{PROQRAMLAŞDIRMA}

STEP IT ACADEMY

UNITIED MODELING LANGUAGE

Dərs №1
UML

Mündəricat

1. Proqram təminatının tərtibatı zamanı yaranan çətinliklər

Bugünkü gündə yaradılan sitemlərin çətinlikləri çox artmışdır.

Bu çətinliklərin özünü göstərməsi yalnız ölçü və funksionallığın artması ilə əlaqəli deyil. Bir çox problemlər istifadəçinin istəklərinin tez-tez dəyişməsi və sitemlərin keyfiyyətinə olan tələblərin artması ilə yaranır.

Çətinlik probleminin tərtibatçı kollektivinin böyüdülməsi, onların ixtisaslaşdırılması, təmiz halda iş bölgüsünün aparılması kimi ənənəvi həlli yolları, nəticələrin razılaşdırılması və hazır sistemlərin yığılmasında daha böyük çətinliklərə gətirib çıxardır.

Çətinlik üzərində qələbəyə aparan ciddi addım, obyekt-yönümlü proqramlaşdırmanın (OYP) yaranması oldu. Lakin bu sadəcə bir addım idi. OYP – nin yaranması tərtibatçı və istifadəçilər arasında mövcud olan qarşılıqlı anlaşılmazlıq, dəyişən tələbatların olduğu şəraitdə tərtibatla qeyri-effektiv idarəetmə, işlərin yerinə yetirilməsi prosesində dəyişiklərin nəzarətsizlik, tərtibat məhsullarının keyfiyyətinin qiymətləndirilməsində subyektivlik və s. kimi problemləri aradan qaldırmadı.

2. OYP – yaranma səbəbləri

İlk kompüterlərdə proqram yazılması üçün əvvəlcə maşın kodlarından, sonra isə assembler dilindən istifadə olunurdu. Lakin, bu dil, bugünkü standartlara uyğun gəlmir. Proqramların çətinlik dərəcələri artdıqca məlum oldu ki, tərtibatçılar, onların proqramlarının sazlanması və təkmilləşdirilməsi üçün lazımi olan informasiyanı yadda saxlamaq iqtidarında deyildir. Registrlarda hansı dəyərlər saxlanılır? Artıq bu cür adla olan dəyişən varmı? Növbəti koda idarəetməni verməkdən öncə hansı dəyişənləri inisializasiya etmək lazımdır?

Bu problemləri qismən də olsa, Fortran, Kobol, Alqol kimi ilk yüksək səviyyəli dillər həll edə bildi. Lakin, proqramların çətinlik dərəcələrinin artması davam edirdi, yeni layihələr yaranırdı və burada da tərtibatçılar bütün detalları yadda saxlamaqda çətinlik çəkirdi. Layihələr üzərində, proqramçı komandaları işləməyə başladı.

Proqram təminatı (PT) hissələrinin əhəmiyyətli qarşılıqlı asılılığı, digər PT-nı material obyektlərin layihələndirilmə tipində yaratmağa mane olur. Məsələn: Bina, avtomobil və elektrik avadanlıqları "sıfırdan" hazırlanması gərəkməyən hazır komponentlərdən yığılır. PT-nın dəfələrlə istifadəsi – məqsəddir və bu məqsədə çatmaq üçün hər zaman çalışırlar və çox nadir hallarda buna nail olurlar. Proqram sistemindən müstəqil fraqmentləri çıxartmaq çox çətindir. OYP isə bu işi asanlaşdırır.

3. Baza anlayışlar

Özlüyündə obyekt-yönümlü dilin istifadəsi obyekt-yönümlü proqramların yazılmasına məcbur etmir, baxmayaraq ki, onların tərtibatını asanlaşdırır.

Prosedur proqramlaşdırmadan fəqli olaraq OYP-nı effektiv istifadə etmək üçün məsələləri başqa bir üsulla müzakirə etmək lazımdır.

Təbii dillərə qarşı belə bir iddia vardır ki, fikrin səsləndiyi dil, düşüncəni istiqamətləndirir. Həm kompüter, həm təbii dillər üçün: dil düşüncələri istiqamətləndirir, lakin, onlara təyinat vermir anlayışı daha doğrudur.

Analoji olaraq, obyekt-yönümlü texnika proqramçını, digər üsullara həlli mümkün olmayan problemin həlli üçün yeni hesablama qüvvəsiylə təmin etmir. Lakin, obyekt-yönümlü yanaşma məsələni daha sadə edərək, onu daha təbii formaya gətirib çıxardır. Bu, problemlə böyük proqram sitemlərinin idarə edilməsinə gözəl şərait yaradan üsulda rəftar etməyə imkan verir.

OYP-nı çox vaxt proqramlaşdırmanın paradiqması kimi adlandırılar. Proqramlaşdırmanın paradiqması – konseptualizasiya üsuludur, o hesablamaların necə aparılmasını müəyyən edir və kompüterin gördüyü işin necə strukturlaşdırılmalı və təşkil edilməli olduğunu deyir.

Əməliyyatın ayrı-ayrı, bir-birinə strukturla bağlı olan hissələrə ayrılması prosesi – **dekompozisiya** adlanır. Prosedur dekompozisiyası zamanı məsələdə alqoritm-

lər və onlarla emal olunacaq məlumat strukturları, məntiqi dekompozisiyada isə - ayrı-ayrı anlayışları bir-birinə bağlayan qaydalar seçilir. Obyekt-yönümlü dekompozisiya zamanı tapşırıqda siniflər və obyektlərin, bu siniflərin obyektlərinin bir-biriylə qarşılıqlı təsir üsulları seçilir.

PT-da olan çətinliyin aradan qaldırılması üçün əsas üsul - **abstraktlaşdırmadır**, yəni, proqramın fraqmentinin məntiqi qiymətinin onun reallaşdırma problemindən ayırılması bacarığıdır.

Abstraktlaşdırma – obyektin digər obyektlərdən fərqlənən mühüm xarakteristikalarının seçilməsidir və bu cür göstərilmiş obyektin, gələcəkdə nəzərə alınma nöqteyi nəzərindən özəllikləri müəyyən olunur. Abstraklaşdırma mühüm özəlliklərlə, mühüm olmayan özəllikləri bir-birindən ayırmağa imkan verir. Abstraktlaşdırma bəzi obyektlərin, onları digərlərindən fərqləndirən və obyektin müşahidəçi nöqteyi nəzərindən konseptual sərhədlərini dəqiqcəsinə qeyd edən mühüm xarakteristikalarını müəyyən edir.

Modulluq — adların məcmusunun və onunla bağlı qiymətlərin idarə edilməsi üçün yaradılan təkmilləşdirilmiş metoddur. Modulun mahiyyəti ad mühitinin iki hissəyə ayrılmasından ibarətdir: açıq (public) hissə moduldan kənarda da mövcuddur, bağlı (private) hissə isə yalnız və yalnız modulun daxilində mövcuddur.

İerarxiya — ranjirə edilmiş (nizamlanmış) abstraksiya sistemidir.

4. Mövcud olan metodologiyaların icmalı

4.1 Prosedur – yönümlü proqramlaşdırmanın metodologiyası

İlk elektron hesablama maşınlarının və ya kompüterlərin yaranması texniki hesablamlamaların inkşafında yeni mərhələ açdı. İlk baxışdan, elementar əməllərin ardıcıllıqlarını tərtib etmək kifayət idi, hansı ki, onların hər birini kompüterə aydın olan isntruksiyalara çevirmək olar və bununla hər bir hesablama məsələləri həll oluna bilər. Bu fikir o qədər inandırıcı idi ki, uzun müddət proqram tərtibatı zamanı bütün proses onun üzərində dominasiya edirdi. Ayrı-ayrı hesablama əməliyyatlarını müvafiq proqram kodlarına çevirə bilən xüsusi proqramlaşdırma dilləri yarandı.

Bu dillərə Assembler, C, Pascal və s. digər dillər aiddir.

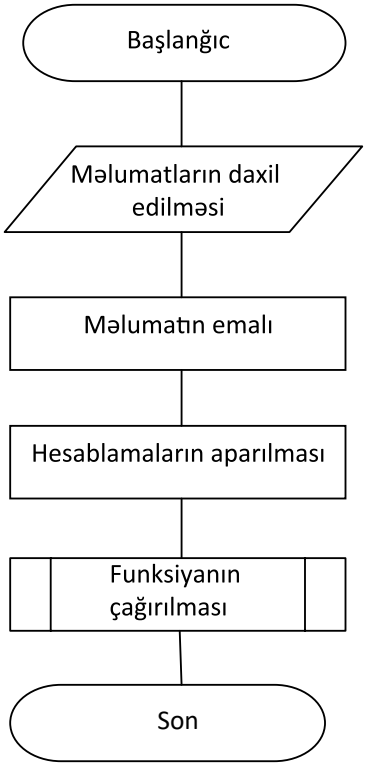
Bu proqramlaşdırma tərtibatı metodologiyasının əsası, proqram kodunun strukturunun prosedur və alqoritmik təşkili idi. Bu hesablama məsələlərinin həlli üçün o qədər təbii idi ki, heç kimdə bu cür yanaşmanın məqsədyönlü olmasına şübhə yaratmırdı. Bu metodologiyanın əsas anlayışı **alqoritm** anlayışı idi. Bu anlayış altında ümumilikdə qarşıya qoyulmuş məqsəd və həll olunması vacib olan məsələlərin həlli üçün əməllərin dəqiq müəyyən olunmuş ardıcıllıqlarını yerinə yetirilməsi başa düşülür.

Alqoritm termini orta əsrlərdə yaşamış riyaziyyatçı Əl – Xarəzminin adı ilə bağlı olduğu hesab edilir. O 825 –ci ildə riyazi hesab əməllərinin onluq sistemində hesablanmasını təsvir etmişdir.

Bütün riyaziyyat tarixi, müəyyən dövr üçün aktual olan məsələlərin həlli üçün özünün bu və ya digər alqoritmlərinin tərtibatıyla sıx əlaqəlidir. Həmçinin, alqoritm anlayışı özü, müəyyən olunmuş nəzəriyyənin - alqoritmlərin ümumi xüsusiyyətlərinin tədqiqi ilə məşğuldur - **alqoritm nəzəriyyəsinin** predmetinə çevrilmişdir.Vaxt keçdikcə bu nəzəriyyənin məzmunu o qədər abstraktlaşdı ki, müəyyən olunmuş nəticələri yalnız mütəxəssislər başa düşürdü. Buna görə proqramlaşdırma dilləri bir müddət alqoritmik adlanırdı, proqramların dsənədləşməsi üçün olan ilk üsul isə **alqoritm blok-sxemi** adını almışdı.

Təcrübənin tələbləri isə, müəyyən funksiyaların həllinin təyin olunmasını və ayrı-ayrı məsələlərin həllini heç də həmişə tələb etmirdi. Proqramlaşdırma dilində yeni - prosedur anlayışı yarandı və özünü təstiq etdi. Bu anlayış kompüterdə məsələ həlli üçün tədbiq olunan alqoritm anlayışını konkretləşdirirdi. Alqoritm kimi, prosedur da ayrı-ayrı məsələ həlli üçün yönəlmiş əməl və əməliyyatların bitmiş ardıcıllığını özündə ifadə edir. Proqramlaşdırma dillərində prosedur adını alan xüsusi sintaksis konstruksiya yarandı. Vaxt keçdikcə böyük proqramların tərtibatı ciddi problemə çevrildi və onların daha kiçik fraqmentlərə bölünməsini tələb etdi. Bu cür bölünmənin təməli kimi, proqramın ayrı-ayrı hissələri və **modulları**

məsələlərin məcmusunun həlli üçün müəyyən prosedurların məcmusundan ibarət **prosedur dekompozisiya** oldu . Prosedur proqramlaşdırmanın əsas özəlliyi onadan ibarətdir ki, proqramın həmişə zaman üzrə başlanğıcı və ya başlanğıc proseduru və həmçinin sonu var. Burada bütün proqram vizual olaraq qrafik primitivlərin və blokların yönləndirilmiş ardıcıllığı şəklində təsvir oluna bilər.



