UNIVERSITATEA DE STAT DIN MOLDOVA

Cu titlu de manuscris
C.Z.U.: 004.434

LATUL GHEORGHE

ELABORAREA METODELOR ȘI MIJLOACELOR PENTRU CREAREA CURSURILOR DE INSTRUIRE ASISTATE DE CALCULATOR

121.03 – Programarea calculatoarelor

Teză de doctor în Informatică

Autor: _____ Latul Gheorghe

Conducător științific: _____ Căpățână Gheorghe,
doctor în științe tehnice,
profesor universitar.

Chișinău 2025

© Latul Gheorghe 2025

Cuprins

ADNOTARE

Gheorghe Latul. „Elaborarea metodelor și mijloacelor pentru crearea cursurilor de instruire asistate de calculator”

Teză de doctor în informatică, Chișinău, 2025.

Structura tezei: Teza a fost elaborată în Universitatea de Stat din Moldova, este scrisă în limba română și constă din introducere, patru capitole, concluzii generale și recomandări, Teza conține 115 pagini text de bază, 17 figuri și 1 tabelă, bibliografie din 113 titluri și 8 anexe. Rezultatele obținute sunt publicate în 28 lucrări științifice.

Cuvinte cheie: instruire, instruire asistată de calculator, e-Learning, no-code, low-code, frame, scenariul, limbajul de programare, interpretorul limbajului, tehnologii de proiectare a softului.

Scopul cercetării: crearea tehnologiei originale și instrumentelor informatice pentru proiectarea și programarea cursurilor de instruire asistată de calculator.

Obiectivele cercetării: elaborarea metodelor și mijloacelor de programare a cursurilor de instruire asistată de calculator, formularea cerințelor principale față de sistemele de instruire asistate de calculator; formularea cerințelor față de limbajul descrierii cursurilor asistate de calculator, elaborarea limbajului formal de descriere a cursurilor asistate de calculator; elaborarea compilatorului și prototipului interpretorului limbajului elaborat.

Noutatea și originalitatea științifică a lucrării constă în noi abordări ale proiectării programelor de cursuri de instruire asistată de calculator bazate pe situații stereotipice de instruire, conceptul cărora este formulat în teză. A fost dezvoltat un limbaj de programare declarativ specific domeniului destinat pentru crearea cursurilor de instruire asistată de calculator bazat pe scenarii cauzale, care permite păstrarea cunoștințelor în frame-uri.

Problema științifică importantă rezolvată în domeniul de cercetare constă în dezvoltarea tehnologiei de proiectare a programelor d e instruire asistată de calculator bazată pe descriere situațiilor instructive stereotipice, definirea scenariilor instructive în baza scenariilor cauzale, elaborarea limbajului de programare specific domeniului de creare a cursurilor de instruire asistate de calculator.

Rezultatul obținut, care contribuie la soluționarea unei probleme științifice importante constă în prezentarea cunoștințelor în cadrul cursurilor de instruire asistată de calculator cu ajutorul scenariilor cauzale – unei varietăți de frame-uri.

Semnificația teoretică constă în elaborarea unei tehnologii inteligente, orientate *frame,* de proiectare a cursurilor de instruire asistată de calculator în baza *scenariilor cauzale*.

Valoarea aplicativă a lucrării: constă în elaborarea unui limbaj original de descriere a cursurilor de instruire asistată de calculator MoCo, a translatorului acestui limbaj și a unei tehnologii informaționale inteligente de elaborare efectivă a cursurilor de instruire asistate de calculator.

Implementarea rezultatelor lucrării. Tehnologii de proiectare a cursurilor asistate de calculator, limbajul MoCo și interpretorul lui elaborate în teza a fost utilizate la crearea cursului de instruire asistat de calculator „Programarea în limbajul C”. Cursul a fost implementat în procesul de studii la Facultatea de Matematică și Informatică la disciplina Fundamentele de Programare.

ABSTRACT

Gheorghe Latul: „Methods and technique development of computer assisted courses creation”

Doctoral thesis in Computer Science, Chisinau, 2025.

Structure of the thesis: the work on the thesis was carried out at the State University of Moldova, it is written in Romanian and consists of an introduction, 4 chapters, general conclusions and recommendations, a bibliography of 113 titles, as well as eight appendices. The dissertation contains 115 pages of main text, 17 figures and 1 table. The main results of the thesis have been published in 28 scientific papers.

Keywords: instruction, computer assisted instruction, e-Learning, no-code, low-code, scenario, programming language, language interpreter, software design technologies.

The purpose of the research: to create original technology and IT tools for the design and programming of computer-assisted instruction courses.

Research objectives: development of methods and tools for programming computer-assisted courses, formulation of the main requirements for to computer-assisted instructional systems; formulation of requirements for the language of the language of computer-assisted course description, development of a formal language for describing computer-assisted courses; development of a compiler and a prototype of the interpreter of the language under development.

Scientific novelty and originality of the thesis consists in new approaches to designing programmes of computer assisted instruction courses based on stereotypical instructional situations, the concept which is formulated in the thesis. A domain specific declarative programming language for computer assisted instruction courses based on causal scenarios has been developed, it makes possible to store domain knowledge in frames.

The obtained result, which contributes to the solution of an important scientific problem, consists in the representation of knowledge in computer assisted instruction courses by means of causal scenarios – a type of frames.

The theoretical significance consists in the development of an intelligent, *frame-oriented* technology for designing computer-assisted instruction courses based on *causal scenarios*.

The applied value of the work: consists in the development of an original language MoCo for describing MoCo computer-assisted training courses, the translator of this language and an intelligent information technology for the effective development of computer-assisted training courses.

Implementation of the results of the work. Computer-aided course design technologies, the MoCo language and its interpreter developed in the thesis were used in the creation of the computer-aided training course "Programming in C language". The course was implemented in the study process at the Faculty of Mathematics and Computer Science in the discipline Fundamentals of Programming.

АННОТАЦИЯ

Георге Латул: „Разработка методов и средств создания компьютерных учебных курсов”.

Докторская диссертация по информатике, Кишинёв, 2025.

Структура диссертации: работа над диссертацией велась в Государственном Университете Молдовы, написана на румынском языке и состоит из введения, 4 глав, общих выводов и рекомендаций, библиографии из 113 наименований, а также восьми приложений. Диссертация содержит 115 страниц основного текста, 17 иллюстраций и 1 таблицу. Основные результаты диссертации опубликованы в 28 научных работах.

Ключевые слова: обучение, автоматизированное обучение, *e-Learning*, *no-code, low-code*, фрейм, сценарий, язык программирования, интерпретатор языка, технологии проектирования программного обеспечения.

Цель исследования: создание оригинальной технологии и ИТ-инструментов для проектирования и программирования компьютерных учебных курсов.

Задачи исследования: разработка методов и средств программирования автоматизированных учебных курсов, формулировка основных требований к автоматизированным обучающим системам; формулирование требований к языку описания компьютерных курсов, разработка формального языка описания компьютерных курсов; разработка компилятора и прототипа интерпретатора разрабатываемого языка.

Научная новизна и оригинальность работы состоит в новых подходах к проектированию программ компьютерных учебных курсов, основанных на стереотипных учебных ситуациях, понятие о которых сформулировано в диссертации. Разработан предметно ориентированный декларативный язык программирования компьютерных учебных курсов на основе каузальных сценариев, позволяющий хранить знания предметной области в фреймах.

Полученный результат, который вносит вклад в решение важной научной проблемы состоит в представлении знаний в компьютерных обучающих курсах с помощью каузальных сценариев –разновидности фреймов**.**

Теоретическая значимость заключается в разработке интеллектуальной, фрейм-ориентированной технологии для проектирования автоматизированных учебных курсов на основе каузальных сценариев.

Практическое значение работы: заключается в разработке оригинального языка описания компьютерных учебных курсов MoCo, транслятора этого языка и интеллектуальной информационной технологии для эффективной разработки компьютерных учебных курсов.

Внедрение результатов работы. Технологии проектирования компьютерных курсов, язык MoCo и его интерпретатор, разработанные в данной работе, были использованы при создании автоматизированного учебного курса «Программирование на языке Си». Курс был внедрен в учебный процесс на факультете Математики и Информатики по дисциплине Основы программирования**.**

Abrevierile utilizate în teză

SIAC Sistemul de Instruire Asistat de Calculator. reprezintă o platformă sau un ansamblu de instrumente software utilizate pentru a sprijini procesul de învățare și predare prin intermediul tehnologiei informatice..

BNF Backus–Naur form, Forma Backus-Naur este un sistem formal de descriere a sintaxei în care unele categorii sintactice sunt definite în mod consecvent prin intermediul altor categorii. BNF este utilizat pentru a descrie gramatici formale fără context.

LL Left Leftmost (gramatica). O gramatică LL este un tip de gramatică formală utilizată în analiza sintactică (parsing) descendentă, care analizează un șir de intrare de la stânga la dreapta (Left-to-right) și construiește derivarea stângă (Leftmost derivation) a acestuia.

MTP Mașina de Traducere Predictivă este un automat finit utilizat în procesul de analiză sintactică descendentă pentru a determina dacă un șir sau codul sursă de intrare aparține unui limbaj formal, conform unei gramatici date. Aceasta funcționează pe baza parsingului predictiv (reguli de producție definite de gramatici formale), utilizând o tabelă de decizie construită.

LI Limbajul Intermediar este o formă de reprezentare a codului sursă într-un format intermediar, care este generat în timpul procesului de compilare sau interpretare și care servește ca o etapă de tranziție între codul sursă și codul mașină sau alt limbaj de nivel înalt.

MoCo Modeler of Courses. Denumirea limbajului descrierii cursurilor de instruire asistate de calculator elaborat de autor.

MDE Model-Driven Engineering – MDE, Ingineria/Proiectarea Dirijată de Modele – este o abordare a dezvoltării de software în care modelele formale devin principalul artefact de proiectare, dezvoltare și întreținere a sistemelor informatice.

GPL Un limbaj cu scop general (*eng*. General-purpose Programming Language – GPL) este un tip de limbaj de programare care nu este specializat pentru o singură sarcină, ci poate fi utilizat pentru a crea software pentru diverse aplicații.

LSD Limbaj Specific unui Domeniu (*eng*. Domain-Specific Language – DSL) este un limbaj de programare sau un limbaj de descriere specializat, conceput pentru a rezolva probleme specifice unui anumit domeniu de aplicare. LSD este optimizat pentru eficiență și expresivitate într-un domeniu restrâns.

LCDP Platforma de dezvoltare cu cod redus (*eng*. **L**ow-**c**ode **D**evelopment **P**latform) este un mediu de dezvoltare software care permite crearea rapidă, cu un minim de scriere manuală de cod.

NCDP Platforma de dezvoltare fără cod (*eng*. **N**o-**c**ode **D**evelopment **P**latform) este un mediu de dezvoltare software care permite utilizatorilor să creeze aplicații fără a scrie cod manual. Aceste platforme utilizează componente, eliminând necesitatea programării tradiționale.

IMS Instructional Management System – este un sistem informatic conceput pentru a organiza, gestiona și livra activități de instruire, facilitând planificarea lecțiilor, distribuția materialelor educaționale și urmărirea progresului cursanților într-un mod eficient și standardizat.

INTRODUCERE

Actualitatea temei de cercetare

În ultimele decenii, informatizarea a pătruns profund în toate domeniile vieții sociale, economice și științifice, generând transformări structurale și funcționale majore. În mod particular, domeniul educației a resimțit în mod direct acest impact, prin digitalizarea resurselor, diversificarea metodelor de predare și integrarea progresivă a tehnologiilor informatice în procesul de instruire. . În prezent, procesul de formare inițială, perfecționare și reconversie profesională a cadrelor se sprijină tot mai mult pe utilizarea sistemelor informatizate de instruire, menite să sporească eficiența și accesibilitatea actului educațional [1].

Instruire asistată de calculator a cunoscut o expansiune considerabilă în ultimele două decenii, impulsionată de accesul tot mai larg la tehnologie și de necesitatea instruirii la distanță. Industria e-learning a crescut exponențial – de exemplu, între anii 2000 și 2020 s-a înregistrat o creștere[1] de 900%.

Evenimente recente, precum pandemia globală, au accelerat adoptarea instruirii online, ducând la mutarea cursurilor în spațiul virtual și la o creștere masivă a cererii de conținut educațional digital[2].

În acest context, dezvoltarea instrumentelor pentru crearea cursurilor de instruire asistată de calculator (IAC) a devenit extrem de actuală și importantă. Instituțiile educaționale și companiile caută soluții eficiente pentru a produce materiale de învățare interactive și adaptabile, fără a compromite calitatea educațională.

Pe lângă contextul educațional, progresul tehnologic a deschis noi oportunități și provocări. Apariția unor tehnologii precum aplicațiile web complexe, dispozitivele mobile, realitatea virtuală sau integrarea inteligenței artificiale în educație a ridicat nivelul de așteptare față de cursurile digitale. Cursurile moderne includ adesea activități interactive, simulări, colaborare online și adaptare la ritmul sau nivelul fiecărui cursant. Pentru a face față acestei complexități, instrumentele de autor (*eng*. authoring tools) pentru cursuri IAC trebuie să permită proiectarea și gestionarea *scenariilor de învățare* tot mai elaborate, menținând totodată ușurința de utilizare pentru experții educaționali care nu sunt neapărat programatori. Astfel, se manifestă o nevoie acută de abordări care să reducă decalajul dintre conceptul pedagogic și implementarea tehnică a unui curs [2]. Acest lucru conferă subiectului

[1] **Latest e-learning Statistics 2025**. Available at: https://elearningstats.education. [Accesed 02.05.2025]

[2] **IEEE Learning Technology Standards Committee (LTSC)** Available at: https://www.ieeeltsc.org/ [Accesed 02.05.2025]

o actualitate deosebită: cercetarea explorează modalități de a facilita crearea cursurilor IAC de către practicieni, folosind abstracții de nivel înalt și automatizând detaliile tehnice.

În acest context, procesul de formare inițială, perfecționare și reconversie profesională se sprijină tot mai mult pe utilizarea sistemelor informatizate de instruire, menite să sporească eficiența și accesibilitatea actului educațional. Astfel**,** sistemele de instruire asistată de calculator (IAC) s-au afirmat ca soluții inovatoare și viabile în sprijinul procesului educațional, oferind funcționalități de predare, învățare și evaluare într-un format digital integrat. Acestea completează sistemele tradiționale prin introducerea de componente interactive și adaptive, susținând emergența unor noi forme de predare precum învățarea bazată pe scenarii, personalizarea traseului educațional și feedback-ul automatizat [1].

Utilizarea calculatorului în procesul de instruire facilitează o asimilare mai rapidă a conținutului didactic, sporește implicarea activă a cursantului și optimizează activitatea cadrului didactic prin automatizarea sarcinilor de rutină. În plus, implementarea rețelelor locale și globale, a platformelor de tip LMS (Learning Management System) și a instrumentelor de tip no-code pentru crearea de conținut educațional permite dezvoltarea cursurilor cu costuri reduse, fără a afecta calitatea instruirii. Platforme precum Moodle, Canvas, Google Classroom sau instrumente interactive precum H5P au devenit componente esențiale în procesul educațional contemporan [3].

Cu toate acestea, practica educațională evidențiază existența unui decalaj semnificativ între potențialul tehnologic al acestor instrumente și capacitatea efectivă a cadrelor didactice de a le valorifica. Proiectarea cursurilor asistate de calculator este realizată adesea în mod intuitiv, pe baza experienței pedagogice individuale, reflectate în planificările lecțiilor, materialele didactice, scenariile educaționale sau testele de evaluare. Lipsa unui instrumentar formalizat, accesibil și adaptat logicii procesului instructiv constituie o provocare în dezvoltarea sustenabilă a cursurilor digitale.

Astfel, se conturează necesitatea dezvoltării unui instrument formal de modelare a cursurilor, care să permită transpunerea eficientă a experienței pedagogice în sisteme informatizate. Un astfel de instrument îl poate reprezenta limbajul formal de descriere a situațiilor instructive, însoțit de un analizor și un interpretator pentru sistemele IAC. Un avantaj important al acestei soluții îl constituie portabilitatea. Acest aspect este în concordanță cu cerințele actuale de interoperabilitate promovate de standarde internaționale [3] precum IMS (Instructional Management System) Learning Design[3].

O direcție promițătoare în dezvoltarea instrumentelor educaționale este utilizarea limbajelor declarative specifice domeniului Un LSD (Limbaj Specific Domeniului) este un limbaj conceput

[3] **IEEE Learning Technology Standards Committee (LTSC)** Available at: https://www.ieeeltsc.org/ [Accesed 02.05.2025]

special pentru un anumit domeniu de aplicație (în cazul de față, educația asistată de calculator), oferind o sintaxă și semantici adecvate pentru a descrie *ce* trebuie realizat în acel domeniu, fără a detalia *cum* se realizează la nivel de cod. În contextul proiectării cursurilor IAC, LSD-urile permit definirea structurii unui curs – activități de învățare, resurse, secvențe de pași, roluri (student, profesor etc.) – într-o manieră declarativă, orientată pe conceptul pedagogic, urmând ca platforma software să interpreteze această specificație și să o transforme în implementare executabilă (ex. pagini web interactive, module e-learning, integrare în Sistemul de Management Educațional etc.).

Avantajele utilizării LSD-urilor în proiectarea cursurilor IAC sunt multiple. În primul rând, LSD-urile reduc complexitatea tehnică percepută de către autorii de conținut educațional. Crearea manuală a unui curs interactiv implică adesea sarcini repetitive, cod duplicat dificil de întreținut, adaptare pentru diferite platforme (desktop, web, mobil) și un risc mare de erori [4].

Un limbaj specific domeniului abstractizează aceste detalii: autorul cursului specifică logica instruirii (de exemplu, ordinea activităților, conținutul și regulile pedagogice), iar motorul din spate se ocupă de *gestionarea resurselor, generarea codului și asigurarea compatibilității tehnice* pe diverse platforme

Astfel, LSD-urile scad necesarul de cunoștințe de programare pentru designerii instrucționali, permițându-le să se concentreze pe aspectele educaționale. Studii în domeniu arată că un editor grafic bazat pe LSD poate îmbunătăți semnificativ experiența autorului comparativ cu un editor tehnic tradițional; de pildă, un mediu de design grafic s-a dovedit mai intuitiv și mai ușor de gestionat vizual decât notările arborescente reflexive folosite anterior în dezvoltarea model-driven [4]. Prin urmare, LSD-urile sporesc accesibilitatea instrumentelor de autor, făcând procesul de creare a cursurilor mai accesibil și mai eficient.

În al doilea rând, LSD-urile pot încorpora *bune practici pedagogice* și șabloane reutilizabile. Pentru că limbajul este construit în jurul conceptelor domeniului educațional, el poate oferi construcții predefinite care reflectă modele pedagogice consacrate. De exemplu, într-un LSD educațional se pot furniza șabloane pentru scenarii de învățare și învățare adaptivă. LSD ajută profesorii să creeze design-uri de activități eficace, pornind de la modele validate în practică. Această posibilitate de reutilizare a elementelor de design didactic crește calitatea cursurilor (prin aplicarea unor strategii testate) și eficiența dezvoltării (prin economie de timp și efort în definirea de la zero a fiecărui scenariu).

În concluzie, stadiul actual al domeniului evidențiază o tendință clară de digitalizare a educației, dar și nevoia stringentă de instrumente specializate care să faciliteze crearea de conținut educațional de calitate, fără a presupune expertiză tehnică avansată din partea autorilor de cursuri.

***Ipoteză de cercetare*.** Elaborarea unui limbaj declarativ specific domeniului și a unei tehnologii de proiectare eficientă, flexibilă și scalabilă a cursurilor de instruire asistată de calculator, aplicabile în diverse contexte educaționale.

Scopul și obiectivele cercetării. Scopul cercetării constă în crearea tehnologiei originale și instrumentelor informatice pentru proiectarea și programarea cursurilor de instruire asistată de calculator. Scopul a determinat următoarele obiective:

- elaborarea metodelor și mijloacelor de programare a cursurilor de instruire asistată de calculator,
- formularea cerințelor principale față de sistemele de instruire asistate de calculator,
- formularea cerințelor față de limbajul descrierii cursurilor asistate de calculator,
- elaborarea limbajului specific domeniului de descriere a cursurilor asistate de calculator,
- elaborarea unei tehnologii de proiectare a cursurilor de instruire asistată de calculator,
- dezvoltarea compilatorului și prototipului interpretorului limbajului elaborat.

Problema de cercetare este în crearea tehnologiei de proiectare originale a programelor de instruire asistată de calculator bazată pe descriere situațiilor instructive stereotipice, definirea scenariilor instructive în baza scenariilor cauzale cu scopul elaborării limbajului de programare specific domeniului de creare a cursurilor de instruire asistate de calculator.

Semnificația teoretică și aplicativă

Semnificația teoretică a lucrării constă în analiza originală a procesului didactic, formalizarea unei submulțimi a acestui proces, introducerea și descrierea situațiilor de instruire stereotipice. În lucrare este introdusă noțiunea de *scenariu de instruire* în baza *scenariului cauzal*.

Semnificația practică a lucrării constă în crearea limbajului descrierii cursurilor de instruire asistate de calculator. Cu ajutorul acestui limbaj a fost creat și implementat cursul de instruire asistat de calculator „*Programarea în limbajul C*”, confirmat de actul de implementare în procesul de instruire a cursului elaborat.

Metodologia cercetării.

Cercetările efectuate în teza au la bază analiza sistemelor de instruire asistată de calculator, a sistemelor de inteligența artificială, modelare informatică în ingineria soft-ului, teoria gramaticelor formale și automatelor, teoria compilării. Ideile și soluțiile propuse au fost testate prin crearea prototipurilor soft-ului.

Aprobarea rezultatelor.

Rezultatele expuse în teză au fost publicate în [5-34] și comunicate în cadrul următoarelor conferințe:

- International Conference on Computer Technologies in Education (ICCTE'93). – (Kiev, Ucraina: 1993);
- International Conference on Computer Technologies in Education (ICCTE'94). – (Crimeea, Ucraina: 1994);
- Tehnologii avansate în pragul secolului XXI. Conferința științifico-practică (Chișinău, Moldova, 5– 7 octombrie, 2000);
- Conferințele Corpului Didactico-Științific al Universității de Stat din Moldova (anii 1992– 2006).
- Conference „Mathematics & Information Technologies: Research and Education" (MITRE 2008 ,2011, 2015, 2016, 2023). Chișinău.
- În cadrul seminarelor Departamentului Informatică a Universității de Stat din Moldova;

O parte din rezultatele expuse în teză sunt obținute de autor în decursul activității la îndeplinirea unor proiecte din domeniul utilizării tehnicii de calcul în procesul de instruire:

- Elaborarea softului instrumental pentru crearea programelor de instruire asistate de calculator. Nr. înregistrării de stat 0195M00126, Nr. de inventar 0396M00141. Chișinău, USM, 1993-1995.
- Elaborarea CASE-tehnologiei de proiectare a cursurilor de instruire asistate de calculator. Nr. înregistrării de stat 0196M00877, Nr. de inventar 0298M00736. Chișinău, USM, 1997-1998. (executant responsabil).
- Proiect instituțional „Dezvoltarea sistemelor informaționale inteligente orientate spre familii de probleme decizionale cu aplicare în educație și cercetare", codul 15.817.02.38A. Chișinău, USM, 2007-2008.
- Proiectul Elaborarea și aplicarea metodelor inovaționale în instruire la distanță, codul 08.815.08.04A. Chișinău, USM, 2018-2019.
- A fost implementat cursul de instruire "Programarea în limbajul C" creat în baza soft-ului elaborat în teza.

Implementarea rezultatelor lucrării: programele instrumentale și cursul de instruire pe limbajul de programare C au fost implementate la Facultate de Matematică și Informatică a Universității de Stat din Moldova (vezi anexa nr.1).

Structura tezei: Teza este scrisă în limba română cu titlu de manuscris. Lucrarea este structură în patru capitole, concluzii generale și recomandări și bibliografie.

Sumarul compartimentelor tezei. În cadrul compartimentului **„Introducere”** este evidențiată importanța și actualitatea temei de cercetare abordate în teză. Sunt prezentate scopul și obiectivele lucrării, alături de contribuțiile științifice originale reflectate prin noutatea rezultatelor obținute. De asemenea, sunt expuse valoarea teoretică și aplicativă a cercetării, precum și informații privind validarea și confirmarea rezultatelor obținute.

Capitolul 1. *„Sistemele de instruire asistată de calculator”* se deschide cu prezentarea unei clasificări a acestor sisteme, realizată în funcție de criteriul destinației funcționale a fiecărui tip de sistem. Clasificarea urmărește să evidențieze diversitatea abordărilor existente în domeniul instruirii asistate de calculator și să fundamenteze alegerea direcției de cercetare adoptate în teză.

În continuare, capitolul include o trecere în revistă a sistemelor de autor (*authoring systems*), considerate componente esențiale ale infrastructurii moderne pentru proiectarea, dezvoltarea și întreținerea cursurilor de instruire asistată de calculator. Sunt analizate funcționalitățile acestor instrumente, avantajele și limitările lor, precum și modul în care acestea pot fi utilizate de cadre didactice sau designeri instrucționali fără o pregătire avansată în programare.

Totodată, se realizează o analiză a caracteristicilor limbajelor de programare utilizate în cadrul diverselor sisteme de autor. Această analiză are rolul de a fundamenta necesitatea dezvoltării unor soluții mai flexibile și mai apropiate de nevoile autorilor cursurilor de instruire.

Capitolul se încheie cu formularea sarcinilor tezei, dintre care cea mai importantă constă în proiectarea și dezvoltarea unui limbaj evoluat de descriere a cursurilor de instruire asistată de calculator, bazat pe un sistem de frame-uri. De asemenea, este subliniată importanța elaborării unor mecanisme originale de interpretare a acestui limbaj, menite să asigure o procesare eficientă a descrierilor și o generare automată a aplicațiilor educaționale corespunzătoare. Aceste direcții de cercetare definesc nucleul contribuției științifice aduse de teză.

Capitolul 2. *„Modelul structural al procesului de instruire pentru cursurile de instruire asistată de calculator”* are ca obiectiv central elaborarea unei tehnologii de proiectare a cursurilor de instruire asistată de calculator, prin identificarea și formalizarea componentelor esențiale ale procesului instructiv. În acest sens, sunt formulate cerințele conceptuale și funcționale față de limbajul de programare destinat definirii acestor cursuri, cerințe care derivă atât din specificul domeniului educațional, cât și din necesitatea de a asigura un grad înalt de flexibilitate și reutilizare a conținutului didactic.

În prima parte a capitolului este analizat procesul general de învățământ, cu accent pe dinamica interacțiunii dintre participantul la instruire și mediul educațional digital. Pe baza acestei analize, sunt

identificate și evidențiate un set de situații instructive stereotipice, care urmează a fi utilizate ca unități de bază în proiectarea cursurilor de instruire asistată de calculator. Aceste situații oferă un cadru conceptual pentru reprezentarea proceselor educaționale într-un mod sistematic.

Ulterior, este prezentat și analizat modelul structural al procesului de instruire, incluzând atât componentele logice ale activității educaționale, cât și mecanismele de dirijare a succesiunii instruirii. Modelul are la bază utilizarea scenariilor cauzale, construite în mod formal cu ajutorul frame-urilor (cadrelor cognitive), care servesc drept instrumente pentru modelarea și reprezentarea cunoștințelor în cadrul cursurilor. Fiecare frame este compus dintr-un număr finit de sloturi, fiecare definit printr-un nume și o valoare asociată, reflectând astfel structura informațională a unei situații instructive. Această abordare permite descrierea clară, modulară și extensibilă a conținutului educațional, precum și adaptarea la diferite contexte și obiective.

Pe baza acestei structuri, este definit și scenariul de instruire, conceput ca o extensie a scenariului cauzal, menit să descrie succesiunea logică și funcțională a situațiilor instructive stereotipice. Acesta oferă o formă formalizată de proiectare a activităților de învățare, contribuind la organizarea coerentă și eficientă a cursurilor asistate de calculator. Se evidențiază, de asemenea, avantajele acestei metode de reprezentare în raport cu abordările tradiționale, inclusiv posibilitatea automatizării a generării de programe și creșterea consistenței a acestora.

Capitolul se încheie cu o serie de recomandări metodologice privind proiectarea cursurilor în conformitate cu modelul propus, precum și cu concluzii referitoare la avantajele utilizării frame-urilor și scenariilor cauzale în reprezentarea proceselor educaționale. Aceste concluzii oferă fundamentul teoretic pentru capitolele următoare, în care va fi prezentată implementarea practică a limbajului și a mediului de programare asociat.

Capitolul 3. *„Limbajul de descriere a cursurilor asistate de calculator MoCo”* este dedicat prezentării limbajului formal elaborat în cadrul acestei teze, destinat descrierii situațiilor instructive specifice cursurilor de instruire asistată de calculator. Limbajul MoCo [23] a fost conceput pentru a susține procesul de proiectare și programare a cursurilor, oferind autorilor de cursuri un instrument expresiv și accesibil.

Limbajul propus permite reprezentarea atât a componentei informaționale a cursului, prin care se transmit cunoștințele către cursant, cât și a componentei de control, responsabilă de logica desfășurării activităților de învățare. În structura sa sunt incluse mijloace pentru organizarea materialului didactic, pentru prezentarea informațiilor în formate variate (text, imagini, elemente grafice și video), pentru analiza automată a răspunsurilor oferite de cursanți, precum și pentru organizarea activităților de verificare și evaluare.

De asemenea, limbajul integrează mecanisme pentru furnizarea de informații de îndrumare, feedback contextual și funcționalități auxiliare necesare pentru crearea cursurilor interactive. Un aspect definitoriu al limbajului este caracterul său neprocedural, ceea ce înseamnă că autorul specifică *ce* trebuie să se întâmple în cadrul procesului instructiv, fără a descrie explicit *cum* se realizează fiecare acțiune.

Limbajul MoCo se bazează conceptual pe noțiunea de situație instructivă, modelată sub forma unui scenariu cauzal, în care acțiunile cursantului determină secvența următoare a instruirii. Pentru formalizarea sintaxei, limbajul este descris utilizând notațiile Backus–Naur, ceea ce asigură claritate și rigurozitate în definirea regulilor de utilizare. În completare, anexa tezei conține diagrame sintactice care ilustrează structura internă a limbajului, facilitând înțelegerea și aplicarea sa practică.

Prin aceste caracteristici, limbajul MoCo oferă un suport solid pentru dezvoltarea cursurilor digitale, răspunzând cerințelor moderne ale instruirii asistate de calculator.

Capitolul 4. „Proiectarea platformei pentru crearea cursurilor de instruire asistată de calculator” este consacrat descrierii procesului de dezvoltare a mijloacelor software necesare pentru elaborarea cursurilor de instruire asistată de calculator, având ca obiectiv general realizarea unei platforme dedicate acestui scop. Platforma a fost concepută pentru a furniza atât programatorilor, cât și specialiștilor din diverse domenii de activitate un mediu prietenos și eficient pentru proiectarea și implementarea programelor educaționale asistate de calculator.

Principalul rezultat constă în faptul că platforma permite, în etapa finală a procesului, generarea automată a codului într-un format interpretabil având ca sursă programul scris în limbajul MoCo, limbaj formal dezvoltat în cadrul acestei teze. Astfel, utilizatorii pot descrie scenarii instructive complexe, care ulterior sunt convertite într-un cod logic formal, pregătit pentru execuție într-un mediu corespunzător.

Pentru atingerea acestui obiectiv, problema generală a fost divizată în mai multe subprobleme tehnice, abordate secvențial în procesul de dezvoltare a platformei. O primă etapă a constat în implementarea procesului de translare a programului sursă într-un cod intermediar, etapă în care s-au elaborat componente specializate pentru analiza lexicală, sintactică și semantică. Analiza sintactică s-a bazat pe o gramatică de tip LL(1), care oferă un cadru formal eficient pentru procesarea predictivă a structurilor limbajului.

Pentru analiza sintactică propriu-zisă s-a utilizat un analizator predictiv stâng (1-predictiv), cunoscut pentru eficiența sa în procesarea descendentă a limbajelor formale. Acest algoritm funcționează pe baza unei succesiuni de lexeme de intrare, a unei memorii de tip stivă și a unui flux

de ieșire, toate coordonate prin intermediul unui tabel de dirijare care guvernează activitatea Mașinii de Traducere Predictivă. Această mașină este un automat finit specializat, utilizat pentru a determina dacă un anumit șir de simboluri aparține limbajului formal definit de gramatica specificată. Funcționarea sa se bazează pe parsing-ul predictiv, adică pe aplicarea regulilor de producție ale gramaticii pentru a anticipa structura corectă a programului sursă. Tabela de decizie aferentă fiecărui limbaj asigură corectitudinea și conformitatea traducerii cu sintaxa formală prestabilită.

O etapă importantă a procesului o constituie generarea codului interpretabil, care se realizează pe baza structurii codului intermediar. Acesta este compus dintr-o succesiune de coduri ale operatorilor și de date, asociate fiecărei structuri logice din cadrul limbajului MoCo. Astfel, se creează o corespondență clară între fiecare construcție specifică limbajului MoCo și un șablon corespunzător în limbajul intermediar, fapt ce facilitează interpretarea și execuția ulterioară a cursurilor.

Pentru a sprijini procesul de creare a cursurilor, în cadrul cercetării a fost elaborat un prototip al unei platforme de tip no-code. Platformele no-code reprezintă o direcție actuală și inovatoare în ingineria software, oferind posibilitatea dezvoltării de aplicații fără necesitatea scrierii efective de cod. Acestea se bazează pe interfețe grafice intuitive și pe mecanisme de configurare declarativă, care permit utilizatorilor să definească structura și comportamentul aplicației prin operații vizuale. Această abordare este deosebit de relevantă în contextul instruirii asistate de calculator, întrucât autorii cursurilor sunt adesea experți în domeniul de conținut, fără competențe avansate în programare. Platforma propusă răspunde acestei nevoi, oferindu-le acestora un instrument accesibil prin care pot proiecta și implementa cursuri interactive, structurate și adaptabile, fără a apela la dezvoltatori software.

Funcționalitatea platformei include: selectarea și combinarea situațiilor instructive predefinite, inserarea de conținut multimedia, definirea regulilor de navigare și control al cursului, precum și configurarea testelor și a mecanismelor de evaluare. Codul rezultat este generat automat în limbajul MoCo, ceea ce asigură integrarea cu motorul de interpretare și facilitează adaptarea rapidă a cursurilor la cerințe diferite sau la evoluții ale conținutului.

Prin combinarea unui limbaj formal bine structurat cu o interfață no-code intuitivă, această platformă reprezintă un pas important către facilitarea procesului de creare a cursurilor asistate de calculator, contribuind la extinderea accesului la instrumente moderne de proiectare educațională și la îmbunătățirea calității resurselor digitale de învățare.

1. SISTEMELE DE INSTRUIRE ASISTATE DE CALCULATOR

1.1. Clasificarea sistemelor de instruire asistate de calculator

Utilizarea sistemelor de instruire asistate de calculator (SIAC) în procesul didactic de instruire facilitează soluționarea a mai multor probleme. SIAC trebuie să ajute profesorul și să-l înlocuiască la îndeplinirea anumitor funcții. Aplicarea SIAC în instruire asigură contactul permanent cu fiecare student în regim de dialog, accelerează procesul de instruire și verificare a cunoștințelor studenților, pune la dispoziția studentului mijloace moderne de calcul, căutare și prelucrare a informației. Scopul aplicării SIAC constă în ridicarea trăiniciei însușirii cunoștințelor, deprinderilor și abilităților, reducerea timpului de studii, dezvoltarea gândirii creatoare a studenților.

Sistemele de instruire asistate de calculator sunt o totalitate funcțional interdependentă de asigurare organizatorică, produse program și inginerești pe baza tehnicii de calcul și a rețelelor de calcul destinată optimizării procesului de instruire în diferite forme și care funcționează în regim de dialog. [35–49].

Clasificarea SIAC [56 -58] care asigură transmiterea și perceperea cunoștințelor, aprecierea calității instruirii este următoarea:

CAI	Computer Aided Instruction	Instruire asistată de calculator
CAL	Computer Aided Learning	Studiere cu ajutorul calculatorului
CBL	Computer Based Learning	Studiere pe baza calculatorului
CBT	Computer Based Training	Antrenare pe baza calculatorului
CAA	Computer Aided Assessment	Evaluare cu ajutorul calculatorului

Pe de altă parte, privite din punct de vedere a destinației lor programele și sistemele de instruire asistate de calculator pot fi clasificate în felul următor:

- sisteme autor care reprezintă mijloacele instrumentale pentru elaborarea programelor de instruire, antrenament și verificare. (Exemple de acest gen de sisteme pot servi АДОНИС, МИДОС, LINKWAY etc.);
- manuale electronice destinate pentru studierea de sine stătătoare de către studenți diferitelor discipline. (Exemple de acest gen de sisteme pot servi COGNITO, ADAPT);
- sisteme de căutare a informației (inclusiv dicționare, enciclopedii, sisteme informative);
- programe de modelare (inclusiv modele matematice, experiment de laborator, animare și grafică interactivă).

Sistemele de antrenare sunt cazuri particulare ale sistemelor de instruire. Aceste sisteme sunt destinate pentru consolidarea materialului studiat în prealabil, perfecționarea deprinderilor și a diferitelor îndemânări, a procedeelor care trebuie să fie reproduse de către studenți la nivelul adus până la automatism. Sistemele de antrenare pot intra în calitate de subsistem în SIAC. Manualele

electronice și sistemele de căutare a informației sunt mijloace de instruire „pasive” și în lucrarea prezentată nu se examinează. Programele de modelare sunt produse program care se elaborează ținând cont de specificul îngust al domeniului investigat și pentru fiecare caz aparte se cer abordări speciale.

1.2. Analiza comparativă a sistemelor de autor

În domeniul instruirii asistate de calculator (IAC), de-a lungul anilor au fost dezvoltate numeroase **sisteme de autor** – platforme software care permit proiectarea și realizarea de cursuri interactive pe calculator. Aceste sisteme prezintă o varietate de paradigme de concepere a lecțiilor asistate. Pentru avaliza vom grupa aceste sisteme pe categorii tematice (de exemplu, sisteme bazate pe cadre, sisteme cu meniuri, sisteme multimedia etc.) și le vom descrie caracteristicile tehnice principale și vom oferi observații critice privind limitele lor.

Sisteme de autor bazate pe cadre (ecrane)

O primă categorie importantă este cea a **sistemelor bazate pe cadre**. În contextul IAC, termenul *cadru* (*frame*) se referă aici la ecranele sau paginile individuale de curs proiectate de autor. Aceste sisteme permit crearea unei secvențe de *cadre* – fiecare cadru conținând conținut educațional (text, grafică, întrebări etc.) – care sunt afișate cursanților exact în forma proiectată. Legăturile dintre cadre, ramificările și ordinea parcurgerii reprezintă structura lecției, stabilită de autor în timpul proiectării.

Un exemplu clasic este MICROTEXT [50], un sistem de autor dezvoltat la National Physical Laboratory (Marea Britanie). MICROTEXT oferă autorilor posibilitatea de a crea dialoguri om-calculator structurate în cadre ecran, integrând text și elemente grafice de tip teletext. Fiecare ecran (cadru) proiectat în MICROTEXT corespunde exact unei pagini pe care studentul o va vedea în programul final. Sistemul include un editor dedicat pentru text și grafică, permițând schimbarea ușoară a layout-ului, culorilor și elementelor vizuale până la obținerea prezentării dorite. Cadrele pot fi stocate și accesate aleatoriu în funcție de ramificările interactive definite de autor. Fiecare lecție conține o mulțime de cadre care reprezintă un element de text sau un fragment al programului cursului. Fiecare cadru are numele său. Cadrul se termină cu datele de dirijare care determină trecerea la cadrele următoare. În total autorul utilizează 17 comenzi pentru scrierea programului. Numărul de comenzi nu este mare și ele pot fi utilizate de către utilizatorii necalificați pentru crearea cursurilor noi.

Astfel, MICROTEXT a facilitat crearea de lecții ramificate, unde succesiunea cadrelor depinde de răspunsurile studentului. Această abordare *bazată pe cadre* a simplificat dezvoltarea lecțiilor IAC, autorul putând vizualiza direct fiecare ecran al cursului și având control asupra legăturilor dintre ecrane.

Un alt exemplu de sistem bazat pe cadre este STAF [50] (Science Teacher's Authoring Facility). STAF a fost dezvoltat ca un mediu de autor dedicat disciplinelor cu conținut științific. Acest sistem oferea autorilor un set de cadre predefinite și mecanisme de interacțiune. Sistemul constă din două componente: prima asigură scrierea programelor de instruire, a doua execută aceste programe și realizează dialogul de instruire.

STAF se baza tot pe conceptul de ecrane parcurse secvențial sau ramificat. O caracteristică importantă era interfața de creare relativ prietenoasă: în locul programării linie cu linie, autorul folosea opțiuni de meniu și șabloane pentru a popula cadrele cu întrebări, explicații și elemente grafice. În acest sens, STAF face trecerea către categoria următoare (sisteme cu meniuri), fiind un exemplu hibrid: lecțiile rezultate erau structurate pe cadre, dar procesul de creare implica selectarea din meniuri a tipurilor de elemente educaționale și a traseelor de ramificare. STAF a fost utilizat în mediul universitar, demonstrând eficiența conceptului bazat pe cadre pentru organizarea materiei și avantajul unei editări interactive a conținutului.

Menționăm și AOS VUZ [56] (Автоматизированная Обучающая Система – ВУЗ, adică sistem automatizat de instruire pentru universități), un proiect de mare amploare din spațiul ex-sovietic. AOS VUZ a fost implementat încă de la începutul anilor 1980 în numeroase instituții (circa 100 de universități în faza experimentală) și a folosit tot o abordare bazată pe cadre și ramificări de tip *Crowder*.

În AOS VUZ, autorul unui curs putea crea ecrane de curs (texte, întrebări) direct de la terminalul său, sistemul facilitând modificarea rapidă a materialului educațional. Fiecare cadru de curs conținea unități de informație sau întrebări, iar sistemul includea logica de prezentare adaptivă: în funcție de răspunsurile studentului la un cadru, sistemul decidea următorul pas, permițând **ramificarea** traseului de învățare.

AOS VUZ a demonstrat fezabilitatea administrării centralizate a cursurilor IAC și posibilitatea ca autorii să dezvolte material direct într-un mediu instituțional. Ulterior, a apărut și AOS-VUZ/Micro, o versiune adaptată pentru microcalculatoarele autohtone ale vremii, extinzând accesul la acest sistem. Limitările AOS VUZ erau legate de infrastructura necesară dar conceptul de cadre cu feedback imediat de la student a fost un pas înainte important, introducând elemente de adaptivitate (trasee individuale în funcție de performanța studentului)

Sistemul ASTRA [50], menționat adesea alături de AOS în literatura de specialitate este un alt sistem bazat pe cadre de învățare și testare, orientat către cursuri cu alegere selectivă. Sistemul ASTRA a fost elaborat la Centrul de Calcul al Institutului Economiei și Transportului din Moscova. ASTRA folosește principii similare: afișa materialul în segmente discrete (ecrane) și permite ramificarea în funcție de răspunsurile cursanților, implementând astfel în practică conceptul de

predare programată adaptivă. Sistemele ASTRA și AOS pot fi considerate reprezentative pentru cursuri bazate pe cadre în contextul academic, anticipând unele trăsături ale tutorilor inteligenți (de exemplu, evidențierea necesității unui *model al cursantului* și adaptarea parcursului educațional).

În aceeași categorie a sistemelor bazate pe cadre se înscrie și LINKWAY [50**]**, un sistem de autor lansat de IBM dedicat realizării de lecții multimedia pe PC-urile IBM. LINKWAY era puternic influențat de paradigma HyperCard (de la Apple) și permitea autorilor să creeze *carduri* (echivalentul cadrelor ecran) conținând text, imagini și grafice, legate între ele prin butoane hyperlink. În esență, LINKWAY era un mediu *WYSIWYG* (What-You-See-Is-What-You-Get) în care fiecare pagină a lecției putea fi editată vizual, iar autorul putea defini legături de navigare între pagini. Deși orientat către multimedia, LINKWAY menține filosofia „ecranului": fiecare ecran este proiectat individual și prezentat cursantului ca atare, iar structura navigabilă (secvențială sau ramificată) este controlată de autor prin stabilirea legăturilor dintre cadre.

LINKWAY a evidențiat potențialul calculatoarelor personale de a oferi suport multimedia (imagini color, eventual clipuri scurte) în instruire, însă fără a introduce noțiuni de *scenariu cauzal* sau inteligență artificială. Limitarea principală era că adaptarea la student se reducea la ramificări prestabilite, neexistând o logică decizională complexă bazată pe modele ale cunoștințelor cursantului.

Sisteme de autor bazate pe limbaje de tip script.

În contrast cu abordarea pe cadre (unde autorul lucrează la nivel de ecran/pagină), o altă categorie de sisteme de autor sunt cele bazate pe limbaje de tip script de programare a lecției. Acestea funcționează similar unui limbaj de programare: autorul scrie o secvență de instrucțiuni textuale (cod) care descriu pas cu pas afișarea conținutului și logica interacțiunii. Rezultatul final pentru cursant poate fi tot o succesiune de ecrane, însă procesul de creare este mai apropiat de programare decât de design vizual.

Reprezentativ în această categorie este sistemul PHILVAS [50], dezvoltat de compania Philips pentru a fi utilizat împreună cu playerul de videodisc Philips. PHILVAS (Philips Interactive Laser Vision Authoring System) este un exemplu tipic de sistem *line-based*: autorul elaborează un *script* textual compus din două tipuri de fragmente – fragmente de program (comenzi) și fragmente de teletexte (conținut de afișat pe ecran). Fragmentele de program sunt alcătuite din instrucțiunile limbajului ALLVA (*Author Language for Laser Vision Applications*). Programul scris în PHILVAS controlează secvențial atât afișarea textului/graficii pe ecranul. De exemplu, PHILVAS oferea comenzi precum „afișează textul X pe ecran", „redă secvența video de la adresa Y" sau „citește introducerea de la tastatură". Autorul trebuia să scrie într-un fișier script toate aceste acțiuni în ordinea dorită, incluzând și ramificările (e.g., *if-then* în cod pentru a reacționa la răspunsurile utilizatorului).

Un avantaj al PHILVAS era posibilitatea integrării strânse cu hardware-ul videodisc – oferind lecții interactive video. De asemenea, PHILVAS includea un *Simulator* pentru testarea lecției (permițând autorului să ruleze scriptul chiar și fără a avea videodiscul fizic atașat, simulând comportamentul acestuia) și un modul de *Compresor* care permitea stocarea scriptului final pe disc, cartuș. Cu toate acestea, PHILVAS prezenta și limite semnificative: era un sistem relativ complex, ce necesita programare în cod proprietar, fiind totodată legat strict de echipamentele Philips. PHILVAS nu utilizează nici cadre declarative, nici scenarii cauzale – lecțiile erau secvențe deterministe de comenzi, a căror adaptare se limita la instrucțiuni condiționale codificate de autor.

Un sistem similar ca abordare a fost VISA (Videodisc Interactive System for Authoring) [50]. VISA funcționa tot pe bază de scripturi liniare: autorii descriau prin cod derularea lecției combinând afișarea textelor pe ecran cu comenzi de control al videodiscului (căutare, pauză, reluare etc.). Deși detaliile implementării diferă de PHILVAS, conceptul-cheie rămâne același: cursul este un program scris, nu un ansamblu de ecrane editate vizual.

Avantajul este finețea controlului – autorul putea specifica cu precizie comportamentul la nivel de cadru video sau caracter text – însă dezavantajul era accesibilitatea redusă: doar programatori sau autori cu pregătire tehnică puteau utiliza eficient astfel de instrumente. VISA permite crearea exercițiilor care sunt compuse dintr-un șir de module numerotate. Fiecare modul este independent și nu depinde de altele. Modulele pot fi de următoarele tipuri: întrebare cu alegerea răspunsului corect dintre câteva răspunsuri propuse, întrebare cu răspunsul liber, completarea textului care conține spații

Ca și PHILVAS, sistemul VISA nu oferă facilități mari: totul este explicit programat pas cu pas, ceea ce înseamnă că flexibilitatea și adaptivitatea depind exclusiv de ceea ce autorul a anticipat în cod.

Un exemplu timpuriu de sistem de autor bazat pe limbaj specializat a fost cel al companiei Texas Instruments, cunoscut sub numele de TOPIC [50]. TOPIC a fost conceput ca un *authoring language* pentru crearea de cursuri IAC într-un mod structurat. În esență, TOPIC punea la dispoziție un set de instrucțiuni și construcții asemănătoare unui limbaj de programare educațional: autorul putea specifica ecrane de material, întrebări și bifurcații prin cod text (de exemplu, definind itemi de tip întrebare cu variante de răspuns și acțiuni asociate fiecărei opțiuni). Ca orice limbaj general, însă, TOPIC nu valorifica optim capabilitățile specifice ale niciunui sistem.

TOPIC exemplifică dilema sistemelor lineare: ele pot fi foarte flexibile ca putere de expresie, dar cer mai mult efort de dezvoltare și cunoștințe de programare, ceea ce le îndepărtează de utilizatorii cu profil pur pedagogic.

De menționat și SEDF/2 [50], un sistem de autor dezvoltat sub egida IBM. SEDF/2 (denumire ce sugerează o a doua versiune a unui mediu de dezvoltare educațională *System for Educational*

Development Facility/2) funcționa pe platforme IBM de tip mainframe sau mini (inclusiv pe sistemul AS/400 conform documentației). SEDF/2 permitea scrierea de *cursuri online* care apoi puteau fi afișate pe terminale legate la sistemul central (prin intermediul unui program de prezentare educațională). Practic, un autor scria un set de scripturi sau folosea un limbaj semiformal pentru a defini lecția – de la structura modulului de curs, până la itemii de evaluare. Fiind un produs IBM, SEDF/2 punea accent pe integrarea cursurilor în mediul corporatist: lecțiile create puteau fi gestionate centralizat, apelate de cursanți conectați la rețea, și se integrau cu bazele de date de training existente. Totuși, din perspectiva designului instrucțional, SEDF/2 nu aducea inovații conceptuale majore: era orientat tot pe *secvențe de ecrane*, fără a introduce paradigme declarative noi. Limbajul sau editorul SEDF/2 impunea autorilor să definească logicile de ramificare clasic (prin condiții programate). Cu alte cuvinte, deși era un progres în ceea ce privește distribuirea cursurilor și standardizarea livrării, SEDF/2 rămânea tributar modelului secvențial/ramificat fără cunoștință de cauzalitate pedagogică.

Sisteme de autor cu interfață de tip meniu.

O evoluție importantă în designul sistemelor de autor a fost tranziția de la programarea explicită (cum am văzut la PHILVAS sau TOPIC) la utilizarea de meniuri și interfețe interactive pentru construirea lecțiilor. Aceste sisteme, numite uneori *shell-uri de autor,* au scopul de a ascunde complexitatea codului în spatele unor opțiuni de configurare ușor de folosit de către autori, astfel încât și persoanele fără experiență de programare să poată crea cursuri IAC.

Un exemplu timpuriu menționat anterior, STAF [50], poate fi văzut ca un precursor al acestei categorii: în loc ca autorul să scrie cod, STAF îi permitea să aleagă din meniuri tipurile de itemi de inclus (de exemplu, „întrebare de tip grilă", „explicație teoretică", „diagramă"), să completeze conținutul și să stabilească navigarea alegând opțiuni predefinite. Astfel de sisteme cu meniuri acționează ca niște *șabloane interactive*: ele ghidează autorul pas cu pas în construcția lecției, întrebându-l ce vrea să facă și generând în fundal structura corespunzătoare. Avantajul evident este utilizabilitate sporită – experții în pedagogie pot implementa un curs fără să învețe un limbaj specializat, ceea ce reduce timpul și costul de dezvoltare.

Un alt exemplu este sistemul RACURS [58] (*rus*. РАКУРС), elaborat la Institutul Inginerilor Aviației Civile din Moscova, este un mediu de autor unde accentul este pus pe prezentarea grafică și navigarea prin meniuri. RACURS asigura un editor interactiv în care autorul putea selecta dintr-un meniu acțiunile de învățare și reacțiile la răspunsuri, fără a coda manual, îl putem privi ca pe un reprezentant al tendinței de a face authoring-ul mai accesibil, prin interfețe om-computer de nivel înalt.

Funcțiile de bază ale sistemului sunt: organizarea bibliotecii programelor cursurilor corespunzătoare disciplinelor de studii, înregistrarea cursului în bibliotecă, elaborarea și corectarea cursului, interpretarea cursului în regim de instruire și control. Cursul se formează automat în baza meniurilor și interviurilor. Elementul de bază al cursului este sistemul de cadre de diferite tipuri pentru completarea cărora există un set de formulare speciale. Cadrele sunt de două tipuri: informaționale și de control. Limbajul este inaccesibil pentru utilizator. Sistemul, de asemenea, oferă posibilitățile de instruire și control al cunoștințelor în grup, gestionarea bazei de date ce conține cursurile elaborate în regim de dialog, culegerea statisticii despre instruire și control.

Sistemul ADONIS [44] Sistemul Adaptiv Informațional De Dialog elaborat la Asociația Științifică de Producere de la Moscova este destinat pentru asigurarea informațională a procesului de studii și organizare a instruirii automatizate și controlului cunoștințelor la diferite discipline. Alcătuirea cursului la fel ca și în sistemul RACURS se bazează pe cadre și formulare speciale.

Tot în această categorie putem include și sistemul de autor MIDOS [50] (МИДОС – Мобильная Инструментальная Диалоговая Обучающая Система) elaborat la Institutul de Energetică din Moscova este destinat pentru automatizarea alcătuirii, depanării și corectării cursurilor computerizate. MIDOS oferă o interfață de meniuri sau formulare pentru introducerea conținutului și a regulilor cursului, rulând pe microcalculatoare și simplifica munca autorilor permițându-le să se concentreze pe conținut, nu pe cod.

Sistemul asigură trei regimuri de elaborare a cursului: introducerea și editarea structurii cursului de instruire, introducerea și editarea textelor cursului, depanarea în regim de imitare a instruirii. Astfel programarea structurii și textelor este divizată. Similar altor sisteme altor sisteme de autor, MIDOS cere descrierea precisă a algoritmului cursului. MIDOS oferea un set fix de structuri de lecție (ramuri de conținut, itemi de evaluare) pe care autorul le putea combina prin alegeri din meniu.

Un alt exemplu de mediu bazat pe meniuri și interfață grafică este Education ONE [50]. Education ONE este un sistem integrat de autor, promovat ca o soluție „totul-în-unul” pentru crearea, gestionarea și livrarea cursurilor electronice. Platforma oferă autorilor un mediu interactiv în care puteau concepe cursul modul cu modul, folosind ferestre de dialog și meniuri pentru a introduce texte, imagini și întrebări, precum și pentru a seta căile de navigare între module. Acest sistem pune accentul pe *ușurința utilizării*: un designer instrucțional putea crea un curs fără a programa, folosind doar opțiunile puse la dispoziție de interfață. Limitările unei asemenea abordări se regăsesc în rigiditatea formatelor – autorul este constrâns de tiparele de lecție suportate de soft. Astfel, flexibilitatea este sacrificată parțial pentru simplitate: lecțiile complexe sau neobișnuite puteau fi greu de implementat dacă nu se încadrau în șabloanele Education ONE. Ca și celelalte, nici acest sistem nu utiliza *scenarii*

cauzale: logica instruirii era predefinită în ramificările selectate de autor, nu exista un motor de decizie adaptiv în funcție de situații emergente.

Un concept asemănător a fost explorat și de ETAS [56], un alt sistem clasic de autor. Numele ETAS este un acronim precum *Expert-Teach Authoring System*, ceea ce indică scopul: o platformă de creare a cursurilor de instruire. Sistemul oferea facilități semi-interactive de concepere a lecțiilor, combinând elemente de programare cu elemente de interfață. Unele sisteme de acest tip permit autorilor să alterneze între modul *vizual* (de exemplu, completarea unor formulare cu întrebări și răspunsuri) și modul *script* pentru utilizatorii avansați. ETAS e destinat să faciliteze procesul de authoring convențional, nu de introducerea unei inteligențe instrucționale.

Sisteme de autor multimedia și hipermedia

O categorie distinctă este categoria sistemelor de autor multimedia. Acestea reprezintă generație de sisteme de autor, extinzând capabilitățile celor anterioare prin integrarea mai multor tipuri de media (text, grafică avansată, audio și video) și adesea oferind interfețe grafice și metafore intuitive de design. Aceste sisteme au orientarea către prezentări bogate media și către design vizual (în detrimentul codării brute), însă la nivel de concept educațional ele nu se îndepărtează fundamental de modelul ramificat/clasic.

Unul dintre cele mai cunoscute astfel de sisteme este Authorware [47,48] (creat inițial de Authorware Inc., ulterior preluat de Macromedia). Authorware a introdus o paradigmă de dezvoltare pe bază de *icoane și flux grafic*: autorul nu mai scrie cod și nici nu editează pagini individuale ca entități fixe, ci construiește un flux logic al lecției folosind pictograme care reprezintă acțiuni (prezentare de conținut, întrebare, ramificare decizională, salt etc.).

Practic, Authorware a oferit o reprezentare vizuală a programului educațional sub forma unei diagrame de flux, ceea ce face logica lecției mult mai ușor de urmărit și de modificat. Fiecărei icoane i se pot asocia media bogate: de exemplu, o icoană de tip „Display” poate conține un ecran ce combină text, imagini și butoane, o icoană „Movie” poate reda un clip video, iar o icoană „Interaction” poate capta răspunsuri și decide ramificarea. Acest mod de lucru a redus considerabil bariera tehnică – autorii se puteau concentra pe ordinea pedagogică a evenimentelor, așezând icoanele în succesiunea dorită, în timp ce codul necesar era generat de sistem în spate.

Cu toate acestea, din punct de vedere conceptual, modelul rămâne *imperativ*: lecția curge de la o etapă la alta conform diagramei create de autor. Dacă se dorește adaptare, autorul trebuie să deseneze explicit ramificații diferite sau să folosească variabile și condiții (posibil tot prin elemente

vizuale). Authorware execută pur și simplu fluxul de instrucțiuni reprezentat de icoane. Toate scenariile posibile trebuie gândite și reprezentate de autor în diagrama de flux.

Un alt sistem în zona multimedia este Asymetrix [45] care cuprinde un set de subsisteme, cum ar fi: IconAuthor, Librarian, ToolBook Instructor, ToolBook Assistant.

