Projektovanje arhitekture

Mirjana Kostić

1035/2013

Oblasti

Odluke prilikom projektovanja arhitekture

Ahitektonski pogledi

Šabloni

Arhitektura aplikacija

Projektovanje arhitekture

Kako sistem treba da bude ogranizovan

Identifikuje glavne strukturne komponente u sistemu i odnose između njih

Prva faza u procesu projektovanja softvera

Arhitektura softvera

“Ono” što se dobija na kraju procesa projektovanja arhitekture

Reprezentacija sistema na visokom nivou

Utiče na performanse,robusnost, distribuiranost i održavanje sistema

Specifikacija sistema ne bi trebalo da sadrži bilo kakvu informaciju o dizajnu

Moguće samo kod malih sitema

Dva nivoa apstrakcije arhitekture softvera

Arhitektura u užem smislu

Arhitektura u širem smislu

Arhitektura u užem smislu

Arhitektura individualnih programa

Način na koji je program razložen na komponente

Arhitektura u širem smislu

Arhitektura kompleksnog sitema kojeg mogu činiti drugi sistemi, programi ili delovi programa.

Prednosti eksplicitnog projektovanja I dokumentovanja arhitekture softvera

Komunikacija među akterima

Projektovanje arhitekture sistema je nešto u šta treba da budu uključeni svi akteri-klijenti, developeri...

Analiza sistema

Analiza funkcionalnih i nefunkcionalnih zahteva.

Ponovna upotrebljivost

Jedna arhitektura se može koristiti na više sistema.

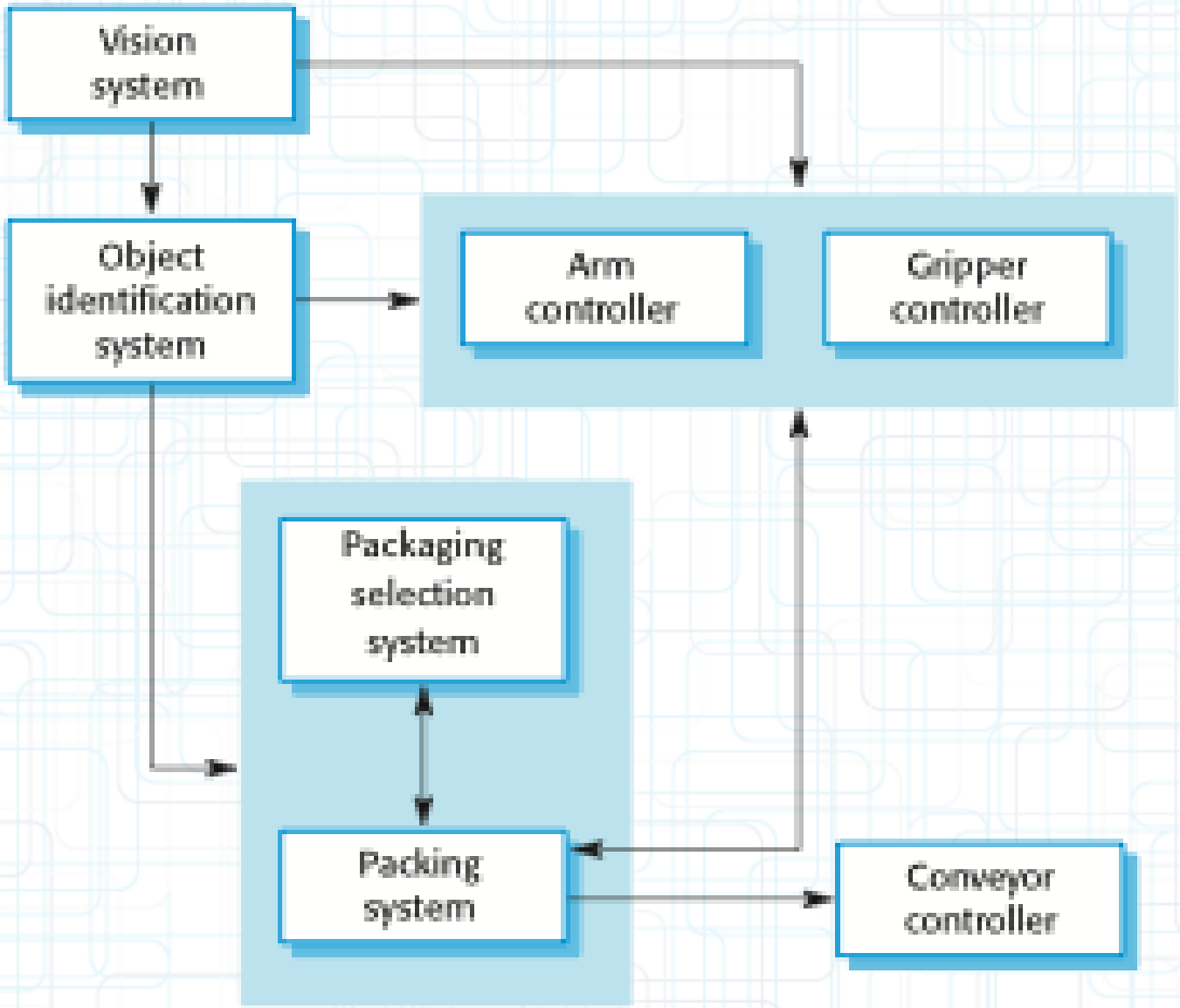
Blok dijagrami

Metoda koja se najviše koristi za reprezentaciju i dokumentaciju arhitekture sistema.

Ne pokazuje tip odnosa medju komponentama sistema a ni vidljive osobine eksternih komponenti.

Koristi se u kombinaciji sa drugim prezentacijama arhitekture

Primer blok dijagrama



Upotreba modela

Sredstvo komunikacije

Sredstvo dokumentacije

Sredstvo komunikacije

Modeli su veoma laki za razumevanje i bez detalja, pa se koriste kako bi se akterima objasnile abstraktne osobine sistema kako bi dalje učestvovali u izgradnji sistema.

Sredstvo dokumentacije

Pravljenje kompletnog modela sistema koji prikazuje različite komponetne u sistemu, njihove veze i interfejse

Odluke prilikom projetkovanja arhitekture

Proces projetkovanja se može predstaviti kao niz odluka koje donosimo u zavisnosti od vrste sistema koji se razvija i njegovih zahteva.

Neke od tih odluka mogu uticati kako na funkcionalne tako i na nefunkcionalne osobine sistema

Osnovna pitanja prilikom dizajniranja sistema

- Da li već postoji arhitektura koju možemo iskoristiti?
- Kako će sistem biti distribuiran?
- Koji stilovi dizajna bi bili prikladni?
- Na koji način će se sistem podeliti na komponente?
- Kako će se sistem podeliti na module?
- Kako će se projektovanje arhitekture oceniti?
- Koji tip arhitekture će zadovoljiti nefunkcionalne potrebe sistema?
- Kako će se proces projektovanja dokumentovati?

Karakteristike sistema od kojih zavisi arhitektura softvera

Performance

Arhitektura se projektuje tako da pronalazi kritične operacije unutar komponenti i umanji komunikaciju medju njima.Radije se koriste velike komponente.

Security

Koristi se slojevita arhitektura gde bi se krtitični delovi nalazili u najdubljem sloju.

Safety

Operacije vezane za sigurnost su smeštene ili u jednoj komponenti ili u nekoliko podkomponenti.

Availability

Arhitektura treba da bude projektovana tako da sadrzi dodatne komponente koje bi omogućile ažuriranje komponenti bez zaustavljanja sistema

Maintainability

Arhitektura treba da bude projektovana tako da se koriste komponente koje se lako mogu izmeniti

Arhitektonski pogledi

Dva problema:

1.Kakav pogled ili perspektiva je korisna prilikom projetovanja ili dokumentovanja arhitektrure?

