Sjemenarstvo 16(99)5

UDK: 378.663(045)=862
Izvorni znanstveni rad

OBJEKTNI PRISTUP MODELIRANJU RAZVOJA I ODRŽAVANJA SIGURNOSNOG KRIPTOGRAFSKOG MODELA

K. ANTOLIŠ[1] i V. GRBAVAC[2]

[1] GS OS RH HVU Učilište hrvatske kopnene vojske, Zagreb

[2] Agronomski fakultet Sveučilišta u Zagrebu
Faculty of Agriculture University of Zagreb

SAŽETAK

Temeljem prethodno postignutog rezultata utemeljenog na objektno orijentiranom dizajnu, dolazi red na integraciju svih elemenata u jednu funkcionalnu cjelinu. Bitan čimbenik pri donošenju odluka o arhitekturi sustava bila je mogućnost proširenja i dodavanja novih performansi. Upravo će se objektno orijentirana strategija objedinjavanja pojedinih funkcija i svojstava u jedinstvene entitete, odnosno koncept objektno orijentiranog razvoja, u radu pokazati presudnom, jer sada ćemo uzeti u obzir konkretne primjere novih potencijalnih i logičkih zahtjeva koji se mogu postaviti pred naš sustav i demonstrirati funkcionalnost ovako stvorene strukture modela školske ploče i to posebice sa stajališta održavanja ovakvog jednog sustava razvijenog objektno orijentiranim pristupom.

1. UVOD

Istraživači s područja umjetne inteligencije ulažu silne napore kako bi unaprijedili razumijevanje ljudskih misaonih procesa. Aktivnost u ovom polju često uključuje konstruiranje inteligentnih sustava koji oponašaju određene aspekte ljudskog ponašanja. Erman, Lark i Hayes-Roth ističu da se "inteligentni sustavi razlikuju od konvencionalnih u nizu karakteristika, od kojih nisu sve uvijek prisutne:

- oni teže prema ciljevima koji se mijenjaju u vremenu,
- oni objedinjuju, koriste i utvrđuju znanje,
- oni upotrebljavaju raznolike, ad hoc podsustave koji obuhvaćaju mnoštvo izabranih metoda,
- oni interaktivno djeluju s korisnicima i drugim sustavima,
- oni sami raspodijeljuju svoje resurse i pažnju."

Bilo koje narečeno svojstvo dovoljno je zahtjevno da učini izradu inteligentnih sustava vrlo teškom zadaćom. Mi ćemo se sada pozabaviti samim razvojem našeg sustava rješavanja kriptograma.

2. INTEGRACIJA SUSTAVA ŠKOLSKE PLOČE

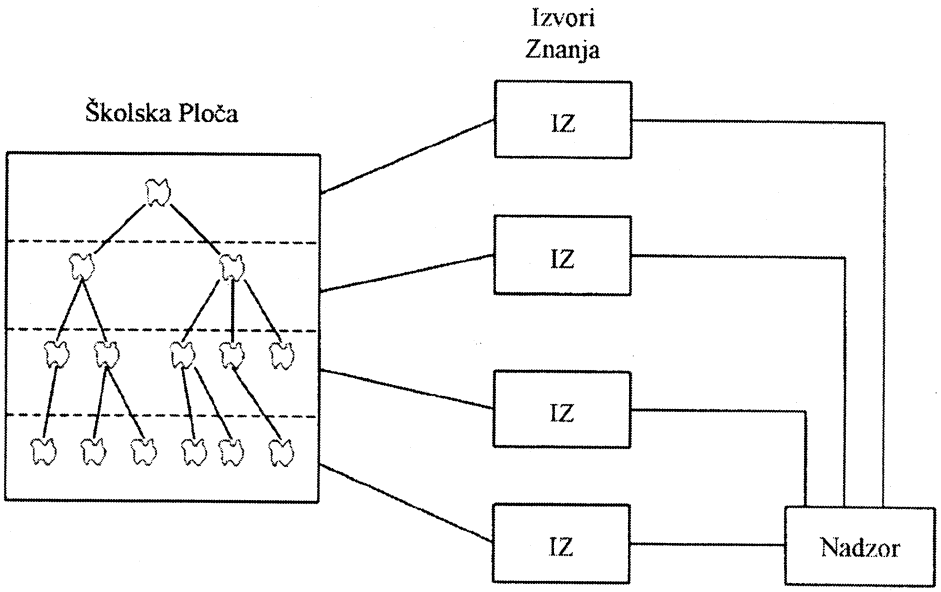
Nakon što smo definirali ključna svojstva u našem području, možemo nastaviti s njihovim objedinjavanjem kako bismo dobili potpunu aplikaciju. Nastaviti ćemo tako da implementiramo i testiramo vertikalni izvadak iz naše arhitekture, a onda ćemo dovršiti sustav mehanizam po mehanizam.

