AGRONOMSKI GLASNIK 3-4/1999.
ISSN 0002-1954

Izvorni znanstveni članak
Original scientific paper

OBJEKTNI PRISTUP MODELIRANJU DIZAJNA SIGURNOSNOG KRIPTOGRAFSKOG MODELA

OBJECT-ORIENTED APPROACH TO CRYPTOGRAPHIC SECURITY MODEL DESIGN MODELING

V. Grbavac, K. Antoliš

SAŽETAK

Pravilno projektirani dizajn koji će omogućiti ostvarivanje zahtjeva sigurnosnog sustava nit je vodilja u ovom radu. Sustav školske ploče, izvora znanja i upravljača, zahtijeva formiranje hijerarhijski uređenog niza razreda s određenim brojem precizno definiranih vitalnih operacija što će tvoriti kostur predloženog rješenja. U radu je dat pregled ključnih elemenata sustava čija su integracija i interakcija s ostalim elementima detaljno razmatrane i definirane, a nadopunjen je dijagramima koji omogućuju vizualni kontakt s razrješavanjem narečene problematike.

Ključne riječi: dizajn sigurnosnog kriptografskog modela

ABSTRACT

Properly planned design, that will allow implementation of the security system requirements represents our basic concern in this work. The system, comprising the blackboard, the knowledge sources and the controller, requires the formation of a hierarchically arranged series of classes as well as a number of precisely defined key operations that will constitute the skeleton of the presented application. A review of the basic system elements, whose integration and interaction with other elements is discussed and defined in details, can be found in this work, along with the concept diagrams that make a visual contact with the described problem-solving domain a reality.

Key words: Cryptographic security model design

1. UVOD

Inteligentni se sustavi razvijaju u mnogim poljima koja imaju utjecaj i na život i na materijalna sredstva, kao što su medicinska dijagnoza ili vojni proces donošenja odluke, primjerice pri navođenju borbenih aviona. To čini našu zadaću vrlo zahtjevnom, budući da ove sustave moramo dizajnirati tako da oni nikad ne dovode u opasnost, pa stoga umjetna inteligencija uglavnom obuhvaća vrlo specijalizirana znanja.

Iako se u ovom području ponekad pretjerivalo u previše zanesenoj štampi, proučavanje umjetne inteligencije dalo nam je neke vrlo razborite i praktične ideje. U njih ubrajamo pokušaje predstavljanja znanja te razvoj uobičajenih arhitektura inteligentnih sustava za rješavanje problema, uključujući stručne sustave zasnovane na pravilima te sustav školske ploče.

2. ARHITEKTURA ŠKOLSKE PLOČE

Sada smo pripravni dizajnirati rješenje problema analize kriptograma rabeći sustav školske ploče opisan u prethodnom odjeljku. Ovo je klasičan primjer opetovane opće primjene, jer smo ponovno u mogućnosti iskoristiti dokazani obrazac arhitekture kao temelj našeg dizajna. Struktura sustava školske ploče indicira da se među objektima najviše razine u našem sustavu nalaze školska ploča, nekoliko izvora znanja i upravljač. Naš je zadatak identificirati razrede specifičnih područja i objekte koji posjeduju navedena osnovna svojstva.

2.1. Objekti školske ploče

Objekti koji se pojavljuju na školskoj ploči nalaze se u strukturnoj arhitekturi koja se podudara s različitim razinama svojstava naših izvora znanja. Stoga imamo sljedeća tri razreda:

- *Rečenica* Potpuni kriptogram.
- *Riječ* Pojedinačna riječ u kriptogramu.
- *Šifrirano slovo* Pojedinačno slovo u riječi.

Izvori znanja također moraju dijeliti znanje o pretpostavkama koje svaki donosi, tako da ćemo uključiti sljedeći razred objekata školske ploče:

- *Pretpostavka* Pretpostavka načinjena od izvora znanja.

Napokon, važno je znati koja su abecedna slova iz običnog teksta i koja iz šifriranog teksta bila iskorištena u pretpostavkama postavljenim od izvora znanja, tako da dodajemo naredni razred:

- *Abeceda* Abeceda običnog teksta, abeceda šifriranog teksta i veze između njih.

Postoji li bilo što zajedničko tim petorim razredima? Odgovaramo s odlučnim da, jer svaki od ovih razreda objekata predstavlja objekte koji se mogu postaviti na školsku ploču, i sama ta karakteristika razlikuje ih od, na primjer, izvora znanja i upravljača. Stoga ćemo kreirati sljedeći razred kao nadrazred svakog objekta koji se može pojaviti na ploči:

class ObjektŠkolskePloče ...

Bacivši jedan pogled izvana na ovaj razred, mogli bismo definirati dvije prikladne operacije:

- *postavi* Postavi objekt na školsku ploču.
- *odstrani* Odstrani objekt sa školske ploče.

Zašto definiramo *postavi* i *odstrani* kao operacije nad objektima razreda *ObjektŠkolskePloče*, umjesto na samoj školskoj ploči? Ova je situacija slična onoj kada se nalaže nekom objektu da se sam ispiše u jednom prozoru. Pravi test kod odlučivanja gdje postaviti ove vrste operacija jest da li sam razred raspolaže s dovoljno znanja ili da li je dužan izvršiti zadanu operaciju. U slučaju operacija *postavi* i *odstrani*, to je zaista i slučaj; objekt školske ploče jedini posjeduje svojstva s detaljnim poznavanjem načina kako da se pridoda ili ukloni s ploče (iako se zasigurno zahtijeva i suradnja s objektom školske ploče). činjenica jest da je nužno da svaki objekt školske ploče bude svjestan da je postavljen na ploču, jer samo tada može započeti djelovanje u oportunističkom rješavanju samog problema na ploči.

2.2. Ovisnosti i izjave

Zasebne rečenice, riječi i šifrirana slova imaju još jednu zajedničku točku: svatko ima određene izvore znanja koji o njima ovise. Neki izvor znanja može izraziti interes za jedan ili nekoliko ovih objekata, i zbog toga, rečenica, riječ i šifrirano slovo moraju posjedovati vezu sa svakim takvim izvorom znanja, tako da kada se pojedina pretpostavka o tom objektu promijeni, odgovarajući izvor znanja može biti izviješten da se nešto zanimljivo dogodilo. Kako bismo

ostvarili ovaj mehanizam, uvodimo jednostavan razred jedinstvenog djelovanja ("mixing class"):

```
class Ovisnik {
public:
  Ovisnik();
  Ovisnik(const Ovisnik&);
  virtual ~Ovisnik();
  ...
protected:
  NeograničeniSkup<IzvorZnanja*> veze;
};
```

Ovdje možemo primijetiti da razred *Ovisnik* sadrži jedinstveni član, objekt koji predstavlja skup pokazivača na izvore znanja[1].

