostalog postiže i tako što se sistem učini otpornim na poremećaje (engl. *fault tolerant*) koji bi mogli da dovedu do otkaza sistema (engl. *system failure*). Jedan od načina da se postigne otpornost na poremećaje kompleksnog sistema kao što je DMS je hardverska redundantnost njegovih komponenti. U našem primeru koristi se dvostruka modularna redudantnost: pojedine važne komponente sistema su duplicirane tako da rade paralelno. Jedna kao aktivna ("vruća" - engl. *hot*) učestvuje u radu sistema, dok je druga tu kao rezervna (engl. *standby*). Ona preuzima aktivnu ulogu u slučaju da ranije aktivna komponenta otkaže. Da bi rezerva uspešno mogla da preuzme aktivnu ulogu, radi se pasivna ili "primarni-bekap" replikacija [3] podataka sa aktivne komponente: aktivna (primarna) komponenta učestvuje u radu sistema, a njeno stanje se povremeno prebacuje na rezervnu (bekap) komponentu. Ovo omogućava da sistem nastavi da radi uprkos otkazu komponente, a događaj kada rezervna komponenta postaje aktivna se naziva fejlover (engl. failover). Kritične komponente DMS sistema koje se na ovaj način udvajaju su SCADA i CNS serveri, kao i mrežne veze između njih. Na slici 1 je prikazan DMS sa po dva SCADA i CNS sistema.

2. PROBLEM INTEGRACIJE

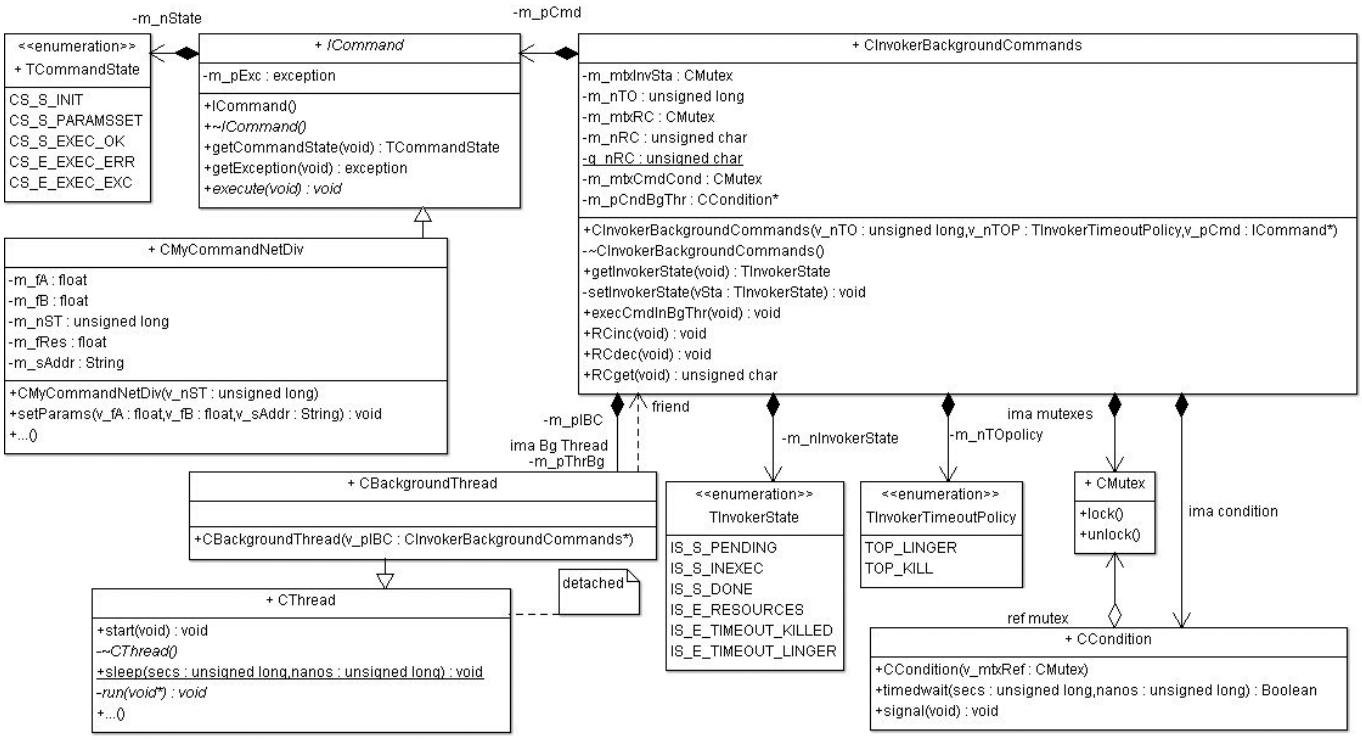
Budući da postoje SCADA sistemi različitih proizvođača, koji imaju različite osobine i koriste različite komunikacione protokole [4], potrebno je prilagoditi ostatak DMS-a (CNS server) kako bi mogao da komunicira sa datim SCADA sistemom. Ovaj postupak se naziva **integracija** SCADA-e i DMS-a. Deo CNS servera koji komunicira sa SCADA-om tako što poznaje osobine datog SCADA sistema, naziva se proksi (engl. *proxy*).

