Proiectarea Sistemelor Software Complexe

Curs 1 – Introducere

1.1 Arhitectura unui Sistem Software.

Obiectivul cursului este acela de a dezvolta abilitățile studenților de a proiecta sisteme software complexe; prin proiectare înțelegând-se definirea arhitecturii unui sistem software complex. În prezent termenul *arhitectură software* este utilizat în foarte multe contexte, dar nu de puține ori este folosit incorect. Există o serie de definiții pentru conceptul de *arhitectură software* [2] în acest curs vor fi discutate doar trei dintre acestea:

Arhitectura software este definită de IEEE ca fiind: *organizarea fundamentală a unui sistem, înglobată în componentele sistemului, în relațiile dintre acestea și în relațiile dintre componentele sistemului și mediul înconjurător, precum și principiile care guvernează proiectarea și evoluția sistemului*. [ANSI/IEEE Std. 1471-2000, Recommended Practice for Architectural Description of Software-Intensive Systems].

Definiția propusă de IEEE spune faptul că arhitectura surprinde structura sistemului în ceea ce privește componentele acestuia precum și modul în care aceste componente interacționează. De asemenea arhitectura unui sistem definește și regulile după care sistemul este proiectat precum și cele care definesc modul în care el poate fi modificat.

L. Bass, P. Clements și R. Kazman definesc arhitectura software ca fiind: *structura sau structurile sistemului, care constau din elemente software, proprietățile vizibile în exterior ale acestor elemente și relațiile dintre ele*. [L. Bass, P. Clements, R. Kazman, Software Architecture in Practice (2nd edition), Addison-Wesley 2003]

Această definiție se bazează pe cea IEEE, astfel este evidențiat caracterul abstract al arhitecturii (se face referire la proprietățile publice ale componentelor sistemului) și la diferitele perspective din care poate fi privită o arhitectură (structurile sistemului).

D. Garlan și M. Shaw în 1993 definesc arhitectura software: *ca mergând dincolo de algoritmi și structuri de date; proiectarea și specificarea structurii întregului sistem fiind văzută ca și o problemă distinctă. Definirea structurii unui sistem constă din definirea în linii mari a organizării sistemului și a structurilor de control globale; a protocoalelor de comunicare, sincronizare și accesare a datelor; asocierea de funcționalitate diferitelor elemente; distribuirea fizică; compunerea elementelor; scalabilitate și performanță; și selectarea metodelor de proiectare potrivite.* [D. Garlan, M. Shaw, An Introduction to Software Architecture, Advances in Software Engineering and Knowledge Engineering, Volume I, World Scientific, 1993]

Analizând definițiile de mai sus se pot scoate în evidență câteva dintre caracteristicile fundamentale ale arhitecturilor software. Aceste caracteristici fiind analizate în secțiunile următoare.

1.1.1 Arhitectura Definește Structura Unui Sistem Software

În proiectarea unei arhitecturi software divizarea aplicației în componente interconectate, module, obiecte sau orice alte unități de partiționare software, reprezintă o sarcină care ocupă o bună parte din timpul unui arhitect software. Împărțirea unei aplicații în componente trebuie realizată în funcție de cerințele și constrângerile pe care aplicația finală trebuie să le îndeplinească.

De exemplu, o cerință pentru un sistem de gestiune a informației ar putea fi ca aplicația să fie distribuită în mai multe locații, iar o constrângere ar putea fi ca anumite date să fie stocate local. Se poate observa cum cerințele impun anumite constrângeri în ceea ce privește structura aplicației și în ceea ce privește împărțirea funcționalității aplicației în componente.

În partiționarea unei aplicații, arhitectul asociază responsabilități fiecărei componente. Aceste responsabilități definind taskurile pe care o componentă trebuie să le îndeplinească în cadrul aplicației. Fiecare componentă joacă un anumit rol, dar și interacționează cu celelalte componente în vederea îndeplinirii funcționalității cerute.

Proiectarea bazată pe responsabilități (propusă de Wirls-Brock) este o tehnică de proiectare orientată pe obiecte, care presupune modelarea comportamentului unei aplicații pornind de la obiecte, responsabilități și colaborare.

Una din cele mai importante reguli de care trebuie să se țină cont în proiectarea unei arhitecturi este aceea de a minimiza dependențele între componente. Între două componente există o dependență dacă modificarea unei componente implică modificarea celeilalte. Prin eliminarea dependențelor inutile modificările sunt localizate și nu se propagă prin întreaga arhitectură. Un număr prea mare de dependențe face ca sistemul să fie greu de modificat, întreținut și testat.