IconAuthor permite crearea cursurilor asistate de calculator în regim de dialog în baza schemei bloc alcătuite de autor, de asemenea este posibilă crearea cursurilor cu ajutorul meniului. Cursurile create pot fi utilizate în variantă locală și de rețea. Librarian are ca funcție monitorizarea și gestionarea setului de cursuri de instruire. ToolBook Assistant este destinat pentru utilizatori care doresc să creeze cursuri asistate de calculator fără programare și proiectare complicată. Librarian are ca funcție monitorizarea și gestionarea setului de cursuri de instruire.

Asymetrix ToolBook Assistant este un sistem de autor bazat pe metafora unei cărți electronice: lecția este concepută ca o carte cu pagini pe ecran, iar autorul poate plasa pe fiecare pagină elemente multimedia (texte formatate, imagini, butoane, controale interactive). ToolBook oferea un mediu de dezvoltare WYSIWYG în Windows, foarte apropiat de conceptul de hipermedia – autorul putea lega paginile prin hyperlink-uri sau butoane de navigare, definind astfel un traseu educațional flexibil, nu neapărat liniar.

O caracteristică tehnică notabilă este existența unui limbaj de scripting (OpenScript) integrat: pentru comportamente standard (de exemplu, buton „următoarea pagină”) nu era necesară programare, dar pentru interactivitate complexă autorul putea adăuga scripturi atașate obiectelor. Astfel, Asymetrix ToolBook combina ușurința designului vizual cu posibilitatea extinderii funcționalității prin cod – ceea ce îl făcea atractiv atât pentru novici (care puteau crea rapid o lecție simplă în stil prezentare), cât și pentru experți. Ca limitare conceptuală, la fel ca la Authorware, logica pedagogică era implicit una de *browsing* controlat: autorul stabilea ce pagini există și cum se leagă, eventual ce se întâmplă la anumite acțiuni, dar sistemul nu „știa” nimic despre substanța conținutului. De exemplu, ToolBook nu își construia o reprezentare a cunoștințelor predate sau a performanței elevului, în absența unor scripturi special scrise de autor. Prin urmare, adaptarea lecției la elev trebuia programată prin ramuri/hyperlinkuri, în lipsa unui motor de inferență educațional.

Printre sistemele multimedia populare se numără și TenCORE [56]. TenCORE (Training Environment for Computer Output Runtime Education) este un pachet software folosit mult în industrie și armată pentru creare de cursuri interactive. TenCORE oferea și el un mediu vizual de dezvoltare, incluzând un editor de ecrane cu text/grafică și un limbaj de scripting pentru logica de curs. Practic, TenCORE se poziționează ca o platformă de “courseware development” cuprinzătoare: autorii pot crea rapid module de instruire, adăugând elemente media, apoi pot programa finețea

interacțiunii (de exemplu, reguli de evaluare a răspunsurilor, calcule ale scorurilor, generare de rapoarte).

Avantajul TenCORE este flexibilitatea tehnică, animații simple, integrare audio etc., permițând realizarea de conținuturi mai captivante. Ca filosofie didactică, TenCORE urmează modelul tutorial ramificat: nu implementa vreo paradigmă nouă de predare, ci îi oferă autorului toate uneltele necesare pentru a transpune metodele clasice într-o formă interactivă și multimedia. Deși autorul poate crea scenarii complexe, acestea sunt tot explicit programate, neexistând o separare între *cunoașterea domeniului* și *strategia de predare*. Cu alte cuvinte, TenCORE executa scenariile exact așa cum erau ele scrise, neputând devia sau genera altceva în afara a ceea ce a prevăzut realizatorul cursului.

Unele sisteme precum Education ONE [50], ADONIS [44] sau altele similare, cu timpul au evoluat spre platforme de e-learning. prin includerea și funcții de administrare a instruirii (management educațional).

1.3. Sarcinile principale ale tezei

Comparând sistemele de autor IAC prezentate în compartimentul precedent, se pot formula o serie de observații privind limitele abordărilor clasice. Ne vom opri la câteva limitări fundamentale din punctul de vedere al ingineriei instruirii:

- *Dependența de secvențe predefinite și lipsa scenariilor cauzale reale:* Poate cea mai importantă constatare este absența totală a oricărei forme de *scenariu cauzal* în sensul inteligenței artificiale. Niciunul dintre sistemele menționate (cu excepția unor încercări marginale de adaptare, precum cele din ADONIS) nu este capabil să *genereze* sau să *aleagă dinamic* acțiuni educaționale pe baza unei înțelegeri de ansamblu a situației curente a cursantului. În schimb, toate urmează scenarii deterministe stabilite apriori: autorul definește din start toate ramificațiile posibile ale lecției, iar sistemul nu face altceva decât să direcționeze studentul pe una dintre aceste ramuri în funcție de răspunsurile sale, dacă un student dă răspunsuri greșite, sistemul nu *deduce* de la sine cauza (o lacună conceptuală anume) și nu schimbă strategia decât dacă autorul a prevăzut explicit o ramificație remediară. Cu alte cuvinte, reacțiile sistemului la acțiunile cursantului nu sunt decât execuții ale unor instrucțiuni condiționale.

- *Absența limbajelor declarative bazate pe frame-uri de cunoaștere***:** În majoritatea sistemelor de autor clasice, *frame*-ul este fie un termen pentru ecran/pagină, fie nu este folosit deloc. Nu întâlnim frame-uri ca structuri de reprezentare a cunoștințelor (în sensul inteligenței artificiale, conform lui Minsky [63]) în aceste sisteme. Limbajele pe care ele le pun la dispoziție sunt imperative (procedurale) sau cel mult descriptive la

nivel de interfață, dar nu declarative în ceea ce privește conținutul pedagogic. Chiar și atunci când unele sisteme au introdus *baze de date* sau *biblioteci* (de exemplu, anumite versiuni avansate puteau avea un modul de *item banking* sau de obiecte de conținut reutilizabile), acestea serveau mai mult organizării și reutilizării materialelor, nu reprezentării cunoașterii într-un mod inteligent. Absența unui limbaj declarativ orientat pe conținut face dificilă adaptarea sau reorganizarea automată a materialului.

- *Rigiditate și efort mare de actualizare:* Pentru că logica instruirii era construită manual, orice modificare a conținutului sau structurii cerea adesea efort considerabil. De exemplu, dacă într-un curs realizat cu Microtext autorul dorea să insereze un nou subiect sau să schimbe ordinea unităților, trebuia să reajusteze manual legăturile dintre cadre și eventual condițiile de ramificare. În lipsa unei separări clare între conținut și strategie pedagogică, actualizarea cursurilor era predispusă la erori (de exemplu, uitarea ajustării unei trimiteri, ducând la *ramuri moarte* sau inconsistente). Multe sisteme nu ofereau un mod ușor de a vizualiza global structura unui curs complex: autorul lucra fie ecran cu ecran, fie scria cod secvențial, ceea ce pentru proiecte mari ducea la pierderea clarității asupra ansamblului. Cu atât mai mult, reutilizarea conținutului între cursuri era dificilă – nu existau standarde de obiecte de învățare reutilizabile sau separări modulare pronunțate (exceptând poate posibilitatea de a copia secvențe de cod sau de cadre dintr-un curs în altul).

- *Dependențe tehnologice și interoperabilitate redusă***:** Multe sisteme clasice erau strâns legate de platforma hardware sau software pentru care au fost create. Acest lucru însemna că un curs creat într-un sistem nu putea fi portat sau distribuit ușor în alt mediu fără o reimplementare considerabilă. Chiar și în cazul sistemelor de PC, formatele proprietare (de ex. formatul .TBK al ToolBook sau .A7P al Authorware) necesitau run-time-uri dedicate pentru livrarea cursurilor, deci fiecare sistem era un ecosistem în sine.

Toate acestea provoacă motive temeinice ce conduc la necesitatea de cercetare și elaborare a unor limbaje originale, bazate pe structura frame, care ar oferi autorilor cursurilor asistate de calculator posibilități didactice evoluate. În legătură cu această exigență scopul principal al tezei este elaborarea metodelor și mijloacelor evoluate de creare a cursurilor de instruire asistate de calculator. Atingerea acestui scop necesită soluționarea unui șir de probleme cercetate în cadrul tezei. Aceste probleme sunt următoarele:

- cercetarea formelor și metodelor de desfășurare a lecțiilor cu ajutorul sistemelor de instruire asistată de calculator; elaborarea tehnologiei de proiectare
- elaborarea modelului structural al procesului de studii;

- formularea cerințelor principale față de sistemele de instruire asistate de calculator;
- formularea cerințelor față de limbajul de descriere a cursurilor asistate;
- dezvoltarea unui limbaj de descriere a cursurilor asistate de calculator evoluat bazat pe un sistem de frame-uri (în sensul Inteligenței Artificiale conform lui Minsky[63]) și a mecanismelor originale de interpretare a acestui limbaj de descriere;
- testarea experimentală a limbajului de descriere a cursurilor asistate de calculator elaborat pe un exemplu de curs de instruire asistat de calculator.

Vom enumera avantajele eventuale și problemele posibile la realizarea sarcinilor tezei.

Sistemul propus în teză se bazează pe *modelarea situațiilor instructive prin scenarii cauzale și structuri de tip frame*, ceea ce reprezintă o abordare radical diferită de cele tradiționale analizate mai sus. În sistemul modern, un *scenariu cauzal* înseamnă că lecția nu mai este o secvență fixă de pași, ci un *ansamblu de situații posibile* interconectate prin relații cauză-efect. Astfel, fluxul instruirii poate *emerge dinamic* în funcție de acțiunile elevului și de starea sistemului, și nu este complet fixat anticipat. De exemplu, dacă elevul manifestă o neînțelegere a unui concept, acest fapt (cauză) declanșează în mod dinamic un scenariu de remediere (efect) adaptat exact conceptului respectiv. Această cauzalitate instrucțională este *explicit reprezentată* în sistemul teză, nu doar codificată implicit. În sistemul bazat pe scenarii cauzale, cunoașterea acestor relații face parte din logica platformei: sistemul are *reguli pedagogice* de tipul „dacă se întâmplă situația A, atunci reacționează cu intervenția B", unde A și B sunt definite în termeni pedagogici (nu doar ca pași de program).

De asemenea, folosirea *structurilor de tip frame* în sens declarativ aduce un avantaj major. În acest context modern, un *frame* este o structură de date ce descrie un concept, o situație sau o unitate de învățare, cu atribute și relații. De exemplu, poate exista un frame pentru un concept-cheie ce conține definirea conceptului, , exemple, greșeli tipice asociate etc. Sistemul de autor teză, bazat pe frame-uri declarative, permite autorului să introducă cunoștințele despre domeniu și despre procesul didactic într-o formă *structurată semantic*. Practic, autorul declară *ce* vrea să predea (și contextul acestor cunoștințe), iar sistemul (folosind motorul bazat pe scenarii cauzale) poate decide *cum și când* să le predea, reacționând la acțiunile elevului. Acesta este un salt calitativ față de sistemele clasice, unde autorul trebuia să se ocupe și de *ce* și de *cum* într-un mod amestecat în scenariul static.

Vom evidenția câteva aspecte cheie:

- *Reprezentarea cunoștințelor și a planului de lecție***:** În sistemul propus, cunoștințele sunt modelate (de exemplu, sub formă de frame-uri cu legături între ele). Strategia de predare poate fi în parte generativă – sistemul poate alege drumul prin materie bazat pe progresul elevului. În sistemele clasice, cunoștințele nu sunt reprezentate

decât ca pagini de text izolate, iar planul lecției este practic *definit regid* de autor. Astfel, sistemul propus permite *reutilizarea cunoașterii* – de exemplu, un frame de concept poate fi folosit în mai multe scenarii, sau un scenariu cauzal generic (precum "dacă elevul uită o definiție, recapitulează definiția") se poate aplica automat la mai multe concepte.

- *Adaptivitate și personalizare:* Sistemul pe scenarii cauzale este, prin natura sa, *adaptive*, întrucât deciziile de livrare se iau în timpul rulării pe baza stării curente. Două parcursuri educaționale pot diverge semnificativ dacă stările (modelele cursantului) diferă, chiar dacă la start lecția e aceeași. Sistemul propus poate susține *trasee complet diferite* pentru cursanți diferiți, explorând mai mult acolo unde e nevoie și trecând peste părțile deja stăpânite.
- *Calitatea învățării:* Deși aici intrăm pe tărâm empiric, ne putem aștepta ca un curs dezvoltat cu scenarii cauzale, deci *inteligent adaptiv*, să ofere o calitate sporită a învățării comparativ cu unul static. Cursantul beneficiază de *feedback personalizat* și de un traseu calibrat pe nevoile sale. Aceasta poate preveni plictiseala pentru cei avansați sistemul sare peste redundanțe.

Desigur, trebuie menționat că introducerea acestor facilități avansate vine și cu o *complexitate sporită a sistemului de autor*. Conceptual, pentru ca un sistem să folosească frame-uri de cunoaștere și scenarii cauzale, el trebuie să încorporeze un oarecare *motor de inferență* sau cel puțin de reglare pe baza stării cursantului. Asta înseamnă dezvoltarea sistemului trebuie să integreze principii de inteligență artificială, de reprezentare a cunoașterii.

2. MODELUL STRUCTURAL AL PROCESULUI DE INSTRUIRE PENTRU CURSURILE DE INSTRUIRE ASISTATE DE CALCULATOR

2.1. Analiza procesului de studii

Procesul de instruire este considerat ca un proces tehnologic divizat în etape și construit după principiul sistemului de dirijare închis. În rezultatul analizei procesului de studii se pot evidenția următoarele etape:

- *etapa acumulării informației* este menită să sistematizeze cunoștințele studentului, să evidențieze lacunele în cunoașterea teoriei și să le lichideze prin consultarea manualelor, diferitelor surse de informații, să desfășoare controlul cunoașterii teoriei;
- *etapa formării înțelegerii* ajută ca studentul corect să interpreteze informațiile acumulate la disciplina dată; să clarifice esența noțiunilor, algoritmilor, în special prin îndeplinirea exercițiilor; să definească locul cunoștințelor obținute în contextul disciplinei studiate;
- *etapa formării îndemânărilor soluționării problemelor* (*practicum*) este destinată pentru formarea deprinderilor de soluționare a problemelor tipice din domeniul studiat;

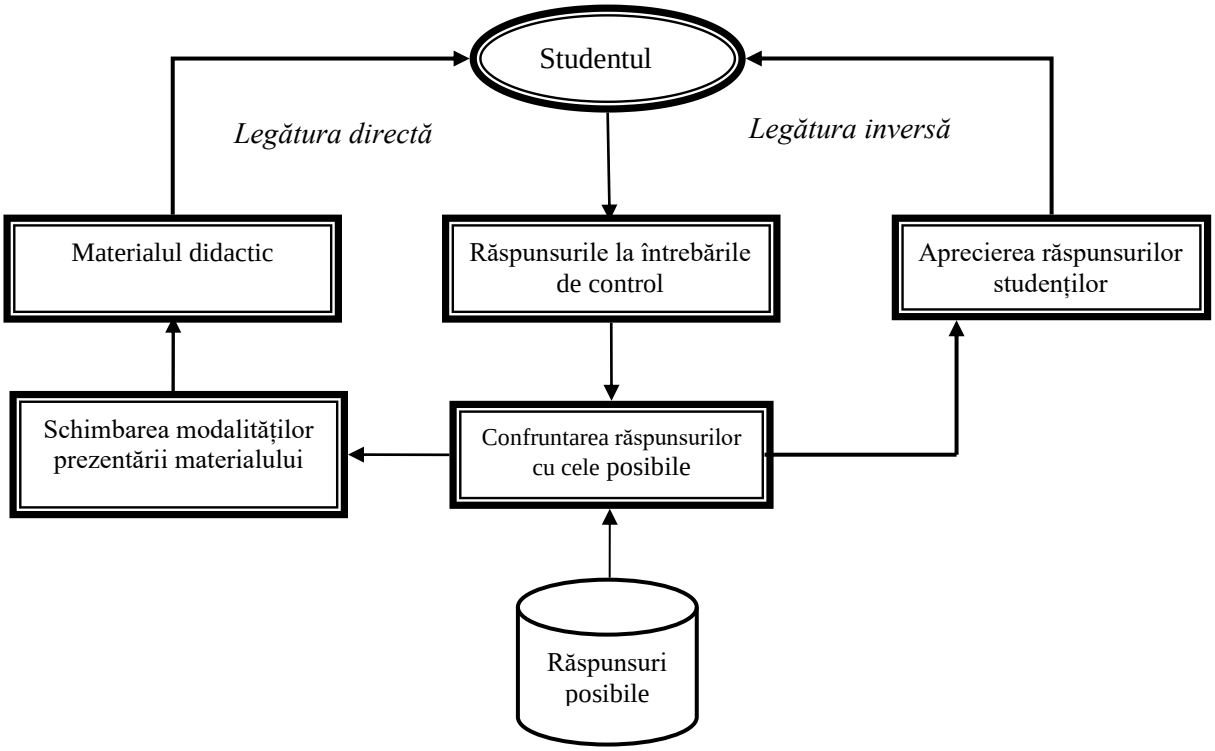


Figura 2.1. Procesul instruirii computerizate ca sistem de dirijare închis.

- etapa formării abilităților este menită pentru instruirea aplicării cunoștințelor din domeniul studiat;

- etapa verificării cunoștințelor are ca scop desfășurarea controlului sumativ a cunoștințelor studentului obținute în rezultatul studierii disciplinei, depistarea lacunelor în cunoștințe, analizei greșelilor făcute în răspunsurile la întrebările de control, efectuarea precizărilor pentru lucrul ulterior cu materialul didactic.

Într-un mod mai general, instruirea asistată de calculator este definită ca formare dirijată pe operații a cunoștințelor, deprinderilor și îndemânărilor [54]. Baza procesului de instruire computerizată constituie programul de instruire sau cursul asistat de calculator. Dirijarea într-un astfel de program reprezintă un proces complicat care, în linii generale, este reprezentat în Figura 2.1.

Programul de instruire este o totalitate de proceduri de instruire, efectuată pe pași, structural compusă din informații didactice, expuse într-un anumit sistem, ce formează cunoștințe speciale care determină îndeplinirea de către studenți a anumitor acțiuni necesare pentru însușirea obiectului studiat și conține indicații pentru îndeplinirea corectă a acestor sarcini (*legătura inversă*).

Programul de instruire constă din pași. Pasul programului de instruire este o totalitate de informații pentru legătura directă și inversă și reguli de treceri la următoarea activitate de cunoaștere. Pasul programului de instruire este prezentat schematic în Figura 2.2.

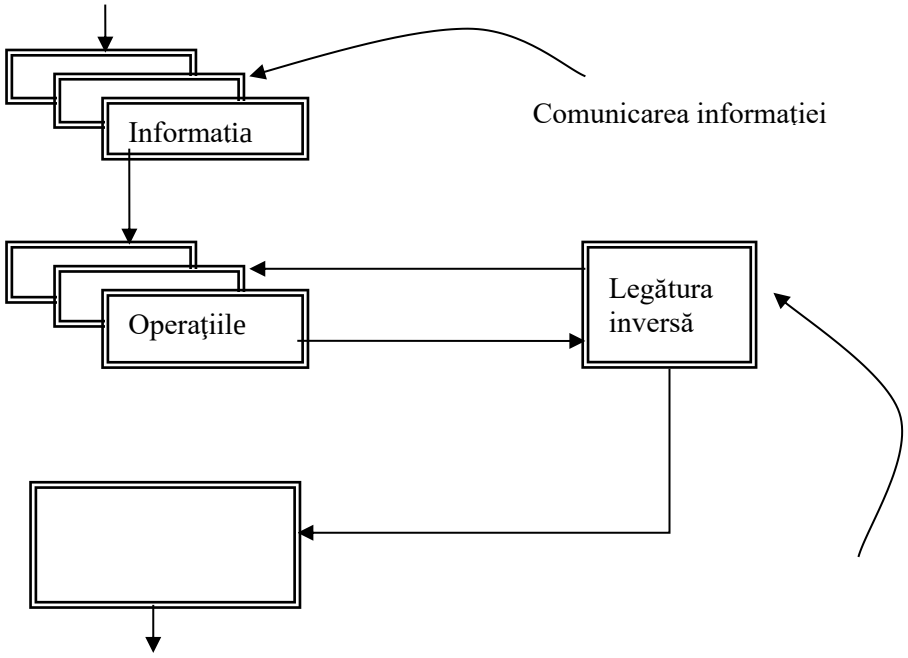


Figura 2.2 Detalierea pasului programului de instruire.

Prin legătura directă materialul didactic este transmis studentului. Legătura inversă se efectuează prin răspunsurile studenților. După studierea porțiunii de informații studentul rezolvă

problemele propuse pentru verificare a cunoștințelor căpătate și răspunde la întrebările de control. Sistemul devine închis: semnalul de la sfârșitul lanțului de dirijare revine la începutul lui. În dependență de răspunsurile studentului se face selectarea și pregătirea următoarei porțiuni de material pentru studiere. Prin aceasta se manifestă importanța legăturii inverse. Eficacitatea instruirii în astfel de sistem depinde de tipul programului de instruire.

Existența legăturilor inverse în instruire permite:

- lichidarea lacunelor în cunoștințe;
- aprofundarea înțelegerii teoriei;
- căpătarea și perfecțiunea deprinderilor și îndemânărilor în rezolvarea problemelor aplicative;
- obținerea ideilor clare despre legăturile materialului studiat cu celelalte compartimente.

Pentru construirea cursurilor asistate de calculator de calitate, eficacitatea cărora depinde nu numai de măiestria pedagogică a alcătuitorului cursului, este necesar de elaborat detaliat modele de instruire [51-54]. În calitate de model conceptual de instruire în prezent se folosește pe larg interpretarea didactică a schemei de dirijare cu obiectul variabil.

Sub *noțiunea de dirijare* se va înțelege o acțiune la obiect direcționată spre atingerea unei anumite stări a lui sau a comportamentului obiectului.

Pentru detailarea schemei de dirijare este necesară o analiză amănunțită și, în final, construirea modelului activității profesorului și studentului pentru adaptarea ulterioară a modelului obținut la tehnologiile de creare a cursurilor de instruire [54-56]. Activitățile profesorului integrează conceptele: lecția, dialogul, consultația, ajutorul etc. Activitățile studentului [58 - 60] supuse modelării instruirii includ conceptele: perceperea informației, îndeplinirea exercițiilor și rezolvarea problemelor, cererea de ajutor, informații și altele.

Realizarea legăturii inverse corespunzătoare procesului de instruire presupune rezolvarea următoarelor două probleme.

Prima este problema de determinare a conținutului legăturii inverse care presupune evidențierea totalității caracteristicilor controlate în baza scopurilor instruirii, cu alte cuvinte, astfel de caracteristici care duc la trecerea dintr-o stare calitativă la o altă stare.

A doua este problema de determinare a periodicității și tipului legăturii inverse. Pe de o parte, datele obținute cu ajutorul legăturii inverse permit introducerea corectărilor necesare în procesul de instruire, iar, pe de altă parte, îndeplinesc funcția de consolidare și de legătură cu sfera de motivare a studentului.

Sunt cunoscute trei scheme principale de dirijare: deschisă, cu legătura inversă închisă și combinată. Avantajul sistemelor de dirijare cu legătură inversă este faptul că scopurile în aceste sisteme sunt atinse în lipsa informației complete despre acțiunea mediului asupra obiectului de dirijare.

Problemele principalele la construirea sistemelor de instruire ca sisteme de dirijare sunt acele probleme care determină scopurilor și criteriilor de dirijare.

În calitate de scop se ia instruirea activității direcționate spre atingerea anumitor obiective într-un domeniu obiectual localizat. Activitatea constă din procesele atingerii obiectivelor $D=\{Pr\}$.

Procesul efectiv este considerat ca o succesiune de acțiuni (d) care conduce spre obiectivul:

$$Pr = d_1, d_2, \ldots, d_k; k = 1,n.$$

În domeniul obiectual pentru student se evidențiază (se localizează) un ansamblu de obiective pentru care el trebuie să construiască procesele efective $S = \{s\}$. Activitatea studentului în domeniul obiectual constă în atingerea obiectivelor din acest ansamblu și, prin urmare, $D_s = \{Pr_s / s \in S\}$. În calitate de criterii de dirijare se poate de evidențiat următoarele: scopurile sistematizate ierarhic, activitatea îndreptată spre un anumit scop și localizarea scopurilor.

În scopurile reprezentării adecvate a modelului structural a procesului de instruire a fost analizat procesul de instruire și au fost evidențiate situații de instruire stereotipice care apar la predarea diferitelor discipline. Printre situații evidențiate ca bază la proiectarea cursurilor asistate de calculator au fost luate următoarele situații: expunerea (explicația) materialului didactic de către profesor, desfășurarea testelor, a lucrărilor de control, îndeplinirea exercițiilor, rezolvarea problemelor, cererile de ajutor ale studenților.

Vom examina mai detaliat componentele modelului structural al procesului de studii. Vom nota cu litere majuscule situațiile instructive stereotipice.

Prin ***P*** vom nota transmiterea studentului unei anumite porțiuni a informației didactice, ce reflectă situația prelegerii, a lucrului cu manualul. În calitate de informație poate să apară texte, diagrame, imagini grafice, tabele, hârți geografice, sunete, fragmente video, etc.;

T – transmiterea informației care anticipă întrebarea și reflectă situația explicațiilor de către profesor, a cerințelor pentru răspuns la întrebare. În calitate de date în aceasta situație poate servi informația analogică cu cea ce se conține în situația ***P***;

Q – propunere pentru student ca să răspunde la o întrebare. În acest caz se reflectă situația testării, a îndeplinirii exercițiului, a rezolvării unei probleme, a lucrării de control etc. În calitate de date în această situație se utilizează textul întrebării, a exercițiului, a problemei, etc. Situația ***Q***

numaidecât trebuie să aibă în componența sa șabloanele răspunsurilor la întrebările puse studentului, după care va fi efectuată analiza corectitudinii răspunsurilor la sarcinile corespunzătoare;

E – analiza corectitudinii răspunsului după șabloanele date. Se reflectă situația verificării lucrării de control, obținerea rezultatelor testării, etc. În calitate de date în acest caz sunt definite șabloanele răspunsurilor;

R – replica. Se reflectă situația reacției profesorului la răspunsul studentului. De regulă, în această situație se culege statistica despre învățarea cursului de către student. Statistica include date despre cantitatea și calitatea răspunsurilor corecte și incorecte, timpul învățării cursului, numărul apelării la ajutor;

S – oferirea încercării răspunsului repetat la întrebarea pusă, exercițiu, etc. Dacă în model această situație lipsește, în acest caz se presupune că întrebările vor fi puse numai o singură dată;

C – controlul depășirii numărului admisibil de încercări de răspunsuri la întrebări. Dacă în model situația ***C*** lipsește, în acest caz la depășirea numărului admisibil de încercări de răspunsuri la întrebări, studentului îi va fi afișat mesajul standard despre imposibilitatea de a răspunde mai departe la aceasta întrebare;

U – situația răspunsului nerecunoscut se utilizează în cazul în care răspunsul la întrebarea pusă nu poate fi recunoscut ca corect sau incorect. Această situație se utilizează pentru dirijarea cu acțiunile studentului la introducerea răspunsului după o formă prevăzută în prealabil, bună pentru analiză;

H – acest element al modelului este destinat acordării ajutorului în cazul în care studentului îi vine greu să răspundă la o întrebare, să rezolvă o problemă etc. În acest caz studentul poate să ceară ajutorul. În model poate fi utilizat un număr arbitrar de situații ***H***. Ordinea afișării ajutorului studentului este determinată de parametrul specificat în descrierea situației. În afară de aceasta, fiecare situație ***H*** este asociată cu un element anumit al modelului ***Q*** care conține întrebare sau problemă. Acest gen de asociații de asemenea trebuie de indicat cu specificator aparte.

Modelarea activității subiectelor procesului de instruire permite elaborare recomandărilor pentru alcătuirea cursurilor asistate de calculator, ale schemelor lor formalizate tipice. În afară de aceasta, pornind de la asemenea modele se poate de propus mijloace adecvate de proiectare automatizată și realizare a cursurilor asistate de calculator, de asemenea, și căile de automatizare a generării cursurilor.

2.2. Modelul structural al procesului de instruire

La crearea modelului structural al procesului de instruire sunt specificate următoarele scopuri:

- descrierea sistemului sau a unei componente al lui la un nivel conceptual ținând cont de posibilitatea realizării lui (nivelul interior) și descrierea domeniilor de aplicare a sistemului funcțional (nivelul exterior);
- utilizarea aparatului matematic pentru obținerea concluziilor relevante despre proprietățile sistemului;
- utilizarea modelului structural în calitate de bază pentru proiectarea sistemelor de instruire și crearea limbajului descrierii cursurilor de instruire asistată de calculator.

Pentru formarea modelului și realizarea lui sunt necesare următoarele premise: cunoașterea procesului soluționării problemei, cunoașterea structurii de bază a sistemului de instruire. Cunoașterea procesului soluționării problemei permite evidențierea părților principale ale modelului. Unele aspecte ale modelării matematice a procesului de instruire sunt examinate în lucrările lui R. Atkison [51].

Modelul materialului didactic

Elementul atomar al modelului structural al materialului didactic constituie o unitate de cunoștințe (noțiune, definiție, algoritm etc.). Cunoștințele reprezintă un set de mulțimi:

$$\boldsymbol{A} = \{\boldsymbol{A_1}, \boldsymbol{A_2}, \ldots, \boldsymbol{A_n}\},$$

$\boldsymbol{n}$ este numărul de tipuri ale elementelor atomare de cunoștințe. Fiecare element atomar $\boldsymbol{a_{ij}} \in \boldsymbol{A_j}$ poate fi în una dintre stările $\boldsymbol{c_i} \in \boldsymbol{C}$, $\boldsymbol{C} = \{0,1,2, \ldots\}$. $\boldsymbol{c_i}$ reflectă relația studentului cu elementul de cunoștințe corespunzător. În cel mai simplu caz, $\boldsymbol{C} = \{0,1\}$ (unde 0 codifică faptul că știe, 1 – nu știe). De asemenea, mulțimea $\boldsymbol{C}$ poate fi interpretată ca note. Așa dar, fragmentul materialului didactic într-un curs asistat de calculator poate fi prezentat ca o mulțime formată din unitățile de cunoștințe și starea $\boldsymbol{c_i}$ maxim posibilă la ieșirea din fragmentul:

$$\boldsymbol{F}_k = \{\boldsymbol{U_1}, \boldsymbol{U_2}, \ldots, \boldsymbol{U_t}, \ldots\}, \text{ unde } \boldsymbol{U_t} = \langle \boldsymbol{a_t}, \boldsymbol{c_t} \rangle, \ \boldsymbol{a_t} \in \boldsymbol{A}, \ \boldsymbol{c_t} \in \boldsymbol{C}; \ \boldsymbol{k = 1,n}.$$

Se poate de introdus caracteristica studentului instruit determinată de lista cunoștințelor obținute cu specificarea stărilor $\boldsymbol{c_i}$ minim admisibile care vor fi atinse după studierea cursului:

$$\boldsymbol{S} = \{\boldsymbol{U'_1}, \boldsymbol{U'_2}, \ldots, \boldsymbol{U'_v}, \ldots\} \ \boldsymbol{U'_v} = \langle \boldsymbol{a_v}, \boldsymbol{c'_v} \rangle; \ \boldsymbol{v = 1, m}.$$

Instruirea studentului poate fi considerată terminată, dacă pentru $(\forall \boldsymbol{a_p})$ $(\boldsymbol{c'_p a_p} <= \boldsymbol{c_p a_p})$ $\boldsymbol{c'_p} \neq 0$ ($\boldsymbol{c_p a_p}$ indică starea studentului). Obiectivul oricărui curs este majorarea lui $\boldsymbol{S}$. Așa dar, problema proiectării consecutivității studierii materialului se reduce la construirea lanțului $\langle \boldsymbol{F_1}, \ldots \boldsymbol{F_n} \rangle$, $\cup F_n \supseteq S$ (în lipsa criteriilor de optimizare).

Datele despre student

Colectarea datelor despre student joacă un rol important în determinarea materialului care se propune și în alegerea metodicii de instruire. Aceste date sunt achiziționate după următorii parametri:

- informația despre nivelul inițial de cunoștințe;
- datele statistice despre procesul de instruire;
- istoria detaliată a instruirii;
- setul de erori ale studentului;
- caracteristicile fiziologice și psihologice ale studentului;

Primii trei parametri sunt necesari pentru selectarea planului de instruire și pentru luarea deciziei despre nivelul de instruire. Acești parametrii pot fi utili, de asemenea, și la testarea cursului de către autor.

În unele cazuri este comod de avut două modele de student: modelul ideal (indicii doriți) și modelul real (indicii reali). Prin compararea acestor indici se poate de înțeles raționamentele greșite ale studentului.

Vom examina, de asemenea, și fluxurile de informații care trebuie percepute de către student [52-54].

Fluxul materialului didactic pe o unitate separată a obiectului studiat se definește prin succesiunea de puncte care corespunde momentelor primei prezentări a materialului și ale momentelor uitării lui. Fluxul însușirii se definește prin succesiunea de puncte care corespunde momentelor terminării însușirii sau studierii repetate a materialului didactic pe unitatea examinată a obiectului studiat.

Dacă o disciplină de studii conține N compartimente care trebuie studiate, atunci de compartimentul i pot fi legate două fluxuri: fluxul materialului de studii F_{1i} și fluxul de însușire F_{2i} ($i=1, N$). Fluxul F_{st} format prin suprapunerea fluxurilor F_{1i} ($i=1, N$) se numește fluxul materialului didactic al disciplinei de studii $F_{st}=\sum_{i=1}^{N} F_{1i}$. Analogic se determină și fluxul F_{rest}, care se formează prin suprapunerea fluxurilor F_{2i} ($i=1, N$), numit flux de însușire și restabilire a cunoștințelor

$$F_{rest}=\sum_{i=1}^{N} F_{2i} .$$

Un rol important în dirijarea cu fluxurile de informații în sistemele de instruire asistate de calculator (SIAC) îl are interfața utilizatorului. Mijloacele interfeței alese în mod corespunzător fac ca SIAC să devină accesibile pentru diferite categorii de utilizatori (de la începători până la cei experimentați) și înlătură bariera psihologică posibilă la interacțiunea cu SIAC. Caracteristicile

principale ale interfeței prietenoase [50-54] sunt simple și comode în comunicare, au un caracter logic, sunt flexibile, apropiere de mediul utilizatorului. Interfața SIAC trebuie să fie construită în așa mod, ca să se poată extinde posibilitățile adaptării la particularitățile individuale ale studentului, ca să se creeze confort psihologic în interacțiune cu SIAC.

Aprecierea cunoștințelor în cadrul cursurilor asistate de calculator

Pentru dirijarea reușită a procesului de instruire în cadrul cursurilor asistate de calculator este necesar să existe un sistem efectiv de apreciere a cunoștințelor studenților. Aplicarea unui astfel de sistem într-un curs de instruire permite nu numai de a-i pune note studentului, ci și de a-l informa în mod operativ despre calitatea lucrului lui în procesul de însușire a materiei. În afară de aceasta, astfel de informație poate fi utilizată pentru un sistem mai flexibil de acordare a ajutorului studentului la studierea materialului didactic și la îndeplinirea temelor și exercițiilor. În cursurile de instruire adaptivă informația despre calitatea lucrului studentului poate fi utilizată pentru alegerea modalităților de reprezentare a materialului didactic, precum și pentru alegerea temelor și întrebărilor de control. Putem spune cu certitudine că fără achiziționarea informației despre îndeplinirea exercițiilor de control, a răspunsurilor studenților la întrebări, la exerciții, dirijarea cu procesul de instruire este imposibilă. Culegerea datelor despre procesul de instruire efectuează conexiunea inversă între studentul și cursul de instruire.

Modelul aprecierii cunoștințelor se bazează pe metode statistice de prelucrare a răspunsurilor la întrebări, exerciții și probleme. Studentului i se propun ***n*** întrebări din ***N*** posibile. Numărul ***n*** de întrebări poate fi dependent de gradul corectitudinii răspunsurilor la întrebările precedente. Din numărul total de întrebări se definește numărul ***m*** de răspunsuri corecte. În dependență de partea răspunsurilor corecte ***b=m/k*** se va o decizie despre alegerea notei ***i, i = 1, ... , k***. În acest caz valoarea ***k*** determină numărul maxim de puncte după scara de note. Cu ***k* = 2** avem două calificative: „*admis*”, „*respins*”.

Astfel, cu abordarea statistică a problemei aprecierea calității cunoștințelor se reduce la stabilirea probabilității obținerii răspunsurilor corecte la ***n*** din ***N*** întrebări aliator alese. Dacă se acceptă ipoteza că studentul va putea să răspundă la ***M*** întrebări din ***N***, această probabilitate este

$$P(n,m) = \frac{C_M^m \bullet C_{N-M}^{n-m}}{C_N^n}$$

unde ***C*** numărul combinărilor ***M*** din ***m***.

Calitatea deprinderilor poate fi apreciată după exactitatea și viteza îndeplinirii operațiilor necesare. Dacă timpul observațiilor e destul de mare, atunci viteza îndeplinirii operațiilor ***c*** poate fi evaluată după formula:

$$c = n / \sum_{i=1}^{n} t_i ,$$

unde**:** t_i – timpul îndeplinirii operației la încercarea ***i; n*** – numărul încercărilor.

Cerințele față de cunoștințe se definesc în conformitate cu parametrii sistemului de apreciere.

De exemplu, în cazul sistemului de control cu zece note:

$0 < p <= p_{10}$ – nota „*zece*”,

$p_{10} < p <= p_9$ – nota „*nouă*”,

. . .

$p_1 < p <= 1$ – nota „*unu*”,

unde $p_{10}, p_9, p_8, \ldots, p_1$ – parametrii sistemului de control.

În cazul sistemului de control cu două note:

$p <= p_1$ – „*admis*”

$p >= p_2$ – „*respins*”

$p_1 < p < p_2$ – zona indiferenței.

Modelul bazat pe metode statistice prevede posibilitatea achiziționării informației despre răspunsurile studentului. Mai departe vom examina metodele verificării răspunsurilor cu expresiile etaloane – în mod special construite.

Organizarea analizei răspunsurilor studenților în cadrul cursurilor asistate de calculator

Analiza răspunsurilor. Eficacitatea unui curs asistat de calculator, într-o mare măsură, se determină de capacitatea mijloacelor analizei răspunsurilor puse la dispoziția elaboratorilor. În scopul determinării mijloacelor analizei răspunsurilor vom examina variantele lor posibile. Mijloacele analizei depind de tipul răspunsurilor și, în consecință, metodele analizei și tipurile răspunsurilor sunt într-o legătură strânsă. Vom evidenția două clase de răspunsuri: textuale și grafice.

Răspunsurile textuale.

1. *Răspuns etalon*. Prin răspuns etalon vom înțelege un singur răspuns selectat de student dintr-o mulțime de variante propuse lui. Analiza răspunsurilor etalon se efectuează în cel mai simplu mod: prin compararea răspunsului cu un singur etalon definit în programul cursului. În funcție de coincidere sau ne coincidere a răspunsului studentului cu răspunsul etalon se planifică reacția cursului.

2. *Răspuns sinonimic*. Analiza acestui răspuns, în mare măsură, este asemănătoare cu analiza răspunsului etalon, însă în programul cursului autorul determină în calitate de etalon o mulțime de sinonime. Răspunsul obținut se consideră corect, dacă el a coincis cu unul din sinonimele determinate.

3. *Răspuns multiplu*. Acest răspuns se introduce de student sub formă de enumerare a obiectelor. Pentru analiza acestui răspuns autorul determină o mulțime care conține enumerarea corectă a obiectelor. Răspunsul se consideră corect, dacă mulțimea obiectelor din răspunsul studentului coincide cu mulțimea de etaloane, indiferent de ordinea de enumerare a lor în răspuns și în mulțimea de etaloane.

Aceste trei tipuri de răspunsuri pentru analiză pot utiliza, de asemenea, și principii de prelucrare parțială prin ignorarea unor simboluri sau a componentelor cuvintelor din răspunsul obținut.

4. *Răspuns numeric*. Acest răspuns se introduce de către student ca o valoare numerică sub formă arbitrară. De exemplu, răspunsurile: 7,+7, 7.0, 0.7E+1, din punct de vedere al studentului, reprezintă una și aceeași valoare. Analiza acestui răspuns este mai complicată, deoarece este imposibil de definit mulțimea finită a tuturor etaloanelor corecte. Pentru analiza acestui tip de răspuns este necesară interpretarea lui ca date aritmetice cu comparare ulterioară cu valoarea adevărată cunoscută a răspunsului.

5. *Răspuns numeric cu precizia stabilită.* Metoda analizei este analogă cu tipul precedent de răspuns, însă la comparare se admite deviere de la valoarea fixată. Acest tip de răspuns se utilizează de către student la efectuarea calculelor aproximative în timpul rezolvării problemelor.

6. *Selectarea unui singular obiect dintr-un meniu*. Răspunsul studentului se efectuează prin alegerea din meniul propus. Această metodă se transformă în efectuarea analizei după un singur etalon.

7. *Selectarea multiplă dintr-un meniu*. Acest răspuns este analogic cu răspunsul multiplu, dar nu cere de la student introducerea răspunsului, ci numai alegerea anumitor obiecte care constituie, din punctul lui de vedere, răspunsul corect.

Răspunsurile grafice.

Dacă sistemul de instruire permite utilizarea imaginilor (diagrame, scheme, desene, etc.) în calitate de răspunsuri, atunci sistemul trebuie să conțină mijloace care va permite autorului formarea etaloanelor răspunsurilor corespunzătoare, a regulilor de echivalență a răspunsurilor studentului. În cursul de instruire studentul trebuie să aibă posibilitatea de a forma răspunsului prin manipularea cu imaginea aplicând anumite reguli dintr-un sistem de reguli.

Vom defini anumite noțiuni:

- prin *transformarea de imagine* vom înțelege transformări geometrice: deplasare, rotire, schimbare a dimensiunilor, adăugare, eliminare sau înlocuirea imaginii întregi sau a unei părți a ei;
- prin *răspuns grafic* vom înțelege o succesiune de acțiuni și/sau transformări, aplicate asupra imaginii inițiale, care, după părerea studentului, satisface condițiile problemei propuse și conduce la rezultatul scontat;
- elementele nederivate, utilizate pentru construirea imaginii, le vom numi *generatrice*;
- prin *obiectul grafic* (OG) vom înțelege o totalitate structurată de generatrice asupra cărei transformarea se îndeplinește ca și asupra unei unității indivizibile;
- prin *biblioteca obiectelor grafice* vom numi o totalitate de OG care este organizată ca un fișier de date și este accesibilă atât autorului cursului, cât și studentului.

Răspunsurile grafice se pot sistematiza după tipurile de manipulări cu obiectul grafic de către student.

Primul tip se alege OG din cele prezentate. În acest caz nu se schimbă nici configurația OG și nici poziția lui pe ecran. Al doilea tip se selectează OG și se schimbă poziției lui.

Al treilea tip este alegerea a câtorva OG și schimbarea pozițiilor lor reciproce și/sau ale configurațiilor OG-uri. Pentru acest tip de răspuns grafic este specific stabilirea anumitor legături structurale între OG-uri.

În al patrulea tip se construiesc noilor OG-uri din generatrice.

În conformitate cu această clasificare vom defini următoarele modalități de răspunsuri grafice care pot fi utilizate de către student.

1. *Răspuns – indicație* la unul sau câteva OG-uri care reprezintă soluția problemei propuse. De exemplu, studentului i se propune câteva obiecte și trebuie de hotărât care dintre ei posedă anumite proprietăți și de a-le marca pe ecran.

2. *Răspuns – deplasare (drag-and-drop)* a OG-ului în poziția necesară a ecranului, totodată se permite rotirea și schimbarea dimensiunilor OG-urilor.

3. *Răspuns – îmbinare a câteva OG-uri* cu respectarea regulilor amplasării reciproce a acestor OG-uri. De exemplu, studentul are la dispoziție „atomi” din care trebuie să compună o „moleculă” ținând cont de componența și legăturile între atomi.

4. *Răspuns – construire.* Acest tip de răspuns este cel mai complicat tip. Acest tip de răspuns grafic apare, de exemplu, la rezolvarea problemelor geometrice de construire.

2.3. Dirijarea procesului de instruire

Procesul de instruire poate fi prezentat cu ajutorul a două grafuri: informațional $G_i(\alpha_n^k, u_n^k)$ și de dirijare $G_d(\alpha_n^k, v_n^k)$ a procesului de instruire. Graful informațional conține vârfurile $\alpha_n^k, (x_n^k, y_n^k)$, unde n este numărul porțiunii de material didactic, k – numărul lecției; x_n^k – mulțimea de lecții, materialul cărora se utilizează la lecția α_n^k; y_n^k – mulțimea de lecții care utilizează materialul lecției α_n^k. Așa dar, vârfurile α_n^k sunt unite cu vârful α_n^m, dacă $\alpha_n^k \subset x_n^k$ sau $\alpha_p^m \subset y_n^k$, adică materialul lecției α_n^k se utilizează la studierea lecției α_p^m. Expunerea materialului nu trebuie să se suprapună sau să se bazeze pe materialul care n-a fost studiat. Astfel de legături pot fi considerate ca informaționale.

Analogic se construiește graful de dirijare $G_d(\alpha_n^k, v_n^k)$, unde v_n^k este mulțimea de muchii ce corespunde legăturilor de dirijare. Legătura între α_n^k și β_κ^k este considerată de dirijare. $\beta_\kappa^k \subset \alpha_\kappa^k$, în corespundere cu care după studierea lecției α_n^k trebuie studiat unul din întrebările β_κ^k. Grafurile respective sunt construite la alcătuirea cursurilor asistate de calculator pentru o disciplină (vezi Figura 2.4).

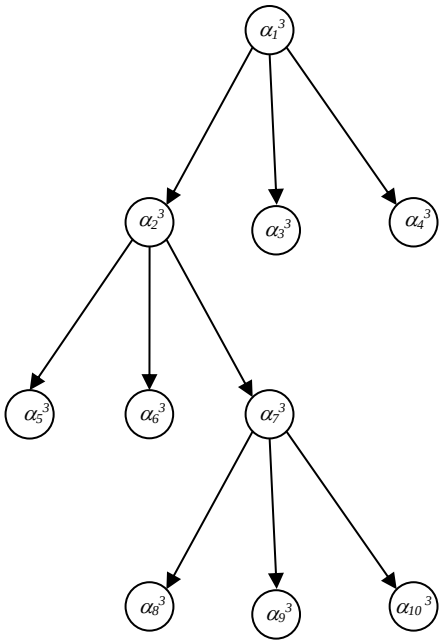


Figura 2.4. Graful de dirijare cu procesul de instruire.

Structura logică a disciplinei poate fi prezentată și prin alte metode, însă modelul cu grafuri este mai evident și efectiv. La utilizarea practică a acestui model este necesar de repartizat vârfurile grafului în nivele. La nivelul zero se include lecția de introducere (vârful de intrare). În mulțimea nivelului n vom include acele lecții, care depind cel puțin de o lecție din nivelul $(n-1)$ și nu depind de nici o lecție care n-a fost inclusă la nivele cu numerele mai mici de $n-2$.

Avantajele acestei metode de reprezentare a materialului sunt:

- posibilitatea de reprezentare multilaterală a temelor materialului studiat;
- excluderea dublării materialului (cu ajutorul muchiilor suplimentare);
- posibilitatea prezentării materialului în adâncime și în lățime;
- ușurează modificarea materialului;

2.4. Scenariile cauzale ca bază în proiectarea cursurilor de instruire asistată de calculator

Cum arată practica la proiectarea cursurilor asistate de calculator profesorul utilizează, de regulă, experiența personală și acel model intuitiv al procesului de instruire care se formează pe parcursul activității sale pedagogice. Experiența aceasta se fixează în planurile lecțiilor, în conspectele prelegerilor, în culegerile materialelor didactice, în lucrările metodice, în scenariile jocurilor de instruire, în testele și lucrările de control. În transpunerea acestei experiențe la calculator, în special, la formarea bazelor de cunoștințe sau la crearea cursurilor de instruire asistate de calculator sunt necesare procedurile formale și structurile de date pentru prezentarea informației și prelucrării ei ulterioare la calculator.

La proiectarea cursurilor asistate de calculator pentru descrierea situațiilor stereotipice, așa ca: expunerea materialului didactic de către profesor, testarea, îndeplinirea lucrării de control, se pot aplica frame-urile sau modificarea lor care se numesc *scenarii* [61-68].

Frame-ul[4] se definește ca o formă specială de reprezentare a cunoștințelor care este definită recursiv. Un frame este format dintr-un număr finit de sloturi, fiecare dintre acestea având un nume și o valoare.

În lucrarea aceasta prin noțiunea de ***frame-ul*** se va înțelege o structură de date care conține informații despre o situație stereotipică. Cu fiecare *frame* se asociază informația care determină în ce mod se va utiliza *frame*-ul în cauză, ce consecințe va avea îndeplinirea lui, ce trebuie de întreprins dacă așteptările acestea nu se vor confirma.

Frame-ul poate fi considerat o rețea ce constă din noduri și legăturile între ei. În calitate de nivelului superior al *frame*-ului la modelarea lecției se utilizează denumirea situației de instruire, iar la nivelele inferioare sunt plasate vârfuri-terminale care trebuie completate cu date concrete. Fiecărui terminal îi pot fi asociate condițiile care determină situațiile în care el poate fi utilizat. Grupele de frame-uri apropiate după semantică se unesc într-un sistem de *frame*-uri.

[4] Толковый словарь по искусственному интеллекту. Авторы-составители: А.Н.Аверкин, М.Г.Гаазе-Рапопорт, Д.А.Поспелов, Available at: https://www.raai.org/pages/UGFnZVR5cGU6MTAwMw==#L588 [Accesed 02.05.2025]

Scenariu se numește descrierea formalizată a unei succesiuni standarde de fapte interdependente care definesc o situație tipică dintr-un domeniu. Acesta poate fi succesiuni de acțiuni sau proceduri care descriu metodele atingerii scopurilor actorilor scenariului (de exemplu, comunicarea informației, obținerea datelor, rezolvarea problemei, răspunsul la o întrebare, determinarea corectitudini răspunsului, aprecierea răspunsului, etc.). Scenariul constă din seturi de sloturi și valorile lor care descriu rolurile, cauzele și succesiunile scenelor, care, la rândul său, sunt succesiuni de anumite acțiuni. Pentru prezentarea situațiilor de instruire dependente de problemă este comod de utilizat așa numitele ***scenarii cauzale*** [63]. Scenariile cauzale, în mod generalizat și structurat, determină o succesiune de acțiuni într-un domeniu în cauză și se descriu ca frame-uri:

CS *numele lecției, numele cursului*:
numele slotului$_1$ (*valoarea slotului*$_1$)
numele slotului$_2$ (*valoarea slotului*$_2$)
...
numele slotului$_n$ (*valoarea slotului*$_n$)

Numele sloturilor (scenariului) reflectă următoarele noțiuni: actorul și participanții scenariului, scopurile și motivele actorilor și ale participanților, cheie, premise, consecințe, situații instructive, numele de sistem al scenariului.

Valoarea slotului poate fi descrisă în notațiile Backus–Naur:

<*valoarea slotului*> ::= <*specificarea valorilor slotului*>: <*valoare*> | <*specificarea valorii*> : <*succesiune de valori*> | NIL
<*specificarea valorilor slotului*> ::= *n* | *s* | *p* | *a* | CS | *f* | *o* | *sys*
<*valoare*> ::= „*nume*”
<*succesiune de valori*> ::= (<*valoare*>, <*valoare*>, … , <*valoare*>) | (<*valoare*>R_1, <*valoare*>R_2, …, <*valoare*>R_n)

Specificarea valorilor slotului indică clasa valorilor slotului dat. Simbolurile la dreapta de la <*specificarea valorilor slotului*> au următoarele valori: *n* – numere, *s* – subiecte de acțiuni, *p* – evenimente, *a* – asociație, CS – scenarii, *f* – proceduri, *o* – obiecte, *sys* – nume de sistem. Simbolurile R_1, R_2, …,R_n determină relațiile de timp sau cauzale.

În scopurile reprezentării adecvate a modelului lecției cu ajutorul scenariilor cauzale a fost analizat procesul de studii și evidențiate situațiile de instruire tipice care apar la predarea diferitelor discipline. Printre situații evidențiate ca bază la proiectarea cursurilor asistate de calculator au fost determinate următoarele situații: expunerea (explicația) materialului didactic de către profesor, desfășurarea testelor, a lucrărilor de control, îndeplinirea exercițiilor, rezolvarea problemelor, cererile de ajutor ale studenților, etc.

În calitate de numele sloturilor care reflectă situațiile instructive se propune utilizarea notațiilor simbolice enumerate mai jos. Numele sloturilor coincid cu numele situațiilor instructive corespunzătoare examinate în 2.1.

P – transmiterea studentului unei anumite porțiuni a informației didactice.

T* –** transmiterea informației care anticipă întrebarea. În calitate de valori ale slotului poate servi informația analogă cu cea ce se conține în slotul ***P.

Q – propunerea studentului să răspundă la o întrebare. În calitate de valori ale slotului se utilizează textul întrebării, al exercițiului, al problemei, etc. Frame-ul care conține slotul ***Q*** trebuie să aibă în componența sa slotul care determină șabloanele răspunsurilor corecte ale studenților.

E – analiza corectitudinii răspunsului după șabloanele date. În calitate de valori ale slotului sunt definite șabloanele răspunsurilor. Corespunderea între sloturile ***Q*** și ***E*** este determinată de specificatorul ***a*** (asociație)

R – slotul replicii reflectă situația reacției profesorului la răspunsul studentului. De regulă, acest slot conține apelul funcției (*f*) care culege statistică despre studierea cursului de către student.

S – oferirea încercării răspunsului repetat la întrebarea propusă. Dacă în scenariu acest slot lipsește, atunci întrebarea va fi propusă o singură dată.

C – controlul depășirii numărului admisibil de încercări de răspunsuri la întrebări. Slotul poate conține apeluri la funcțiile statistice.

U – slotul răspunsului nerecunoscut.

H – acest element al scenariului este destinat pentru acordarea ajutorului. scenariul poate să utilizeze un număr arbitrar de sloturi *H*. Ordinea afișării ajutorului studentului este determinată de parametrul ***n*** al slotului. În afară de aceasta, fiecare slot ***H*** se asociază cu un anumit slot ***Q*** care conține întrebarea sau problema în cauză. Acest gen de asociații ale sloturilor este determinat de specificatorul ***a***.

Pentru a ilustra metoda propusă pentru proiectarea cursurilor de instruire asistate de calculator, vom prezenta un fragment de scenariu al cursului asistat de calculator pentru disciplina „*Limbajul de programare C*”, tema „*Preprocesorul limbajului*”:

(**CS** „*limbajul C*”, „*preprocesor*”:
premise (**CS**:(„*limbajul C*”, „*constante simbolice*” R_1,
„*limbajul C*”, „*test n:12*” R_1**)**
P**(***n*:*13*;

Pentru definirea constantelor simbolice în limbajul C se utilizează directiva #*define*. Ca toate directivele preprocesorului, directiva #*define* începe cu caracterul #. În cel mai simplu caz forma generală a directivei #*define* este următoarea:

#*define* <*identificatorul*><*șirul de substituție*>

Q(*n:1*;

Aveți în fața dvs. un fragment de program:

```
 #define MULT(a, b) a*b
...
{int x, y, z; ... y = 2; z = 4; x = 100/MULT(y-3, z)*3; ... }
```

Cu ce va fi egal *x* după îndeplinirea acestui fragment?)

H(*n:1; a:1;*

Întâlnind undeva în program numele macro definiției, urmat de lista parametrilor actuali, preprocesorul o înlocuiește cu șirul de substituție. Toate intrările parametrilor formali în șirul de substituție sunt înlocuite cu parametrii actuali corespunzători.**)**

H(*n:2; a:1;*

MULT(*y*-3, *z*) se înlocuiește cu *y*-3*z, și instrucțiunea *x* = 100/MULT(*y*-3, *z*)*3; „se transformă” în *x* = 100/*y*-3**z*3;. La îndeplinirea acestei instrucțiuni y este egal cu 2, *z* = 4. Îndepliniți operațiile aritmetice și să obțineți rezultatul corect.

H(*n:3; a:1;*

Ținând cont că prioritatea operațiilor de înmulțire și împărțire este mai mare decât prioritatea adunării și scăderii, în primul rând se efectuează înmulțirea și împărțirea: 100/2=50, 3*4*3=36. După aceasta se îndeplinește scădere: 50-36=14. Acesta și va fi răspunsul corect – 14.**)**

S(*n:4***)**

C(*a:1*;

Cu părere de rău, dvs. n-ați reușit să răspundeți la întrebare. Îndeplinind extensia macroapelului MULTC(*y*-3, *z*), obținem *x* =100/*y*-3**z**31. La îndeplinirea acestei instrucțiuni *y* este egal cu doi, *z* este egal cu patru. Prin urmare, *x*=100/2-3*4*3. Prioritatea operațiilor de înmulțire și împărțire este cea mai mare. Deci, *x* = 50-36. Îndeplinim scăderea și obținem *x*=14. Astfel, răspunsul corect – 14.**)**

E(*a:1;* [*xegal*, xEGAL,x=]14 14**)**

R(*f:stat;* Exact. Răspunsul este absolut corect.**)**

E(*a:1;* [*xegal*, xEGAL,x=]-30 –30**)**

R(*f:stat;***)**

Incorect. Preprocesorul nu pune paranteze și nu efectuează calculele. Acest răspuns ar fi corect dacă directiva #define ar fi avut aspectul următor: #define MULT(a,b) (a*b). Dar în cazul nostru a*b fără paranteze. De aceea la început se efectuează împărțirea și înmulțirea și după aceasta operația de scădere.**)**

E(*a:1;* [*xegal*, xEGAL,x=]-1200 –1200**)**

R(*f:stat;*

Incorect. Acest răspuns ar fi corect dacă directiva #define ar fi avut aspectul următor: #define MULT(a,b) (a)*(b). În acest caz preprocesorul mai întâi va îndeplini împărțirea și înmulțirea și pe urmă operația de scădere.**)**

E(*a:1;* [*xega*l, xEGAL, x=]-75 –75**)**

R(*f:stat;*

Dvs. ați dat răspuns incorect. Preprocesorul face numai substituție textuală. Mai întâi executați substituția formală și apoi efectuați calculele.**)**

Q(*n:2*;

Aveți în fața dvs. fragmentul programului:

```
#define MULT(a, b) a*b
{int x, y, z; ... y=2; z=4; x=100/MULT(y-3, z)*3; ... }
```

Scrieți directiva *#define* MULT(a, b) a*b în așa mod, ca la utilizarea acestei directive înmulțirea să se efectueze complet între termenii a și b. În cadrul răspunsului utilizați MULT ca numele macro funcției și parametrii formali a și b.