Za funkcionisanje DMS-a ključno je da podaci koje SCADA prikupi stignu do CNS servera, kao i da upravljački signali stižu od CNS-a do SCADA-e u okviru zadatog vremenskog ograničenja. Zbog toga je neophodno da postoji pouzdana, visoko dostupna komunikacija među njima.

Problem koji se javlja u komunikaciji između proksi komponente i SCADA-e je da odgovor SCADA-e na pozive CNS-a u komunikaciji ponekad stižu kasno, odnosno vreme potrebno da se dobije odziv ponekad prevazilazi ograničenje. Budući da je ova komunikacija najčešće sinhrona, zastoji u komunikaciji uzrokuju "zaglavljivanje" CNS-servera. Ovakvi zastoji koji traju duže nego što je dozvoljeno su neprihvatljivi, jer ugrožavaju rad čitavog DMS-a.

Na žalost često se dešava da nije moguće definisati i ograničiti vreme trajanja izvršavanja poziva srednjeg sloja, npr. ako se koristi TCP [5], [6], [7], [8], [9], [10], [11]. Tako se dolazi u situaciju da je poziv srednjeg sloja zaglavljen i zbog toga nije moguće uraditi fejlover ili isključenje servera koji komunicira sa eksternim sistemom. Da bi se ovaj problem prevazišao na generički način, kao jedno rešenje, napravljen je IBC šablon (engl. *Invoker of Background Commands pattern*) čiji je dijagram klasa prikazan na slici 2.

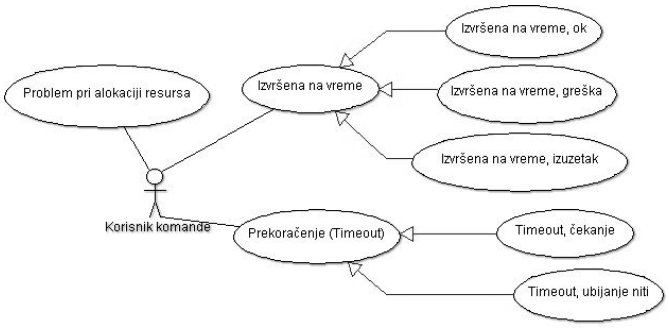
IBC je sloj softvera koji se umeće između razvijenog softvera i srednjeg sloja koji za integraciju nudi eksterni sistem. Korišćenje IBC-ase svodi na izvođenje dodatnih klasa iz klase ICommand i implementaciju metoda interfejsa



Slika 2. *Dijagram klasa koji ilustruje IBC rešenje*

specijalizovane komande koja enkapsulira originalni pozive srednjeg sloja, npr. metoda: get/setParams(), execute() ili getResults(). Enkapsuliraju se metode srednjeg sloja, koje ne podržavaju definisanje vremena dozvoljenog za izvršavanje poziva i dobijaju željene funkcionalnosti koje poštuju definisano vreme odvojeno za njihovo izvršavanje. Korisnik instancira objekat izvedene klase komande, u primeru je to CMyCommandNetDiv i jedan objekat klase CInvokerBackgroundCommands. Objekat komande se prosleđuje invokeruprilikom konstruisanja. Kasnije se poziva metoda za izvršenje definisane komande u pozadini execCmdInBgThr(). Invoker objekat će tada instancirati dodatnu nit koja će izvršiti komandu, dok paralelno glavna nit čeka tačno određeno vreme na izvršenje ove komande [12], [13]. Obrađeni su razni slučajevi korišćenja (engl. *use cases*) koji mogu da se dese pri ovim aktivnostima (slika 3):

