PRIKUPLJANJE DISTRIBUIRANIH PODATAKA POMOĆU SMART KARTICA I XML-A U SLUČAJU NEDOSTATKA ILI OTKAZA RAČUNARSKE MREŽE

Aleksandar Marković, *Vojnotehnička akademija, Smer informatike*
Miloš Merdžanović, *Vojnotehnička akademija, Katedra RTI*
Milan Knol, *Poslovno informacioni sistemi doo.*

Sadržaj – *U radu su predstavljeni načini prikupljanja distribuiranih podataka u slučaju da računarska mreža nije na raspolaganju iz bilo kojih razloga. Pored toga, prikazana je konkretna implementacija jednog takvog rešenja na primeru informacionog sistema studentskih restorana. Rešenje se zasniva na primeni java smart kartica i XML tehnologije. Naime, svaka smart kartica sadrži informacije o tome koliko student ima obroka na raspolaganju. U restoranima se studentima programski umanjuje količina obroka na kartici, a evidentiranje se vrši upisivanjem podataka u XML fajl. U okviru rešenja predviđena je i serverska komponenta koja iz čita podatke iz XML fajlova i upisuje ih u bazu podataka. Prednost ovakvog rešenja je to što postojanje mreže ničim ne uslovljava razvoj informacionog sistema i ne utiče na njegove performanse.*

1. UVOD

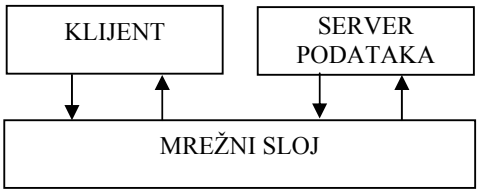
Čest je slučaj da razne organizacije i ustanove nisu u mogućnosti da uspostave računarsku mrežu između svih svojih organizacionih jedinica. Ukoliko one poseduju informacioni sistem, javlja se problem prikupljanja podataka na jednom mestu i upisivanje u centralnu bazu podataka. Mnoge kompanije nude svoja, komercijalna, rešenja ovog problema, ali ona su uglavnom zavisna od postojanja računarske mreže, bilo lokalne, bilo Interneta.

U radu se predlaže rešenje ovog problema koje ne zahteva postojanje računarske mreže. Daje se i konkretna implementacija rešenja na primeru informacionog sistema studentskih restorana. Ovaj informacioni sistem je realizovan za Ustanovu Studentski centar Beograd i opisan je u radu [1]. Kao neophodan uslov bio je da se podaci o realizovanim obrocima iz restorana prenesu u centralnu bazu podataka bez postojanja mreže. Naime, neki restorani su povezani u lokalnu mrežu, a neki nisu. Zato je bilo neophodno implementirati fleksibilno rešenje koje može da iskoristi, ali ne zahteva mrežu računara.

U sledećem odeljku se opisuje problem i moguća rešenja, a u 3. odeljku se opisuje arhitektura predloženog rešenja. Konkretna implementacija se obrazlaže u 4. odeljku, a na kraju se daje zaključak.

2. OPIS PROBLEMA I MOGUĆA REŠENJA

U ovom odeljku se obrađuje problem prenosa distribuiranih podataka u informacionim sistemima u slučaju kada je mreža na kojoj je sistem zasnovan nestabilna ili kada je uopšte nema. Arhitektura jednog informacionog sistema najčešće podrazumeva troslojnu klijent server arhitekturu [2]. Radi pojednostavljenja objašnjenja mi ćemo ovaj model uprostiti i pretpostavićemo da aplikacionog sloja nema, tako da se sistem sastoji samo od klijentskog i serverskog softvera gde se odgovornost aplikacionog sloja raspodeljuje između klijenta i servera, što je prikazano na slici 2. Na istoj slici je predstavljeno i da se kompletna komunikacijija između klijenta i servera obavlja preko mreže.