H(*n:1; a:2;*

Întâlnind undeva în program numele macro definiției, urmat de lista parametrilor actuali, preprocesorul o înlocuiește cu șirul de substituție. Toate intrările parametrilor formali în șirul de substituție sunt înlocuite cu parametrii actuali corespunzători.)

(mai departe urmează fragmentul scenariului care după structură repetă pe cel expus anterior)

…

cheie (*p:* răspunsuri la întrebări**)**

consecințe(*p:* puncte, acumulate de student în timpul răspunsurilor la întrebări**)**

numele de sistem (*sys: CS 2***))**

În cazul nostru ***R₁*** este relație de precedență. Această relație notează evenimentul care trebuie să fie mai înainte, adică scenariile **CS**:(„limbajul C”, „constante simbolice”) și **CS** („limbajul C”, „test *n*:*12*”**)** trebuie studiate de student obligatoriu înainte de scenariul **CS**:(„limbajul C”, „preprocesor”).

Slotul **cheie** determină evenimentul principal care definește tipul scenariului dat. Slotul **premise** determină condițiile necesare pentru îndeplinirea scenariului, adică condițiile necesare pentru realizarea evenimentului cheie. Slotul **consecințe** determină rezultatele îndeplinirii evenimentului cheie. Scenariul se consideră încheiat dacă a fost realizat evenimentul cheie.

Scenariile elaborate sunt stocate într-o bază de date specială. Ulterior ele se unesc într-o lecție, care, la rândul său, se unesc într-un curs de instruire. Astfel de curs poate fi interpretat de un interpretor special sau în baza acestui curs se poate de generat un modul executabil.

2.5. Recomandări pentru crearea cursurilor de instruire asistate de calculator

Procesul de instruire este o activitate interdependentă de pedagog și elevii săi. Predarea și studierea sunt două laturi a unui și aceluiași proces care are ca scop însușirea de către studenți a unui sistem de cunoștințe, formarea unor deprinderi și îndemânări.

În procesul de instruire profesorul trebuie nu numai să comunice elevilor, studenților anumite cunoștințe, ci și să dirijeze cu activitatea lor în procesul de însușire a cunoștințelor. Totuși, în practica instruirii tradiționale aceste procese sunt departe de a fi organizate optimal.

Neajunsurile principale ale sistemului existent de instruire constă în instruire nediferențiată, expunere nediferențiată a materialului didactic pentru studenți, în influența slabă sau lipsa de influență a rezultatelor controlului curent al însușirii asupra procesului ulterior al instruirii, în pasivitatea relativă a studenților în timpul lecțiilor și în lipsa garanțiilor că studenții vor asimila un volum anumit de cunoștințe prevăzut de program.

Ridicarea eficacității procesului de studii în desfășurarea instruirii asistate de calculator este asigurată prin extinderea componentelor lecțiilor în care sunt create posibilități de diferențiere și individualizare a instruirii. Aceasta permite nu numai de a utiliza sub diferite aspecte particularitățile individuale ale studenților, ci și de a desfășura instruirea efectiv ținând cont de specificul procesului de însușire a materialului de către fiecare student, a face controlul curent al însușirii ca un mijloc care reglează consecutivitatea și conținutul instruirii ulterioare. Controlul instruirii în baza metodelor statistice ale analizei procesului de instruire permite profesorilor și studenților a analiza obiectiv în în ce măsură eforturile lor contribuie la atingerea scopurilor stabilite.

SIAC trebuie să îndeplinească următoarele funcții:

- gestionarea efectivă a activității studentului la studierea disciplinelor;
- stimularea activității de studii și de cunoaștere;
- asigurarea îmbinării raționale a diferitelor metode de activitate de cunoaștere și de studii;
- îmbinarea rațională a diferitelor tehnologii de prezentare a materialului didactic (texte, grafică, sunete, video, animare, etc.).

Materialul didactic se prezintă prin două părți principale: informațională și de control.

Partea informațională dezvăluie conținutul instruirii: definirea noțiunilor, expunerea faptelor, descrierea legităților proceselor și fenomenelor, caracteristice pentru obiectul studiat.

Partea de control este destinată pentru evaluarea activității studenților și se reprezintă prin lucrările de control, problemele, exercițiile, pe care studenții trebuie să le îndeplinească pentru perfecționarea deprinderilor ți îndemânărilor acumulate. Gestionarea instruirii se efectuează în funcție de rezultatele lucrului părții de control care influențează activitatea studentului pentru însușirea mai efectivă a materialului propus.

Structura tipică a unui element de program de instruire, care realizează studierea unei noțiuni, idei, reprezintă seria următorilor pași:

- asigurarea atenției;
- informarea studentului despre scopul lecției;
- aducere aminte despre cunoștințele obținute anterior;
- prezentarea exemplelor tipice;
- instruirea dirijată;
- eliberarea temelor;
- asigurarea legăturii inverse;
- exerciții de control;
- ridicarea capacității de memorare și eficacității de instruire.

La proiectarea cursurilor asistate de calculator pentru descrierea situațiilor stereotipice de instruire se poate de aplicat *scenariile de instruire.*

Scenariul de instruire este o modificare a *scenariului cauzal* și se utilizează pentru descrierea succesiunii de situații de instruire stereotipice. Acestea pot fi succesiuni de situații instructive examinate în 2.4.

Pentru fiecare din situații sunt înaintate anumite cerințe. În particular:

- la expunerea materialului didactic apare necesitatea de prezentare atât a informației textuale, cât și a sunetelor, imaginilor grafice și video, etc.
- la propunerea întrebării răspunsul poate fi obținut în formă construită în mod arbitrar, în forma alegerii unui răspuns dintr-o mulțime de variante, în forma alegerii a mai multor răspunsuri dintr-o mulțime de variante. În afară de aceasta, în situația dată trebuie de reacționat la răspunsurile corecte și incorecte, de oferit studentului un ajutor de mai multe nivele, posibilitatea de redactat răspunsul obținut, de limitare a numărului de încercări de răspuns, etc.
- în timpul testării și lucrărilor de control apare necesitatea creării unui set de întrebări cu mai multe variante. Trebuie de ținut cont că diferite întrebări pot fi apreciate în mod diferit, cu

o notă mai mare sau mai mică. În afară de aceasta, rezultatele, totalurile lucrărilor pot fi calculate după metodici diferite;

- pentru crearea cursurilor adaptive este necesar în anumite locuri ale cursului de a face concluzii despre însușirea materialului pentru a trece la compartimentele cursului care corespund nivelului atins de student.

Au fost expuse unele recomandări proprii pentru organizarea cursurilor asistate de calculator în baza *scenariilor de instruire* și a *situațiilor stereotipice de instruire*. Alte aspecte ale organizării cursurilor asistate de calculator sunt examinate în [55-58].

Tehnologia de proiectare a cursurilor de instruire asistată de calculator elaborate în acest capitol se bazează pe modelul conceptual al domeniului de instruire, concretizat prin identificarea și formalizarea *situațiilor instructive stereotipice*. Tehnologie aceasta se încadrează în paradigma ingineriei/proiectării dirijate de modele [69] (*eng. Model-Driven Engineering* – MDE), care presupune utilizarea modelelor formale drept principal instrument de reprezentare, analiză și generare a componentelor funcționale ale sistemului educațional asistat de calculator.

Proiectarea dirijată de model reprezintă o paradigmă de dezvoltare software în care componentele modelului devin artefacte centrale ale procesului de inginerie. În contextul instruirii asistate de calculator, MDE [70-72] permite definirea explicită a structurilor și comportamentelor educaționale prin intermediul modelelor, facilitând astfel automatizarea și adaptarea procesului de proiectare cursurilor de instruire asistată de calculator.

Limbajul MoCo [23], dezvoltat în cadrul acestei teze și care va fi prezentat în capitolul următor, oferă un cadru formal pentru descrierea situațiilor instructive stereotipice. Prin utilizarea MoCo, se pot defini modele educaționale care pot fi ulterior transformate automat în aplicații de instruire, reducând astfel efortul de dezvoltare și asigurând coerența procesului educațional.

2.6. Concluzii la capitolul 2

Interesul față de procesul de automatizare a instruirii nu se atenuează, el este provocat nu numai de modificările esențiale în soft-ul pentru instruirea asistată de calculator, ci și de necesitățile crescânde în instruire și reciclare a specialiștilor în diferite domenii. Problema organizării muncii profesorilor, autorilor cursurilor asistate de calculator, perfecționării tehnologiei creării cursurilor asistate de calculator capătă o importanță deosebită în societatea informațională. Există o mulțime de limbaje de autor orientate spre probleme, însă nici unul din ele nu satisface, în plină măsură, cerințele înaintate față de limbajele pentru crearea cursurilor asistate de calculator.

Analizând starea actuală în domeniul de computerizare a procesului de instruire, abordat în lucrarea prezentă și în rezultatul analizei procesului de studii și a modelului lui structural se poate de făcut următoarele concluzii:

1. Ținând cont de cerințele pentru fiecare situație în procesul de instruire putem spune că situațiile stereotipice de instruire pot servi ca baza pentru crearea limbajului de descriere a cursurilor de instruire;
2. Modelul structural al procesului de studii poate ajuta profesorii și programatorii la proiectarea și crearea cursurilor asistate de calculator;
3. Situațiile stereotipice de instruire cercetate și modelul structural permit apropierea maximă a limbajului descrierii cursurilor de instruire de limbajul natural în domeniul cercetat;
4. Noțiunea introdusă de *scenariul de instruire* permite crearea limbajului neprocedural de programare pentru descrierea cursurilor de instruire.

5. Limbajul descrierii cursurilor de instruire trebuie să posede următoarele proprietăți:
    - posibilități de creare a unei interfețe multifuncționale, ierarhice și flexibile;
    - forme diverse de prezentare a materialului didactic;
    - mijloace de analiză a răspunsurilor studenților și de dirijare efectuare a controlului procesului de instruire;
    - mijloace de organizare a sistemului HELP-ADVISE;
    - oportunități de utilizare a diverselor strategii de instruire.

3. LIMBAJUL DE DESCRIERE A CURSURILOR ASISTATE DE CALCULATOR MoCo

Limbajul de descriere a situațiilor instructive, dezvoltat în această lucrare, este destinat pentru crearea cursurilor asistate de calculator în baza scenariilor. Limbajul propus permite descrierea atât a părții informaționale, cât și a părții de control. În componența limbajului de descriere a situațiilor instructive sunt incluse mijloacele pentru structurarea materialului didactic, pentru prezentarea informației studentului, inclusiv imagini grafice și video, de analiză a răspunsurilor la întrebări, de organizare a lucrărilor de control și testare, precum și mijloace de includere a informației de îndrumare și a utilitarilor auxiliare [73]. Limbajul se bazează pe noțiunea de situație instructivă stereotipică reprezentată prin scenariul cauzal și este un limbaj declarativ (neprocedural).

Limbajul MoCo [23, 33] este un limbaj specializat și face parte, de asemenea, din categoria limbajelor specifice domeniului [74-78], limbajelor descrierilor cursurilor de instruire asistate de calculator.

În continuare vom evidenția particularitățile și avantajele limbajelor specifice domeniului.

3.1. Avantajele și problemele limbajelor specifice domeniului

Un element-cheie al acestei abordări este utilizarea limbajelor specifice domeniului (LSD LSD-urile permit descrierea cursurilor și a situațiilor de instruire într-o formă apropiată de limbajul de specialitate educațional, facilitând astfel comunicarea între experții pedagogi și dezvoltatorii de software. De exemplu, în loc să se scrie cod general într-un limbaj de programare complex pentru a defini logica unui curs, un expert în instruire poate folosi un LSD dedicat instruirii pentru a specifica secvențe de predare-învățare folosind termeni pedagogici familiari. Prin urmare, LSD-urile contribuie la reducerea distanței semantice dintre conceptele din domeniul educației și implementarea tehnică, făcând posibilă o proiectare *mai rapidă, mai fiabilă și mai prietenoasă* a cursurilor asistate de calculator.

Capitolul de față își propune să caracterizeze limbajele specifice domeniului și să evidențieze avantajele acestora, având în vedere atât literatura de specialitate, cât și exemple relevante din cercetare și industrie. Vom aborda, de asemenea, limitările și provocările asociate cu LSD-urile – aspecte ce trebuie luate în considerare atunci când adoptăm o astfel de soluție. În final, vom sublinia importanța LSD-urilor în domeniul educațional (în special în contextul tezei) și vom prezenta concluziile rezultate.

Caracterizarea limbajelor specifice domeniului (LSD).

Definiție și trăsături generale. Un *limbaj specific unui domeniu* este un limbaj de programare (sau de modelare) dedicat unui *domeniu aplicativ restrâns***,** având o putere de exprimare limitată la

acel domeniu. Spre deosebire de limbajele de uz general (*General-Purpose Languages* – GPL) precum C++, Java, Python etc., care pot rezolva o varietate foarte largă de probleme, un LSD este conceput pentru a reprezenta concis și direct *problemele* și *soluțiile* dintr-un anumit domeniu. Cu alte cuvinte, LSD-urile sacrifică generalitatea în favoarea expresivității într-un domeniu restrâns: ele oferă notații și constructe specializate pentru acel domeniu, făcând soluțiile mai naturale și mai ușor de formulat în limbajul problemei [77]. Astfel, un LSD bine conceput permite exprimarea conceptelor din domeniu la nivelul de abstractizare adecvat, lucru ce duce la câștiguri substanțiale în expresivitate și ușurință în utilizare în comparație cu limbajele de uz general, atunci când este folosit în domeniul său specific de aplicație[78]. Această expresivitate sporită cere o limitare a scopului: un LSD nu este potrivit și de obicei nici capabil să rezolve probleme care ies din sfera lui de aplicabilitate..

În literatura de specialitate, LSD-urile mai apar și sub alte denumiri echivalente, precum *limbaje orientate pe aplicație*, *limbaje specializate*, *limbaje de generația a patra*. Toate aceste denumiri subliniază ideea de *orientare restrânsă* către un anumit set de sarcini sau un anumit domeniu.

Clasificări. Limbajele specifice domeniului pot fi clasificate din mai multe perspective. O împărțire uzuală este în *LSD-uri externe* și *LSD-uri interne* (cunoscute și ca *înglobate* sau *embedded LSD*). Un LSD extern este un limbaj separat, cu propria notație și gramatică, necesitând un *translator* (compilator sau interpret) care să îl transforme fie în cod executabil, fie într-o formă de date manipulabilă de un software gazdă. Prin contrast, un LSD intern este creat *în interiorul* unui limbaj de programare gazdă, folosind idiomuri specifice ale acelui limbaj pentru a obține o sintaxă apropiată de limbajul natural al domeniului. De pildă, în limbajul Ruby au fost construite numeroase LSD-uri interne (sub forma așa-numitelor *fluent interfaces*), profitând de sintaxa supleantă a Ruby, în timp ce LSD-urile externe necesită dezvoltarea de la zero a unui parser. Ambele abordări prezintă avantaje: LSD-urile interne pot reutiliza direct infrastructura limbajului gazdă (compilator, mediu de execuție), pe când cele externe oferă o libertate dea defini orice sintaxă și construcții, nefiind constrânse de regulile unui limbaj existent. Există și LSD-uri *textuale* (majoritatea LSD-urilor clasice, bazate pe text, similare cu limbajele de programare) și LSD-uri *vizuale*, numite adesea *limbaje specifice de modelare* (DSML – *Domain-Specific Modeling Languages*). Acestea din urmă folosesc diagrame sau alte reprezentări grafice pentru a exprima modele de domeniu. De exemplu, UML poate fi privit ca un set de DSML generice pentru modelarea software.

Exemple din diverse domenii. LSD-urile [78] au o istorie îndelungată în informatică și există foarte multe LSD-uri în uz astăzi. Multe nici nu sunt percepute ca limbaje de programare de către utilizatorii lor, deoarece au devenit standarde de facto în domeniul respectiv. De exemplu: SQL – limbajul de interogare a bazelor de date relaționale, HTML – limbaj de marcare pentru pagini web, CSS – limbaj pentru stilizarea paginilor web, BNF – notație pentru specificarea gramaticilor formale,

Makefile – limbaj pentru automatizarea construcției software sau MATLAB – limbaj orientat pe calcule tehnice și matrice, folosit intens în inginerie. Toate acestea sunt LSD-uri în sens larg, fiind create pentru un scop precis: de exemplu, SQL permite exprimarea operațiilor pe baze de date la nivel declarativ, mult mai concis și intuitiv decât dacă s-ar folosi un limbaj general.

În concluzie, putem spune că LSD-urile se caracterizează prin focusul lor îngust și prin apropierea de conceptele unui anumit domeniu. Prin design, ele nu sunt menite să înlocuiască limbajele cu destinație generală, ci să le completeze.

Avantajele limbajelor specifice domeniului

Utilizarea LSD-urilor în dezvoltarea de sisteme software aduce o serie de beneficii semnificative, confirmate de numeroase studii și experiențe practice. În esență, avantajele derivă din faptul că un LSD vorbește limbajul domeniului: soluțiile se exprimă direct în termenii problemelor reale, și nu în jargon tehnic general. Principalele avantaje ale LSD-urilor:

- **Expresivitate și concizie sporite.** LSD-urile [79] permit formularea soluțiilor la un nivel de abstractizare mai înalt, folosind idiomul specific al domeniului. Astfel, o problemă poate fi descrisă într-o formă mult mai *concisă* și *intuitivă* decât ar fi într-un limbaj general. Codul sau modelul scris în LSD omite detaliile nerelevante domeniului și se concentrează doar pe elementele esențiale. Ca urmare, programele LSD rezultate tind să fie *mai scurte* și în mare măsură *auto-documentate*, fiind ușor de înțeles de către cunoscătorii domeniului.

- **Claritate și comunicare îmbunătățite.** Pentru că notarea este apropiată de limbajul de specialitate, un LSD produce specificații pe care experții de domeniu (care nu sunt programatori) le pot *înțelege*. Codul LSD servește astfel și ca documentație a soluției, într-o formă accesibilă părților interesate non-tehnice. Acest lucru îmbunătățește comunicarea între echipa de dezvoltare și experții din domeniu: se creează un teren comun pe care ambele părți îl pot discuta. Chiar dacă în multe cazuri experții de domeniu nu ajung să scrie singuri cod LSD de la zero, simplul fapt că pot parcurge și comenta codul existent reprezintă un avantaj major în eliminarea neînțelegerilor dintre cerințe și implementare. În domeniul educației, de pildă, un instructor poate revizui un scenariu didactic exprimat într-un LSD educațional și poate confirma că acesta corespunde metodologiei dorite, fără a avea nevoie de cunoștințe de programare propriu-zise.

- **Productivitate și rapiditate în dezvoltare.** Prin automatizarea sarcinilor și reutilizarea *șabloanelor* de soluții, LSD-urile contribuie la o dezvoltare mai rapidă a aplicațiilor. Un LSD acționează adesea ca un *generator de cod*: specificațiile în limbajul de domeniu sunt transformate de un compilator în cod executabil sau intermediar.

Un efect colateral este și portabilitatea sporită – specificațiile într-un LSD pot fi folosite pentru a genera implementări pe platforme diverse (de exemplu, același model de domeniu al unui curs ar putea genera atât conținut pentru web, cât și pentru o aplicație mobilă), ceea ce sporește reutilizarea și scade efortul de migrare pe tehnologii noi.

- **Fiabilitate și reducerea erorilor.** LSD-urile [80] pot îmbunătăți calitatea și fiabilitatea sistemelor prin faptul că elimină posibilitatea unor erori de programare de nivel scăzut. LSD-urile duc la creșterea fiabilității sistemelor rezultate. Un alt aspect este că, lucrând la un nivel mai aproape de problemă, dezvoltatorii fac mai puține greșeli de interpretare a cerințelor – codul LSD reflectă direct cerința, deci există mai puține transformări mentale ce pot da greș.

- **Adaptabilitate și reutilizare multiplatformă.** În contextul MDE, specificațiile realizate într-un LSD pot fi considerate *platform-independent models*. Astfel, prin scrierea unei singure descrieri de domeniu, se pot obține **mai multe implementări** pe platforme sau medii diferite, schimbând doar traducătorul (transformarea modelului). Această adaptabilitate înseamnă că aplicațiile bazate pe LSD pot ține pasul cu evoluția tehnologică mai ușor: de exemplu, un LSD educațional care descrie un curs la nivel logic poate fi adaptat să genereze conținut SCORM pentru un LMS sau, alternativ, cursuri interactive pentru o platformă nouă, fără a schimba descrierea de bază. De asemenea, dacă intervin schimbări în cerințele de **business** sau de **politici** (în educație, de exemplu, apar noi metode pedagogice), LSD-ul poate fi extins relativ ușor pentru a acoperi noile concepte, menținând compatibilitatea cu cele existente. Această *evoluție controlată* este susținută și de faptul că LSD-urile **conservă cunoștințele** – ele documentează explicit ce face sistemul, permițând noilor membri ai echipei să învețe mai repede logica domeniului (pentru că e scrisă într-un mod accesibil și structurat).

- **Colaborare între experți de domeniu și programatori.** Poate cel mai important beneficiu în cazul aplicațiilor complexe (precum cele educaționale) este că LSD-urile [81] fac posibilă o colaborare strânsă între cei care dețin cunoștințele de domeniu (ex. pedagogi) și echipa tehnică de implementare. După cum am discutat, un LSD acționează ca un limbaj comun – experții de domeniu pot participa activ la definirea cerințelor direct în LSD sau pot valida specificațiile scrise de dezvoltatori. LSD bine conceput poate fi *înțeles chiar mai bine de expertul de domeniu decât de programator* în unele cazuri, deoarece terminologia îi este familiară. LSD-urile sprijină colaborarea interdisciplinară, ceea ce este esențial pentru succesul proiectelor precum cele din e-learning, unde expertiza pedagogică și cea software trebuie îmbinate.

Având în vedere aceste numeroase beneficii – de la cod mai expresiv și mai clar, la productivitate crescută, colaborare îmbunătățită și calitate superioară LSD-urile au o perspectivă promițătoare în domeniul instruirii asistate de calculator.

Limitări și provocări ale LSD-urilor

Deși avantajele LSD-urilor sunt convingătoare, abordarea bazată pe limbaje specifice domeniului implică și o serie de provocări ce trebuie luate în considerare. Adoptarea unui LSD presupune un compromis între beneficiile obținute și efortul suplimentar de a defini și menține un limbaj nou. Mai jos analizăm principalele limitări și dificultăți ale LSD-urilor [78], așa cum reies din l din experiența practică:

- **Costuri de dezvoltare și mentenanță a limbajului.** Crearea unui LSD de la zero este un proces complex, care necesită atât expertiză în domeniu**, cât și** competențe de inginerie a limbajelor formale (parsere, compilatoare, unelte de editare). Costurile inițiale includ: definirea sintaxei și a semanticii limbajului, implementarea uneltelor (compilator/interpret, editori, eventual integrări IDE). Acest efort poate fi semnificativ, mai ales pentru proiecte mici.

- **Costuri de instruire și adoptare.** Un LSD [81] aduce o notă de noutate în echipă sau în comunitatea de utilizatori: atât programatorii, cât și eventualii experți de domeniu care îl vor folosi trebuie să învețe limbajul. Chiar dacă unul din avantaje este că LSD-ul folosește termeni familiari domeniului, utilizatorii trebuie să se obișnuiască cu sintaxa, cu paradigma de programare impusă de LSD-ul respectiv și cu uneltele specifice. Disponibilitatea limitată a LSD-urilor este o problemă: adesea, nu există un LSD perfect predefinit pentru fiecare nevoie, iar cele existente pot să nu fie cunoscute pe scară largă.

- **Dificultatea delimitării domeniului și a scopului.** Una dintre cele mai subtile provocări în crearea unui LSD este definirea corectă a ariei de acoperire a limbajului. Dacă LSD-ul este definit prea *îngust* (prea specific), s-ar putea să nu acopere toate cerințele reale din domeniu, forțând dezvoltatorii să apeleze din nou la cod într-un limbaj cu destinație generală pentru aspectele neacoperite – ceea ce diminuează mult utilitatea LSD-ului. Pe de altă parte, dacă se încearcă includerea prea multor funcționalități (*supra-extinderea* limbajului), LSD-ul poate deveni greoi, complicat și începe să semene periculos de mult cu un limbaj general (reinventând roata). Găsirea echilibrului între specific și general cere eforturi.

- **Echilibrul între specificitate și reutilizare/integrabilitate.** LSD-urile, prin natura lor, excelează în a rezolva un set specific de probleme, însă asta le poate face mai puțin interoperabile cu alte componente ale sistemului. De exemplu, un modul scris într-un LSD extern trebuie integrat cu restul aplicației. Dacă LSD-ul produce cod într-un limbaj cu destinație generală atunci acel cod trebuie poate ajustat manual la împrejurimi sau depanat, ceea ce complică procesul.

  O altă problemă legată de specificitate este eficiența: un LSD [81] poate sacrifica optimizări fine, iar codul generat ar putea fi mai lent sau mai consumator de resurse decât cel scris manual de un expert. Totuși, trebuie menționat că adesea compilatorul LSD-ului poate aplica optimizări de nivel înalt care compensează aceste pierderi, sau că hardware-ul modern face neglijabile diferențele.

- **Acceptare de către utilizatori.** În mod paradoxal, un LSD foarte puternic și expresiv riscă să devină *complicat de utilizat* pentru specialiștii țintă al domeniului. Cu atât crește complexitatea sintaxei și a conceptelor sale, făcându-l dificil de învățat și folosit de nespecialiști. În general, dacă LSD-ul țintit are ca utilizatori finali persoane fără pregătire tehnică, designerii LSD trebuie să acorde o atenție deosebită simplității notației și suportului pentru utilizator. Fără aceasta, există riscul ca LSD-ul să fie respins de comunitatea pe care ar trebui să o ajute.

- **Limitări date de domeniu și evoluția tehnologică.** Un LSD este strâns legat de *momentul* în care a fost definit și de înțelegerea domeniului la acel moment. Dacă domeniul respectiv suferă schimbări radicale sau apar paradigme noi, LSD-ul existent ar putea să nu mai fie potrivit și ar necesita modificări majore sau chiar reproiectare. Progresul tehnologic poate face anumite LSD-uri învechite sau poate cere adaptarea lor la noi. Așadar, un LSD trebuie privit ca un *artefact viu*, care necesită actualizare continuă în raport cu mediul în care operează.

În sinteză, *dezavantajele și provocările* LSD-urilor includ costurile semnificative de realizare și adoptare, dificultatea proiectării corecte (scop și echilibru), potențiale probleme de interoperabilitate și eficiență, precum și riscuri legate de sustenabilitate în timp. Un LSD *prost proiectat* sau *prost gestionat* poate eșua în a oferi beneficiile scontate, ba chiar poate încetini dezvoltarea dacă devine un obstacol în loc de o facilitate. De aceea, decizia de a introduce un LSD trebuie cântărită în funcție de context: complexitatea domeniului, volumul proiectului, frecvența schimbărilor de cerințe, disponibilitatea de expertiză etc. În multe cazuri, totuși, aceste dificultăți pot

fi atenuate prin bune practici: proiectare iterativă a LSD-ului cu feedback de la utilizatori, folosirea de unelte de dezvoltare a limbajelor, crearea prototipurilor, documentare.

În concluzie putem constata că Limbajele specifice domeniului reprezintă o metodă puternică de a aborda complexitatea dezvoltării software prin *aducerea soluțiilor mai aproape de problemă*. Ele oferă un limbaj comun între experții de domeniu și dezvoltatori, cresc expresivitatea și claritatea specificațiilor și permit automatizarea unei părți a procesului de implementare. Avantajele LSD-urilor – de la productivitate și fiabilitate sporite, la o mentenanță mai ușoară. În domeniul educațional, LSD-urile pot să faciliteze proiectarea cursurilor. În contextul acestei teze, adoptarea unui LSD pentru modelarea situațiilor instructive stereotipice este fundamentată de tocmai aceste beneficii: posibilitatea de a formaliza metodologii educaționale într-un mod clar și verificabil..

Pe de altă parte, analiza limitărilor arate că LSD-urile nu sunt un panaceu universal. Ele vin cu costuri și riscuri: necesită investiție inițială semnificativă, un proces riguros de proiectare și testare. Importanța LSD-urilor constă așadar în aplicarea lor judicioasă, acolo unde complexitatea domeniului justifică efortul de specializare. În astfel de situații, beneficiile depășesc cu mult costurile, LSD-urile permit echipelor de dezvoltatori să creeze soluții altfel greu de obținut, reducând distanța de la idee la implementare.

Avantajele limbajelor specifice domeniului – expresivitate, claritate, reducerea erorilor, automatizare, adaptabilitate, colaborare facilitată, suport pentru validare și verificare – le recomandă ca instrumente valoroase în ingineria software modernă bazată pe modele. LSD-urile pot juca un rol crucial în dezvoltarea de sisteme educaționale inovative. Prin utilizarea LSD-urilor, putem realiza sisteme mai apropiate de cerințele utilizatorilor finali, mai ușor de evoluat odată cu domeniul și, în ultimă instanță, *mai eficiente și mai fiabile*.

În continuare vom prezenta descrierea limbajului specific domeniului de instruire asistată de calculator – MoCo.

3.2. Structurile de bază

Structurile de bază ale limbajului sunt determinate în modul următor:

<cifră> ::= ***0*** | ***1*** | ***2*** | ***3*** | ***4*** | ***5*** | ***6*** | ***7*** | ***8*** | ***9***

<literă> ::= ***A*** | ***…*** | ***Z*** | ***a*** | ***…*** | ***z*** | ***A*** | ***…*** | ***Я*** | ***a*** | ***…*** | ***я***

<simbol special> ::= ***#*** | ***%*** | ***@*** | ***$*** | ******* | ***/*** | ***+*** | ***-*** | ***_*** | ***(*** | ***)*** | ***[*** | ***]*** | ***{*** | ***}*** | ***^*** | ***“*** | ***,*** | ***.*** | ***:*** | ***;*** | ***<*** | ***>*** | ***=*** | ***!*** | ***?*** | | | ***~***

<simbol> ::= *<literă>* | *<cifră>* | *<simbol special>*

<întreg fără semn> ::= *<cifră>* [*<întreg fără semn>*]

<întreg cu semn> ::= [*<plus-minus>*] *<întreg fără semn>*

<*plus-minus*> ::= **+** | **-**

<*identificator*> ::= <*literă*> [<*identificator1*>]

<*identificator1*> ::= <*literă*> [<*identificator1*>] | <*cifră*> [<*identificator1*>]

<*denumirea fișierului*> ::= <*identificator*> [. <*identificator1*>]

<*valoarea culorii*> ::= ***Red*** **|** ***Green*** **|** ***Blue*** **|** ***Yellow*** **|** ***Black*** **|** ***White*** **|** ***Magenta*** **|** ***Brown*** **|** ***Cyan*** **|** ***Gray*** **|** <*întreg fără semn*>

<*etichetă*> ::= #<*denumirea etichetei*> :

<*denumirea etichetei*> ::= <*identificator*>

<*expresie*> ::= <*expresie simplă*> [<*operator de comparare*> <*expresie simplă*>]

<*operator de comparare*> ::= < | <= | = | >= | > | !=

<*expresie simplă*> ::= [<*plus-minus*>] <*term*> [<*succesiune de termi*>]

<*succesiune de termi*> ::= <*plus-minus*> <*term*> [<*succesiune de termi*>] | ***Or*** <*term*> [<*succesiune de termi*>]

<*term*> ::= <*factor*> [<*succesiune de factori*>]

<*succesiune de factori*> ::= <*înmulțire-împărțire*> <*term*> [<*succesiune de factori*>] | ***And*** <*term*> [<*succesiune de factori*>]

<*înmulțire-împărțire*> ::= * | /

<*factor*> ::= <*datele din bd*> | **(**<*expresie*>**)** | ***Not*** <*factor*> | [<*diapazonul lecțiilor*>] | <*constanta*>

<*diapazonul lecțiilor*> ::= <*numărul lecției*> [. <*numărul lecției*>] [<*diapazonul lecțiilor*>]

<*constanta*> ::= <*număr*> | "<*text*>" | ***True*** | ***False***

<*număr*> ::= <*întreg cu semn*> | <*număr real*>

<*număr real*> ::= <> [**.**<Întreg fără semn>]

<*datele din bd*> ::= ***LName*** **|** ***Fname*** **|** ***Chapter*** **|** ***Fdate*** **|** ***Ftime*** **|** ***Ldate*** **|** ***Ltime*** **|** **Ctime** **|** **CDate** **|** ***Active_Time*** **|** ***R_Answer_total*** **|** ***W_Answer_total*** **|** ***R_Answer*** **|** ***W_Answer*** **|** ***N_of_Questions*** **|** ***Current_N_of_Questions*** **|** ***N_Help*** **|** <*parametrul autorului*> | <*denumirea câmpului din bd*>

<*denumirea câmpului din bd*> ::= <*text*>

<*text*> ::= <*simbol*> [<*text*>]

<*parametrul autorului*> ::= ***P0*** | ***P1*** | ***P2*** | ***P3*** | ***P4*** | ***P5*** | ***P6*** | ***P7*** | ***P8*** | ***P9*** | ***P10***

Datele din Baza de Date au următorul sens:

- ***Name –*** nume;
- ***Fname*** – prenume;
- ***Chapter*** – denumirea compartimentului studiat la moment;

- ***Fdate–*** data începerii lucrului cu acest curs;
- ***Ftime –*** timpul începerii lucrului cu acest curs;
- ***Ldate –*** data curentă a sfârșitului lucrului cu cursul;
- ***Ltime –*** timpul curent a sfârșitului lucrului cu cursul;
- ***Ctime –*** timpul curent;
- ***Date –*** data curentă;
- ***Active_Time*** – durata lucrului cu acest curs;
- ***R_Answer_total –*** numărul total de răspunsuri corecte;
- ***W_Answer_total*** – numărul total de răspunsuri incorecte;
- ***R_Answer –*** numărul răspunsurilor corecte în compartimentul curent;
- ***W_Answer –*** numărul răspunsurilor incorecte în compartimentul curent;
- ***N_of_Questions –*** numărul de întrebări adresate studentului în acest curs;
- ***Current_N_of_Questions –*** numărul de întrebări adresate studentului în compartimentul curent;
- ***N_Help –*** numărul de apelări la ajutor.

Numele altor câmpuri ale Bazei de Date sunt date de autor în situațiile instructive, informația care se păstrează în Baza de Date. În calitate de denumire a câmpului al Bazei de Date poate fi denumirea testului, a lucrării de control, etc.

În limbaj sunt prevăzute linii-comentarii. Pentru introducerea comentariilor în curs este necesară înscrierea simbolului @ în prima poziție a rândului respectiv în fișierul textual.

3.3. Structura cursului

Cursul este totalitatea *lecțiilor* anticipate de frontispiciul cursului. Cursul poate fi localizat în unul sau mai multe fișiere. Toate fișierele surse ale cursului au extensii „.moc” (moco course file). Fișierul proiectului de curs are extensia ".mcp" (moco project) și structura standard a fișierului *.ini* în Windows.

Sintaxa

<*curs*> ::= <*frontispiciul cursului*> <*blocul lecțiilor*>

<*blocul lecțiilor*> ::= <*lecția*> | [<*blocul lecțiilor*>]

Semantica. La lansarea cursului apare o imagine grafică sau alte informații și butoanele pentru intrare în program. Ulterior apare conținutul cursului format automat din denumirile lecțiilor (*Lesson*).

3.4. Frontispiciul cursului

Frontispiciul cursului este informația care apare la lansarea cursului.

Sintaxa

<*frontispiciul cursului*> ::= **_Start_Up** <*informație*> **_End_Start_Up**

Exemplu

_Start_Up *English Lessons* |***Image** = Welcome.jpg*| *Computer assisted course for engineer students*.
_End_Start_Up

Semantica. La începutul execuției cursului apare informația din *Start_Up*. În acest caz apare denumirea cursului și imaginea *Welcome.jpg*. împreună cu butoanele (*hyperlink*-uri): „*lansarea cursului* (*începutul cursului*)", „prelungirea cursului (reluarea de la ultima deconectare)" și „Ieșirea din curs". În general, <*informație*> are același sens ca și în situația ***Information***. Situația *Start_Up* se include într-un fișier aparte.

3.5. Lecția

Fiecare lecție este o totalitate de situații instructive, situații de luare a deciziilor și apelare a funcțiilor auxiliare. Ca bază au fost luate următoarele situații instructive:

- expunerea materialului didactic;
- întrebările;
- testările;
- lucrările de control.

Sintaxa

<*lecția*>::= [<*etichetă*>] ***_Lesson*** <*identificatorul lecției*> <*blocul situațiilor lecției*> ***_End_of_Lesson***

<*identificatorul lecției*> ::= **(**<*numărul lecției*> "<*denumirea lecției*>" [<*condiția intrării în lecție*>] **)**

<*numărul lecției*> ::= <*întreg fără semn*> [**.**<*numărul lecției*>]

<*denumirea lecției*> ::= <*text*>

<*condiția intrării în lecție*> ::= <*expresie*>

<*blocul situațiilor lecției*> ::= [<*etichetă*>] <*situația instructivă*> [<*blocul situațiilor lecției*>] | [<*etichetă*>] <*parametrul cursului – manual*> [<*blocul situațiilor lecției*>]

<*parametrul cursului – manual*> ::= ***_Manual_Enable*** | ***_Manual_Disable***

<*situația instructivă*> ::= <*situația expunerii informației*> | <*situația întrebare*> | <*situația test*> | <*situația lucrare de control*> | <*situația de ramificare*> |

<situația apel de funcție>

Exemplu

*_Lesson (*10 “Verbele în limba engleză” [1..2, 4..7])
    _Information În această lecție vă veți familiariza cu …
    …
    _Question Răspundeți la întrebarea …
    ...
_End_of_Lesson
*_Lesson (*11 “Timpurile verbelor în limba engleză”)
    _Information În această lecție vă veți familiariza cu …
    …
    _Question Răspundeți la întrebarea …
    ...
_End_of_Lesson

Condiția de intrare în lecția menționată ne stipulează că lecția zece poate fi parcursă de student numai dacă au fost parcurse prima, a doua lecție și lecțiile de la patru la șapte.

Semantica. În cadrul alcătuirii cursului de instruire lecțiile sunt unități de bază. Eticheta lecției se utilizează la organizarea ramificațiilor în curs. Numerele lecțiilor și denumirea lor (identificatori) servesc pentru formarea automată a conținutului cursului. Condiția de intrare în lecție servește ca un criteriu de bază pentru a putea fi admis la această lecție. Intrare în lecție se permite numai la îndeplinirea unor anumite condiții.

3.6. Situații instructive

Operatorul situației instructive începe cu notația situației respective și continuă până la următoarea situație.

Vom trece în revistă aceste situații.

Situația expunere a materialului didactic (Information)

Această situație se utilizează pentru a oferi studentului materialul didactic necesar.

Sintaxa

<situația expunerii informației> ::= ***_Information*** *<informația>*

<informația> ::= *<informația1>* [*<informația>*]

<informația1> ::= *<text>* | *<informație adăugătoare>*

<informație adăugătoare> ::= **|*** *<date din bd>* ****|*** | **|*** *<parametru adăugător>* ****|***

<parametru adăugător> ::= *<parametru image>* | *<parametru audio>* | *<parametru video>* |
    <parametru fontcolor> | *<parametru font>* | *<parametru color>*

<parametru image> ::= ***_Image =*** *<denumirea fișierului>*

<parametru audio> ::= ***_Audio =*** *<denumirea fișierului>*

<parametru video> ::= ***_Video =*** *<denumirea fișierului>*

<*parametru fontcolor*> ::= ***_Font_Color =*** <*culoare*>

<*parametru color*> ::= ***_Color =*** <*culoare*>

<*parametru font*> ::= ***_Font*** = <*stilul caracterelor*>

<*stilul caracterelor*> ::= ***B***[***old***] [<*stilul caracterelor*>] | ***I***[***talic***] [<*stilul caracterelor*>] | ***U***[***nderlined***] [<*stilul caracterelor*>]

Exemplu

_Information Aveți în față textul ce ilustrează utilizarea verbelor incorecte în limba engleză la timp trecut. |* *Color = Blue* *| |* *Font_Color = Red* *| Cu litere roșii pe fond albastru |* *Color = White* *| |* *Font_Color =Black* *| sunt însemnate verbele incorecte. Dictorul va expune textul. |* *Audio = Verbs.wav* *| Today we are going …

Semantica. În situația didactică de mai sus în fereastra cursului pentru student poate fi oferit un text, o informație din baza de date, o imagine grafică, video și sunet. Textul poate fi prezentat în diverse culori. Culorile indicate în parametrii ***Color*** și ***Font_Color*** persistă până la sfârșitul textului sau până la modificarea lor.

Situația întrebare (Question)

Situația indicată se utilizează pentru verificarea însușirii materialului. În text vom întâlni prescurtarea P – Parametru.

Sintaxa

<*situația întrebare*> ::= <*întrebare free*> | <*întrebare single*> | <*întrebare multiple*>

<*întrebare free*> ::= ***_Question (***[***_Free*** [***,*** <*numărul încercărilor*>]]***)***
<*informație*> <*blocul întrebării free*>

<*întrebare single*> ::= ***_Question (_Single*** [***,*** <*numărul încercărilor*>]***)***
<*informație*> <*blocul întrebării single*>

<*întrebare multiple*> ::= ***_Question (_Multiple*** [***,*** <*numărul încercărilor*>]***)***
<*informație*> <*blocul întrebării multiple*>

<*numărul încercărilor*> ::= <*întreg fără semn*>

<*blocul întrebării free*> ::= <*p răspuns de tip free*> | <*p reacție*> | <*p ajutor*> |
<*p depășirea încercărilor*> | <*p răspuns neprevăzut*> | <*p redactare*> |
<*blocul întrebării free*> [<*blocul întrebării free*>]

<*blocul întrebării single*> ::= <*p răspuns de tip single*> | <*p reacție*> | <*p ajutor*> |
<*p depășirea încercărilor*> |
<*blocul întrebării single*> [<*blocul întrebării single*>]

<*blocul întrebării multiple*> ::= <*p răspuns de tip multiple*> | <*p reacție*> | <*p ajutor*> |
<*p depășirea încercărilor*> | <*p diagnostica*> |
<*blocul întrebării multiple*> [<*blocul întrebării multiple*>]

Exemplu

_Question (_Free , 3*)* Introduceți, vă rog, verbul “to go” la timpul trecut.
$(_Correct, 2**,** 1 *)* [W,w]ent
$(_Reaction) Corect!
$(_Wrong, 2**,** -1 *)* [G,g]one
$(_Reaction) Greșeală. Gone – este participiu trecut al verbului “to go”.
$(_Undefined) Răspunsul este neclar. Răspundeți laconic, printr-un cuvânt.
$(_eXceed) Deja aveți patru încercări de răspuns la această întrebare. Nu ghiciți. Răspunsul corect – “went”.

În exemplul acesta studentului îi va fi propusă o întrebare la care trebuie să răspundă într-o formă de expunere liberă. Numărul maxim de încercări la întrebarea dată este - 3. Dacă studentul răspunde “Went” sau “went”, atunci apare inscripția „Corect!” și trece la următoare întrebare sau situație instructivă. În cazul răspunsului “Gone” sau “gone” apare replica “Greșeală. Gone – este participiu trecut al verbului “to go” și dacă nu sunt consumate toate încercările pentru răspuns va avea posibilitate încă o dată să răspundă la această întrebare. În cazul dacă încercările la răspunsuri sunt consumate în afară de replica “Greșeală. Gone – este participiu trecut al verbului “to go”, apare încă o replică: „Deja aveți patru încercări de răspuns la această întrebare. Nu ghiciți. Răspunsul corect – went”. În cazul răspunsului diferit de “Went”, “went”, “Gone” sau “gone” apare mesajul: „Răspunsul este neclar. Răspundeți laconic, printr-un cuvânt.”.

Exemplu

_Question (_Single, 2*)* Alegeți printre verbele enumerate timpul trecut al verbului “shake”. (Apăsați pe butonul situat vizavi de răspunsul corect).
$(_Wrong, 0*)* shared
$(_Wrong, 0*)* shone *$(_Reaction)* Greșeală. Este un alt verb.
$(_Wrong, 0*)* shaken *$(_Reaction)* Greșeală.
$(_Correct, 1*)* shook *$(_Reaction)* Corect!
$(_eXceed) Nu ghiciți. Concentrați-vă înainte de a alege răspunsul.

În acest exemplu se oferă două încercări de răspuns la întrebare. Din patru variante de răspunsuri numai unul este corect. La alegerea primei sau a doua variantă apare replica „Greșeală. Este un alt verb.”. La alegerea variantei trei replica este „Greșeală.”. Alegând varianta a patra de răspuns apare replica „Corect!”. Afișarea replicilor este determinată de parametrul ***Reaction***. Primele trei variante de răspuns sunt marcate ca fiind incorecte (parametrul ***Wrong***) și sunt apreciate cu 0 puncte, ultima variantă corecta (***Correct***) obținând 1 punct.

Exemplu

_Question (_Multiple) Alegeți printre verbele enumerate timpul trecut al verbului “to drink”. (Apăsați pe butonul situat vizavi de răspunsul corect).
$(_Wrong) drinked
$(_Correct) drank
$(_Wrong) drinkered
$(_Correct) drunk
$(_Wrong) drinking
$(_Diagnostics _Correct, _All _Correct , 2*)*

$(_Reaction) Absolut corect.
$(_Diagnostics _Correct, _Less 2 *_Wrong 1 _Correct,* 1*)*
$(_Reaction) Parțial corect. Ați ales numai un răspuns corect.
$(_Diagnostics _Wrong, _More 1 *_Wrong,* 0*)*
$(_Diagnostics _Wrong, _More 2 *_Wrong,* -1*)*
$(_Reaction) Greșeală.

Acest tip de întrebare presupune prezența a mai multor variante corecte de răspuns. Ca și în exemplul anterior, variantele corecte de răspunsuri sunt marcate cu parametrul ***Correct***, cele incorecte – cu ***Wrong.*** Punctele obținute pentru răspunsurile corecte și incorecte sunt indicate în parametrii ***Diagnostics Correct*** și ***Diagnostics Wrong***. Alături de acești parametri se înscrie parametrul ***Reaction***, care răspunde de replicile respective. Amănunte despre parametrul ***Diagnostics*** sunt expuse în capitolul „Parametrul Diagnostică”.

Semantica. Situația instructivă ***Question*** (întrebare) este concepută pentru elaborarea întrebărilor, analiza răspunsurilor, acumularea punctelor pentru răspunsurile corecte și incorecte și determinarea numărului de încercări pentru răspunsurile oferite de student. În această situație poate fi prevăzut un ajutor pentru student în cazul răspunsului la întrebare. Răspunsul poate fi redactat înainte de a fi analizat sau verificat.

În acest caz:

- *întrebare Free* este întrebarea, răspunsul la care se introduce sub o formă de expunere liberă;
- *întrebare Single* – întrebarea cu o mulțime de variante de răspunsuri, din care o variantă e corectă;
- *întrebare Multiple* – întrebarea cu o mulțime de variante de răspunsuri, din care pentru alcătuirea răspunsului corect este necesară alegerea a câtorva variante are răspunsului corect.

Implicit tipul de întrebare este ***Free***. Numărul încercărilor determină numărul *maxim* de încercări de răspunsuri la întrebare, implicit se consideră 1.

Șabloanele răspunsurilor sunt determinate de parametrii ***Wrong*** și ***Correct***. Replicile la răspunsuri se specifică de parametrul ***Reaction***. Replica la un răspuns neprevăzut este determinată de parametrul ***Undefined***. Pentru replica în cazul depășirii numărului de încercări de răspunsuri la întrebarea propusă răspunde parametrul ***eXceed***. Numărul de puncte acordat pentru răspunsurile corecte și incorecte sunt determinate de parametrii ***Wrong*** și ***Correct*** pentru întrebările de tip *Free* și *Single* și de parametrii ***Diagnostics Correct***, ***Diagnostics Wrong*** pentru întrebările de tip *Multiple.* Ajutorul oferit pentru răspuns la întrebarea propusă este determinat de parametrul ***Help***. Prelucrarea prealabilă a răspunsului este realizată în conformitate cu parametrului ***Edit***.

Parametrul *Ajutor* (*Help*)

Parametrul ***Ajutor*** este prevăzut ca în anumite situații să ajute studenții să răspundă la întrebări sau să rezolve exerciții.

Sintaxa

<*p ajutor*> ::= ***$(_Help)*** <*informație*>

Exemplu

_Question (_Free , 3*)* Traduceți în limba engleză fraza: „Ați putea să mă ajutați?”
$(_Edit) CM?.SP
$(_Correct, 2**,** 1*)* [C,c]ouldyouhelpmeplease
$(_Reaction) Corect!
$(_Correct, 2**,** 1*)* [C,c]ouldyouhelpme
$(_Reaction) Corect!
$(_Wrong, 2*)* &
$(_Reaction) Greșeală.
$(_Undefined) Răspunsul este neclar.
$(_eXceed) Cu părere de rău n-ați reușit să răspundeți la această întrebare. Răspunsul corect: Could you help me, please? sau Could you help me?
$(_Help) Răspunzând la această întrebare ați avea nevoie de următoarele cuvinte: could, me, you, help, please.
$(_Help) Nu uitați despre ordinea cuvintelor în propozițiile la modul interogativ. În cazul acesta verbul ajutător este în față, după care urmează pronumele etc.
$(_Help) În limba engleză fraza: „Ați putea vă rog să mă ajutați?” se traduce ca fiind: “Could you help me, please?”. Acordați atenție ordinii cuvintelor în propoziție.

Semantica. Ajutorul apare de fiecare dată dacă aste apăsată tasta funcțională ***F1*** sau butonul pe ecran cu inscripția respectivă. (Se presupune că dacă în situația corespunzătoare se prevede un ajutor, atunci în fereastra corespunzătoare va apare butonul ***Ajutor***). De regulă, ajutorul este organizat în câteva nivele. La prima apelare la ajutor se oferă ajutorul nivelului unu, la apelarea repetată nivelul doi, etc. Dacă numărul apelării este mai mare decât numărul de parametri ***Help,*** atunci se utilizează ultimul parametru ***Help.*** În cazul în care ajutorul nu este prevăzut, atunci apare mesajul: „Ajutorul nu este prevăzut”.

Parametrul *Redactare* (*Edit*)

În cazul dacă răspunsul este acceptat în formă de expunere liberă, poate apărea necesitatea redactării prealabile a lui (de exemplu, înlăturarea spațiilor). Pentru aceasta este prevăzut parametrul Redactare.

Sintaxa

<*p redactare*> ::= ***$(_Edit)*** <*succesiunea semnelor de înlăturare*>

<*succesiunea semnelor de înlăturare*> *::=* <*identificatorul semnului de înlăturare*> [, <*succesiunea semnelor de înlăturare*>]

<*identificatorul semnului de înlăturare*> ::= ***SP*** | ***CM*** | ***DG*** | ***LT*** | ***SC*** | <*simbol*>

Exemplu

$(_Edit) CM?!.4

Din răspunsul obținut vor fi înlăturate toate virgulele (CM), semnele de întrebare și exclamare, punctele și cifrele 4.

Semantica. Cu ajutorul parametrului ***Edit*** din răspuns sunt înlăturate simbolurile ce nu afectează sensul răspunsului, dar care pot fi utilizate de student. De regulă, acestea sunt semne de punctuație, semnul plus în fața numărului pozitiv, etc.

În parametrul ***Edit*** este posibilă utilizarea următoarelor notații: *SP* (*space*) – spațiu, *CM* (*comma*) – virgulă, *DG* (*digits*) – toate cifrele, *LT* (*letters*) – toate literele, *SC* (*special characters*) – toate simbolurile speciale.

Intrarea repetată a unui oarecare simbol în succesiunea semnelor de înlăturare se ignoră. În cazul absenței parametrului dat răspunsul nu se redactează.

Parametrul *Depășire a încercărilor de răspuns* (*eXceed*)

Parametrul **eXceed** se utilizează pentru oferirea replicii în cazul depășirii numărului de încercări de răspunsuri la întrebare.

Sintaxa

<*p depășirea încercărilor*> ::= ***$(_eXceed)*** <*informație*>

Exemplu

_Question (_Free, 3*)* Traduceți în limba engleză fraza …
…
$(_eXceed) Cu părere de rău, nu ați reușit să răspundeți la întrebare.
…

În exemplul dat după trei încercări fără succes va apărea replică: „Cu părere de rău nu ați reușit să răspundeți la întrebare”.

Semantica. Informația scrisă în parametrul ***eXceed*** se utilizează pentru oferirea replicilor în cazul depășirii încercărilor de răspuns la întrebare definite în situațiile ***Question.*** În cazul absenței acestui parametru apare mesajul standard „Ați depășit numărul încercărilor de răspuns”.

Parametrul *Răspuns neprevăzut (Undefined)*

Parametrul ***Undefined*** se utilizează pentru oferirea replicii în cazul dacă răspunsul nu a coincis nici cu unul din șablonurile prevăzute de răspuns.

Sintaxa

<*p răspuns neprevăzut*> ::= ***$(_Undefined)*** <*informație*>

Exemplu

$(_Undefined) Răspunsul este neclar. Răspundeți laconic, printr-un cuvânt.

Semantica. Parametrul ***Undefined*** se utilizează împreună cu răspunsul în forma de expunere liberă (*Free*). În cazul dacă rezultatul obținut este neprevăzut, atunci apare inscripția cu informația

corespunzătoare. În cazul absenței acestui parametru va fi oferită inscripția standard „Răspunsul este neclar”.

Parametrul *Răspuns de tip liber(Free)*

Parametrul *răspuns de tip* ***Free*** determină faptul că răspunsul la întrebare va fi analizat cu ajutorul șabloanelor.

Sintaxa

<p răspuns de tip free> ::= ***$***(*<tipul răspunsului>, <modul de comparare cu șablon>* [, *<puncte pentru răspuns>*]**)** *<șablon>*

<tipul răspunsului> ::= ***_Correct*** | ***_Wrong***

<modul de comparare cu șablon> ::= ***1*** | ***2*** | ***3***

<șablon> ::= *<text>*

<puncte pentru răspuns> ::= *<întreg cu semn>*

Exemplu

_Question (_Free , 2*)* Traduceți cuvântul englez „magazin”.
$(_Correct, 2, 1*)* revistă
$(_Reaction) Bravo! Traducerea excelentă.
$(_Wrong, 3, -1*)* alimentară magazin
$(_Reaction) Este absolut incorect.
$(_Undefined) Greșeală.
$(_eXceed) Aveți deja trei încercări de răspuns. Răspunsul corect revistă.

Răspunsul la aceasta întrebare va fi dat în formă de text. Răspunsul corect se consideră cuvântul „revistă”. Pentru acest răspuns se oferă un punct. Analiza răspunsului se realizează conform modului doi de analiză. Răspunsul „alimentară” se consideră incorect și din cauza lui din numărul total de puncte se scoate un punct. Analiza răspunsului se va efectua conform modului trei.

Semantica. Parametrul *răspuns de tip Free* se utilizează în cazul întrebării de tip *Free* ***(_Question(_Free))***. Se presupune că răspunsul la întrebare apare sub formă textuală liberă. Prelucrarea răspunsului se realizează cu ajutorul șabloanelor. *Șablonul* este textul format după anumite reguli. Mai detailat despre structura și construirea șabloanelor se descrie capitolul „Șabloane”.

Modul de comparare a răspunsului cu șablonul poate fi unul din următoarele: 1 – analiza după șablon unitar, 2 – analiza după șablon special, 3 – analiza după mai multe șabloane.

Răspunsul poate fi considerat corect ***Correct*** sau incorect ***Wrong***, numai în cazul dacă el este recunoscut. Pentru răspuns se indică numărul de puncte acordat pentru răspunsuri fie corecte, cât și cele incorecte. Implicit – 0 puncte. Pentru răspunsuri incorecte are sens acordarea punctelor cu semn negativ sau 0.

Parametrul *Răspuns de tip singur* (*Single)*

Acest parametru se utilizează pentru formarea întrebării în formă de meniu, pentru împărțirea răspunsurilor în corecte și incorecte și pentru acordarea punctelor la alegerea variantelor respective de răspuns.

Sintaxa

<p răspuns de tip single> ::= ***$(****<tipul răspunsului>*[**,** *<puncte pentru răspuns>*]***)*** *<informație>*

<tipul răspunsului> ::= ***_Wrong*** | ***_Correct***

<puncte pentru răspuns> ::= *<întreg cu semn>*

Exemplu

_Question (_Single, 3*)* Determinați partea de vorbire la care se referă cuvântul "shaken". (Alegeți din răspunsurile indicate mai jos răspunsul corect)
$(_Wrong, 0*)* substantiv
$(_Wrong, 0*)* verb
$(_Wrong, -1*)* prepoziție *$(_Reaction)* Eroare.
$(_Correct, 1*)* participiu *$(_Reaction)* Corect!
$(_eXceed) Nu ghiciți. Concentrați-vă asupra alegerii răspunsului corect.

Răspunzând la această întrebare studentul poate face o alegere unică din patru răspunsuri propuse. Sunt oferite două încercări de răspuns la o întrebare. Din patru variante de răspunsuri numai unul este corect. La alegerea unuia din primele trei variante de răspuns apare replica „Eroare". La alegerea variantei patru replica va fi „Corect!". Primele trei variante de răspunsuri sunt marcate ca fiind incorecte (parametrul ***Wrong***). Pentru varianta unu și doi se oferă câte 0 puncte, pentru varianta trei se penalizează cu 2 puncte. Ultima variantă este corectă (***Correct***), pentru ea se oferă 1 punct.

Semantica. Variantele răspunsului se enumeră în ordinea alcătuirii meniului. În fața răspunsului este înscris parametrul ce determină răspunsul corect sau incorect (***Correct*** sau ***Wrong)*** și numărul de puncte. În calitate de replică care ar trebui să apară în cazul alegerii variantei se ia informația din cel mai apropiat parametrul ***Reaction***.

Parametrul *Răspuns de tip multiplu (Multiple)*

Aceasta situație se utilizează pentru formarea meniului de alegere și pentru separarea variantelor de răspuns în incorecte și corecte.

Sintaxa

<p răspuns de tip multiple> ::= ***$(****<tipul răspunsului>)* *<informație>*

Exemplu

_Question (_Multiple) Determinați partea de vorbire la care se referă cuvântul "cut". (Alegeți din răspunsurile de mai jos răspunsul corect)
$(_Wrong) gerunziu
$(_Correct) verbul la timpul trecut
$(_Wrong) substantiv
$(_Correct) participiu timpului trecut

$(_Wrong) adjectiv
$(_Diagnostics _Correct, _All _Correct , 2) *$(_Reaction)* Absolut corect.
$(_Diagnostics _Correct, _Less 2 *_Wrong*; 1 *_Correct*, 1) *$(_Reaction)* Parțial corect. Ați ales doar un singur răspuns corect.
$(_Diagnostics _Wrong, _More 1 *_Wrong*, 0)
$(_Diagnostics _Wrong, _More 2 *_Wrong*, -1) *$(_Reaction)* Eroare.