Proqramın prosedur ardıcıllığı şəklində qrafik təsviri

Bu cür proqramların vacib xüsusiyyətlərindən biri, sonrakı prosedurun fəaliyyətinin başlanması üçün əvvəlki prosedurun bütün əməllərinin bitirilməsinin zəruriliyidir. Hətta, bir prosedur daxilində əməllərin sıralanmasının dəyişməsi, məsələ həllinin aralıq nəticələrindən asılı olaraq hesablama prosesinin şaxələnməsi üçün proqramlaşdırma dilinə xüsusi şərti **if-else** və **goto** kimi operatorların daxil edilməsini tələb etdi.

Şərti operatorların və şərtsiz keçid operatorlarının yaranması və istifadəsi proqramlaşdırma üzrə mütəxəssislərin kəskin diskussiyalarına gətirib çıxartdı. Məsələ burasındadır ki, şərtsiz keçid operatoru **goto** – nun nəzarətsiz tətbiqi, kodun anlaşıqlı olmasını ciddi çətinləşdirə bilər. Bir proqram fraqmentindən digər fraqmentə olan çoxsaylı keçidləri və ya daha pisi, son proqram operatorlarından onun ilk operatorlarına qayıtmasını nəzərə alaraq, müvafiq proqramları spaqetti ilə müqayisə etməyə başladılar. Vəziyyət o qədər dramatik idi ki, ədəbiyyatda **goto** operatorunu proqramlaşdırma dillərindən çıxartmaq üçün çağırışlar səslənirdi. Məhz o vaxtlardan **goto** – suz proqramlaşdırma, proqramlaşdırmda yaxşı stil hesab olunur.

Nəzərdən keçirilmiş ideyalar proqram tərtibatının və kod yazılmasının prosesinə baxış sitemlərinin qurulmasına töhvə vermişdir və buna s**truktur proqramlaşdırmanın metodologiyası** adı verilmişdir. Bu metodologiyanın əsası, proqram sisteminin prosedur dekompozisiyası və

ayrı-ayrı modulların yerinə yetirilən prosedurlarının cəmi halında təşkil edilməsidir. Bu metodologiya çərçivəsində proqramların eniş layihələndirilməsi və ya "yuxarıdan aşağıya" proqramlaşdırması inkşaf etməyə başladı. Struktur proqramlaşdırma ideyalarının məşhur olduğu dövr 70-ci illərin əvvəli, 80-cı illərin sonuna düşür. Proqram kodunun strukturlaşdırılmasında köməkçi vasitə kimi hər sətrin əvvəlində boşluq qoyulmasına məsləhət görülmüşdür ki, bu da daxil edilmiş dövrləri və şərt operatorlarını ayırmalı idi. Bütün bunlar proqramın özünün anlaşıqlı və oxuna bilinən olması üçün nəzərdə tutulmuşdu. C dilində proqram yazılışının bu özəlliyini illustrasiya edən misal:

```
#include <iostream>
#include <stdlib.h>
#include <time.h>
using namespace std;
void main()
{
	const int n=20;
	int Mas[n];
	srand(time(0));
	for(int i=0;i<n;i++)
	{
		Mas[i] = rand()%100;
		cout<<Mas[i]<<" ";
	}
	for(int i=0;i<n;i++)
	{
		bool sort = true;
		for(int k=0;k<n-1;k++)
		{
			if(Mas[k]>Mas[k+1])
			{
				int n = Mas[k];
				Mas[k] = Mas[k+1];
		Mas[k+1]=n;
				sort=false;
			}
```

```
            }
            if(sort==true) break;
        }
        cout<<endl<<endl;
        for(int i=0;i<n;i++)
        {
            cout<<Mas[i]<<" ";
    }
}
```

Bu dövrdə proqram tərtibatının əsas çətinliyi kimi, onun ölçüləri hesab edilirdi. Proqramın çətinlik dərəcəsi, proqram kodunun sətirlərinin sayı ilə qiymətləndirilirdi. Düzdür, bu zaman müəyyən qaydalara riayət etməli olan sətirlərin sintaksisləriylə bağlı bəzi ehtimallar edilirdi. Proqramların tərtibatının ümumi ağırlığı xüsusi ölçü vahidiylə qiymətləndirilirdi – "adam-ay" və ya "adam-il". Proqramçının peşəkarlığı isə bilavasitə bir ay müddətində yazıb, düzəliş edə biləcəyi proqram kodunun sətirlərinin sayı ilə bağlı olurdu.

4.2. Obyekt – yönümlü proqramlaşdırmanın metodologiyası

Vaxt keçdikcə vəziyyət xeyli dəyişməyə başladı. Məlum oldu ki, ilkin proqramlaşdırma mərhələlərində proqram tətbiqlərinin hazırlanmasının ağırlığı həqiqətən də sərf olunan səylərdən aşağı qiymətləndirilirdi, bu da əlavə xərc və proqram hazırlığının müddətinin son müddətinin uzadılmasına gətirib çıxarırdı. Proqram tətbiqlərinin tərtibatı prosesində sifarişçinin funksional tələbləri dəyişirdi, bu da proqramçıların işinin bitməsi müddə-

tini uzadırdı. Proqramların ölçüsünün böyüməsi daha çox proqramçıların cəlb olunmasına gətirirdi ki, bu da öz növbəsində onların razılaşdırılmış işinin təşkili üçün əlavə resurslar tələb edirdi. Lakin, kompüterlərin istifadəsi aksentinin sürüşməsi nəticəsində yaranan keyfiyyət dəyişiklikləri də mühüm rol oynadı. Əgər "böyük maşınlar" dövründə proqram təminatının əsas istehlakçısı iri müəssisələr, şirkətlər və idarələr idisə, sonra yaranan fərdi kompüterlər kiçik və böyük bizneslərin hər yerdə istifadə etdiyi atributa çevrildi. Hesablama və alqoritmik – haqq-hesab məsələləri bu sahədə ənənəvi olaraq ikinci planda yerdə dururdu, ön planda isə məlumatların emalı və manipulyasiyası məsələləri dururdu.

Aydın oldu ki, prosedur proqramlaşdırmanın ənənəvi metodları nə proqramların artan çətinlikləri ilə, nə də onların etibarlılığının artmasının zəruriliyinin öhdəsindən gələ bilmir. 80-cı illərin ikinci yarısında bütün bu cür kompleks problemlərin öhdəsindən gələ bilən yeni proqramlaşdırma metodologiyasına kəskin ehtiyac yarandı. Bu metodolgiya **obyekt-yönümlü proqramlaşdırma** (**OYP**) oldu.

OYP-nin fundamental anlayışı class və obyekt anlayışıdır. Belə ki, class altında ümumi xassələ dəsti və eyni davranışa malik olan obyektlərin cəminin abstraksiyası anlaşılır. Bu halda hər bir obyektə müvafiq class nümunəsi kimi baxılır. Bütövlüklə eyni xüsusiyyətlərə və eyni davranışa malik olmayan obyektlər məzmuna əsasən görə bir class-a aid edilə bilməz. OYP-nin əsas prinsipləri vərəsəlik, inkap-

sulyasiya və polimorfizmdir.

Vərəsəlik – daha geniş kateqoriya haqqında olan bilikləri, daha dar kateqoriya üçün istifadə etməyə icazə verən prinsipdir.

İnkapsulyasiya – siniflərin daxili qurğularının ayrı-ayrı hissələrinin ona görə xarici sayılan obyekt və istifadəçilərdən gizlədilməsidir. İnkapsulyasiya öz kökünü modulun bir neçə proqramlaşdırma dilində iki hissə və ya seksiyaya: interfeys və realizasiya - bölünməsindən götürür. İnterfeys digər obyektlərlə qarşılıqlı əlaqədə olmaq üçün bütün informasiyanı özündə saxlayır. Realizasiya digər obyektlərdən obyektlərlə qarşılıqlı əlaqə prosesinə aidiyyatı olmayan detalları gizlədir və ya maskalayır.

Polimorfizm – bəzi obyektlərin vəziyyətdən asılı olaraq müxtəlif xarici formalar almaq qabiliyyətidir. OYP-yə tətbiq zamanı polimorfizm, eyniadlı metodlarla tədbiq edilən əməllər bu və ya digər meotodun hansı class-a aid olmasından asılı olaraq fərqlənməsi deməkdir. Obyekt-yönümlü dillərin polimorfizmi funksiyaların yenidən yüklənməsi ilə bağlı olsa da, lakin, onunla eyni deyil. Yaddan çıxarmaq lazım deyil ki, metodların adları onların təsvir olunduğu class-ların adı ilə sıx bağlıdır. Bu proqramın işi zamanı müəyyən dərəcədə etibarlılıq təmin edir, belə ki, ona xas olmayan məsələlərin həllində metodların təsadüfən istifadəsini istisna edir.

Obyekt yönümlü proqramlaşdırma dillərinin yaranması class və obyektlərin sintaksis mərhələdə realizasiya konsepsiyasının zəruriliyi ilə bağlı idi. Məşhur **C** və **Pascal**

dillərinə class-ların daxil edilməsi, **C++** və **Object Pascal** dillərinin yaranmasına səbəb oldu.

OYP metodologiyasının geniş yayılması proqramların tərtibat prosesinə təsir etdi. Xüsusilə, proqramların prosedur yönümlü dekompozisiyası yerini o**byekt yönümlü dekompozisiyaya** verdi, burada proqramın ayrı-ayrı struktur vahidləri kimi prosedur və funksiyalar deyil, müvafiq xüsusiyyət və metodlu class və obyektlər oldu. Nəticədə, proqram, artıq qabaqcadan müəyyən olunmuş əməllərin kodlaşdırılma mərhələsinin davamı deyildi, o, **hadisəyə görə idarəedilən** oldu. Sonuncu hal müasir tətbiqlərin geniş dairəsinin tərtibatında hakim mövqe tutdu. Bu halda hər bir proqram qabaqcadan müəyyən olunmuş hadisələrin sonsuz gözləmə dövründən ibarətdir. Hadisələrin təşəbbüskarı kimi digər proqram və ya istifadəçilər ola bilər.

OYP metologiyasının inkşafında mühüm amil , proqram kodunun yazılma prosesi, proqramın strukturunun layihələndirmə prosesindən ayrıla bilməsi faktının dərk edilməsi oldu. Həqiqətən, class-ların, onların xassələrinin və metodlarının proqramlaşdırılmasından öncə, bu classların nə olduğunu müəyyən etmək lazımdır. Üstəlik, bu tip suallara cavab vermək lazımdır: qoyulmuş məsələnin həlli üçün nə qədər və hansı class-ları müəyyən etmək lazımdır, class-lara tələb olunan davranışı vermək üçün hansı xassə və metodlar vacibdir və həmçinin class-lar arası qarşılıqlı əlaqə qurmaq lazımdır.