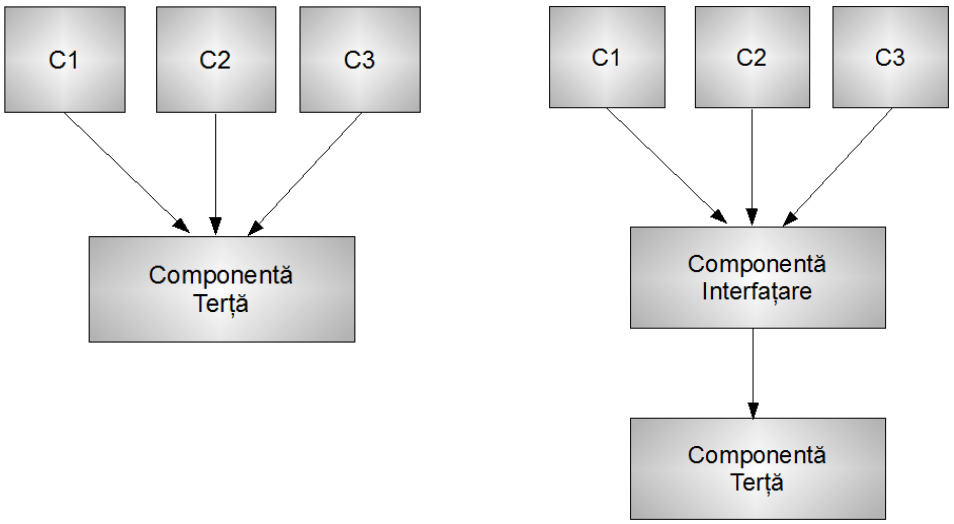

Fig. 1.1. Exemplu de dependență între componente.

În Fig. 1.1 sunt prezentate două exemple de împărțire în componente. În Fig. 1.1a este prezentată o aplicație care constă din trei componente fiecare dintre ele depinzând de o a patra componentă dezvoltată de o altă companie decât cea care dezvoltă aplicația. Faptul că fiecare componentă din

aplicație depinde de o componentă terță constituie un grad ridicat de risc și anume: daca interfața de comunicare cu componenta terță se modifică toate cele trei componente care depind de ea vor trebuie să fie modificate. Riscul poate fi însă redus dacă se proiectează o componentă care să intermedieze comunicarea cu componenta terță, așa cum este cazul în Fig. 1.1b.

1.1.2 Arhitectura Specifică Comunicarea Între Componentele Unui Sistem Software

Odată definite componentele unui sistem software este necesar să se definească modul în care aceste componente comunică între ele, și anume cum se realizează transferul de date și al informației de control între aceste componente. De exemplu componentele se pot afla în același spațiu de memorie, caz în care ele pot comunica prin apeluri de metode. Se poate însă întâmpla ca ele să se execute în fire de execuție sau chiar procese diferite, caz în care comunicarea trebuie să utilizeze mecanisme de sincronizare.

O serie de structuri care facilitează comunicarea între mai multe componente de un anumit tip și care au fost folosite cu succes au fost catalogate formând așa numitele modele de proiectare sau în engleză “design patterns”. Aceste modele reprezintă bucăți de arhitectură care pot fi reutilizate. Fiecare model are caracteristici bine cunoscute care îl fac potrivit pentru rezolvarea unui anumit tip de cerință. De exemplu, modelul de comunicare client-server are următoarele caracteristici: comunicarea între client și server este sincronă, bazată pe cerere-răspuns, iar serverul poate comunica cu unul sau mai mulți clienți printr-o interfață bine definită. Opțional clientul poate crea o sesiune pe server. Arhitectura client-server de asemenea trebuie să ofere un mecanism prin care clientul poate să localizeze serverul, un mecanism care să permite tratarea erorilor, precum și un mecanism care opțional poate să securizeze accesul pe server.

Marele avantaj al acestor modele de proiectare constă în faptul că ele au fost deja testate, iar dacă sunt folosite în mod corespunzător într-o arhitectură, practic se refolosesc cunoștințe de proiectare deja existente. Sistemele software complexe tind să folosească mai multe modele de proiectare, combinate în așa fel încât satisfac cerințele impuse arhitecturii. Un alt avantaj important al folosirii modelelor de proiectare constă în faptul că ele facilitează înțelegerea arhitecturii aplicației de către toți membrii echipei, reprezentând un mijloc de comunicare foarte eficient.

1.1.3 Arhitectura Specifică Cerințe Non-funcționale

Cerințele non-funcționale sunt acele cerințe care definesc *cum* asigură aplicația funcționalitatea cerută și nu *ce* trebuie să facă aplicația. Există trei tipuri distincte de cerințe non-funcționale:

- **Constrângeri tehnologice**: constrâng arhitectura unei aplicații prin specificarea anumitor tehnologii care trebuie folosite; de exemplu: nu avem decât programatori Java, în concluzie trebuie să folosim Java ca și limbaj de programare.
- **Constrângeri impuse de politica firmei**: acestea reduc opțiunile de proiectare pe baza unor constrângeri impuse de politica firmei. De exemplu, pentru a lărgi piața de desfacere trebuie să interfațăm cu mai multe produse.
- **Indicatorii de calitate**: definesc cerințele unei aplicații în ceea ce privește scalabilitatea, fiabilitatea, performanța etc.