Ca și la întrebarea de tip ***Single***, variantele corecte de răspuns sunt notate cu parametrul ***Correct***, iar cele incorecte – ***Wrong***. Totodată punctele meritate pentru răspunsurile corecte sau incorecte sunt indicate în parametrii ***Diagnostics Correct*** și ***Diagnostics Wrong***. Concomitent cu acești parametri se înscrie și parametrul ***Reaction***, responsabil de replicile corespunzătoare. Amănunte despre parametrul ***Diagnostics*** sunt expuse în capitolul corespunzător.

Semantica. Întrebarea de tip ***Multiple*** presupune că pentru răspunsul corect este necesar de ales câteva puncte din meniu. În acest caz parametrii ***Correct*** și ***Wrong*** nu conțin numărul de puncte și nu determină corectitudinea răspunsului, dar servesc pentru marcarea punctelor meniului care sunt incluse în răspunsurile corecte sau incorecte. Dat fiind faptul că studentul poate alege nu toate răspunsurile corecte sau împreună cu cele corecte să aleagă și cele incorecte, atunci problema despre numărul de puncte oferit se rezolvă cu parametrul ***Diagnostics.*** Parametrul ***Reaction*** se înscrie lângă parametrul corespunzător ***Diagnostics***.

Parametrul *Diagnostică (Diagnostics)*

În cazul întrebării de tip ***Multiple*** pentru determinarea corectitudinii și aprecierii răspunsului obținut se utilizează parametrul ***Diagnostics.***

Sintaxa

<p diagnostică> ::= ***$***(*<tipul diagnostice>*, *<raportul răspunsurilor corecte-incorecte>* [, *<puncte pentru răspuns>*])

<tipul diagnostice> ::= ***_Diagnostics_Correct*** |***_Diagnostics_Wrong***

<raportul răspunsurilor corecte-incorecte> ::= *<cantitate>* *<tipul răspunsului>* [***;*** *<cantitate>* *<tipul răspunsului>*]

<cantitate> ::= *<întreg fără semn>* |***_All*** |***_None*** |***_Less*** *<întreg fără semn>* |
_More *<întreg fără semn>*

Dacă *<tipul răspunsului>* este prezent de două ori în construcția programului este evident că odată este de tip ***Correct***, următorul este de tip ***Wrong***.

Exemplu. Să cercetăm exemplul din capitolul precedent:
_Question (_Multiple) Determinați partea de vorbire la care se referă cuvântul “cut”. (Alegeți din răspunsurile indicate mai jos răspunsul corect)
$(_Wrong) gerunziu
$(_Correct) verbul la timpul trecut
$(_Wrong) substantiv

$(_Correct) participiu timpului trecut
$(_Wrong) adjectiv
$(_Diagnostics _Correct, _All _Correct ,* 2) *$(_Reaction) Absolut corect.
$(_Diagnostics _Correct, _Less 2 *_Wrong*; 1 *_Correct,* 1) *$(_Reaction)* Parțial corect. Ați ales doar un singur răspuns corect.
$(_Diagnostics _Wrong, _More 1 *_Wrong,* 0)
$(_Diagnostics _Wrong, _More 2 *_Wrong,* -1) *$(_Reaction)* Eroare.

Dacă au fost alese toate răspunsurile corecte (***_All _Correct***), atunci se va obține două puncte și va apare replica „Absolut corect.”. Răspunsul se consideră corect (***_Diagnostics Correct***). Dacă sunt alese mai puțin de două puncte incorecte (***_Less*** 2 ***_Wrong***) și unul corect (1***_Correct***), atunci numărul de puncte câștigate este 1 și răspunsul se consideră ca fiind corect (***Diagnostics Correct***). Văzând replica „Parțial corect. Ați ales doar un singur răspuns corect.”. În cazul alegerii mai mult de o variantă incorectă (***_More*** 1 ***Wrong***) numărul punctelor obținute este 0 și răspunsul se consideră fiind incorect ***(_Diagnostics Wrong)***. Apare replica „Eroare.”. Dacă sunt alese mai mult de două variante incorecte (***_More*** 2 ***Wrong***) din numărul total de puncte obținut se exclude un punct și răspunsul se consideră incorect (***_Diagnostics _Wrong***). Apare replica „Eroare.”.

Semantica. Deoarece întrebarea de tip ***Multiple*** propune răspunsul care se conține în alegerea din meniu a câtorva variante, clasificarea răspunsurilor corecte sau incorecte se realizează în mod deosebit, cu ajutorul parametrului ***Diagnostics***.

Pentru marcarea răspunsului ca fiind corect se utilizează parametrul ***Diagnostics Correct***, iar pentru marcarea răspunsului ca fiind incorect ***Diagnostics Wrong***. Apoi se arată raportul între răspunsurile corecte și cele incorecte, poate fi indicat exact sau aproximativ numărul variantelor corecte și/sau incorecte permis pentru răspunsul dat. Exemplu, 2 ***Wrong;*** 1 ***Correct*** înseamnă că în răspunsul pentru diagnostica corespunzătoare pot fi alese două variante incorecte și una corectă. Pentru a arăta numărul aproximativ de variante corecte sau incorecte se utilizează cuvintele-cheie ***Less*** (mai puțin), ***More*** (mai mult). Pentru a arăta că toate variantele corecte sau incorecte trebuie selectate se utilizează cuvântul-cheie ***All*** (toate). Indicând că nici una din variantele corecte sau incorecte nu au fost selectate, se utilizează cuvântul cheie ***None*** (nici una). Replica care va apărea pe monitor depinde de parametrul cel mai apropiat corespunzător diagnostice ***Reaction***, cu condiția că el este.

Parametrul *Reacție (Reaction)*

Această situație se utilizează pentru a comenta răspunsul, pentru a da explicații suplimentare, etc.

Sintaxa

<*p reacție*> ::= ***$*(*_Reaction***) <*informație*>

Exemplu

_Question (_Free , 2*)* Traduceți în limba engleză expresia „programe soft”.
$(_Correct, 2, 1*)* software
$(_Reaction) Corect.
$(_Wrong, 2, 0*)* program&
$(_Reaction) Expresia se traduce printr-un singur cuvânt. Acest cuvânt nu este program.

Dacă răspunsul coincide cu șablonul “software”, apare replica „Corect”. Dacă răspunsul coincide cu șablonul “program&” apare replica „Expresia se traduce printr-un singur cuvânt. Acest cuvânt nu este program.”.

Exemplu
_Question (_Single, 2*)* În informatică cuvântul “hardware” înseamnă:
$(_Wrong, 0**)** produse de fierărie *$(_Reaction)* Nu. Acest cuvânt poate fi tradus ca „produse de fierărie”, dar nu în informatică.
$(_Correct, 1*)* aparataj *$(_Reaction)* Corect!
$(_Wrong, 0*)* programe soft
$(_Wrong, -1*)* tare *$(_Reaction)* Eroare.

La alegerea primei variante apare replica „Nu”. Acest cuvânt poate fi tradus ca „produse de fierărie”, dar nu în informatică. La alegerea variantei doi – „Corect!”, iar la alegerea variantei trei sau patru – „Eroare.”.

Exemplu
_Question (_Multiple) Alegeți traduceri corecte ale cuvântului „face”.
$(_Wrong) din față
$(_Correct) față
$(_Wrong) personalitate
$(_Correct) fațadă
$(_Wrong) exterior
$(_Diagnostics _Correct, _All _Correct, 2*) $(_Reaction)* Absolut corect.
$(_Diagnostics _Wrong, _None _Correct, 0*)*
$(_Diagnostics _Wrong, _More 1 *_Wrong,* -1*) $(_Reaction)* Eroare.

Dacă sunt selectate toate variantele corecte (***_All Correct***), apare replica „Absolut corect”. Dacă nu este aleasă nici o variantă corectă (***_None _Correct***) sau va fi aleasă mai mult de o variantă incorectă concomitent cu altele (***_More*** 1 ***_Wrong***), atunci va apărea replica „Eroare”.

Semantica. În cazul de coincidență a șablonului răspunsului (la întrebarea de tip *Free*), de alegere a variantei de răspuns corecte (la întrebarea de tip *Single*) și în caz de îndeplinire a condiției în parametrul ***Diagnostics*** (la întrebarea de tip *Multiple*) poate fi prevăzută o replică oarecare în parametrul ***Reaction***. La întrebarea de tip *Free* replica va fi oferită numai în cazul coincidenței răspunsului cu șablonul. Parametrul ***Reaction*** ce conține replica corespunzătoare se înscrie nemijlocit după parametrii ***Wrong*** sau ***Correct***. La întrebarea de tip *Single* replica este dată de parametrul ***Reaction*** cel mai aproape de variantă aleasă. Se recomandă a fi atent dacă sunt neordonate variantele corecte și incorecte. La întrebarea de tip *Multiple* se oferă următoarea cea mai apropiată după parametrul ***Diagnostics*** replica, dacă condiția parametrului a fost îndeplinită.

Situația Testare (Test)

Aceasta situație instructivă reprezintă în sine totalitatea întrebărilor pentru testare și diagnostică după rezultatul testării.

Sintaxa

<situație test> ::= ***_Test***(*"<denumirea testului>"*) *<informație> <secție de întrebări> <evaluarea rezultatelor testului>*

<denumirea testului> ::= *<text>*

<secție de întrebări> ::= ***_Part***(*<cantitatea întrebărilor>*) *<consecutivitatea întrebărilor>*

<cantitatea întrebărilor> ::= *<număr>* | ***_All***

<consecutivitatea întrebărilor> ::= *<situație întrebare>* [*<consecutivitatea întrebărilor>*]

<evaluarea rezultatelor testului> ::= ***_Evaluate*** *<blocul evaluării rezultatelor testului>* [***_Evaluate***(***_Other***) *<informație>*]

<blocul evaluării rezultatelor testului> ::= (*<diapazonul punctelor>*) *<informație>* [*<blocul evaluării rezultatelor testului>*]

<diapazonul punctelor> ::= *<puncte>* [**.** *<puncte>*] [*<diapazonul punctelor>*]

Exemplu

_Test (Test_Words) Se propune testul de traducere a termenilor utilizați în informatică
_Part (5)
_Question (_Free) Traduceți cuvântul "life" din punct de vedere al tehnicii de calcul.
$(_Edit) SP.,!
*$(_Correct***,** 2**,** 1*)* resurse lonjeviv resurse de durată
$(_Reaction) Corect.
$(_Correct, 3**,** 1*)* resurse lonjeviv
$(_Reaction) Corect.
*$(_Wrong,*2*)* &
$(_Reaction) Eroare.
_Question (_Free) Traduceți expresia „impuls video".
$(_Correct, 2**,** 1*)* videopulse
$(_Reaction) Corect.
*$(_Wrong,*2*)* &
$(_Reaction) Eroare.
_Question (_Free) Traduceți cuvântul „subcircuit"
$(_Correct, 2**,** 1*)* subcircuit
$(_Reaction) Corect.
*$(_Wrong***,**2*)* &
$(_Reaction) Eroare.
_Question (_Free) Traduceți cuvântul "plex" din punct de vedere al tehnicii de calcul.
$(_Edit) SP.,!
$(_Correct, 2**,** 1*)* împletire rețea ramificație de rețea
$(_Reaction) Corect.
$(_Correct, 3**,** 1*)* împletire de rețea
$(_Reaction) Corect.
*$(_Wrong,*2*)* &

$(_Reaction) Eroarea.
_Question (_Free) Traduceți cuvântul “plotter”
$(_Correct, 2**,** 1*)* constructor de grafici
$(_Reaction) Corect.
*$(_Wrong,*2*)* &
$(_Reaction) Eroarea.
_Question (_Free) Traduceți cuvântul “feedback.”
$(_Edit) SP
$(_Correct, 2**,** 1*)* legătură inversă
$(_Reaction) Corect.
*$(_Wrong,*2*)* &
$(_Reaction) Eroare.
_Question (_Free) Traduceți cuvântul “data”.
$(_Edit) SP.,!
$(_Correct, 2**,** 1*)* datele informații datele
$(_Reaction) Corect.
$(_Correct, 3**,** 1*)* datele informații
$(_Reaction) Corect.
*$(_Wrong,*2*)* &
$(_Reaction) Eroare.
_Question (_Free) Traduceți cuvântul “automate”.
$(_Correct, 2**,** 1*)* automatizare
$(_Reaction) Corect.
*$(_Wrong,*2*)* &
$(_Reaction) Eroare.
_Evaluate (5 *)* Nota dvs. *excelent*.
_Evaluate (4 *)* Nota dvs. *bine*.
_Evaluate (2..3 *)* Nota dvs. *satisfăcător*.
_Evaluate(_Other) Nu v-ați isprăvit cu testul. Vă recomand să repetați materialul.

Acest test este compus dintr-o secție (***Part***) în care sunt incluse opt întrebări. Cinci alese aliator vor fi oferite studentului. După testare va fi oferită diagnostica corespunzătoare. În acest test la toate întrebările pentru răspunsul corect se oferă un punct, iar răspunsul incorect – 0 puncte. Parametrul ***Evaluate*** determină că pentru 5 puncte obținute se oferă nota excelentă, pentru 4 puncte– nota bine, iar de la 2 la 3 puncte se oferă notă satisfăcător, în celelalte (***Other***) cazuri apare mesajul „Nu v-ați isprăvit cu testul. Vă recomand să repetați materialul”. Denumirea testului: “***Test_Words***”, sub această denumire rezultatele sunt transferate în baza de date.

Semantica. Situația instructivă ***Test*** este destinată organizării testelor în cursurile de instruire asistate de calculator. Antetul testului este compus din denumirea testului, pentru a avea posibilitatea de păstrat rezultatele testului în baza de date. Mai departe urmează informația instructivă și întrebările. Întrebările sunt grupate în blocuri prin intermediul parametrului ***Part*** în care se determină numărul întrebărilor care vor fi propuse. Întrebările oferite sunt alese aliator din blocul respectiv de întrebări. Dacă în calitate de număr de întrebări este un număr ce coincide cu numărul total de întrebări din bloc, sau este indicat de parametrul ***All***, atunci vor fi date toate întrebările. În cazul dacă numărul

întrebărilor din parametrul ***Part*** este mai mare decât numărul lor real, apare posibilitatea repetării unora de mai multe ori, de aceia trebuie evitate asemenea situații.

Consecutivitatea întrebărilor testului este compusă din situații de întrebări.

Pentru aprecierea rezultatelor testului este utilizat parametrul ***Evaluate*** în care este indicat diapazonul sau numărul exact de puncte și diagnostica corespunzătoare. Pentru punctajul, ce nu se încadrează în nici unul din diapazoane, se utilizează ***Evaluate* (*_Other*)**. Evident că ***Evaluate* (*_Other*)** trebuie să fie ultim în șirul parametrilor de punctaj.

Situația Lucrare de control (Check)

Situația lucrare de control se utilizează pentru desfășurarea lucrărilor de control.

Sintaxa

<*situația lucrare de control*> ::= ***_Check*** (*"*<*denumirea lucrării*>*"* [<*număr de puncte maxim*>, <*numărul metodicii*>]) <*informație*> <*secție de întrebări*>
<*evaluarea rezultatelor lucrării de control*>

<*denumirea lucrării*> ::= ***<Text>***

<*număr de puncte maxim*> ::= <*întreg fără semn*>

<*numărul metodicii*> ***::= 1 | 2 | 3 | 4***

<*evaluarea rezultatelor lucrării de control*> ::= ***_Evaluate*** <*blocul de evaluare a rezultatelor lucrării de control*> [***_Evaluate* (*_Other*** [**,** <*notă*>])<*informație*>]

<*blocul de evaluare a rezultatelor lucrării de control*> ::= ([<*diapazonul punctelor*>, <*nota*>]) <*informația*> [<*blocul de evaluare a rezultatelor lucrării de control*>]

<*notă*> ::= <*literă*> | <*cifră*>

Numărul metodicii corespunde următoarelor metodici:

1 - de cinci baluri (1,…,5),

2 - de cinci baluri USA (A,…,E),

3 - de zece baluri (1,…,10),

4 - de o sută de baluri (1,…,100).

Exemplu

_Check (*"*Verbs. Past&Participle*"* 4**,** 1) Se propune lucrare de control despre formele verbului în limba engleză.
_Part (2 *)*
_Question (_Free) Introduceți timpul trecut al verbului "to teach" (a învăța).
$(_Correct, 2**,** 1*)* taught
$(_Reaction) Corect.
$(_Wrong) &
$(_Reaction) Eroare.
_Question (_Free) Introduceți participiul verbului "to go"(a merge) la timpul trecut .
$(_Correct, 2**,** 1*)* gone

$(_Reaction) Corect.
$(_Wrong) &
$(_Reaction) Eroare.
_Question (_Free) Introduceți timpul trecut al verbului “to do” (a face).
$(_Correct, 2**,** 1*)* did
$(_Reaction) Corect.
$(_Wrong) &
$(_Reaction) Eroare.
_Part (2 *)*
_Question (_Free) Introduceți timpul trecut al verbului “to go” (a merge).
$(_Correct, 2**,** 1*)* went
$(_Reaction) Corect.
$(_Wrong) &
$(_Reaction) Eroare.
_Question (_Free) Introduceți participiul timpului trecut al verbului “to teach” (a învăța).
$(_Correct, 2**,** 1*)* taught
$(_Reaction) Corect.
$(_Wrong, 2*)* &
$(_Reaction) Eroare.
_Question (_Free) Introduceți participiul timpului trecut al verbului “to do” (a face).
$(_Correct, 2**,** 1*)* done
$(_Reaction) Corect.
$(_Wrong, 2*)* &
$(_Reaction) Eroare.
_Evaluate (_Other) Felicitări. V-ați isprăvit de minune în aceasta lucrare de control. Nota dvs. |* Verbs.Past&Participle *|

Încheierea acestei situații a lucrării de control poate fi și în alt mod:
*_Evaluate (*4**,**5*)* Felicitări. V-ați isprăvit de minune cu această lucrare de control. Nota dvs. – 5.
*_Evaluate (*2..3,4*)* Ați terminat lucrarea și nota dvs. este 4 (după modul de apreciere de cinci baluri).
_Evaluate (_Other, 0*)* Ați terminat lucrarea. Cu părere de rău, nu ați acumulat suficiente puncte pentru a lua notă de trecere.

În lucrarea de control propusă, având denumirea „Verbs.Past&Participle”, se propun întrebări din două blocuri. În fiecare din aceste blocuri sunt prevăzute câte trei întrebări. Persoanei testate se va oferi din fiecare bloc câte două întrebări alese aliator. În antetul lucrării de control este determinat că la sfârșit de lucrare pot fi acumulate maximum 4 puncte și nota va fi oferită conform metodicii de cinci baluri. La sfârșitul lucrării cu ajutorul parametrului ***Evaluate*** se determină automat nota care este introdusă în baza de date sub numele „Verbs.Past&Participle”. Nota se calculează rezultând din numărul de puncte acumulate, din punctele maximal posibile și din metodica indicată de apreciere.

În varianta a doua de evaluare va fi ignorat punctajul posibil maxim și metodica aprecierii. Nota va fi calculată exclusiv în conformitate cu faptul, în ce diapazon nimerește numărul de puncte acumulate. Dacă numărul de puncte este patru, atunci nota este 5, dacă este de la trei până la patru, atunci nota este 4, în celelalte cazuri nota este 0.

Semantica. Realizarea lucrării de control coincide cu testarea, cu excepția că lucrarea de control se sfârșește cu note determinate de anumite metodici. În antetul lucrării de control se indică denumirea lucrării care se utilizează pentru introducerea rezultatelor în baza de date. Pe lângă aceasta, poate fi indicat numărul de puncte maximal ce poate fi obținut în cadrul lucrării de control și numărul metodicii, în baza căreia se va efectua aprecierea rezultatelor. Dacă punctajul maxim și numărul metodicii nu este indicat, atunci nota este oferită de profesor de sine stătător utilizând parametrul ***Evaluate*****(**<*diapazonul punctelor*>**,** <*notă*>**)** și ***Evaluate*** **(*_Other,*** [<*notă*>]**)**.

Parametrul *Part* are aceiași semnificație ca și în situația *Test*.

Situațiile auxiliare

Pe lângă situațiile instructive în limbaj sunt prevăzute situații auxiliare, pentru conectarea modulelor exterioare și realizarea ramificațiilor în curs.

Situația Apel la funcție (Execute)

Aceasta situație se utilizează pentru apelul modulelor externe și este destinată pentru extinderea posibilităților cursului, crearea noilor situații didactice realizate de autor.

Sintaxa

<*situația apel la funcție*> ::= ***_Execute*** <*denumirea fișierului*>

Exemplu
_Execute demo.exe
Este apelat și executat programul extern.

Exemplu
_Execute subroutine.mcf
Este apelat și executat fișierul indicat al cursului.

Exemplu
Execute javaclass.class
Este apelat și executat Java-class.

Semantica. În calitate de modul extern poate fi fișierul de tip *.exe (modul executabil), *.mcf (fișierul cursului) sau *.class (Java-class). Modulul extern este considerat ca fiind subprogram al cursului. După execuția modulului, conducerea se întoarce la următoarea situație didactică după ***Execute***.

Situația Ramificare (Branch)

Situația ramificare servește pentru luarea deciziilor în cadrul cursului.

Sintaxa

<*situația de ramificare*> ::= ***_Branch*** <*succesiune de treceri*>

<*succesiune de treceri*> ::= [<*expresie*> **,**] <*denumirea etichetei*> [/<*succesiune de treceri*>]

Exemplu
_Question Răspundeți la întrebarea 1…
…

_Branch Label_1
#Label_x:
_Question Răspundeți la întrebarea 2…
…
#Label_1:
_Question Răspundeți la întrebarea 3…

În acest fragment după prima întrebare se realizează o trecere necondiționată la situația didactică marcată prin etichetă Label_1. Bineînțeles că situația ce urmează după o trecere necondiționată trebuie să fie notată printr-o etichetă.

Exemplu
_Branch R_Answer >= 2*W_Answer, Label_1
Dacă numărul răspunsurilor corecte în capitolul curent este de două ori mai mare decât numărul răspunsurilor incorecte, atunci se realizează o trecere la situația cu etichetă *Label_1*.

Exemplu
_Branch Test_Words < 5, *Chapter1* / Test_Words >= 10, *Chapter2*
Dacă rezultatele testului păstrate în baza de date sub denumirea „Test_Words" este mai mic decât 5, atunci cursul va fi prelungit de la situația didactică marcată *Chapter1*. Dacă este mai mare sau egală cu 10, atunci următoarea situație va fi situația marcată cu etichetă *Chapter2*. În cazul dacă ambele expresii sunt false, se realizează situația următoare după ***Branch***.

Semantica. Această situație auxiliară permite organizarea ramificațiilor în cadrul cursului în conformitate cu următorii indicatori:

- indicatorul de reușită la capitolul curent;
- indicatorul de reușită la întregul curs;
- indicatorul rezultatelor la testări;
- indicatorul rezultatelor la lucrările de control;

Succesiunea de treceri conține expresii care determină condiția trecerii la etichetă corespunzătoare. Expresia se consideră corectă, dacă rezultatul calculărilor este *True* sau diferit de zero. Dacă în succesiunea de treceri se conține mai mult de o condiție de tipul <*expresie*>**,** <*denumirea etichetei*>, atunci trecerea se realizează la prima etichetă, condiția căreia este adevărată. Dacă expresia lipsește, atunci se realizează o trecere necondiționată la eticheta corespunzătoare.

Parametrii cursului

În procesul de instruire apare necesitatea folosirii informației de îndrumare. Pentru scopul acesta se utilizează parametrul cursului *Manual*.

Sintaxa

<*parametrul cursului manual*> ::= ***_Manual_Enable | _Manual_Disable***

Exemplu
_Manual_Enable

Semantica. Parametrul ***Manual*** se utilizează pentru permiterea sau interzicerea utilizării manualului în situații didactice. Acțiunea parametrului ***Manual*** se răsfrânge asupra tuturor situațiilor didactice până la sfârșitul cursului sau până la următoarea folosire a acestui parametru. Implicit parametrul *Manual* este dezactivat (***Manual_Disable***).

Fișierul proiectului cursului

Proiectul cursului este amplasat în fișierul cu extensia "mcp". Fișierul proiectului conține informații despre parametrii interfeței cursului și are următoarea structură:

[*course*]

name = <text>

start up = <denumirea fișierului>

manual = <denumirea fișierului, ce conține manual>

files = <lista de fișiere>

<lista de fișiere>::= *<denumirea fișierului>* | [**;** *<lista de fișiere>*]

[*window*]

height = <număr întreg>

width = <număr întreg>

[*defaults*]

index = true | false

caption index = <text>

caption manual = <text>

caption next = <text>

caption previous = <text>

caption sign off = <text>

caption sign on = <text>

caption login = <text>

caption password = <text>

caption first name = <text>

caption last name = <text>

caption group = <text>

default help = <text>

default exceed = <text>

default undefined answer = <text>

Dacă unul din parametrii enumerați mai sus este omis se recurge la starea implicită.

- *Index* reprezintă prezența sau nu a cuprinsului cursului și, ca urmare, prezența butonului pe ecran „Cuprinsul” În cazul omiterii parametrului cuprinsul se afișează.
- *Manual* include denumirea fișierului ce conține materialul didactic care va fi utilizat în procesul studierii cursului. Manualul poate fi sub formă de fișier .txt, .mcf, .exe. În cazul absenței parametrului dat, materialul didactic nu se utilizează și, ca urmare, butonul „Manual” lipsește.
- *Window Height* și *Width* determină dimensiunile inițiale ale ferestrei cursului. În cazul absenței, fereastra cursului va ocupa toată suprafața de lucru a ecranului.
- *Caption Index* determină textul ce va apare pe butonul Cuprinsul „Sumarul”.
- *Caption Manual* determină textul ce va apare pe butonul „Manualul”.
- *Caption Next* determină textul ce va apare pe butonul „Următorul”.
- *Caption Previous* determină textul ce va apare pe butonul „Înapoi”.
- *Caption Sign Off* determină textul ce va apare pe butonul „Deconectarea”.
- *Caption Sign On* determină textul ce va apare pe butonul „Conectarea”.
- *Caption Login* determină textul ce va apare în fața câmpului „Login” în fereastra de identificare a studentului.
- *Caption Password* determină textul ce va apare în fața câmpului „Parola” în fereastra de identificare a studentului.
- *Caption First Name* determină textul ce va apare în fața câmpului „Numele” în fereastra de identificare a studentului.
- *Caption Last Name* determină textul ce va apare în fața câmpului „Prenume” în fereastra de identificare a studentului.
- *Caption Group* determină textul ce va apare în fața câmpului „Grupa” în fereastra de identificare a studentului.
- *Default Help* mesajul standard de ajutor.
- *Default EXceed* – mesajul standard oferit în cazul depășirii numărului de încercări de răspunsuri la întrebări.
- *Default Undefined Answer* – mesajul standard oferit în cazul răspunsului neprevăzut.

Exemplu

[*course*]
name = English Verbs
start up = EnglishStartUp.moc
files = EnglishF1.moc;
EnglishF2.moc;
EnglishF3.moc;
EnglishF4.moc
manual= EnglishBook.txt
[*window*]
height = -1
width = 700
[*defaults*]
index=true
default help = Ajutorul nu se oferă.
default exceed = Ați depășit numărul posibil de încercări.
default undefined answer = Răspunsul dvs este neclar.

Textul „English Verbs" din parametrul *Name* va apărea ca denumire a tuturor ferestrelor cursului: ferestrei principale, ferestrei ajutorului etc. Dimensiunile câmpului principal al cursului sunt determinate de parametrii *Width* = 700 și *Height* = -1 (-1 în calitate de valoare a parametrului sau absența valorii reprezintă valoarea maximă admisibilă). Frontispiciul cursului se află în fișierul EnglishStartUp.moc. Lecțiile cursului se află în fișierele EnglishF1.moc, EnglishF2.moc, EnglishF3.moc și EnglishF4.moc. Manualul cursului se află în fișierul EnglishBook.txt. Valoarea *true* a parametrului *Index* arată că trebuie de colectat și de arătat cuprinsul cursului după prefața. În secția *Defaults* sunt, de asemenea, definite textele replicilor implicite corespunzătoare.

3.7. Șabloanele răspunsurilor studentului

Șabloanele permit analiza răspunsurilor ce sunt date de student într-un mod destul de comod și efectiv.

Analiza în baza unui singur șablon

În acest caz șablonul este considerat ca un tot întreg, chiar dacă el constă din mai multe cuvinte. Spațiile (în caz că sunt), care despart cuvintele, reprezintă caractere semnificative. În acest caz se cere coincidența absolută a răspunsului cu șablonul declarat.

Șablonul: *drum de țară*

Răspunsul corect: *drum de țară*
Răspunsul incorect: *drum di țară*

Analiza în baza unui șablon cu caracterele de prelucrare parțială

În acest caz răspunsul studentului se compară cu un șablon special declarat. La declararea a unui astfel de șablon se utilizează caracterele speciale *** % \ [] # @,** precum și ^. Să examinăm destinația caracterelor speciale:

1. Caracterul ***** indică că în locul acesta al răspunsului poate fi orice semn.

Șablonul: *ţar******	Răspunsurile corecte: *ţară, ţara* etc. deoarece caracterul de pe al patrulea loc nu este semnificativ.

2. Caracterul **%** indică că caracterul ce urmează după el poate să lipsească.

Șablonul: *%02%05*	Răspunsurile corecte*: 0025, 205, 25, 000025, 20005,* etc.
Șablonul: *%01*	Răspunsurile corecte: *001, 1,0000001* etc.

3. Semnele \\ înseamnă că caracterele, ce se află între aceste semne pot să lipsească sau să fie prezente parțial.

Șablonul: *dir\ectory*	Răspunsurile corecte: *dir, directory* etc.

4. Caracterele **[ ]**. În paranteze pătrate se înscriu caracterele separate, șiruri de semne și intervale de caractere. Ultimele sunt permise în cazul dacă codurile caracterelor reprezintă un șir consecutiv. Caracterele separate, șirurile de semne și intervalele de caractere sunt separate prin virgule. Intervalele se indică prin semnul -(minus), de exemplu **a-e** (literele de la **a** la **e**) sau **0-9** (cifrele de la **0** la **9**). Selectarea informației între **[ ]** se face de la stânga la dreapta și este oprită dacă este găsită coincidența dorită. Dacă în poziția curentă în răspuns nu se întâlnește nici unul din grupurile de caractere declarate, răspunsul este considerat incorect.

Șablonul: *Anul 19**[**80, 81, 90, 91**]***	Răspunsurile corecte: *Anul 1980, Anul 1981, Anul 1990, Anul 1991*
Șablonul: ***[**0-9, A-Z, a-z**]***	Ca răspuns corect poate servi orice cifră sau orice caracter latin.
Șablonul: *IBM PC **[**XT, AT**]***	Răspunsurile corecte: *IBM PC XT, IBM PC AT*. Răspunsul *IBM PC* va fi considerat incorect.
Șablonul: *Păd**ur[**e,i,ile**]***	Răspunsurile corecte: *Pădure, Păduri, Pădurile*.

5. Caracterele **#** și **@** sunt folosite pentru a arăta numărului de repetări posibile a elementelor șablonului, înscrise în paranteze pătrate. Caracterul **#** indică numărul maxim de repetări, iar caracterul **@** – numărul exact de repetări. Numărul de repetări se scrie direct după caracterul # sau **@** în formă de constantă întreagă.

Șablonul: *#3[a]*	Răspunsurile corecte: *a, aa, aaa*.
Șablonul: ***@**2[ma]*	Răspunsul corect: *mama*
Șablonul: ***@**2[1-3]*	Răspunsurile corecte: *12, 13, 11, , 23, 21, 22, 31, 32, 33.*
Șablonul: *#2[1-3]*	Răspunsurile corecte: *1, 2, 3, 12, 13, 11, 23, 21, 22, 31, 32, 33.*
Șablonul: *[A-Z,a-z]#5[A-Z,a-z,0-9]*	Răspunsurile corecte: *X1, ZET,Alfa, Omega2* etc.

Ultimul exemplu ne indică cum se poate efectua verificarea apartenenței răspunsului unei mulțimi de identificatori ai limbajului de programare Fortran (o consecutivitate de caractere cu lungime mai mică de 6 ce se începe cu o literă și constă din litere și cifre).

6. Caracterul **&** indică ignorarea unor caractere din răspuns. Dacă în șablon a fost folosit caracterul **&**, atunci în analiza răspunsului se vor ignora toate caracterele, până nu va fi întâlnit caracterul indicat în șablon după semnul **&**. Dacă în șablon după semnul **&** nu urmează nimic, atunci în răspuns se ignorează toate caracterele neanalizate rămase.

Șablonul: *pădur***&**	Răspunsurile corecte: *pădure, păduri, pădurar* etc. Vor fi considerate corecte toate răspunsurile ce se încep cu *pădur*
Șablonul: **&** *complex***&**	Răspunsurile corecte: *numere complexe, număr complex* etc. Este considerat corect orice răspuns ce conține cel puțin două cuvinte consecutive, cel de al doilea începându-se cu cuvântul complex.
Șablonul: *if (***&***) then* **&** *else* **&***;*	Un răspuns corect este : *if (a) then b else c;* sau *if (a>5) then b=d*c else c:=c+1;*

Astfel caracterele ce se află în răspuns pe locul caracterului **&** se consideră nesemnificative.

7. Caracterul ^ este folosit pentru cercetarea prezenței în răspuns a unuia din semnele & *** % @ # [ ]**, care sunt din categoria caracterelor speciale. Pentru aceasta, în șablonul corespunzător, în fața unui astfel de caracter, se înscrie semnul ^.

Șablonul:^&^&	Răspunsul corect: *&&*
Șablonul: **&** *semn***&** ^* *și* ^^	Un răspuns corect poate fi propoziția: *Unul din semnele* ***** *sau* ^
Șablonul: ^*[Alfa^]*	Răspunsul corect: *[Alfa]*
Șablonul: *50^%*	Răspunsul corect: *50%*

Analiza în baza a mai multor șabloane (șablonul multiplu)

În cazul acesta răspunsul se analizează cu ajutorul a mai multor șabloane. Fiecare șablon în parte se definește așa cum e descris mai sus – în cazul unui singur șablon. Analiza se efectuează până când răspunsul nu coincide cu un șablon. Șabloanele se despart prin spații.

Șablonul: **&***Salut hallo***&**	Răspunsurile corecte în cazul acesta: *salut* sau *hallo* sau *te salut* sau *hallo fellow* ș.a.m.d.

La definirea a mai multor șabloane se pot folosi toate facilitățile descrise mai sus ca și în cazul unui singur șablon, cu deosebirea că într-un șablon se scriu mai multe șabloane speciale.

Orice răspuns va fi corect, dacă el conține minimum două șabloane speciale. Folosind șabloanele, autorul cursului are posibilitatea de a analiza destul de exact răspunsurile studentului.

3.8. Concluzii la capitolul 3

În acest capitol au fost efectuate următoarele:

1. Este descrisă strict sintaxa limbajului descrierii cursurilor de instruire cu ajutorul BNF. În anexă sunt arătate diagramele sintactice ale acestui limbaj;

2. Este definită semantica limbajului și sunt aduse exemple care ilustrează utilizarea limbajului;
3. Sunt propuse metode de analiză a răspunsurilor textuale libere ale studenților;
4. Limbajul expus în capitolul prezent acoperă toată mulțimea situațiilor instructive evidențiate în capitolul precedent;
5. Limbajul poate fi clasificat ca LSD – *limbajul specific domeniului* (*eng*. DSL – ***d****omain* ***s****pecific* ***l****anguage*).[82, 83]

4. PROIECTAREA PLATFORMEI PENTRU CREAREA CURSURILOR DE INSTRUIRE ASISTATE DE CALCULATOR

Limbajul MoCo permite crearea cursurilor asistate de calculator nu numai programatorilor, ci și specialiștilor din alte domenii. Totuși programarea cere eforturilor considerabile. Existența mediului de dezvoltare al programelor specializat favorizează crearea mai rapidă produselor program.

Problema poate fi rezolvată pe două căi:

- crearea unei platforme de dezvoltare cu programare redusă (eng. *low-code development platform)*;
- crearea unei platforme de dezvoltare fără programare (eng. *no-code development platform)*;

Platforma cu programare redusă necesită analiza lexicală, sintactică și semantică și compilatorul într-un limbaj intermediar. Platforma fără programare produce codul direct în limbajul intermediar pentru interpretarea ulterioară a codului de către interpretorul. În capitolele următoare va fi examinate aceste soluții.

4.1. Analiza lexicală, sintactică și semantică a limbajului MoCo

Limbajul de descriere a cursurilor asistate de calculator MoCo, propus în aceasta lucrare este definit de gramatica **G** = (***N, ∑, P, S*****),** unde:

- ***N*** – mulțimea simbolurilor neterminale (sau ajutătoare) utilizate în cadrul descrierii limbajului;
- ***∑*** – mulțimea simbolurilor terminale care este compusă din cuvintele-cheie ale limbajului, texte, constante;
- ***S*** – axioma gramaticii, simbolul evidențiat din mulțimea ***N*** care este numit simbol inițial.
- ***P*** – mulțimea finită de reguli ce descriu limbajul. Regulile, sub forma normală Backus-Naur, au fost descrise în capitolul 3.

Gramatica dată este independentă de context [84, 85]. Ca urmare, există un automat cu memorie stivă ce determină limbajul cercetat. Majoritatea automatelor cu memorie stivă necesită un timp mare pentru analiză și anume: în cazul prelucrării șirului de intrare de lungime ***n*** se consumă timp de recunoaștere n^3. Dar există o clasă mai restrânsă de gramatici independente de context ***LL***, pentru care se poate construi o mașină de traducere predictivă (MTP), pentru care timpul de analiza depinde liniar de lungimea șirului inițial.

Printre clasa gramaticilor ***LL*** [86, 87] cele mai simple și efective în utilizare sunt gramaticile ***LL(1).*** Pentru gramaticile ***LL(1)*** are loc următoarea afirmație: gramatica independentă de context **G = (*N, ∑, P, S***) este gramatică ***LL(1)*** atunci și numai atunci dacă pentru două reguli diferite $A \rightarrow \beta$ și $A \rightarrow \gamma$ intersecția $FIRST_1(\beta\ FOLLOW_1(A)) \cap FIRST_1(\gamma\ FOLLOW_1(A)) = \varnothing$ pentru orice $A \in N$.

Analiza gramaticelor ***LL(k)*** se realizează mai efectiv cu ajutorul *algoritmului k-predictiv* de analiză. Algoritmul *k*-predictiv A pentru gramatica independentă de context **G** = (***N, ∑, P, S***) are la bază șirul de intrare, memorie stivă și șirul de ieșire. Acest algoritm încearcă să se urmărească procesul de derivare stângă a șirului inițial. La citirea șirului inițial ce se analizează, scris pe bandă de intrare, capul de citire poate „privi înainte" cu *k* simboluri următoare. Mai detailat lucrul acestui algoritm va fi descris în capitolul 4.3 „Analiza sintactică".

Reieșind din necesitatea construirii unui translator efectiv, gramatica inițială a limbajului de descriere a cursurilor asistate de calculator MoCo, descrisă prin regulile în forma normală Backus-Naur (capitolul 3), a fost adusă la gramatica ce satisface condiției ***LL(1)***.

Pentru modificarea gramaticii inițiale **G** au fost folosite transformări echivalente, bazate pe algoritmul de eliminare a recursiei de stânga și algoritmul de factorizare stânga.

În rezultat a fost obținută gramatica **G'** = (***N', Σ, P', S***), unde mulțimea simbolurilor terminale $\sum$ și axioma ***S*** coincid cu cele din gramatica **G**, iar mulțimea producțiilor ***P'*** și mulțimea simbolurilor neterminale ***N'*** sunt expuse integral în anexa 4.

Pentru gramatica obținută **G'** au fost construite mulțimile ***FIRST***, ***FOLLOW*** și σ (anexa 5). Evident este faptul că pentru toate simbolurile neterminale ***A*** are loc egalitatea

$$\boldsymbol{FIRST}_1(\beta\,\boldsymbol{FOLLOW}_1(\boldsymbol{A})) \cap \boldsymbol{FIRST}_1(\gamma\,\boldsymbol{FOLLOW}_1(\boldsymbol{A})) = \varnothing,$$

unde $\boldsymbol{A \rightarrow \beta}$ și $\boldsymbol{A \rightarrow \gamma}$ două producții diferite ale gramaticii **G'**. Prin urmare, conform teoremei, gramatica **G'** obținută cu ajutorul modificărilor echivalente este gramatică ***LL(1).***

În continuare prin gramatica ce definește limbajul de descriere a cursurilor MoCo se va subînțelege anume gramatica modificată **G'**.

4.2. Etapele de translare

Programul sursă al cursului, scris în limbajul de programare MoCo, se prelucrează de analizor; care îl consideră ca o consecutivitate de simboluri. În rezultat analizorul transformă aceasta consecutivitate într-o succesiune de octeți care reprezintă codul de interpretare. În acest caz pot fi evidențiate următoarele faze ale translatorului [88 - 90]:

1. Analiza lexicală.
2. Analiza sintactică.
3. Analiza semantică sau generarea codului intermediar.

Translatorul este dat ca mulțimea perechilor (*x*, *y*), unde *x* este programul sursă în *limbajul inițial*, *y* este *programul obiect*. Problema principală constă în alcătuirea algoritmului efectiv care pentru datele de intrare *x* se construiește ieșirea *y*. Mulțimea perechilor (*x*, *y*) se numește *traducere*.

Dacă *x* este consecutivitatea simbolurilor alfabetului **Σ***, iar *y* este consecutivitatea simbolurilor alfabetului **Γ*,** atunci traducerea este imaginea mulțimii **Σ*** în **Γ***. În acest caz ca program obiect, ce reprezintă rezultatul translatorului, se consideră un alt program scris în limbaj intern special. Acest program „intern" pe urmă se execută de *interpretor* [91, 92].

La realizarea traducerii mai comod se consideră traducerea ca fiind compoziția a două procese mai simple. Prima din ele, *analiza sintactică,* asociază cu fiecare intrare (program în limbaj sursă) o structură arborescentă, care servește ca argument al procesului doi, numit *analiza semantică.* Structura arborelui sintactic reflectă regulile sintactice ale limbajului de programare, în care este scris programul sursă. Într-un arbore sintactic nodurile interioare preponderent corespund operațiilor, iar frunzele reprezintă operanzi care constă din indicele ce indică la tabele de constante

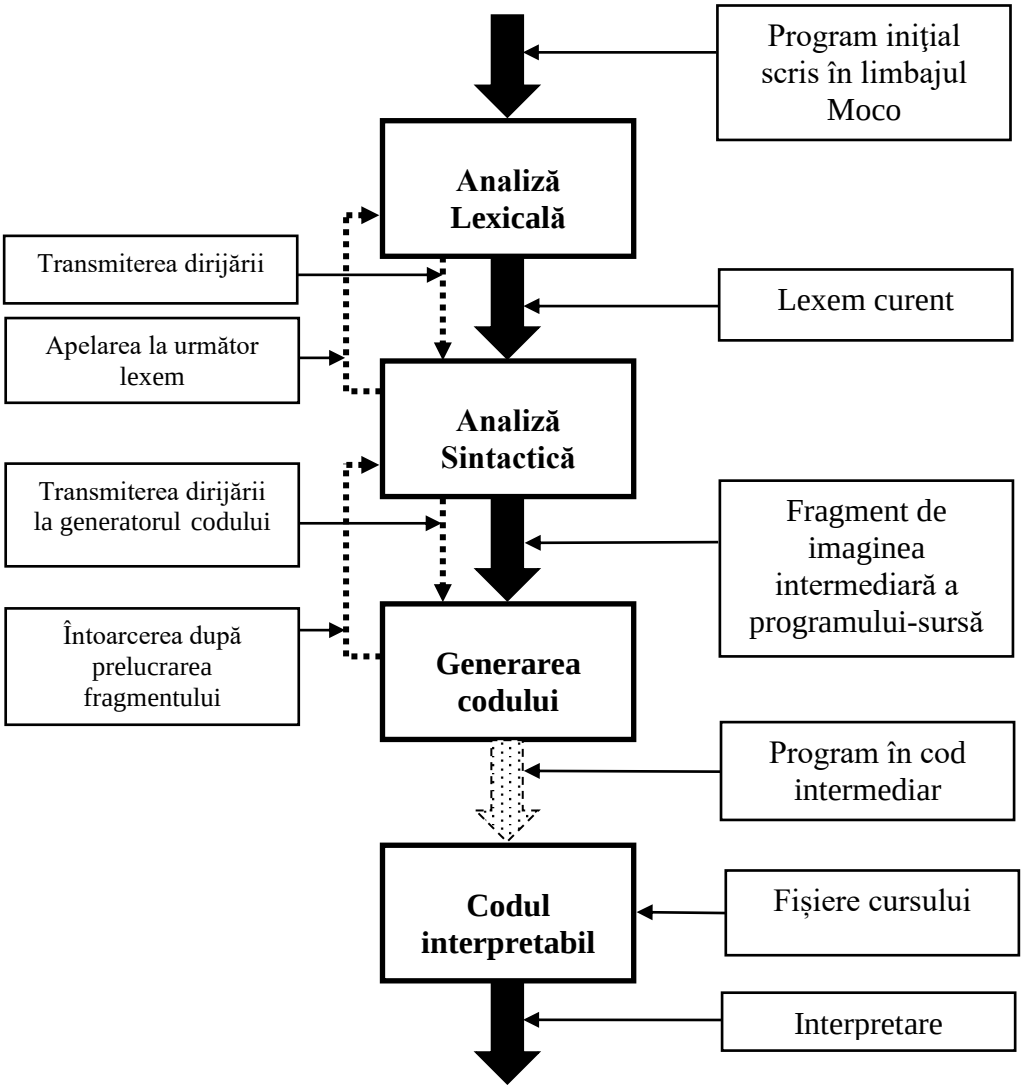


Figura 4.1. Schema translatorului limbajului MoCo.

și texte. Programul scris în limbajul sursă de programare poate fi împărțit în componente sintactice în conformitate cu regulile sintactice ce dirijează acest limbaj.

Procesul de căutare a structurii sintactice a propoziției inițiale se numește *analiza sintactică*. A doua parte a traducerii se numește *analiza semantică* și constă în transformarea intrării structurate într-o ieșire structurată care reprezintă un program în limbajul interpretorului.

În figură 4.1 se expune modelul conceptual de translator al limbajului MoCo. Aici faza de analiza sintactică este urmată de faza de generare a codului intermediar, iar în practica aceste două faze sunt unite într-o singură faza.

De menționat că translatorul cercetat realizează faza de translare în două etape. Prima etapă corespunde fazei de analiză lexicală, etapa a doua – fazei unite de analiză sintactică și semantică.

4.3. Analiza lexicală

Prima fază a translării este analiza lexicală. Consecutivitatea de simboluri a programului sursă în limbajul MoCo servește drept date de intrare a translatorului. Alfabetul de intrare al limbajului MoCo conține simbolurile următoare:

A B C ... Z a b c … z 0 1 2 ... 9 spațiu **_ $ # = + - * / .. ( ) , . ; : ‘ & | [ ] > < ? % ! @ ~**

În program anumite combinații de simboluri adeseori se consideră ca obiecte unice. Ca exemple tipice pot fi indicate următoarele:

1. Succesiunea ce constă din una sau mai multe spații se consideră ca un singur spațiu.
2. În limbaj sunt cuvinte-cheie cum ar fi: _Lesson, _Information, _Question, _Test, _Help etc, fiecare din acestea se consideră ca fiind un singur simbol.
3. Fiecare consecutivitate de simboluri ce reprezintă o constanta numerică se consideră ca fiind un element al textului.
4. Identificatorii folosiți în calitate de denumire a câmpurilor din baza de date, etichete, etc., la fel sunt considerate ca unități lexicale ale limbajului de programare.

Funcționarea analizatorului lexical constă în gruparea anumitor simboluri terminale în obiecte sintactice unice, numite *lexeme*. *Lexem* reprezintă consecutivitatea simbolurilor terminale cu care este asociată o structură lexicală, ce constă din perechea ce are forma (*tipul lexemului, datele*). Prima din componentele perechii reprezintă categoria sintactică cum ar fi „constantă” sau „text”, iar componenta secundă reprezintă un indice la adresa celulei unde se păstrează informația despre lexemul concret. Pentru orice limbaj numărul tipurilor de lexeme este finit.

Deci analizator lexical reprezintă un translator, în calitate de intrare a căruia servește programul inițial în formă de consecutivitate de simboluri, iar în calitate de ieșire se obține o consecutivitate de lexeme. Aceste date de ieșire servesc drept date de intrare pentru analizorul sintactic.

Cercetăm următorul operator al limbajului MoCo:

_Information Hello, World! |* _Audio = hello.wav *|

La etapa de analiză lexicală va fi evidențiat că _Information și _Audio sunt lexeme de tip cuvinte-cheie; „Hello, World!” și „hello.wav” reprezintă lexeme de tip text, iar semnele =, |* și *| sunt lexeme propriu zise. Toate elementele de tip text este necesar să fie transformate în lexeme de tip <Word>. Se consideră că componenta secundară a lexemului reprezintă un indice al unui element din tabel, ce conține un text concret. Prima componentă se folosește de analizorul sintactic, iar cea de a doua se va utiliza la etapa generării codului intermediar.

Astfel, în calitate de date de ieșire a analizatorului lexical, care analizează consecutivitatea inițială indicată mai sus, va fi consecutivitate de lexeme:

<*Information*> <*Word*>$_1$ <|*> <*Audio*> <=> <*Word*>$_2$ <*|>

Componenta secundară a lexemului (indicele la date) este indicată cu indicele de jos. Simbolurile = , |*, *|, Information, Audio sunt considerate ca lexeme tipul cărora este definit de ele însăși. Ele nu au datele asociate, și, prin urmare, nu au indici.

La analiza lexicală se consideră că lexemele, alcătuite din mai mult de un simbol, sunt izolate cu ajutorul simbolurilor ce servesc drept lexeme. Problema legată de faptul că unele simboluri pot fi atât lexeme independente, cât și părți componente ale altor lexeme (de exemplu, >, =, și >=), se rezolvă prin devansarea unui pas, totodată verificându-se posibilitatea reunirii lexemului precedent cu lexemul curent. Decizia luată despre această reunire reiese din regulile gramaticii și din posibilitatea apariției consecutive a două semne ca niște lexeme separate. Determinarea apartenenței semnelor la text se bazează pe același principiu de devansare cu un pas.

În rezultatul analizei lexicale informația despre unele lexeme, cum ar fi textul sau constante, se acumulează în tabele. Aceste tabele trebuie să asigure:

- adăugarea rapidă a noilor obiecte și a datelor despre ele,
- căutarea rapidă a informației referitoare la obiectul dat.

În lucrul cu tabelele este posibilă utilizarea memoriei dinamice sub formă de stivă sau a unei părți din memorie alocate anterior, cu folosirea metodei funcțiilor Hash [93].

Deoarece nu apare necesitatea urmăririi tuturor intrărilor a unui obiect oarecare, fie text sau constantă din programul sursă., pentru lucrul cu tabelele în cadrul acestui translator se utilizează memoria stivă. Este suficient de înscris un obiect curent în vârful stivei fără verificarea conținutului tabelelor. Dimensiunea stivei nu este determinată anterior, ci depinde doar de numărul obiectelor întâlnite, ceea ce ne permite o utilizare mai flexibilă a resurselor memoriei. S-ar putea de menționat și faptul că în cazul accesului la o anumită celulă se utilizează adresarea directă. Utilizând memoria dinamică în cazul tabelelor putem obține o eficacitate mărită de lucru cu memoria dinamică

4.4. Analiza sintactică și analiza semantică

După cum a fost menționat ieșirea analizatorului lexical este o consecutivitate de lexeme. Ea formează intrarea în analizorul sintactic, ce cercetează primele componente ale lexemelor. Informația despre fiecare lexem (componenta a doua din perechea (*tipul lexemului, datele*)) se utilizează la etapă următoare, pentru generarea codului de interpretare.

Analiza sintactică reprezintă un proces în care se cercetează consecutivitatea de lexeme și se determină dacă ea satisface condițiilor de structură, clar formulate în definirea sintaxei limbajului. Datele de ieșire a analizatorului este un arbore, ce reprezintă structura sintactică caracteristică programului sursă.

Cum a fost menționat mai sus, gramatica studiată reprezintă gramatica ***LL(1)***. Pentru analiza sintactică, în cazul dat, a fost utilizat analizatorul 1-predictiv stâng. Acest algoritm utilizează consecutivitatea de lexeme de intrare, memoria stivă și consecutivitatea de ieșire (Figura 4.1).

În timpul citirii consecutivității analizate situate la intrare, capul de citire poate „vizualiza" 1 simbol următor [93, 94]. Acest simbol se numește *avan simbol*. În figura 4.1 în calitate de avan simbol servește simbolul *u* al consecutivității de intrare *wux.*

Stiva conține consecutivitatea $X\alpha\nabla$, unde $X\alpha$ este consecutivitatea simbolurilor stivei, în care ∇ este un simbol special, utilizat în calitate de simbol de marcare a bazei stivei, și X — simbolul din vârful stivei. Alfabetul simbolurilor stivei (fără ∇) se notează prin $\mathbf{\Gamma}$.

Banda de ieșire conține consecutivitatea π, alcătuită din numerele ce corespund producțiilor.

Configurația algoritmului predictiv se definește prin tripletul (x, $X\alpha$, π), unde: x este partea neutilizată a consecutivității inițiale de intrare, $X\alpha$ – consecutivitatea în stivă (X este simbolul din vârful stivei), π – consecutivitate pe banda de ieșire. De exemplu, în Figura 4.1 este reprezentată următoarea configurație $(ux, X\alpha\nabla, \pi)$.

Funcționarea algoritmului 1-predictiv A este condusă de *tabelul de dirijare* MTP, ce definește imaginea mulțimii $(\Gamma \cup \{\nabla\}) \times \sum^{*k}$ în mulțimea ce constă din

- (β, i), unde $\beta \in \Gamma^*$, iar i numărul producției (se presupune că β reprezintă partea dreaptă a producției i a gramaticii),
- avans,
- acceptare,
- eroare.

La fiecare tact se determină inițial avan simbolul *u* și simbolul din vârful stivei *X*. Prin urmare, pentru determinarea faptului ce trebuie de făcut concret se studiază elementul MTP*(X, u) al* tabelei de dirijare.

Fie u = $FIRST_k(x)$.

1. *(x, Xα, π)* |— *(x,* βα, π *i*), dacă MTP(*X,* u) = (β, *i*). În acest cazi simbolul superior al stivei *X* se înlocuiește cu șirul β∈ Γ* și la șirul de ieșire se adaugă numărul producției *i*. Capul de citire nu se deplasează.
2. *(x, aα, π)* |— *(x′,* α, π*)*, dacă MTP*(a, u) = avans* și *x = ax′*. Dacă simbolul superior al stivei coincide cu simbolul de intrare curent (avan simbol), el este exclus din stivă și capul de citire se deplasează cu un simbol spre dreapta.
3. Dacă algoritmul ajunge la configurația (ε, ∇, π), lucrul se termină și șirul de ieșire π se numește *derivarea* consecutivității de intrare inițiale. Se admite că permanent MTP(∇, ε) = *acceptare*. Configurația (ε, ∇, π) se numește *de acceptare*.
4. Dacă algoritmul ajunge la configurația *(x, Xα, π)* și MTP*(X, u) = eroare*, atunci analiza se întrerupe și se comunică despre eroare. Configurația *(x, X*α, π) se numește *de eroare.*

Configurația (ω, X_0 ∇, ε), unde ω ∈ ∑* este șirul analizat, iar X_0 este simbolul inițial evidențiat, se numește *configurație inițială.* Dacă (ω, X_0 ∇, ε) |—* (ε, ∇, π), atunci aceasta se va nota prin *A* (ω) = π și se numește derivare π a algoritmului *A* pentru intrarea ω. Dacă din configurația (ω, X_0 ∇, ε) nu se atinge configurația de acceptare, atunci se consideră că valoarea *A(ω)* este nedeterminată. Traducerea determinată de algoritmul *A* se numește mulțimea τ(*A*)={(ω,π) | *A*(ω) = π}.

Nucleul algoritmului 1-predictiv de analiză reprezintă tabelul de dirijare MTP. Pentru alcătuirea tabelului de dirijare a fost utilizat următorul algoritm:

1. Dacă A → α este producția cu numărul *i*, atunci MTP(A, a) = (α, *i*) pentru ∀a≠ε, ce aparțin $FIRST_1$ (α). Dacă ε ∈ $FIRST_1$ (α), atunci MTP(A, b)=(α,*i*) pentru ∀ b∈$FOLLOW_1$(A) .
2. MTP(a, a) = avans pentru ∀a ∈ ∑.
3. MTP(∇, ε) = acceptare.
4. În celelalte cazuri MTP(X, a) = eroare pentru X∈N∪∑∪{∇} și a∈ ∑ ∪ {ε}.

Drept exemplu de funcționare a algoritmului predictiv descris poate servi analiza consecutivității de intrare, obținute la etapa de analiză lexicală a programului inițial:

<Lesson> <(> <Number> <”> <Word> <”> <[> <Number> <]> <)> <EndOfLesson>

Partea tabelului de dirijare MTP ce se folosește în exemplul dat este expusă în Anexa 7.