Məsələlərin bu məcmusu tərtib olunan proqramın kod

yazılmasından daha çox gələcək proqramın tələblərinin ümumi analizi və həmçinin bu proqramın tətbiq olunduğu müyyən predmet sahəsinin analizi ilə bağlıdır. Bütün bu hallar xüsusi **obyekt yönümlü analiz və layiləndirmə metologiyasının (OYAP)** yaranmasına gətirib çıxartdı.

4.3. Obyekt-yönümlü analiz və layihələndirmə metologiyası

Artıq çoxdan başa düşülürdü ki, geniş miqyaslı layihələrin tərtibatı zamanı proqram yazılmasından öncə predmet sahəsinin analizi zəruridir.

Bu və ya digər məsələlərin həlli üçün zəruri olan, predmet sahəsinin başlanğıc və ya baza komponentlərinin seçilməsi ümumi halda mühüm problem yaradır. Bu problemin mürəkkəbliyi bu məqsəd üçün istifadə oluna bilən prosedur və qaydaların qeyri-formal xarakterində özünü göstərir. Üstəlik, bu cür iş predmet sahəsini yaxşı bilən mütəxəssis və ekspertlərlə birgə yerinə yetirilməlidir.

Predmet sahəsinin komponetnlərinin seçilməsi və ya identifikasiyası üçün bir neçə üsul və qaydalar təklif olunmuşdu. Bu proses – predmet sahəsinin **konseptualizasiyası** adını almışdır. Burada komponent adı altında funksionallığa malik, yəni qoyulmuş məsələlərin həlli ilə bağlı müəyyən əməlləri yerinə yetirə bilən bəzi abstrakt vahid anlaşılır.

OYAP metologiyasının yaranması bir tərəfdən, predmet sahəsinin müxtəlif tərtibat vasitələrinin, digər tərəfdən, bu metologiyaya malik olan müvafiq mütəxəssislərin olmasını tələb etdi.

Bu mərhələdə nisbətən yeni tip yaranır ki, o da a**nalitik** və ya **arxitektor** adını alır. Predmet sahəsi üzrə mütəxəssislərlə yanaşı analitik gələcək proqramın konseptual sxeminin qurulmasında iştirak edir ki, bu da daha sonra proqramçılar tərəfindən koda çevrilir. Burada ayrı-ayrı komponentlər bucür seçilir ki, növbəti tərtibatda onları class və obyekt formasında təsvir etmək rahat olsun.

Bu halda, predmet sahəsində konseptual sxem barədə informasiya verən dil özü də böyük əhəmiyyət kəsb edir. Mürəkkəb proqram tətbiqlərinin tərtibat prosesinin ayrı-ayrı mərhələlərə bölünməsi, proqramın həyat dövrü konsepsiyasının yaranmasına səbəb oldu. Proqramın həyat dövrü (HD) adı altında, ona qarşı tələblərin tərtibatından başlayaraq, onun istifadəsindən tam imtina edilməsinədək qarşılıqlı əlaqəli və zamanla ardıcıl olan mərhələlər başa düşülür. Qəbul olunmuş baxışlara görə proqramın HD aşağıdakı mərhələlərdən ibarətdir:

- Predmet sahəsinin analizi və proqrama olan tələblərin formalaşdırılması
- Proqramın strukturunun layihələndirilməsi
- Proqramın kodda realizasiyası
- Proqramın daxil olunması
- Proqramın müşaiyət olunması
- Proqramın istifadəsindən imtina

Predmet sahəsinin analizi tələblərinin formalaşdırılması mərhələsində funksiyaların təyin olunması həyata keçirilir ki, bu zaman tərtib olunan proqram və predmet sahəsinin konseptualizasiyası yerinə yetirməlidir. Bu mərhələnin nəticəsi olaraq tərkibində yerinə yetirilməli əsas komponent və funksiyaların təsviri olan bəzi konseptual sxemlər olmalıdır.

Proqramın strukturunun layihələndirmə mərhələsi gələcək proqramın detallı sxeminin tərtibatından ibarətdir və bu sxemdə class, onların xassə və metodları, həmçinin onlar arasında müxtəlif qarşılıqlı əlaqələr göstərilir. Bir qayda olaraq, bu mərhələdə həm analitiklər, arxitektorlar, həm də ayrıca peşəkar proqramçılar iştirak edə bilər. Bu mərhələnin nəticəsi kimi, üzərində proqramın fəaliyyət prosesində class və onlar arasında qarşılıqlı əlaqəsi göstərilən proqramın detallaşdırılmış sxemi olmalıdır. OYAP metodologiyasına əsasən, məhz bu sxem proqram kodunun yazılmasında ilkin informasiya olmalıdır.

Proqramlaşdırma mərhələsinin nəticəsi, tələb olunan funksioanllığa malik olan və konkret predmet sahəsində lazım olan məsələləri həll edə bilən proqram tətbiqidir.

Proqramın daxil edilməsi və müşayiət olunma mərhələləri proqramlaşdırma mühitinin sazlanma və konfiqurasiya edilmə zəruriyyəti ilə, həmçinin, onun istifadə prosesində yaranan səhvlərin aradan qaldırılması ilə bağlıdır. Hərdən isə, ayrıca mərhələ kimi proqramın test olunmasını da qeyd edirlər, burada ilkin məlumatların

bəzi məcmusunda və ya ekspluatasiyanın xüsusi rejimində proqramın işləmə qabiliyətinin yoxlanılması başa düşülür. Bu mərhələlərin nəticəsi kritik halların və ya bu tətbiqi istifadə edən şirkətə ziyan dəyməsini istisna edən proqram tətbiqinin etibarlılığının yüksəlməsidir.

OYAP metodologiyası proqram təminatının avtomatlaşdırılması konsepsiyası (**Computer Aided Software Engineering, CASE**) ilə sıx bağlıdır. İlk **CASE** - vasitələrinin meydana gəlməsi müəyyən ehtiyatla qarşılanmışdır.Vaxt ötdükcə onların imkanları barədə həm coşqun rəylər, həm də kritik qiymətləndirilmələr yarandı. Bu qədər ziddiyyətli fikirlərin yaranmasının bir neçə səbəbləri var idi. Onlardan birincisi bundan ibarətdir ki, ilk **CASE** – vasitələr bəzi məlumat bazalarının idarəetmə sistemlərinə sadə üstqurum idi. Hərçənd ki, məlumat bazalarının konseptual sxeminin tərtibat sxeminin vizualizasiyası böyük əhəmiyyət kəsb edir, o digər tipdə olan tətbiqlərin tərtibat problemini həll etmir.

İkinci səbəb daha mürəkkəb təbiətə malikdir, belə ki, bu və ya digər **CASE** -vasitədə realizasiya olmuş qrafik notasiya ilə bağlıdır. Əgər, proqramlaşdırma dillərinin ciddi sintaksisi var idisə, onda məlumat bazalarının konseptual sxeminə uyğun gələn vizual təsvirin təklif cəhdləri birmənalı qarşılanmırdı. Bu fonda proqramın HD-nin iki mərhələsinin məsələlərini həll etməyə yönəlmiş unifikasiyalaşdırılmış modelləşdirmə dilinin (**Unified Modeling Language, UML**) yaranması böyük optimistliklə qarşılanmışdı.

4.4. Sistem analizinin və sistem modelləşdirmənin metodologiyası.

Sistem analizi, elmi istiqamət kimi OYP və OYAP dan fərqli olaraq, daha uzun tarixə və öz tədqiqat predmetinə malikdir. Sistem analizinin mərkəzi anlayışı sistem anlayışıdır, bunun altında bəzi bütövlük əmələ gətirən sərbəst təbiətli obyektlərin, komponentlərin və elementlərin məcmusu anlaşılır. Sistem kimi bəzi məcmuların seçilməsinin həlledici şərti, onu təşkil edən elementlərdə olmayan yeni xassələrin onda yaranmasıdır. Sistemlərə kifayət qədər çox misallar gətirmək olar – fərdi kompüter, avtomobil, insan, biosfer, proqram və s.

İstənilən sistemin vacib xarakteristikaları onun strukturu və fəaliyyət prosesidir. Sistemin **strukturu** adı altında onun elementləri və komponentləri arasında qarşılıqlı əlaqəsinin zamanla davamlı olan məcmusu başa düşülür. Məhz bu struktur bütün elementləri bir araya gətirir və sistemin ayrı-ayrı komponentlərə ayrılmasının qarşısını alır. Sistem müxtəlif qarşılıqlı əlaqələri əks edə bilir, həmçinin bir sistem elementlərinin digərinin daxilinə yerləşdirilməsii də. Bu halda daha kiçik və ya daxilə yerləşdirilmiş sistemi a**ltsistem**, daha irisini isə **metasistem** adlandırırlar.

Sistemin fəaliyyət prosesi onun xassələrinin və zamanla onun davranışının dəyişilməsiylə sıx bağlıdır. Burada sistemin vacib xarakteristikası onun **vəziyyətidir**, bunun anlayış altında sistemin daha əhəmiyyətli xüsusiyyətləri-

ni hər an əks edən xassə və əlamətlərin məcmusu başa düşülür. Sistemin fəaliyyət prosesi sitemin zamanla davranışını əks etdirir və onun vəziyyətlərinin davamlı dəyişilməsi kimi təsvir oluna bilər.

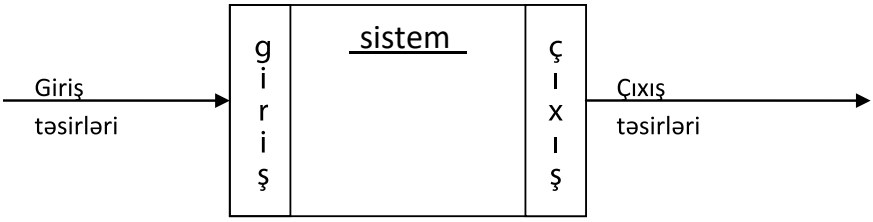
Əgər sistem özünün bir vəziyyətini digərinə dəyişirsə, onda belə demək olar ki, **sistem** bir vəziyyətdən digərinə keçir. Əlamətlərin və ya sistemin vəziyyətinin dəyişilməsi şəraitinin məcmusu **keçid** adlanır.

Sistem analizinin metodologiyası, predmet sahəsinin sistem - yönümlü dekompozisiyasının konseptual əsası kimi xidmət edir. Bu halda, konseptualizasiyanın ilkin komponentləri sistem və onlar arasındakı qarşılıqlı əlaqədir. Burada OYAP –ın class və obyekt anlayışlarından fərqli olaraq sistem anlayışı ümumidir. Sistem analizinin nəticəsi sitemin bəzi modelinin və ya predmet sahəsinin qurulmasıdır.

Modellərin qurulma zəruriyyəti, xassələr və sistem-orijinal davranışı barəsində informasiya almaq üçün istifadənin mümkünlüyüdür. Bu zaman sistem-orijinal barəsində informasiya almaq üçün modellərin qurulması və sonrakı istifadə prosesi **modelləşdirmə** adını almışdır.