Definirat ćemo sljedeće operacije u ovom razredu:

- *dodaj* Dodaj vezu prema izvoru znanja.
- *ukloni* Ukloni vezu prema izvoru znanja.
- *brojOvisnika* Obavijesti o broju ovisnika.
- *izvijesti* Pošalji operaciju svakom ovisniku.

Operacija *izvijesti* ima značenje pasivnog iteratora, odnosno kada je pozovemo, možemo dostaviti neku operaciju koju želimo obaviti nad svakim od ovisnika u skupini.

Ovisnost je neovisna osobina koja se može uvesti u druge razrede. Na primjer, šifrirano slovo je objekt školske ploče, kao i ovisnik, tako da možemo kombinirati ova dva svojstva kako bismo ostvarili željeno djelovanje. Korištenje razreda jedinstvenog djelovanja na ovaj način povećava mogućnost ponovne uporabe i razdvajanje zamisli u našoj arhitekturi.

Šifrirana slova i abeceda imaju još jednu zajedničku osobinu: pretpostavke se mogu donijeti za objekte obaju razreda (prisjetite se da je i tvrdnja također vrsta *ObjektaŠkolskePloče*). Na primjer, određeni izvor znanja mogao bi pretpostaviti da šifrirano slovo *K* predstavlja obično slovo *P*. Kako se približavamo rješenju našeg problema, mogli bismo napraviti nepromjenjivu tvrdnju da *G* predstavlja *J*. Stoga ćemo uključiti sljedeći razred:

```
class Izjava ...
```

[1] Što se tiče arhitekture temeljnih razreda, ranije smo primijetili da je neograničenim strukturama potreban jedan upravljač spremišta. Zbog jednostavnosti, izostavit ćemo ovaj sustavni argument iz dotične, kao i iz ostalih sličnih deklaracija u ovom radu. Naravno, potpuna implementacija morala bi poštivati sve osnovne mehanizme sustava.

Dužnosti ovog razreda su čuvanje pretpostavki ili tvrdnji o pridruženom objektu. Ne koristimo *Izjavu* kao razred jedinstvenog djelovanja, nego za sjedinjenje. O slovima se *daju* izjave, ona nisu *vrste* izjava.

U našoj arhitekturi davat ćemo izjave samo o pojedinačnim slovima, kao kod šifriranih slova i abecede. Kao što je naš raniji scenarij podrazumijevao, šifrirana slova predstavljaju pojedinačna slova o kojima se mogu dati izjave, a abeceda sadržava više slova, od kojih se o svakom mogu dati različite izjave. Definiranje *Izjave* kao neovisnog razreda stoga služi da bi objedinio zajedničko djelovanje ovih dvaju u osnovi različitih razreda.

Definirat ćemo sljedeće operacije za objekte ovog razreda:

- *formiraj* Formiraj izjavu.
- *povuci* Povuci izjavu.
- *šifriranitekst* Na osnovi danog običnog slova, vrati njegov šifrirani ekvivalent.
- *običnitekst* Na osnovi danog šifriranog slova, vrati njegov ekvivalent iz običnog teksta.

Daljnja analiza daje do znanja da bismo u ranoj fazi trebali razlučivati između dviju uloga koje odigrava izjava: pretpostavka, koja predstavlja privremenu vezu između šifriranog slova i svog ekvivalenta iz običnog teksta, i tvrdnje, koja je trajna veza, što znači da je veza definirana i stoga nepromjenjiva. Tijekom rješavanja kriptograma, izvori će znanja napraviti mnoge pretpostavke, i kako se primičemo rješenju, te veze će na kraju postajati tvrdnje. Kako bismo modelirali ove promjenjive uloge, usavršit ćemo prethodno identificiran razred *Pretpostavka*, i uvesti novi podrazred *Tvrdnja*, a objekti oba razreda bit će upravljani od strane objekata razreda *Izjava* te postavljeni na školsku ploču. Počet ćemo dovršenjem deklaracije ulaznih argumenata i izlaznih vrijednosti za operacije *formiraj* i *povuci*, kako bismo uključili jedan argument *Pretpostavke* ili *Tvrdnje*, a zatim dodati sljedeće selektore:

- *ObičnoSlovojePotvrđeno* Selektor: da li je obično slovo definirano?
- *šifriranoSlovojePotvrđeno* Selektor: da li je šifrirano slovo definirano?
- *ObičnoSlovoImaPretpostavku* Selektor: postoji li pretpostavka o običnom slovu?
- *ŠifriranoSlovoImaPretpostavku* Selektor: postoji li pretpostavka o šifriranom slovu?

Zatim ćemo definirati razred *Pretpostavka*. Budući da ovaj razred posjeduje uvelike strukturalna svojstva, ostavit ćemo neka od njegovih stanja neobjedinjena:

```
class Pretpostavka : public ObjektŠkolskePloče {
public:
  ...
  ObjektŠkolskePloče* cilj;
  IzvorZnanja* tvorac;
  Niz<char> razlog;
  char običnoSlovo;
  char šifriranoSlovo;
};
```

Obratite pažnju da smo ponovno upotrijebili jedan razred iz ranije opisanog sustava, sustavni razred *Niz*.

Pretpostavke su vrste objekata školske ploče jer predstavljaju stanje koje je od općeg interesa svim izvorima znanja. Razni objektni članovi predstavljaju sljedeća svojstva:

- *cilj* Objekt školske ploče o kojem je donesena pretpostavka.
- *tvorac* Izvor znanja koji je donio pretpostavku.
- *razlog* Razlog zbog kojeg je izvor znanja donio pretpostavku.
- *običnoSlovo* Obično slovo o kojem je donesena pretpostavka.
- *šifriranoSlovo* Pretpostavljena vrijednost običnog slova.

Potreba za svakim od ovih svojstava dolazi poglavito od prirode same pretpostavke: pojedini izvor znanja donosi neku pretpostavku o vezi između jednog običnog i šifriranog slova, i čini to s nekim razlogom (obično zbog nekog pravila). Potreba za prvim članom, *ciljem*, manje je očigledna. On je ovdje uključen zbog problema traganja unatrag. Ako ćemo ikada morati preinačiti neku pretpostavku, moramo o tome izvijestiti sve objekte školske ploče za koje je ta pretpostavka prvotno donesena, tako da oni onda mogu upozoriti izvore znanja o kojima ovise (pomoću mehanizma ovisnosti) da se njihovo značenje promijenilo.