2.1. Integriranje najviših objekata

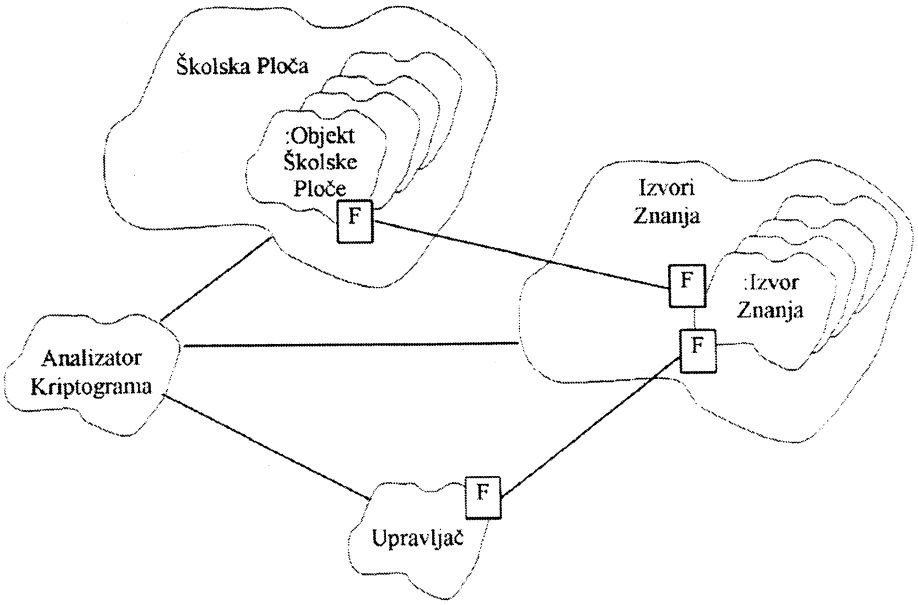
Ilustracija jedanog objektnog dijagram prikazana slikom 2. obuhvaća naš dizajn najviših objekata u sustavu, a koji odgovara strukturi generičnog sustava (kojem parametre objekata treba unijeti izvana) školske ploče na slici 1. Na slici 2, prikazujemo fizički sadržaj objekata školske ploče skupom *ŠkolskaPloča*, a izvora znanja skupom *IzvoriZnanja*, koristeći stenografski stil identičan onome koji pokazuje podrazrede.

U ovom dijagramu, uvodimo objekt jednog novog razreda kojeg ćemo nazvati *AnalizatorKriptograma*. Namjena ovog razreda jest da posluži kao jedna cjelina koja će obuhvaćati školsku ploču, izvore znanja i upravljač. Na taj način, naša bi aplikacija omogućila više objekata ovog razreda, tako da bismo u pogonu mogli istodobno imati i nekoliko školskih ploča.

Slika 1. Sustav Školske Ploče.



Slika 2. Objektni Dijagram Analize Kriptograma.



Definirati ćemo dvije temeljne operacije za ovaj razred:

- *resetiraj* Ponovno pokreni školsku ploču.
- *odgonetni* Riješi zadani kriptogram.

Djelovanje koje zahtijevamo kod konstrukcije ovog razreda jest stvaranje ovisnosti između školske ploče i njenih izvora znanja, te između izvora znanja i upravljača. Metoda resetiranja je slična po tome što se ove veze jednostavno resetiraju dok se školska ploča, izvori znanja i upravljač vraćaju natrag u stabilno početno stanje.