Sl.1. *Dvoslojna Klijent – Server arhitektura sa mrežnim resursima na koje se oslanja*

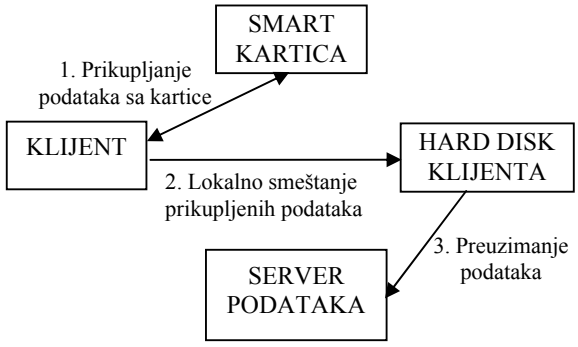
Raspored komponenti je takav da se klijentske i serverske aplikacije nalaze na posebnim računarima i da je za komunikaciju između njih potrebna mreža. Da bi klijent funkcionisao, često je potrebno da ima stalnu konekciju sa serverom kako bi korisniku obezbedio odgovarajuće informacije prilikom obavljanja transakcija u okviru sistema.

Jedno od rešenja da se izbegne stalna komunikacija sa serverom je upotreba smart kartica koje čuvaju sve neophodne podatke. Klijentski softver komunicira sa smart karticom prilikom obavljanja transakcije i od nje dobija sve potrebne podatke, pa stalna veza sa serverom nije neophodna. Sve što klijent treba da radi je da prikuplja podatke i da s vremena na vreme ažurira podatke na serveru koje je prikupio sa kartice. Međutim ovi podaci se ne mogu čuvati u radnoj memoriji računara na kome se klijent izvršava, jer klijent može da prikupi veliki broj informacija dok se ne uspostavi konekcija sa serverom, a u međuvremenu može doći i do neželjenog prekida rada računara čime bi svi podaci bili izgubljeni.

Međutim, mora se predvideti i situacija kada računarska mreža nije aktivna sve vreme ili uopšte ne postoji. U tom slučaju čuvanje informacija u radnoj memoriji ne bi imalo smisla. Prirodno rešenje je da se informacije smeštaju lokalno, u trajnoj memoriji (hard disk), dok se veza sa serverom ne uspostavi. Po uspostavljanju konekcije treba da se obave transakcije servera podataka, sa podacima koji su smešteni na lokalnom hard disku klijenta. Ovaj prenos može da se obavi tako što bi se klijent konektovao na server i izvršio transakciju sa serverom uz pomoć lokalno smeštenih podataka. Međutim ova varijanta zahteva da klijent bude pokrenut u trenutku kada server preuzima podatke.

Ali, opet se nameće pitanje šta treba uraditi u varijanti kada mreže uopšte nema. Zato je predloženo rešenje da klijent smešta podatke lokalno, a da ih server prikuplja po uspostavljanju konekcije. U varijanti bez mreže podaci bi se do servera podataka prenosili na nekom medijumu (disketa ili flash memorija) ili bi mogli da se pošalju elektronskom poštom.

Opisano rešenje predstavlja klasičan primer asinhrone perzistentne komunikacije između klijenta i servera [3], gde je prosleđivanje poruka (prikupljenih podataka) zagarantovano. Ovim pristupom klijent može prosleđivati podatke serveru čak i kada server nije pokrenut, a isto tako server može primati podatke od klijenta kada klijent nije pokrenut. Rešenje je fleksibilno i pokriva priličan broj slučajeva, a prikazano je na slici 2.



Sl. 2. *Prikaz predloženog rešenja*

Pošto sada server i klijent komuniciraju preko podataka uskladištenih na lokalnom računaru, postavlja se pitanje formata podataka. S obzirom na to da se jedan informacioni sistem može nalaziti na različitim platformama, kao prirodno rešenje nameće se upotreba nekog portabilnog formata. Takođe, format bi trebalo da nudi mogućnost da se lokalno podaci pregledaju i iz njih generišu izveštaji, implementiranjem posebne softverske komponente ili korišćenjem nekog standardnog alata. U informacionim sistemima se često sreće potreba da se podaci pregledaju na ovaj način i da se na osnovu njih kreiraju razni izveštaji.