[<Lesson> <(> <Number> <”> <Word> <”> <[> <Number> <]> <)> <EndOfLesson>∇; KursFile ∇; ε] ⊢—
⊢— [<Lesson> <(> <Number> <”> <Word> <”> <[> <Number> <]> <)> <EndOfLesson> ∇; L_1 L_2 ∇; 0]
⊢— [<Lesson> <(> <Number> <”> <Word> <”> <[> <Number> <]> <)> <EndOfLesson> ∇; M1 Lesson I_4 L_4 EndL L_2 ∇; 0, 30]
⊢— [<Lesson> <(> <Number> <”> <Word> <”> <[> <Number> <]> <)> <EndOfLesson> ∇; Lesson I_4 L_4 EndL L_2 ∇; 0, 30, 48]
⊢— [<Lesson> <(> <Number> <”> <Word> <”> <[> <Number> <]> <)> <EndOfLesson> ∇; <Lesson> I_4 L_4 EndL L_2 ∇; 0, 30, 48, 31]
⊢— [<(> <Number> <”> <Word> <”> <[> <Number> <]> <)> <EndOfLesson> ∇; I_4 L_4 EndL L_2 ∇; 0, 30, 48, 31]
⊢— [<(> <Number> <”> <Word> <”> <[> <Number> <]> <)> <EndOfLesson> v; <(> N_2 <”> T_1 <”> V_2 <)>L_4 EndL L_2 ∇; 0, 30, 48, 31, 35]
⊢— [<Number> <”> <Word> <”> <[> <Number> <]> <)> <EndOfLesson> ∇; N_2 <”> T_1 <”> V_2 <)>L_4 EndL L_2 ∇; 0, 30, 48, 31, 35]
⊢— [<Number> <”> <Word> <”> <[> <Number> <]> <)> <EndOfLesson> ∇; <Number> X_7 <”> T_1 <”> V_2 <)>L_4 EndL L_2 ∇; 0, 30, 48, 31, 35, 36]
⊢— [<”> <Word> <”> <[> <Number> <]> <)> <EndOfLesson> ∇; X_7 <”> T_1 <”> V_2 <)>L_4 EndL L_2 ∇; 0, 30, 48, 31, 35, 36]
⊢— [<”> <Word> <”> <[> <Number> <]> <)> <EndOfLesson> ∇; <”> T_1 <”> V_2 <)>L_4 EndL L_2 ∇; 0, 30, 48, 31, 35, 36, 38]
⊢— [<Word> <”> <[> <Number> <]> <)> <EndOfLesson> ∇; T_1 <”> V_2 <)>L_4 EndL L_2 ∇; 0, 30, 48, 31, 35, 36, 38]
⊢— [<Word> <”> <[> <Number> <]> <)> <EndOfLesson> ∇; X_3 X_2 <”> V_2 <)>L_4 EndL L_2 ∇; 0, 30, 48, 31, 35, 36, 38, 16]
⊢— [<Word> <”> <[> <Number> <]> <)> <EndOfLesson> ∇; <Word> X_2 <”> V_2 <)>L_4 EndL L_2 ∇; 0, 30, 48, 31, 35, 36, 38, 16, 13]
⊢— [<”> <[> <Number> <]> <)> <EndOfLesson> ∇; X_2 <”> V_2 <)>L_4 EndL L_2 ∇; 0, 30, 48, 31, 35, 36, 38, 16, 13]
⊢— [<”> <[> <Number> <]> <)> <EndOfLesson> ∇; <”> V_2 <)>L_4 EndL L_2 ∇; 0, 30, 48, 31, 35, 36, 38, 16, 13, 18]
⊢— [<[> <Number> <]> <)> <EndOfLesson> ∇; V_2 <)>L_4 EndL L_2 ∇; 0, 30, 48, 31, 35, 36, 38, 16, 13, 18]
⊢— [<[> <Number> <]> <)> <EndOfLesson> ∇; V_4 X_{20} <)>L_4 EndL L_2 ∇; 0, 30, 48, 31, 35, 36, 38, 16, 13, 18, 215]
⊢— [<[> <Number> <]> <)> <EndOfLesson> ∇; T_4 V_5 X_{20} <)>L_4 EndL L_2 ∇; 0, 30, 48, 31, 35, 36, 38, 16, 13, 18, 215, 218]
⊢— [<[> <Number> <]> <)> <EndOfLesson> ∇; F_2 T_5 V_5 X_{20} <)>L_4 EndL L_2 ∇; 0, 30, 48, 31, 35, 36, 38, 16, 13, 18, 215, 218, 232]
⊢— [<[> <Number> <]> <)> <EndOfLesson> ∇; <[> D_4 <]> T_5 V_5 X_{20} <)>L_4 EndL L_2 ∇; 0, 30, 48, 31, 35, 36, 38, 16, 13, 18, 215, 218, 232, 240]
⊢— [<Number> <]> <)> <EndOfLesson> ∇; D_4 <]> T_5 V_5 X_{20} <)>L_4 EndL L_2 ∇; 0, 30, 48, 31, 35, 36, 38, 16, 13, 18, 215, 218, 232, 240]
⊢— [<Number> <]> <)> <EndOfLesson> ∇; D_5 X_{21} <]> T_5 V_5 X_{20} <)>L_4 EndL L_2 ∇; 0, 30, 48, 31, 35, 36, 38, 16, 13, 18, 215, 218, 232, 240, 244]
⊢— [<Number> <]> <)> <EndOfLesson> ∇; N_2 X_{22} X_{21} <]> T_5 V_5 X_{20} <)>L_4 EndL L_2 ∇; 0, 30, 48, 31, 35, 36, 38, 16, 13, 18, 215, 218, 232, 240, 244, 247]

⊢ [<Number> <]> <)> <EndOfLesson> ∇; <Number> X_7 X_{22} X_{21} <]> T_5 V_5 X_{20} <)>L_4 EndL L_2 ∇; 0, 30, 48, 31, 35, 36, 38, 16, 13, 18, 215, 218, 232, 240, 244, 247,36]

⊢ [<]> <)> <EndOfLesson> ∇; X_7 X_{22} X_{21} <]> T_5 V_5 X_{20} <)>L_4 EndL L_2 ∇; 0, 30, 48, 31, 35, 36, 38, 16, 13, 18, 215, 218, 232, 240, 244, 247,36]

⊢ [<]> <)> <EndOfLesson> ∇; X_{22} X_{21} <]> T_5 V_5 X_{20} <)>L_4 EndL L_2 ∇; 0, 30, 48, 31, 35, 36, 38, 16, 13, 18, 215, 218, 232, 240, 244, 247,36, 38]

⊢ [<]> <)> <EndOfLesson> ∇; X_{21} <]> T_5 V_5 X_{20} <)>L_4 EndL L_2 ∇; 0, 30, 48, 31, 35, 36, 38, 16, 13, 18, 215, 218, 232, 240, 244, 247,36, 38, 249]

⊢ [<]> <)> <EndOfLesson> v; <]> T_5 V_5 X_{20} <)>L_4 EndL L_2 ∇; 0, 30, 48, 31, 35, 36, 38, 16, 13, 18, 215, 218, 232, 240, 244, 247,36, 38, 249, 245]

⊢ [<)> <EndOfLesson> ∇; T_5 V_5 X_{20} <)>L_4 EndL L_2 ∇; 0, 30, 48, 31, 35, 36, 38, 16, 13, 18, 215, 218, 232, 240, 244, 247,36, 38, 249, 245]

⊢ [<)> <EndOfLesson> ∇; V_5 X_{20} <)>L_4 EndL L_2 ∇; 0, 30, 48, 31, 35, 36, 38, 16, 13, 18, 215, 218, 232, 240, 244, 247,36, 38, 249, 245, 233]

⊢ [<)> <EndOfLesson> ∇; X_{20} <)>L_4 EndL L_2 ∇; 0, 30, 48, 31, 35, 36, 38, 16, 13, 18, 215, 218, 232, 240, 244, 247,36, 38, 249, 245, 233, 229]

⊢ [<)> <EndOfLesson> ∇; <)>L_4 EndL L_2 ∇; 0, 30, 48, 31, 35, 36, 38, 16, 13, 18, 215, 218, 232, 240, 244, 247,36, 38, 249, 245, 233, 229, 217]

⊢ [<EndOfLesson> ∇; L_4 EndL L_2 ∇; 0, 30, 48, 31, 35, 36, 38, 16, 13, 18, 215, 218, 232, 240, 244, 247,36, 38, 249, 245, 233, 229, 217]

⊢ [<EndOfLesson> ∇; EndL L_2 ∇; 0, 30, 48, 31, 35, 36, 38, 16, 13, 18, 215, 218, 232, 240, 244, 247,36, 38, 249, 245, 233, 229, 217, 40]

⊢ [<EndOfLesson> ∇; <EndofLesson> L_2 ∇; 0, 30, 48, 31, 35, 36, 38, 16, 13, 18, 215, 218, 232, 240, 244, 247,36, 38, 249, 245, 233, 229, 217, 40, 33]

⊢ [∇;L_2 ∇; 0, 30, 48, 31, 35, 36, 38, 16, 13, 18, 215, 218, 232, 240, 244, 247,36, 38, 249, 245, 233, 229, 217, 40, 33]

⊢ [∇; ∇; 0, 30, 48, 31, 35, 36, 38, 16, 13, 18, 215, 218, 232, 240, 244, 247,36, 38, 249, 245, 233, 229, 217, 40, 33, 214]

Prin urmare, derivarea stângă a consecutivității inițiale are următoare formă: 0, 30, 48, 31, 35, 36, 38, 16, 13, 18, 215, 218, 232, 240, 244, 247, 36, 38, 249, 245, 233, 229, 217, 40, 33, 214 (cifrele reprezintă numerele producțiilor aplicate).

Arborele construit de analizorul sintactic se utilizează pentru obținerea traducerii programului de intrare. Această traducere poate fi un program într-un cod intern al calculatorului, dar în cazul prezentat el reprezintă un program scris în limbajul intermediar, care este prelucrat de interpretor.

4.5. Problemele generării codului intermediar. Avantajele și limitările

Limbajul MoCo [19, 33] (*Course Modeler*) este un limbaj specific domeniului DSL[94, 95] de tip declarativ, conceput pentru descrierea cursurilor de instruire asistată de calculator. În implementarea acestui limbaj apar două opțiuni principale de generare a codului executabil pentru aplicațiile rezultate: *compilarea directă în cod mașină (cod executabil nativ)* sau *generarea unui cod intermediar*, care apoi este executat de un interpretorul dedicat – o mașină virtuală specifică limbajului [96, 97]. În continuare caracterizăm aceste două abordări, le comparăm avantajele și

dezavantajele, iar ulterior argumentăm alegerea soluției de cod intermediar pentru limbajul MoCo, având în vedere natura declarativă a DSL-ului și cerințele de portabilitate, flexibilitate și mentenanță.

Compilarea în cod executabil nativ.

Generarea codului executabil presupune compilarea programelor scrise în MoCo direct într-o formă binară nativă, de exemplu un fișier executabil sau cod de obiect, pentru o platformă țintă. Compilatorul ar traduce constructele declarative ale limbajului într-un cod mașină echivalent sau într-un cod sursă în limbaj de nivel jos, precum C/C++, urmat de compilare, producând un program autonom.

Avantaje: Principala atracție a compilării directe este *eficiența la execuție*. Codul mașină rezultat poate rula direct pe procesorul țintă, fără interpret suplimentar, atingând performanțe superioare interpretării. Acest lucru este important mai ales dacă aplicațiile generate ar necesita un timp de răspuns minim sau ar efectua calcule intensive deși, în cazul MoCo, scenariile de curs nu au calcule -intensive. Un alt avantaj este posibilitatea de a realiza o analiză statică și optimizare programelor înainte de execuție. Un compilator poate efectua verificări semantice complete, raportând erori la compilare și asigurând anumite proprietăți înainte de rulare. De pildă, un *generator de aplicații* (compilator LSD) poate realiza toată analiza statică asupra specificației și aplica optimizări de domeniu înainte de generarea codului final[98 - 100].

Astfel, compilarea ar putea oferi o detectare timpurie a erorilor în cursurile MoCo și un cod optimizat specific fiecărei platforme. În plus, codul nativ compilat poate oferi un anumit grad de protecție a proprietății intelectuale, deoarece distribuirea se face sub formă binară, ascunzând logica sursă față de utilizatorul final.

Dezavantaje: Pe de altă parte, generarea de cod executabil pentru fiecare platformă implică probleme majore de *portabilitate și întreținere*. Un program compilat este *legat de arhitectura și sistemul de operare* pentru care a fost generat. Cu alte cuvinte, un curs MoCo transpus într-un cod executabil Windows va rula doar pe Windows, necesitând o recompilare separată pentru Linux sau pentru dispozitive mobile. Această lipsă de interoperabilitate impune menținerea mai multor versiuni de cod și procesoare de compilare, mărind considerabil efortul de dezvoltare și testare *multi-platformă*.

Distribuirea aplicațiilor ca binare compilate le leagă de setul de instrucțiuni al procesorului și de interfața specifică a sistemului de operare, ceea ce devine *constrângător într-un mediu eterogen*, unde ideal ar fi ca software-ul să circule la fel de liber ca datele. În plus, realizarea unui compilator optimizant pentru un limbaj declarativ de nișă este un proces complex. Implementarea de la zero a unui *procesor de limbaj* complet (parsing, analiză, optimizare, generare de cod mașină) necesită un

efort substanțial de dezvoltare și cunoștințe avansate de compilare, ceea ce poate depăși resursele disponibile într-un proiect de cercetare.

Flexibilitatea este și ea redusă: odată ce limbajul evoluează sau apar noi cerințe, extinderea unui compilator optimizat poate fi dificilă. Orice modificare semnificativă în limbajul MoCo, de exemplu, adăugarea unui nou tip de element de curs ar impune ajustarea generatorului de cod și recompilarea acestuia pentru fiecare platformă suportată – un ciclu costisitor de dezvoltare. De asemenea, *depanarea* și testarea cursurilor dezvoltate în MoCo ar fi mai dificil de realizat la nivel de LSD în abordarea compilată. În general, în cazul unui LSD compilat, execuția are loc „prin proxy" – codul LSD este transpus într-un alt limbaj de nivel mai jos, astfel că depanarea ar trebui făcută la nivelul codului țintă (de exemplu, cod C++ generat), fără *feedback* direct în termeni de concepte MoCo. Astfel, autorul unui curs ar pierde legătura cu nivelul înalt al limbajului în timpul depanării, erorile fiind raportate în termeni de cod compilat, nu de elemente de curs MoCo.

Acest lucru îngreunează considerabil identificarea și corectarea problemelor în logica cursului. Pe scurt, abordarea compilării native oferă viteză de rulare și posibilitatea optimizării off-line, dar sacrifică portabilitatea și *ușurința evoluției* sistemului.

Generarea de cod intermediar și interpretarea acestuia

A doua opțiune este *generarea unui cod intermediar*, urmată de interpretarea lui cu ajutorul unui interpretor specific – o mașină virtuală. În această abordare, programele MoCo nu sunt traduse până la cod mașină real, ci într-o reprezentare intermediară abstractă, adesea numită *bytecode*, *IR – Intermediate Representation* sau *IL – Intermediate Language*). Acest cod intermediar conține instrucțiuni la un nivel mai înalt decât codul de asamblare native și este conceput pentru a fi *independent de platformă*. Un modul de runtime – interpretul MoCo va citi și executa aceste instrucțiuni intermediare *în timpul rulării*, realizând legătura cu sistemul de operare și hardware-ul curent.

Avantaje: Principala motivație pentru cod intermediar interpretat este *portabilitatea*. Un program MoCo, odată tradus în codul său intermediar, poate fi rulat pe orice platformă pentru care există implementat interpretul limbajului. Practic, distribuția cursului e un pachet format din codul intermediar, identic pe toate sistemele, împreună cu interpretul (specific fiecărui SO). Această filozofie "write once, run anywhere" a stat la baza multor platforme de succes (e.g. Java, care compilează în bytecode portabil pe JVM). În cazul MoCo, în loc să menținem compilatoare separate pentru Windows, Linux, Android etc., am menține un singur interpret (eventual scris în C++ sau alt limbaj portabil) pe care îl compilăm pentru fiecare platformă. *Codul cursului rămâne neschimbat,*

deci autorii de curs și utilizatorii nu trebuie să-și facă griji despre compatibilitate – orice curs MoCo poate rula oriunde interpretul este disponibil.

Un alt avantaj major este *flexibilitatea și ușurința evoluției*. Un interpret de limbaj declarativ este, în esență, un *motor de execuție* care parcurge structurile de date ale programului și implementează semantica limbajului pas cu pas. Această structură tinde să fie mai simplă și mai ușor de modificat comparativ cu un compilator optimizant.

Interpretarea are avantaje precum *simplitatea și capacitatea de extensie sporită* a limbajului, atâta timp cât performanța nu este critică. Într-adevăr, interpretorul oferă un *control mai mare asupra mediului de execuție* și este mai ușor de extins cu noi constructe, pe când compilatoarele au, de obicei, arhitecturi rigide.

În contextul MoCo, unde limbajul este orientat pe descriere declarativă a scenariilor educaționale, este de așteptat ca pe parcursul cercetării să apară cerințe de extindere (noi tipuri de activități, noi reguli pedagogice de ramificare a cursului etc.).

Implementarea acestor schimbări direct în interpretor este mult mai rapidă și mai sigură: se pot adăuga noi instrucțiuni în codul intermediar și suporta în interpretor, fără a modifica întregul flux de generare a codului pentru fiecare platformă. Practic, arhitectura bazată pe interpret favorizează un ciclu de dezvoltare *iterativ și incremental*: se poate testa imediat efectul unei modificări în limbaj prin rularea interpretorului pe un set de cursuri de test fără etapa costisitoare de compilare completă. Acest lucru crește productivitatea dezvoltatorului limbajului și permite o evoluție mai agilă a platformei.

Ușurința depanării este un alt atu al interpretării. Fiind un limbaj specific domeniului educațional, este de dorit ca autorii cursurilor care pot fi experți educaționali, nu neapărat programatori să poată înțelege și urmări execuția cursului pentru a depana eventuale probleme de logică. Un interpretor poate oferi direct *feedback* în termeni de concepte MoCo în timpul execuției: de exemplu, se poate indica în ce modul al cursului sau la ce întrebare a rămas studentul, ce ramură a scenariului a fost luată, etc. Într-un LSD interpretat *starea dinamică a modelului și pașii de execuție pot fi observați la nivelul conceptelor de domeniu*, permițând vizualizarea directă a progresului execuției în termeni familiari domeniului [101, 102].

În schimb, într-un LSD compilat, execuția se traduce la nivelul limbajului țintă, astfel că depanarea implică informații la nivel inferior, pierzând legătura cu modelul inițial [103, 104]. Prin urmare, interpretarea facilitează dezvoltarea unui debugger la nivel de LSD, care să permită autorilor de curs să urmărească scenariul educațional pas cu pas și să detecteze ușor eventualele erori în logica cursului. De asemenea, rapoartele de eroare pot fi formulate direct în termenii limbajului MoCo (ex:

"Răspuns neprevăzut negestionat la întrebarea X din modulul Y"), ceea ce este esențial pentru un domeniu unde utilizatorii finali ai limbajului sunt experți în educație, nu în programare.

Raportarea de erori poate fi mult îmbunătățită în cazul LSD-urilor executabile, tocmai datorită posibilității de a controla interpretarea și de a valida reguli specifice domeniului [104].

În ceea ce privește *performanța*, interpretarea adaugă, desigur, un накладные costurile suplimentare la execuție. Fiecare instrucțiune intermediară trebuie decodificată și executată de software-ul interpretor, spre deosebire de codul nativ care rulează direct pe hardware. Aceasta înseamnă că aplicațiile MoCo interpretate vor rula mai lent decât cele compilate. Totuși, în contextul aplicațiilor de instruire asistată care sunt predominant *I/O-bound* și *event-driven*, așteptând input de la utilizator, afișând informații etc., penalizarea de viteză este rar critică.

MoCo fiind un limbaj declarativ, orientat pe orchestrarea unor resurse educaționale și logică de predare, *viteza absolută de execuție nu este parametru decisiv*. Prin design, limbajul favorizează claritatea și flexibilitatea față de optimizarea de cicluri CPU. Astfel, atâta timp cât interfața rămâne responsivă pentru utilizator, interpretarea este viabilă. În plus, există tehnici de *compilare just-in-time (JIT)* sau caching al codului interpretat care pot atenua diferența de performanță, dacă vreodată va deveni o problemă. Complexitatea acestor optimizări este însă justificată doar la nevoie; deocamdată, simplitatea interpretorului are prioritate.

În concluzie, abordarea cu cod intermediar interpretat oferă portabilitate maximă, facilitează evoluția limbajului și simplifică depanarea, în schimbul unei complexități de execuție ușor sporite și a unei potențiale penalizări de viteză, considerată acceptabilă în domeniul țintă.

Justificarea alegerii interpretării codului intermediar pentru MoCo

Având în vedere caracteristicile limbajului MoCo și cerințele sistemului, opțiunea generării de cod intermediar urmată de interpretare a fost considerată superioară. În cele ce urmează, sintetizăm motivele acestei alegeri, corelate cu argumente din literatura de specialitate:

- **Natura declarativă și specifică domeniului a limbajului.** MoCo este un LSD declarativ, concentrându-se pe *ce* trebuie realizat într-un curs: structura modulelor, întrebărilor, condițiilor de tranziție etc., nu pe *cum* să fie implementat fiecare pas. Acest caracter înalt-abstract îl aseamănă cu limbaje precum SQL sau HTML, care de regulă sunt interpretate sau compilate în cod intermediar, nu în cod mașină specific fiecărei platforme.

  Caracterul declarativ implică faptul că interpretorul are libertatea de a decide ordinea exactă a execuției și gestionarea resurselor, atâta timp cât respectă specificația. Prin urmare, un interpretor dedicat poate evalua flexibil scenariile didactice, permițând *reguli de execuție*

dinamice de exemplu, sărirea peste anumite secțiuni în funcție de performanța cursantului, mai ușor decât un cod precompilat rigid.

LSD-urile declarative beneficiază de pe urma interpretării, deoarece aceasta poate acomoda comportamente dinamice și extensii ulterioare mult mai simplu. Totodată, folosind interpretarea, limbajul poate rămâne *aproape de notația de domeniu*, permițând experților să îl înțeleagă direct și să îl valideze ceea ce este un deziderat important în proiectarea LSD-urilor de succes.

- **Portabilitatea aplicațiilor pe Windows, Linux, mobil.** O cerință esențială a platformei MoCo este ca aplicațiile generate (cursurile offline) să fie utilizabile pe diferite sisteme de operare și dispozitive (PC-uri cu Windows sau Linux, tablete Android etc.). Alegând interpretarea, obținem *portabilitate aproape imediată*: același cod intermediar MoCo rulează pe orice sistem care are interpretul instalat. Nu este nevoie să recompilăm fiecare curs pentru multiple platforme, ci doar să furnizăm interpretorul potrivit. Această abordare este similară cu cea a mașinilor virtuale portabile (ex: JVM pentru Java). Sloganul „Scrieți o dată, rulați oriunde" promovat de Java se aplică și aici – un curs MoCo dezvoltat pe Windows poate rula neschimbat pe Linux sau pe un telefon, interpretorul acționând ca strat de abstractizare între codul cursului și platforma hardware.

  Prin contrast, dacă ar fi să alegem compilarea nativă, ar fi trebuit fie: să generăm cod pentru un limbaj intermediar portabil precum Java/C# (introducând însă dependențe de platforme externe), fie să compilăm separat pentru x86/Windows, x86/Linux, ARM/Android etc., cu costurile de rigoare.

  Practic, interpretarea ne scutește de implementarea multiplă a generării de cod, *mutând efortul de portare de la etapa de compilare la etapa de execuție*. E mai eficient să implementăm **o singură dată** interpretorul decât să implementăm generatorul de cod pentru fiecare platformă. Practica confirmă că limbajele interpretate sunt mai ușor de făcut *independente de platformă* – aplicațiile pot fi transferate ca atare pe diferite arhitecturi unde doar interpretorul diferă.

  Această portabilitate crește considerabil atractivitatea și longevitatea soluției MoCo: cursurile create vor putea fi folosite în medii diverse: laboratoare universitare cu diferite sisteme de operare, pe laptopuri personale ale studenților, sau pe dispozitive mobile în afara sălii de clasă fără efort suplimentar de conversie.

- **Flexibilitate, întreținere și evoluție continuă.** Un argument strategic pentru alegerea interpretorului este *necesitatea de a acomoda schimbări și extinderi* în timp, cu efort minim. Abordarea interpretativă este mai viabilă: modificările în limbaj se pot implementa prin adăugarea/modificarea unor rutine în interpretor, fără a schimba infrastructura de compilare pe fiecare platformă. *Interpretoarele sunt relativ ușor de extins*, comparativ cu compilatoarele, care sunt dificil de extins decât dacă au fost special concepute pentru extensibilitate. De exemplu, dacă dorim să introducem un nou tip de interacțiune educațională, vom adăuga o instrucțiune nouă în setul codului intermediar MoCo și vom scrie funcționalitatea ei în interpretor o singură dată. În schema compilată, ar fi trebuit să ne asigurăm că generatorul produce cod corect pentru acea interacțiune pe *toate* platformele, posibil necesitând cod specific fiecărui OS, dacă interacțiunea implică API-uri native). În plus, *timpul de dezvoltare* și testare în cazul interpretării este mai scurt: nu trebuie să așteptăm compilarea completă a sistemului de fiecare dată; putem testa imediat schimbările la nivel de interpretor rulând secvențe de cod intermediar de test fiindcă orice modificare necesită relansarea procesului de compilare care, pentru proiecte mari, poate dura considerabil, pe când limbajele interpretate permit un ciclu rapid editare-execuție

  Acest ciclu rapid crește productivitatea și facilitează *întreținerea* pe termen lung, deoarece corecțiile de erori sau optimizările minore în comportamentul limbajului pot fi distribuite utilizatorilor doar printr-un update al interpretorului, nu prin regenerarea tuturor aplicațiilor existente.

  În contextul MoCo, dacă după livrarea a zeci de cursuri se descoperă o problemă în mecanismul de evaluare a răspunsurilor, corectarea interpretorului va rezolva instant problema pentru toate cursurile, fără ca autorii acestora să trebuiască să le recompileze. Aceasta echivalează cu un cost mult redus de *mentenanță* și o capacitate de reacție sporită la cerințe noi.
- **Ușurința depanării și extinderii funcționalităților la nivel de interpretor.** După cum a fost menționat, interpretarea oferă suport nativ pentru un depanator la nivel de LSD și pentru mesaje de eroare comprehensibile pentru utilizatori. Acest lucru este esențial pentru adoptarea cu succes a limbajului de către experți non-tehnici. Alternativa, un sistem cu compilare nativă, i-ar fi forțat pe dezvoltatori să se bazeze pe depanatoare de nivel jos sau ar fi impus efortul suplimentar de a construi un strat de *mapping invers* de la erorile platformei către elementele LSD – exact problema identificată în literatura despre depanatoare de LSD compilate [101, 105].

Prin folosirea unui interpretor, am ales calea simplă și robustă: *execuția în mediul propriu limbajului*, unde dispunem de toată informația necesară pentru a asocia starea execuției cu elementele modelului educațional.

Pentru limbajele executabile, *interpretarea (semantica operațională directă) oferă observabilitate și control asupra execuției*, permițând instrumente de urmărire și depanarea la nivel de model, pe când compilarea (semantica de translare) pierde această transparență, necesitând mecanisme complexe de feedback. Cu interpretarea, implementarea unor unelte precum trace-uri de execuție, vizualizări de parcurgere scenariilor de instruire, instrumente de profilare a învățării devine realizabilă într-un mod natural în cadrul platformei MoCo.

Rezumând comparația, **Tabelul 1** sintetizează principalele diferențe între cele două opțiuni:

Tabelul 1. Comparația generării de cod executabil vs. cod intermediar interpretat pentru MoCo.

Aspect	**Generare cod executabil (compilare)**	**Generare cod intermediar (interpretare)**
Performanță	Foarte ridicată (cod nativ, optimizat)	Moderată, depinde de viteza interpretorului (suficientă pentru aplicații IAC)
Portabilitate	Redusă – necesită recompilare pe fiecare platformă	Foarte mare – codul rulă pe orice platformă cu un interpret
Efort de dezvoltare al infrastructurii	Mare – compilator complex cu toolchain per SO	Mai redus – interpret relativ simplu, reutilizat pe toate SO
Extensibilitate (evoluția limbajului)	Greu de extins (compilatorul dificil de adaptat)	Ușor de extins (adăugare de noi instrucțiuni în interpret)
Mentenanță	Costisitoare – fiecare modificare impune recompilare multiplatformă	Facilă – modificările se fac o singură dată în interpret
Depanare și testare	Dificilă la nivel LSD – nevoie de unelte suplimentare de mapare	Facilă – interpretul oferă feedback în termeni LSD
Analiză statică	Completă – verificări exhaustive înainte de execuție	Limitată – necesită implementare separată dacă e cazul (altfel erori la runtime)
Dimensiune distribuție	Cod binar optimizat, fără surse (protecția IP)	Cod intermediar cu interpret (eventual mărime mai mare, dar acceptabilă)

Din analiza de mai sus reiese că, pentru cazul limbajului MoCo, *beneficiile interpretării codului intermediar depășesc dezavantajele*. Limbajul este orientat pe exprimarea la nivel înalt a unor scenarii de instruire, unde *portabilitatea* și *flexibilitatea* sunt cruciale, iar *performanța* extremă nu este critică. Abordarea prin interpretare corespunde de altfel tendințelor din domeniul limbajelor dedicate: LSD-

urile sunt adesea implementate interpretativ când se urmărește maximizarea productivității utilizatorilor și a dezvoltatorilor limbajului. Prin alegerea generării de cod intermediar pentru MoCo, ne asigurăm că platforma de cursuri asistate de calculator va fi ușor de *portat*, de *extins* și de *menținut* pe termen lung, permițând totodată autorilor de curs să beneficieze de un mediu de execuție *transparent și prietenos* pentru depanarea și îmbunătățirea continuă a conținutului educațional. Această decizie tehnologică aliniază dezvoltarea limbajului MoCo cu cele mai bune practici și cu cerințele către ale sistemele de instruire moderne.

4.6. Limbajul intermediar. Generarea codului interpretabil

Codul intermediar reprezintă consecutivitatea codurilor operatorilor și a datelor. Ca simbol delimitator în limbajul intermediar servește simbolul cu codul #00. *Codul operatorului* este un număr întreg unic, care urmează după simbolul delimitator și ocupă 1 octet (numărul total al operatorilor limbajului permite utilizarea unui octet pentru codificare). Fiecărei structuri a limbajului MoCo îi corespunde o structură anumită în limbajul intermediar. Lista integrală de cuvinte-cheie ale limbajului intermediar este expusă în anexa 6.

Vom aducem exemple ale codului intermediar.

- Diapazonul

#00	Codul Diapazonului	Diapazonul de numere

Diapazonul de numere reprezintă consecutivitatea Numerelor, despărțită de Cod & sau Cod To (aceste coduri se încep cu simbolul #00).

Exemplu de diapazonul numerelor:

Textul inițial: *9, 12..45, 67, 5*

Rezultatul - #00 #28 $(9)_4$ #00 #46 #00 #28 $(12)_4$ #00 #45 $(45)_4$ #00 #46 $(67)_4$ #00 #46 $(5)_4$, unde indicele de jos reprezintă numărul de octeți, alocat numărului, #28 – Cod Number, #45 Cod To, și #46 Cod &.

- Lecția

#00	Cod Lesson	Consecutivitatea de simboluri (excluzând simbolul 00) – numărul lecției			
#00	Cod Text	Denumirea lecției	#00	Cod Condition	Condiția

Informația amplasată după #00 Cod Lesson până la următorul #00 Cod Lesson sau până la sfârșitul fișierului se consideră o lecție.

Exemplu de lecție:

Textul inițial: *_Lesson (10 "History" [3,6..9])*

Rezultatul – #00 #24 10 #00 #38 History #00 #06 #00 #11 #00 #28 $(3)_4$ #00 #46 #00 #28 $(6)_4$ #00 #45 #00 #28 $(9)_4$

- Trecerea

#00	Cod Condition	Condiția	#00	Cod GoTo	Consecutivitatea de simboluri – denumirea etichetei

Consecutivitatea arbitrară de treceri reprezintă Blocul de treceri.

Exemplu de Branch (Blocul de treceri):
Text inițial: _Branch Time < 10, Label1 / Time >= 20, Label2

Rezultat: #00 #02 #00 #06 #00 #08 Time #00 #53 #00 #28 (10)4 #00 #19 Label1 #00 #06 #00 #08 Time #00 #54 #00 #28 (20)4 #00 #19

- Question Free, Question Single sau Question Multiple

#00	Cod Free Cod Single Cod Multiple	Numărul încercărilor de răspuns (1 octet)	Bloc informațional, formularea întrebării.	Bloc Free, Bloc Single sau Bloc Multiple, respectiv

Consecutivitate de întrebări este o îmbinare de instrucțiuni Question Free, Question Single sau Question Multiple.

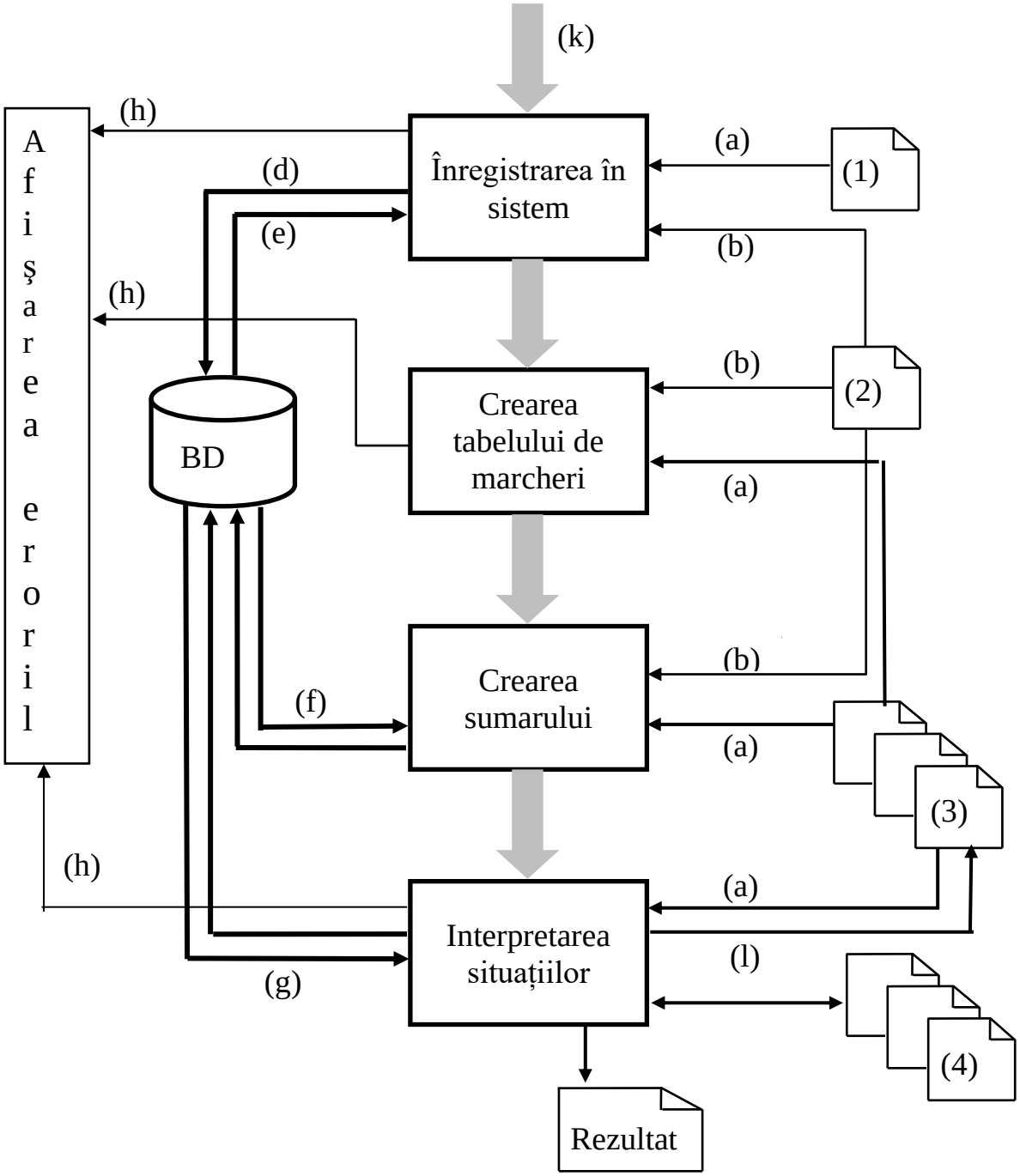
Descrierea completă a limbajului intermediar este inclusă în anexa 8.

În baza limbajului intermediar se generează codul de interpretare cu ajutorul arborelui sintactic, obținut la etapa de analiză sintactică, și a informației păstrate în tabelul constantelor și tabelul de texte se. Deși construirea arborelui sintactic și generarea codului sunt descrise ca procese consecutive, în practică, ele se realizează concomitent.

Pentru generarea codului de interpretare, în baza gramaticii **G'** a fost construită gramatică $\mathbf{G_2}$ ce diferă de cea inițială prin absența simbolurilor terminale neesențiale.

În procesul de analiză a consecutivității de lexeme de intrare se construiesc doi arbori ce reprezintă un program sursă [87]. Frontieră primului arbore A_1 coincide exact cu simbolurile terminale ale programului sursă. Frontieră arborelui al doilea A_2 reprezintă program echivalent în limbajul intermediar. Simbolurile terminale ale arborelui A_2 sunt salvate în fișier ce reprezintă ieșirea analizatorului semantic.

Deci, după terminarea analizei semantice la ieșirea translatorului va fi codul limbajului intern al interpretatorului, ce corespunde programului sursă la intrarea translatorului.



(a) codul intermediar;
(b) parametrii, denumirea BD;
(c) lista fișierelor, parametrii;
(d) loginul, parola;
(e) identificatorul, nivel de acces a utilizatorul;
(f) informația despre lecții trecute;
(g) parametrii cursului;
(h) codul erorii.

(k) denumirea fișierului proiectului;
(l) poziția și denumirea fișierului;
(m) începutul cursului;

(1) prefața cursului (*.mcf);
(2) fișierul proiectului (*.mcp);
(3) fișiere cursului (*.mcf);
(4) fișiere adăugătoare

Figura 4.2. Schema interpretorului limbajului MoCo.

Codul interpretabil, având structura sus descrisă, se păstrează într-un fișier cu extensie și în continuare acest fișier va fi prelucrat de un interpretator. Schema generală a interpretatorului limbajului MoCo este prezentată în Figura 4.2.

4.7. Analiza și corectarea erorilor

Până acum am considerat că datele de intrare a translatorului sunt programe corecte și fiecare fază de translare poate fi adusă logic la un final. În practică, adeseori această nu are loc. Sunt diverse posibilități de apariție a erorilor în programele surse.

Translatorul are posibilitatea de a descoperi erorilor în program cel puțin la trei etape de translare:

- pe parcursul analizei lexicale (din cauza lipsei lexemului în programul sursă sau din cauză prezentei lexemului incorect),
- analizei sintactice (consecutivitatea de lexeme obținută nu se asociază nici cu un arbore sintactic)
- a generării codului de interpretare (din consecutivitatea de intrare nu poate fi obținut nici un cod de interpretare logic).

Drept exemplu de un asemenea caz poate servi cazul când în situația _Question apar două răspunsuri identice, descrise ca fiind corect și incorect simultan. Analizatorul sintactic ignoră componenta informațională a lexemului, deci el nu va observa aceasta eroare.

Algoritmul ***LL*** de analizare, folosit la etapa analizei sintactice are următoarea proprietate. La descoperirea faptului că algoritmul poate anunța faptul că consecutivitatea de intrare nu mai poate fi continuată până la o consecutivitate corectă de ieșire. Totodată tabelul de dirijare MTP este construit astfel, încât în toate celulele de erori este înscris codul erorii respective. Cu acest cod este asociată o diagnostică – text a erorii. Datorită acestuia și faptului că translatorul păstrează poziția lexemelor analizate referitor la începutul programului, se efectuează o diagnostică efectivă a textului sursă.

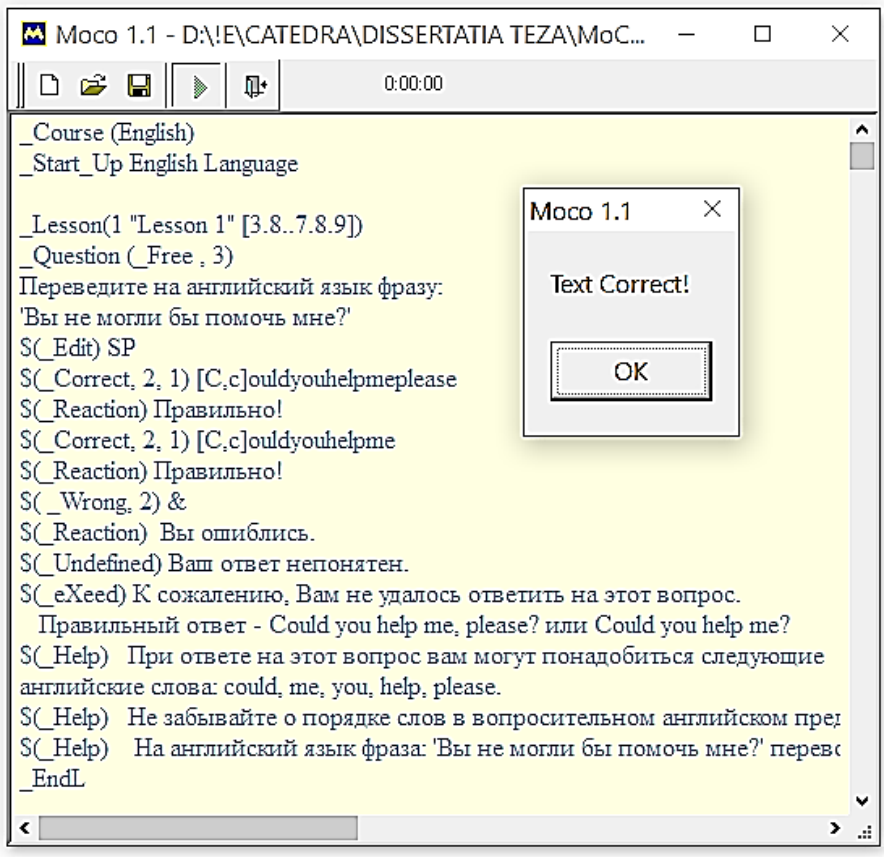


Figura 4.3. Un exemplu de lucru al compilatorului limbajului MoCo.

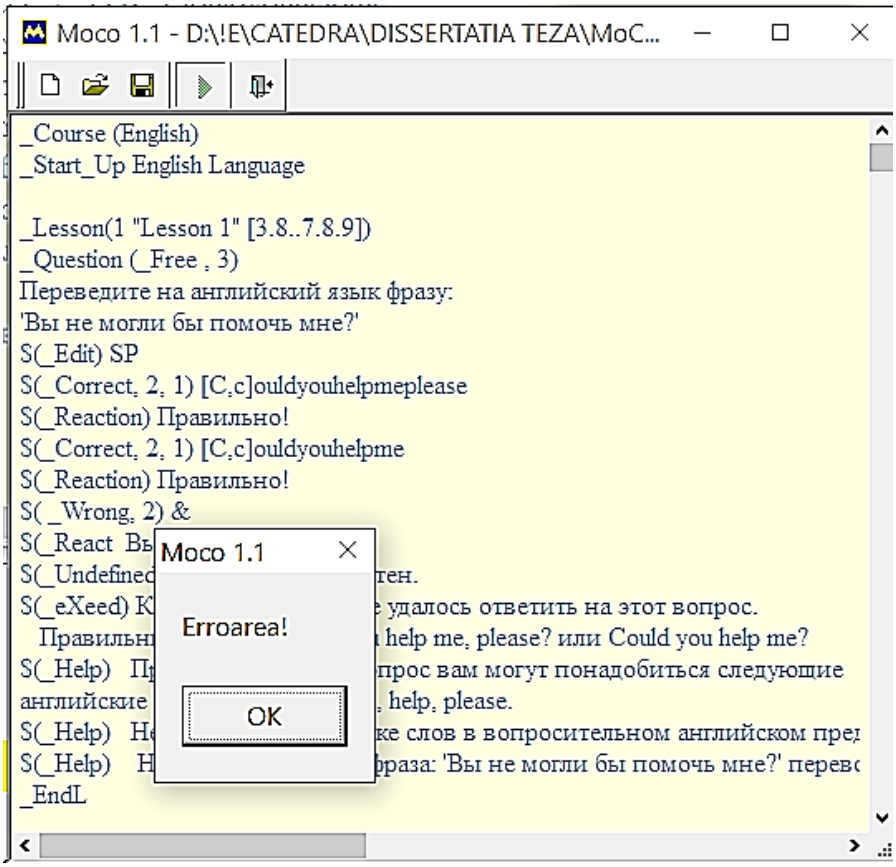


Figura 4.4. Un exemplu de lucru al compilatorului limbajului MoCo.

4.8. Platforma *no-code* pentru dezvoltarea cursurilor de instruire asistate de calculator

În primul rând, să definim conceptele de platforme low-code și no-code.

Low-code este o metodă de proiectare și dezvoltare a aplicațiilor care utilizează instrumente intuitive și funcții integrate care reduc cerințele tradiționale (profesionale) de scriere a codului. Programarea profesională rămâne o parte a procesului de dezvoltare, dar dezvoltarea low-code sprijină o experiență sporită și simplificată pe care utilizatorii obișnuiți o pot stăpâni rapid. Platformele de dezvoltare cu cod redus (LCDP) permit accelerarea dezvoltării aplicațiilor.

No-code platforme dezvoltare (NCDP) permit crearea de aplicații software prin intermediul interfețelor grafice cu utilizatorul și al configurării aplicațiilor, în locul programării tradiționale pe calculator. Platformele de dezvoltare no-code sunt strâns legate de platformele de dezvoltare low-code, deoarece ambele sunt concepute pentru a accelera procesul de dezvoltare a aplicațiilor. Cu toate acestea, spre deosebire de low-code, platformele de dezvoltare no-code nu necesită deloc scrierea de cod, oferind în general șabloane predefinite cu ajutorul cărora întreprinderile pot crea aplicații. În ultimii ani aceste platforme cresc în popularitate fiindcă permit crearea programelor de către utilizatori care nu sunt specialiști în programare, de exemplu profesori din diferite domenii de educație.

Principala diferență între platformele de dezvoltare *low-code* și *no-code* este cantitatea de cunoștințe de programare necesară utilizatorului. Platformele de dezvoltare low-code (LCDP) [106, 107] necesită anumite cunoștințe de bază de programare pentru a dezvolta și integra aplicații complexe, în timp ce platformele de dezvoltare no-code (NCDP) nu necesită deloc abilitatea de a scrie cod [108, 110].

Având în vedere că autorii cursurilor de instruire asistată de calculator sunt de cele mai multe ori specialiști în domeniul lor de activitate și nu sunt programatori, platformele NCDP sunt mai potrivite pentru ca aceștia să dezvolte cu succes cursuri.

În acest capitol vom descrie un prototip al unei astfel de platforme dezvoltat de autor pe baza limbajului MoCo. Această platformă este o platformă NCDP.

Ideile principale.

Succesul multor limbaje de programare și limbaje de descrieri se bazează pe diferiți factori. Unul din cele mai importanți factori prezintă posibilitatea creării rapide și comode a produsului final.

Platforme NCDP de programare de obicei rezolvă problema construirii interfeței cu utilizator și problema simplificării elaborări aplicațiilor. Aceasta se realizează prin înlocuirea metodei de scriere a programei cu metoda construirii. Platforme NCDP presupun utilizarea programării vizuale care presupune prezentarea elementelor constructive a programei în mod intuitiv. A fost creată platforma

NCDP care conține mediului vizual care ar permite, crearea cursurilor asistate de calculator fără cunoașterea a limbajului MoCo.

Pe parcursul elaborării programului platforma generează codul programului.

În mediul MoCo se realizează conceptul de creare a cursului din situații instructive. Interfața a situației instructive definește felul de comportare și modul de legătura cu alte situații în curs. Interfața constă din datele următoare:

- *Atribute* – datele la care au acces alte situații instructive. Aceste date reprezint conținut și stările situației. De exemplu acestea pot fi textul întrebării, denumirea testului, denumirea unui fișier etc.
- *Evenimente* – semnale transmise de la situația pentru schimbarea stării sale, de exemplu, despre răspuns incorect la o întrebare.
- *Acțiuni* – operații sau funcții ce pot fi executate de o situație. De exemplu, reacția la depășirea numărului de încercări de răspunsuri la întrebare sau oferirea ajutorului.

În Figurile 4.5 – 4.6 este prezentată interfața mediului NCDP platformei de creare a cursurilor de instruire asistate de calculator. Aici sunt prezente componentele de bază a platformei.

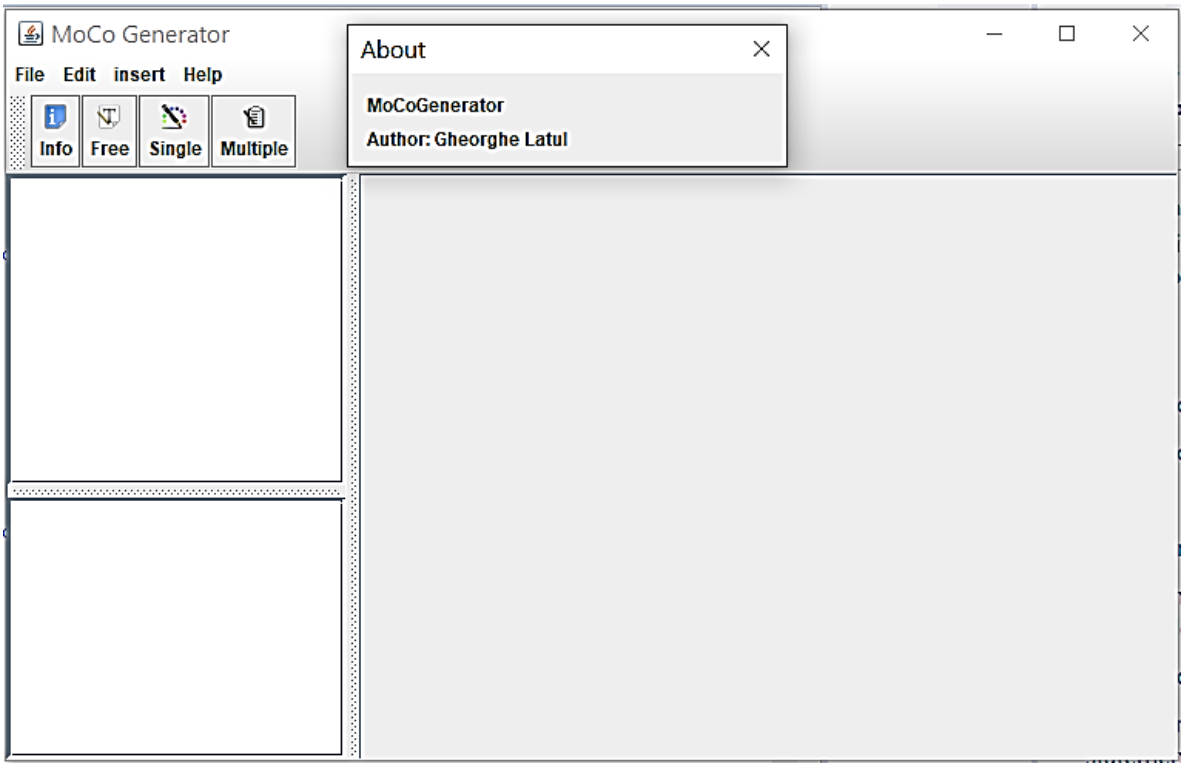


Figura 4.5. Componentele de bază a platformei

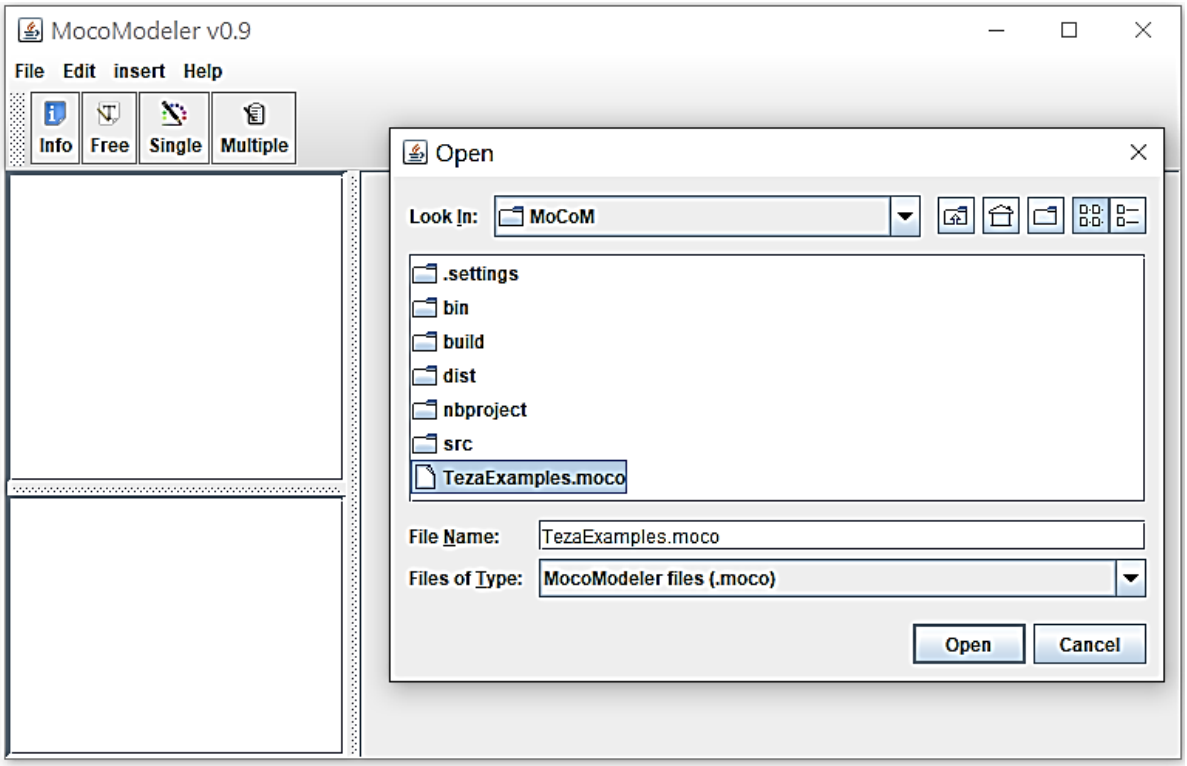


Figura 4.6. Componentele de bază a platformei

În Figurile 4.7 – 4.8 este prezentată interfața mediului NCDP platformei de creare a cursurilor de instruire asistate de calculator. Aici sunt prezente componentele de bază a platformei.

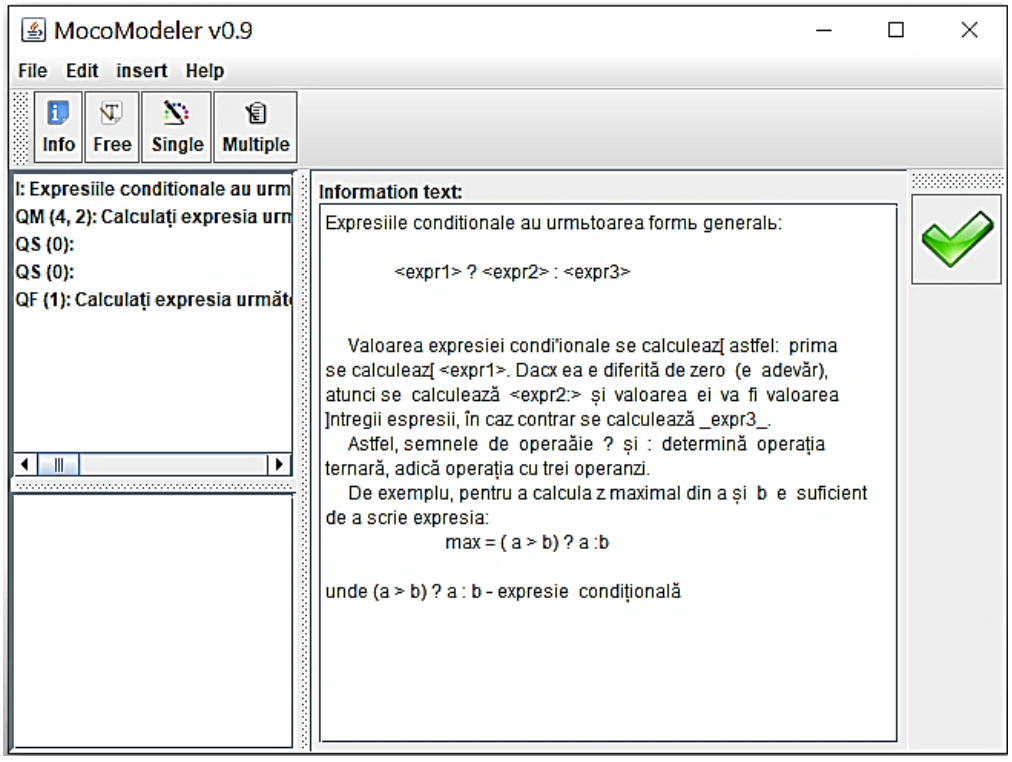


Figura 4.7. Interfața generatorului. Crearea situației „Information – I”

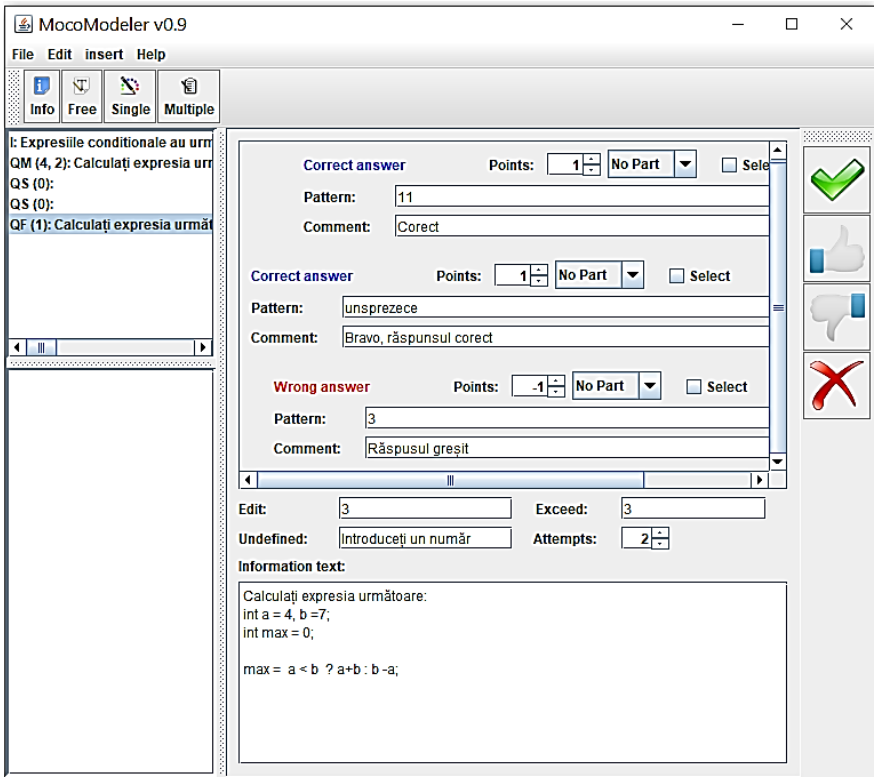


Figura 4.8. Interfața generatorului. Crearea întrebărilor pentru un test.