Sistemin daha ümumi modeli "qara qutu" adlanan modeldir. Bu halda sistem, onun daxili qurğusunun analitikdən gizlin və naməlum olan düzbucaq kimi təsvir olunur. Lakin, sistem xarici mühitdən tamamilə izolyasiya edilməyib, belə ki, sonuncu, sistemə bəzi informasiya və material təsir göstərir. Bu cür təsirlər **giriş təsirləri** adını almışdır. Öz növbəsində sistem, mühitə və digər sistem-

lərə, giriş təsirləri adı almış informasiyalı və material təsir göstərir. Bu model qrafik olaraq bu cür təsvir oluna bilər:



"Qara qutu" ya bənzər bu modellərin dəyəri çox şərtidir. Ancaq, sistemin ümumi modeli, sistemin davranışını təsvir edən funksional xüsusiyyətləri barədə vacib informasiyaya malikdir. Həqiqətən, sistemin hansı əməllərə reaksiya verməsi və bu reaksiyanın onun əhatəsində olan obyekt və sistemlərdə özünü necə göstərməsinin ümumi informasiyasından əlavə başqa informasiya ala bilmirik.

Adekvat modellərin tərtibat və onların sonrakı konstruktiv tədbiq prosesi, tək sistem analizinin ümumi metologiya biliyinin olmasını tələb etmir, həm də modelləşdrimənin nəticələrinin fiksasiya edilməsi və sənədləşdirilməsi üçün müvafiq təsvir vasitələri və dillərinin olmasını da tələb edir. Aydındır ki, bu məqsəd üçün təbii dil tam uyğun gəlmir, belə ki, təbii dillər birmənalıdır və qeyri-müəyyəndir. Modellərin qurulması üçün kifayət qədər ciddi nəzəri metodlar tərtib olunmuşdu, bu metodlar modelləşdirmənin riyazi və məntiqi vasitələrinə əsasən qurulmuşdu və həmçinin həll edilən məsələlərin spesi-

fikasiyasını əks etdirən müxtəlif formal və qrafik notasiyalar təqdim olunmuşdur.

Təsvir etmək vacibdir ki, istənilən modelləşdirmə dilinin unifikasiyası sistem modelləşdirməsinin metodologiyası ilə sıx bağlıdır. Sistemin mürəkkəbliyinə və uyğun olaraq, onun modelinin mürəkkəbliyinə müxtəlif nəzər nöqtələrindən baxıla bilər. İlk öncə, sistemin strukturunun mürəkkəbliyini qeyd etmək olar ki, sistem elemntlərinin sayı və bu elementlər arasında olan qarşılıqlı əlaqələrin müxtəlif tipləri ilə xarakterizə edilir. Mürəkkəbliyin ikinci aspekti, sistemin fəaliyyət prosesinin mürəkkəbliyidir. Bu, sistemin həm gözlənilməz davranış xarakteri ilə , həm də giriş təsirlərinin çıxış təsirlərinə keçid qaydalarının formal təsvirinin qeyri-mümkünlüyü ilə bağlı ola bilər.

5. Class-lar və obyektlər

“Obyekt, verilmiş predmet sahəsində vacib funksional təyinatı olan xüsusi tanınma predmetini, blok və ya varlığı (real və ya abstrakt) özündə təsvir edir ” (Smitt, Tokey).

Müəyyənliyə görə nəzərdə olan tədbiqi problemin kontekstində məna daşıyan anlayışı, abstraksiyanı və ya dəqiq cızılmış sərhədləri olan istənilən şeyi obyekt adlandıracayıq. Obyektin daxil edilməsi iki məqsədi güdür:

- Tədbiqi məsələ (problem) anlayışı;
- əsasın kompüterdə realizasiyası üçün daxil edilməsi.

Obyektin vəziyyəti, davranışı və individuallığı vardır. Obyektlər zamanda mövcuddurlar, dəyişirlər, daxili vəziyyətlərə malikdirlər, keçicidirlər, yarana, məhv ola və parçalana bilirlər.

Oxşar obyektlərin strukturu və davranışı onlar üçün ümumi class-ı müəyyən edir.

Obyektin vəziyyəti verilmiş obyektin sadalanmış mümkün (adətən statistik) xassələri və hər bir xassənin cari qiymətləri (adətən dinamik) ilə xarakyerizə edilir.

Davranış obyekt üzərində aparılan əməllərin ardıcıllığı ilə müəyyən edilir. Davranış, obyektin necə təsir etdiyini və digər obyektlərin ona təsirini bu obyektlərin vəziyyətinin dəyişməsi və ismarıcların göndərilməsi nöqteyi nəzərindən xarakterizə edir.

Obyektlərin class-lara müəyyən edilməsi, məsələyə abstraksiyanın yeridilməsinə və onun daha ümumi quruluşda baxılmasına imkan verir. Class onun bütün

obyektlərinə aid olan ada malikdir. Bundan başqa, class-da obyektlər üçün müəyyən edilmiş atributların adı daxil edilir. Bu mənada class-ın təsviri struktur tipinin təsvirinə (yazılmasına) analojidir, burada hər bir obyekt struktur nümunəsində (müvafiq tipin dəyişkən və konstantı (sabiti) olan mənaya malikdir.

Atribut – onun class-dakı obyekti xarakterizə edən qiymətdir. Atributlara misallar.

Atributlar arasında daimi atributlar (konstantlar) və dəyişkən atributlar seçilir. Daimi atributlar obyekti onun class-ında xarakterizə edir. (məsələn: hesab nömrəsi, kateqoriya, insan adı və s.) Dəyişkən atributların cari qiymətləri obyektin vəziyyətini xarakterizə edir. (məsələn: hesab balansı, insanın yaşı və s.) Bu obyektlərin dəyərlərini dəyişərək, biz obyektin vəzyiyətini də dəyişmiş oluruq.

Obyekt üzərində əməliyyatlar

Əməliyyat –verilmiş class obyektlərinə qarşı tədbiq edilə bilən funksiyadır (və ya şəklini dəyişmə).

Verilmiş class-ın bütün obyektləri hər əməliyyatın eyni nümunəsini istifadə edir (yəni, hansısa class-ın obyektlərinin sayının artırılması, yüklənmiş proqram kodunun sayının artırılmasına gətirmir). Əməliyyatın çağırıldığı obyekt, ona onun aşkar olmayan arqumenti (this parametri) kimi ötürülür.

Eyni əməliyyat, ümumilikdə desək, ayrı-ayrı class-lara tədbiq oluna bilər: bu cür əməliyyat polimorf adlanır, belə ki, o ayrı-ayrı classlar üçün ayrı-ayrı formalara malik ola bilər. Məsələn: vektor və kompleks rəqəm class-nın

obyektləri üçün + əməliyyatını müəyyən etmək olar. Bu əməliyyat polimorf olacaq, belə ki, vektorların və kompleks rəqəmlərinin toplanması, ümumilikdə desək, ayrı-ayrı əməliyyatlardır.

Hər əməliyyata hər hansı bir metod müvafiqdir – verilmiş class obyektləri üçün əməliyyatın realizasiyası metodu. Beləliklə, əməliyyat – metodun spesifikasiyasıdır, əməliyyatın realizasiya metodu.

Əməliyyat (və onu realizasiya edən metodlar) öz siqnaturası ilə müəyyən edilir, hansı ki, əməliyyatın adından başqa, onun bütün arqumentlərinin tiplərini (class-larını) və nəticənin (qayıdan dəyər) tipini (class-nı) özündə birləşdirir. Əməliyyatı realizasiya edən bütün metodlar, onlar tərəfindən realizasiya edilən əməliyyatda olan siqnaturaya malik olmalıdırlar.

Sistemin modelləşdirilməsi zamanı yan təsirə (bu effektlər obyektin atributlarının dəyərlərinin dəyişilməsində, yəni onun vəziyyətinin dəyişilməsi zamanı özünü biruzə verir) malik olan əməliyyatları və obyektin vəziyyətini dəyişmədən tələb olunan dəyəri verən əməliyyatları ayırd etmək yaxşı olardı. Bu sonuncu əməliyyatlar sorğu adlanır.

Obyektin bəzi atributlarının qiymətləri yalnız bu obyektin əmələliyyatları üçün mümkün ola bilər.

Arqumentsiz sorğular (əməliyyatın tədbiq edildiyi aşkar olmayan arqument - obyekt istisna olmaqla) törəmə atributlar kimi baxıla bilər.

Daha çox yayılmış əməliyyat növləri:

1. Modifikator – obyektin vəziyyətini yazı və keçidlə dəyişir.

2. Selektor – obyekti dəyişmədən onun vəziyətinin müəyyənləşdirilməsi üçün keçid verir. (oxuma əməliyyatı)
3. İterator – obyektin hissələrinə keçidin müəyyən ardıcıllıqla təşkili.
4. Konstruktor – obyektin yaradılması və inisializasiyası.
5. Destruktor – obyektin məhvi və ya yaddaşda tutduğu yerin azad edilməsi.

Obyektlər arasında münasibət

Obyektlər arasında məlumatlara uyğun bağlılıq qurmaq olar. Bu bağlılıqlar göstərilən obyektlərin class-ları arasında olan əlaqə və münasibətləri ifadə edir. İki istənilən obyektin münasibətləri onların bir-biri haqqında yerinə yetirilə bilən əməliyyatlar və gözlənilən davranış barədə bilgilərinə əsaslanır.

Obyektlər arasındakı münasibətlərin növləri:

1. Əlaqələr
2. Aqreqasiya

Əlaqələrə olan münasibətdə obyektlər müxtəlif rolları icra edə bilər:

- Aktyor və ya aktiv obyekt (**actor**) – obyekt təsir göstərə bilər, lakin, heç vaxt təsirə məruz qalmır (aktiv obyekt).
- Server və ya icraçı (**server**) – digər obyektlər tərəfindən idarəyə məruz qalır, lakin, heç vaxt aktiv deyil.
- Agent və ya vasitəçi (**agent, broker**) – həm aktyorun, həm də serverin rolunu icra edir, bir qayda olaraq, aktiv obyektin maraqlarında yaradılır.
- Aqreqasiya tipli əlaqə hissə - bütövlük tipli münasibətdir (**A part-of**). Bütövdən (aqreqatdan) başla-

yaraq onun hissələrinə (atributlarına) gəlməyə imkan yaradır.

Mənaca, aqreqasiya bir obyektin digərinə (obyektin tərkib və ya strukturu) fiziki olaraq daxil olması deməkdir, lakin, başqa bir semantika da ola bilər, məsələn bir obyektin digərinə mənsub olması.

CLASS

Class – ümumi struktur və davranışlarla bağlı olan çox saylı obyektlərdir.

Bir qayda olaraq interfeys (hamıya görünən xarici görünüş və dəstəklənən metodların toplusu) və class realizasiyasını (başqalarından qorunan daxili qurğu) qeyd edirlər.

Class-ı təyin etmək üçün bu class-ın adını göstərmək lazımdır, sonra onun atribut və əməliyyatlarını (və ya metodlarını) sadalamaq lazımdır. Hər bir obyektlə, məlumatların strukturu bağlıdır, bu obyektin sahələri onun atributları və bu obyektin əməliyyatlarını həyata keçirən funksiyaların (kod fraqmentlərinin) göstəriciləridir (qeyd edək ki, funksiyaların göstəriciləri, kodun optimallaşdırılması nəticəsində bu funksiyalara müraciətlərlə əvəz olunur). Beləliklə, obyekt, məlumatların bir strukturudur, obyektin class-ına müvafiq olan bir növdür.