Premda ih ovdje nećemo detaljno prikazati, ulazne i izlazne vrijednosti operacije *odgonetni* uključuju jedan niz, pomoću kojeg dostavljamo šifrirani tekst koji treba biti riješen. Na taj način, osnova našeg programa postaje upravno banalno jednostavna, kao što je to uobičajeno u dobro dizajniranim objektno orijentiranim sustavima:

```
char* riješiProblem(char* šifriranitekst)
{
   AnalizatorKriptograma analizatorKriptograma;
   return analizatorKriptograma.odgonetni(šifriranitekst);
}
```

Implementacija operacije *odgonetni* je, što nije neočekivano, malo složenija. U biti, prvo moramo pozvati operaciju *utvrdiProblem* kako bismo

zadani problem postavili na školsku ploču. Zatim moramo pokrenuti izvore znanja skretanjem njihove pažnje na taj novi problem. I na kraju, moramo kružiti na način da uputimo upravljač da obradi sljedeću pomoć u svakom novom prolazu, sve dok se problem ne riješi ili dok svi izvori znanja ne zaglave. Mogli bismo upotrijebiti jedan interakcijski ili objektni dijagram za prikaz ovog toka kontrole, iako djelići C++ koda jednako dobro služe u slučaju tako jednostavnog algoritma:

```
ŠkolskaPloča.utvrdiProblem();
IzvoriZnanja.resetiraj();
while (!Upravljač.jeRiješeno() || Upravljač.nesposobanZaNastavak())
   Upravljač.obradiSljedećuPomoć();
if (ŠkolskaPloča.jeRiješeno())
   return ŠkolskaPloča.vratiRješenje();
```

Kao dio našeg razvoja, bilo bi vrlo uputno dovršiti dovoljno relevantnih mehanizama i svojstava arhitekture kako bismo mogli dogotoviti ovaj algoritam te istog izvršiti. Iako bi u ovom trenutku imala minimalnu funkcionalnost, njegova implementacija pomoću vertikalnog izreza kroz arhitekturu primorala bi nas da potvrdimo određene ključne arhitekturne odluke kao valjane.

U nastavku, razmotrimo dvije primarne operacije korištene u *odgonetni*, a to su *utvrdiProblem* i *vratiRješenje*. Operacija *utvrdiProblem* je osobito interesantna, pošto mora generirati cijeli niz objekata školske ploče. U formi jednostavnog teksta, naš je algoritam sljedeći:

```
odvoji sve čelne i završne razmake iz niza
vrati se ako je rezultirajući niz prazan
stvori objekt rečenice
priključi rečenicu školskoj ploču
kreiraj objekt riječi (to će biti prva riječ slijeva u rečenici)
priključi riječ ploči
priključi riječ rečenici
za svaki znak u nizu, slijeva na desno
   ako je znak razmak
      postavi da je trenutna riječ odsada prethodna riječ
         kreiraj objekt riječi
         priključi riječ ploči
         priključi riječ rečenici
      inače
         kreiraj objekt šifriranog slova
         priključi slovo ploči
         priključi slovo riječi
```