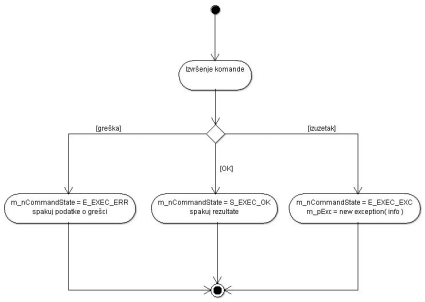
- problemi sa alokacijom resursa,
- komanda izvršena na vreme, sve je u redu,
- komanda izvršena na vreme, došlo je do greške (nešto očekivano i nešto na čega postoji odgovor),
- komanda izvršena na vreme, došlo je do izuzetka (engl. *exception*, nešto neočekivano),
- komanda nije izvršena na vreme, ostavljena da radi u pozadinskoj niti (engl. *linger*),
- komanda nije izvršena na vreme, pozadinska nit ubijena od strane glavne, u trenutku kad je u glavnoj niti isteklo vremensko ograničenje određeno za izvršenje komande (engl. *kill*).



Slika 3. *IBC dijagrami slučajeva korišćenja*

Detaljna analiza komandi i mogućih scenarija izvršavanja komandi, kao i rezultata komandi je rezultovala skupom stanja koje komanda može da ima (slika 2. TCommandState) i dijagramom aktivnosti komande (slika 4). Na slici 2. se može videti atribut CInvokerBackgroundCommands::v_nTO, koji je *timeout* parametar koji se zadaje prilikom konstruisanja objekta. Predstavlja željeno vremensko ograničenje za dužinu trajanja izvršenja komande. U slučaju da komanda vremenski traje duže nego što je ovim parametrom definisano, smatra se da je u pitanju *timeout*, koji je greška i glavni program nastavlja da radi, dok pozadinska nit (Bg Thread) biva ubijena ili se ostavlja da čeka, tj. radi dokle god se izvršenje komande ne završi i tada se oslobađaju svi zauzeti resursi.

ICommand::execute() treba da se štiti od izuzetaka (engl. *exceptions*), tj. da ih hvata i pakuje u svoj atribut.



Slika 4. *Dijagram aktivnosti komande*

Rezultat analize ponašanja glavne i pozadinske niti, u kojoj se izvršava komanda, je rezultovala slikom 5. na kojoj je dijagram aktivnosti ove dve niti. U pozadinskoj niti postoji aktivnost označena kao "Pripremne radnje pozadinske niti", koja podrazumeva da pozadinska nit kreće da radi. Izvrši se RCinc() da se poveća brojač referenci objekta klase CInvokerBackgroundCommands, koji postaje 2, tj. RC == 2. Zatim se pozadinska nit zaključa, a zatim otključa *mutex* objekta klase CInvokerBackgroundCommands (m_pIBC->m_mtxCmdCond) da bi se sinhronizovala sa izvršenjem glavne niti i da se izvršavanje nastavi, tek kada glavna nit bude u timedwait().

Postoje standardni protokoli koji omogućavaju rešenje ovog problema. Primer za to su protokoli iz familije *Fieldbus* [14].

Takođe, *Microsoft .NET framework 4* obezbeđuje klasu [15], čiji deo, vezan za asinhrono izvršavanje, predstavlja jedno rešenje, koje je slično rešenju prikazanom u ovom radu.

Funkcionalnost vremenskog ograničenja izvršavanja funkcija može da nudi i operativni sistem. Primer ovakvog rešenja je koncept rezervacije resursa, implementiran na operativnom sistemu MINIX3 [16].