Hərdən altclass-da onun superclass-larında müəyyən olunan əməliyyatı yenidən təyin etmək vacibdir. Bunun üçün, varəsəlik nəticəsində superclass-dan alına bilən əməliyyat, altclass-da müəyyən olunur. Onun bu təkrar təyini onun superclass-da təyin olunmasının "qabağını kəsir", buna görə class-da varis əməliyyat yox, yenidən təyin

olunma əməliyyatı tətbiq olunur. Yadda saxlamaq lazımdır ki, hər bir əməliyyat öz siqnaturasıyla müəyyən olunur, beləliklə, əməliyyatın təkrar təyini siqnaturası super-class-da olan əməliyyatın siqnaturasıyla uyğun gəlməlidir, hansı ki, verilmiş əməliyyatla təkrar təyin edilir.

Təkrar təyin aşağıdakı məqsədlərdən birini güdə bilər:

- Genişləndirmə: yeni əməliyyat atributlarının təsirini nəzərə alaraq varis əməliyyatı genişləndirə bilər.
- Məhdudiyyət: yeni əməliyyat, altclass-ın obyektlərinin spesifikasını istifadə edərək, varis əməliyyatın fəaliyyətinin bir hissəsinin yerinə yetirilməsi ilə məhdudlaşır.
- Optimallaşdırma: altclass obyektlərinin spesifikasının istifadəsi müvafiq metodu sadələşdirməyə və sürətləndirməyə imkan verir.
- Rahatlıq.

Proqramlaşdırma dilində realizasiya edilən class arasında münasibət:

- Assosasiya – class-lar arasında məna və semantik əlaqə. Biri-birinə, biri–çoxlarına, çoxları-çoxlarına münasibətləri.

- Ümumiləşdirmə və varislik obyektlərin müxtəlif classları arasındakı analogiyaları aşkarlamağa imkan verir, obyektlərin çoxmərhələli klassifikasiyasını müəyyən edir. Belə, qrafik sistemlərdə müxtəlif həndəsi fiqurların təsvirini müəyyən edən class-lar mövcud ola bilər, məsələn: nöqtə, xəttlər (splaynlarla müəyyən olunan düz xəttlər, dairələrin və əyri xəttlərin qövsləri), çoxbucaqlar, dairələr və s.

Realizasiya üsulu – deleqirləşdirmə, obyektlərə prototiplər kimi (nümunələr) baxılma, bu prototiplər yeni class-ların yaradılması tələbətanı istisna edərək, öz davranışlarını digər obyektlərə deleqirləşdirir. Ümumilik və assosiativlik münasibətini realizasiya edir. Abstrakt class-lar (obyekt realizasiyası olmayan). Ən ümumi class – baza class-ı.

- Aqreqasiya – kompozisiyalı obyekt və class-lara, bu obyektlərin komponentlərini təsvir edən bağlılıq ("bütöv" – "hissə" münasibəti).
- Müxtəliflik və ümumilik münasibəti (klassifikasiya)
- Polimorfizm (bir neçə növə ayırd edilmə)
- İstifadə - Bir class-ın realizasiyasında digər sinif istifadə oluna bilər. Məsələn: Xətt – nöqtə.
- Doldurulma. Yığma class. Həmcins class-lar – bir class-dan olan obyektlərdən ibarətdir. Həmcins olmayan class-lar – müxtəlif class-lardan ibarət. Digər class-ları toplusundan ibarət class, ekzemplyar. Parametrləşdirilmiş class-lar.
- Metaclass-lar (class-ların class-ı) class-ı obyekt kimi izah edir.

Class və obyekt arasında münasibətlər.

Klassifikasiya – biliklərin qaydaya salınma vasitəsi.

Klassifikasiyanın metodları.

Kateqoriyalar üzrə klassik bölüşdürülmə.Bu xüsusiyyətə malik obyektlər bir sinfə aid edilir.

Konseptual birləşdirmə.İlk öncə konseptual təsvir formalaşır, sonra isə kateqoriyalar üzrə bölüşdürülmə. (gizli əlamətlər).

Prototip nəzəriyyəsi. Class-ın prototipi ilə oxşarlıq.

6. Diaqramlara ekskurs

Struktur sistem analizi altında sistemin tədqiqat metodu başa düşülür. Bu onun davranışının ayrı-ayrı aspektlərinin və fəaliyyətinin sonrakı detalizasiyanın daha ümumi təsviri ilə başlayır. Burada sistemin ümumi modeli bəzi ierarxik strukturda qurulur, hansı ki, hər mərhələdə məhdud sayda komponentləri olan abstraksiyanın müxtəlif mərhələlərini əks edir. Struktur sistem analizinin əsas prinsiplərindən biri, abstraksiyanın hər bir mərhələsində sistemin yalnız daha önəmli komponentlərinin və elementlərinin seçilməsidir.

Proqram mühəndisliyi çərçivəsində diaqram adı almış 3 qrafik notasiyaları nəzərdən keçirilir:"varlıq - əlaqə" diaqramları (**Entity-Relationship Diagrams, ERD**), Funksional modelləşdirmə diaqramları (**Structured Analysis and Design Techique, SADT**), Məlumat axını diaqramları (**Data Flow Diagrams, DFD**).

6.1 "Varlıq - əlaqə" diaqramları

"varlıq - əlaqə" diaqramları proqram sistemiylə tərtib olunmuş məlumat modellərinin qrafik təsviri üçündür və məlumatların və onlar arasındakı münasibətin təyin olunması üçün standart nişanlanmalar təklif edir. Bu diaqram növünün köməyilə tərtib olunan sistem üçün vacib olan məlumat modellərinin konseptual komponentlərini və onlar arasındakı əlaqənin məcmusunu təsvir etmək olar.

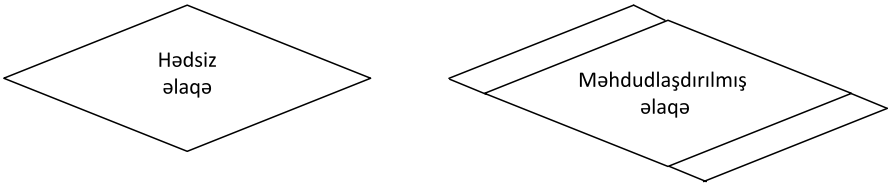
Bu notasiyanın əsas anlayışları varlıq və əlaqə anlayışıdır.

Burada (**entity**) varlığı anlayışı altında hər biri eyni xassə və xarakteristikalara malik olan bir çox real və abstrakt obyektlər başa düşülür. Bu halda hər nəzərdən keçirilən obyekt yalnız bir varlığın **nümunəsi** ola bilər, onun unikal adı və identifikatoru olmalıdır, həmçinin verilmiş varlığın digər nümunələrndən fərqlənməlidir.

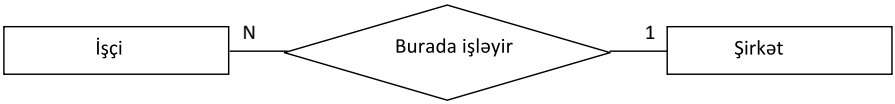
Varlığa misal bank, bank müştərisi, müştərinin hesabı, hava limanı, sərnişin, reys, kompüter, terminal, avtomobil, sürücü və s. ola bilər. Hər bir varlıq detalizasiyanın müxtəlif dərəcəsi və abstraksiyanın müxtəlif mərhələlərində nəzərdən keçirilə bilər, bu da konkret qoyulmuş məsələylə təyin olunur. Varlığın qrafik təsviri üçün xüsusi nişanlamalardan istifadə olunur:

Əlaqə (relationship) ayrı-ayrı varlıqlar arasında davranış və bəzi assosiasiya kimi müəyyən olunur. Buna misal olan əlaqələr "ata-oğul" tipli qohum əlaqələr və ya "rəis – onun tabeliliyində olan" tipli istehsalat münasibətləri ola bilər. Digər əlaqə növü bu cür münasibətlərlə verilir: "öz məxsusluğunda olmaq" və ya "xassəyə malik olmaq".

Əlaqələrin müxtəlif növləri növün özünəməxsus adı ilə romb şəklində təsvir olunur.



Məlumatların qrafik modeli elə qurulur ki, ayrı-ayrı varlıqlar arasında olan əlaqə müvafiq münasibətin tək semantik xarakterini yox, həm əlaqələrin məcburiliyinin əlavə aspektləri, həm də, bu əlaqələrdə iştirak edən varlıqların nümunələrinin qısalığını əks etdirsin. İki varlıqla təsvir edilən vəziyyəti sadə nümunə kimi nəzərdən keçirək: "İşçi" və "Şirkət". Burada təbii ki, əlaqə kimi işçinin şirkətə olan mənsubiyyətini istifadə etmək lazımdır. Əgər bu barədə fikirləri nəzər alsaq ki, şirkətdə bir neçə işçilər işləyir və bu işçilər digər şirkətlərin işçisi ola bilməz, onda bu məlumatın qrafik təsviri aşağıdakı "varlıq-əlaqə" diaqramında olacaq:



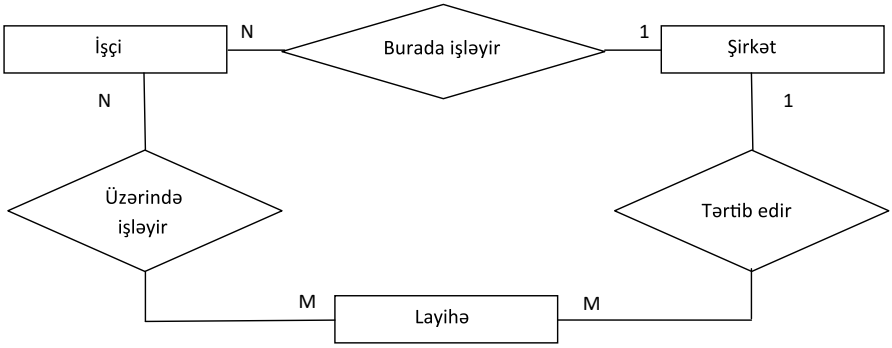
Bu şəkildə əlaqənin yanındakı "**N**" hərfi bu faktı bildirir

ki, şirkətdə bir işçidən artıq işçi işləyə bilər, belə ki, burda N-in mənası əvvəlcədən fiksə edilmir. Digər tərəfdəki "**1**" rəqəmi isə bildirir ki, işçi yalnız konkret bir şirkətdə işləyə bilər, yəni, şirkətdə eyni vaxtda həm o həm də bu şirkətdə işləyən işçilərin işləməsinə icazə verilmir.

Digər bu varlıqların nəzərdən keçirilməsi zamanı bir qədər başqa vəziyyət yaranır "işçi" və "layihə" və əlaqələr "layihə üzərindəki işdə iştirak edir".

Bir işçi bir neçə layihə tərtibatında iştirak edə bilər, bir layihənin tərtibatında isə bir neçə işçi işləyə bilər.Yəni, bu əlaqə çoxmənalı ola bilər. Bu fakt bilərəkdən müvafiq varlıqların yanında "N" və "M" hərfləriylə işarələnib, burada konkret hərflərin seçilməsi prinsipial deyil.

Üzərində şirkətin proqramlarının tərtibatında iştirak edən işçilər barədə məlumat verilən nəzərdən keçirilmiş iki diaqram bir-birinə birləşdirilə bilər. Burada bu şirkətin layihələrini xarakterizə edən əlavə əlaqə də daxil edilə bilər.