2.Kakve notacije bi trebalo da se koriste za opis modela?

4 + 1 model pogleda

Krutchen (1995)

Četiri osnovna arhitektonska pogleda koji su povezani pomoću slučajeva upotrebe ili scenarija.

1. Logical View

Bitne apstrakcije u sistemu prikazuje kao objekte ili klase objekata.

2. Process View

Pokazuje kako je sistem u run-time sačinjen od interagujućih procesa

3. Development View

Na koji način je sistem razložen radi razvijanja.

4. Physical View

Pokazuje sistemski hardver i na koji način su komponente softvera distribuirane preko procesora u sistemu

Conceptual View

Hofmeister (2000) predlaže upotrebu sličnih pogleda sa dodatkom Konceptulanog pogleda.

Ovaj pogled može biti osnova za dekompoziciju zahteva na visokom nivou na detaljnije specifikacije, pomaže oko donošenja odluka koje komponente se mogu ponovo upotrebiti.

Šabloni

Šablon je sredstvo za prezentaciju, deljenje i ponovno korišćenje naučenog o softveru.

Šablon za projektovanje je stilizovan opis dobrog dizajna koji je isproban i testiran u različitim okruženjima.

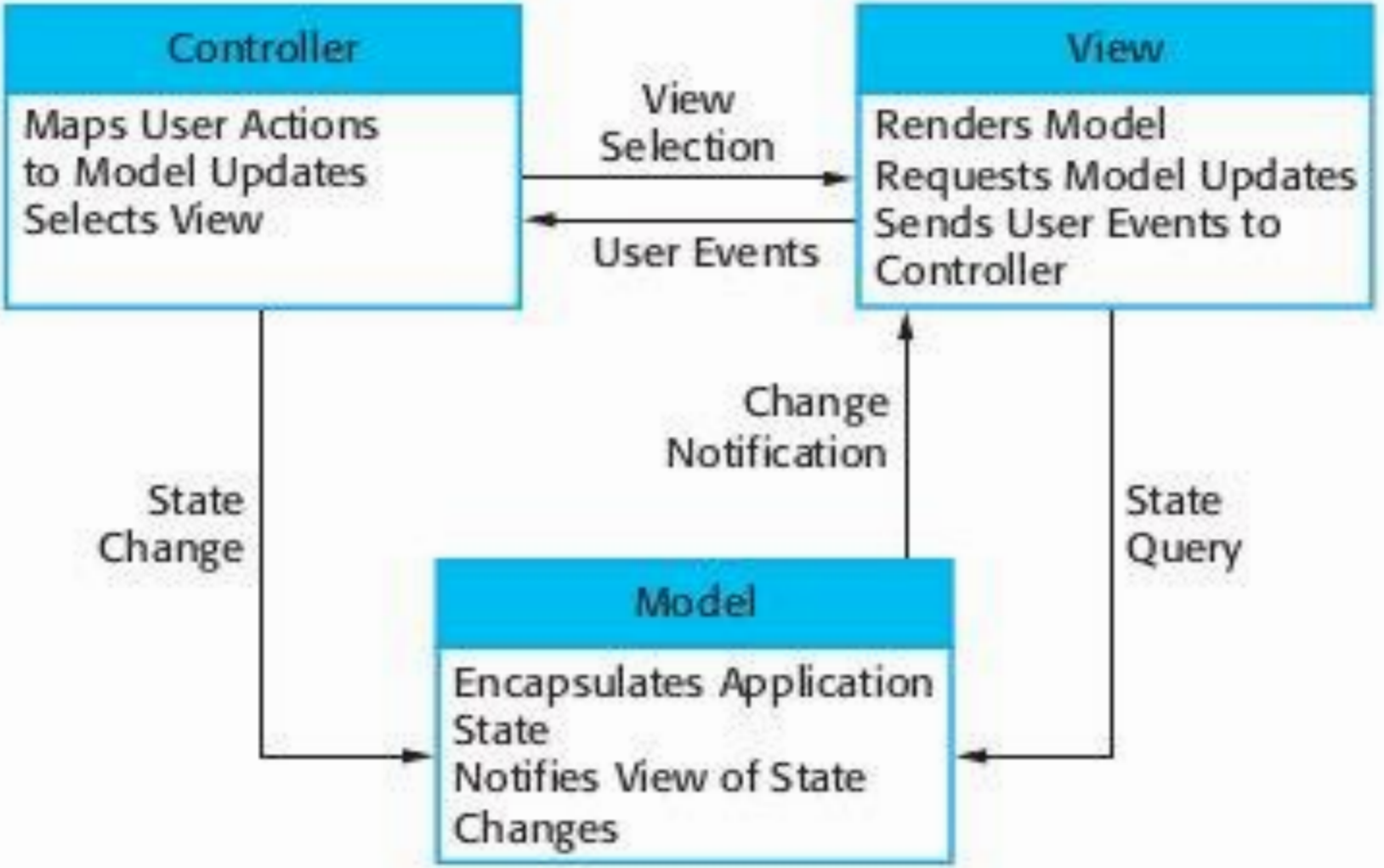
Šabloni treba da sadrže ime, kratak opis, informaciju na kojim vrstama sistema se koriste i kada, njegove prednosti i mane.

Predstavlja se grafički ili tabelarno.