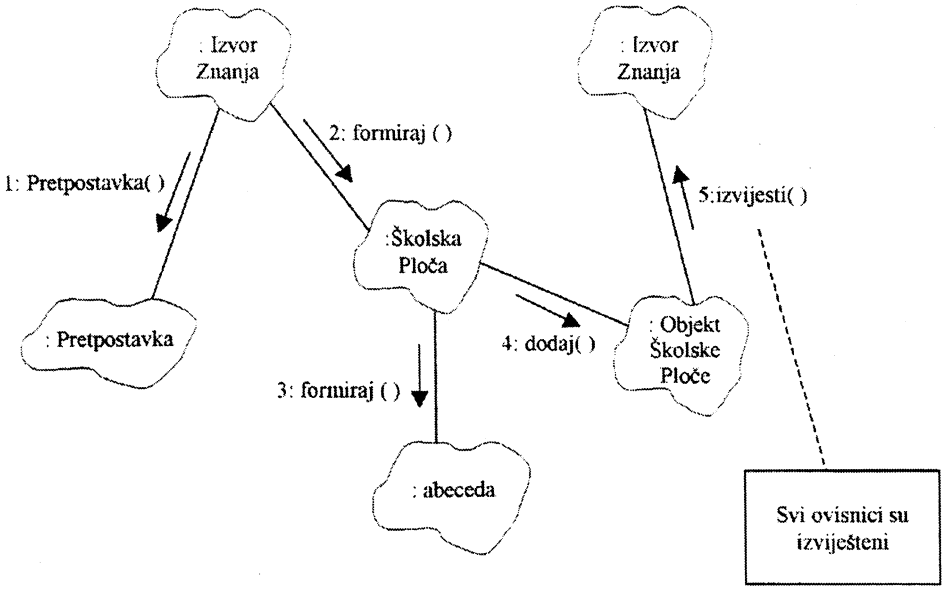
Kako smo ranije primjetili, svrha dizajna jest da omogući iscrpan nacrt za implementaciju. Gornji tekst daje dovoljno detaljan algoritam, tako da nije potrebno prikazivati kompletnu implementaciju u C++-u.

Operacija *vratiRješenje* je daleko jednostavnija jer jedino što moramo jest vratiti vrijednost rečenice školskoj ploči. Još jednom, pozivanje operacije *vratiRješenje* prije nego što selektor *jeRiješeno* procijeni istinitost donosi samo djelomična rješenja.

2.2. Implementacija mehanizma pretpostavke

U ovom trenutku, implementirali smo mehanizme koji nam omogućavaju postavljanje i vraćanje vrijednosti za objekte školske ploče. Naredna značajnija funkcija obuhvaća mehanizam za donošenje pretpostavki o objektima školske ploče. Ovo je osobito važan predmet, pošto su pretpostavke dinamične (odnosno one se rutinski stvaraju i uklanjaju tokom procesa formiranja rješenja) i njihovo postavljanje ili odstranjivanje aktivira djelovanje upravljača.

Slika 3. Mehanizam Pretpostavke.



Osnovni scenarij kada neki izvor znanja ponudi jednu pretpostavku prikazan je slikom 3. Kao što dijagram pokazuje, jednom kada neki izvor znanja stvori pretpostavku, on o tome izvještava školsku ploču, koja zatim donosi pretpostavku za svoju abecedu i onda za svaki objekt školske ploče na koji se ta pretpostavka odnosi. Rabeći mehanizam ovisnosti, dotični bi objekt školske ploče mogao nakon toga izvjestiti sebi ovisne izvore znanja.

U svojoj najosnovnijoj implementaciji, povlačenje pretpostavke jednostavno poništava rad ovog mehanizma. Na primjer, kako bismo povukli jednu

pretpostavku o šifriranom slovu, prizovemo skup pretpostavki do, i uključujući onu koju povlačimo. Na taj se način, dotična pretpostavka kao i sve one koje su stvorene na njenoj osnovi poništavaju.

Moguć je i malo sofisticiraniji mehanizam. Na primjer, uzmimo da smo donijeli pretpostavku da je određena jednoslovna riječ upravo slovo *I* (pretpostavljajući da nam je potreban samoglasnik). Mogli bismo kasnije stvoriti drugu pretpostavku da određena riječ dvostrukog slova jest *JJ* (pod pretpostavkom da nam treba suglasnik). Ako zatim uvidimo da moramo povući prvu pretpostavku, vjerojatno je da ne moramo povući i drugu. Takav pristup zahtijeva od nas da pridodamo novo djelovanje razredu *Pretpostavka*, tako da on može voditi evidenciju koje su pretpostavke ovisne o drugima. S razlogom možemo odložiti ovo proširenje za mnogo kasniju fazu u razvoju ovog sustava, pošto uključivanje ovog djelovanja nema utjecaja na samu arhitekturu.

