# Diagrama de componente (component diagram)

Toate diagramele cercetate anterior reflectau aspectele conceptuale de proiectare a unui model de sistem și se refereau la nivelul logic de reprezentare. Specificul reprezentării logice constă în faptul că el utilizează noțiuni care nu au personificare materială proprie. Cu alte cuvinte diferite elemente ale reprezentării logice (clase, asocieri, stări, mesaje) nu există în mod material sau fizic. Ele numai reflectă înțelegerea noastră despre structura sistemul fizic sau despre aspectele comportamentului acestui sistem.

Destinația principala a reprezentării logice constă în analiza relațiilor structurale și funcționale între elementele unui model de sistem. Pentru crearea unui sistem fizic real este necesar de a transforma toate elementele reprezentării logice în entități materiale. Pentru descrierea acestor entități este destinat aspectul fizic al reprezentării modelelor.

Pentru a explica diferența între reprezentarea logică și fizică vom cerceta procesul de elaborare a unui sistem de program. În calitate de reprezentare logică inițială a acestui sistem pot fi schemele structurale ale algoritmelor și procedurilor, descrieri a unor interfețe și baza de date conceptuală. Pentru realizarea acestui sistem este necesar de a elabora codul sursă în anumit limbaj (C++, Pascal, JAVA). Totodată în codul sursă se presupune divizarea acestui cod în module aparte.

Cu toate că codul sursă inițial reprezintă fragmente ale reprezentării fizice a unui proiect, ele nu reprezintă realizarea finală a lui. Sistemul program poate fi considerat realizat numai în caz daca el va putea executa funcțiile destinației sale. Aceasta este posibil dacă codul sursă al unui sistem va fi realizat în forma de module executate, biblioteci ale claselor și procedurilor, interfețelor grafice, fișiere a bazelor de date. Anume aceste componente sunt necesare pentru reprezentarea fizică a unui sistem.

Proiectul complet al unui sistem al programului reprezintă o totalitate de modele ale reprezentării logice și fizice care sunt coordonate între ele. În limbajul UML pentru reprezentarea fizică a unui model al sistem sunt utilizate diagramele de realizare (implementation diagrams) care includ două diagrame: diagrama de componente și diagrama de plasare.

Diagrama de componente, spre deosebire de diagramele cercetate, descrie particularitățile reprezentării fizice a unui sistem. Diagrama de componente permite determinarea arhitecturii sistemului elaborat prin stabilirea relațiilor de dependență între componentele programului, în calitate de care poate fi codul inițial, binar și executabil. În mai multe medii de dezvoltare un modul sau componenta corespund unui fișier. Relațiilor de dependență care leagă modulele sau componentele reprezintă dependența analogice celor ce au loc la compilarea codurilor surse. Elementele grafice de bază al diagramei de componente sunt componentele, interfețele și dependențele între ele.

Diagrama de componente se elaborează pentru următoarele scopuri:

* Vizualizarea structurii comune a codului sursă a unui sistem de program;
* Specificarea variantei executabile a unui sistem de program;
* Asigurarea utilizării repetate a unor fragmente ale codului sursă;
* Reprezentarea conceptuală și fizică a schemelor bazei de date.

În elaborarea diagramei de componente participă analiștii de sistem, arhitecții, și programatorii. Diagrama de componente asigură trecerea de la reprezentare logică spre realizarea proiectului în formă de cod sursă. Unele componente pot exista numai la etapa compilării codului sursei, altele – la etapa realizării lui. Diagrama de componente reflectă dependențele dintre componentele sistemului.

## Componente

Pentru reprezentarea entităților fizice în limbajul UML se utilizează termenul de componentă (component). Componenta realizează un set de interfețe și desemnează elementele reprezentării fizice a unui model. Grafic componenta se reprezintă printr-un dreptunghi cu anexe (fig. 1). În interiorul acestui dreptunghi se indică numele componentei și posibil informația suplementară. Reprezentarea acestui simbol variază în dependență de informația asociată cu componenta dată.

În metamodelul limbajului UML componentul este descendentul clasificatorului. El reprezintă organizarea în cadrul unui pachet fizic cu care el este asociat prin intermediul elementelor modelului. În calitate de clasificator componentul poate să aibă așa proprietăți ca atribute și operații.



**Fig. 1.** Componenta.

În primul caz (fig. 1, a) cu exemplarul componentei se leagă numai numele lui, în al doilea caz (fig. 1, b) se leagă în completare numele pachetului și valoarea marcată.

### Numele componentului

Numele componentului este subordonată conform regulilor generale a numelor elementelor modelului în limbajul UML și poate fi compus din orice literă, cifre și anumite semnări de punctuație. Un component poate fi reprezentat la nivel de tip sau de exemplar. Deși grafic se reprezentă la fel, regulile de notare a numelui componentului diferă puțin. Dacă componentul este reprezentat la nivelul tipului, atunci ca numele lui este scris numai numele tipului cu majusculă.