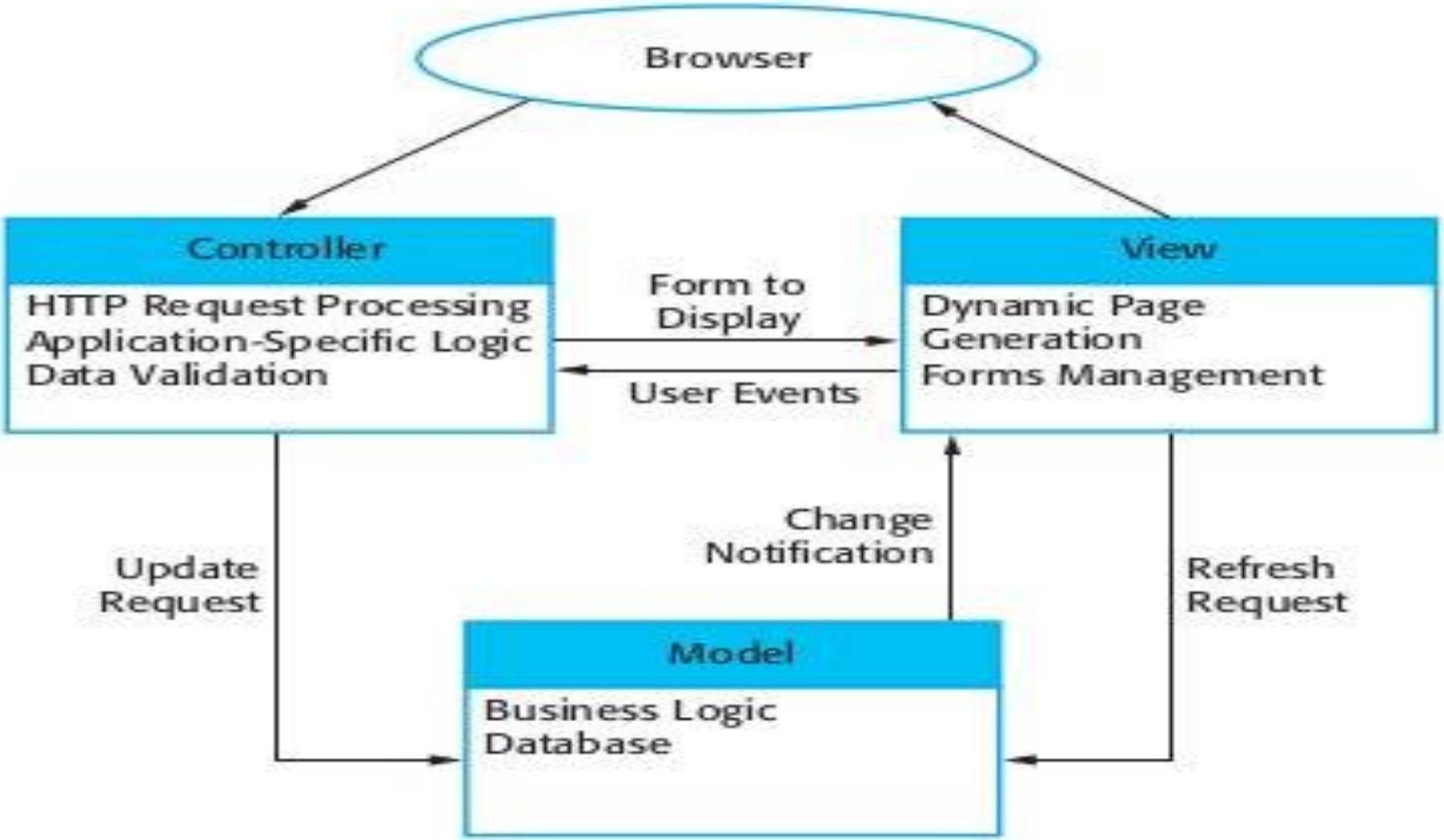
MVC šablon – primer opisa

Name	MVC (Model-View-Controller)
Description	Separates presentation and interaction from the system data. The system is structured into three logical components that interact with each other. The Model component manages the system data and associated operations on that data. The View component defines and manages how the data is presented to the user. The Controller component manages user interaction (e.g., key presses, mouse clicks, etc.) and passes these interactions to the View and the Model. See Figure 6.3.
Example	Figure 6.4 shows the architecture of a web-based application system organized using the MVC pattern.
When used	Used when there are multiple ways to view and interact with data. Also used when the future requirements for interaction and presentation of data are unknown.
Advantages	Allows the data to change independently of its representation and vice versa. Supports presentation of the same data in different ways with changes made in one representation shown in all of them.
Disadvantages	Can involve additional code and code complexity when the data model and interactions are simple.

MVC – grafički model, konceptualni pogled



MVC- moguća run-time arhitektura kada se šablon koristi za upravljanje interakcijom u Web sistemima



Slojevita arhitektura

Sistemska funkcionalnost je organizovana u zasebne slojeve a svaki sloj se oslanja samo na objekte i usluge koje nudi sloj koji se nalazi odmah ispod njega.

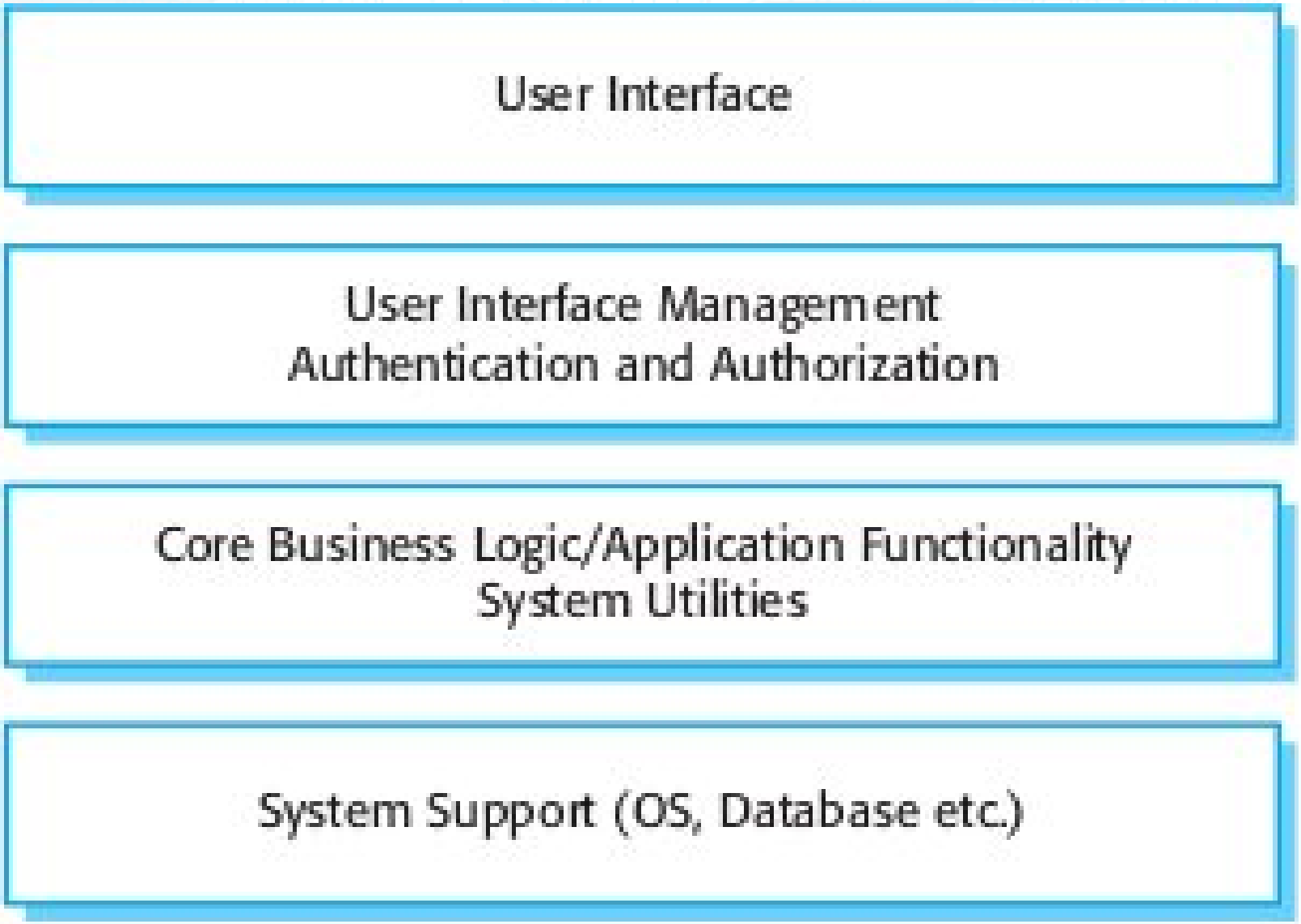
Način za postizanje nezavisnosti, prilagodjava se promenama unutar sloja

Podržava inkrementalni razvoj sofvera.