3. DODAVANJE NOVIH IZVORA ZNANJA

Sada kada imamo ključna svojstva i mehanizme sustava školske ploče na svojim mjestima, i jednom kada mehanizmi za pokretanje i povlačenje pretpostavki prorade, naš naredni korak je implementiranje razreda *UređajZaključivanja*, pošto su svi izvori znanja o njemu ovisni. Kako je ranije napomenuto, ovaj razred posjeduje samo jednu zaista interesantnu operaciju, *procijeniPravila*. Nećemo ovdje prikazati njene detalje, pošto ova specifična metoda ne obuhvaća nove važne predmete dizajna.

Jednom kada budemo sigurni da naš uređaj za zaključivanje radi ispravno, u mogućnosti smo pojedinačno priključiti svaki izvor znanja. Podvlačimo važnost uporabe pojedinačnog procesa iz dva razloga:

- Za dani izvor znanja, nije u potpunosti jasno koja su pravila zaista važna sve dok ih ne primijenimo na stvarne probleme.
- Otklanjanje grešaka baze znanja daleko je lakše ako prvo implementiramo i testiramo manje, srodne skupove pravila, nego ako ih pokušamo testirati sve od jednom.

U osnovi, implementacija svakog izvora znanja poglavito je problem formiranja znanja. Za određeni izvor znanja, moramo se savjetovati sa stručnjakom (po mogućnosti analizatorom kriptograma) kako bismo odlučili koja su pravila bitna. Tokom testiranja svakog izvora znanja, naša bi analiza mogla pokazati da su neka pravila beskorisna, dok su neka ili previše specifična ili previše općenita, dok neka možda i nedostaju. Tada bismo mogli odlučiti promijeniti pravila dotičnog izvora znanja ili čak dodati nove izvore znanja.

Kako implementiramo svaki izvor znanja, možda otkrijemo postojanje uobičajenih pravila kao i uobičajenog djelovanja. Na primjer, mogli bismo

primijetiti da *IzvorZnanjaStruktureRiječi* i *IzvorZnanjaStruktureRečenice* dijele zajedničko djelovanje, koje se očituje u tome da oboje moraju znati kako procijeniti pravila glede pravovaljanog uređenja određenih konstrukcija. Prvi izvor znanja je zainteresiran za poredak slova, dok je drugi zainteresiran za poredak riječi. U oba slučaja, obrada je ista, pa nam je stoga svrsishodno izmijeniti strukturu razreda izvora znanja razvijanjem novog razreda jedinstvenog djelovanja, nazvanog *IzvorZnanjaStrukture*, u koji postavljamo ovo zajedničko djelovanje.

Ova nova hijerarhija razreda izvora znanja baca dodatno svjetlo na činjenicu da procjena skupa pravila ovisi i o vrsti izvora znanja kao i o vrsti objekta školske ploče. Na primjer, za određeni izvor znanja moglo bi se koristiti povezivanje unaprijed nad jednom vrstom objekta školske ploče, i povezivanje unatrag nad drugom. Uz to, za određeni objekt školske ploče, način na koji se on procijenjuje ovisi o tome koji se izvor znanja primjenjuje.

4. ODRŽAVANJE

U ovom odjeljku, razmotriti ćemo poboljšanje funkcionalnosti sustava analize kriptograma i promatrati kako naš dizajn podnosi tu promjenu.

U bilo kojem inteligentnom sustavu, bitno je znati što je krajni odgovor na neki problem, ali je često jednako važno znati kako je sustav došao do tog rješenja. Stoga zahtijevamo od naše aplikacije da bude samo-ispitivačka, odnosno da vodi evidenciju o tome koji su izvori znanja bili aktivirani, koje su pretpostavke donesene i zašto, i tako dalje, kako bismo je kasnije mogli ispitivati, na primjer, o tome zašto je donijeta neka pretpostavka, kako se došlo do neke druge pretpostavke, te kada je pojedini izvor znanja bio aktiviran.

4.1. Pridavanje nove funkcionalnosti

Kako bismo pridodali ovu novu funkcionalnost, potrebno je uraditi dvije stvari. Prvo, moramo osmisliti mehanizam za vođenje evidencije o radu upravljača i svakog izvora znanja, i drugo, trebamo modificirati odgovarajuće operacije kako bi one mogle bilježiti ove informacije. U biti, dizajn traži od izvora znanja i upravljača da bilježe što rade u nekom središnjem spremištu.