Dar dacă componentul este reprezentat la nivelul exemplarului, atunci numele este scris <numele componentului’:’numele tipului>. În urma căruia toată alinierea numelui este subliniată.

În calitate de nume simple sunt utilizate numele fișierelor executabile (cu indicarea extensiei .exe), numele librăriilor dinamice (cu extensia .dll), numele Web – paginilor (cu extensia .html), numele fișierelor de text (cu extensia .txt sau .doc) sau fișiere de adeverință (.hip), numele fișierelor bazelor de date (.db) sau numele fișierelor cu cod sursă a programelor (cu extensia .h, .cpp pentru limbajul C++, cu extensia .java pentru limbajul Java), scripturi (.pi, .asp) și altele.

Întrucât realizarea reprezentării logice a modelului sistemului depinde de programele utilizate, de aceea numele componentelor vor fi definite în dependență de particularitățile sintaxei limbajului de programare respectiv.

### Feluri de componente

Întrucât componentul ca element a realizării fizice a modelului reprezintă un modul al codului, deseori el este comentat cu indicarea sibilelor grafice adăugătoare, care marchează particularitățile concrete realizării lui. Aceste notații adăugătoare pentru adnotare nu sunt specificate în limbajul UML. Totuși utilizarea lor simplifică înțelegerea diagramei de componente și perfecționează reprezentarea ei grafică. Unele notații pentru componente sunt prezentate mai jos (fig. 2).

În limbajul UML sunt specificate trei tipuri de componente:

* În primul rând componente de regrupare, care specifică executarea de către sistem a funcțiilor sale. Așa fel de componente pot fi librării conectate dinamic cu extensia .dll
(fig. 2, a), Web – pagini în limbajul de trasare hipertextului cu extensia .html (fig. 2, b) și fișierele de adeverință cu extensia .hip (fig. 2, c).
* În al doilea rând, componente – produse de lucru. Ca regulă acestea sunt fișierele cu cod sursă a programului, de exemplu, cu extensia .h sau .cpp pentru limbajul C++ (fig. 2, d).
* În al treilea rând, componentele de executare, ce reprezintă modulele – fișierele cu extensia .exe. Ele se indică obișnuit.



**Fig. 2.** Variantele reprezentării grafice a componentelor diagramei de componente.

Aceste elemente uneori sunt numite artefacte, în așa fel subliniind conținutul lor informațional, dependent de tehnologie de realizare concretă a componentelor respective. Mai mult decât atât, elaboratorii pentru acest scop pot utiliza notații independente, deoarece în limbajul UML nu există notare strictă pentru reprezentarea grafică a notațiilor.

Un alt mod de specificare a componentelor este indicarea stereotipului componentului înaintea numelui lui. În limbajul UML pentru componente sunt specificate următori stereotipuri:

* Librărie (library) – definește primul tip a componentului, care reprezentă librărie dinamică sau statică;
* Tabel (table) – definește primul tip a componentului, care reprezentă un tabel de baze de date;
* Fişier (file) – definește al doilea tip a componentului, care reprezintă un fișier cu cod sursă a programului;
* Document (document) – definește al doilea tip a componentului, care reprezintă un document;
* Executare (executable) – definește al treilea tip a componentului, care poate fi executat în nod.

## Interfețe

Următorul element a diagramei de componente este interfața. Ele au fost descrise anterior, de aceea aici vor fi indicate numai acele proprietăți, care sunt tipice pentru reprezentarea lor în diagramele de componente. Ne reamintim că în cazul general interfața este reprezentată în formă de circumferință, care este legat cu componentul cu ajutorul relației de realizare (fig. 3, a). În urma căruia numele interfeței, care obligatoriu trebuie să fie scrisă cu majusculă «I», este scrisă alături de circumferință. Semantic linia înseamnă interfața, iar prezența interfețelor la componente înseamnă că componentul dat realizează setul de interfețe respective.



**Fig. 3.** Reprezentarea grafică interfeţelor în diagrama de componente.

Un alt mod de reprezentare a interfețelor în diagrama de componente este reprezentarea lui în forma de dreptunghi a clasei cu stereotipul «interfaţa» şi cu secțiuni posibile a atributelor și operațiilor
(fig. 3, b). Ca regulă, acest caz de notare este utilizat pentru reprezentarea structurii interne a interfeței, care poate fi importantă pentru realizare.

În urma elaborării sistemelor interfețele asigură nu numai compatibilitatea diferitor versiuni, dar și posibilitatea de introducere a schimbărilor în unele pârți a programului neschimbând alte pârți a ei. În așa fel, destinația interfețelor este mai adâncă, decât specificația interacțiunii cu utilizatorii sistemului (actorii).