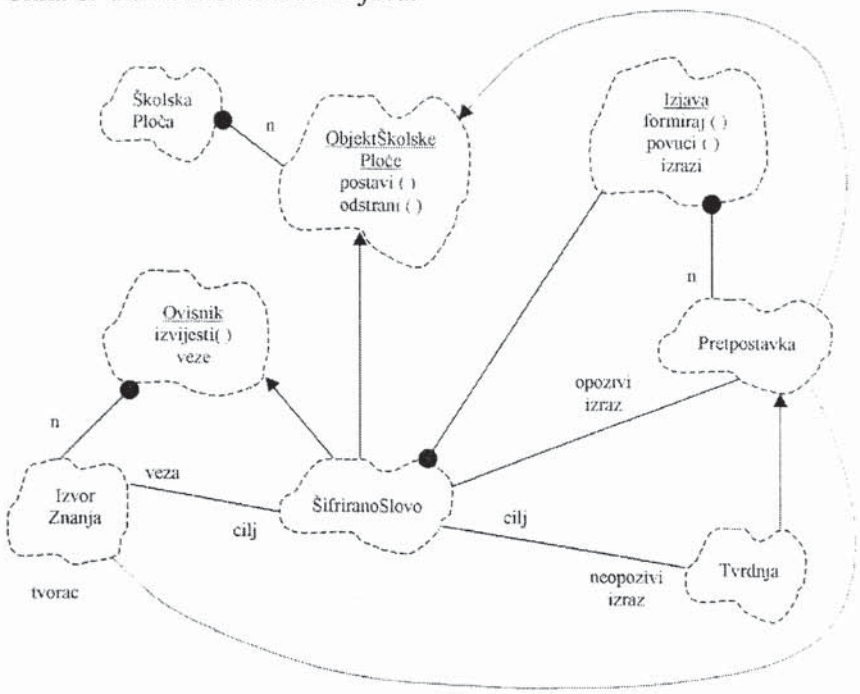
Nadalje, imamo podrazred *Tvrdnja*:

```
class Tvrdnja : public Pretpostavka ...
```

Razredi *Pretpostavka* i *Tvrdnja* dijele sljedeću operaciju, između ostalih:

- *jeOpozivo* Selektor: da li je veza privremena?

Slika 1. Razredi *Ovisnost* i *Izjava*.



Svi objekti pretpostavki odgovaraju s istinito selektoru *jeOpozivo*, dok svi objekti tvrdnji odgovaraju s neistinito. Uz to, jednom donesena, tvrdnja ne može biti ni ponovno postavljena niti povučena.

Slika 1 prikazuje dijagram razreda koji ilustrira suradnju između razreda ovisnosti i izjava. Obratite osobitu pažnju na uloge koje svako svojstvo odigrava u raznim asocijacijama. Na primjer, *IzvorZnanja* je tvorac jedne *Pretpostavke*, a isto tako je veza jednog *ŠifriranogSlova*. Budući da uloga ima drugačiji prizor za svijet oko sebe nego što ga ima svojstvo, očekivali bismo različit protokol između izvora znanja i pretpostavki, nego između izvora znanja i slova.

2.3. Dizajn objekata školske ploče

Odvojimo sada malo vremena kako bismo dovršili naš dizajn razreda *Rečenica*, *Riječ* i *ŠifriranoSlovo*, a zatim i razreda *Abeceda*. Rečenica je

prilično jednostavna, ona je objekt školske ploče te ovisnik, i označava listu riječi koje tvore dotičnu rečenicu. Stoga možemo pisati

```
class Rečenica : public ObjektŠkolskePloče,
        virtual public Ovisnik {
public:
  ...
protected:
  Lista<Riječ*> riječi;
};
```

Deklarirat ćemo nadrazred *Ovisnik* kao virtualan, jer držimo mogućim postojanje i drugih podrazreda *Rečenica* koji će također pokušati preuzeti strukturu ili djelovanje od *Ovisnika*. Označavanjem ovog nasljednog odnosa *virtualnim*, omogućit ćemo tim podrazredima da dijele svojstva jedinstvenog nadrazreda *Ovisnik*.

Osim operacija *postavi* i *odstrani* definiranih u nadrazredu *ObjektŠkolskePloče*, i četiriju operacija definiranih u *Ovisniku*, dodat ćemo sljedeće dvije posebne operacije nad rečenicama:

- *vrijednost* Izvijesti o trenutnoj vrijednosti rečenice.
- *jeRiješeno* Izvijesti *istinito* ako postoji po jedna tvrdnja za svaku riječ u rečenici.

Na početku problema, *vrijednost* vraća jedan niz koji predstavlja originalni kriptogram. Jednom kada *jeRiješeno* procijeni istinitost, operacija *vrijednost* može se iskoristiti da bismo dobili rješenje u formatu običnog teksta. Pristupanjem *vrijednosti* prije nego *jeRiješeno* doznači istinito postići će samo djelomično rješenje.

Baš kao i razred rečenica, i riječ je jedna vrsta objekta školske ploče i vrsta ovisnika. Nadalje, riječ označava niz slova. Kako bismo podržali izvore znanja koji manipuliraju riječima, uključit ćemo vezu od riječi prema njenoj rečenici, te od riječi prema prethodnoj i sljedećoj riječi u rečenici. Stoga možemo napisati sljedeće:

```
class Riječ : public ObjektŠkolskePloče,
        virtual public Ovisnik {
public:
  ...
  Rečenica& rečenica() const;
  Riječ* prethodna() const;
```

```
  Riječ* sljedeća() const;
protected:
  Lista<ŠifriranoSlovo*> slova;
};
```

Kao što smo to učinili i kod operacija nad rečenicama, definirat ćemo naredne dvije operacije za razred *Riječ*:

- *vrijednost* Izvijesti o trenutnoj vrijednosti riječi.
- *jeRiješeno* Izvijesti *istinito* ako postoji po jedna tvrdnja za svako slovo u riječi.

Sada možemo definirati razred *ŠifriranoSlovo*. Objekt ovog razreda je vrsta objekta školske ploče i vrsta ovisnika. Osim djelovanja koje je preuzelo, svako šifrirano slovo ima vrijednost (na primjer, šifrirano slovo *H*) uz skup pretpostavki i tvrdnji u svezi s odgovarajućim običnim slovom. Možemo upotrijebiti razred *Izjava* kako bismo prikupili ove izraze. Tako možemo napisati sljedeće:

```
class ŠifriranoSlovo : public ObjektŠkolskePloče,
                       virtual public Ovisnik {
public:
  ...
  char vrijednost() const;
  int jeRiješeno() const;...
protected:
  char slovo;
  Izjava izjave;
};
```

Obratite pažnju da smo uključili selektore *vrijednost* i *jeRiješeno*, slično našem dizajnu *Rečenice* i *Riječi*. Na kraju ćemo također morati priskrbiti operacije za klijente *ŠifriranogSlova* kako bismo mogli pristupiti pretpostavkama i tvrdnjama na jedan siguran način.