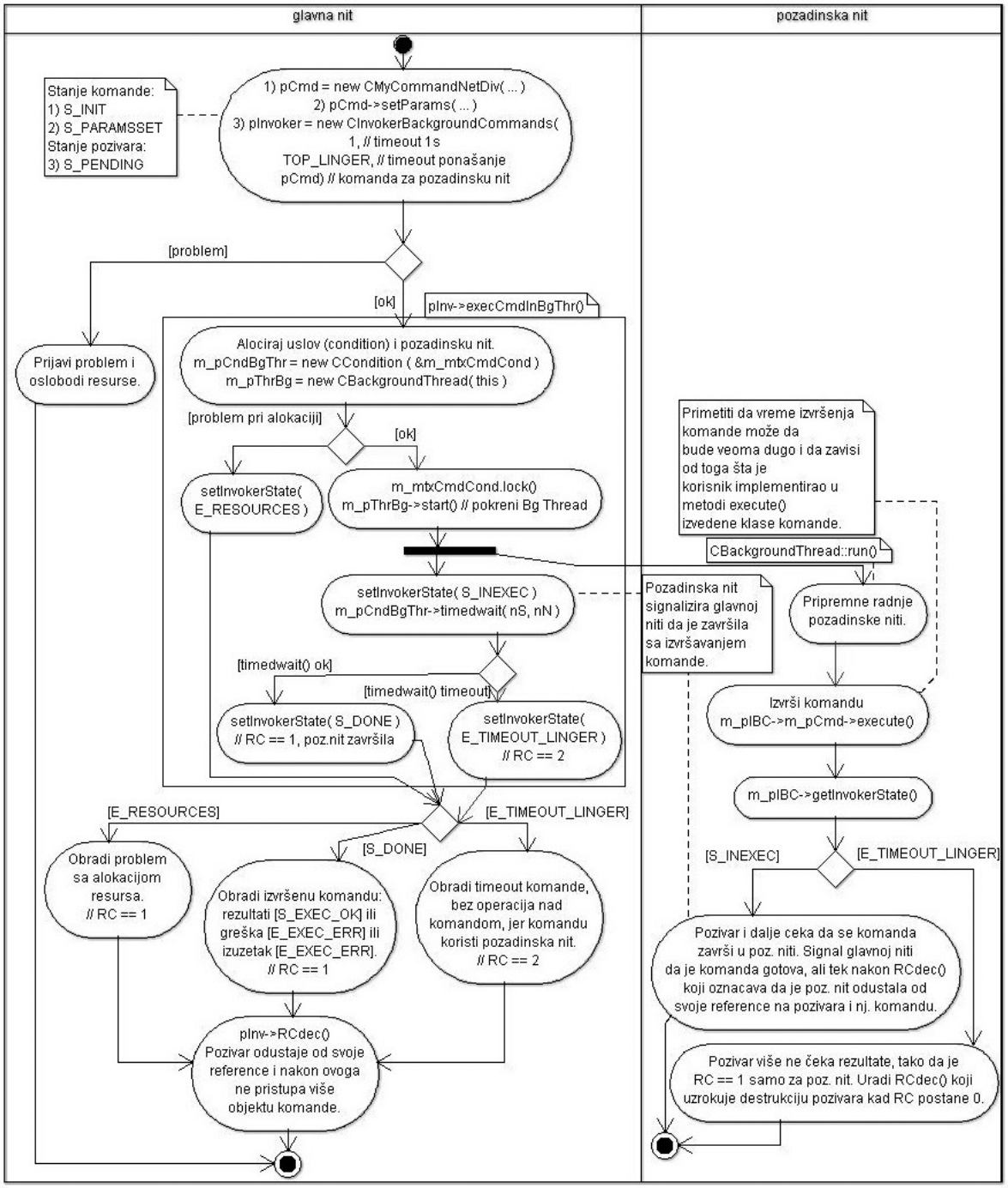
Implementacija vremenskih ograničenja u računarskoj komunikaciji se često pokaže potrebnom, iako u prvom momentu ne mora da bude deo osnovnog skupa funcionalnosti nekog komunikacionog protokola. Na primer, protokol TFTP u prvim verzijama ([17] iz 1992.) nije podržavao vremenska ograničenja, a u kasnijim verzijama su ona dodata ([18] iz 1998.).

Jedan od primera upotrebe koncepta opisanog u ovom radu bi bio dodavanje funkcionalnosti vremenskog ograničenja na komunikacione protokole koji je nemaju.