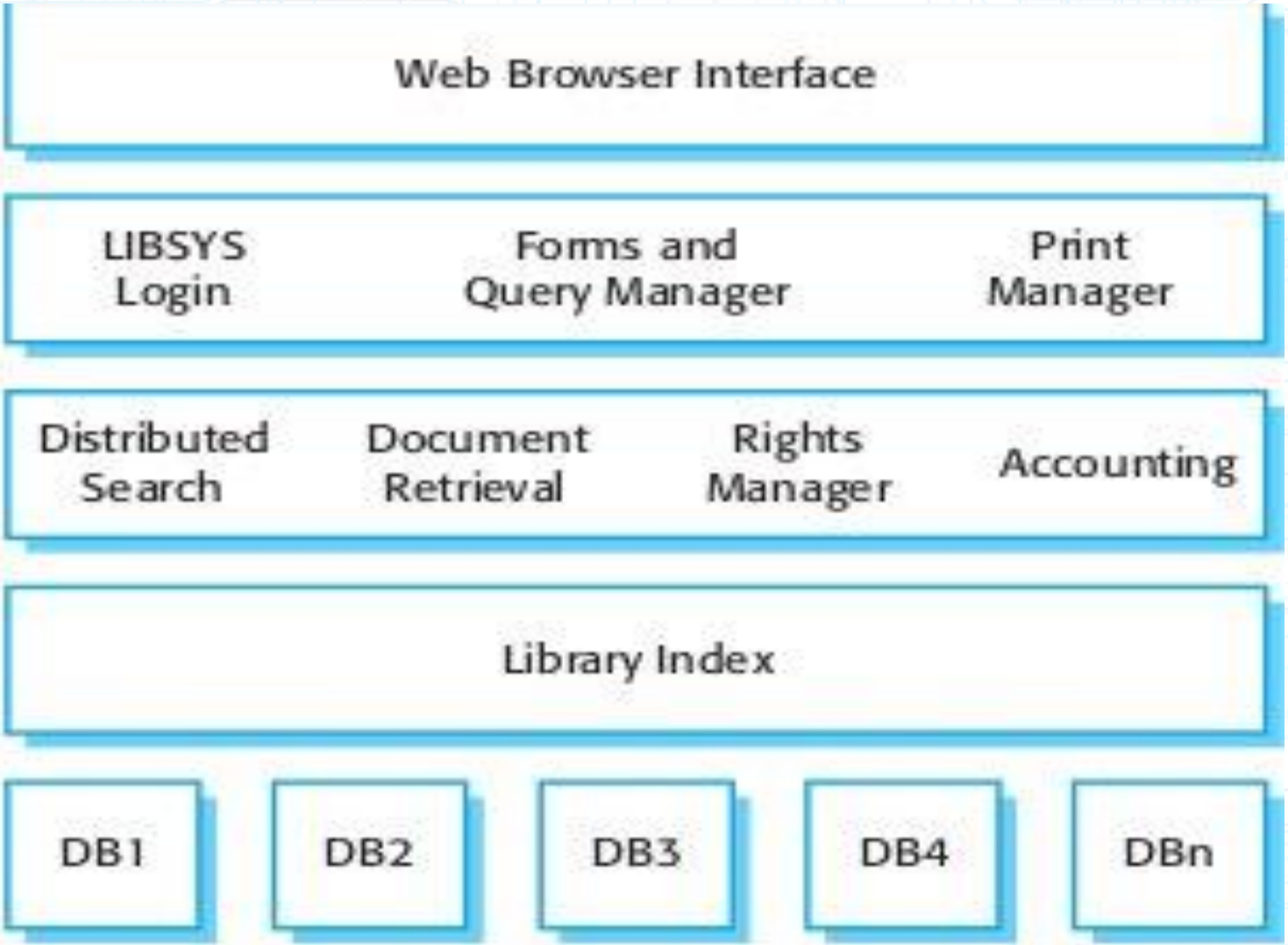
Šablon slojevite arhitekture

Name	Layered architecture
Description	Organizes the system into layers with related functionality associated with each layer. A layer provides services to the layer above it so the lowest-level layers represent core services that are likely to be used throughout the system. See Figure 6.6.
Example	A layered model of a system for sharing copyright documents held in different libraries, as shown in Figure 6.7.
When used	Used when building new facilities on top of existing systems; when the development is spread across several teams with each team responsibility for a layer of functionality; when there is a requirement for multi-level security.
Advantages	Allows replacement of entire layers so long as the interface is maintained. Redundant facilities (e.g., authentication) can be provided in each layer to increase the dependability of the system.
Disadvantages	In practice, providing a clean separation between layers is often difficult and a high-level layer may have to interact directly with lower-level layers rather than through the layer immediately below it. Performance can be a problem because of multiple levels of interpretation of a service request as it is processed at each layer.

Slojevita arhitektura- četiri sloja



Šablon slojevite arhitekture primenjen na sistem LIBSYS



Repository arhitektura

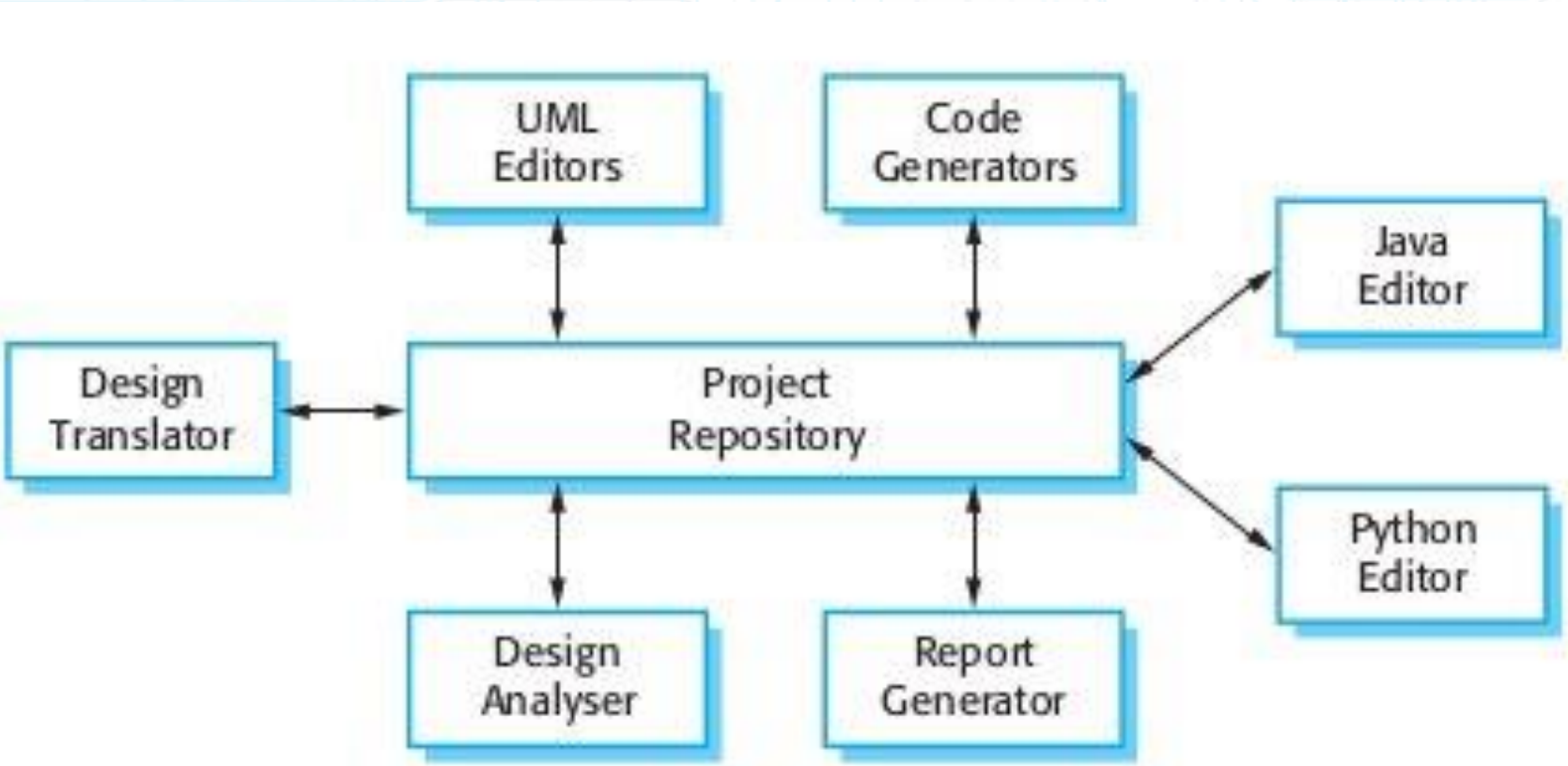
Opisuje kako skup interaktivnih komponenti može da deli veliki broj podataka.