Jedna napomena glede objektnog člana *izjave*: očekujemo da je ovo skup pretpostavki i tvrdnji uređenih prema trenutku njihova nastajanja, s time da najnoviji izraz u skupu predstavlja trenutnu pretpostavku ili tvrdnju. Razlog zbog kojeg smo odlučili voditi evidenciju o svim pretpostavkama jest omogućavanje izvorima znanja da pogledaju ranije pretpostavke koje su bile odbačene, kako bi mogli učiti iz prijašnjih grešaka. Ova odluka utječe na naš dizajn razreda *Izjava*, kojem ćemo pridodati sljedeće operacije:

- *najNovija* Selektor: izvješćuje o najnovijoj pretpostavci ili tvrdnji.
- *izrazN* Selektor: izvješćuje o *n*-tom izrazu.

Nakon što smo dotjerali njegovo djelovanje, možemo donijeti razumnu implementacijsku odluku o razredu *Izjava*. Konkretno, možemo uključiti sljedeći zaštićeni objektni član:

NeograničeniUređeniSkup<Pretpostavka*> izrazi;

NeograničeniUređeniSkup još je jedan ponovno upotrebljiv, ranije opisan razred.

Promotrite sada razred nazvan *Abeceda*. Ovaj razred predstavlja cijelu abecedu običnog teksta i šifriranog teksta, kao i veze između njih. Ova je informacija značajna jer je svaki izvor znanja može koristiti kako bi odgonetnuo koje su veze do tada bile uspostavljene a koje tek treba odrediti. Na primjer, ako već postoji tvrdnja da je šifrirano slovo *C* u stvari slovo *M*, onda jedan abecedni objekt bilježi ovu vezu tako da niti jedan drugi izvor znanja ne može upotrijebiti obično slovo *M*. Radi djelotvornosti, potrebno je ispitati vezu u oba smjera: na dano šifrirano slovo, vraća li se njegova veza u običnom tekstu, a na dano obično slovo, vraća li se njegova veza u šifriranom tekstu. Razred *Abeceda* možemo definirati na sljedeći način:

```
class Abeceda : public ObjektŠkolskePloče {
public:
  ...
  char običantekst(char) const;
  char šifriranitekst(char) const;
  int jeVezano(char) const;
};
```

Kao i za razred *ŠifriranoSlovo*, i ovdje ćemo uključiti jedan zaštićeni objektni član *izjave* i omogućiti odgovarajuće operacije kako bi se moglo pristupiti njegovom stanju.

Sada smo spremni definirati razred *ŠkolskaPloča*. Ovaj razred ima jednostavnu dužnost prikupljanja objekata razreda *ObjektŠkolskePloče* i njegovih podrazreda. Stoga možemo pisati:

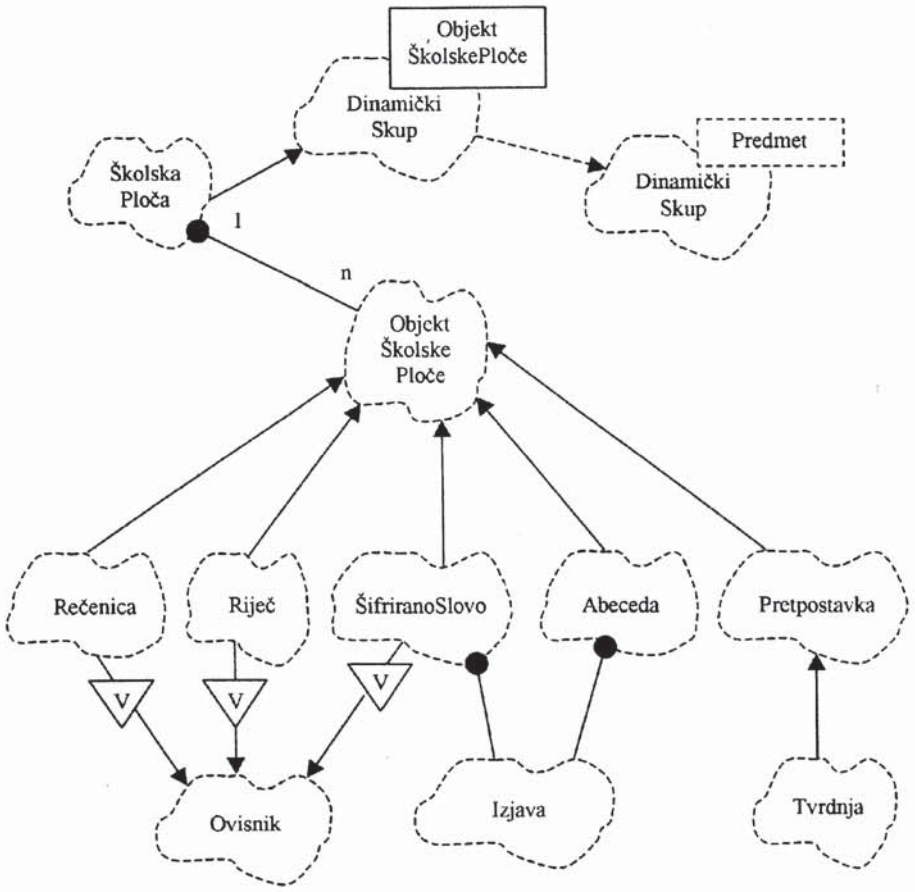
class ŠkolskaPloča : public DinamičkiSkup<ObjektŠkolskePloče*> ...

Odlučili smo da radije preuzmemo nego da obuhvatimo objekt razreda *DinamičkiSkup*, budući da *ŠkolskaPloča* uspješno prolazi naš test za preuzimanje: školska ploča zaista jest jedna vrsta skupa.

Razred *ŠkolskaPloča* omogućava operacije kao što su *dodaj* i *ukloni*, koje preuzima od razreda *Skup*. Naš dizajn uključuje pet posebnih operacija za školsku ploču:

- *resetiraj* Počisti školsku ploču.
- *postaviProblem* Postavi početni problem na ploču.
- *spoji* Priključi izvor znanja ploči.
- *jeRiješeno* Izvijesti istinito ako je rečenica riješena.
- *vratiRješenje* Izvijesti o riješenoj rečenici u običnom tekstu.

Slika 2. Dijagram Razreda Školske Ploče



Druga je operacija potrebna kako bi se stvorila ovisnost između školske ploče i njenih izvora znanja.