3. TEST PRIMER

Primer aplikacije napravljene za proveru predloženog rešenja su C++/C klijent/server aplikacije sa primitivnom realizacijom softverskog međusloja. U ovom primeru klijent aplikacija od servera zahteva deljenje dva broja tipa float. Komunikacija se obavlja kroz običnu TCP/IP vezu. Klijent serveru šalje dva float broja i vreme spavanja u sekundama, server ih podeli, sačeka određeni broj sekundi i rezultat koji je takođe float vraća klijentu.

Pretpostavlja se da je zadatak da se implementira rešenje bez korišćenja funkcije select() ili poll(), koje se inače koriste za prevazilaženje problema blokiranja poziva kod aplikacija koje koriste *socket* API za TCP/IP povezivanje. Takođe ne treba da se koriste ni opcije *socketa* vezane za funkciju setsockopt(): SO_SNDTIMEO ili SO_RCVTIMEO. socket(), connect(), send(), recv() i close() koje klijent koristi se enkapsuliraju izvedenom komandom u njenoj execute() metodi i klijent pri pozivu definiše vremensko ograničenje za izvršavanje komande. U kodu



Slika 5. *Dijagram aktivnosti glavne i pozadinske niti*

postoji i podrška za slučajeve kada dolazi do greške ili izuzetka, kako bi se u testovima proverilo i potvrdilo željeno ponašanje i u ovim slučajevima.

Da bi se podržao primer u kome istekne vreme određeno za izvršenje komande (engl. *timeout*), server aplikacija prima parametar koliko dugo nakon deljenja brojeva da čeka, pre nego što klijentu pošalje rezultate. Ovim se postiže da programski može da se simulira dugo trajanje izvršenja komande, bez fizičkog prekida mrežne veze. Npr. klijent će za *timeout* usvojiti vreme od jedne sekunde, a server će biti namešten da računa deset sekundi. Ovim će izvršenje komande trajati duže od vremena koje je klijent odvojio za njeno izvršavanje, čime je urađen test prekoračenja vremena odvojenog za izvršavanje komande.

Izvršavanjem klijent i server test aplikacija, na primeru je pokazano da prikazani IBC mehanizam, rešava zadatke prikazane u scenarijima na *use case* dijagramu na slici 3.

Timeout policy kill nije testiran na primeru, jer *omnithread* API [19] koji je korišćen, ne obezbeđuje pozive za nasilno zaustavljanje niti, tako da je implementiran i testiran samo *timout policy linger*. U slučaju kada pozadinska nit ostaje zaglavljena duže vremena da izvršava komandu, bilo zbog poziva srednjeg sloja ili nekorektno napisane komande, može da dođe do situaci-

je da pozadinska nit zauzima računarske resurse sve vreme izvršavanja programa, što može da dovede do iscrpljivanja resursa. Rešavanje ovog problema će biti predmet daljih istraživanja.

4. ZAKLJUČAK

Predloženo IBC rešenje je provereno na test primeru koji čine klijent i server C++/C aplikacije. Ukoliko se iskoristi u aplikacijama kojima smeta dugačko zaglavljivanje niti, ono je generičko rešenje ovakvih problema i obezbeđuje da se glavna nit ne zaglavi duže nego što korisnik IBC mehanizma definiše. Rešenje nije ograničeno na C++, jer dati UML dijagrami mogu da se implementiraju u raznim jezicima. Potrebno je da jezik izabran za implementaciju obezbeđuje API za više nitno programiranje (engl. *multi threaded*) i mehanizme za sinhronizaciju niti tipa *mutex* i *condition*.

S obzirom da je u implementaciji korištena *omnithread* biblioteka, koja ne podržava ubijanje pozadinske niti, dalji pravac rada može da bude istraživanje i implementacija *policy kill*, upotrebom neke *multi threaded* biblioteke koja ovo podržava (POSIX *threads* pthread_kill() ili C++11).

Sa stanovišta performansi i efikasnije upotrebe resursa, tj. ograničenja upotrebe resursa, dalje je moguće IBC mehanizam razrađivati tako da se za pozadinske niti, koje izvršavaju komande, ne instanciraju uvek nove niti u trenutku kad korisnik izda zahtev, već da se iskoristi skup niti (engl. *thread pool*) i da se po potrebi niti iz njega uzimaju i u njega vraćaju nakon izvršenja komande.