Arhitectura unei aplicații trebuie să țină cont în mod explicit de toate aceste aspecte. Un arhitect software trebuie să înțeleagă cerințele funcționale ale aplicației și să proiecteze o platformă care să satisfacă în același timp și cerințele non-funcționale.

1.1.4 Arhitectura Este o Abstractizare

Pentru ca o arhitectură să poată fi înțeleasă de cât mai multă lume (membrii echipei, client, etc.) este absolut necesar ca în realizarea ei să se folosească un anumit nivel de abstractizare. Astfel, detaliile care nu sunt importante trebuie ignorate pentru a se putea pune accentul pe problemele care sunt importante din punctul de vedere al arhitecturii. Acest lucru se realizează de cele mai multe ori prin folosirea cutiilor negre ca și reprezentare a componentelor, specificând doar proprietățile vizibile din exterior ale acestor componente.

Unul din cele mai puternice mecanisme pentru descrierea unei arhitecturi este reprezentat de descompunerea ierarhică. În Fig. 1.2 este prezentat un exemplu de arhitectură care a fost proiectată utilizând descompunerea ierarhică. Astfel, la nivelul cel mai de sus aplicația constă din trei componente care interacționează. Componenta C2 este descompusă în alte două componente, C21 și C22, iar componenta C3 este descompusă în trei componente, C31, C32 și C33. Având în vedere arhitectura prezentată este destul de probabil ca cele trei componente să fie dezvoltate de echipe diferite în acest fel fiind foarte clar care sunt responsabilitățile fiecărei echipe.

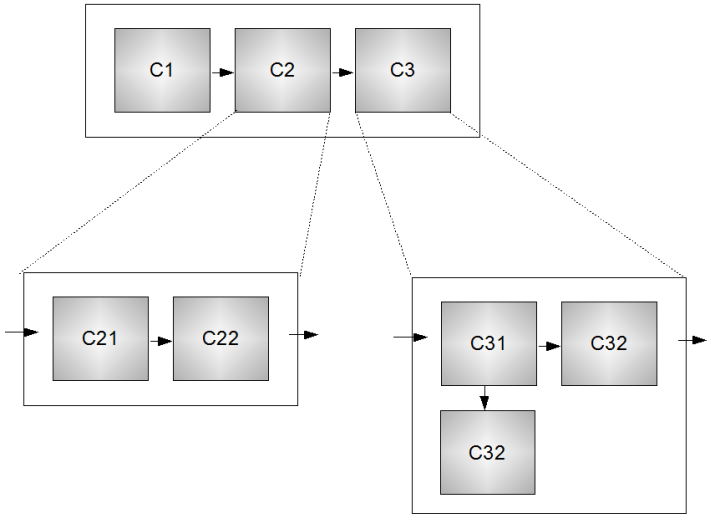

Fig. 1.2. Exemplu de descompunere ierarhică.

În exemplul din Fig. 1.2 componentele C2 și C3 au fost rafinate pentru că anumite cerințe au sugerat faptul că definirea detaliată a celor două componente este necesară. Pe de altă parte componenta C1 nu a fost rafinată pentru că structura ei internă a fost considerată nerelevantă pentru arhitectura de ansamblu a aplicației.

1.1.5 Perspectivele Unei Arhitecturi Software

Având în vedere complexitatea arhitecturii unui sistem software este evident faptul că există mai multe moduri (perspective) în care o arhitectură poate fi privită. În continuare vor fi enunțate patru astfel de moduri:

- **Perspectiva logică**: descrie elementele semnificative ale unei arhitecturi și relațiile dintre ele. Perspectiva logică surprinde structura unei aplicații utilizând diagrame de clase.
- **Perspectiva proces**: accentul se pune pe descrierea concurenței și a comunicării între componentele unui sistem software. Principalele obiective din punctul de vedere al acestei

perspective sunt descrierea componentelor multi-threading și a celor replicate, precum și a mecanismelor de comunicare sincronă și asincronă.

- **Perspectiva fizică**: surprinde modul în care principalele procese și componente sunt mapate peste echipamentul hardware. De exemplu, poate să arate cum baza de date și serviciul web sunt distribuite peste un anumit număr de mașini server.
- **Perspectiva dezvoltare**: surprinde organizarea internă a componentelor software. De exemplu, pachetele și clasele unei aplicații Java reprezintă perspectiva dezvoltare.