Există două feluri de legătură dintre interfețe și componente. Dacă componentul realizează o anumită interfață, atunci această interfață este numită de export, deoarece acest component reprezintă în el modul de serviciu pentru altele componente. Dacă componentul utilizează o anumită interfață, care este realizată de un alt component, atunci acea interfață pentru primul component este numită de import. Particularitățile interfeței de import constă în aceea că în diagrama de componente această relație este reprezentată cu ajutorul dependenței.

## Dependenţe

În cazul general relația de dependență a fost examinată anterior. Reamintim că relația de dependență nu este asociere, dar este utilizată numai pentru reprezentarea faptului existenței acestei legături, când modificarea unui element a modelului influențează sau duce la schimbarea altui element a modelului. Relația de dependență în diagrama de componente reprezintă o linie întreruptă cu săgeată orientată de la client (element dependent) la sursă (element independent).

Relația poate indica legăturile modulelor programului la etapa de compilare și la generarea codului. În alt caz dependența poate indica existenta în componentul independent descrierea clasei, care sunt utilizate în componentul dependent pentru crearea obiectelor respective. În diagrama de componente dependențele pot conecta componentele și interfețele de import de component, dar și diferite feluri de componente între sine.

În primul caz se desenează săgeată de la component – client la interfața de import (fig. 4). Prezenta săgeții înseamnă că componentul nu realizează interfața respectivă, dar utilizează în ea procesul său de executare. În această diagramă mai poate exista și un alt component, care realizează această interfață. De exemplu, fragmentul diagramei de componente prezentat mai jos reprezintă o informație despre componentul cu numele «main.exe» dependent de interfața de import I dialog, care la rândul său este realizată de componentul cu numele «image.java». Pentru al doilea component aceeași interfață este de export.



**Fig. 4.** Fragmentul diagramei de componente cu relația de dependență.

Observăm, că al doilea component cu numele «image.java» nu poate fi reprezentat în formă de variant de adnotare (este posibil), deoarece acest component realizează interfața.

Un alt caz de relație de dependență în diagrama de componente este relația între diferite feluri de componente (fig. 5). Prezența dependenței în cazul dat înseamnă că schimbările în cod sursă (texte) a programelor sau librării dinamice vor duce la schimbarea componentului. În urma căruia caracterul schimbării poate fi indicat adăugător.



**Fig. 5.** Reprezentarea grafică a relației de dependență între componente.

Și în sfârșit, în diagrama de componente pot fi reprezentate relațiile de dependență între componente și clasele realizate în ele (fig. 6). Această informație este foarte importantă pentru coordonarea reprezentării logice și fizice a modelului sistemului. Modificările în structura descrierii claselor poate duce la schimbarea componentului. Mai jos este prezentat fragmentul dependenței, când un anumit component depinde de clasele respective.



**Fig. 6.** Reprezentarea grafică a dependenței între componente și clase.

În cazul dat, din diagrama de componente nu reiese că clasele sunt realizate de acest component. Dacă este necesar de subliniat că care-va component realizează anumite clase, atunci pentru indicarea componentului este utilizat simbolul extins al dreptunghiului. În urma căruia dreptunghiul componentului este divizează în două secțiuni cu linia orizontală. Secțiunea de sus este utilizată pentru notarea numelui componentului, iar cea de jos – pentru indicarea informației adăugătoare (fig. 7).



**Fig. 7.** Reprezentarea grafică a componentului cu informația adăugătoare despre clasele realizate.

În cadrul simbolului componentului pot fi indicate și alte elemente a notației grafice, așa ca clase (componentele nivelului de tip) sau obiectele (componentele nivelului de exemplare). În acest caz simbolul componentului este reprezentat în așa fel ca să conțină aceste simboluri adăugătoare. De exemplu, componentul realizat mai jos (fig. 8) este exemplar și realizează trei obiecte.



**Fig. 8.** Reprezentarea grafică a componentului nivelului de exemplar, ce realizează obiectele.

Obiecte, care se află în cadrul componentul – exemplar sunt reprezentate ca elemente depuse în simbolul componentului dat. Așa fel de depunere înseamnă că executarea componentului duce la executarea obiectelor respective. Cu alte cuvinte, existenta componentului în timpul executării programului cu posibilitatea de a accesa toate obiectele depuse. Referitor la accesul acestor obiecte, el poate fi adăugător specificat cu ajutorul specificatorului de vizibilitate ca și la vizibilitatea pachetelor. Sensul vizibilității poate fi diferit pentru diferite feluri de pachete.

De exemplu, pentru pachetul programului cu cod sursă (text) vizibilitatea înseamnă posibilitatea introducerii schimbării în texte a programului respectiv cu recompilarea lor. Pentru componente cu codul programului executabil vizibilitatea poate caracteriza posibilitatea lansării spre execuție a componentului respectiv sau chemarea operațiilor sau metodelor realizate în el.