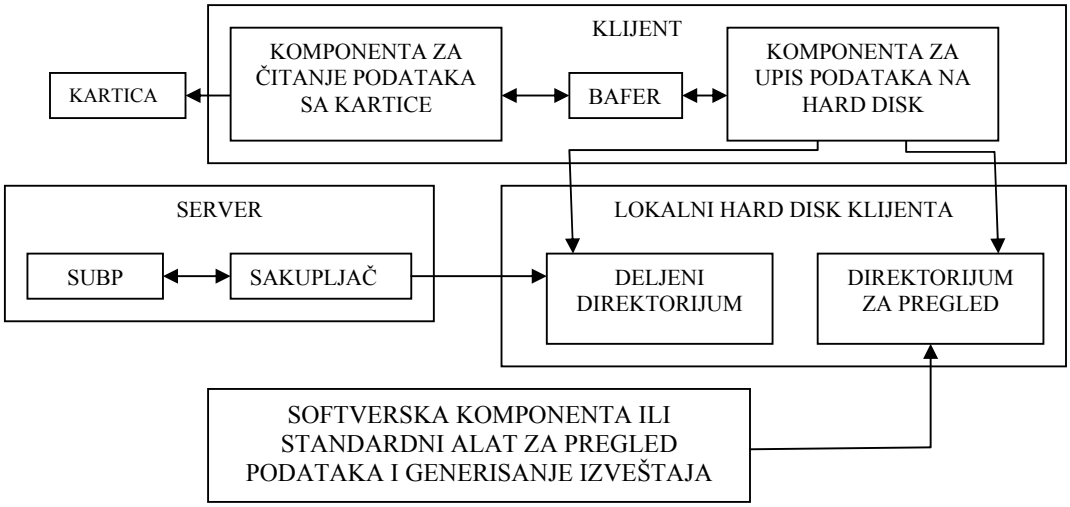
U zavisnosti od primene informacionog sistema može se desiti da se često čitaju podaci sa smart kartica. Stoga ovo može biti vremenski osetljiva operacija. S obzirom da se prikupljeni podaci smeštaju na lokalni hard disk, što može biti vremenski zahtevno potrebno je obezbediti da smeštanje podataka na hard disk ne blokira čitanje podataka sa kartice. Jedno od rešenja ovog problema je da postoje dve komponente unutar klijenta gde će jedna služiti za prikupljanje podataka sa kartice, a druga za smeštanje podataka na hard disk. Neophodno je da obe komponenete rade kao nezavisne niti koje će biti sinhronizovane preko bafera sa FIFO disciplinom pristupa, kao što je prikazano na slici 3 u komponenti klijenta. Odgovornost komponente za čitanje je da pročita podatke i da ih smesti u bafer (ova operacija ne bi trebalo nikada da blokira i brzo se odigrava). Veličina bafera ovde nije od velike važnosti jer će podaci u dogledno vreme iz njega biti upisani na hard disk. Komponenta koja upisuje podatke na hard disk će biti blokirana sve dok je bafer prazan, a kada podaci budu smešteni u bafer, ona će ih pokupiti.

Kao što je rečeno, razmena informacija u predloženom rešenju se vrši na nivou fajlova. Zato je bitno da se razmotri regulisanje veličine i imenovanje fajlova.

Ako sistem radi potpuno bez mreže onda veličina fajlova koji se prenose od klijentske do serverske mašine nije toliko bitna. Ali ako se transfer fajlova vrši preko mreže onda bi trebalo da veličina fajlova bude ograničena nekom vrednošću. Naime, ako se prenose veći fajlovi, biće potreban i duži vremenski period u kome je verovarnoća da dođe do kraha računarske mreže veća.

Za imenovanje fajlova postoje dve mogućnosti. Fajlovima se mogu zadavati imena koja bi se generisala slučajno, ali tako da je verovatnoća da dva fajla u sistemu imaju isto ime praktično nemoguća. Druga mogućnost je da se imena fajlovima zadaju smisleno, ali takođe da se izbegne kolizija imena u sistemu. Na primer, ime fajla može da se sastoji od imena računara na kom se nalazi, datuma kada je nastao i rednog broja fajla za taj dan. Problemi koji se opisuju nastaju u slučaju kada klijent smešta gotove fajlove u neki lokalni direktorijum za koji server zna. U ovoj varijanti klijent i server ne moraju da vrše sinhronizaciju na nivou fajlova, jer klijent dostavlja serveru gotov fajl u deljeni direktorijum koji server odatle preuzima. Međutim, postoji i mogućnost da klijent smešta podatke u jedan fajl iz koga bi server čitao podatke, a zatim bi pročitane podatke uklanjao iz tog fajla. Za ovo je potrebno ostvariti sinhronizaciju procesa na nivou fajla.