Na slici 2, ukratko ćemo prikazati naš dizajn razreda koji surađuju sa *ŠkolskomPločom*. Ovaj dijagram prvenstveno pokazuje odnose preuzimanja, ali zbog jednostavnosti, ne koristi riječ odnos, kao što je onaj između pretpostavke i objekta školske ploče.

Primijetite da u ovom dijagramu prikazujemo da razred *ŠkolskaPloča* daje vrijednosti parametara i preuzima ih od sistemskog razreda *DinamičkiSkup*. Ovaj dijagram također zorno pokazuje zašto je uvođenje razreda *Ovisnik* kao razreda jedinstvenog djelovanja bila dobra odluka pri dizajnu. Konkretno, *Ovisnik* predstavlja djelovanje koje obuhvaća samo dio podrazreda *ObjektŠkolskePloče*. Mogli smo uvesti *Ovisnika* kao jedan među-nadrazred, ali njegovim definiranjem kao razreda jedinstvenog djelovanja radije nego pokušajem njegova vezanja na hijerarhiju *ObjektaŠkolskePloče*, uvećali smo vjerojatnost da će se taj razred ponovno koristiti.

3. DIZAJN IZVORA ZNANJA

U prethodnom odjeljku, identificirali smo trinaest izvora znanja koji se odnose na ovaj problem. Baš kao što smo to uradili za objekte školske ploče, možemo dizajnirati strukturu razreda koja obuhvaća ove izvore znanja i na taj način izdignuti sve zajedničke karakteristike do razreda viših razina.

3.1. Dizajn specijaliziranih izvora znanja

Pretpostavite na trenutak postojanje jednog apstraktnog razreda (razred bez objekata ali koji obuhvaća podrazrede) nazvanog *IzvorZnanja*, čija je namjena uglavnom nalik na onu razreda *ObjektŠkolskePloče*. Radije nego da tretiramo svaki od trinaest izvora znanja kao direktne podrazrede ovog višeg razreda, korisno je prvo obaviti analizu područja i promotriti da li postoje bilo kakve skupine izvora znanja. I zaista, one postoje. Neki izvori znanja djeluju na osnovi cijelih rečenica, drugi na osnovi cijelih riječi, treći na bazi nizova susjednih slova, dok ostali djeluju na temelju pojedinačnih slova. Ove dizajnerske misli možemo ostvariti na sljedeći način:

class IzvorZnanjaRečenice : public IzvorZnanja ...
class IzvorZnanjaRiječi : public IzvorZnanja...
class IzvorZnanjaSlova : public IzvorZnanja...

Za svaki od ovih apstraktnih razreda, možemo pripremiti specifične podrazrede. Na primjer, podrazredi apstraktnog razreda *IzvorZnanjaRečenice* uključuju:

class IzvorZnanjaStruktureRečenice : public IzvorZnanjaRečenice ...
class IzvorZnanjaRješenja : public IzvorZnanjaRečenice ...

Slično tome, podrazredi međurazreda *IzvorZnanjaRiječi* uključuju:

class IzvorZnanjaStruktureRiječi : public IzvorZnanjaRiječi ...
class IzvorZnanjaKratkihRiječi : public IzvorZnanjaRiječi ...
class IzvorZnanjaPodudaranjaObrazaca : public IzvorZnanjaRiječi ...

Posljednji razred zahtijeva objašnjenje. Ranije smo rekli da je namjena ovog razreda predložiti riječi koje odgovaraju određenom obrascu. Možemo koristiti simbole za podudaranje obrazaca za uobičajene izraze, koji su slični onima koje koristi UNIX *grep* alatka:

- Bilo koji predmet ?
- Ne predmet (koji nije) ~
- Zaključni predmet *
- Početak grupe {
- Završetak grupe }

Ovim bismo simbolima mogli dati jednom objektu iz ovog razreda obrazac *?E~{AEIOU}*, i na taj ga način pozvati da izvijesti o svim troslovnim riječima iz svog rječnika koje započinju s bilo kojim slovom, iza kojeg slijedi *E* i zatim bilo koje slovo osim samoglasnika.

Podudaranje obrazaca je općenito korisna opcija, tako da nije iznenađenje da nas potraga za sličnim razredima vodi prema razredima podudaranja obrazaca koje smo ranije opisali. Stoga možemo ocrtati naš izvor znanja o podudaranjima obrazaca na sljedeći način, posudivši pritom od već postojećih razreda:

```
class IzvorZnanjaPodudaranjaObrazaca : public IzvorZnanjaRiječi {
public:
  ...
protected:
  static OgraničeniSkup<Riječ*> riječi;
  PodudaranjeObrazaca PodudarateljObrazaca;
};
```

Svi objekti ovog razreda dijele zajednički rječnik, i svaki objekt ima na raspolaganju vlastita sredstva za provjeru podudaranja obrazaca za uobičajene izraze.

Detaljno djelovanje ovog razreda nije nam toliko bitno u ovoj fazi našeg dizajna, tako da ćemo za kasnije odložiti razvijanje ostalih svojstava, mehanizama te implementaciju ovog razreda.

U nastavku, možemo deklarirati podrazrede razreda *IzvorZnanjaNiza* na sljedeći način:

class IzvorZnanjaUobičajenihPrefiksa : public IzvorZnanjaNiza ...
class IzvorZnanjaUobičajenihSufiksa : public IzvorZnanjaNiza ...
class IzvorZnanjaDvostrukihSlova : public IzvorZnanjaNiza ...
class IzvorZnanjaVažećihNizova : public IzvorZnanjaNiza ...

Na kraju, možemo uvesti i podrazrede apstraktnog razreda *IzvorZnanjaSlova*:

class IzvorZnanjaDirektneSupstitucije : public IzvorZnanjaSlova ...
class IzvorZnanjaUčestalostiSlova : public IzvorZnanjaSlova ...
class IzvorZnanjaSamoglasnika : public IzvorZnanjaSlova ...
class IzvorZnanjaSuglasnika : public IzvorZnanjaSlova ...

3.2. Uopćavanje izvora znanja

Analiza pokazuje da postoje samo dvije osnovne operacije primjenjive na sve ove specijalizirane razrede:

- *resetiraj* Ponovno pokreni izvor znanja.
- *procijeni* Procijeni stanje na školskoj ploči.

Razlog ovih jednostavnih svojstava jest da su izvori znanja relativno autonomni entiteti, što znači da ukažemo jednome na neki zanimljiv objekt školske ploče, i onda mu naložimo da procijeni svoja pravila u odnosu na trenutno opće stanje na ploči. Kao dio procjene pravila, određeni bi izvor znanja mogao poduzeti jednu od sljedećih aktivnosti:

- Donijeti pretpostavku o supstituciji slova.
- Otkriti proturječje u prethodnim pretpostavkama, i potaknuti da odgovarajuća pretpostavka bude povučena.
- Predložiti tvrdnju o supstituciji slova.
- Izvijestiti upravljača da može ponuditi neku korisnu informaciju.