Većina sistema koji koriste velike količine podataka su organizovani oko deljene baze odnosno skladišta

Repository arhitektura - šablon

Name	Repository
Description	All data in a system is managed in a central repository that is accessible to all system components. Components do not interact directly, only through the repository.
Example	Figure 6.9 is an example of an IDE where the components use a repository of system design information. Each software tool generates information which is then available for use by other tools.
When used	You should use this pattern when you have a system in which large volumes of information are generated that has to be stored for a long time. You may also use it in data-driven systems where the inclusion of data in the repository triggers an action or tool.
Advantages	Components can be independent—they do not need to know of the existence of other components. Changes made by one component can be propagated to all components. All data can be managed consistently (e.g., backups done at the same time) as it is all in one place.
Disadvantages	The repository is a single point of failure so problems in the repository affect the whole system. May be inefficiencies in organizing all communication through the repository. Distributing the repository across several computers may be difficult.

Repository arhitektura za IDE



Klijent- server arhitektura

Pokazuje kako se podaci i obrada distribuiraju preko niza komponenti

Glavne komponente su:

1.Skup samostalnih servera koji pružaju specifične servise poput štampanja, obrada podataka i td.

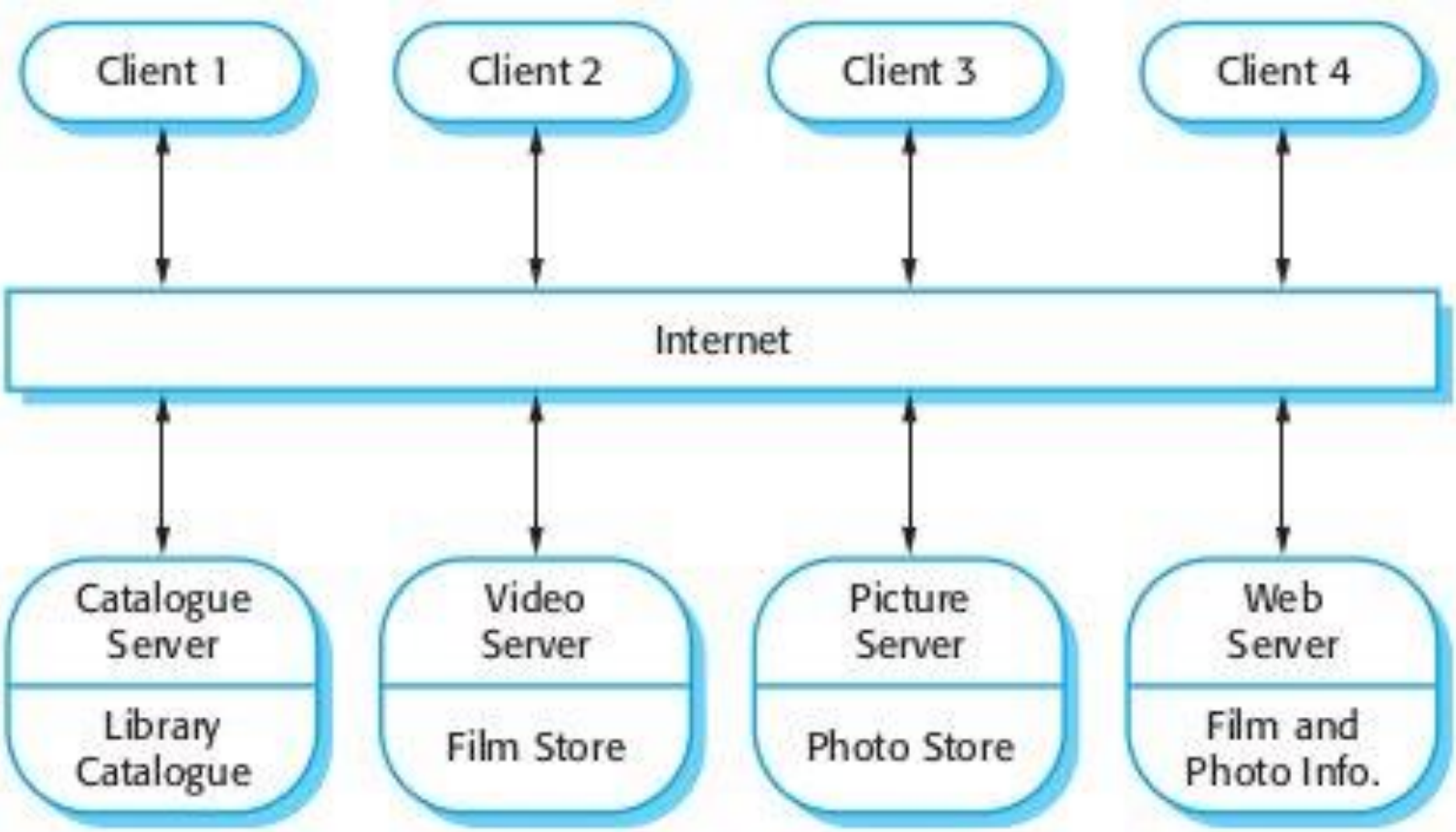
2.Skup klijenata koji pozivaju/koriste te servise

3.Mreža pomoću koje klijenti pristupaju serverima

Šablon za klijent-server arhitekturu

Name	Client–server
Description	In a client–server architecture, the functionality of the system is organized into services, with each service delivered from a separate server. Clients are users of these services and access servers to make use of them.
Example	Figure 6.11 is an example of a film and video/DVD library organized as a client–server system.
When used	Used when data in a shared database has to be accessed from a range of locations. Because servers can be replicated, may also be used when the load on a system is variable.
Advantages	The principal advantage of this model is that servers can be distributed across a network. General functionality (e.g., a printing service) can be available to all clients and does not need to be implemented by all services.
Disadvantages	Each service is a single point of failure so susceptible to denial of service attacks or server failure. Performance may be unpredictable because it depends on the network as well as the system. May be management problems if servers are owned by different organizations.

Upotreba šablona za klijent-server arhitekturu za biblioteku filmova



Pipe and filter arhitektura

Model run-time organizacije sistema gde funkcionalne transformacije obradjuju ulaze i vraćaju izlaze.

Podatak teče od jedne do druge sekvence i tom pilikom se transformiše.

Često su u upotrebi varijante ovog šablona .