"Varlıq-əlaqə" diaqramlarının məhdudluğu, konseptual modelin modelləşdirilən proqram sisteminin daha detallı təsvirinə konkretləşdirilməsində özünü göstərir, hansı ki, statistik əlaqələrdən savayı, onun ayrı-ayrı komponentlərinin davranış və fəaliiyəti barədə informasiyanı özündə təşkil etməlidir. Bu məqsədlər üçün məlumat axını diaqramları adını alan digər diaqram növü istifadə edilir.

6.2. Funksional modelləşdrimə diaqramları

Funksional modelləşdirmə diaqramlarının tərtibatının başlanğıcı 1960-cı illərin ortalarına aiddir, o zaman Duqlas T.Ross **SADT** (**Structured Analysis & Design Technique**) adı almış modelləşdirmə texnikasını təklif etmişdir. ABŞ-ın hərbi-hava qüvvələri **SADT** metodikasını özünün kompüter və sənaye texnologiyalarının inteqrasiyası proqramının (**Integrated Computer Aided Manufacturing, ICAM**) hissəsi kimi istifadə etmişdirlər və onu

IDEF0 (**Icam DEFinition**) adlandırmışdılar. **ICAM** proqramının məqsədi kompüter texnologiyalarının, yeni silah vasitələrinin tərtibatı və hərbi əməliyyatların aparılmasında effektliyinin yüksəldilməsi idi. Bu tədqiqatların nəticələrindən biri bu nəticə idi ki, mürəkkəb sistemlərin fəaliyyət prosesinin modelləşdirilməsi və sənədləşdirilməsi üçün təsviri dillər effektli deyil. Təbii dillərdə bu cür təsvirlər modelləşdirmə məsələlərinin həllində qiymətlərinin üstünlük təşkil etdiyi üstünlük təşkil etdiyi ziddiyyətsizlik və tamlığın tələb olunan səviyyəsini təmin etmir.

ICAM proqramı çərçivəsində aşağıdakı adları alan qrafik modelləşdirmə dilləri tərtib olunmuşdur:

- IDEF0 notasiyası – istehsalat proseslərinin sənədləşdirilməsi və sistemin hər tərtibat mərhələsində resusların istifadəsi barədə məlumatların əks olunması üçün.
- IDEF1 notasiyası – sistemin istehsalat ətrafı barədə məlumatın sənədləşdirilməsi üçün.
- IDEF3 notasiyası – xüsusi olaraq biznes proseslərinin modelləşdirilməsi üçün.

IDEF2 notasiyası heç vaxt tam realizasiya edilməmişdir. **IDEF1** notasiyası 1985-ci ildə genişləndirilmiş və ona **IDEFIX** adı verilmişdir. **IDEF-SADT** metodologiyası dövlət və kommersiya qurumlarında tətbiq olunmağa başladı, belə ki, o zamankı dövrdə geniş sistem siniflərinin modeləşdirilməsinə təqdim olunan tələbləri tam qane edirdi.

1990 – cı ilin əvvəlində ABŞ –ın Dövlət Standartlaşdır-

ma və Texnologiya İnstitutu ilə (**National Institutes for Standards and Technology, NIST**) əməkdaşlıq da məhz **IDEF** üçün yaradılan istifadəçi qrupu (**IDEF User Group**) **IDEF0** və **IDEFIX** üçün standartın yaradılması üçün səylər etmişdir. Bu səylər uğurla başa çatmışdır və 1993-cü ildə ABŞ hökuməti tərəfindən iki, **IDEF0** və **IDEFIX** texnologiyaları üçün **FIPS** kimi məşhur olan standartın qəbul olunması ilə bitmişdir. Sonradan gələn illər ərzində bu standart aktiv olaraq inkşaf edirdi və bəzi **CASE**-vasitələrinin realizasiyasında əsas rol oynadı.

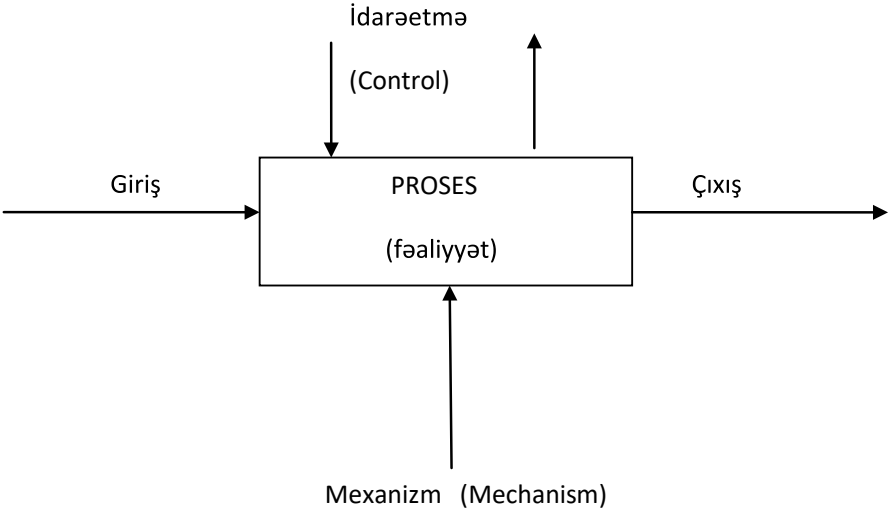
IDEF-SADT metodologiyası hər hansı bir predmet sahəsinin funksional modelinin qurulmasına xidmət edən metod, qayda və prosedurların məcmusundan ibarətdir. **SADT** funksional model sistemin və onun ayrıca altsistemlərinin fəaliyyət prosesinin strukturunu əks edir, yəni onun yerinə yetirdiyi tədbirlər və bu tədbirlər arasındakı əlaqəni. Bu məqsəd üçün ayrıca modellər hazırlanır, hansılar ki, əyani formada müəyyən tədbirlərin ardıcıllığını təsvir etməyə imkan verir. **IDEF0** prosesinin istənilən modelinin ilkin tikinti blokları, fəaliyyət (activity) və oxlardır (arrows).

IDEF-SADT metodologiyasının, funksional modelləşdirmənin diaqramlarının qurulmasında istifadə olunan əsas anlayışlarına bir nəzər salaq. Fəaliyyət fiksə edilmiş məqsədi olan və bəzi son nəticəyə gətirən fəaliyyət və ya fəaliyyət toplusunu özündə təmsil edir. Hərdən fəaliyyəti – proses də adlandırırlar. IDEF0 modelləri, sistemin müxtəlif növ fəaliyyətlərini, onların təsvirini və qarşılıqlı

fəaliyyətini izləyirlər. Fəaliyyət diaqramlarında fəaliyyətlər və ya proses blok adlanan düzbucaqlı kimi təsvir olunur. Ox bəzi daşıyıcı və ya təsirin nişanlanması üçün xidmət edir, bunlar bir fəaliyyətdən digərinə məlumatların daşınmasını təmin edir. Oxlar həmçinin fəaliyyətin məhz nə istehsal etdiyini və hansı resurları sərf etdiyini təsvir etmək üçün lazımdır. Bu **ICOM** adlanan oxların rolları **IDEF0** müvafiq oxlarının adlarındakı ilk hərflərin qısaldılmasıdır. Burada oxların dörd növü ayırd edilir:

- **I (Input)** – giriş, yəni prosesə daxil olan və proses tərəfindən sərf edilən hər şey.
- **C (Control)** – idarəetmə və prosesin əməliyyatlarının yerinə yetirilməsinə məhdudiyyətlər.
- **O (Output)** – prosesin çıxış yolu və ya nəticəsi.
- **M (Mechanism)** – prosesin yerinə yetirilməsi üçün istifadə olunan mexanizm.

IDEF0 metodologiyası hər **ICOM** növünün oxlarının diaqramlarda necə təsvir olunmasını birmənalı müəyyən edir. (**Input**) giriş oxu iş sahəsinin sol çərçivəsindən çıxır və prosesin düzbucağına soldan daxil olur. (**Control**) – idarəetmə oxu yuxarıdan çıxır və daxil olur. (**Output**) – prosesin sağ tərəfində çıxır və çərçivənin sağ tərəfinə daxil olur. (**Mechanism**) – Mexanizm proses düzbucağına altdan daxil olur. Beləliklə, IDEF0 diaqramlarında prosesin baza təsviri aşağıdakı formaya malikdir:

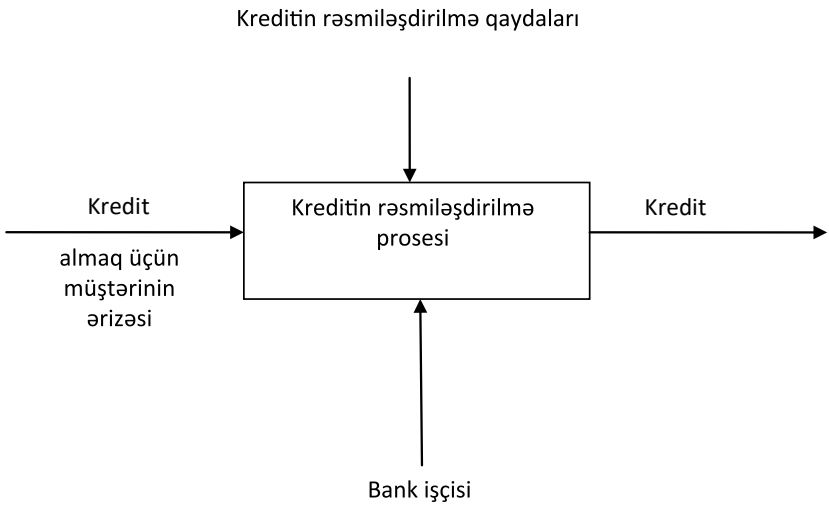


Diaqram qurulmasının texnikası **IDEF-SADT** metodologiyasının əsas xüsusiyyətini özündə təsvir edir. Oxun blokla birləşdiyi yer interfeysin növünü təyin edir. Burada modelləşdirilən sistemin bütün funksiyaları və diaqramlardakı interfeyslər **ICOM** –un müvafiq proses blokları və oxları şəklində təsvir olunur. Solda təsvir olunan informasiya düzəlişə məruz qaldığı vaxtda, idarəedici informasiya bloka yuxarıdan daxil olur. Prosesin nəticələri prosesin çıxışı kimi təsvir olunur və blokun sağ tərəfində göstərilir. Bu əməliyyatı həyata keçirən mexanizm kimi həm insan, həm avtomatlaşdırılmış sistem çıxış edə bilər. Müvafiq mexanizm bloka aşağıdan daxil olan ox kimi göstərilir.

IDEF-SADT metodologiyasının vacib xüsusi-

yyətlərindən biri ayrıca diaqram tərtibatı zamanı sistemin daha detallı təsvirlərinin tədricən daxil edilməsidir. **IDEF-SADT** modelinin qurulması bütün sistemin, sistemdən xaric obyektlərlə qarşılıqlı fəaliyyətin əsas növlərini təsvir etmək üçün xidmət edən , **ICOM** –un proses və oxlarının bir blokundan ibarət, sadə diaqram şəklində təsvir olunmasından başlayır. Belə ki, ilkin proses bütün sistemi tam bir vəhdət şəklində təsvir edir, bu təsvir daha ümumidir və gələcək dekompozisiyaya məruz qalır.