Sve su ovo općenite aktivnosti neovisne o specifičnoj vrsti izvora znanja. I još općenitije, ove aktivnosti predstavljaju djelovanje jednog uređaja za zaključivanje. Govoreći jednostavno, *uređaj za zaključivanje* je objekt koji, na osnovi danih pravila, procjenjuje ta pravila kako bi kreirao nova pravila (povezivanje unaprijed) ili dokazao neku hipotezu (povezivanje unatrag). Stoga predlažemo sljedeći razred:

```
class UređajZaključivanja {
public:
  UređajZaključivanja (DinamičkaGrupa<Pravila*>);
  ...
};
```

Osnovna dužnost konstruktora je da stvori jedan objekt u ovom razredu i popuni ga sa skupom pravila, koji se onda mogu koristiti kod procjenjivanja. U biti, ovaj razred ima samo jednu kritičnu operaciju koja ga čini vidljivim izvorima znanja:

- *procijeni* Procijeni pravila uređaja za zaključivanje.

Ovo je dakle način na koji izvori znanja surađuju: svaki specijalizirani izvor znanja definira svoja vlastita specifična pravila, i delegira dužnost procjene tih pravila razredu *UređajZaključivanja*. Točnije rečeno, mogli bismo kazati da operacija *IzvorZnanja::procijeni* na kraju poziva operaciju *UređajZaključivanja::procijeni*, čiji se rezultati koriste za izvršenje bilo koje od ranije razmatranih četiriju aktivnosti. Na slici 3 ilustrirat ćemo jedan uobičajeni scenarij ove suradnje.

Što je zapravo pravilo? Koristeći format poput Lisp-ovog, mogli bismo sastaviti sljedeće pravilo za izvor znanja uobičajenih sufiksa:

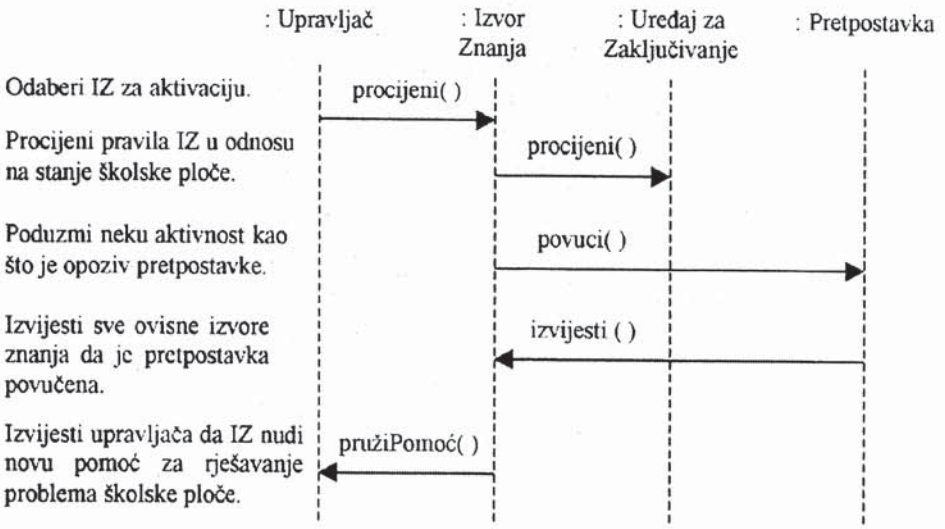
```
((*I??)
  (*ING)
  (*IES)
  (*IED))
```

Ovo pravilo znači da, s danim nizom slova koja se podudaraju s uobičajenim obrascem za izraze *I?? (izraz koji se zamjenjuje), sufiksi kandidati uključuju *ING*, *IES* i *IED* (zamjenski izrazi). U C++-u, mogli bismo definirati razred koji predstavlja neko pravilo na sljedeći način:

```
class Pravilo {
public:
   ...
   int poveži(Niz<char>& prethodnik, Niz<char>& sljedbenik);
   int ukloni(Niz<char>& prethodnik);
   int ukloni(Niz<char>& prethodnik, Niz<char>& sljedbenik);
   int postojiProturječje(const Niz<char>& prethodnik) const;
protected:
   Niz<char> prethodnik;
   Lista<Niz<char>> sljedbenici;
};
```

Predviđena namjena ovih operacija u skladu je s njihovim imenima. Opet smo upotrijebili neke od ranije opisanih razreda.

Slika 3. Scenarij za Procjenu Pravila Izvora Znanja.



U smislu njegove strukture razreda, možemo reći da je izvor znanja jedna vrsta uređaja za zaključivanje. Nadalje, svaki izvor znanja mora na neki način biti povezan s objektom školske ploče, budući da tu pronalazi objekte nad

kojima vrši operacije. I napokon, svaki izvor znanja mora biti povezan i s upravljačem, s kojim surađuje tako da mu šalje pomoći za moguća rješenja. Zauzvrat, upravljač može s vremena na vrijeme aktivirati taj izvor znanja.

Ove bismo dizajnerske odluke mogli izraziti na sljedeći način:

```
class IzvorZnanja : public UređajZaključivanja,
                    public Ovisnik {
public:
  IzvorZnanja(ŠkolskaPloča*, Upravljač*);
  ...
  void resetiraj();
  void procijeni();
protected:
  ŠkolskaPloča* školskaploča;
  Upravljač* upravljač;
  NeograničeniUređeniSkup<Pretpostavka*> prethodnePretpostavke;
};
```

Također smo uveli i zaštićeni objekt *prethodnePretpostavke*, tako da bi izvor znanja mogao voditi evidenciju o svim pretpostavkama i tvrdnjama koje je ikad donosio i tako bio u mogućnosti učiti iz svojih pogrešaka.