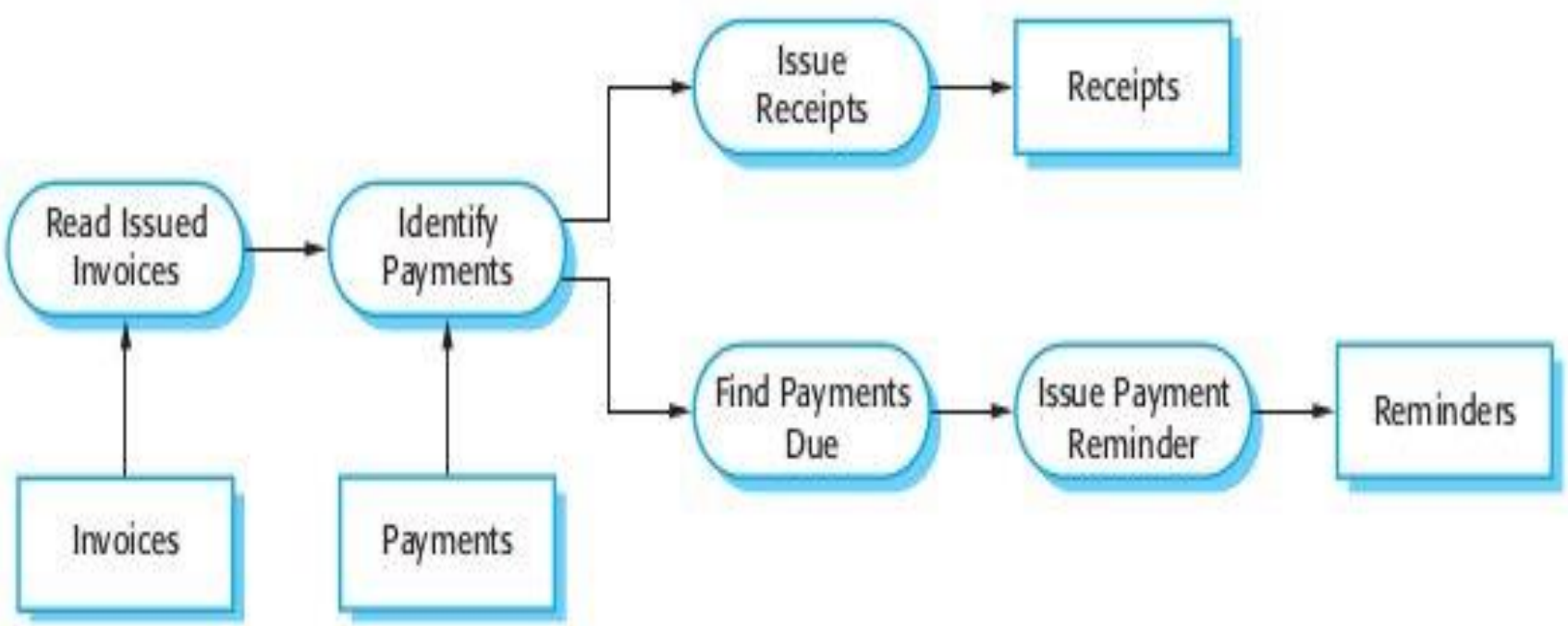
Kada su trasformacije sekvencijalne to se naziva batch sekvencijalni model.

Šablon nije pogodan za interaktivne sisteme

Pipe and filter arhitektura – šablon

Name	Pipe and filter
Description	The processing of the data in a system is organized so that each processing component (filter) is discrete and carries out one type of data transformation. The data flows (as in a pipe) from one component to another for processing.
Example	Figure 6.13 is an example of a pipe and filter system used for processing invoices.
When used	Commonly used in data processing applications (both batch- and transaction-based) where inputs are processed in separate stages to generate related outputs.
Advantages	Easy to understand and supports transformation reuse. Workflow style matches the structure of many business processes. Evolution by adding transformations is straightforward. Can be implemented as either a sequential or concurrent system.
Disadvantages	The format for data transfer has to be agreed upon between communicating transformations. Each transformation must parse its input and unparse its output to the agreed form. This increases system overhead and may mean that it is impossible to reuse functional transformations that use incompatible data structures.

Upotreba pipe and filter šablona – batch obrada



Arhitektura aplikacija

Arhitektura aplikacija se projektuje tako da zadovoljava organizacione potrebe nekog posla ili preduzeća.

Kao što preduzeća imaju mnogo toga zajedničkog, tako i njihovi aplikativni sistemi imaju sličnu arhitekturu.

Generička arhitektura aplikacija je arhitektura za određeni tip sistema i ta arhitektura se može ponovo iskoristiti ili prilagoditi prilikom razvijanja novog sistema koji je istog tipa

Načini korišćenja modela arhitekture aplikacija

Početna tačka za projektovanje arhitekture

Praćenje napretka

Organizacija posla

Sredstvo za procenu komponenti za ponovnu upotrebu

Rečnik za tipove aplikacija

Sistemi za obradu transakcija

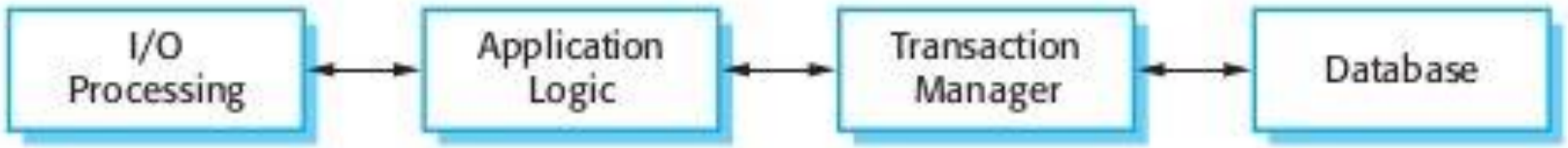
Obrađuju zahteve korisnika, prikazuju ili ažuriraju podatke iz baze.

Interaktivni sistemi u kojima korisnici asinhrono šalju zahteve.

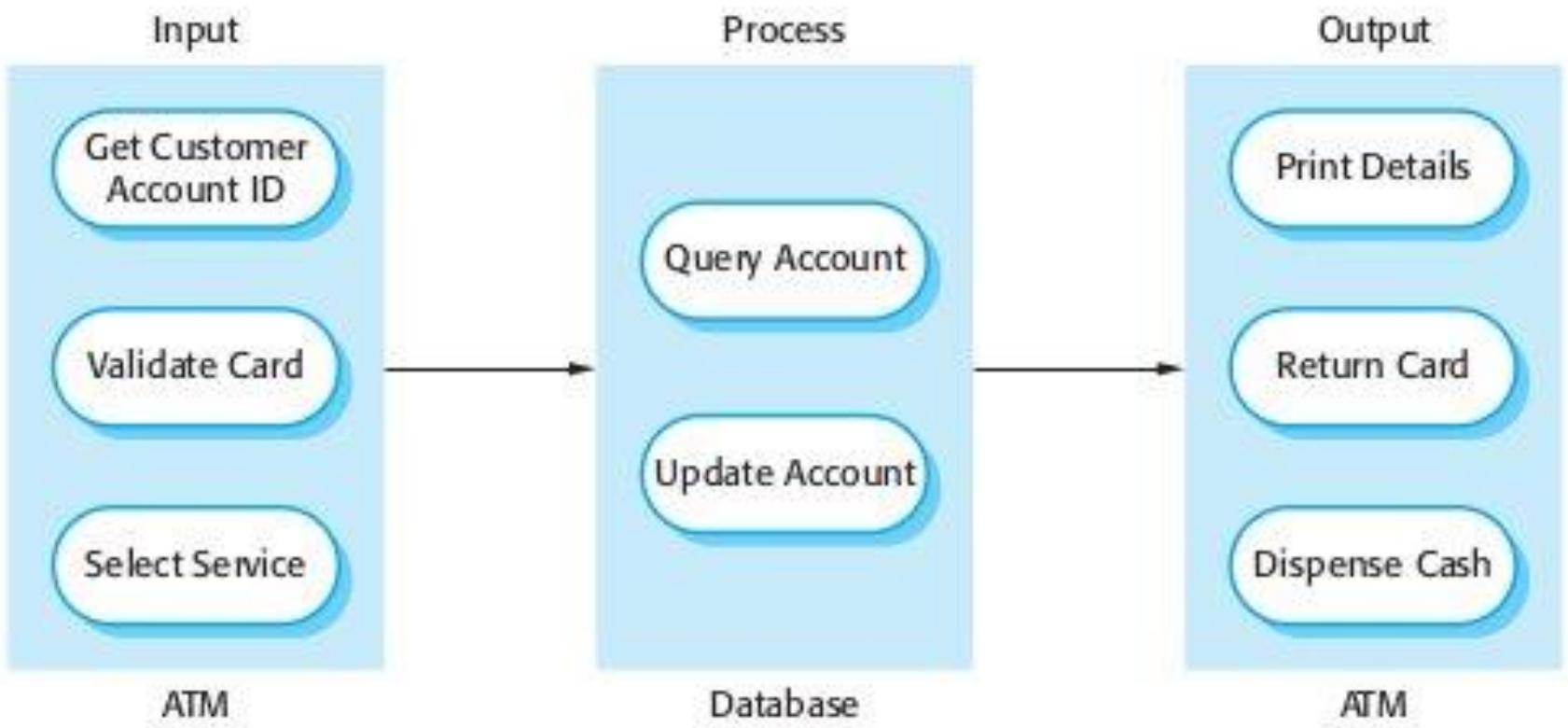
Transakcija iz ugla korisnika je bilo koja povezana sekvenca operacija koja dolazi do željenog cilja.