Započnimo sa stvaranjem razreda potrebnih za podržavanje ovog mehanizma. Prvo bismo mogli definirati razred *Aktivnost*, koji služi za bilježenje što je poduzeo pojedini izvor znanja ili upravljač:

```
class Aktivnost {
public:
   Aktivnost(IzvorZnanja* tko, ObjektŠkolskePloče* što, char* zašto);
   Aktivnost(Upravljač* tko, IzvorZnanja* što, char* zašto);
   ...
};
```

Na primjer, ako je upravljač odabrao pojedini izvor znanja za aktivaciju, on bi formirao jedan objekt iz ovog razreda, postavio da argument *tko* ukazuje na sebe samog, argument *što* da pokazuje na taj izvor znanja i argument *zašto* na neko objašnjenje (koje možda uključuje i trenutni prioritet pomoći).

Prvi dio našeg zadatka je napravljen, dok je drugi dio gotovo jednako lagan. Prisjetite se na trenutak gdje se zbivaju važni događaju u našoj aplikaciji. Kako proizlazi, postoji pet temeljnih vrsta operacija koje su zahvaćene:

- Metode koje izražavaju pretpostavku.
- Metode koje povlače pretpostavku.
- Metode koje aktiviraju izvor znanja.
- Metode koje uzrokuju procjenu pravila.
- Metode koje primaju pomoći od strane izvora znanja.

U stvarnosti, ovi su događaji uvelike ograničeni na dvije lokacije u arhitekturi, kao dio upravljačkog uređaja konačnog stanja, i kao dio mehanizma pretpostavke. Naš zadatak održavanja stoga ima za posljedicu doprijeti do do svih metoda koje odigravaju neku ulogu u te dvije lokacije, što je zamorno, ali ni u kom slučaju velika nauka. I zaista, najvažnije otkriće jest da dodavanje novog djelovanja ne zahtijeva značajnu promjenu arhitekture.

Kako bismo dovršili naš posao ovdje, također je potrebno implementirati razred koji može odgovoriti na tko, što, kada i zašto pitanja korisnika. Dizajn takvog objekta nije strašno kompliciran jer sve informacije koje su mu potrebne mogu se pronaći kao stanja objekata razreda *Aktivnosti*.

4.2. Promjena Zahtjeva

Jednom kada imamo postojanu implementaciju na svom mjestu, mnogi se novi zahtjevi mogu objediniti s minimalnom promjenom našeg dizajna. Promotrimo tri vrste novih zahtjeva:

- Sposobnost odgonetavanja drugih jezika osim hrvatskog.
- Sposobnost odgonetavanja koristeći i premještanje slova uz jednostruku supstituciju slova.
- Sposobnost učenja iz iskustva.

Prva je promjena prilično jednostavna, zato što je činjenica da naša aplikacija koristi hrvatski jezik najvećim dijelom nebitna za naš dizajn. Pod pretpostavkom da je korišten isti skup znakova, u pitanju je prvenstveno mijenjanje pravila glede svakog izvora znanja. Zapravo, niti mijenjanje skupa znakova nije tako teško, pošto što čak niti razred *abeceda* nije ovisan o tome kojim se znakovima manipulira.

Druga je promjena mnogo teža, ali je ipak moguća u kontekstu sustava školske ploče. U biti, naš je pristup dodavanje novih izvora znanja koji utjelovljuju informacije o premještanju slova. Ponovno, ova promjena ne mijenja bilo koje postojeće ključno svojstvo ili mehanizam u našem dizajnu. Ona prije uključuje dodavanje novih razreda koji koriste postojeća sredstva, kao što su razred *UređajZaključivanja* i mehanizam pretpostavke.