Server podataka se može realizovati na različite načine. Ono što treba izdvojiti je komponenta koja prikuplja podatke sa klijenata i smešta ih na definisano odredište na serveru (relaciona baza podataka ili neka druga softverska komponeneta za smeštanje podataka) i nju ovde nazivamo *Sakupljač*. U opštem slučaju, Sakupljač ne mora biti na istom računaru na kome je i server baza podataka i u tom slučaju se može posmatrati kao deo aplikacionog sloja. Zbog jednostavnosti sistema u ovom radu se posmatra kao komponenta sloja podataka.



Sl. 3. *Detaljan pregled komponenti sistema*

Funkcionalnost Sakupljača može biti jednostavna – da prikupi fajlove sa lokalnih računara na serverski računar, a može biti i složenija. U najprostojoj varijanti Sakupljač samo sakuplja podatke sa lokalnih hard diskova klijenata, a nakon toga briše te fajlove sa klijenata. Ako brisanje fajla ne uspe, onda se i kopija briše sa servera. U ovoj varijanti Sakupljač je odgovoran samo za prenos fajlova sa klijenata do servera.

Složenija varijanta je da Sakupljač vrši i sakupljanje podataka i njihovo pohranjivanje u bazu podataka. U ovoj varijanti prikupljanje podataka se radi na isti način, a razlika je u tome što Sakupljač pohranjuje preneti fajl u bazu. U varijanti bez mreže Sakupljač sakuplja fajlove iz nekog lokalnog direktorijuma u koji se podaci smeštaju i njima se pohranjuje baza podataka.

3. IMPLEMENTACIJA NA KONKRETNOM PRIMERU

Razmatrano rešenje je realizovano na primeru informacionog sistema studentskih restorana Ustanove Studentski centar u Beogradu. Ova ustanova u svom sastavu ima više restorana koji se nalaze u različitim delovima Beograda. Neki od tih restorana imaju pristup lokalnoj mreži Studentskog centra, ali postoje i oni koji trenutno nemaju tu mogućnost. Iz navedenog razloga, opisano rešenje je moguće primeniti na isti način, bez obzira na to da li restoran ima pristup mreži.

Ideja rešenja se zasniva na metodama koje su opisane u prethodnoj analizi. Projektovanje aplikacije je izvršeno objektno orijentisanom metodologijom na jeziku UML [4]. Kao format za razmenu podataka odabran je XML [5], a aplikacija je implementirana Java tehnologijom.

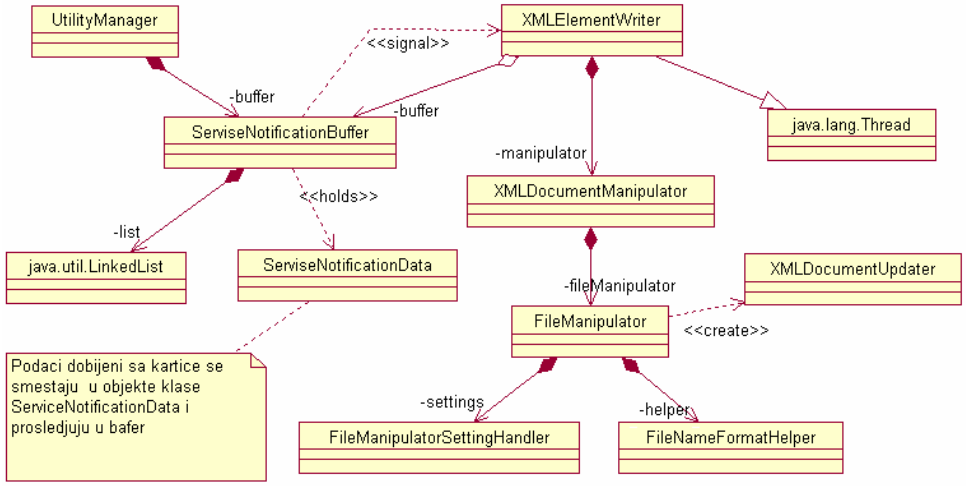
U analizi je navedeno da bi format podataka trebalo da bude neki portabilni format koji će ujedno obezbediti i lokalni pregled podataka sa nekim standardnim alatom. Za takav format je odabran XML jer uz pomoć XSLT-a nudi mogućnost prikaza podataka i generisanja izveštaja u standarnim alatima. Pored ovih pogodnosti, XML je format koji podržava i većina proizvođača baza podataka. U slučaju da mreža ne radi i da Sakupljač ne postoji, moguće je sa lokalnog serverskog računara pohraniti bazu podataka podacima iz XML fajlova uz pomoć alata koji dolaze u servere baza podataka. Na primer, SQL Server 2000 ima mogućnost jednostavnog pohranjivanja baze podataka direktno iz XML fajlova, kao i njihovo dodatno transformisanje pre pohranjivanja u bazu uz pomoć DTS - Data Transformation Services alatke[6].