Platforma poate fi considerată ca softul orientat pe o *familie de probleme* [111] – crearea cursurilor asistate de calculator. Această no-code platforma generează cod corect în limbajul MoCo, care se bazează pe frame-uri (pe cadre), iar acest lucru face posibilă crearea unui interpretor al codului relativ ușor de implementat.

Programul generat este trecut la intrarea interpretorului, care lucrează folosind structura de frame-uri a limbajului. În Figurile 4.9 – 4.13 este prezentat lucrul iinterpretorul limbajului MoCo

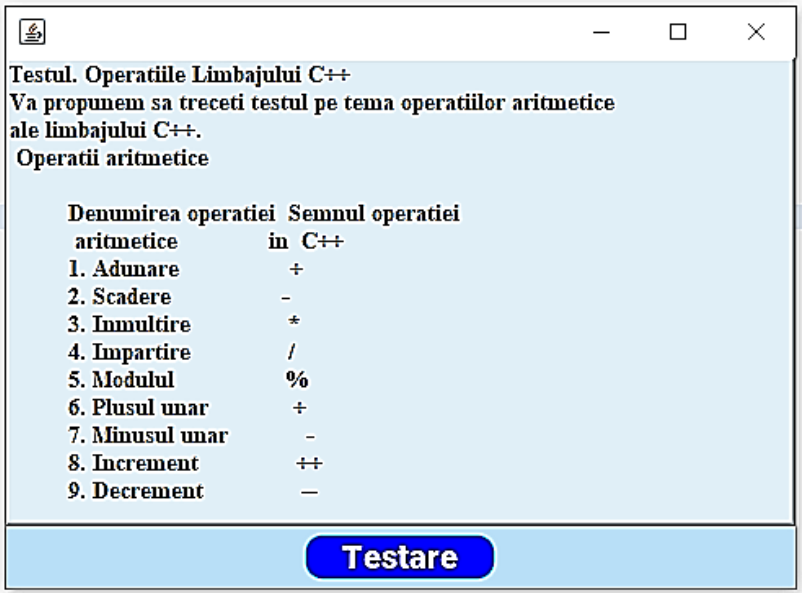


Figura 4.9. Lucrul iinterpretorul limbajului MoCo – afișarea informației

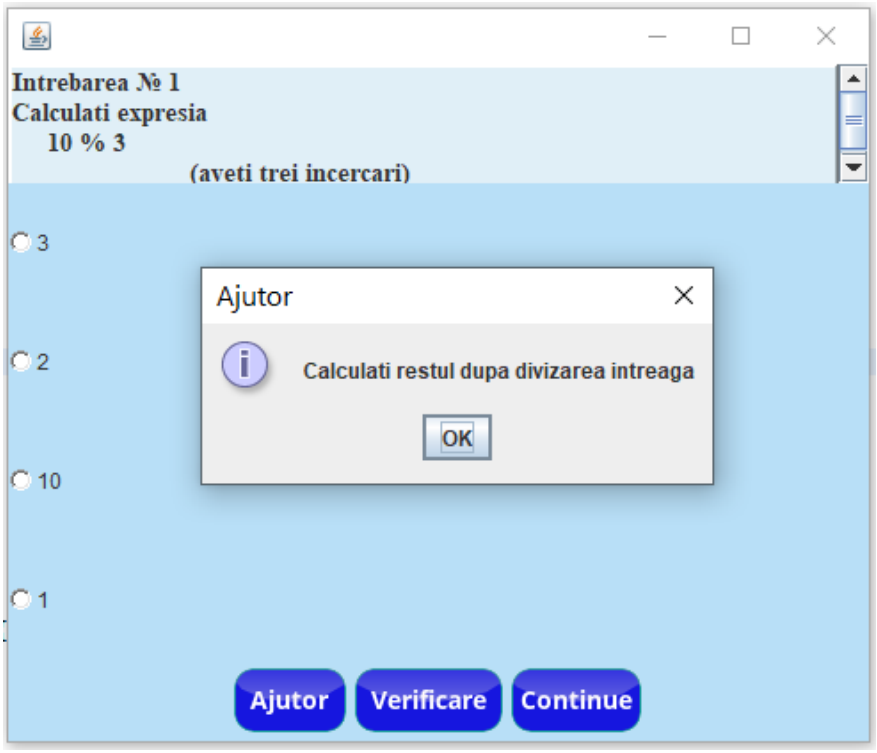


Figura 4.10. Lucrul iinterpretorul limbajului MoCo afișarea ajutorului

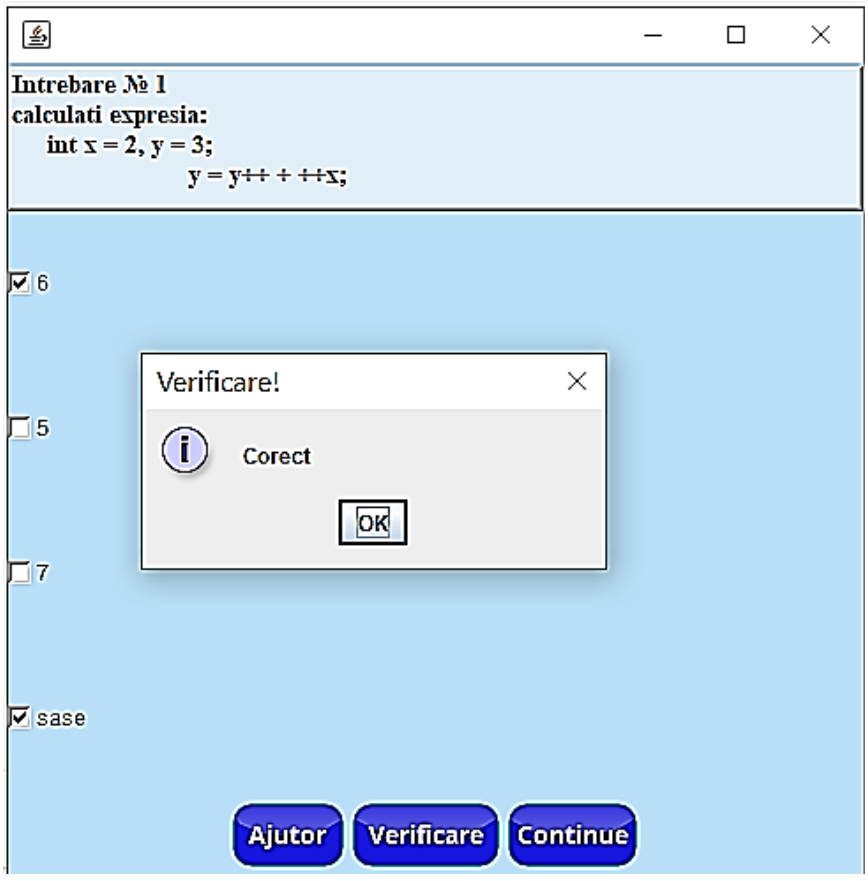


Figura 4.11. Lucrul iinterpretorul limbajului MoCo – verificarea răspunsului

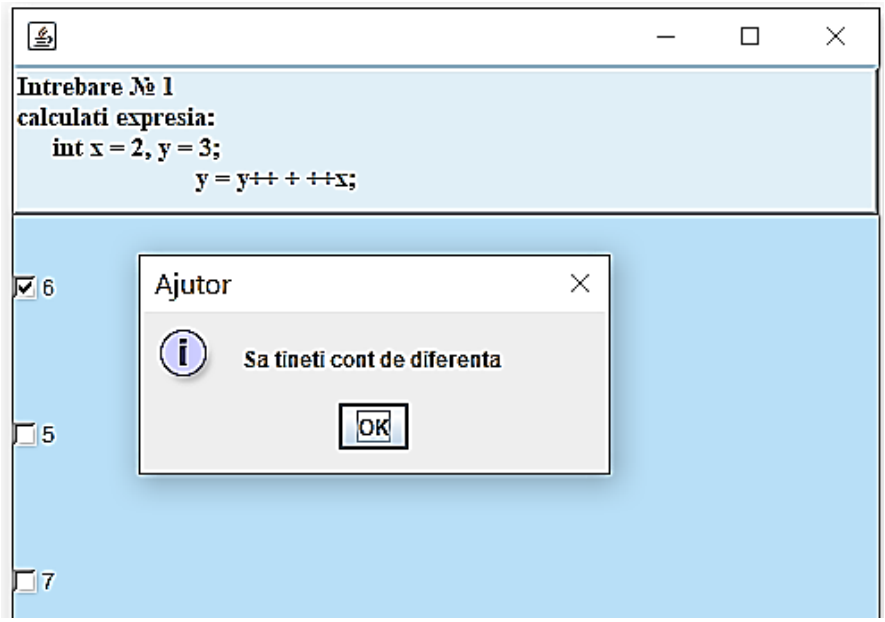


Figura 4.12. Lucrul iinterpretorul limbajului MoCo afișarea ajutorului

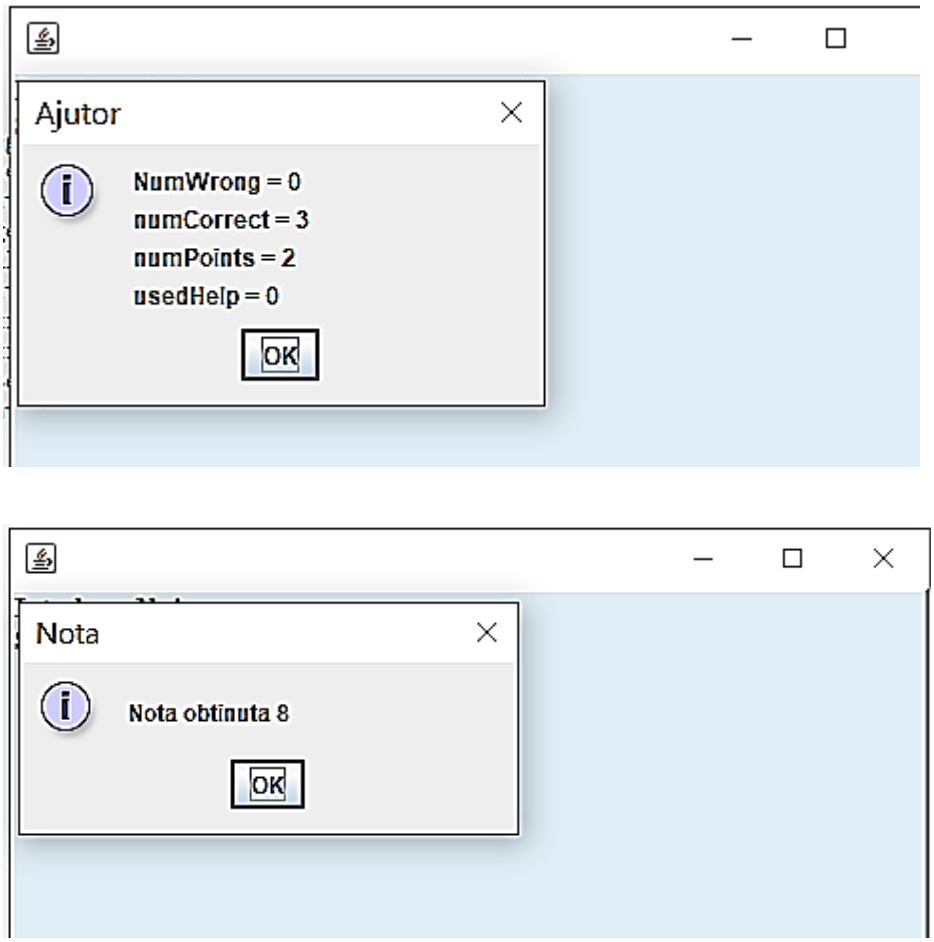


Figura 4.13. Lucrul iinterpretorul limbajului MoCo. Nota și statistica

O astfel de platformă poate permite dezvoltarea rapidă a aplicațiilor de către neprofesioniști, reducând timpul și costurile asociate cu metodele tradiționale de programare.

Pe perspectivă, ținând cont de experiența de crearea prototipului, este posibil să se creeze un IDE – mediu de dezvoltare integrat multifuncțional care va include toate subsistemele, inclusiv interpretorul.

4.9. Concluzii la capitolul 4

1. A fost definită gramatica limbajului MoCo care este o gramatică independentă de context. Ca urmare, există un automat cu memorie stivă care acceptă (determină) limbajul elaborat de către autor.
2. A fost definită mulțimea cuvintelor cheie ale limbajului MoCo, mulțimea simbolurilor auxiliare utilizate la descrierea limbajului, mulțimea producțiilor ***P'*** și ale simbolurilor neterminale ***N'*** a limbajului.
3. Reieșind din necesitatea construirii unui algoritm efectiv gramatica inițială a limbajului descrierii cursurilor asistate de calculator MoCo descris prin regulile în forma normală Backus Naur a fost transformată în gramatică care satisface condițiile ***LL(1)***.
4. Au fost definite etapele principale ale translatorului limbajului.
5. A fost definită metoda generării codului intermediar și interpretabil al limbajului.
6. A fost creat compilatorul limbajului MoCo.
7. A fost create prototipurile *no-code* platformei și a interpretorului. Avantajul unei astfel de platforme față de Sistemele de Management al Învățării, ca de exemplu *Moodle*[5], constă în aceea că platforma dezvoltată nu este pasivă și interacționează cu studentul și gestionează procesul de învățare.

[5] The Moodle. Available at: https://moodle.org/ [Accesed 02.05.2025]

CONCLUZII GENERALE ȘI RECOMANDĂRI

Crearea instrumentelor pentru dezvoltarea cursurilor de instruire asistată de calculator împreună cu alte mijloace, cunoscute sub denumirea generică de e-Learning, constituie o direcție actuală și prioritară în cadrul domeniului dezvoltării moderne de software. Avansul rapid al tehnologiilor informaționale, împreună cu cerințele tot mai accentuate de digitalizare a proceselor educaționale, subliniază necesitatea elaborării unor soluții specializate, care să sprijine proiectarea, implementarea și livrarea eficientă a conținutului educațional interactiv.

În acest context, cercetarea desfășurată confirmă importanța dezvoltării unor tehnologii și limbaje specifice domeniului educațional, care să permită nu doar automatizarea procesului de creare a cursurilor, ci și adaptarea acestora la diverse scenarii educaționale și tehnologice. Rezultatele obținute reflectă contribuția lucrării la consolidarea unei baze metodologice și instrumentale în domeniul instruirii asistate de calculator.

Studiul rezultatelor obținute permite formularea următoarelor concluzii generale:

- Analiza instrumentelor existente pentru crearea cursurilor de instruire asistată de calculator, precum și a limbajelor de programare utilizate în cadrul acestor sisteme, evidențiază existența unui număr semnificativ de resurse și soluții tehnice disponibile. Cu toate acestea, pe fondul progresului continuu al tehnologiilor informaționale și al diversificării cerințelor, se conturează o nevoie tot mai accentuată pentru instrumente mai flexibile, mai accesibile și adaptate specificului domeniului educațional. În acest context, dezvoltarea de limbaje declarative specializate (domain-specific languages) devine o direcție importantă pentru a susține proiectarea eficientă și personalizată a cursurilor de instruire asistată de calculator.
- Tehnologia dezvoltată pentru proiectarea programelor cursurilor de instruire asistată de calculator, care se bazează pe situații instructive stereotipe, a constituit un cadru metodologic solid pentru elaborarea unui limbaj de programare dedicat acestui domeniu. Această abordare a permis formalizarea structurii cursurilor în termeni pedagogici standardizați și a condus la definirea unor constructe specifice, relevante pentru descrierea logicii instruirii. Ca rezultat, a fost posibilă dezvoltarea unui mediu de programare destinat autorilor de cursuri, care facilitează definirea și implementarea scenariilor educaționale într-un mod eficient și intuitiv. Acest mediu oferă suport pentru generarea automată a conținutului, reducând complexitatea tehnică și contribuind la creșterea calității procesului de instruire asistată de calculator.

- În cadrul cercetării a fost elaborat limbajul de descriere a cursurilor de instruire asistată de calculator, denumit MoCo, care asigură o reprezentare adecvată, structurată și expresivă a materialului didactic în format digital. Printre caracteristicile definitorii ale acestui limbaj se numără apropierea semantică maximă de situațiile reale din domeniul educațional, ceea ce facilitează utilizarea sa de către specialiștii în instruire. Totodată, MoCo se remarcă prin caracterul său intuitiv, care reduce bariera de acces tehnologic pentru autori, prin suportul pentru dezvoltarea cursurilor adaptive și prin aplicarea unor constrângeri minime asupra conținutului didactic. Aceste proprietăți contribuie la crearea de cursuri flexibile, accesibile și relevante din punct de vedere didactic.
- Descrierile cursurilor elaborate cu ajutorul limbajului MoCo constituie baze de cunoștințe, exprimate într-un limbaj orientat pe frame-uri, special conceput pentru reprezentarea structurilor specifice instruirii asistate de calculator. Limbajul MoCo permite modelarea logicii de instruire într-un mod expresiv și formalizat, facilitând astfel procesul de programare a cursurilor. Interpretorul MoCo acționează ca un motor inferențial, având rolul de a interpreta aceste baze de cunoștințe și de a genera comportamentul corespunzător al aplicației în baza regulilor și relațiilor definite în timpul descrierii cursului.
- Pe baza translatorului limbajului MoCo, a fost elaborat în premieră un prototip funcțional al unei platforme de tip *no-code* pentru dezvoltarea cursurilor de instruire asistată de calculator. Această platformă permite autorilor să creeze conținut educațional complex fără a apela la programare, facilitând astfel accesul larg la tehnologii educaționale moderne.

Direcțiile prioritare recomandate pentru viitoarele cercetări și dezvoltări în domeniul instruirii asistate de calculator ar putea include:

- Dezvoltarea viitoare a instrumentarului propus, mai concret elaborarea unei interfețe performante, atât pentru utilizator, cât și pentru autorul cursurilor (*eng*. authoring tools), care să asigure o interacțiune intuitivă, eficientă și adaptabilă cu sistemul. O astfel de interfață ar permite proiectarea ușoară a conținutului educațional, fără a necesita cunoștințe tehnice avansate, facilitând astfel implicarea directă a cadrelor didactice în procesul de creare a cursurilor de instruire asistată de calculator și sporind aplicabilitatea practică a soluției dezvoltate.

- O altă direcție importantă pentru dezvoltarea ulterioară constă în extinderea limbajului de programare propus cu mijloace specifice de integrare a facilităților multimedia în cadrul programelor cursurilor de instruire asistată de calculator. Includerea nativă a suportului pentru elemente precum audio, video, animații și interacțiuni vizuale ar permite crearea unor materiale educaționale mai dinamice, atractive și eficiente din punct de vedere pedagogic, contribuind la sporirea gradului de implicare a cursanților și la îmbunătățirea procesului de învățare asistat de calculator.
- Pentru a îmbunătăți instrumentele de creare a cursurilor de instruire asistată de calculator, se recomandă integrarea facilităților oferite de sistemele de inteligență artificială, în special a modelelor lingvistice de mari dimensiuni (Large Llanguage Model). Aceste modele pot asista autorii în generarea și personalizarea conținutului educațional, optimizând astfel procesul de dezvoltare a cursurilor. Implementarea se poate realiza prin utilizarea unor interfețe de programare a aplicațiilor (API) specializate, care permit integrarea eficientă a LLM-urilor în platformele de e-learning. Un exemplu relevant este utilizarea API-urilor oferite de platforme precum OpenAI, care permit dezvoltatorilor să încorporeze modele lingvistice avansate în aplicațiile lor. Aceste API-uri facilitează generarea automată de texte, răspunsuri la întrebări și alte funcționalități utile în context educațional. [112, 113]

Bibliografia

[1] GROS, B., GARCÍA-PEÑALVO, F.J. Future Trends in the Design Strategies and Technological Affordances of E-Learning. In: „*Learning, Design, and Technology*". Springer, Cham. 2016. pp. 1-23. Online ISBN: 978-3-319-17727-4 [citat 16.12.2024]; Accesibil la adresa: https://doi.org/10.1007/978-3-319-17727-4_67-1

[2] MIAO, Y., SODHI, T., BROUNS, F., SLOEP, P., KOPER, R. Bridging the Gap between Practitioners and E-Learning Standards: A Domain-Specific Modeling Approach. În: „*Technologies Across Learning Contexts"*. EC-TEL 2008. Lecture Notes in Computer Science, vol 5192. Springer, Berlin, Heidelberg. pp 284–289. ISBN978-3-540-87604-5. [citat 16.12.2024]; Accesibil la adresa: https://doi.org/10.1007/978-3-540-87605-2_32

[3] DALZIEL, James. și al. The Larnaca Declaration on Learning Design. În: „*Journal of Interactive Media in Education*". 2016(1): 7, pp. 1–24, [citat 16.12.2024]; Accesibil la adresa: http://dx.doi.org/10.5334/jime.407 https://www.lamsfoundation.org/ https://www.1edtech.org/standards/learning-design

[4] [SEBASTIÁN, G., TESORIERO, R., GALLUD, J.A. A domain specific language notation for a language learning activity generation tool. În: "*Multimed Tools Appl"* 80. 2021. pp. 36275–36304. [citat 16.12.2024]; Accesibil la adresa: https://doi.org/10.1007/s11042-021-11296-y

[5] **ЛАТУЛ, Г.В**. Опыт разработки автоматизированного обучающего курса по языку ПЛ/1. În: *Применение автоматизированных обучающих систем в учебном процессе. Тезисы докладов межзональной научно-методической конференции*. Минск, БГУ, 1984, p. 133.

[6] КРАСНОВ, М.А., **ЛАТУЛ, Г.В**. РЫНГАЧ, В.Д. Технология и инструментальная система генерации обучающих программ. În: *Материалы IV Всесоюзного семинара Разработка и применение программных средств ПЭВМ в учебном процессе*. Москва, ИПИАН, 1988, pp.77-78.

[7] **ЛАТУЛ, Г.В**., РЫНГАЧ, В.Д. Технология проектирования и автоматическая генерация компьютерных учебных курсов. În*: Методы и системы технической диагностики. Часть I. Экспертные обучающие системы. Межвузовский сборник научных трудов*. Саратов, СГУ, 1989, pp. 54-57. ISBN 5-292-00143-0.

[8] РЫНГАЧ, В.Д., **ЛАТУЛ, Г.В**., СОКОЛ, В.В. Информационно-обучающая среда „Программирование на Турбо Си". În: *Conferința științifică a corpului didactic -științific. Tezele referatelor. Totalurile activității științifice în anii 1991/92*. Chișinău. Ch.: Centrul ed. al USM, 1992, P. 53.

[9] КРАСНОВ, М.А., **ЛАТУЛ, Г.В**., ЛЫСИКОВ, А.Ф., РЫНГАЧ, В.Д., ЛЕМПЕРТ, Р.М. Технологические средства генерации обучающих программ. În: *Исследования по прикладной математике и информатике*. Chișinău, Știința, 1990, pp. 130-138.

[10] RYNGATCH, V., **LATUL, GH**. The informative-training medium „Turbo C Programming" În: *Computer Technologies in Education. Proceedings of the International Conference on Computer Technologies in Education (ICCTE'93).* Kiev, 1993, pp. 198-199.

[11] **ЛАТУЛ, Г.В**. Генерация программ в диалоге с ЭВМ. În: *Conferința științifică a corpului didactico-științific. Tezele referatelor. Totalurile activității științifice în anii 1991/92*. Chișinău. Ch.: Centrul ed. al USM, 1992, P. 55.

[12] **ЛАТУЛ, Г.В**. Средства для автоматизированной генерации компьютерных учебных курсов. În: Materialele conferinței tehnico-științifice. „Informatica și tehnica de calcul. Academia de Științe, Chișinău, 1993, pp. 83-85.

[13] **LATUL**, GH. The media for automatic design of computer assisted courses. În: *Computer Technologies in Education. Proceedings of the International Conference on Computer Technologies in Education. Part 2. (ICCTE'94).* Ucraina, Crimeea, 1994, pp. 159-160.

[14] **ЛАТУЛ, Г.В**. Интерпретация обучающих курсов в системе автоматизированной генерации программ. În: *Materialele conferinței corpului didactico-științific. Bilanțul activității științifice a U.S.M. pe anii 1993-1994. Științe naturale*. Chișinău. Ch.: Centrul ed. al USM, 1995, p. 60.

[15] **LATUL, GH**. Modelul aprecierii cunoștințelor în cursuri asistate de calculator. În: *Anale științifice ale Universității de Stat din Moldova. Seria „Științe fizico-matematice”*. Chișinău. Ch.: Centrul ed. al USM, 1998, pp. 36-41. ISBN 9975-917-04-6

[16] **ЛАТУЛ, Г**. Модели обучения для компьютерных учебных курсов. În: *Conf. corpului didactico-științific Bilanțul activității științifice a USM pe anii 1996/97. Rezumatele comunicărilor. Științe naturale*. Chișinău. Ch.: Centrul ed. al USM, 1998. pp. 70.

[17] **ЛАТУЛ, Г**. Модели урока при проектировании компьютерных учебных курсов. În: *Anale științifice ale Universității de Stat din Moldova. Seria „Științe fizico-matematice”*. Chișinău. Ch.: Centrul ed. al USM, 1999, pp. 252-256. ISBN 9975-917-26-7

[18] **ЛАТУЛ, Г**. Инструментарий для создания компьютерных учебных курсов. În: *Tehnologii avansate în pragul secolului XXI. Materialele conferinței științifico-practice*. Chișinău, Î.E.P. Știința, 2000, pp. 22-24. ISBN 9975-67-215-9.

[19] **LATUL, GH**., CARELOVA, L. Limbajul de descriere a situațiilor instructive și analizorul lui. În: *Conferința corpului didactico-științific. Bilanțul activității științifice a USM pe anii 1998/99. Rezumatele comunicărilor. Științe fizico-matematice*. Chișinău. Ch.: Centrul ed. al USM, 2000, pp. 117-118. ISBN 9975-917-58-5.

[20] **LATUL, GH**. Generarea paginilor Web după descrierea scenariilor de instruire. În: *Conferința corpului didactico-științific. Bilanțul activității științifice a USM pe anii 1998/99. Rezumatele comunicărilor. Științe fizico-matematice*. Chișinău. Ch.: Centrul ed. al USM, 2000, pp 119-120. ISBN 9975-917-58-5.

[21] **ЛАТУЛ, Г**. В. Использование шаблонов при создании компьютерных учебных курсов. În: *Conferința corpului didactico-științific. Bilanțul activității științifice a USM în anii 2000 2003. Rezumatele comunicărilor. Științe fizico-matematice*. Chișinău. Ch.: Centrul ed. al USM, 2003, pp. 188-189. ISBN 9975-70-265-1.

[22] **ЛАТУЛ, Г**. Проблемы организации анализа ответов обучаемых в компьютерных учебных курсах. În: *Analele științifice ale Universității de Stat din Moldova. Seria „Științe fizico-matematice”*. Chișinău. Ch.: Centrul ed. al USM, 2004. pp. 132-136. ISSN 1811-2641

[23] **ЛАТУЛ, Г**. Язык описания компьютерных учебных курсов. În: *Analele științifice ale Universității de Stat din Moldova. Seria „Științe fizico-matematice”*. Chișinău. Ch.: Centrul ed. al USM, 2004.pp. 129-131. ISSN 1811-2641

[24] **ЛАТУЛ Г**. Проблемы проектирования компьютерных учебных курсов при помощи шаблонов. În: *Conferința Internațională Tendințele de Dezvoltare a Societății Informaționale Academiei de Studii Economice*. Chișinău, 2004. pp. 180-181. ISBN 9975-75-265-9

[25] **LATUL, GH**. Modelul aprecierii cunoștințelor în cursurile asistate de calculator. În: *Conferința Științifico-practică „Problemele actuale ale dezvoltării social – economice a Republicii Moldova”, 18-19 noiembrie, 2005*. Chișinău, Ch.: Centrul ed. al USM, 2006. pp.146-151.

[26] **LATUL, GH**. Perspectivele instruirii la distanță în Republica Moldova. În: *Învățământul superior și cercetarea – piloni ai societății bazate pe cunoaștere. Conferința științifică internațională. Rezumatele comunicărilor Științe Reale*. Chișinău. Ch.: Centrul ed. al USM, 2006. pp. 53-54. ISBN 978-9975-70-670-4.

[27] BRAGARU. T., **LATUL. GH**. Crearea cursurilor de instruire la distanta. In: *Conference „Mathematics & Information Technologies: Research and Education" (MITRE-2008). Chișinău, October 1-4 October, 2008.* Abstracts. Chișinău, Ch.: Centrul ed. al USM, 2008, pp. 71-72. ISBN 978-9975-70-752-4

[28] CĂPĂȚÂNĂ, GH**., LATUL, GH**. *Sisteme inteligente de e-learning.* În: *Mathematics & Information Technologies: Research and Education (MITRE 2011)*. Abstracts. Chișinău. Ch.: Centrul ed. al USM, 2011, pp. 157 -158, ISBN 978-9975-71-144-9.

[29] **ЛАТУЛ, Г**. Построение генератора компьютерных учебных курсов. În: *Integrare prin cercetare și inovare, conferință științifică națională cu participare internațională Rezumatele ale comunicărilor. Științe ale naturii. Științe exacte*. Chișinău, Ch.: Centrul ed. al USM, 2014, pp. 150 -151. ISBN 978-9975-71-573-7.

[30] CĂPĂȚÂNĂ, GH., **LATUL, GH**., PERETEATCU, S. Analiza răspunsurilor grafice în cursuri de instruire asistate de calculator. În: *International Conference „Mathematics & Information Technologies: Research and Education" (MITRE - 2015)*. Chișinău. Ch.: Centrul ed. al USM, 2015, P. 109, ISBN 978-9975-71-678-9.

[31] CĂPĂȚÂNĂ, GH., **LATUL GH**. Frame using in the language for computer assisted courses programming. În: *Mathematics & Information Technologies: Research and Education (MITRE 2016). International conference.* Abstracts. Chișinău. Ch.: Centrul ed. al USM, 2016, pp. 77, ISBN 978-9975-71-794-6.

[32] **LATUL, GH**. Model of authoring system of computer assisted courses generator. In: *Mathematics & Information Technologies: Research and Education (MITRE 2023). International conference*. Chișinău. Ch.: Centrul ed. al USM, 2023, pp. 81-82, ISBN 978-9975-62-535-7

[33] **LATUL, GH**. Modelul conceptual și arhitectura platformei no-code de generare automată a cursurilor de instruire asistată de calculator. În: *Intellectus*, nr. 1, 2024, p. 147 – 156. ISSN 1810-7087, Tip B. [citat 15.07.2024]; Accesibil la adresa:

https://agepi.gov.md/ro/intellectus/intellectus-1-2024/modelul-conceptual-%C8%99i-arhitectura-platformei-no-code-de-generare

[34] **LATUL, GH**. Reprezentarea cunoștințelor în cursurile de instruire asistată de calculator. În: *Acta et commentationes, seria Științe ale Educației*, nr. 1(35), 2024, p. 103-108. ISSN 1857-0623, E-ISSN 2587-3636, Tip B. [citat 15.07.2024]; Accesibil la adresa:

https://revistaust.upsc.md/index.php/acta_educatie/article/view/979/958

[35] ЗАХАРОВА, И. Г. Информационные технологии в образовании. Москва. Издательский центр Академия, 2003, 192 p.

[36] ЗАЙЦЕВА, Л.В., НИЦЕЦКИЙ, Л.В., НОВИЦКИЙ, П.П., СУРКОВСКИЙ, У.А, ШИТИКОВ, В.С. Автоматизированная обучающая система КОНТАКТ на базе ЕС ЭВМ. Версия „КОНТАКТ ОС". Рига, РПИ, 1989, p. 57.

[37] ДОБРЫНИНА, Л.Б., ЦЕНТОВСКАЯ, Н.Г., АХМИРОВА, Р.Р. și alt. Автоматизированная обучающая система КУРСОР. În: *Материалы научно-практической конференции „Роль университетов в подготовке кадров высшей квалификации"* Москва. ВДНХ СССР, 1983, pp. 156-157.

[38] ДОВГЯЛЛО, А.М., ИЩЕНКО, Е.Л. Обучающие системы нового поколения. În: *УСАМ* Киев, 1988, nr1, pp. 83-86.

[39] СОЛОВОВ, А.В. Электронное обучение как новая парадигма образования 21-го века. În: *Международная научно-техническая конференция проблемы развития одаренной молодежи в информационном обществе. Самарский государственный аэрокосмический университет*, Россия, Самара, 2006.

[40] НОВИКОВ, В.А. și alt. Программное обеспечение автоматизированных обучающих систем. Минск, Изд-во БГУ, 1999. 203р.

[41] САМОФАЛОВ, К.Г., СЛИПЧЕНКО, В.Г., НОВИКОВ, В.А., КОРНЕЙЧУК, В.М., СОРОКО В.Н. Обучающие машины, системы и комплексы. Справочник. Киев, Вища школа, 1987, 303р.

[42] МОРЕВ, И. А. Образовательные информационные технологии. Часть 1. Обучение: Учеб. пособие. Владивосток, Изд-во Дальневосточного университета, 2004. 162 р.

[43] СОЛОВОВ, А.В. Проектирование компьютерных систем учебного назначения: Учебное пособие. Самара, СГАУ, 1995. 138р.

[44] КРИВИЦКИЙ, Б., Х. Обучающие компьютерные программы: психология разработки преподавателями обучающих курсов в ACO. Educational Technology & Society 10(3). 2007. PP. 395-406. ISSN 1436-4522. [citat 1.10.2024].; Accesibil la adresa:

hhttps://cyberleninka.ru/article/n/obuchayuschie-kompyuternye-programmy-psihologiya-razrabotki-prepodavatelyami-obuchayuschih-kursov-v-aso/viewert

[45] JOHN, E. Hilbert. Learning Asymetrix Toolbook II Assistant 6.5: A Self-Paced Guide – Teletraining Institute, Incorporated, 1998, ISBN 1893485013, 9781893485013, .235p. [citat 1.10.2024].; Accesibil la adresa: https://books.google.md/books/about/Learning_Asymetrix_Toolbook_II_Assistant

[46] CHEMEY, A. Creating AICC and SCORM-Compliant Content with with Authorware 7 [citat 1.10.2024]; Accesibil la adresa: *http://www.adobe.com/devnet/authorware/articles/compliant_files.html*

[47] ГУЛЬТЯЕВ, А. К. Macromedia Authorware 6.0. Разработка мультимедийных учебных курсов. Санкт-Петербург: КОРОНА-принт, 2002, 400 р. ISBN 5-7931-0205-1

[48] Macromedia Authorware 7.0. [citat 10.10.2024]; Accesibil la adresa: https://macromedia-authorware.software.informer.com/7.0/

[49] КОЛЕСНИКОВ, К. А. "Net Школа 3.0": создание единого информационного пространства, 2010. [citat 12.06.2024]; Accesibil la adresa: *http://www.thg.ru/education/net_school/index.html*

[50] БЕСПАЛЬКО, В.П. Образование и обучение с участием компьютеров (педагогика третьего тысячелетия). Москва, Издательство Московского психолого-социального института; Воронеж, Издательство НПО „МОДЭК”, 2004, 352 р. ISBN 5-89395-384-3

[51] АТКИСОН, Р., БАУЭР, Г., КРОТЕРС, Э. Введение в математическую теорию обучения. Москва: Мир, 1980. 487 р.

[52] GAINDRIC, C. Luarea deciziilor, metode și tehnici. Chișinău: Știința, 1998. 164 p.

[53] СВИРИДОВ, А.П. Основы статистической теории обучения и контроля знаний. Москва: Высшая школа, 1981, 243 р.

[54] РЫБАКОВ, Ф.И. Системы эффективного взаимодействия человека и ЭВМ. Москва: Радио и связь, 1985. 263 р.

[55] АВАНЕСОВ, В. С. Композиция тестовых заданий. Учебная книга. Москва: Центр тестирования, 2002, 240 р.

[56] БАШМАКОВ, А., БАШМАКОВ, И. Разработка компьютерных учебников и обучающих систем. Москва: Информационно-издательский дом „Филинъ”, 2003. 616р. ISBN 5-9216-044-X

[57] АГАПОНОВ, С. В., ДЖАЛИАШВИЛИ, З.О. и др. Средства дистанционного обучения. Методика, технология, инструментарий. Санкт-Петербург: БХВ Петербург, 2003. 336р. ISBN 5-94157-214-7

[58] КАРЛАЩУК, В. И. Обучающие программы. Москва: Солон-Р, 2001. 528 р.

[59] ФЕДОРОВА, С.А. Методика определения эффективности автоматизированных учебных курсов. În: *Средства обучения в высш. и сред. спец. школе: Обзор. информ./НИИВШ; Вып. 8.* Москва. 1987. pp. 48-54.

[60] ХОРТОН, У., ХОРТОН, К. Электронное обучение: инструменты и технологии. Москва: Кудиц-Образ, 2005. 640 р. ISBN 5-9579-0068-0

[61] Представление знаний в системах искусственного интеллекта. Москва: Знание, 1980. 300 р.

[62] Представление знаний в человеко-машинных и робототехнических системах.
Том А. Фундаментальные исследования в области представления знаний. 250 р.
Том В. Инструментальные средства разработки систем, ориентированных на знания. 327 р.
Том С. Прикладные человеко-машинные системы, ориентированные на знания. 306 р.
Москва: ВЦ АН СССР, 1984.

[63] МИНСКИЙ, М. Фреймы для представления знаний. Москва: Энергия, 1979. 152р.

[64] ОСУГА, С. Обработка знаний. Москва: Мир, 1989. 250 р.

[65] Искусственный интеллект: В 3 кн. Кн.1 Системы общения и экспертные системы. Справочник. Москва: Радио и связь, 1990. 464 р.

[66] Искусственный интеллект: В 3 кн. Кн.2 Модели и методы. Справочник. Москва: Радио и связь, 1990. 304 р.

[67] Искусственный интеллект: В 3 кн. Кн.3 Программные и аппаратные средства. Справочник. Москва: Радио и связь, 1990. 368 р.

[68] ЛИТВИНЦЕВА, Л. В. Сценарии. În: *Искусственный интеллект. – В 3-х кн. Кн. 2. Модели и методы: Справочник*. Москва. Радио и связь, 1990, pp. 304-307.

[69] BRAMBILLA, Marco, CABOT, Jordi, WIMMER, Manuel. Model-Driven Software Engineering in Practice (Synthesis Lectures on Software Engineering). Morgan & Claypool. 2017. 191 p. ISBN-10: 1627057080. ISBN-13: 978-1627057080.

[70] ZIMAREV, Alexey. Hands-On Domain-Driven Design with .NET Core: Tackling complexity in the heart of software by putting DDD principles into practice. Packt Publishing. 2019. 446 p. ISBN-10: 1788834097. ISBN-13: 978-1788834094.

[71] MICHAL, Śmiałek, NOWAKOWSKI, Wiktor. From Requirements to Java in a Snap: Model-Driven Requirements Engineering in Practice. Springer.2015. 602 p. ISBN-13: 978-3319128382.

[72] DÍAZ Vicente et. al. Progressions and Innovations in Model-Driven Software Engineering. IGI Global Scientific Publishing. 2013. 388 p. ISBN13: 9781466642171. ISBN10: 1466642173. DOI: 10.4018/978-1-4666-4217-1

[73] JEFFERY, Clinton, L. Build Your Own Programming Language: A programmer's guide to designing compilers, interpreters, and DSLs for modern computing problems. Packt Publishing. 2024. 556 p. ISBN-10: 1804618020. ISBN-13: 978-1804618028

[74] LUCAS, Oliver, JR. Domain-Specific Languages Mastery : The Power Of Custom Language. Kindle Edition. 2024. 84 p. ISBN: B0DL3CSWWK

[75] FOWLER, M. Domain-Specific Languages. Addison-Wesley Professional. 2010. 640 p. ISBN-10: 0321712943. ISBN-13: 978-0321712943

[76] WĄSOWSKI Andrzej, BERGER, Thorsten. Domain-Specific Languages: Effective Modeling, Automation, and Reuse. Springer.2024 . 504 p. ISBN-10: 3031236688. ISBN-13: 978-3031236686.

[77] KOSAR, Tomaž, MARTINEZ, López Pablo E., BARRIENTOS, Pablo, MERNIK, Marjan. A preliminary study on various implementation approaches of domain-specific language. In Information and Software Technology [On-Line], Volume 50, Issue 5, (4) 2008, pp. 390-405. [citat 1.10.2024]; Accesibil la adresa: https://doi.org/10.1016/j.infsof.2007.04.002

[78] KOSAR, Tomaž, SUDEV, Bohra, MERNIK, Marjan. Domain-Specific Languages: A Systematic Mapping Study, În *Information and Software Technology*, 71(3) 2016, pp. 77-91. [citat 1.10.2024]; Accesibil la adresa: https://doi.org/10.1016/j.infsof.2015.11.001

[79] EDET, Theophilus. Domain-Specific Languages (DSLs): Custom Languages Tailored for Specific Application Domains to Enhance Productivity (Programming Models). Kindle Edition. 2025. 619 p. ISBN-13: 979-8308400400

[80] VOELTER, Markus. DSL Engineering: Designing, Implementing and Using Domain-Specific Languages. CreateSpace Independent Publishing Platform. 2013. 558 p. ISBN-10: 1481218581. ISBN-13: 978-1481218580.

[81] FOWLER, Martin. Domain-Specific Languages. Addison-Wesley Professional. 2010. 640 p. ISBN-10: 0321712943. ISBN-13: 978-0321712943.

[82] MILETTE, Greg. Building Domain Specific Languages in Kotlin: Kotlin language features and software design patterns for creating DSLs and improving code. Independently published. 2018. 68 p. ISBN-10: 1661636160. ISBN-13: 978-1661636166.

[83] BETTINI, Lorenzo. Implementing Domain-Specific Languages with Xtext and Xtend. Kindle Edition. Packt Publishing. 2016. 428 p. ISBN: 1786464969.

[84] THAIN, DouglaS. Introduction to Compilers and Language Design. Independently published. 2020. 247 p. ISBN-13: 979-8655180260.

[85] MOGENSEN, Torben, Ægidius, Introduction to Compiler Design. Kindle Edition. Springer. 2024. 488 p. ISBN-13: 978-3031464607.

[86] AHO, Alfred V., LAM, Monica S., ULLMAN, Jeffrey D. Compilers: Principles, Techniques, and Tools. Addison Wesley. 2023. 1040 p. ISBN-10: 0321486811, ISBN-13: 978-0321486813

[87] AHO Alfred V. COMPILERS: principles, techniques, and tools. Pearson India. 2023. 224p. ISBN-10: 9357054111, ISBN-13: 978-9357054119

[88] COOPER, Keith, D., TORCZON, Linda. Engineering a Compiler. Morgan Kaufmann. 2022. 848p. ISBN-10: 9780128154120. ISBN-13: 978-0128154120.

[89] SMITH, James. From Source Code To Machine Code: Build Your Own Compiler From Scratch. 2023. 117 p. Independently published. ISBN-13: 979-8395129604.

[90] RAGHAVA, Rao, NEERUKATTU, Ravi, KUMAR, Poluru, RAJITHA Akula. Compiler Design Techniques. LAP LAMBERT Academic Publishing. 2024. 160 p. ISBN-10: 6207450094. ISBN-13: 978-6207450091.

[91] NYSTROM, ROBERT. Crafting Interpreters. Kindle Edition. Genever Benning 2021. p. 1192. ISBN-13: 978-0132634526.

[92] MAK, Ronald. Writing Compilers and Interpreters: A Software Engineering Approach. Wiley. 2009. 864 p. ISBN-10: 0470177071. ISBN-13: 978-0470177075.
[93] ЛЬЮИС, Ф., РОЗЕНКРАНЦ, Д., СТИРНЗ, Р. Теоретические основы проектирования компиляторов. Москва: Мир, 1979. 654 p. ISBN13: 978-0201144550
[94] MOORE, John, I. Compiler Design Using Java(R): An Object-Oriented Approach. Softmoore Consulting. 2022. 352 p. ISBN-10: 734139129. ISBN-13: 978-1734139129.
[95] MOGENSEN, Torben, Ægidius, Introduction to Compiler Design. Kindle Edition. Springer. 2024. 488 p. ISBN-13: 978-3031464607.
[96] SEBASTIÁN Gabriel, GALLUD Jose A., TESORIERO, Ricardo. Code generation using model driven architecture: A systematic mapping study, În: „*Journal of Computer Languages*", (56), 2021. ISSN: 2590-1184. [citat 16.12.2024]; Accesibil la adresa: https://doi.org/10.1016/j.cola.2019.100935
[97] BABKIN, Eduard, ULITIN, Boris. Ontology-Based Evolution of Domain-Oriented Languages: Models, Methods and Tools for User Interface Design in General-Purpose Software Systems. Springer. 2024. 160 p. ISBN-10: 3031422015. ISBN-13: 978-3031422010.
[98] SMITH, J. E., NAIR, R.The Architecture of Virtual Machines. În: *IEEE Computer*. 2005, *38*(5), pp. 32-38.
[99] MERNIK, M., HEERING, J., SLOANE, A. M. When and how to develop domain-specific languages. *ACM Computing Surveys*, *37*(4), 2005, pp. 316-344.
[100] KOSAR, T., ŠUDIĆ, A., BOHRA, S., MERNIK, M. Domain-Specific Languages: A Systematic Mapping Study. *Information and Software Technology*,2016, 71, 77-91. DOI: 10.1016/j.infsof.2015.11.001
[101] BOUSSE, E., MAYERHOFER, T., COMBEMALE, B., Domain-Level Debugging for Compiled DSLs with the GEMOC Studio. În *Proceedings of the Workshop on Debugging in Model-Driven Engineering (MDEbug)*. CEUR-WS Vol-2019, 2019. [citat 16.01.2025]; Accesibil la adresa: https://ceur-ws.org/Vol-2019/mdebug_3.pdf
[102] BUCCHIARONE, Antonio, CICCHETTI, Antonio, CICCOZZI, Federico et al. Domain-Specific Languages in Practice: with JetBrains MPS. Kindle Edition. 2021. 348 p. ISBN-13: 978-3030737580
[103] BETTINI, Lorenzo. Implementing Domain-Specific Languages with Xtext and Xtend. Packt Publishing. 2013. 324 p. ISBN 978-1-78216-030-4.
[104] MERNIK, Marjan. Formal and Practical Aspects of Domain-Specific Languages: Recent Developments. Information Science Reference. 2012. 651 p. ISBN-10: 1466620927. ISBN-13: 978-1466620926.
[105] BOERSMA, Meinte. Building User-Friendly DSLs. Manning. 2024. 504 p. ISBN-10: 1617296473. ISBN-13: 978-1617296475.
[106] CABALLAR, Rina Diane. Programming Without Code: The Rise of No-Code Software Development. IEEE Spectrum magazine. 11 March 2020. [citat 1.10.2024]; Accesibil la adresa:https://spectrum.ieee.org/programming-without-code-no-code-software-development
[107] HARRIS, Richard . Low code and no code app development benefits. AppDevelopers Magazine. 25 October, 2017. [citat 15.11.2024]; Accesibil la adresa: https://appdevelopermagazine.com/Low-code-and-no-code-app-development-benefits/
[108] ROKIS, K., KIRIKOVA, M. Challenges of Low-Code/No-Code Software Development: A Literature Review. In: Nazaruka, Ē., Sandkuhl, K., Seigerroth, U. (eds) Perspectives in Business Informatics Research - BIR 2022. Lecture Notes in Business Information Processing, vol 462. Springer, Cham. pp. 3-17. [citat 15.11.2024]; Accesibil la adresa: https://doi.org/10.1007/978-3-031-16947-2_1

[109] CABOT, Jordi. The low-code handbook: Learn how to unlock faster and better software development with low-code solutions. Publisher: Jordi Cabot. 2024. 113 p. ISBN-10: 9998778506. ISBN-13: 978-9998778504.
[110] CABOT, Jordi. The low-code handbook: Learn how to unlock faster and better software development with low-code solutions. Publisher: Jordi Cabot. 2024. 113 p. ISBN-10: 9998778506. ISBN-13: 978-9998778504.
[111] CĂPĂȚÂNĂ, GH. A Programming Paradigm Oriented to Families of Problems. In.: *The Scientific Bulletin Addendum No. 4/2019, The Official Catalogue of the „Cadet INOVA" Exhibition Research and Innovation in the Vision of Young Researchers The International Student Innovation and Scientific Research Exhibition „Cadet INOVA'19"*, „Nicolae Bălcescu" Land Forces Academy Sibiu, April 11-13, 2019, pp. 72-83, ISSN 2501-3157, ISSN-L 2501-3157.
[112] Ghid de utilizare a instrumentelor de inteligență artificială în procesele academice. Școala Națională de Studii Politice și Administrative (SNSPA). 16 p. [citat 15.01.2025]; Accesibil la adresa:https://snspa.ro/wp-content/uploads/2024/12/Ghid-de-utilizare-a-instrumentelor-AI-in-procesele-academice.pdf?utm_source=chatgpt.com
[113] MITTAL, Aayush. Urmărirea modelelor lingvistice mari (LLM) cu MLflow: un ghid complet. 2024. [citat 15.01.2025]; Accesibil la adresa: https://www.unite.ai/ro/urm%C4%83rirea-modelelor-de-limb%C4%83-mari-llm-cu-mlflow-un-ghid-complet/?utm_source=chatgpt.com

ANEXE

Anexa 1. Actul de implementare a rezultatelor tezei

"A P R O B"
Prim-prorector, prorector pentru activitate didactică şi studenţească a Universităţii de Stat din Moldova, doctor habilitat, profesor universitar

Otilia DANDARA

decembrie 2024

A C T

referitor la implementarea în procesul de studii la Facultatea Matematică şi Informatică a Universităţii de Stat din Moldova a rezultatelor tezei de doctor "Elaborarea metodelor şi mijloacelor pentru crearea cursurilor asistate de calculator" a lectorului universitar al Departamentului Informatică Gheorghe LATUL

Prin prezentul, subsemnaţii, decanul Facultăţii de Matematică şi Informatică doctor, conferenţiar universitar Angela NICULIŢĂ şi directorul Departamentului de Informatică doctor, conferenţiar universitar Titu CAPCELEA, confirmă implementarea în procesul de studii la Facultate a unor rezultate a tezei de doctor a domnului Gheorghe Latul lector universitar, Departamentul Informatică. „"

Obiectul implementării:

- Cursul de instruire asistat de calculator „Programarea în limbajul C" pentru studenţii anului I al Facultăţii Matematică şi Informatică;
- Limbajul descrierii situaţiilor instructive;
- Interpretorul unei submulţimi a acestui limbaj.

Disciplina şi contingentul de studenţi

Cursul de instruire asistat de calculator "Programarea în limbajul C" este utilizat la predarea disciplinei Fundamentele Programării la specialitatea Informatica şi Informatica Aplicată pe parcursul anilor de studii 2020-2024, numărul total de studenţi constituind în medie 65.

1

Metodele de aprobare:

- Cursul de instruire asistat de calculator „Programarea în limbajul C" a fost elaborat utilizând limbajul descrierii situaţiilor instructive. Funcţionarea cursului la calculator este asigurată de interpretorul corespunzător elaborat în cadrul tezei de doctor;
- În cadrul prelegerilor studenţii sunt familiarizaţi cu modalităţile de lucru cu produsul program elaborat;
- În cadrul orelor practice şi de laborator studenţii studiază limbajul de programare *C* utilizând cursul asistat de calculator, însuşind materiale didactice, îndeplinind exerciţii şi lucrări de control;

Concluzii:

- Utilizarea cursului de instruire asistat de calculator permite intensificarea procesului de studii, individualizarea şi ridicarea calităţii instruirii comparativ cu metode tradiţionale;
- Metodele de instruire propuse favorizează procesul de studii, observându-se o sporire a interesului din partea studenţilor faţă de disciplina respectivă;
- Propunem domnului Gheorghe LATUL să elaboreze lucrarea metodică de utilizare a produsului program în scopul utilizării acestuia la proiectarea şi a altor cursuri de instruire asistate de calculator pentru Facultate şi Universitate.

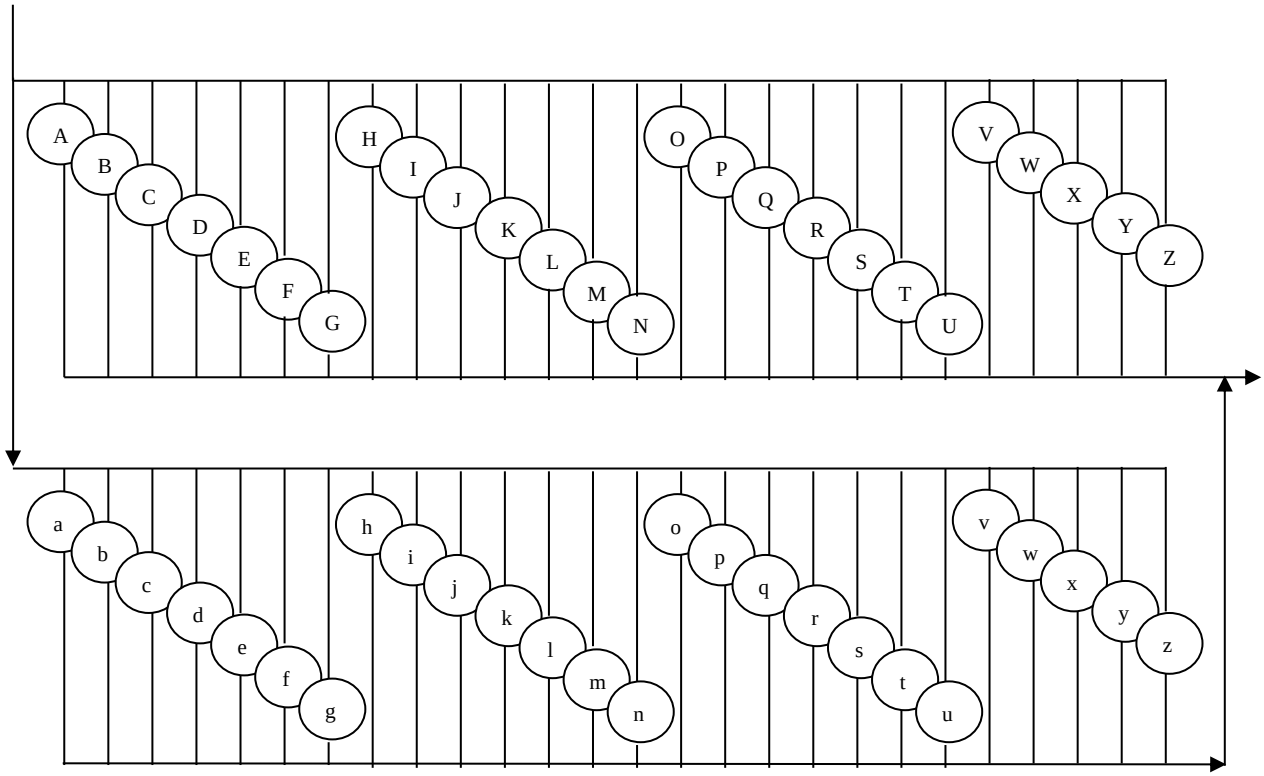
Decanul Facultatea Matematică şi Informatică, doctor, conferenţiar universitar — Angela NICULIŢĂ

Director Departament Informatică, doctor, conferenţiar universitar — Titu CAPCELEA

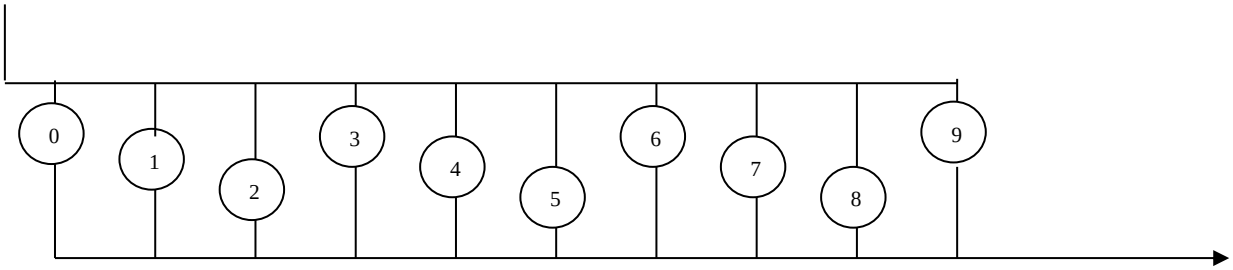
Anexa 2. Diagrame sintactice

Diagrame pentru meta denumiri *Litera, Cifra, Simbol, Număr întreg fără semn* și *Identificator* descriu formarea lexemelor din simboluri. Alte diagrame descriu formarea construcțiilor sintactice din lexeme.