Treća je promjena najteža od svih, uglavnom zbog toga što je učenje strojeva na granicama našeg znanja o umjetnoj inteligenciji. Kao jedan pristup, kada upravljač otkrije da više ne može nastaviti, mogao bi upitati korisnika za pomoć. Bilježenjem ove pomoći, uz aktivnosti koje su dovele da se sustav zaglavi, aplikacija školske ploče u mogućnosti je ubuduće izbjegnuti sličan problem. Mogli bismo objediniti ovaj jednostavni mehanizam učenja bez da naveliko mijenjamo bilo koji od naših postojećih razreda, te kao što je to slučaj sa svim ostalim promjenama, i ova se može izgraditi na postojećoj strukturi.

5. ZAKLJUČAK

U radu su bile razmotrene tri vrste novih zahtjeva koji se sasvim prirodno mogu nadovezati na već postojeće kao logičko strukturno proširenje. Niti jedna od njih nije neizvediva u već formiranim okvirima, i zapravo jedino ograničenje predstavljaju naše mogućnosti konstrukcije modela umjetne inteligencije koji će realno iskazivati svojstva koja demonstriraju samo inteligentna bića, kao što su sposobnosti učenja i adaptacije. Pošto je već uvelike u tijeku konstrukcija i takvih sustava, nije teško prepoznati izvanredne potencijale koji se otkrivaju integracijom takvih sustava u jedinstvenu cjelinu i njihovom primjenom na rješavanje i najtežih poznatih matematičkih i inih problema. Niz novih spoznaja i postignuća na ovom području, te razrješavanje poteškoća nastalih brzim razvojem, moguće je razriješiti upravo predloženim objektno orijentiranim pristupom modeliranju razvoja i održavanja sigurnosnog kriptografskog modela predloženog u ovom radu.

OBJECT-ORIENTED APPROACH TO CRYPTOGRAPHIC SECURITY MODEL DEVELOPMENT AND MAINTENANCE MODELING

SUMMARY

Based on the previously achieved result founded on object-oriented design, we come to the integration of all elements into a functional unit. A major factor when deciding about the system architecture was the possibility of extension and adding of new

performances. It is the object-oriented strategy of enclosing certain functions and characteristics into individual entities, that is, the concept of object-oriented development, that will turn out to be crucial, since now we will take into account specific examples of new potential and logical requirements that can be posed before our system and demonstrate the functionality of thus created blackboard model structure, especially from the viewpoint of maintaining such a system built with an object-oriented approach.

6. LITERATURA

1. Andrzej Lasota, Michael C. MacKey : Chaos, Fractals, and Noise : Stochastic Aspects of Dynamics (Applied Mathematical Sciences, Vol 97) / Published 1994.
2. Antoliš K. et al: Transitional dynamics implication in methodology of integral information system design, Informatologija, Zagreb, 1998.
3. Antoliš K. et al : Metodologija planiranja i upravljanja razvojem ekspertnih sustava, Business system managment – UPS '97., Mostar, 1997.
4. Antoliš K. et al : Determiniranost informacijskog razvitka računalnim konceptima šeste generacije / II dio Zbornik radova, Međunarodno znanstveno savjetovanje "Promet na prijelazu u 21. stoljeće", Opatija, 1999.
5. Edward Yourdon, Carl A. Argila: Case Studies in Object-Oriented Analysis and Design (Yourdon Press Computing Series) / Published 1996.
6. Grbavac V. et al : Ekspertni sustavi u funkciji razvoja prometa, IESP '96. Ljubljana, 1996.
7. Kim Walden, Jean-Marc Nerson: Seamless Object-Oriented Software Architecture : Analysis and Design of Reliable Systems (The Object-Oriented) / Published 1995.
8. Martin, J.: Information engineering Book III Design and Construction, Prentice Hall, College Technical and Reference Division Englewood Chliffs, N.J. 1990.
9. Merle, P., Martin, J.: Analysis and Design of Business Information Systems, Maxwell Macmillan International Publishing Group, Singapore, 1991.

Adresa autora - Author's address:
Dr. sc. Krunoslav Antoliš
GS OS RH HVU Učilište hrvatske kopnene vojske
10000 Zagreb, Hrvatska
Prof. dr. sc. Vitomir Grbavac
Agronomski fakultet Sveučilišta u Zagrebu
10000 Zagreb, Hrvatska

Primljeno – Received:
20. 11. 1999.