O altă posibilă clasificare a perspectivelor din care poate fi privită o arhitectură a fost introdusă în lucrarea "Views and Beyond"[3] în care au fost definite următoarele perspective:

- **Modul**: reprezintă o perspectivă structurală cuprinzând module de cod precum clasele, pachetele și subsistemele. De asemenea surprinde și descompunerea în module, moștenirea, asocierea și agregarea.
- **Componentă și Conector**: această perspectivă surprinde aspecte legate de comportamentul sistemului. Prin componente se înțelege obiecte, fire de execuție sau procese, iar un conector descrie modul în care interacționează componentele. Exemple de conectori sunt: socket-urile, memoria partajată sau middleware-uri (de ex.: CORBA).
- **Alocare**: arată cum procesele sunt mapate peste hardware și cum se realizează comunicare intre ele prin intermediul rețelei și/sau al bazei de date. De asemenea surprinde și codul în sistemul de management al configurației, precum și cine din grupul de dezvoltare este responsabil pentru fiecare modul.

1.2 Rolul Unui Arhitect Software

Un arhitect software are patru roluri esențiale:

- **Liant**: Un arhitect joacă mai multe roluri de liant. El este cel care face legătura între client și echipa tehnică, de cele mai multe ori împreună cu analiști de cerințe și cei de business. Face legătura între diferitele echipe implicate în proiect, el fiind punctul central pentru fiecare dintre echipe. Comunică cu managerul în vederea justificării deciziilor și a costului. Comunică cu departamentul de vânzări în vederea promovării produsului.
- **Inginer software**: Capacitatea de a proiecta este ceea ce face dintr-un inginer software un arhitect. Ingineria software este o calitate care este absolut necesară pentru un arhitect. Proiectul realizat de un arhitect trebuie să fie foarte bine documentat și comunicat. Trebuie să lucreze cu echipa de testare, documentare și release.
- **Sursă de cunoștințe tehnologice**: Un arhitect trebuie să aibă o bună înțelegere a tehnologiilor din domeniile care sunt relevante pentru tipul de produs la care lucrează. Pe baza acestor cunoștințe putând lua decizii de a folosi anumite componente third-party. Trebuie să urmărească progresul tehnologic și să înțeleagă cum noile standarde și produse pot fi exploatate cu succes în proiectul la care lucrează.
- **Managementul riscului**: Un bun arhitect trebuie să fie precaut. Trebuie să enumere și să evalueze în permanență riscurile asociate cu deciziile luate în ceea ce privește proiectarea și tehnologiile folosite. Trebuie să documenteze și să gestioneze aceste riscuri împreună cu finanțatorul proiectului și cu managerul proiectului. Trebuie să se asigure că nici un dezastru neașteptat nu va apare.

1.3 Influența Tehnologiilor Asupra Arhitecturii Unui Sistem Software

De cele mai multe ori un arhitect trebuie să ia hotărâri importante la începutul unui proiect, acest lucru face ca aceste decizii să fie greu, chiar imposibil de testat și validat. Datorită acestei imposibilități de a

testa arhitectura în primele faze ale proiectării de cele mai multe ori un arhitect se bazează în rezolvarea anumitor tipuri de probleme pe abordări care au fost deja testate și validate. Astfel, de cele mai multe ori se recurge la modele de proiectare (design patterns).

Modelele de proiectare reprezintă o arhitectură abstractă întrucât ele pot fi implementate în diverse feluri. De exemplu modelul "publică-subscrie" descrie un mecanism de comunicare slab cuplat de tipul mulți-la-mulți între componentele care publică mesaje și cele care subscriu pentru a primi mesaje. Modelul nu specifică însă cum trebuie implementate componentele care publică sau cele care subscriu și nici care este protocolul de comunicare; toate aceste informații fiind considerate detalii de implementare.

Un alt avantaj al folosirii modelelor de proiectare este faptul ca cele mai uzuale modele au fost implementate de către diferiți producători de software (Microsoft, IBM etc.), ele putând fi folosite ca și bază de plecare pentru dezvoltarea unui proiect. Astfel dacă într-un proiect este nevoie de modelul publică-subscrie, sau de un broker de mesaje, sau de o arhitectură pe trei niveluri, atunci pot fi găsite o serie de tehnologi care pot fi folosite.

Această varietate de opțiuni ridică însă și o serie de probleme pentru un arhitect, de cele mai multe ori neexistând un singur criteriu care permite compararea tehnologiilor existente, iar arhitectul trebuie să cunoască foarte bine produsele existente pentru a putea decide care dintre tehnologi este potrivită pentru un anumit proiect.

Bibliografie

[1] Ian Gorton. Essential Software Architecture. Editura Springer. 2006.
[2] http://www.sei.cmu.edu/ architecture/definitions.html
[3] Paul Clements, Felix Bachmann, Len Bass, David Garlan, James Ivers, Reed Little, Robert Nord, Judith Stafford. Documenting Software Architectures: Views and Beyond. Editura Addison-Wesley Professional. 2002