IDEF-SADT metodoligiyasının əsas ideyalarının illustrasiyası üçün növbəti sadə misala nəzər yetirək. Proses kimi bankda kreditin rəsmiləşdirilməsi fəaliyyətini götürək. Bu prosesin girişi kimi müştərinin kredit alması üçün ərizə təqdim etməsidir, çıxış kimi isə müvafiq nəticə, yəni bilavasitə kredit. Belə ki, müvafiq maliyyə vasitələrinin kredit götürmək üçün reqlament etdiyi şərtlər, idarəedici faktor kimi kredit rəsmiləşdirilməsinin qaydalarıdır. Bu prosesin mexanizmi kredit rəsmiləşdirmə əməliyyatlarına səlahiyyəti olan bank işçisidir.



Son nəticə olaraq **IDEF-SADT** modeli mürəkkəb sistemin ilkin təsvirini ayrı tərkib hissələrinə bölən, ierarxik bağlılığı olan müşayiətli sənədləşdirməli diaqramların seriyasını özündə təsvir edir. Hər əsas prosesin detalları digər diaqramlarda daha detallı proses kimi təsvir olunur. Bu halda aşağı səviyyənin hər diaqramı, daha ümumi diaqramın bəzi prosesinin dekompozisiyasıdır. Buna görə dekompozisiyanın hər addımında daha ümumi diaqram bir sıra daha detallı diaqrama konkretləşir.

Hal-hazırda **IDEF-SADT** struktur sistem analizinin diaqramları bir sıra çoxsaylı təşkilatlar tərəfindən, müəssisədə mövcud olan biznes-proseslərinin qurulması və detallı analizi üçün, həmçinin yeni biznes-proseslərinin tərtibatı üçün istifadə olunmağa davam edir. Bu metodologiyanın əsas qüsuru mürəkkəb sistemlərin modelləri-

nin obyekt-yönümlü təsviri üçün lazım olan məlum vasitələrin olmamasıdır. Belə ki, bəzi analitiklər **IDEF-SADT** notasiyasının biliyi və tədbiq olunmasının vacibliyini qeyd edir, bu metologiyanın məhdudiyyətli imkanlarının, obyekt-yönümlü proqram kodunda müvafiq qrafik modellərinin realizasiyasında tədbiqi, onun köməyilə həll olan məsələlərin diapazonunu əhəmiyyətli dərəcədə daraldır.

6.3. Məlumat axınları diaqrmaları

Bu informasiya sistemlərinin qrafik modelləşdirmə metodologiyasının əsası, **DFD** məlumat axınlarının diaqramlarının qurulmasının xüsusi texnologiyasıdır.

DFD kontekstində sistem modeli bəzən informasiya modeli kimi təsvir olunur, bu modelin əsas komponentləri kimi bir altsistemdən digərinə informasiya daşıyan müxtəlif məlumat axınlarıdır. Hər altsistem daxil olan məlumat axınlarının müəyyən dəyişilməsini yerinə yetirir və digər altsistemlərinə informasiyanın dəyişilmə nəticələrini məlumat axını kimi ötürür.

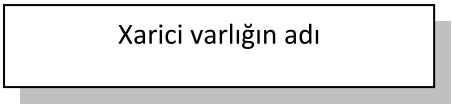
Məlumat axını diaqramlarının əsas komponentləri bunlardır:

- Xarici varlıqlar
- Proseslər
- Sistemlər/altsistemlər
- Məlumat yığıcıları və ya anbarlar
- Məlumat axınları

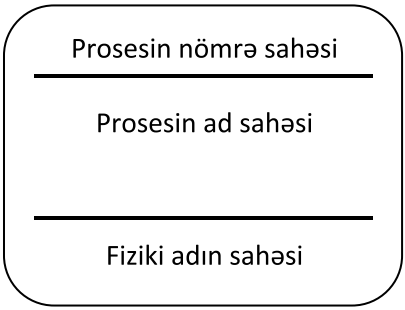
Xarici varlıq ya material obyekt, ya da, informasiya mənbəyi və ya qəbuledicisi kimi çıxış edən fiziki şəxs ola bilər.

Bəzi obyektin və ya sistemin xarici şəxs kimi müəyyən edilməsi sərt fiksə edilməyib. Belə ki, xarici varlıq nəzərdən keçirilən sistemin sərhədlərindən kənardadır, sonrakı analiz prosesi zmanı bəzi xarici varlıqlar sistemin modelinin diaqramının içinə keçirilə bilər. Digər tərəfdən ayrı proseslər diaqramdan kənara keçirilə və xarici varlıq kimi təqdim oluna bilər. Xarici varlıqlara misal kimi, təşkilatın müştəriləri, sifarişçilər, personal, təchizatçılar göstərilə bilər.

Xarici varlıq kölgəsi olan düzbucaq kimi işarələnir, içində onun adı göstərilir. Burada ismin adlıq halında olan adın yazılması məsləhət görülür. Hərdən daxili varlığı terminator adlandırırlar.



Proses təyin olunmuş müvafiq alqoritmlə və ya qaydayla giriş axınlarının çıxış axınlarına çevrilməsi əməliyyatlarının məcmusun özündə təmsil edir. Belə ki, fiziki olaraq proses mütəlif üsullarla həyata keçirilə bilər, daha çox prosesin proqamlı realizasiyası nəzərdə tutulur.Məlumat axınları diaqramındakı proses üç seksiyaya və ya sahəyə üfüqi xəttlərlə bölünmüş, küncləri yumrulaşdırılmış düzbucaq kimi təsvir olunur.



Prosesin nömrə sahəsi, sonuncunun identifikasiyasına xidmət edir. Ad kimi lazımi əlavələri olan felin qeyri-müəyyən formasından istifadə etmək məsləhət görülür. Aşağı sahədə prosesin fiziki realizasiya üsülunun olduğuna göstəriş var.

Sistemin informasiya modeli, üzərinə məlumatların müvafiq dəyişilmə proseslərinin altsistemi şəklində, ilkin modelin ardıcıl olaraq təsvir olunduğu bəzi ierarxik sxem, kontekst diaqramı şəklində qurulur. Burada DFD kontekst diaqramında olan altsistem və ya sistem, prosesdə olduğu kimi - küncləri yumrulanmış düzbucaq kimi təsvir olunur.

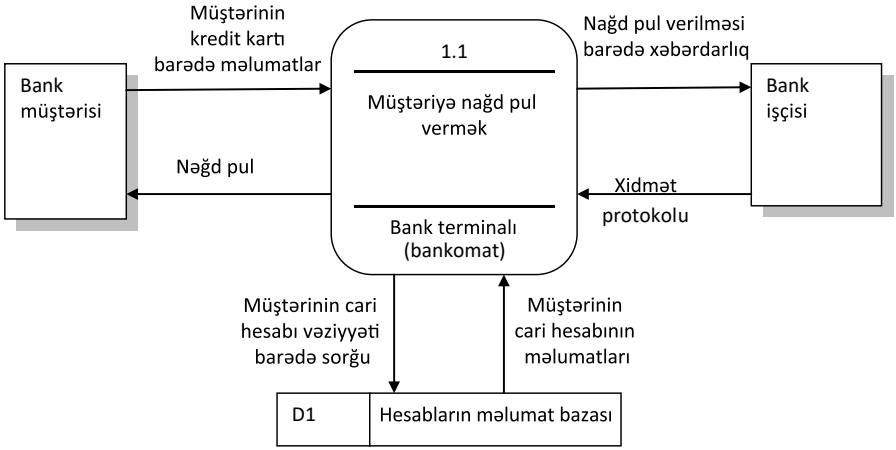
Məlumat yığıcıları və ya anbarlar proseslər arasında yerdəyişmə edən abstrakt qurğu və ya məlumat saxlama üsuludur. Ehtimal olunur ki, məlumatları istənilən vaxt yığıcıya yerləşdirmək və müəyyən vaxtdan sonra oradan çıxarmaq olar, belə ki, məlumatların yerləşdirmə və çıxarılmasının fiziki üsulu sərbəst ola bilər. Məlumat yığıcısı fiziki olaraq müxtəlif üsullarla realizasiya oluna bilər, lakin, çox vaxt onun realizasiyası maqnit daşıyıcılarla elektron şəkildə nəzərdə tutulur.Məlumat yığıcısı məlumat axınları diaqramında iki sahəsi olan diaqram kimi təsvir olunur. Birincisi "**D**" hərfi ilə başlayan yığıcının nömrə və ya identifikatorunu göstərmək üçün xidmət göstərir. İkinci sahə ad göstərməyə xidmət edir. Bu halda müvafiq informasiyanın saxlanma üsulunu xarakterizə edən, isim formasında adı göstərmək məsləhət görülür.

Nömrə sahəsi	Yığıcının ad sahəsi

Sonda, bəzi birləşmə vasitəsilə mənbədən qəbulediciyə ötürülən məlumat axını, informasiyanın keyfiyyətli xarakterini müəyyən edir. Real məlumat axını, şəbəkə vasitəsilə iki kompüter arasında və yaxud başqa üsulla məlumatların çıxarılması və tələb olunan formatda bərpa edilməsinə icazə verilərək ötürülə bilər. DFD diaqramında məlumat axını bir tərəfində ox şəkli olan xəttlə təsvir olunur, burada ox məlumat axınının istiqamətini göstərir. Hər bir

məlumat axını onun adını əks edən şəxsi ada malikdir.

Beləcə, DFD notasiyasında olan sistemin informasiya modeli məlumat axını diaqramı şəklində qurulur, hansı ki, müvafiq nişanlama sistemiylə qrafik təsvir olunur. Misal olaraq sadə bir modelə baxaq, burada bank müştərisinin kredit kartı ilə müəyyən məbləğdə nəğd pul götürməsini nəzərdə tutaq. Bu misalın xarici varlığı kimi bankın müştərisi, həm də müştərilərə göstərilən xidmətə nəzarət edən bank işçisi də ola bilər. Məlumat yiğıcısı kimi, bankın ayrıca müştərilərinin hesablarının vəziyyəti barədə məlumat bazası ola bilər. Ayrıca məlumat axınları banka xidmət göstərmək üçün lazım olan ötürülən informasiyanın xarakterini əks edir. Bu misala müvafiq gələn model aşağıdakı kimi təsvir oluna bilər:



Hal-hazırda məlumat axınlarının diaqramları bəzi CASE-vastilərindən məlumat düzəlişi siteminin informasiya modellərinin qurulması üçün istifadə olunur. Bu metodologiyanın əsas çatışmazlığı mürəkkəb sistemlərinin obyekt-yönümlü təsviri, həmçinin məlumat düzəlişlərinin mürəkkəb alqoritmlərinin təsviri üçün aşkar vasitələrin olmamasıdır. Belə ki, DFD diaqramlarında ayrıca proseslərin icrası üçün vaxtın xarakterisitikası və proseslər arasında məlumatın ötürülməsi göstərilmir, bu zaman məlumatların sinxron düzəlişinin sistem modelləri DFD notasiyasında adekvat təsvir oluna bilməz. Strukturlu sistem analizinin metodologiyasının bütün bu xüsusiyyətləri, onun geniş istifadəsinin imkanlarını məhdudlaşdırdı və unifikasiyalaşdırılmış modelləşdirmə dilinə müvafiq vasitələrin daxil edilməsində əsas kimi xidməti göstərdi.