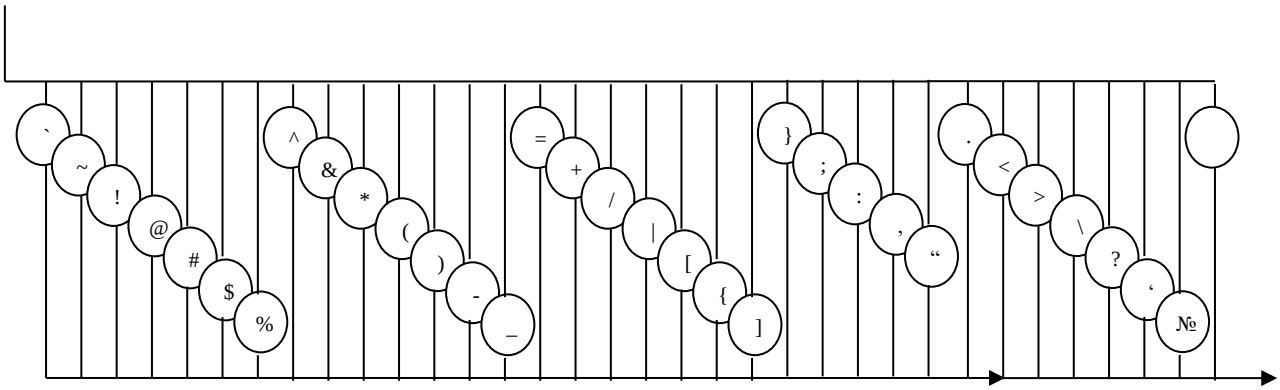
Litera



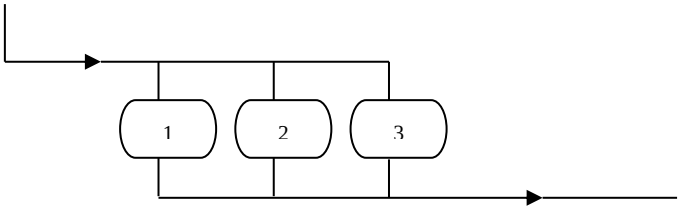
Cifra



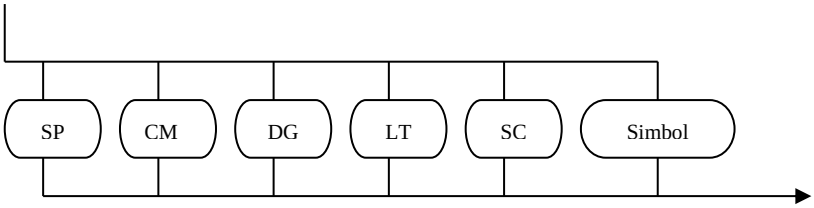
Simbol



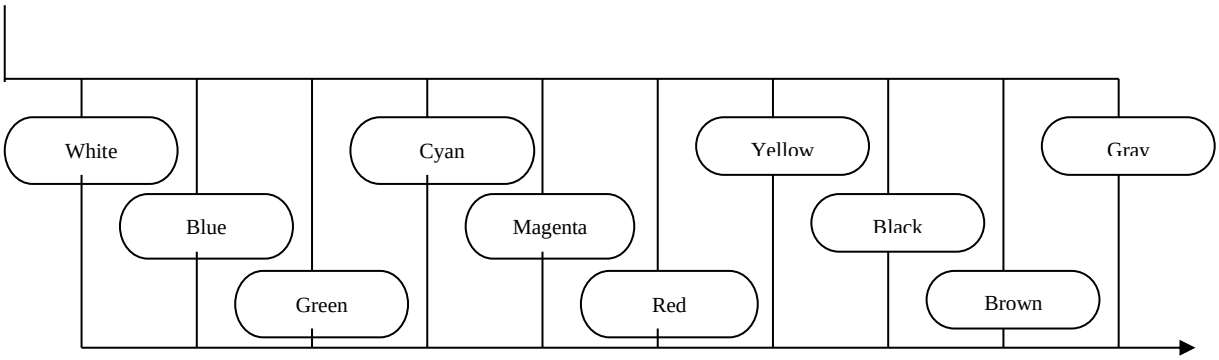
Modul de comparare



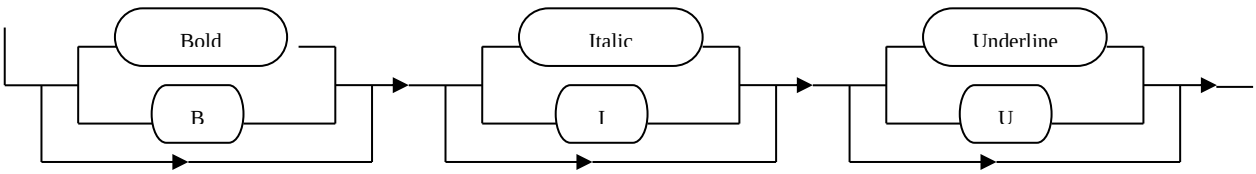
Identificatorul unui simbol



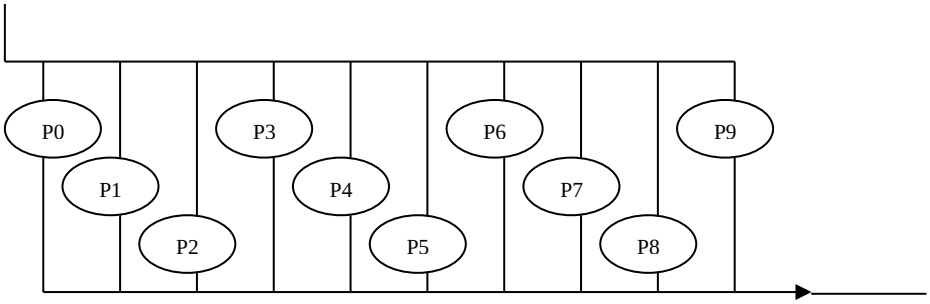
Valoarea culorii



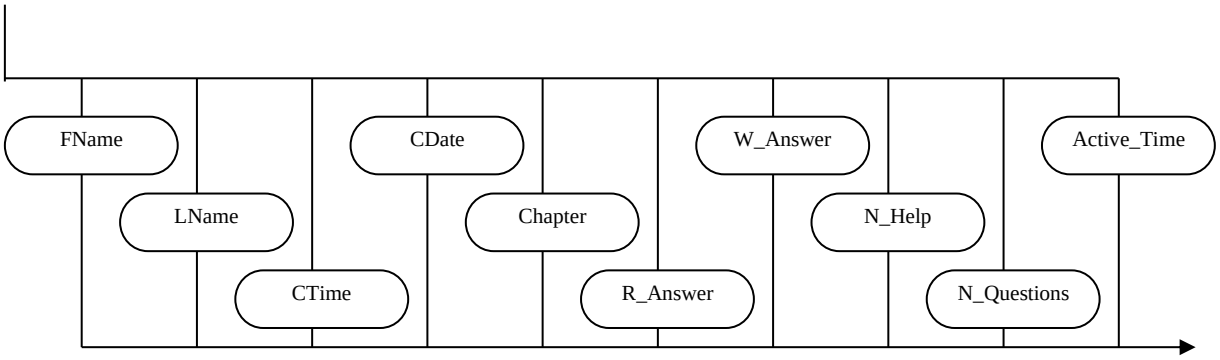
Stilul fontului



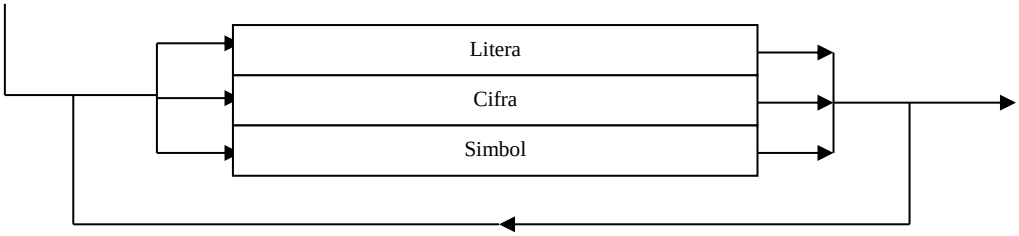
Parametrii cursului



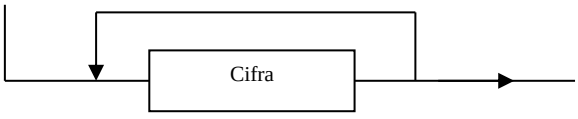
Datele din Baza de Date



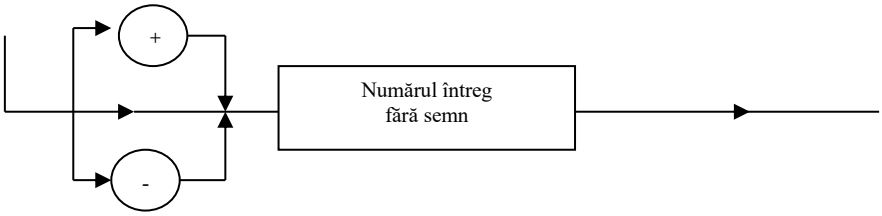
Text



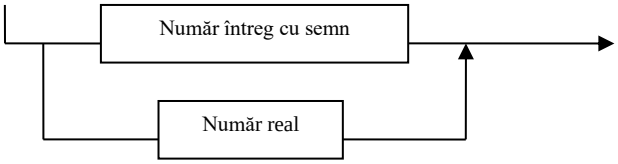
Numărul întreg fără semn



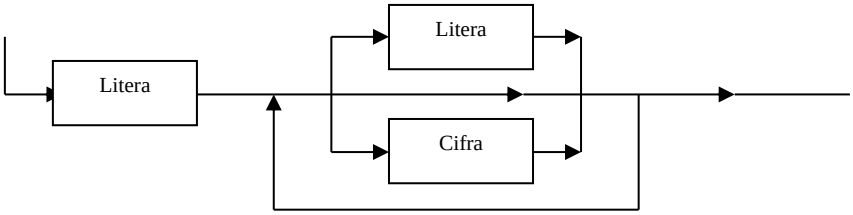
Numărul întreg cu semn



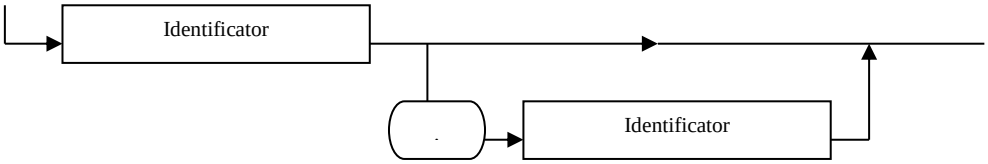
Constanta



Identificator

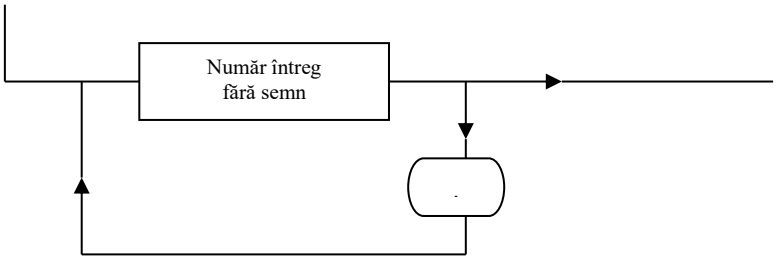


Denumirea fişierului

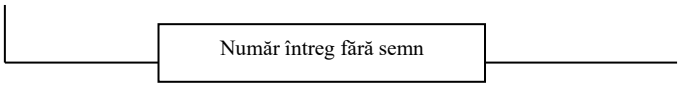


Nota, Puncte

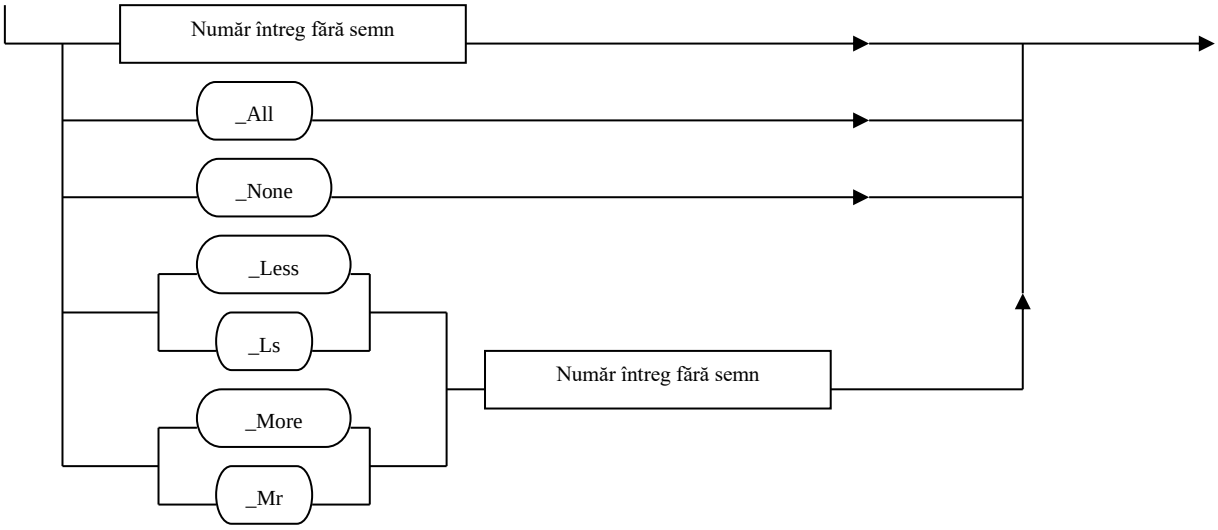
Numărul lecţiei



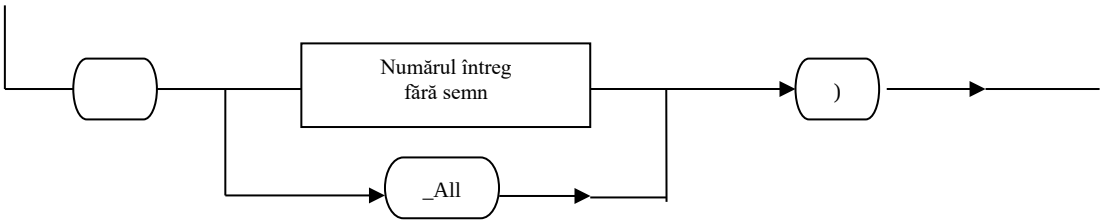
Numărul încercărilor



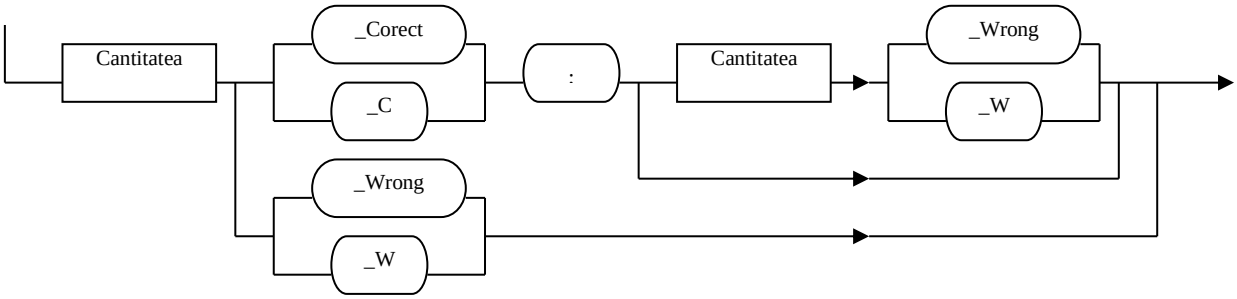
Cantitatea



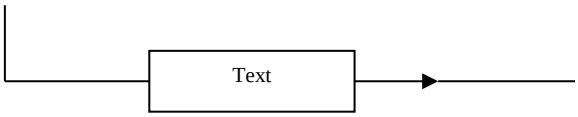
Cantitatea întrebărilor din secţie



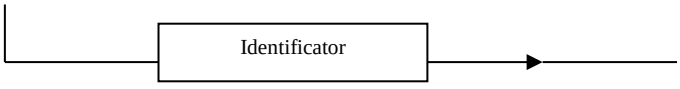
Tipul răspunsului



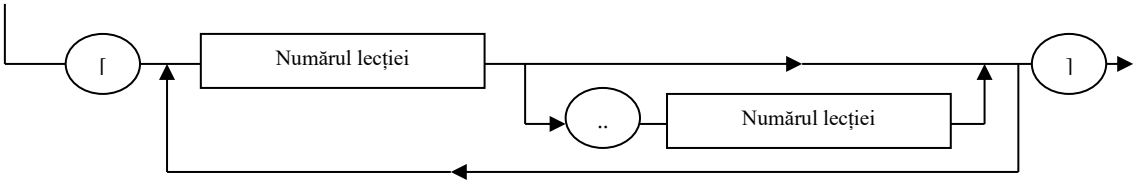
Denumirea, Denumirea lecţiei, Şablon



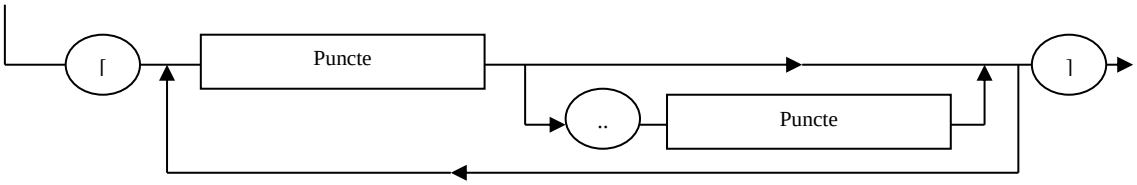
Denumirea etichetei



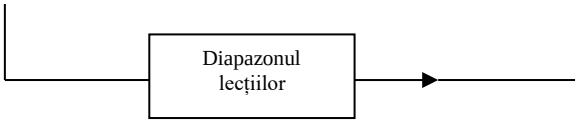
Diapazon lecţiilor



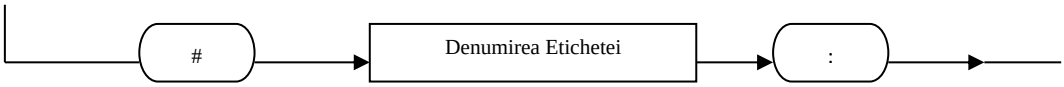
Diapazon punctelor



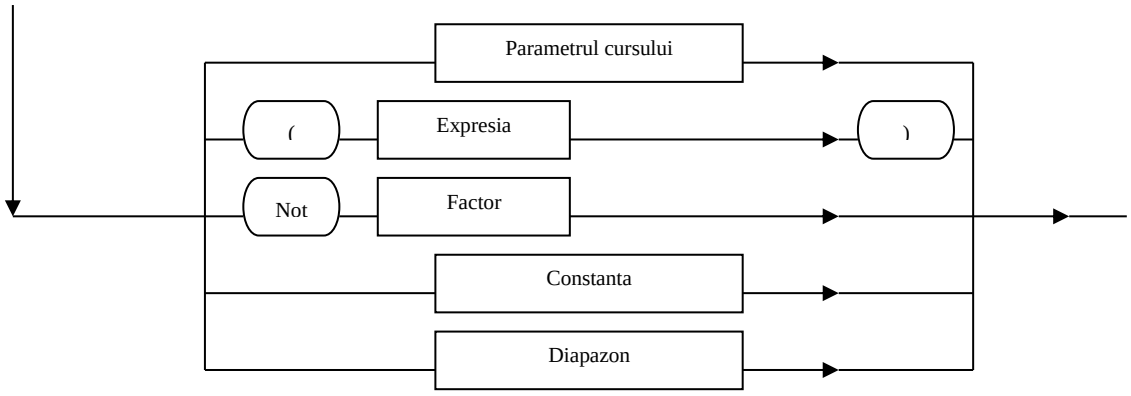
Condiţia întrării în lecţie



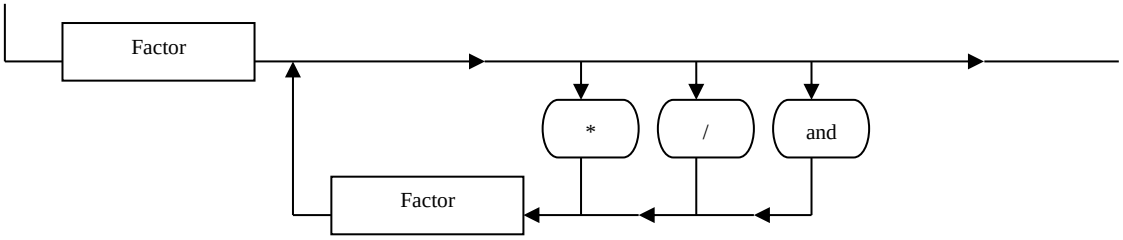
Etichetă



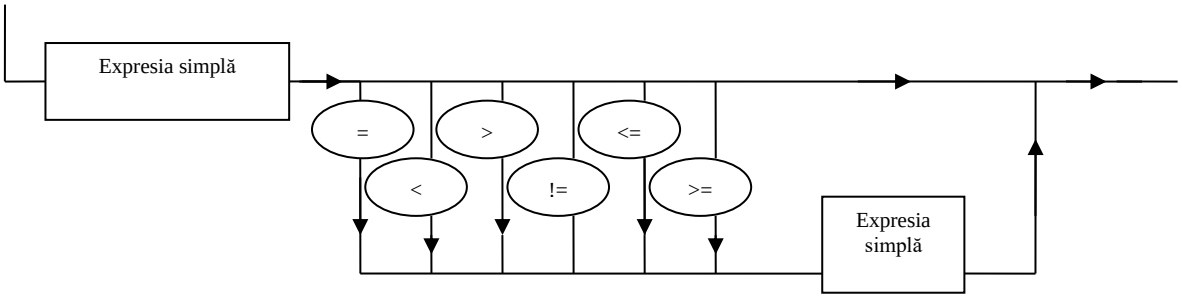
Factor



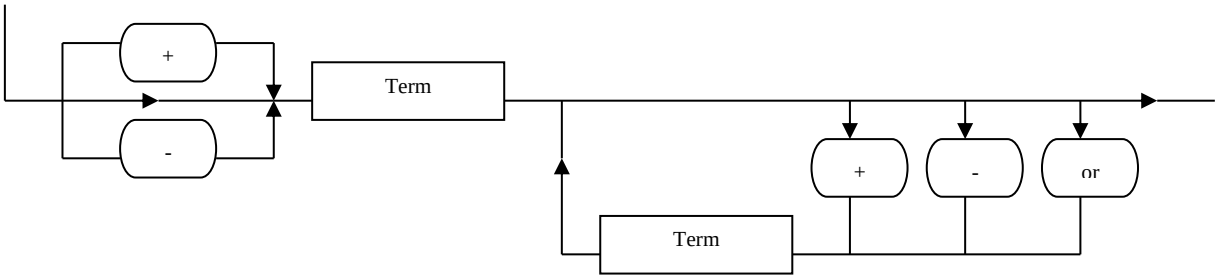
Term



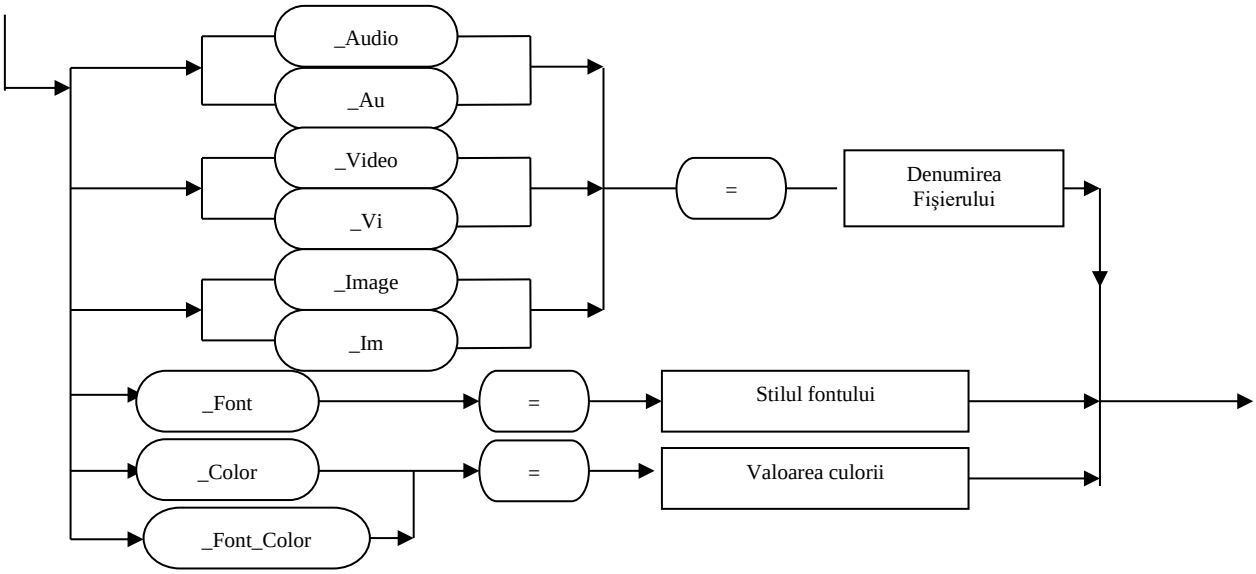
Expresia simplă



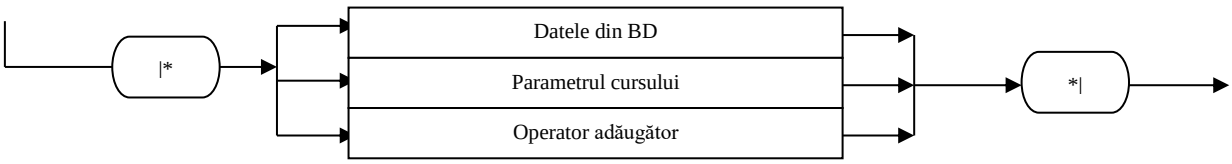
Expresia



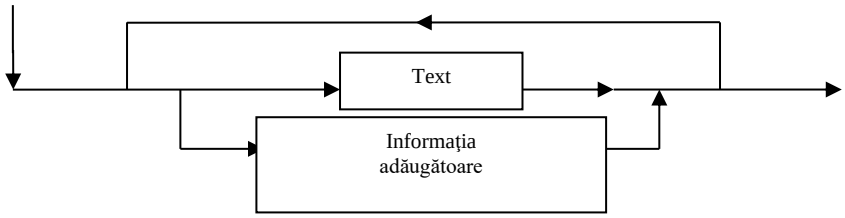
Operator adăugător



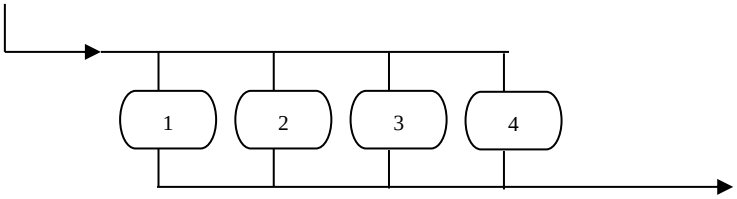
Informaţia adăugătoare



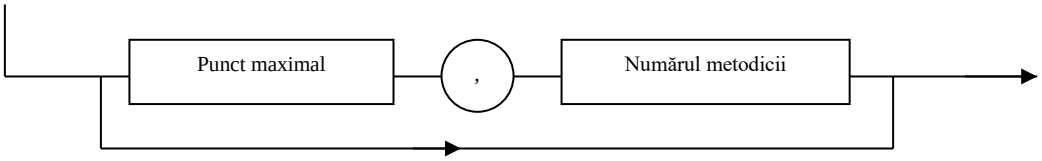
Bloc informaţional



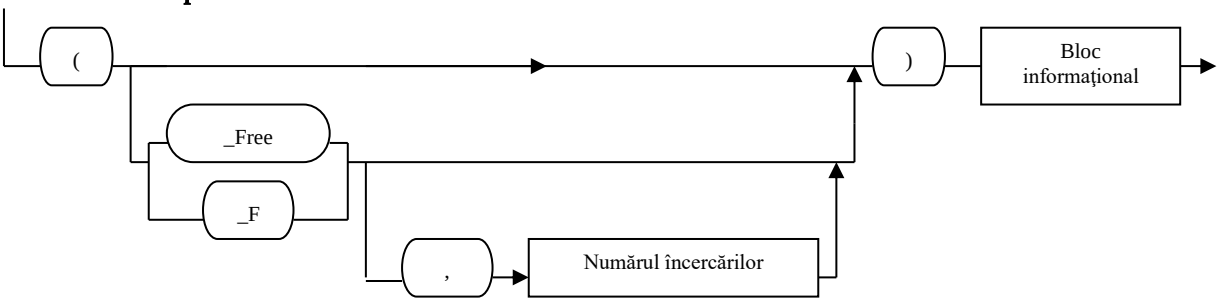
Numărul metodicii

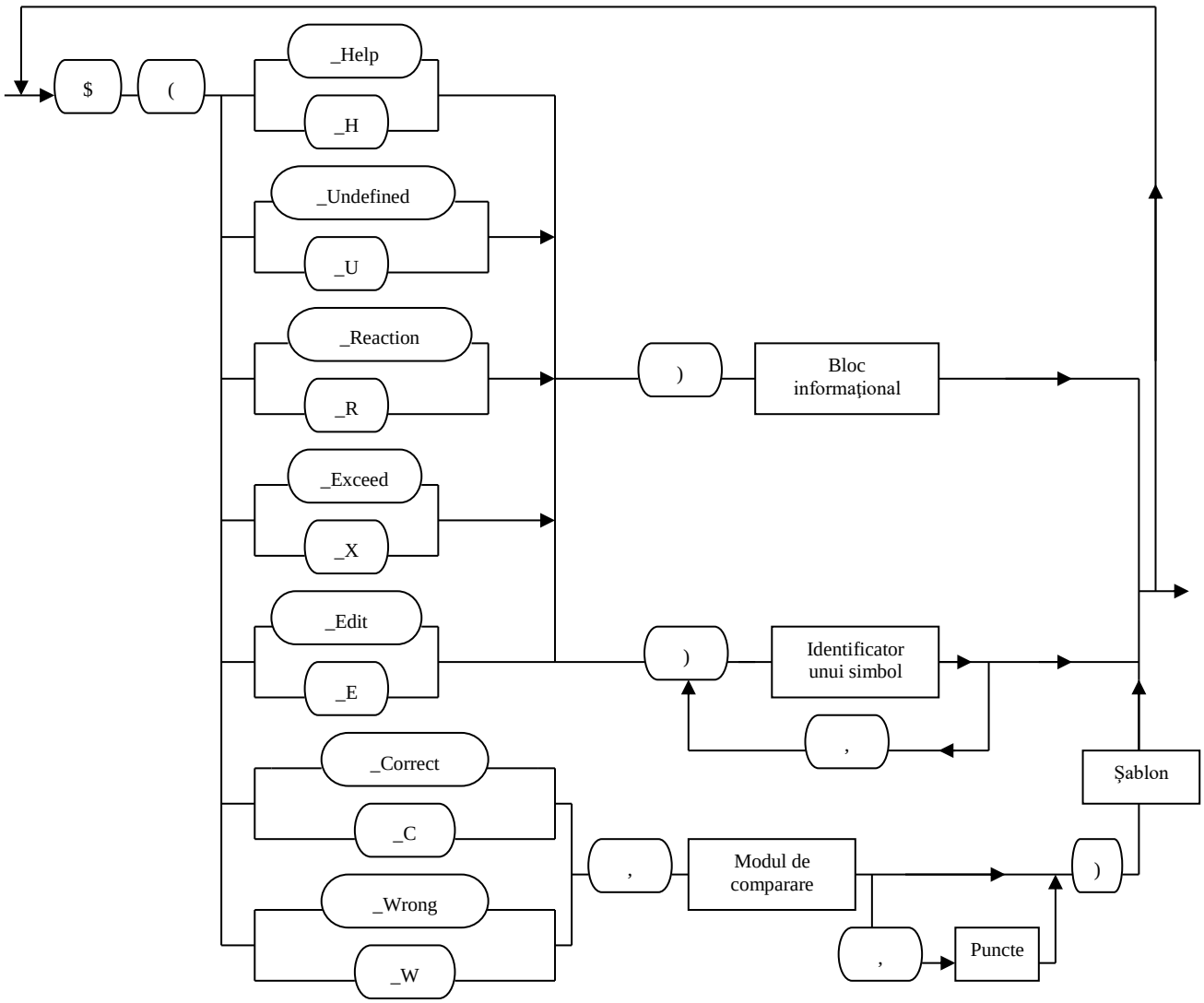


Bloc metodic

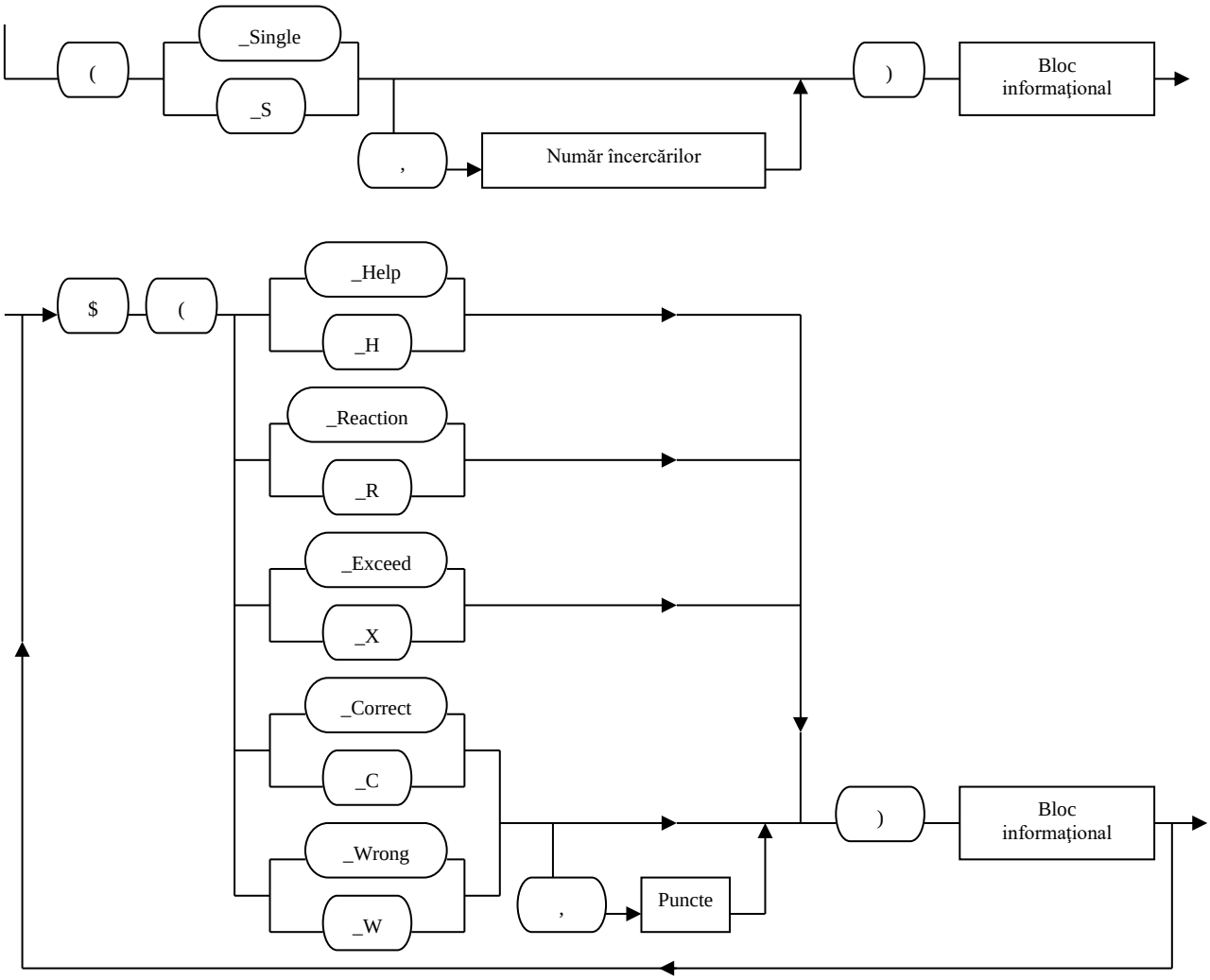


Întrebarea tip Free

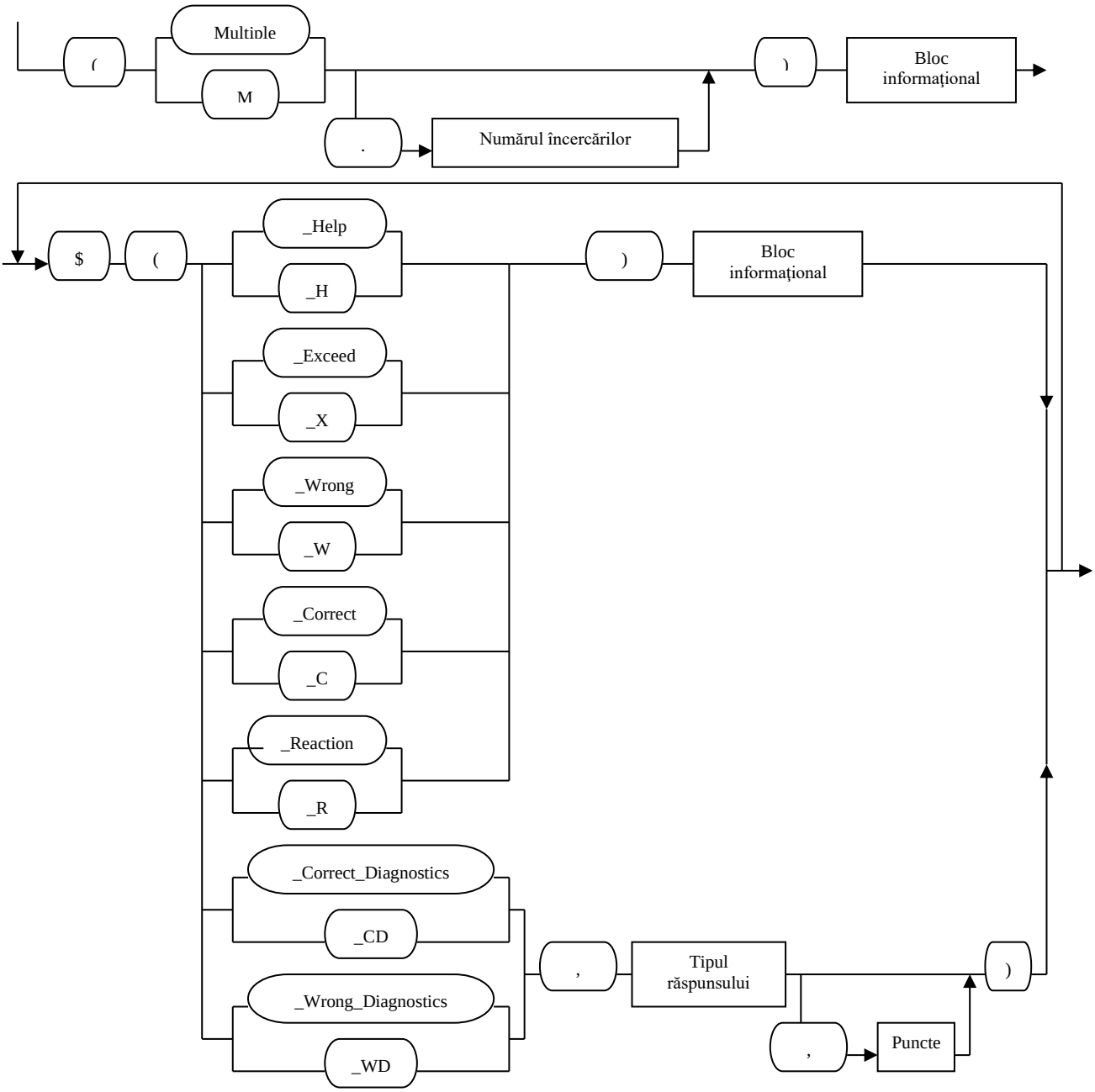




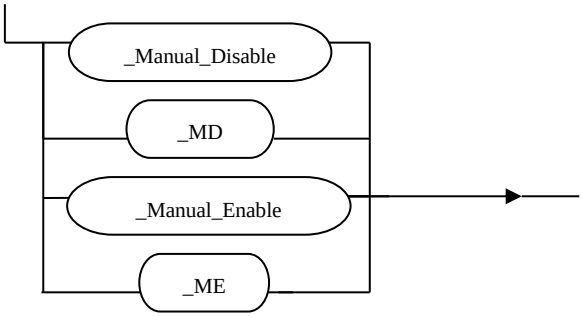
Întrebarea tip Single



Întrebarea tip Multiple

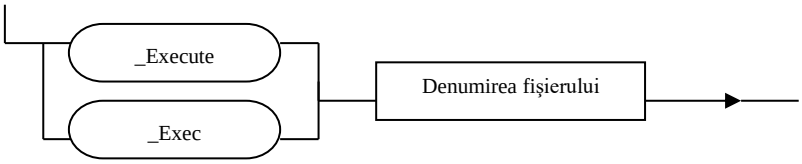


Situaţia Manual

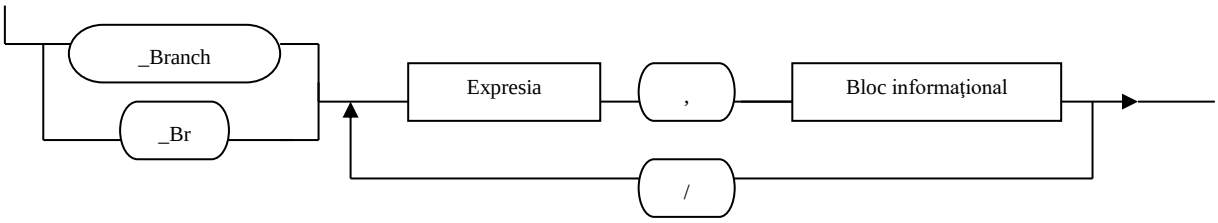


Situaţia Execute

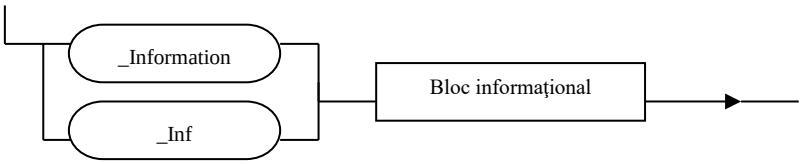
]



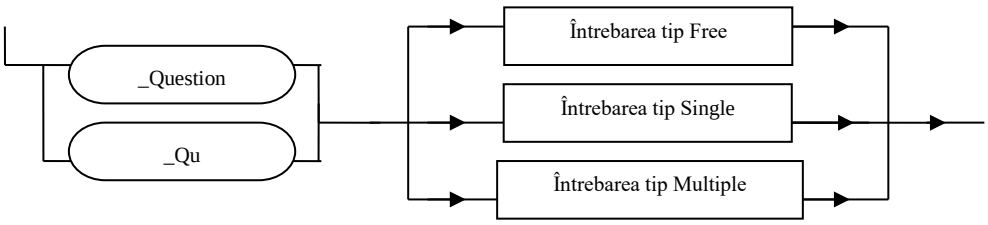
Situaţia Branch



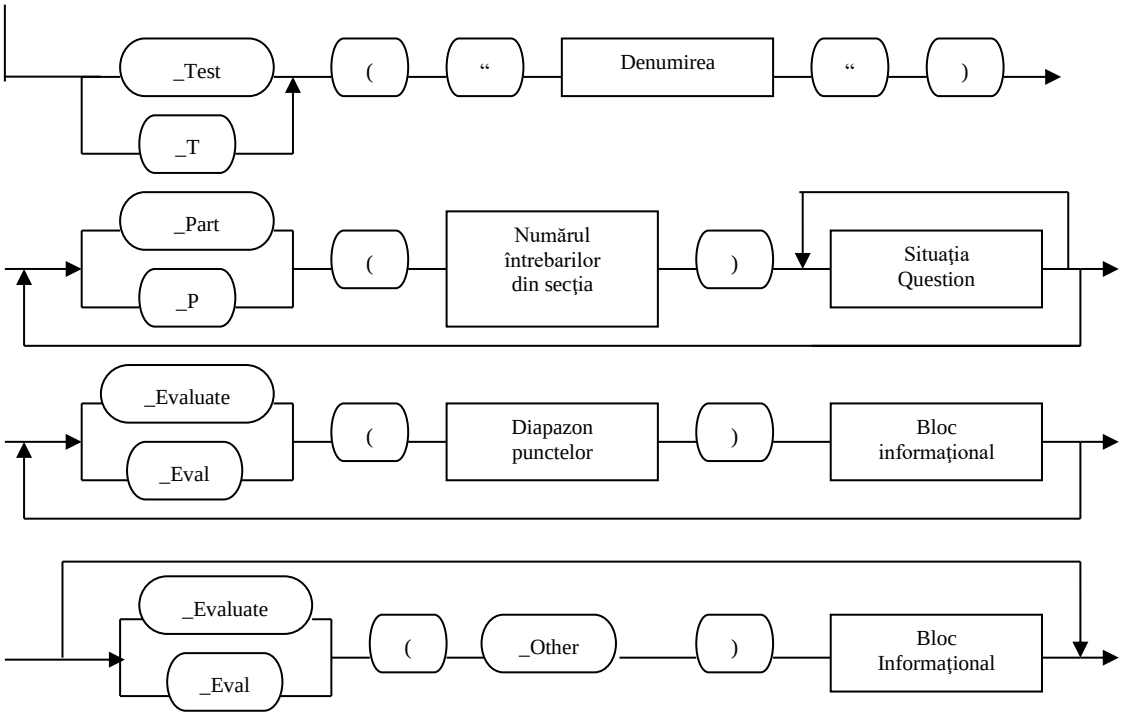
Situaţia Information



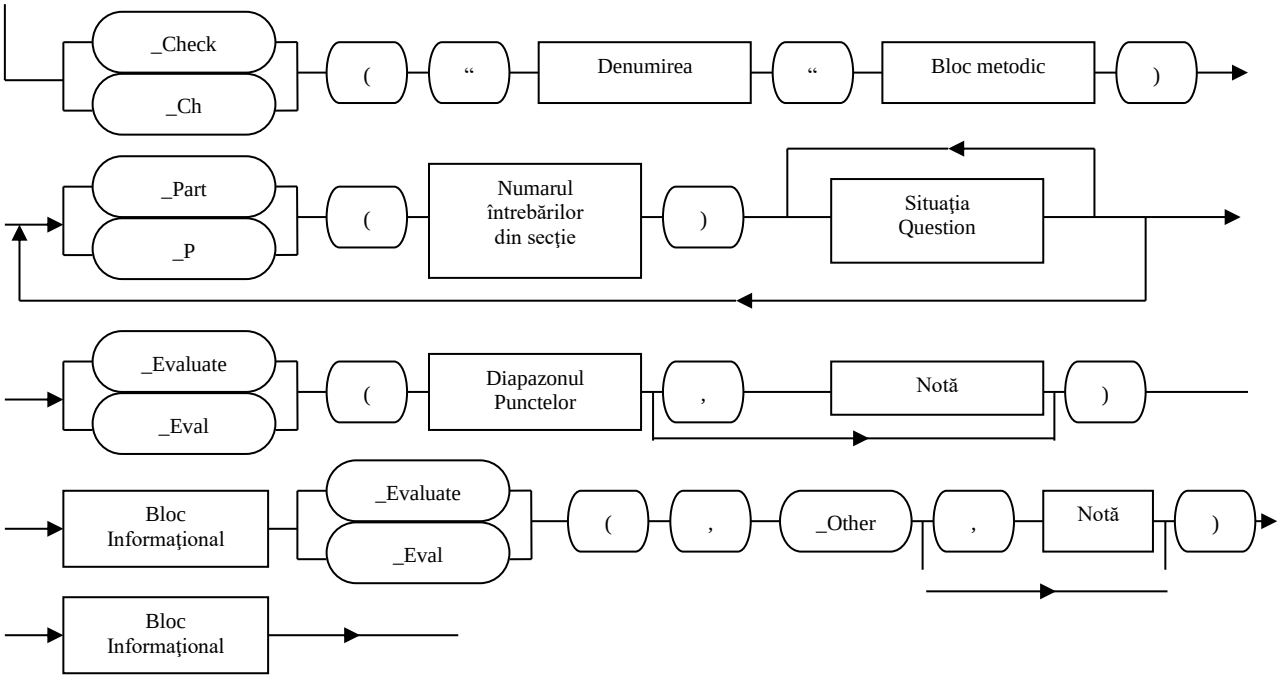
Situaţia Question



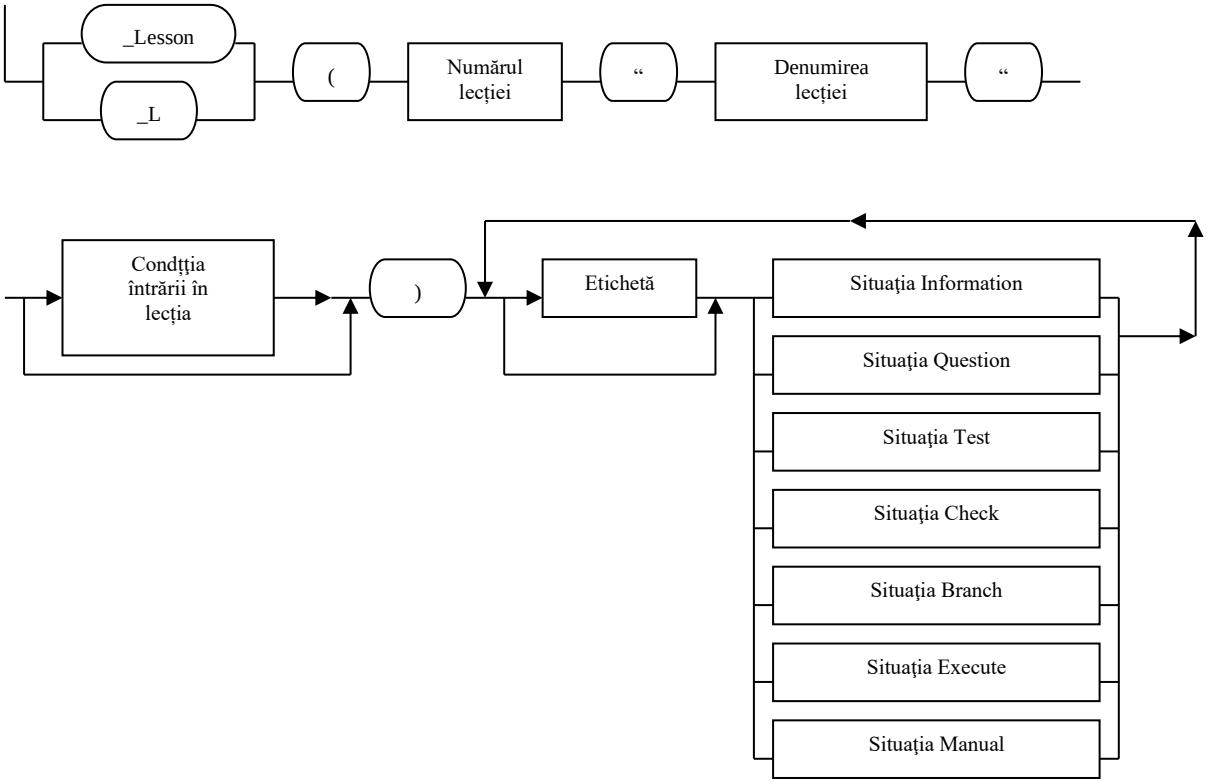
Situaţia Test



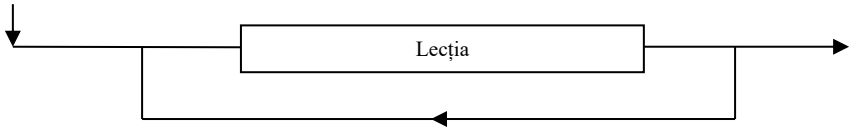
Situaţia Check



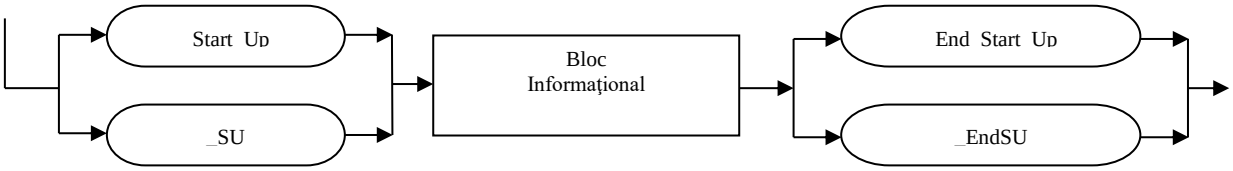
Lecţia



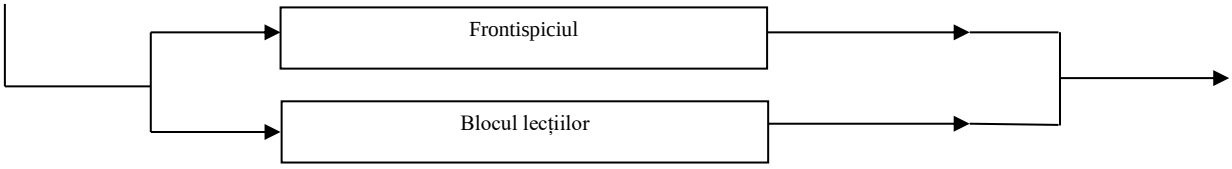
Blocul lecţiilor



Frontispiciul



Fişierul cursului



Anexa 3. Cuvintele-cheie a limbajului de descriere a cursurilor

Este expusă lista integrală a cuvintelor-cheie a limbajului de descriere a cursurilor asistate de calculator MoCo.

|*	*_Diagnostics_Wrong*	*_Undefined*
*|	*_Edit*	*_Video*
(	*_End_of_Lesson*	*_White*
)	*_End_Start_Up*	*_Wrong*
[	*_Evaluate*	*_Yellow*
]	*_eXceed*	*Active_Time*
..	*_Execute*	*Bold*
#	*_Font*	*CM*
.	*_Font_Color*	*Ctime*
,	*_Free*	*Current_N_of_Questions*
$	*_Gray*	*Cyan*
:	*_Green*	*Date*
+	*_Help*	*DG*
-	*_Image*	*False*
/	*_Information*	*Fdate*
*	*_Less*	*Fname*
=	*_Lesson*	*Ftime*
>	*_Magenta*	*Italic*
<	*_Manual_Disable*	*Ldate*
>=	*_Manual_Enable*	*LT*
<=	*_More*	*Ltime*
!=	*_Multiple*	*N_Help*
“	*_None*	*N_of_Questions*
_All	*_Not*	*Name*
_And	*_Or*	*R_Answer*
_Audio	*_Other*	*R_Answer_total*
_Black	*_Part*	*SC*
_Branch	*_Question*	*SP*
_Brown	*_Reaction*	*True*
_Check	*_Red*	*Underlined*
_Color	*_Single*	*W_ Answer_total*
_Correct	*_Start_Up*	*W_Answer*
_Diagnostics_Correct	*_Test*	

Anexa 4. Gramatica G'

Gramatica G' = (N', ∑', P', S'), unde

N' – simbolurile neterminale gramaticii G', în tabel sunt reprezentaţi ca identificatori ce încep cu literă mare (exemplu L_4, X_{41}, KursFile).

∑' – simbolurile terminale gramaticii G', în tabel sunt reprezentaţi ca simbolurile sau şiruri de simboluri încadrate între apostrofe. Simbolurile terminale sunt de asemenea Number şi Word care corespund constantelor şi textelor.

S' – axioma gramaticii este simbol neterminal KursFile.