Komponenta klijenta koja čita podatke sa kartice i komponenta koja ih upisuje na lokalni hard disk su implementirane kao niti u Javi (nasleđivanjem klase *Thread*). Java takođe nudi ugrađenu podršku za sinhronizaciju niti na nivou objekata (korišćenjem metoda *wait()* i *notify()* klase *Object* i korišćenjem ključne reči *synchronized* [7]).

Server podataka se sastoji od Sakupljača i od baze podataka. U varijanti koja je realizovana, Sakupljač prikuplja podatke sa klijenata na nivou fajlova. To znači da Sakupljač čuva listu direktorijuma klijenata u kojima se nalaze XML fajlovi sa podacima i kopira ih u definisanim vremenskim intervalima.

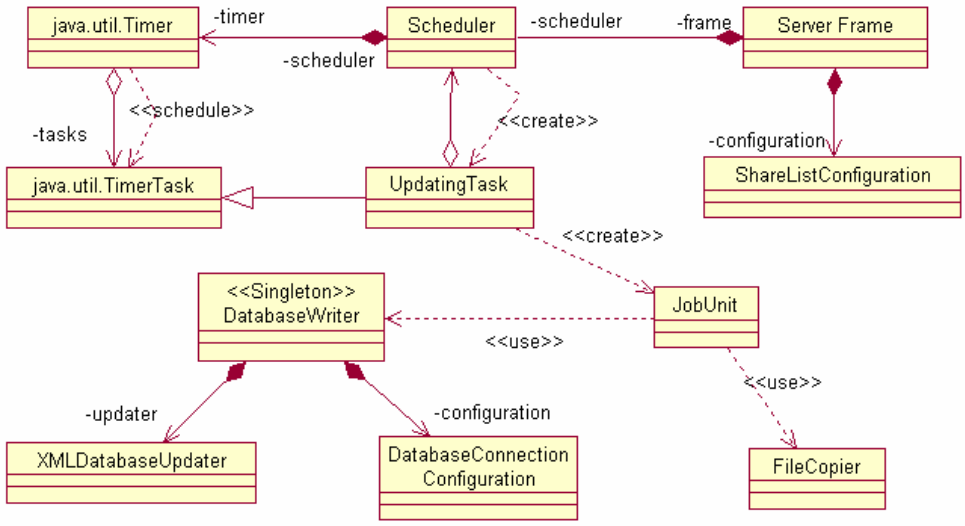
Sakupljač treba da bude transparentan u odnosu na bazu podataka, tj. promena implementacije baze podataka ne sme da zahteva ponovnu implementaciju Sakupljača. Korišćenjem programskog jezika Java i njenog standardnog interfejsa prema bazama podataka (JDBC – Java Database Connectivity) moguće je iz Java programa pristupati bazama podataka na uniforman način [7].

Na slici 4. je prikazan UML dijagram klasa komponente za smeštanje podataka prikupljenih sa kartice. Klasa *Utility-Manager* zajedno sa klasom *ServiceNotificationBuffer* predstavlja interfejs ove komponente prema ostatku sistema. *UtilityManager* se koristi za administrativne poslove ove komponente i za tu svrhu može kreirati dijalog interfejs koji može da se pozove iz ostatka sistema. U objekte klase *ServiceNotificationBuffer* se smeštaju podaci sa kartice i to radi nit sistema koja pristupa kartici. Klasa *XMLElementWriter* predstavlja posebnu nit koja čita podatke iz bafera i smešta ih u neki fajl. Bafer implementira klasa *ServiceNotificationBuffer* čije metode su *putElement()* i *getElement()*, synchronized metode. Metoda *putElement()* vrši obaveštavanje niti *XMLElementWriter*-a da je upisan objekat u bafer ako je nit blokirana. Metoda getElement blokira nit *XMLElementWriter*-a ako je bafer prazan. Za smeštanje podataka *ServiceNotificationBuffer* koristi jedan od standardnih kontejnera u Javi, klasu *LinkedList*.