Može se ogranizovati kao pipe and filter arhitektura

Konceptualna arhitektura sistema za obradu transakcija



Primer korišćenja arhitekture sistema za obradu transakcija - ATM

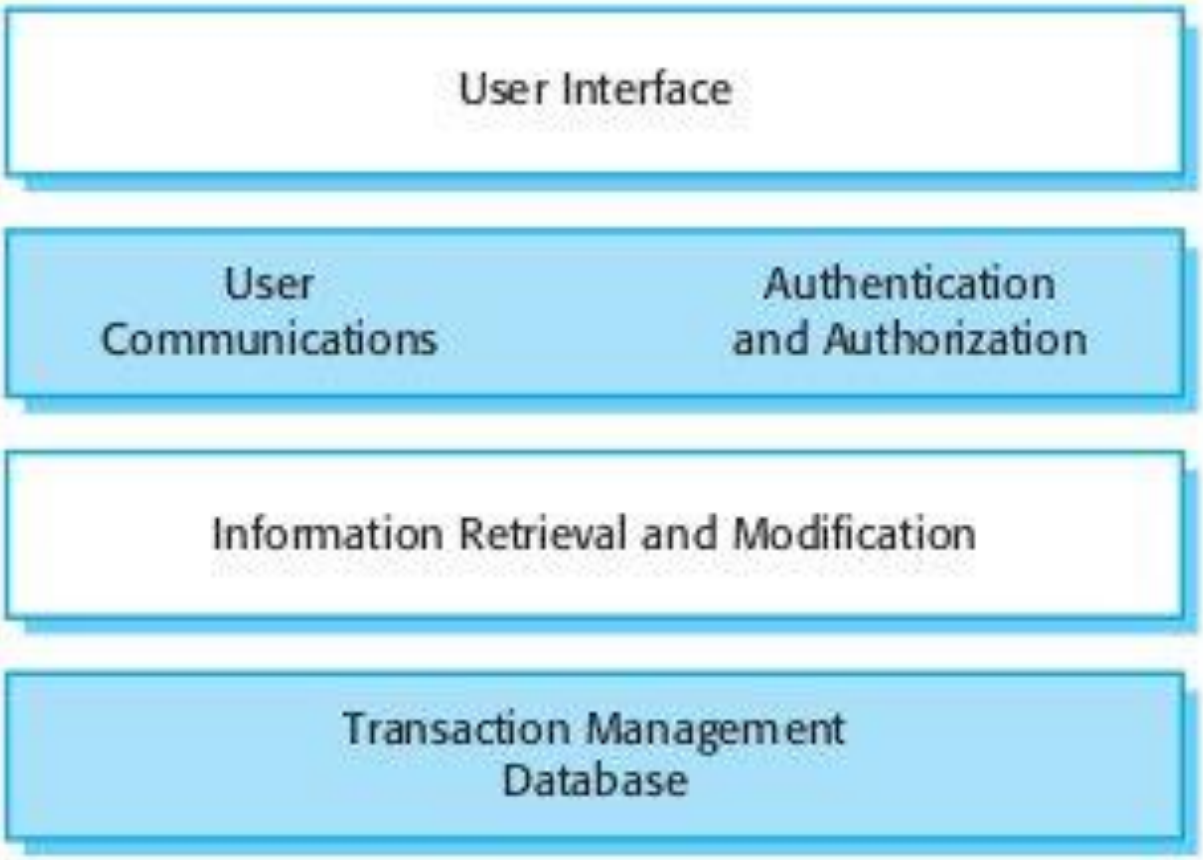


Informacioni sistemi

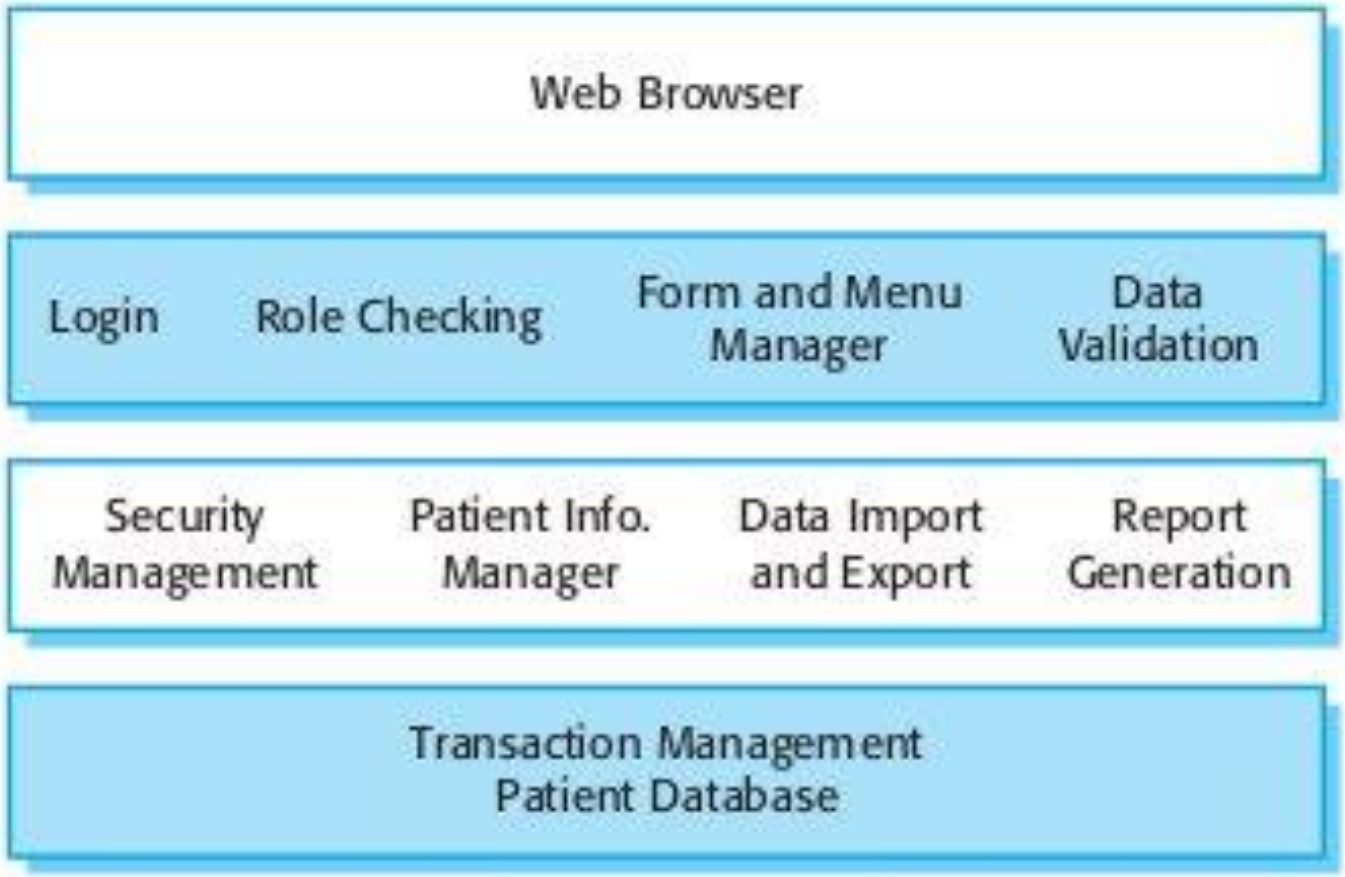
Omogućavaju kontolisan pristup velikim bazama podataka (katalog biblioteke, podaci o svim studentima na univerzitetu, kartoni svih pacijenata u bolnici)

U današnje vreme, informacionim sistemima se sve više pristupa pomoću web browser-a.