P' – mulţimea producţiilor gramaticii G'

Tabelul 1. Gramatica G'

0. KursFile → L_2 L_1	1. KursFile → S_1
2. I_1 → T_1 X_4	3. I_1 → I_2 X_5
4. I_1 → ε	5. X_4 →I_2 X_5
6. X_4 → ε	7. X_5 → T_1 X_4
8. X_5 → ε	9. I_2 → '|*' I_3 '*|'
10. I_3 → D_1 I_3	11. I_3 → O_1 I_3
12. I_3 → ε	13. X_3 → Word
14. X_3 → Number	15. X_3 → C_2
16. T_1 → X_3 X_2	17. X_2 →X_3 X_2
18. X_2 → ε	19. O_1 → Audio = N_1
20. O_1 → Video = N_1	21. O_1 → Image = N_1
22. O_1 → '_Font' = C_3	23. O_1 → '_Color' = C_4
24. Audio → '_Audio'	25. Audio → '_Au'
26. Video → '_Video'	27. Video → '_Vi'
28. Image → '_Image'	29. Image → '_Im'
30. L_2 →M_1 Lesson I_4 L_4 EndL	31. Lesson → '_Lesson'
32. Lesson → '_L'	33. EndL → '_End_of_Lesson'
34. EndL → '_EndL'	35. I_4 → '(' N_2 '" ' T_1 '" ' V_2 ')'
36. N_2 → Number X_7	37. X_7 →'.'
38. X_7 → $\boldsymbol{\varepsilon}$	39. L_4 → M_1 O_2 L_4
40. L_4 → $\boldsymbol{\varepsilon}$	41. O_2 → I_5
42. O_2 → Q_1	43. O_2 → T_2

44. O_2 → C_6	45. O_2 → V_1
46. O_2 → F_1	47. M_1 → '#' Word ':'
48. M_1 → ε	49. I_5 → Information I_1
50. Information→ '_Information'	51. Information → '_Inf'
52. F_1 → Execute	53. Execute → '_Execute'
54. Execute → '_Exec'	55. V_1 → Branch
56. Branch → '_Branch'	57. Branch → '_Br'
58. V_3 → V_2 ',' Word X_8	59. X_8 → '/' V_3
60. X_8 → ε	61. Q_1 → Question '(' Q_2
62. Question → '_Question'	63. Question → '_Qu'
64. Q_2 → Q_3	65. Q_2 → Q_4
66. Q_2 → Q_5	67. Q_3 → Q_6 I_1 B_3
68. Q_6 → Free	69. Q_6 → '('
70. X_9 → '('	71. X_9 → ',' Number ')'
72. Free → '_Free'	73. Free → '_F'
74. B_3 → '$' '(' O_3 X_{10}	75. X_{10} → B_3
76. X_{10} → ε	77. H_1 → Help ')' I_1
78. Help → '_Help'	79. Help → '_H'
80. U_2 → Undefined ')' I_1	81. Undefined → '_Undefined'
82. Undefined → '_U'	83. R_1 → Reaction ')'
84. Reaction → '_Reaction'	85. Reaction → '_R'
86. E_2 → Exceed ')'	87. Exceed → '_Exceed'
88. Exceed → '_X'	89. E_3 → Edit ')' I_6
90. Edit → '_Edit'	91. Edit → '_E'
92. I_6 → I_7 X_{11}	93. X_{11} → ',' I_6
94. X_{11} → ε	95. I_7 → 'SP'
96. I_7 → 'CM'	97. I_7 → 'DG'
98. I_7 → 'LT'	99. I_7 → 'SC'
100. I_7 → C_2	101. A_1 → A_2 ',' Number B_2 ')'
102. A_2 → Correct	103. A_2 → Wrong
104. Correct → '_Correct'	105. Correct → '_C'
106. Wrong → '_Wrong'	107. Wrong → '_W'

108. $O_3 \rightarrow H_1$	109. $O_3 \rightarrow U_2$
110. $O_3 \rightarrow R_1$	111. $O_3 \rightarrow E_2$
112. $O_3 \rightarrow E_3$	113. $O_3 \rightarrow A_1$
114. $B_2 \rightarrow$ ',' C_8	115. $B_2 \rightarrow \varepsilon$
116. $C_4 \rightarrow$ '_Red'	117. $C_4 \rightarrow$ '_Green'
118. $C_4 \rightarrow$ '_Blue'	119. $C_4 \rightarrow$ Number
120. $C_3 \rightarrow$ 'B'	121. $C_3 \rightarrow$ 'I'
122. $C_3 \rightarrow$ 'U'	123. $C_3 \rightarrow \varepsilon$
124. $N_1 \rightarrow$ Word	125. $N_1 \rightarrow$ Number
126. $X_{13} \rightarrow$ '.'	127. $X_{13} \rightarrow \varepsilon$
128. $Q_4 \rightarrow Q_7\ I_1\ B_4$	129. $Q_7 \rightarrow$ Single X_9
130. Single → '_Single'	131. Single → '_S'
132. $B_4 \rightarrow$ '$' '(' $O_4\ X_{14}$	133. $X_{14} \rightarrow B_4$
134. $X_{14} \rightarrow \varepsilon$	135. $O_4 \rightarrow H_1$
136. $O_4 \rightarrow R_1$	137. $O_4 \rightarrow E_2$
138. $O_4 \rightarrow A_3$	139. $A_3 \rightarrow A_2\ B_2$ ')' I_1
140. $Q_5 \rightarrow Q_8\ I_1\ B_5$	141. $Q_8 \rightarrow$ Multiple X_9
142. Multiple → '_Multiple'	143. Multiple → '_M'
144. $B_5 \rightarrow$ '$' '(' $O_5\ X_{15}$	145. $X_{15} \rightarrow B_5$
146. $X_{15} \rightarrow \varepsilon$	147. $O_5 \rightarrow H_1$
148. $O_5 \rightarrow R_1$	149. $O_5 \rightarrow E_2$
150. $O_5 \rightarrow A_4$	151. $O_5 \rightarrow D_2$
152. $A_4 \rightarrow A_2$ ')' I_1	153. $D_2 \rightarrow D_3$ ',' $T_3\ B_2$ ')'
154. $D_3 \rightarrow$ Diagnostics A_2	155. Diagnostics → '_Diagnostics'
156. Diagnostics → '_D	157. $T_3 \rightarrow C_7\ A_2\ X_{16}$
158. $X_{16} \rightarrow$ ':'	159. $X_{16} \rightarrow \varepsilon$
160. $C_7 \rightarrow$ Number	161. $C_7 \rightarrow$ All
162. $C_7 \rightarrow$ None	163. $C_7 \rightarrow$ Less Number
164. $C_7 \rightarrow$ More Number	165. All → '_All'
166. All → '_A'	167. None → '_None'
168. None → '_N'	169. Less → '_Less'
170. Less → '_Ls'	171. More → '_More'

172. More → '_Mr'	173. C_8 → '+' Number
174. C_8 → '-' Number	175. C_8 → Number
176. T_2 → Test I_8 I_1 P_1 E_5	177. Test → '_Test'
178. Test → '_T'	179. P_1 → Part '(' C_9 ')' Q_9 X_{17}
180. X_{17} → P_1	181. X_{17} → ε
182. Part → '_Part'	183. Part → '_P'
184. C_9 → Number	185. C_9 → All
186. Q_9 → Q_1 X_{18}	187. X_{18} → Q_9
188. X_{18} → ε	189. E_5 → Evaluate '(' B_6
190. Evaluate → '_Evaluate'	191. Evaluate → '_Eval'
192. B_6 → V_2 → ')' I_1 X_{30}	193. B_6 → '_Other' ')' I_1
194. X_{30} → E_5	195. X_{30} → ε
196. I_8 → '(' I_9 ')'	197. I_9 → ' " ' T_1 ' " '
198. C_6 → Check	199. Check → '_Check'
200. Check → '_Ch'	201. I_{10} → '(' I_9 Number M_2 ')' I_1
202. M_2 → ε	203. M_2 → ',' Number
204. E_6 → Evaluate '('	205. B_8 → V_2 O_6 ')' I_1 X_{32}
206. B_8 → '_Other' O_6 ')' I_1	207. X_{32} → E_6
208. X_{32} → ε	209. O_2 → M_3
210. O_6 → ',' X_{19}	211. O_6 → ',' X_{19}
212. X_{19} → Number	213. X_{19} → Word
214. X_{19} → ε	215. V_2 → V_4 X_{20}
216. X_{20} → Z_1 V_4	217. X_{20} → ε
218. V_4 → T_4 V_5	219. Z_1 → '='
220. Z_1 → '<'	221. Z_1 → '>'
222. Z_1 → '<='	223. Z_1 → '>='
224. Z_1 → '!='	225. Z_2 → '+'
226. Z_2 → '-'	227. Z_3 → '*'
228. Z_3 → '/'	229. V_5 → ε
230. V_5 → Z_2 T_4 V_5	231. V_5 → '_Or' T_4 V_5
232. T_4 → F_2 T_5	233. T_5 → ε
234. T_5 → '_And' F_2	235. T_5 → Z_3 F_2

236. $F_2 \rightarrow D_1$	237. $F_2 \rightarrow$ '(' V_2 ')'
238. $F_2 \rightarrow$ '_Not' F_2	239. $F_2 \rightarrow C_{10}$
240. $F_2 \rightarrow$ '[' D_4 ']'	241. $C_{10} \rightarrow C_8 X_{23}$
242. $X_{23} \rightarrow$ '.' Number	243. $X_{23} \rightarrow \varepsilon$
244. $D_4 \rightarrow D_5 X_{21}$	245. $X_{21} \rightarrow \varepsilon$
246. $X_{21} \rightarrow$ ',' D_4	247. $D_5 \rightarrow N_2 X_{22}$
248. $X_{22} \rightarrow$ '..' N_2	249. $X_{22} \rightarrow \varepsilon$
250. $D_1 \rightarrow$ 'Time'	251. $D_1 \rightarrow$ 'Fname'
252. $D_1 \rightarrow$ 'Lname'	253. $C_2 \rightarrow S_2$
254. $C_2 \rightarrow S_3$	255. $S_2 \rightarrow$ '/'
256. $S_2 \rightarrow$ '.'	257. $S_2 \rightarrow$ ','
258. $S_2 \rightarrow$ '['	259. $S_2 \rightarrow$ ']'
260. $S_2 \rightarrow$ ':'	261. $S_2 \rightarrow$ '+'
262. $S_2 \rightarrow$ '-'	263. $S_2 \rightarrow$ '*'
264. $S_2 \rightarrow$ '>'	265. $S_2 \rightarrow$ '<'
266. $S_2 \rightarrow$ '='	267. $S_2 \rightarrow$ '>='
268. $S_2 \rightarrow$ '<='	269. $S_2 \rightarrow$ '!='
270. $S_2 \rightarrow$ '..'	271. $S_3 \rightarrow$ '('
272. $S_3 \rightarrow$ ')'	273. $I_{11} \rightarrow T_{11} X_{44}$
274. $I_{11} \rightarrow I_2 X_{45}$	275. $I_{11} \rightarrow \varepsilon$
276. $X_{41} \rightarrow$ Word	277. $X_{41} \rightarrow$ Number
278. $X_{41} \rightarrow S_2$	279. $T_{11} \rightarrow X_{41} X_{42}$
280. $X_{42} \rightarrow X_{41} X_{42}$	281. $X_{42} \rightarrow \varepsilon$
282. $X_{44} \rightarrow I_2 ZX_{45}$	283. $X_{44} \rightarrow \varepsilon$
284. $X_{45} \rightarrow T_{11} X_{44}$	285. $X_{45} \rightarrow \varepsilon$
286. $L_1 \rightarrow L_2 L_1$	287. $L_1 \rightarrow \varepsilon$
288. $S_1 \rightarrow$ StartUp I_1 EndSU	289. StartUp → '_StartUp'
290. StartUp → '_SU'	291. EndSU → '_End_StartUp'
292. EndSU → '_EndSU'	293. $M_3 \rightarrow$ ManualEnable
294. $M_3 \rightarrow$ ManualDisable	295. ManualEnable → '_Manual_Enable'
296. ManualEnable → '_ME'	297. ManualDisable →'_Manual_Disable'
298. ManualDisable → '_MD'	

Anexa 5. Mulțimea σ de simboluri directoare ale gramaticii G'

Pentru fiecare producţie gramaticii este dată mulţimea simbolurilor directoare. În baza datelor din tabel se demonstrează faptul că gramatica G' aparţine clasei gramaticilor *LL(1)*. Pe lângă acest fapt, tabelul dat a fost folosit la construirea tabelului de dirijare MTP.

Tabelul 2. Gramatica Mulțimea σ de simboluri directoare ale gramaticii G'

0.	KursFile	'#', '_Lesson', '_L'
1.	KursFile	'_Start_Up', '_SU'
2.	I_1	Word, Number, '/', '.', ',', '(' , ')', '[', ']', ':', '+', '-', '*', '>', '<', '=', '>=', '<=', '!='
3.	I_1	'|*'
4.	I_1	'$', '#', '_Information', '_Inf', '_Check', '_Ch', '_Branch', '_Br', '_Question', '_Qu', '_Test', '_T', '_Execute', '_Exec', '_Manual_Enable', '_ME', '_Manual_Disable', '_MD', '_End_of_Lesson', '_EndL', '_Evalute', '_Eval', '_Part', '_P', '_Lesson', '_L'
5.	X_4	'|*'
6.	X_4	'$', '#', '_Information', '_Inf', '_Check', '_Ch', '_Branch', '_Br', '_Question', '_Qu', '_Test', '_T', '_Execute', '_Exec', '_Manual_Enable', '_ME', '_Manual_Disable', '_MD', '_End_of_Lesson', '_EndL', '_Evalute', '_Eval', '_Part', '_P', '_Lesson', '_L'
7.	X_5	Word, Number, '/', '.', ',', '(' , ')', '[', ']', ':', '+', '-', '*', '>', '<', '=', '>=', '<=', '!='
8.	X_5	'$', '#', '_Information', '_Inf', '_Check', '_Ch', '_Branch', '_Br', '_Question', '_Qu', '_Test', '_T', '_Execute', '_Exec', '_Manual_Enable', '_ME', '_Manual_Disable', '_MD', '_End_of_Lesson', '_EndL', '_Evalute', '_Eval', '_Part', '_P', '_Lesson', '_L'
9.	I_2	'|*'
10.	I_3	'Time', 'Fname', 'Lname'
11.	I_3	'_Audio', '_Au', '_Video', '_Vi', '_Image', '_Im', '_Font', '_Color'
12.	I_3	'*|'
13.	X_3	Word
14.	X_3	Number
15.	X_3	'/', '.', ',', '(' , ')', '[', ']', ':', '+', '-', '*', '>', '<', '=', '>=', '<=', '!='
16.	T_1	Word, Number, '/', '.', ',', '(' , ')', '[', ']', ':', '+', '-', '*', '>', '<', '=', '>=', '<=', '!='
17.	X_2	Word, Number, '/', '.', ',', '(' , ')', '[', ']', ':', '+', '-', '*', '>', '<', '=', '>=', '<=', '!='
18.	X_2	'|*', ' ' , '$', '#', '_Information', '_Inf', '_Check', '_Ch', '_Branch', '_Br', '_Question', '_Qu', '_Test', '_T', '_Execute', '_Exec', '_Manual_Enable',

		‘_ME’, ‘_Manual_Disable’, ‘_MD’, ‘_End_of_Lesson’, ‘_EndL’, ‘_Evalute’, ‘_Eval’, ‘_Part’, ‘_P’, ‘_Lesson’, ‘_L’
19.	O_1	‘_Audio’, ‘_Au’
20.	O_1	‘_Video’, ‘_Vi’
21.	O_1	‘_Image’, ‘_Im’
22.	O_1	‘_Font’
23.	O_1	‘_Color’
24.	Audio	‘_Audio’
25.	Audio	‘_Au’
26.	Video	‘_Video’
27.	Video	‘_Vi’
28.	Image	‘_Image’
29.	Image	‘_Im’
30.	L_2	‘#’, ‘_Lesson’, ‘_L’
31.	Lesson	‘_Lesson’
32.	Lesson	‘_L’
33.	EndL	‘_End_of_Lesson’
34.	EndL	‘_EndL’
35.	I_4	‘(’
36.	N_2	Number
37.	X_7	‘.’
38.	X_7	‘ ’ ’, ‘..’, ‘,’, ‘]’
39.	L_4	‘#’, ‘_Information’, ‘_Inf’, ‘_Check’, ‘_Ch’, ‘_Branch’, ‘_Br’, ‘_Question’, ‘_Qu’, ‘_Test’, ‘_T’, ‘_Execute’, ‘_Exec’, ‘_Manual_Enable’, ‘_ME’, ‘_Manual_Disable’, ‘_MD’
40.	L_4	‘_End_of_Lesson’, ‘_EndL’
41.	O_2	‘_Information’, ‘_Inf’
42.	O_2	‘_Question’, ‘_Qu’
43.	O_2	‘_Test’, ‘_T’
44.	O_2	‘_Check’, ‘_Ch’
45.	O_2	‘_Branch’, ‘_Br’
46.	O_2	‘_Execute’, ‘_Exec’
47.	M_1	‘#’

48.	M_1	'_Lesson', '_L', '_Information', '_Inf', '_Check', '_Ch', '_Branch', '_Br', '_Question', '_Qu', '_Test', '_T', '_Execute', '_Exec', '_Manual_Enable', '_ME', '_Manual_Disable', '_MD'
49.	I_5	'_Information', '_Inf'
50.	Information	'_Information'
51.	Information	'_Inf'
52.	F_1	'_Execute', '_Exec'
53.	Execute	'_Execute'
54.	Execute	'_Exec'
55.	V_1	'_Branch', '_Br'
56.	Branch	'_Branch'
57.	Branch	'_Br'
58.	V_3	'(', '[', '_Not', 'Time', 'Fname', 'Lname', '+', '-', Number
59.	X_8	'/'
60.	X_8	'#', '_Information', '_Inf', '_Check', '_Ch', '_Branch', '_Br', '_Question', '_Qu', '_Test', '_T', '_Execute', '_Exec', '_Manual_Enable', '_ME', '_Manual_Disable', '_MD', '_End_of_Lesson', '_EndL'
61.	Q_1	'_Question', '_Qu'
62.	Question	'_Question'
63.	Question	'_Qu'
64.	Q_2	'_Free', '_F', ')'
65.	Q_2	'_Single', '_S'
66.	Q_2	'_Multiple', '_M'
67.	Q_3	'_Free', '_F', ')'
68.	Q_6	'_Free', '_F'
69.	Q_6	')'
70.	X_9	')'
71.	X_9	','
72.	Free	'_Free'
73.	Free	'_F'
74.	B_3	'$'
75.	X_{10}	'$'
76.	X_{10}	'#', '_Information', '_Inf', '_Check', '_Ch', '_Branch', '_Br', '_Question', '_Qu', '_Test', '_T', '_Execute', '_Exec', '_Manual_Enable', '_ME',

		'_Manual_Disable', '_MD', '_End_of_Lesson', '_EndL', '_Evaluate', '_Eval', '_Part', '_P'
77.	H_1	'_Help', '_H'
78.	Help	'_Help'
79.	Help	'_H'
80.	U_2	'_Undefined', '_U'
81.	Undefined	'_Undefined'
82.	Undefined	'_U'
83.	R_1	'_Reaction', '_R'
84.	Reaction	'_Reaction'
85.	Reaction	'_R'
86.	E_2	'_Exceed', '_X'
87.	Exceed	'_Exceed'
88.	Exceed	'_X'
89.	E_3	'_Edit', '_E'
90.	Edit	'_Edit'
91.	Edit	'_E'
92.	I_6	'SP', 'CM', 'DG', 'LT', 'SC', '/', '.', ',', '(', ')', '[', ']', ':', '+', '-', '*', '>', '<', '=', '>=', '<=', '!='
93.	X_{11}	','
94.	X_{11}	'$', '#', '_Information', '_Inf', '_Check', '_Ch', '_Branch', '_Br', '_Question', '_Qu', '_Test', '_T', '_Execute', '_Exec', '_Manual_Enable', '_ME', '_Manual_Disable', '_MD', '_End_of_Lesson', '_EndL', '_Evaluate', '_Eval', '_Part', '_P'
95.	I_7	'SP'
96.	I_7	'CM'
97.	I_7	'DG'
98.	I_7	'LT'
99.	I_7	'SC'
100.	I_7	'/', '.', ',', '(', ')', '[', ']', ':', '+', '-', '*', '>', '<', '=', '>=', '<=', '!='
101.	A_1	'_Correct', '_C', '_Wrong', '_W'
102.	A_2	'_Correct', '_C'
103.	A_2	'_Wrong', '_W'
104.	Correct	'_Correct'
105.	Correct	'_C'

106.	Wrong	'_Wrong'
107.	Wrong	'_W'
108.	O_3	'_Help', '_H'
109.	O_3	'_Undefined', '_U'
110.	O_3	'_Reaction', '_R'
111.	O_3	'_Exceed', '_X'
112.	O_3	'_Edit', '_E'
113.	O_3	'_Correct', '_C', '_Wrong', '_W'
114.	B_2	','
115.	B_2	')'
116.	C_4	'_Red'
117.	C_4	'_Green'
118.	C_4	'_Blue'
119.	C_4	Number
120.	C_3	'B'
121.	C_3	'I'
122.	C_3	'U'
123.	C_3	'*|'
124.	N_1	Word
125.	N_1	Number
126.	X_{13}	'.'
127.	X_{13}	'*|' '$', '#', '_Information', '_Inf', '_Check', '_Ch', '_Branch', '_Br', '_Question', '_Qu', '_Test', '_T', '_Execute', '_Exec', '_Manual_Enable', '_ME', '_Manual_Disable', '_MD', '_End_of_Lesson', '_EndL', '_Evaluate', '_Eval', '_Part', '_P', 'Time', 'Fname', 'Lname', '_Audio', '_Au', '_Video', '_Vi', '_Image', '_Im', '_Font', '_Color'
128.	Q_4	'_Single', '_S'
129.	Q_7	'_Single', '_S'
130.	Single	'_Single'
131.	Single	'_S'
132.	B_4	'$'
133.	X_{14}	'$'
134.	X_{14}	'#', '_Information', '_Inf', '_Check', '_Ch', '_Branch', '_Br', '_Question', '_Qu', '_Test', '_T', '_Execute', '_Exec', '_Manual_Enable', '_ME',

		'_Manual_Disable', '_MD', '_End_of_Lesson', '_EndL', '_Evaluate', '_Eval', '_Part', '_P'
135.	O_4	'_Help', '_H'
136.	O_4	'_Reaction', '_R'
137.	O_4	'_Exceed', '_X'
138.	O_4	'_Correct', '_C', '_Wrong', '_W'
139.	A_3	'_Correct', '_C', '_Wrong', '_W'
140.	Q_5	'_Multiple', '_M'
141.	Q_8	'_Multiple', '_M'
142.	Multiple	'_Multiple'
143.	Multiple	'_M'
144.	B_5	'$'
145.	X_{15}	'$'
146.	X_{15}	'#', '_Information', '_Inf', '_Check', '_Ch', '_Branch', '_Br', '_Question', '_Qu', '_Test', '_T', '_Execute', '_Exec', '_Manual_Enable', '_ME', '_Manual_Disable', '_MD', '_End_of_Lesson', '_EndL', '_Evaluate', '_Eval', '_Part', '_P'
147.	O_5	'_Help', '_H'
148.	O_5	'_Reaction', '_R'
149.	O_5	'_Exceed', '_X'
150.	O_5	'_Correct', '_C', '_Wrong', '_W'
151.	O_5	'_Diagnostics', '_D'
152.	A_4	'_Correct', '_C', '_Wrong', '_W'
153.	D_2	'_Diagnostics', '_D'
154.	D_3	'_Diagnostics', '_D'
155.	Diagnostics	'_Diagnostics'
156.	Diagnostics	'_D'
157.	T_3	Number, '_All', '_A', '_None', '_N', '_Less', '_Ls', '_More', '_Mr'
158.	X_{16}	':'
159.	X_{16}	',', ')'
160.	C_7	Number
161.	C_7	'_All', '_A'
162.	C_7	'_None', '_N'
163.	C_7	'_Less', '_Ls'

164.	C_7	'_More', '_Mr'
165.	All	'_All'
166.	All	'_A'
167.	None	'_None'
168.	None	'_N'
169.	Less	'_Less'
170.	Less	'_Ls'
171.	More	'_More'
172.	More	'_Mr'
173.	C_8	'+'
174.	C_8	'-'
175.	C_8	Number
176.	T_2	'_Test', '_T'
177.	Test	'_Test'
178.	Test	'_T'
179.	P_1	'_Part', '_P'
180.	X_{17}	'_Part', '_P'
181.	X_{17}	'_Evaluate', '_Eval'
182.	Part	'_Part'
183.	Part	'_P'
184.	C_9	Number
185.	C_9	'_All', '_A'
186.	Q_9	'_Question', '_Qu'
187.	X_{18}	'_Question', '_Qu'
188.	X_{18}	'_Part', '_P', '_Evaluate', '_Eval'
189.	E_5	'_Evaluate', '_Eval'
190.	Evaluate	'_Evaluate'
191.	Evaluate	'_Eval'
192.	B_6	'(', '[', 'Not', 'Time', 'Fname', 'Lname', '+', '-', Number
193.	B_6	'_Other'
194.	X_{30}	'_Evaluate', '_Eval'

195.	X_{30}	'#', '_Information', '_Inf', '_Check', '_Ch', '_Branch', '_Br', '_Question', '_Qu', '_Test', '_T', '_Execute', '_Exec', '_Manual_Enable', '_ME', '_Manual_Disable', '_MD', '_End_of_Lesson', '_EndL'
196.	I_8	'('
197.	I_9	' " '
198.	C_6	'_Check', '_Ch'
199.	Check	'_Check'
200.	Check	'_Ch'
201.	I_{10}	'('
202.	M_2	')'
203.	M_2	','
204.	E_6	'_Evaluate', '_Eval'
205.	B_8	'(', '[', 'Not', 'Time', 'Fname', 'Lname', '+', '-', Number
206.	B_8	'_Other'
207.	X_{32}	'_Evaluate', '_Eval'
208.	X_{32}	'#', '_Information', '_Inf', '_Check', '_Ch', '_Branch', '_Br', '_Question', '_Qu', '_Test', '_T', '_Execute', '_Exec', '_Manual_Enable', '_ME', '_Manual_Disable', '_MD', '_End_of_Lesson', '_EndL'
209.	O_2	'_Manual_Enable', '_ME', '_Manual_Disable', '_MD'
210.	O_6	')'
211.	O_6	','
212.	X_{19}	Number
213.	X_{19}	Word
214.	X_{19}	')'
215.	V_2	'(', '[', 'Not', 'Time', 'Fname', 'Lname', '+', '-', Number
216.	X_{20}	'<', '>', '=', '<=', '>=', '!='
217.	X_{20}	')', ','
218.	V_4	'(', '[', 'Not', 'Time', 'Fname', 'Lname', '+', '-', Number
219.	Z_1	'='
220.	Z_1	'<'
221.	Z_1	'>'
222.	Z_1	'<='
223.	Z_1	'>='
224.	Z_1	'!='

225.	Z_2	'+'
226.	Z_2	'-'
227.	Z_3	'*'
228.	Z_3	'/'
229.	V_5	'<', '>', '=', '<=', '>=', '!=', ')', ','
230.	V_5	'+', '-'
231.	V_5	'_Or'
232.	T_4	'(', '[', 'Not', 'Time', 'Fname', 'Lname', '+', '-', Number
233.	T_5	'_Or', '+', '-', '<', '>', '=', '<=', '>=', '!=', ')', ','
234.	T_5	'_And'
235.	T_5	'*', '/'
236.	F_2	'Time', 'Fname', 'Lname'
237.	F_2	'('
238.	F_2	'_Not'
239.	F_2	'+', '-', Number
240.	F_2	'['
241.	C_{10}	'+', '-', Number
242.	X_{23}	'.'
243.	X_{23}	'_And', '_Or', '+', '-', '<', '>', '=', '<=', '>=', '!=', ')', ','
244.	D_4	Number
245.	X_{21}	']'
246.	X_{21}	','
247.	D_5	Number
248.	X_{22}	'..'
249.	X_{22}	',', ']'
250.	D_1	'Time'
251.	D_1	'Fname'
252.	D_1	'Lname'
253.	C_2	'/', '.', ',', '[', ']', ':', '+', '-', '*', '>', '<', '=', '>=', '<=', '!=', ' '
254.	C_2	'(', ')'
255.	S_2	'/'
256.	S_2	'.'

257.	S_2	‘,’
258.	S_2	‘[’
259.	S_2	‘]’
260.	S_2	‘:’
261.	S_2	‘+’
262.	S_2	‘-’
263.	S_2	‘*’
264.	S_2	‘>’
265.	S_2	‘<’
266.	S_2	‘=’
267.	S_2	‘>=’
268.	S_2	‘<=’
269.	S_2	‘!=’
270.	S_2	‘..’
271.	S_3	‘(’
272.	S_3	‘)’
273.	I_{11}	Word, Number, ‘/’, ‘.’, ‘,’, ‘[‘, ‘]’, ‘:’, ‘+’, ‘-’, ‘*’, ‘>’, ‘<’, ‘=’, ‘>=’, ‘<=’, ‘!=’
274.	I_{11}	‘|*’
275.	I_{11}	‘#’, ‘_Information’, ‘_Inf’, ‘_Check’, ‘_Ch’, ‘_Branch’, ‘_Br’, ‘_Question’, ‘_Qu’, ‘_Test’, ‘_T’, ‘_Execute’, ‘_Exec’, , ‘_Manual_Enable’, ‘_ME’, ‘_Manual_Disable’, ‘_MD’ ‘_End_of_Lesson’, ‘_EndL’
276.	X_{41}	Word
277.	X_{41}	Number
278.	X_{41}	‘/’, ‘.’, ‘,’, ‘[‘, ‘]’, ‘:’, ‘+’, ‘-’, ‘*’, ‘>’, ‘<’, ‘=’, ‘>=’, ‘<=’, ‘!=’
279.	T_{11}	Word, Number, ‘/’, ‘.’, ‘,’, ‘[‘, ‘]’, ‘:’, ‘+’, ‘-’, ‘*’, ‘>’, ‘<’, ‘=’, ‘>=’, ‘<=’, ‘!=’
280.	X_{42}	Word, Number, ‘/’, ‘.’, ‘,’, ‘(’, ‘)’, ‘[‘, ‘]’, ‘:’, ‘+’, ‘-’, ‘*’, ‘>’, ‘<’, ‘=’, ‘>=’, ‘<=’, ‘!=’
281.	X_{42}	‘|*’,‘#’, ‘_Information’, ‘_Inf’, ‘_Check’, ‘_Ch’, ‘_Branch’, ‘_Br’, ‘_Question’, ‘_Qu’, ‘_Test’, ‘_T’, ‘_Execute’, ‘_Exec’, ‘_Manual_Enable’, ‘_ME’, ‘_Manual_Disable’, ‘_MD’, ‘_End_of_Lesson’, ‘_EndL’
282.	X_{44}	‘|*’
283.	X_{44}	‘#’, ‘_Information’, ‘_Inf’, ‘_Check’, ‘_Ch’, ‘_Branch’, ‘_Br’, ‘_Question’, ‘_Qu’, ‘_Test’, ‘_T’, ‘_Execute’, ‘_Exec’, ‘_Manual_Enable’, ‘_ME’, ‘_Manual_Disable’, ‘_MD’, ‘_End_of_Lesson’, ‘_EndL’
284.	X_{45}	Word, Number, ‘/’, ‘.’, ‘,’, ‘[‘, ‘]’, ‘:’, ‘+’, ‘-’, ‘*’, ‘>’, ‘<’, ‘=’, ‘>=’, ‘<=’, ‘!=’

285.	X_{45}	‘(’,‘#’, ‘_Information’, ‘_Inf’, ‘_Check’, ‘_Ch’, ‘_Branch’, ‘_Br’, ‘_Question’, ‘_Qu’, ‘_Test’, ‘_T’, ‘_Execute’, ‘_Exec’, ‘_Manual_Enable’, ‘_ME’, ‘_Manual_Disable’, ‘_MD’, ‘_End_of_Lesson’, ‘_EndL’
286.	L_1	‘#’, ‘_Lesson’, ‘_L’
287.	L_1	∇
288.	S_1	‘_StartUp’, ‘_SU’
289.	StartUp	‘_StartUp’
290.	StartUp	‘_SU’
291.	EndSU	‘_End_StartUp’
292.	EndSU	‘_EndSU’
293.	M_3	‘_Manual_Enable’, ‘_ME’
294.	M_3	‘_Manual_Disable’, ‘_MD’
295.	ManualEnable	‘_Manual_Enable’
296.	ManualEnable	‘_ME’
297.	ManualDisable	‘_Manual_Disable’
298.	ManualDisable	‘_MD’

Anexa 6. Cuvintele-cheie a limbajului intermediar

Este expusă lista completă a cuvintelor-cheie şi codurilor respective limbajului intermediar, care este limbajul de ieşire a translatorului şi limbajul de intrare a interpretatorului.

Tabelul 3. Cuvintele-cheie a limbajului intermediar

#01	Audio	#29	Other
#02	Branch	#30	Part
#03	Check	#31	Reaction
#04	CM	#32	SC
#05	Color	#33	Single
#06	Condition	#34	SP
#07	Correct	#35	Startup
#08	DB	#36	Symbol
#09	DG	#37	Test
#10	Diagnostics	#38	Text
#11	Diapason	#39	Undefined
#12	Edit	#40	Video
#13	Evaluate	#41	Wrong
#14	Exceed	#42	And
#15	Execute	#43	Not
#16	Font	#44	Or
#17	Font Color	#45	To
#18	Free	#46	&
#19	Go To	#47	+
#20	Help	#48	-
#21	Image	#49	*
#22	Information	#50	/
#23	Label	#51	=
#24	Lesson	#52	>
#25	LT	#53	<
#26	Manual	#54	>=
#27	Multiple	#55	<=
#28	Number	#56	!=

Anexa 7. Codul intermediar. Descrierea structurilor.

- Numere

#00	Cod Number	4 octeţi – valoarea numărului

- Datele din Baza de Date

#00	Cod DB	Consecutivitatea de simboluri (excluzând simbol cu codul 00), care determină denumirea câmpului în baza de date.

- Diapazon

#00	Cod Diapazon	Diapazonul de numere

Diapazonul de numere reprezintă consecutivitatea Numerelor, despărţită de Cod & sau Cod To (aceste coduri se încep cu simbolul #00).

Exemplu de diapazonul numerelor:

Text iniţial: *9, 12..45, 67, 5*

Rezultat: #00 #28 $(9)_4$ #00 #46 #00 #28 $(12)_4$ #00 #45 $(45)_4$ #00 #46 $(67)_4$ #00 #46 $(5)_4$,

Unde indice de jos reprezintă numărul de octeţi, ocupat de număr, #28 este Cod Number, #45 este Cod To, şi #46 este Cod &.

Condiţia reprezintă consecutivitatea de Cod +, Cod –, Cod *, Cod /, Cod And, Cod Or, Cod Not, Cod =, Cod >, Cod <, Cod >=, Cod <=, Cod != şi Factori. Fiecare din aceste codurile expuşi este precedat de simbolul #00.

Factor reprezintă datele din baza de date, numere, diapazoane sau condiţii.

Exemplu de condiţie:

Text iniţial: *-8*Time / 2 != 10*

Rezultat: #00 #48 #00 #28 $(8)_4$ #00 #49 #00 #08 Time #00 #50 #00 #28 $(2)_4$ #00 #56 #00 #28 $(10)_4$

- Text, Audio, Video, Imagini

#00	Cod Text Cod Audio Cod Video Cod Image	Consecutivitatea de simboluri (excluzând simbol cu codul 00). Consecutivitatea de simboluri în câmpul de date a operatorilor Audio, Video şi Image reprezintă denumirea fişierului

- Font

#00	Cod Font	1 octet, conţinutul a 3 biţi inferiori a căruia defineşte stilul fontului. Cel mai inferior bit – stilul Bold, al doilea – Italic şi al treilea – Underline.

- Color, FontColor

#00	Cod Color Cod FontColor	3 octeţi, conţinutul cărora defineşte culoarea după principiu RGB.

Consecutivitatea arbitrară de operatori Text, Audio, Video, Image, Font, Color, FontColor şi DB reprezintă Bloc informaţional.

Exemplu de bloc informaţional:

Text iniţial: *Welcome! /*_Audio=hi.wav*/ Into the world of History! /*_Image = im.bmp*/*
Rezultat: #00 #38 Welcome! #00 #01 hi1.wav #00 #38 Into the world of History! #00 #21 im.bmp

- Eticheta

#00	Cod Label	4 octeţi (număr întreg) ce reprezintă numărul.

- Frontispiciul cursului

#00	Cod StartUp	Bloc informaţional.

- Lecţia

#00	Cod Lesson	Consecutivitatea de simboluri (excluzând simbolul 00) – numărul lecţiei			
#00	Cod Text	Denumirea lecţiei	#00	Cod Condition	Condiţia

Informaţia amplasată după #00 Cod Lesson până la următorul #00 Cod Lesson sau până la sfârşitul fişierului se consideră o lecţie.

Exemplu de lecţie:
Text iniţial: *_Lesson (10 "History" [3,6..9])*
Rezultat: #00 #24 10 #00 #38 History #00 #06 #00 #11 #00 #28 $(3)_4$ #00 #46 #00 #28 $(6)_4$ #00 #45 #00 #28 $(9)_4$

- Information

#00	Cod Information	Bloc informaţional

- Execute

#00	Cod Execute	Consecutivitatea de simboluri (excluzând simbol cu codul 00) care reprezintă denumirea fişierului.

- Manual

#00	Cod Manual	1 octet, conţinutul bitului minor al căruia defineşte starea manualului cursului (1 activ, 0 nu).

- Branch

#00	Cod Branch	Blocul de treceri

- Trecere

#00	Cod Condition	Condiţia	#00	Cod GoTo	Consecutivitatea de simboluri – denumirea etichetei

Consecutivitatea arbitrară de treceri reprezintă Blocul de treceri.

Exemplu de Branch:
Text iniţial: *_Branch Time < 10, Label1 / Time >= 20, Label2*
Rezultat: #00 #02 #00 #06 #00 #08 Time #00 #53 #00 #28 $(10)_4$ #00 #19 Label1 #00 #06 #00 #08 Time #00 #54 #00 #28 $(20)_4$ #00 #19

- Question Free, Question Single sau Question Multiple

#00	Cod Free Cod Single Cod Multiple	1 octet numărul încercărilor de răspuns	Bloc informaţional – formularea întrebării.	Bloc Free, Bloc Single sau Bloc Multiple respectiv

- Help

#00	Cod Help	Bloc informaţional

- Exceed

#00	Cod Exceed	Bloc informaţional

- Reaction

#00	Cod Reaction	Bloc informaţional

- Undefined

#00	Cod Undefined	Bloc informaţional

- Edit

#00	Cod Edit	Blocul simbolurilor de înlăturare

- Simbol de înlăturare

#00	Cod Symbol	1 octet – simbol

- Grup de simboluri de înlăturare

#00	Cod SP, Cod CM, Cod DG, Cod LT sau Cod SC

Blocul simbolurilor de înlăturare reprezintă consecutivitatea de simboluri de înlăturare sau de grupe de simboluri de înlăturare.

Exemplu de Edit:
Text iniţial: *_Edit SP:CMDG.*/*
Rezultat: #00 #12 #00 #34 #00 #36 $(:)_1$ #00 #04 #00 #09 #00 #36 $(.)_1$ #00 #36 $(*)_1$ #00 #36 $(/)_1$

- Correct Free, Wrong Free

#00	Cod Correct Cod Wrong	1 octet – modul de comparare	2 octeţi – puncte pentru răspuns.	Consecutivitatea de simboluri (excluzând simbol cu codul 00) – şablonul răspunsului

Exemplu de Correct Free:
Text iniţial: *$(_Correct, 1, 10) *tion*
Rezultat: #00 #07 $(1)_1$ $(10)_2$ (*tion)

- Correct Single, Wrong Single

#00	Cod Correct Cod Wrong	2 octeți – puncte pentru răspuns.	Bloc informațional – varianta răspunsului.

Exemplu de Wrong Single:
Text inițial: *$(_Wrong, -5) Paris /*_Image = p1.gif */*
Rezultat: #00 #41 $(-5)_2$ #00 #38 Paris #00 #21 p1.gif

- Correct Multiple, Wrong Multiple

#00	Cod Correct Cod Wrong	Bloc informațional – varianta răspunsului

- Diagnostics Correct, Diagnostics Wrong

#00	Cod Diagnostics	Cod Correct Cod Wrong	1 octet – numărul necesar de răspunsuri corecte (-1 reprezintă orice număr)
1 octet – cantitatea necesară de răspunsuri incorecte (-1 reprezintă orice cantitate).			2 octeți – puncte pentru răspuns.

Exemplu de Diagnostics Correct:
Text inițial: *$(_Diagnostics _Correct , 5 _Correct; _All _Wrong, 12)*
Rezultat: #00 #10 #07 $(5)_1$ $(-1)_1$ $(12)_2$

Bloc Free reprezintă consecutivitatea arbitrară de operatori Help, Undefined, Reaction, Exceed, Edit, Correct Free, Wrong Free.

Bloc Single reprezintă consecutivitatea arbitrară de operatori Help, Reaction, Exceed, Correct Single, Wrong Single.

Bloc Multiple reprezintă consecutivitatea arbitrară de operatori Help, Reaction, Exceed, Correct Multiple, Wrong Multiple, Diagnostics Correct, Diagnostics Wrong.

- Test

#00	Cod Test	#00	Cod Text	Consecutivitatea de simboluri (excluzând simbol cu codul 00) – denumirea testului
Blocul secțiilor Part				Blocul secțiilor Evaluate Test

- Check

#00	Check	#00	Cod Text	Consecutivitatea de simboluri (excluzând simbol cu codul 00) – denumirea lucrării de control
Blocul secțiilor Part				Blocul secțiilor Evaluate Check

- Part

#00	Cod Part	1 octet – numărul întrebărilor din secție (-1 reprezintă toate întrebările)	Consecutivitatea întrebărilor

Exemplu de Part:
Text inițial: *_Part(2)*
_Question(_S)
…

_Question(_F)
…
_Question(_M)
…

Rezultat: #00 #30 $(2)_1$ #00 #33 … #00 #18 … #00 #27 …

Consecutivitatea întrebărilor reprezintă combinaţia arbitrară de operatori Question Free, Question Single sau Question Multiple.

Blocul secţiilor Part reprezintă consecutivitatea arbitrară de operatori Part.

- Evaluate Test

#00	Cod Evaluate	Diapazonul punctelor	Bloc informaţional

- Evaluate Test Other

#00	Cod Evaluate	#00	Cod Other	Bloc informaţional

Exemplu de Evaluate Test

Text iniţial: *_Evaluate (30..40) You are good student!*

Rezultat: #00 #13 #00 #11 #00 #28 $(30)_4$ #00 #45 #00 #28 $(40)_4$ #00 #38 You are good student!

- Evaluate Check

#00	Cod Evaluate	Diapazonul punctelor	2 octeţi nota	Bloc informaţional

- Evaluate Check Other

#00	Cod Evaluate	#00	Cod Other	2 octeţi nota	Bloc informaţional

Exemplu de Evaluate Test

Text iniţial: *_Evaluate (_Other, 5) Excelent!*

Rezultat: #00 #13 #00 #29 $(5)_2$ #00 #38 Excelent!

Blocul secţiilor Evaluate Test reprezintă consecutivitatea operatorilor Evaluate Test, care poate fi urmată de operator Evaluate Test Other.

Blocul secţiilor Evaluate Check reprezintă consecutivitatea operatorilor Evaluate Check, care poate fi urmată de operator Evaluate Check Other.

Anexa 8. Partea tabelului de dirijare MTP.

Tabelul 4. Partea tabelului de dirijare MTP

	Lesson	*Endof Lesson*	*Number*	*Word*	*"*	*(*	*)*	*[*	*]*	∇
Lesson	avans									
Endof Lesson		avans								
Number			avans							
Word				avans						
"					avans					
(						avans				
)							avans			
[								avans		
]									avans	
KursFile	0									
D_4			244							
F_2			239			237		240		
I_4						35				
L_1	286									
L_2	30									214
L_4		40								
M_1	48									
N_2				36						
T_1			16	16		16	16	16	16	
T_4				232		232		232		
T_5							233			
V_2				215		215		215		
V_4				218		218		218		
V_5							229			
X_2	18	18	17	17	18	17	17	17	17	
X_3			14	13		15	15	15	15	
X_7					38				38	
X_{20}							217			
X_{22}									249	
EndL		33								
Lesson	31									
∇										accept

DECLARAȚIA PRIVIND ASUMAREA RĂSPUNDERII

Subsemnatul, declar pe răspundere personală că materialele prezentate în teza de doctorat sunt rezultatul propriilor cercetări şi realizări ştiinţifice. Conştientizez că, în caz contrar, urmează să suport consecinţele în conformitate cu legislaţia în vigoare.

Gheorghe Latul

Semnătura 

Data 27.04.2025

CURRICULUM VITAE

Nume Latul	Prenume Gheorghe	Funcţie lector universitar	
Instituţie Universitatea de Stat din Moldova, Facultatea Matematică şi Informatică	Adresa Chişinău, str. Alexei Mateevici 60, MD-2009		Ţara Republica Moldova
Telefon +373 79 215 729	email gheorghe.latul@usm.md	Data naşterii 05.02.1958	

Educaţie

Studii doctorale, Universitatea de Stat din Moldova, Chişinău, specialitatea 121.03 Programarea calculatoarelor, 2006.

Facultate de Matematică şi Cibernetică, Universitatea de Stat din Moldova, Chişinău, specialitatea Matematica aplicată, 1975-1980.

Experienţă profesională

2012 -prezent, lector univ., Departamentul Informatică, Facultate Matematică şi Informatică, Universitate de Stat din Moldova

1994-2012, lector-superior, cercetător ştiinţific, Catedra Tehnologii de Programare, Facultate Matematică şi Informatică, Universitate de Stat din Moldova

1994-1993, lector-superior, cercetător ştiinţific, Facultate Matematică şi Cibernetică, Catedra Limbaje Algoritmice şi Programare, Universitate de Stat din Moldova

1988-1994, lector, Catedra Limbaje Algoritmice şi Programare, Facultate Matematică şi Cibernetică, Universitate de Stat din Moldova

1985-1988, Inginer-programator, Facultate Matematică şi Cibernetică, Universitate de Stat din Moldova

1983-1985, Inginer-programator superior, Centrul de Calcul Interuniversitar al Ministerului Educaţiei

1980-1983, Inginer-programator, Centrul de Calcul Interuniversitar al Ministerului Educaţie

Cercetare

Domeniile de cercetare sunt: limbaje de programare, limbaje de programare specifice domeniului (*eng. domain-specific languages)* , instruirea asistată de calculator, e-Learning.

Publicaţii

Lucrări ştiinţifice şi ştiinţifico-metodice publicate – 49, printre care articole în reviste ştiinţifice şi culegeri – 6, articole în lucrările conferinţelor şi altor manifestări ştiinţifice – 36 manuale – 3, lucrări metodice – 4.

1. **ЛАТУЛ, Г.В**., ПРОКОПАН, Г.А., КРОИТОРУ, И.С., ОСМАТЕСКУ, А.П. Система автоматизированного обучения и обслуживания пользователей ЭВМ на базе ППП АОС/ВУЗ. В сборнике конференции: *Кибернетика и исследование операций в управлении учебным процессом*. Riga: Universitatea Tehnică, 1984. pp. 125-127.

2. **ЛАТУЛ, Г.В**. Опыт разработки автоматизированного обучающего курса по языку ПЛ/1. În: *Применение автоматизированных обучающих систем в учебном процессе. Тезисы докладов межзональной научно-методической конференции*. Минск: БГУ, 1984, pp. 133.
3. БАРДИЕР, Г.Л., КРОИТОРУ, И.С., **ЛАТУЛ, Г.В**., ЯЦКО, Е.И. Разработка и применение автоматизированных учебных курсов – АУК. Методические указания. Кишинев: КГУ, 1984. 42 p.
4. КРАСНОВ, М.А., **ЛАТУЛ, Г.В**. РЫНГАЧ, В.Д. Технология и инструментальная система генерации обучающих программ. În: *Материалы IV Всесоюзного семинара Разработка и применение программных средств ПЭВМ в учебном процессе*. Москва, ИПИАН, 1988, pp.77-78.
5. **ЛАТУЛ, Г.В**., РЫНГАЧ, В.Д. Технология проектирования и автоматическая генерация компьютерных учебных курсов. В сборнике: *Методы и системы технической диагностики. Часть I. Экспертные обучающие системы. Межвузовский сборник научных трудов*. Саратов: СГУ, 1989. pp. 54-57. ISBN 5-292-00143-0.
6. КРАСНОВ, М.А., **ЛАТУЛ, Г.В**., ЛЫСИКОВ, А.Ф., РЫНГАЧ, В.Д., ЛЕМПЕРТ, Р.М. Технологические средства генерации обучающих программ. В сборнике: *Исследования по прикладной математике и информатике*. Кишинев: Штиинца, 1990. pp. 130-138.
7. ЛАТУЛ, Г.В. Генерация программ в диалоге с ЭВМ. În *Conferința științifică a corpului didactico-științific. Tezele referatelor. Totalurile activității științifice în anii 1991/92*. Chișinău. Ch.: Centrul ed. al USM, 1992, P. 55.
8. РЫНГАЧ, В.Д., **ЛАТУЛ, Г.В**., СОКОЛ, В.В. Информационно-обучающая среда „Программирование на Турбо Си”. În *Conferința științifică a corpului didactico-științific. Tezele referatelor. Totalurile activității științifice în anii 1991/92*. Chișinău. Ch.: Centrul ed. al USM, 1992, P. 53.
9. RYNGATCH, V., **LATUL, GH**. The informative-training medium „Turbo C Programming” În: *Computer Technologies in Education. Proceedings of the International Conference on Computer Technologies in Education (ICCTE'93)*. Kiev, 1993, pp. 198-199.
10. ЛАТУЛ, Г.В. Средства для автоматизированной генерации компьютерных учебных курсов. În: *Materialele conferinței tehnico-științifice. „Informatica și tehnica de calcul*. Academia de Științe, Chișinău, 1993, pp. 83-85.
11. RÎNGACI, V.D., **LATUL, GH.V**., SOCOL, V.V. Programare în limbajul Turbo C. Manual în trei părți. Partea 1. Tipurile principale de date, operații și instrucțiuni. Chișinău. Ch.: Centrul ed. al USM, 1993. 85 p.
12. RÎNGACI, V.D., **LATUL, GH.V**., SOCOL, V.V. Programare în limbajul Turbo C. Manual în trei părți. Partea 2. Tipuri de date derivate. Chișinău. Ch.: Centrul ed. al USM,1994. 57 p.
13. RÎNGACI, V.D., **LATUL, GH.V**., SOCOL, V.V. Programare în limbajul Turbo C. Manual în trei părți. Partea 3. Funcții, clase de memorie și preprocesor. Chișinău. Ch.: Centrul ed. al USM,1994. 87 p.
14. **LATUL, GH**. The media for automatic design of computer assisted courses. În: *Computer Technologies in Education*. *Proceedings of the International Conference onComputer Technologies in Education. Part 2. (ICCTE'94)*. Ucraina, Crimea, 1994, pp. 159-160.
15. **ЛАТУЛ, Г.В**. Интерпретация обучающих курсов в системе автоматизированной генерации программ. În: *Materialele conferinței corpului didactico-științific. Bilanțul activității științifice a U.S.M. pe anii 1993-1994. Științe naturale*. Chișinău. Ch.: Centrul ed. al USM, 1995, p. 60.
16. RÂNGACI, V., KRASNOV, M, **LATUL, GH**., LÎSICOV, A., GONCIARENCO R. Elaborarea softului instrumental pentru crearea programelor de instruire asistate de calculator. Darea de seamă (finală). Nr. înregistrării de stat 0195M00126, Nr. de inventar 0396M00141. Chișinău, USM, 1995. 71 p.

17. RÂNGACI, V., **LATUL, GH**., SIBIRSCHI, T., GONCIARENCO, R., FAN, PHUONG. DAT. Elaborarea CASE-tehnologiei de proiectare a cursurilor de instruire asistate de calculator. Darea de seamă. Nr. înregistrarii de stat 0196M00877, Nr. de inventar 0298M00736. Chișinău, USM, 1997. 54 p.
18. **ЛАТУЛ, Г**. Модели обучения для компьютерных учебных курсов În: *Conf. corpului didactico-ştiinţific Bilanţul activităţii ştiinţifice a USM pe anii 1996/97. Rezumatele comunicărilor. Ştiinţe naturale*. Chișinău. Ch.: Centrul ed. al USM, 1998. pp. 70.
19. ЛАТУЛ, Г., ПУШКАШУ, Д. Представление баз данных в компьютерных сетях În: *Bilanţul activităţii ştiinţifice a USM pe anii 1996/97. Conferinţa corpului didactico-ştiinţific . Rezumatele comunicărilor. ştiinţe naturale*. Chișinău. Ch.: Centrul ed. al USM,1998. pp. 71.
20. **LATUL, GH**. Modelul aprecierii cunoștințelor în cursuri asistate de calculator. În: *Anale ştiinţifice ale Universităţii de Stat din Moldova. Seria „Ştiinţe fizico-matematice”*. Chișinău. Ch.: Centrul ed. al USM, 1998, pp. 36-41. ISBN 9975-917-04-6
21. **ЛАТУЛ, Г**. Модели урока при проектировании компьютерных учебных курсов. În: *Anale ştiintifice ale Universităţii de Stat din Moldova. Seria „Ştiinţe fizico-matematice”*. Chișinău. Ch.: Centrul ed. al USM, 1999, pp. 252-256. ISBN 9975-917-26-7
22. RÂNGACI, V. **LATUL, GH**., GONCEARENCO, R. Elaborarea mijloacelor instrumentale și aplicative cu destinația instructivă pe baza rețelelor de calculatoare. Darea de seamă (finală). Nr. înregistrarii de stat 0196M00877, Nr. de inventar 0201MD01363. Chișinău, USM, 2000. 82 p.
23. **LATUL, GH**. Utilizarea intreruperilor in Asamblor IBM PC: Indicati metodice pentru lucrarile de laborarator. Chișinău. Ch.: Centrul ed. al USM, 2000 . 27 p.
24. **LATUL, GH**., CARELOVA, L. Limbajul de descriere a situațiilor instructive și analizorul lui. În: *Conferinţa corpului didactico-ştiinţific. Bilanţul activităţii ştiinţifice a USM pe anii 1998/99. Rezumatele comunicărilor. Ştiinţe fizico-matematice*. Chișinău. Ch.: Centrul ed. al USM, 2000, pp. 117-118. ISBN 9975-917-58-5.
25. **LATUL, GH**. Generarea paginilor Web după descrierea scenariilor de instruire. În: *Conferinţa corpului didactico-ştiinţific. Bilanţul activităţii ştiinţifice a USM pe anii 1998/99. Rezumatele comunicărilor. Ştiinţe fizico-matematice*. Chișinău. Ch.: Centrul ed. al USM, 2000, pp 119-120. ISBN 9975-917-58-5.
26. **ЛАТУЛ, Г**. Инструментарий для создания компьютерных учебных курсов. În: *Tehnologii avansate în pragul secolului XXI. Materialele conferinţei ştiinţifico-practice*. Chișinău: Î.E.P. Ştiința, 2000, pp. 22-24. ISBN 9975-67-215-9.
27. **ЛАТУЛ, Г. В**. Использование шаблонов при создании компьютерных учебных курсов. În: *Conferinţa corpului didactico-ştiinţifi. Bilanţul activităţii ştiinţifice a USM în anii 2000 2003. Rezumatele comunicărilor. Ştiinţe fizico-matematice*. Chișinău. Ch.: Centrul ed. al USM, 2003, pp. 188-189. ISBN 9975-70-265-1.
28. **ЛАТУЛ, Г**. Проблемы организации анализа ответов обучаемых в компьютерных учебных курсах. În:*Analele ştiintifice ale Universităţii de Stat din Moldova. Seria „Ştiinţe fizico-matematice”*. Chișinău. Ch.: Centrul ed. al USM, 2004. pp. 132-136. ISSN 1811-2641
29. **ЛАТУЛ, Г**. Язык описания компьютерных учебных курсов. În:*Analele ştiintifice ale Universităţii de Stat din Moldova. Seria „Ştiinţe fizico-matematice”*. Chișinău. Ch.: Centrul ed. al USM, 2004.pp. 129-131. ISSN 1811-2641
30. **ЛАТУЛ, Г**. Проблемы проектирования компьютерных учебных курсов при помощи шаблонов. În:*Conferinţa Internaţională Tendinţele de Dezvoltare a Societăţii Informaţionale Academiei de Studii Economice*. Chișinău: ASEM, 2004. pp. 180-181. ISBN 9975-75-265-9
31. **ЛАТУЛ, Г**., ГОНЧАРЕНКО, Р. Информационные технологии и доступность образования. În: *Standardele şi calitate în învăţământ continuu: Materialele conferinţei internaţionale. şt. – practice.* Chișinău.: Institutul de Instruire Continuă, 2005. pp. 249-251.

32. LATUL, GH. Modelul aprecierii cunoștințelor în cursurile asistate de calculator. În: *Conferința Ştiiţifico-practică „Problemele actuale ale dezvoltării social – economice a Republicii Moldova”, 18-19 noiemb., 2005*. Chișinău, Ch.: Centrul ed. al USM, 2006. pp.146-151.
33. **LATUL, GH**. Perspectivele instruirii la distanță în Republica Moldova. În: *Învățământul superior și cercetarea – piloni ai societății bazate pe cunoaștere. Conferința științifică internațională. Rezumatele comunicărilor Științe Reale*. Chișinău. Ch.: Centrul ed. al USM, 2006. pp. 53-54. ISBN 978-9975-70-670-4.
34. РЫНГАЧ, В.Д., **ЛАТУЛ, Г.**В. Программирование на языке Си: Учебное пособие. Chișinău. Ch.: Centrul ed. al USM, 2006. P.188.
35. BRAGARU, T., CĂPĂȚÂNĂ, GH., PLEȘCA, N., LATUL, GH., CÎRHANA, V., EFROS, I., CRĂCIUN, I., BABAN, T., BARON, GH. Învățământ la distanță: concept și terminologie. (Ghid de inițiere). Chișinău, Ch.: Centrul ed. al USM, 2008. 101 p., ISBN 978-9975-70-764-0.
36. BRAGARU. T., **LATUL. GH**. Crearea cursurilor de instruire la distanta. In: *Conference „Mathematics & Information Technologies: Research and Education” (MITRE-2008). Chișinău, October 1-4 October, 2008.* Abstracts. Chișinău, Ch.: Centrul ed. al USM, 2008, pp. 71-72. ISBN 978-9975-70-752-4
37. **LATUL, GH**., PERETEATCU, A., PERETEATCU, S. Programare orientată pe obiecte (în baza C++): Suport de curs. Chișinău, Ch.: Centrul ed. al USM, 2010. 201p. ISBN 978-9975-70-926-2
38. BRAGARU, T., LATUL, GH. Dezvoltarea cursurilor de instruire la distanță. In: *Proceedings of the Conferences „Mathematics & Information Techonilogies: Research and Education” (MITRE 2008 – 2009.* Chișinău, Ch.: Centrul ed. al USM, 2010, pp. 149 -1 54, ISBN 978-9975-70-995-8
39. CĂPĂȚÂNĂ, GH., **LATUL, GH**. Sisteme inteligente de e-learning. In: *Mathematics & Information Techonilogies: Research and Education (MITRE 2011)*. Abstracts. Chișinău, Ch.: Centrul ed. al USM, 2011, pp. 157 -158, ISBN 978-9975-71-144-9.
40. **LATUL GH**. Learning material presentation in computer assisted courses interpretation with XML aid. In *Mathematics & Information Techonilogies: Research and Education (MITRE-2013). International Conference.*Abstracts. Chișinău, Ch.: Centrul ed. al USM, 2013, pp. 110-111, ISBN 978-9975-71-411-2.
41. GONCEARENCO, R., STURZA, G., BUTNARU, M., **LATUL, GH**. Programare în limbaj de asamblare: Îndrumar pentru lucrările de laborator. Chișinău, Ch.: Centrul ed. al USM, 2014. 94p. ISBN 978-9975-71-525-6.
42. **ЛАТУЛ, Г.** Построение генератора компьютерных учебных курсов. În: *Integrare prin cercetare și inovare, conferință științifică națională cu participare internațională Rezumatele ale comunicărilor. Științe ale naturii. Științe exacte*. Chișinău. Ch.: Centrul ed. al USM, 2014, pp. 150 -151. ISBN 978-9975-71-573-7.
43. CĂPĂȚÂNĂ, GH., **LATUL, GH**., PERETEATCU, S. Analiza răspunsurilor grafice în cursuri de instruire asistate de calculator. În: *International Conference „Mathematics & Information Technologies: Research and Education” (MITRE - 2015)*. Chișinău. Ch.: Centrul ed. al USM, 2015, pp. 109, ISBN 978-9975-71-678-9.
44. CĂPĂȚÂNĂ, GH., **LATUL GH**. Frame using in the language for computer assisted courses programming. În: *Mathematics & Information Technologies: Research and Education (MITRE 2016). International conference.* Abstracts. Chișinău. Ch.: Centrul ed. al USM, 2016, pp. 77, ISBN 978-9975-71-794-6.
45. ГОНЧАРЕНКО, Р., СТУРЗА, Г., БУТНАРУ, М., **ЛАТУЛ,** Г. Программирование на языке Ассемблер: Лабораторный практикум. Chișinău, Ch.: Centrul ed. al USM, 2016. 90 p. ISBN 978-9975-71-800-4.
46. CROITOR, Mihail, BRAGARU, Tudor**, LATUL, Gheorghe**. Management system for bank of items and tests. In: *Mathematics & Information Technologies: Research and Education (MITRE*

2019), International conference (2019; Chișinău). Abstracts. Chișinău, Ch.: Centrul ed. al USM, 2019, pp. 74, ISBN 978-9975-149-17-4.

47. **LATUL, Gheorghe**, BRAGARU Tudor, CROITOR MihaiL. Objective items development system. In *Mathematics & Information Technologies: Research and Education (MITRE 2019). International Conference*. Chișinău. Ch.: Centrul ed. al USM, 2019. 104p, P. 77-78, ISBN 978-9975-149-17-4.
48. **LATUL, GH**. Model of authoring system of computer assisted courses generator. In *Mathematics & Information Technologies: Research and Education (MITRE 2023). International conference*. Chișinău. Ch.: Centrul ed. al USM, 2023, pp. 81-82, ISBN 978-9975-62-535-7.
49. **LATUL, GH**. Modelul conceptual și arhitectura platformei no-code de generare automată a cursurilor de instruire asistată de calculator. În: *Intellectus*, nr. 1, 2024, p. 147 – 156. ISSN 1810-7087, Tip B. Accesibil la adresa: https://agepi.gov.md/ro/intellectus/intellectus-1-2024/modelul-conceptual-%C8%99i-arhitectura-platformei-no-code-de-generare>
50. **LATUL, GH**. Reprezentarea cunoștințelor în cursurile de instruire asistată de calculator. În: *Acta et commentationes, seria Științe ale Educației*, nr. 1(35), 2024, p. 103-108. ISSN 1857-0623, E-ISSN 2587-3636, Tip B. Accesibil la adresa: https://revistaust.upsc.md/index.php/acta_educatie/article/view/979/958>