Sl. 4. *UML dijagram klasa komponente klijenta koja upisuje podatke na lokalni hard disk*

Sakupljač je predstavljen dijagrammom klasa na slici 5. Odgovornost klase *ServerFrame* je korisnički interfejs za prikaz tekućih operacija Sakupljača, za administriranje Sakupljača i za eksplicitno pokretanje sakupljanja podataka. Klasa *Scheduler* upravlja radom Sakupljača uz pomoć standardnih Java API-ja *Timer* i *TimerTask*. Klasa *UpdatingTask* enkapsulira operacije koje Sakupljač treba da obavi na događaj tajmera ili na eksplicitni poziv korisnika. Ona kreira objekte klase *JobUnit* koji predstavljaju jednu transakciju Sakupljača. Transakcija se sastoji od kopiranja specificiranog fajla uz pomoć klase *FileCopier* i ažuriranja podataka u bazi uz pomoć klase *DatabaseWriter*. Klase *ShareListConfiguration* i *DatabaseConnectionConfiguration* enkapsuliraju konfiguracije lokacija sa kojih se podaci preuzimaju i parametre konekcije na bazu podataka, respektivno. Ove klase preuzimaju konfiguracione podatke iz konfiguracionih fajlova koji se zbog portabilnosti čuvaju kao XML fajlovi.



Sl. 5. *UML dijagram klasa Sakupljača*

4. ZAKLJUČAK

U radu su opisana moguća rešenja prenosa distribuiranih podataka od klijenta do servera u slučaju da postoji nepouzdana mreža ili da uopšte ne postoji. Takođe, prikazana je i implementacija jednog od mogućih rešenja na primeru prenosa podataka u informacionom sistemu studentskih restorana. Prednost opisanog rešenja je to što egzistencija računarske mreže ni u kom pogledu ne utiče na performanse sistema.

Prikazano rešenje je fleksibilno i može se primeniti u širokom spektru aplikacija. Ono ne zavisi od konkretnih podataka koji se prenose, niti od destinacije, tj. baze podataka u koju se smeštaju.

Na kasnije implementacije i eventualna unapređenja rešenja može uticati promena tehnologije i poboljšanja koja može doneti. U nekim situacijama će biti bolje da se primeni neki od drugačijih pristupa problemu koji su opisani u radu.

LITERATURA

[1] M. Merdžanović, V. Rakić, M. Knol, "Implementacija i primena Java smart kartica u informacionom sistemu studentskih restorana", *YUINFO 2005, Zbornik radova na CD-u*

[2] Eckerson, V. Wayne, Tree Tier Client/Server Architecture: Achieving Scalability, Perfomance And Efficiency In Client Server Applications, *Open Information Systems* 10, 1, January 1995.

[3] A. S. Tanenbaum and M. Vaan Steen, *Distributed Systems Principles And Paradigms*, New Jersey, Prentice – Hall, Inc., 2002.

[4] G. Booch, J. Rumbaugh, I. Jacobson, *The Unified Modeling Language User Guide*, Addison-Wesley, 1999.

[5] A. Bergholz, "Extending Your Markup: An XML Tutorial," *IEEE Internet Computing*, Vol. 4, No. 4, July/August 2000, pp 74-79.

[6] K. Henderson, "*Using SQL Server's XML Support"*, http://www.informit.com

[7] J2SDK 1.4.2 Documentation, http://java.sun.com/j2se/1.4.2/docs/index.html

Abstract – This paper presents different ways for gathering distributed data when computer network is not available. Also, a particular implementation of such solution is presented on students' restaurants information system. This solution is based on java smart cards and XML technology. Information about how many meals a student has is placed on each smart card. Meals amounts on card are reduced in restaurants and data about that transaction are stored in an XML file. Also, a server component reads data from XML file and stores them in a database. Advantage of this solution is that it does not matter whether computer network exists or not, so the development of information system is independent of network resources.

GATHERING DISTRIBUTED DATA USING SMART CARDS AND XML IN ABSENCE OR BREAKDOWN OF COMPUTER NETWORK

Aleksandar Marković, Miloš Merdžanović, Milan Knol