Slojevita arhitektura informacionog sistema



MHC-PMS


Web Browser
Login
Role Checking
Form and Menu Manager
Data Validation
Security Management
Patient Info. Manager
Data Import and Export
Report Generation
Transaction Management
Patient Database

Kako su današnji informacioni sistemi web-based, oni se implementiraju pomoću višestruke klijent-server arhitekture.

Korišćenje više servera omogućava visok protok podataka kao i rukavanje stotinama transakcija u minutu

Veb server je odgovoran za svu korisničku komunikaciju i za komunikaciju sa korisničkim interfejsom

Aplikativni server je odgovoran za implementaciju aplikativnih logika kao i za skladištenje i pronalaženje zahteva

Server baze podataka premešta podatke od i do baze

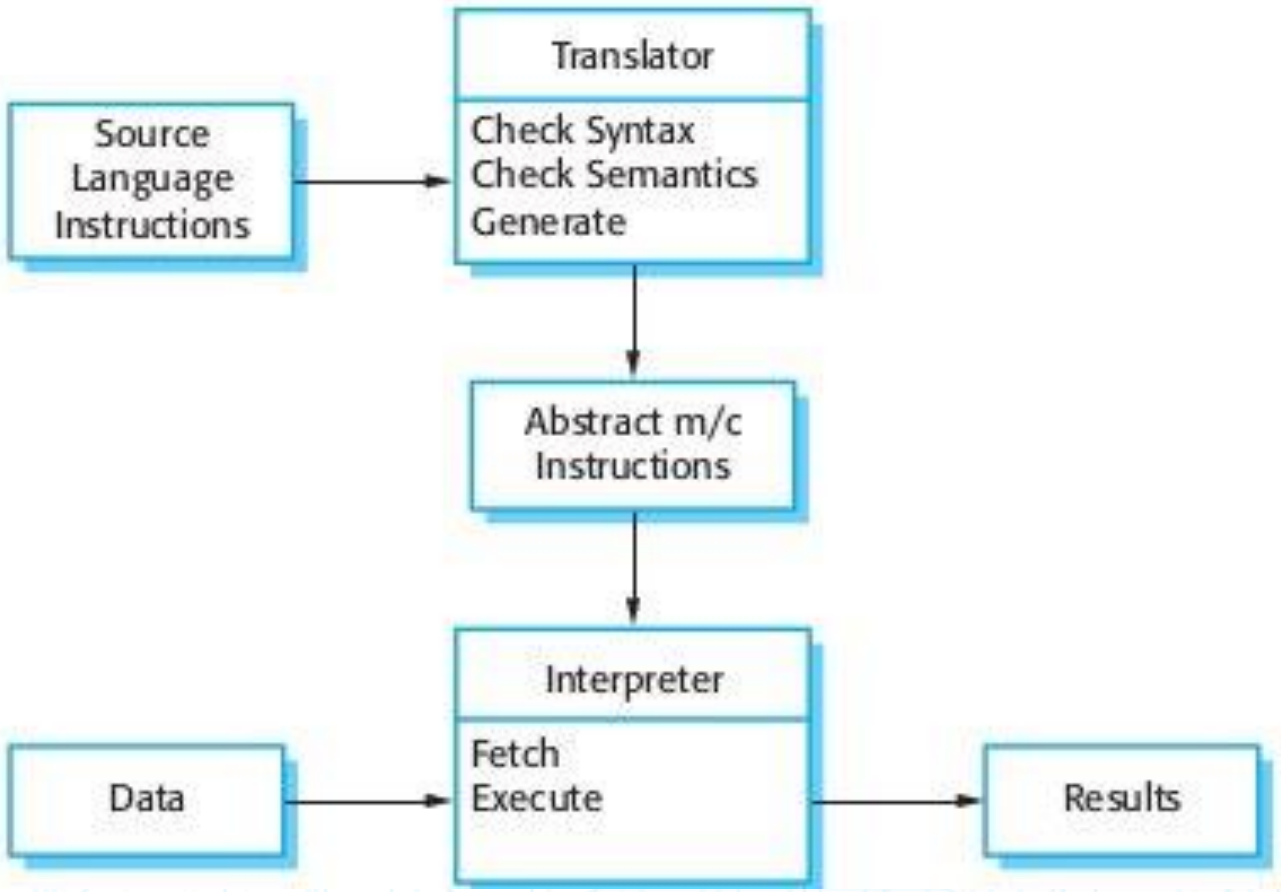
Sistem za obradu jezika

Prevodi prirodni ili veštački jezik u drugu prezentaciju tog jezika.

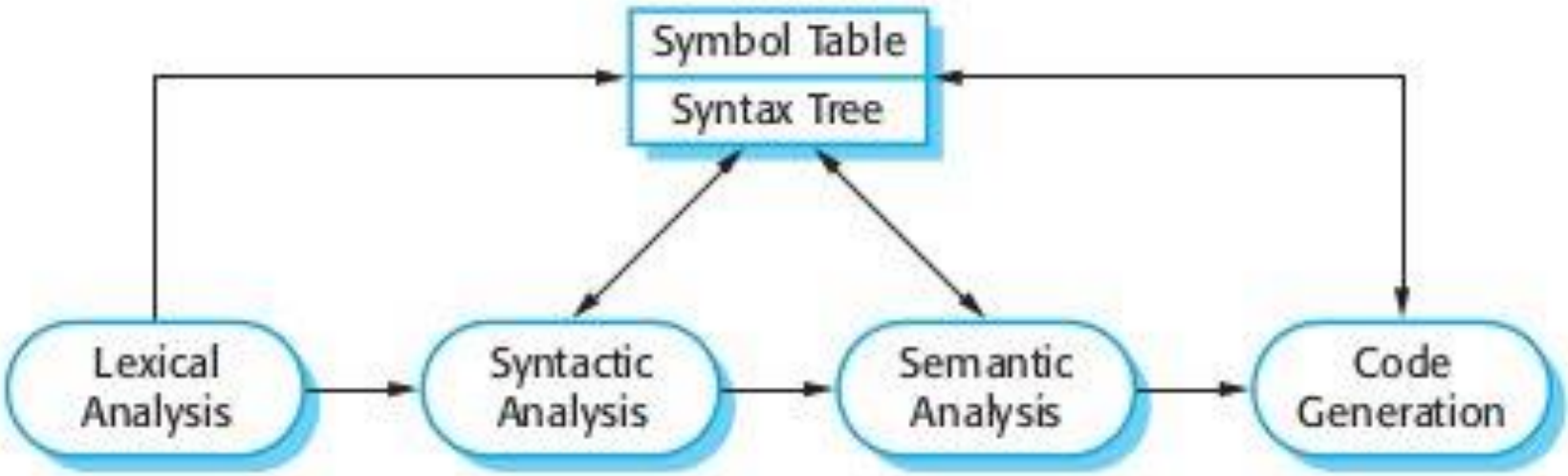
Za programske jezike može izvršiti dobijeni kod.

Kompalacija izvršnog koda u mašinski, prevodjenje sa Francuskog na Norveški

Arhitektura sistema za obradu jezika



Pipe and filter arhitektura



Repository arhitektura