Objekti razreda *ŠkolskaPloča* služe kao spremište objekata školske ploče. Iz sličnog razloga potreban nam je razred *IzvoriZnanja* koji označava cijeli skup izvora znanja za pojedini problem. Stoga možemo pisati:

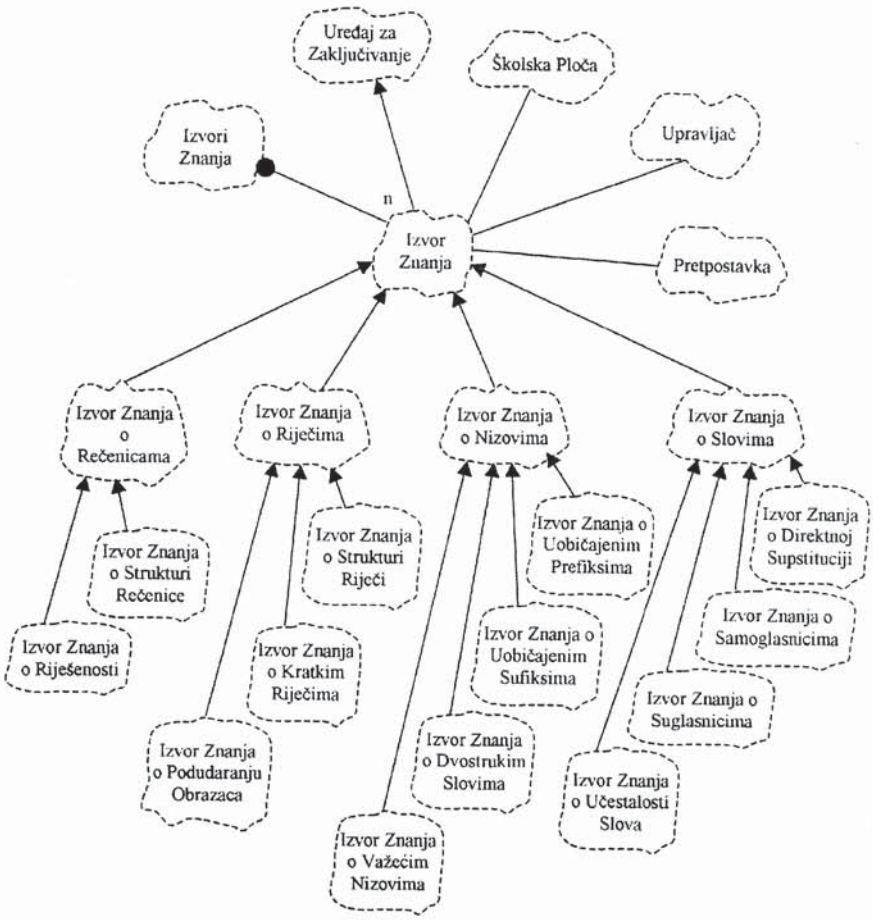
```
class IzvoriZnanja : public DinamičkiSkup<IzvorZnanja*> ...
```

Jedna od dužnosti ovog razreda jest da kada stvorimo jedan objekt u razredu *IzvoriZnanja*, moramo također kreirati i trinaest pojedinačnih objekata izvora znanja. U prilici smo izvršiti tri operacije nad objektima ovog razreda:

- *ponovnopokreni* Ponovno pokreni izvore znanja.
- *pokreniIzvorZnanja* Saopći određenom izvoru znanja njegove početne uvjete.
- *spoji* Priključi izvor znanja školskoj ploči ili upravljaču.

Slika 4 ilustrira razrednu strukturu *IzvoraZnanja*, u skladu s navedenim dizajnerskim odlukama.

Slika 4. Dijagram Razreda Izvora Znanja.



4. DIZAJN UPRAVLJAČA

Razmislite na trenutak o načinu na koji se vrši interakcija između upravljača i pojedinih izvora znanja. U svakoj fazi rješavanja kriptograma, određeni izvor znanja mogao bi otkriti da može dati smisleni doprinos, te stoga šalje pomoć upravljaču. I obratno, taj izvor znanja mogao bi odlučiti da njegova ranija pomoć više ne važi, i stoga je povući. Jednom kada je svim

izvorima znanja dana prilika, upravljač odabire pomoć koja najviše obećava i aktivira odgovarajući izvor znanja pozivajući njegovu operaciju *procijeni.*

Na koji način upravljač odlučuje koji izvor znanja treba aktivirati? Tu bismo mogli osmisliti nekoliko prikladnih pravila:

- Tvrdnja je većeg prioriteta od pretpostavke.
- Izvor znanja - rješavatelj snabdijeva najkorisnije pomoći.
- Izvor znanja - podudaratelj obrazaca pribavlja prioritetnije pomoći od izvora znanja o strukturi rečenice.

Upravljač, dakle odigrava ulogu posrednika odgovornog za posredovanje između raznih izvora znanja koji djeluju nad školskom pločom.

Upravljač mora biti povezan sa svojim izvorima znanja, kojima može pristupiti preko odgovarajuće nazvanog razreda *IzvoriZnanja.* Uz to, upravljač mora imati kao jedan od svojih prioriteta sakupljanje pomoći, uređene na prioritetnoj bazi. Na taj način, upravljač može za aktivaciju odabrati jednostavno onaj izvor znanja koji nudi najinteresantniju pomoć.

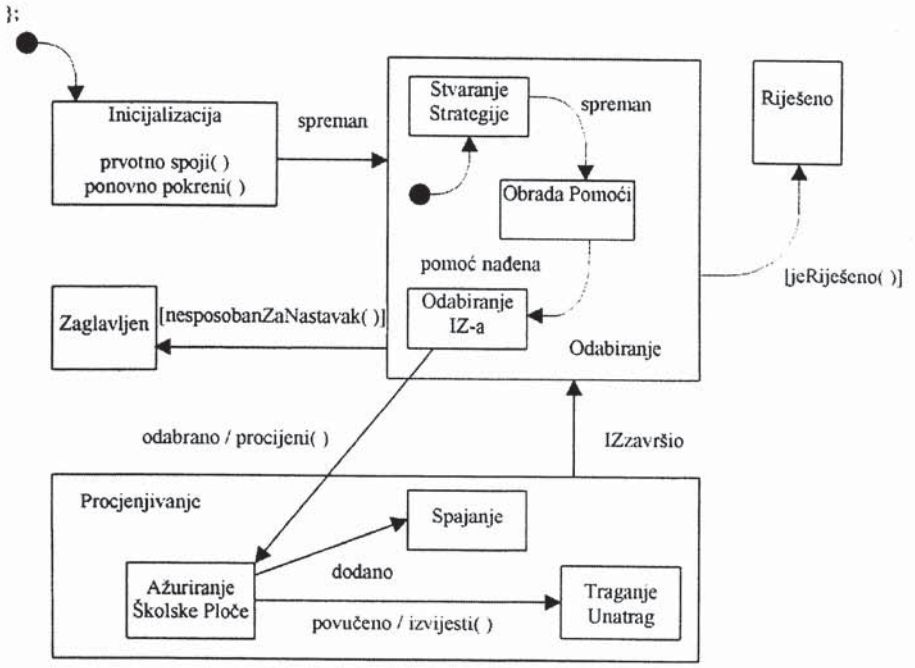
Upustivši se malo u dizajn odvojenijih razreda, ponudit ćemo sljedeće operacije za razred *Upravljač*:

- *resetiraj* — Ponovno pokreni upravljač.
- *pružiPomoć* — Pruži pomoć izvoru znanja.
- *povuciPomoć* — Povuci pomoć od izvora znanja.
- *obradiSljedećuPomoć* — Procijeni sljedeću pomoć najvišeg prioriteta.
- *jeRiješeno* — Selektor: izvijesti *istinito* ako je problem riješen.
- *nesposobanZaNastavak* — Selektor: izvijesti *istinito* ako su izvori znanja zaglavili.
- *spoji* — Priključi upravljač izvoru znanja.