7. UML dilinin inkişaf tarixi

Obyekt-yönümlü modelləşdirmənin ayrıca dilləri müxtəlif tədqiqatçılar və proqramçıların OYAP-a (obyekt-yünümlü analiz və layihələndirmə metodologiyası) öz yanaşmalarını təklif etdiyi, 1970-ci illərin ortası və 1980-cı illərin sonu olan dövrlərdə yaranmağa başladı. 1989-1994-cü illər arasındakı dövrdə daha məlum modelləşdirmə dillərinin sayı 10-dan, 50-dən çox saya artmağa başladı. Bir çox istifadəçilər OYAP dilinin seçimində ciddi çətinliklər çəkirdilər, belə ki, onlardan heç biri mürəkkəb sistem modellərinin qurulmasına tədbiq olunan tələblərə cavab verə bilmirdi. **IDEF0, IDEFIX** standartları kimi qəbul olunan metodikalar və qrafik notasiyalar, 90-cı illərin əvvəlində onlar arasındakı barışmaz rəqabət vəziyyətini dəyişə bilmədi.

1990-cı illərin ortasında metodlardan bəziləri əhəmiyyətli dərəcədə təkmilləşdirilmişdi və müxtəlif OYAP məsələlərindəki həllərdə müstəqil dəyər aldı. Bu dövrdə daha çox məşhur olanlar:

- Şərti Booch adı almış Qradi Buç (**Grady Booch**) metodu.
- Object Modeling Technique — OMT adı almış Ceyms Rumbax (**James Rumbaugh**) metodu.
- **Object-Oriented Software Engineering — OOSE** adı almış Ayvar Cekobson (**Ivar Jacobson**) metodu.

Bu metodlardan hər biri OYAP –ın ayrıca mərhələlərinin dəstəyinə istiqamətlənmişdi. Məsələn, **OOSE** metodu biznes-tətbiqlərinin tərtibat prosesininin analiz tələbləri mərhələsində vacib məna kəsb edən istifadə variantlarının təsvir vasitələrinə malik idi. **OMT** metodu informasiya sistemlərində məlumat düzəlişinin analiz prosesinə daha çox uyğun gəlirdi. Booch metodu daha çox müxtəlif proqram sistemlərinin layihələndirmə və tərtibat mərhələlərində tədbiq olunmağa başladı.

UML dilinin inkşaf tarixi öz başlanğıcını 1994-cü ilin oktyabrından götürür, Rational Software Corporation-dan olan Qradi Buç və Ceyms Rumbax Booch və OMT metodları üzrə unifikasiya işlərinə başladırlar. Belə ki, bu metodlar özlüyündə kifayət qədər populyar idi, bu metodlar üzrə müştərək tətqiqin məqsədi bütün obyekt-yönümlü metodların bütün üstünlüklərinin birləşdirilməsi idi. Burada Q. Buç və C. Rumbax səylərini öz işinin nəticələrinin tam unifikasiyasında cəmləşdirmişdilər. Bu cür adlanan unifikasiya edilmiş metodun (Unified Method) layihəsi 1995-ci ilin oktyabr ayında tədqiq edilmiş və dərc edilmişdi. Həmin ilin payızında onlara A.Cekobson, Objectory AB (İsveç) şirkətinin baş texnoloqu qoşulur, onun məqsədi öz OOSE metodunu öncəki iki metodla inteqrasiya etmək idi.

Əvvəlcə Booch, OMT və OOSE metodlarının müəllifləri ancaq bu metodikalar üçün modelləşdirmənin unifikisasiya edilmiş dilini tərtib etmək istəyirdilər. Bir tərəfdən bu metodlardan hər biri praktikada yoxlanmışdı

və ayrıca OYAP məsələlərinin həllində öz konstruktivliyini göstərmişdi. Bu da ayrıca anlayış və nişanlamaların mövcud olan uyğunsuzluqlarının aradan götürülməsi əsasında onların gələcək modifikasiyalarına əsas verirdi. Digər tərəfdən, semantika və notasiyanın unifikasiyası müvafiq proqram instrumentarilərinin uğurlu təşviqi üçün, obyekt-yönümlü CASE vasitələrinin bazarındakı lazım olan bəzi stabilliyi təmin etməli idi.

Sonda müştərək iş bu üç metodun əhəmiyyətli islahına ümid verirdi. Öz metodlarının unifikasiyasına başlayaraq, Q. Buç, C. Rumbax, A. Cekobson modelləşdirmə dilinə aşağıdakı tələbləri ifadə etmişdir. O gərək:

- Tək proqram təminatını yox, həm də daha geniş sistem class-larının və biznes-tətbiqlərini də obyekt-yönümlü anlayışlardan istifadə edərək modelləşdirməyə yol versin.
- Konseptual və fiziki mərhələlərin modellərinin baza anlayışları arasında qarşılıqlı əlaqəni aşkar təmin etsin.
- Modellərin miqyaslaşmasını təmin etsin, bu mürəkkəb çoxməqsədli sistemlərin vacib xüsusiyyətidir.
- Analitik və proqramçılara başa düşülən olmalı, müxtəlif kompüter platformalarında realizasiya edilən xüsusi instrumental vasitələrlə dəstəklənməlidir.

İşarələmə sisteminin və ya OYAP notasiyalarının tərtibatı yeni dilin tərtib olunmasına oxşar olmadı. Birincisi, iki problemi həll etmək vacib idi:

- Notasiya tələblərin spesifikasyasını özündə ifadə etməlidirmi?
- Bu notasiyanı vizual proqramlaşdırma dərəcəsinə qədər genişləndirmək lazımdırmı?

İkincisi, dilinin ifadəlilik və sadəliyi arasında uğurlu balansı tapmaq lazım idi. Bir tərəfdən, çox sadə notasiya müvafiq işarələmə sisteminin köməyilə həll oluna bilən potensial problemlərin əhatə dairəsini məhdudlaşdırır. Digər tərəfdən, çox mürəkkəb notasiya onun proqramçı və analitiklər tərəfindən öyrənilməsi və tədbiq edilməsinə əlavə çətinliklər yaradır. Mövcud olan metodların unifikasiya halında, onunla iş təcrübəsi olan mütəxəssislərin maraqlarını da nəzər almaq lazımdır, belə ki, yeni notasiyaya ciddi dəyişikliklərin edilməsi, öz ardınca onun əvvəlki metodikalarının istifadəçilərinin anlaşılmazlıq və inkarına gətirib çıxara bilər.Mütəxəssislər tərəfindən qeyri aşkar müqaviməti aradan götürmək üçün, istifadəçilərin daha geniş əhatəsinin maraqlarını nəzərə almaq lazımdır. UML dili üzərindəki sonrakı iş bütün bu vəziyyətləri nəzərə almalı idi.

Bu dövrdə UML dilinin tərtibatının dəstəyi OMG (Object Management Group) konsorsiumun məqsədlərindən birinə çevrilir. Belə ki, OMG konsosiumu 1989-cu ildə CORBA obyekt və komponent texnologiyalarının standart- laşması üzrə təkliflərin tərtibatı məqsədiylə yaranmışdı, UML dili OMG işində ikinci strateji istiqamət statusunu aldı. Məhz OMG çərçivəsində Riçard Solinin rəhbərliyi altında tərtibatçı komandası yaradılır, hansı ki,

UML dilinin unifikasiyası və standartlaşması üzrə gələcək işi təmin etməli idi. 1995-ci ilin iyun ayında OMG, bütün böyük mütəxəssislər və OYAP metodologiyası üzrə konsorsiuma daxil olan şirkətlərin nümayəndələrinin müşavirəsini keçirdi, burada ilk dəfə beynəlxalq miqyasda, OMG himayədarlığı altında modelləşdirmə dillləri sahəsində industrial standartların axtarışının məqsədəuyğunluğu qəbul edilmişdi.

Q. Buçun, C. Rubaxın və A. Cekobsonun səyləri, 0.9 (1996, iyun) versiyalı və 0.91 (1996 oktyabr) versiyalı UML dilinin təsvir olunduğu ilk sənədlərin yaranmasına gətirib çıxartdı. RFP (Request For Proposal) təklif sorğusu statusuna malik bu sənədlər, UML dilinin bir çox müxtəlif mütəxəssis kateqoriyasının geniş müzakirəsində özünəməxsus katalizator rolunu oynadı. UML dilinə ilk rəy və reaksiyalar ona ayrıca anlayış və konstruksiyalarla əlavə edilməsinə işarə edirdi.

Həmin vaxt aydın oldu ki, bəzi şirkət və idarələr UML dilində öz biznesi üçün strateji maraq xəttini görürlər. Rational Software şirkəti UML dilinin 1.0 versiyasının ciddi təyininin tərtibatına resurs ayıran bir neçə idarələrlə birgə UML partnyorlarının konsorsiumunu yaratdı. Bu konsorsiuma daxil olan şirkətlər notasiyanın sonrakı daha dəqiq və ciddi işinin dəstəklənməsini təmin etdilər, bu da UML dilinin 1.0 versiyasının yaranmasına gətirib çıxartdı. 1997-ci ilin yanvarında RFD təkliflərinin sorğularına ilkin cavab variantı kimi UML 0.1 dilinin təsviri olan sənəd dərc olunmuşdu. Modelləşdirmə dilinin bu verisyası ki-

fayət qədər yaxşı təyin olunmuşdu, tələb olunan ifadə və güclə təmin edirdi və geniş məsələ class-ına həlli ehtimal edirdi.

İnstrumental CASE – vasitələr və onların prkatiki tədbiq diapazonu böyük dərəcədə müvafiq modelləşdirmə dilinin semantika və notasiyasının uğurlu təyinindən asılıdır. İnstrumental CASE-vasitələr və onların praktiki tədbiq diapazonu müvafiq modelləşdirmə dilinin semantikası və notasiyasının uğurlu tədbiqindən aslıdır. UML dilinin spesifikası bundan ibarətdir ki, o semantik metamodeli təyin edir, konkret modelin interfeysini və təsvir üsullarını və ya komponentlərin realizasiyanı yox.

1997-ci ilin yanvarında bir sıra digər şirkətlər OMG – nin baxılması üçün RFP təkliflərinin sorğusuna öz cavablarını təqdim etdilər. Sonradan bu şirkətlər UML parrtnyorlarına UML dilinə öz bəzi ideyalarını daxil etməyə təklif verərək daxil oldular. UML partnyorlarıyla birgə müştərək işin nəticəsində UML dilinin 1.1 -yenidən baxış keçirilmiş versiyası təklif edilmişdi. UML 1.1 dilinin tərtibatında UML 1.0 ilə müqayisədə dilin semantikasında böyük aydınlığa nail olunmasına, həmçinin yeni partnyorların təkliflərinin nəzərə alınmasına əsas diqqət yetirilmişdi. Dilin bu versiyası OMG-nin baxilmasına təqdim olunmuşdu və 1997-ci ilin noyabrında OMG standartı kimi razılaşdırılmış və qəbul olunmuşdu.

Bu dilin inkşafının növbəti mərhələsi 1999-cu ilin martında, OMG konsorsiumu tərəfindən UML 1.3 dilinin təsviri dərc olunduğu zaman bitdi.

UML dilinin statusu onun düzəliş və təkmilləşdirilməsi üçün hamıya açıqdır. UML dili özü kiminsə şəxsi mülkiyyəti deyildir və heç kim tərəfindən patentləşdirilməyib. Eyni zamanda UML abbreviaturası onun qanuni sahiblərinin ticarət markasıdır.

UML dili, OYAP-ın geniş məsələ class-larının müxtəlif istifadəçi və elmi cəmiyyətlər tərəfindən həlli üçün modelləşdirmə dili kimi tədbiq olunmasına istiqamətləndirilmişdir.