5. LITERATURA

[1] Coulouris G., Dollimore J., Kindberg T., and Blair G. Distributed Systems: Concepts and Design (5th ed.). Addison-Wesley Publishing Company, 2011.

[2] Popovic, D.S., "Power Application - A Cherry on the Top of the DMS Cake", *DA/DSM DistribuTECH Europe*, 2000.

[3] Wiesmann, M.; Pedone, F.; Schiper, A.; Kemme, B.; Alonso, G.; , "Understanding replication in databases and distributed systems," *Distributed Computing Systems, 2000. Proceedings. 20th International Conference on* , vol., no., pp.464-474, 2000.

[4] Kalapatapu, R.; Scada Protocols and Communication Trends, *ISA EXPO 2004*, 2004.

[5] Postel, J.; RFC 793 "Transmission Control Protocol", Internet Engineering Task Force, IETF, 1981.

[6] Libman, L. and Orda, A.; "Optimal retrial and timeout strategies for accessing network re
sources," *IEEE/ACM Trans. Netw.* (10:4), pp. 551--564, 2002.

[7] Scott, M. L. ; "Non-blocking timeout in scalable queue-based spin locks", *Proceedings of the twenty-first annual symposium on Principles of distributed computing*, pp. 31--40, 2002.

[8] Ma, L., Barner, K. E. and Arce, G. R. "Statistical analysis of TCP's retransmission timeout algorithm," *IEEE/ACM Trans. Netw.* (14:2), pp. 383--396, 2006.

[9] Kesselman, A. and Mansour, Y. "Optimizing TCP retransmission timeout", *Proceedings of the 4th international conference on Networking - Volume Part II*, pp. 133--140, 2005.

[10] Ludwig, R. and Katz, R. H. "The Eifel algorithm: making TCP robust against spurious retransmissions," *SIGCOMM Comput. Commun. Rev.* (30:1), pp. 30--36, 2000.

[11] Gurtov, A. "Responding to spurious timeouts in TCP"', *In Proc. of IEEE INFOCOM'03*, 2003.

[12] Barney, B. "Introduction to Parallel Computing", Lawrence Livermore National Laboratory, https://computing.llnl.gov/tutorials/parallel_comp/ , preuzeto decembra 2012.

[13] Barney, B. "POSIX Threads Programming", Lawrence Livermore National Laboratory, https://computing.llnl.gov/tutorials/pthreads/ , preuzeto decembra 2012.

[14] Maynard, C. A. "Fieldbus Tutorial", Curtin University of Technology, URL: http://kernow.curtin.edu.au/www/Fieldbus/fieldbus.htm , preuzeto decembra 2012.

[15] Microsoft Corporation, "Task Class", URL: http://msdn.microsoft.com/en-us/library/system.threading.tasks.task.aspx , preuzeto decembra 2012.

[16] Mancina, A., Faggioli, D., Lipari, G., Herder, J. N., Gras, B., and Tanenbaum, A. S. "Enhancing a Dependable Multiserver Operating System with Temporal Protection via Resource Reservations", *Real-Time Systems*, 43(2):177–210, 2009.

[17] Sollins, K. "The TFTP Protocol (Revision 2)"(1350), Internet Engineering Task Force, IETF, RFC 1350, 1992.

[18] Malkin, G. and Harkin, A. "TFTP Timeout Interval and Transfer Size Options"(2349), Internet Engineering Task Force, IETF, RFC 2349, 1998.

[19] Richardson, T. "The OMNI Thread Abstraction", ATT Laboratories Cambridge, Revised November 2001, 2001.

Nebojša Trninić
nebojsa.trninic@schneider-electric-dms.com
Zaposlen u Schneider Electric DMS NS u Novom Sadu.
Oblasti interesovanja: integracije, računarske mreže, distribuirani
računarski sistemi

Lazar Stričević
lucky@uns.ac.rs
Zaposlen na Katedri za primenjene računarske nauke Fakulteta tehničkih nauka Univerziteta u Novom Sadu.
Oblasti interesovanja: arhitektura računara, računarske mreže, distribuirani računarski sistemi

Miroslav Hajdukovic
hajduk@uns.ac.rs
Zaposlen na Katedri za primenjene računarske nauke Fakulteta tehničkih nauka Univerziteta u Novom Sadu.
Oblasti interesovanja: računarske nauke