Ove odluke možemo ostvariti sljedećom deklaracijom:

```
class Upravljač {
public:
  ...
  void resetiraj();
  void spoji(IzvorZnanja&);
  void pružiPomoć(IzvorZnanja&);
  void povuciPomoć(IzvorZnanja&);
  void obradiSljedećuPomoć();
  int jeRiješeno() const;
  int nesposobanZaNastavak() const;
};
```

Slika 5. Upravljač Uređaja Konačnog Stanja.



Upravljač je u neku ruku vođen pomoćima koje dobiva od raznih izvora znanja. Kao takvi, uređaji konačnog stanja ("finite state machines") vrlo su prikladni za obuhvaćanje dinamičkog djelovanja ovog razreda.

Na primjer, promotrite dijagram prijelaza stanja prikazanog na slici 5. Ovdje vidimo da se upravljač može naći u jednom od pet glavnih stanja: *Inicijalizacija*, *Odabiranje*, *Procjena*, *Zaglavljen* i *Riješeno*. Najinteresantnija aktivnost upravljača događa se između stanja *Odabiranje* i *Procjena*. Za vrijeme odabira, upravljač po prirodi stvari prelazi od stanja *StvaranjeStrategije* do *ObradePomoći* i na kraju do *OdabiraIzvoraZnanja*. Ako je neki izvor znanja zaista odabran, onda upravljač prelazi do stanja *Procjena*, u kojem je prvo u stanju *AžuriranjeŠkolskePloče*. Prelazi do stanja *Spajanje* ako su pridodani objekti, i do *TraganjaUnatrag* ako su pretpostavke povučene, kada također o tome izvještava sve ovisnike.

Upravljač bezuvjetno prelazi u stanje *Zaglavljen* ako ne može nastaviti, a u *Riješeno* ako na školskoj ploči pronađe riješeni problem.

5. ZAKLJUČAK

Kostur sustava za rješavanje kriptograma logičkom je objektno orijentiranom raščlambom razdijeljen na sastavne dijelove. Kritične su operacije definirane i više ne preostaje drugo nego prihvatiti se završavanja same izvedbe ovako koncipiranog modela i ostvariti ga u praksi kroz implementaciju preostalih detalja koji dosad nisu bili uzeti u obzir. Bilo kakve promjene koje se mogu nametnuti zbog određenih okolnosti vrlo će se jednostavno dati implementirati njihovom integracijom u odgovarajući segment strukture dizajna bez potrebe za krupnim adaptacijama, koje bi možda uključivale i pomalo neizvjestan ishod. Upravo ovakvo postignuće koje je rezultat ovoga rada bilo je ostvarivo tek definiranjem u radu uvedenog objektno orijentiranog pristupa modeliranju dizajna sigurnosnog kriptografskog modela.

6. LITERATURA

Andrzej Lasota, Michael C. MacKey (1994): Chaos, Fractals, and Noise : Stochastic Aspects of Dynamics (Applied Mathematical Sciences, Vol 97).

Antoliš, K. et al (1998): Razvitak infosustava i umjetne inteligencije u prometnoj medicini, 6. međunarodno znanstveno savjetovanje "Tehnika i logistika prometa, Opatija.

Antoliš, K. et al (1997): Modularna filozofija izgradnje ekspertnih sustava i implementacija, Agronomski glasnik.

Antoliš, K. et al (1997): Metodologija planiranja i upravljanja razvojem ekspertnih sustava, Business system managment – UPS '97., Mostar.

Antoliš, K. et al (1999): Determiniranost informacijskog razvitka računalnim konceptima šeste generacije / II dio Zbornik radova, Međunarodno znanstveno savjetovanje "Promet na prijelazu u 21. stoljeće", Opatija.

Craig Larman (1997): Applying Uml and Patterns / An Introduction to Object-Oriented Analysis and Design /.

David R. C. Hill (1996): Object-Oriented Analysis and Simulation Modeling .

Derek Coleman, et al (1994): Object-Oriented Development : The Fusion Method (Prentice-Hall Object-Oriented Series).

Edward Yourdon, Carl A. Argila (1996): Case Studies in Object-Oriented Analysis and Design (Yourdon Press Computing Series).

Grbavac V. et al (1998): The concept of info function model in complex organisational system, Informatologija.

Grbavac V. et al (1999): Informacijski aspekti strategijskog pristupa izgradnji inteligentnih transportnih sustava / II dio Zbornik radova, Međunarodno znanstveno savjetovanje "Promet na prijelazu u 21. stoljeće", Opatija.

Hugh Beyer, Karen Holtzblatt (1997): Contextual Design: A Customer-Centered Approach to Systems Designs.

Jeffrey A. Hoffer, et al (1996): Modern Systems Analysis and Design.

Kim Walden, Jean-Marc Nerson (1995): Seamless Object-Oriented Software Architecture : Analysis and Design of Reliable Systems (The Object-Oriented).

Martin Fowler (1996): Analysis Patterns: Reusable Object Models (Addison-Wesley Object Technology: Addison-Wesley Object Technology Series).

Merle, P., Martin, J. (1991): Analysis and Design of Business Information Systems, Maxwell Macmillan International Publishing Group, Singapore.

Methods Jeffrey L. Whitten, et al (1997): Systems Analysis and Design.

Richard A. Layton (1998): Principles of Analytical System Dynamics (Mechanical Engineering Series).

Robert B. Angus et al (1997): Planning Performing and Controlling Projects: Principles and Applications.

Ronald J. Norman (1996): Object-Oriented Systems Analysis and Design (Prentice Hall Series in Information Management).

William W. Cotterman, James A. Senn (Editor) (1992): Challenges and Strategies for Research in Systems Development (John Wiley Series in Information Systems).

Adrese autora – *Author's address*:

Prof. dr. sc. Vitomir Grbavac
Agronomski fakultet Sveučilišta u Zagrebu
Dr. sc. Krunoslav Antoliš
GS OS RH HVU Učilište hrvatske kopnene vojske,
Zagreb, Hrvatska

Primljeno: 20. 12. 1999.