Revista Română de Interacţiune Om-Calculator 6 (1) 2013, 23-34 

# Cuplarea componentelor active pentru realizarea mediilor inovative de învățare

Ion Smeureanu[1], Narcisa Isăilă[2]

[1]Academia de Studii Economice Bucureşti
Piaţa Romană nr.6, Bucureşti
*E-mail: smeurean@ase.ro*

[2]Universitatea Creştină „Dimitrie Cantemir" Bucureşti
Splaiul Unirii nr. 176, Bucureşti
*E-mail: isaila_narcisa@yahoo.com*

**Rezumat.** Mediile inovative de învățare reprezintă o preocupare atât a educatorilor cât şi a dezvoltatorilor, această preocupare fiind influențată şi de tehnologiile informaționale aflate într-o continuă dezvoltare. Pe de altă parte, legislația europeană încurajează crearea mediilor de învățare bazate pe metode pedagogice inovative, avîndu-se în vedere principiul "învățării pe tot parcursul vieții". În strânsă legătură cu scopul educației, din punct de vedere informatic, se doreşte ca prin cuplarea componentelor de învățare la nivel de lecție să se diminueze efortul de integrare a resurselor educaționale într-un mediu inovativ, puternic personalizat şi adaptat nevoilor de învățare a studenților.Lucrarea îşi propune să analizeze măsura în care tehnologia componentelor permite realizarea acestui deziderat, avîndu-se în vedere beneficiile oferite studenților dar şi profesorilor prin eliberarea capacităților de creativitate şi inovativitate.

**Cuvinte cheie**: componente complexe de învățare, cuplarea componentelor, „point & shoot", „drag & drop", matematică, inovativitate.

## 1. Proiectarea de arhitecturi software

Potrivit lui Garlan & Shaw (1993), stilul (modelul) arhitectural se defineşte prin componentele şi conectorii utilizați, împreună cu un set de constrângeri referitoare la modul în care componentele pot fi combinate.

Literatura de specialitate abordează modelele arhitecturale comune, în principal după elementul cheie specific acestora (Tabel 1), cu avantajele şi dezavantajele pe care le oferă utilizarea lor în realizarea aplicațiilor dorite, însă în practică se utilizează variante combinate de stiluri (Microsoft, 2009).

Tabel 1. Gruparea modelelor arhitecturale

| Aria de focalizare | Modelul arhitectural |
|---|---|
| Structura | Component-Based<br>Object-Oriented<br>Layered Architecture |
| Comunicare | Service-Oriented Architecture (SOA)<br>Message Bus |
| Implementare | Client- Server<br>3-Tier<br>n- Tier |
| Domeniu | Domain Driven Design |

Astfel, o arhitectură SOA (Service-Oriented Architecture) poate fi compusă din servicii dezvoltate folosind stilul *layered* (organizat pe straturi) şi stilul *object-oriented* (orientat- obiect). De asemenea, se poate alege stilul *3-Tier* (pe trei nivele) sau *n-Tier* (pe n nivele) când accentul este pus pe cerințele de securitate.

Necesitatea proiectării de arhitecturi software se datorează în principal dezvoltării de sisteme informatice tot mai complexe, puternic personalizate. Deşi personalizarea poate fi realizată la diferite niveluri (Fig. 1), alături de tehnologia folosită sunt necesare cunoştinţe specifice domeniului.

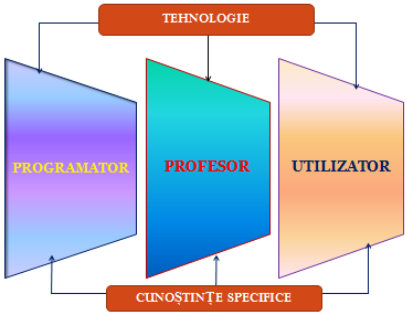


*Figura 1. Niveluri de realizare a personalizării aplicațiilor e-learning*

Combinația *client-server*, pentru sistemele distribuite şi *component-based architecture* (arhitectura bazată pe componente), pot fi soluția pentru folosirea de componente care expun interfețe de comunicare adecvate, iar abordarea *object-oriented*, îmbunătăţeşte reutilizarea şi flexibilitatea lor.

În cazul aplicațiilor de e-learning personalizarea asigură eficientizarea învățării şi pentru a realiza acest lucru sunt necesare eforturi considerabile, care în mare parte depind de tehnologia informatică folosită.

Proiectarea unei arhitecturi software vizează distribuirea funcționalității pe componente, însă în egală măsură presupune acordarea unei atenții sporite şi altor aspecte precum: accesul la date, protocoalele de comunicație între module, controlul global al structurii de execuție, etc., foarte importante în asigurarea calității acesteia (Fig.2).

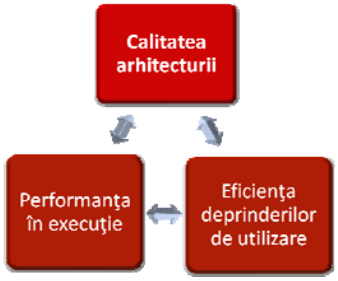


*Figura 2. Influența calității arhitecturii software*

De toate acestea depind performanţa în execuție şi eficiența formării deprinderilor în utilizare, mai ales în sistemele cu utilizare repetată şi pe măsura forțării scalabilității.

Deoarece noile tehnologii de programare oferă modalități noi de comunicare între componentele software, este necesară o modelare mai strictă a arhitecturii software dincolo de simple reguli şi principii.

Crearea platformelor, incluzând diferite tipuri de obiecte care se conectau pe baza unor restricții sau reguli, a permis experimentarea diverselor arhitecturi de comunicare. S-au identificat astfel diferite pattern-uri (mostre), comunicarea făcându-se prin intermediul unor interfețe predefinite.

La baza acestei comunicări stă apelul prin metode sau proceduri, apel care poate fi clasic (explicit) sau bazat pe evenimente (implicit) recunoscute în sistem (Garlan & Shaw, 1993). Problemele apărute în acest tip de arhitectură se referă atât la necunoaşterea de către elementul notificator a reacției componentelor la evenimente şi a ordinii în care acestea reacționează. În consecință, algoritmul nu trebuie să se bazeze pe o succesiune dată.

Un alt tip de arhitectură folosit se referă la organizarea pe straturi a componentelor (layered systems) care interacționează bazându-se pe interfețe generice, asemeni modelului conectării componentelor hardware. Cea mai cunoscută arhitectură este 3- tired (interfață, date, logică de business) cu avantajele pe care le oferă, respectiv: scalabilitate, mentenabilitate, flexibilitate crescută (Microsoft, 2009).

## 2. Arhitecturile bazate pe componente active

Componentele reprezintă entități software care dispun de capacități de autocunoaştere (self-contained), ceea ce le permite integrarea automată într-un context dat, atât din punct de vedere tehnic, cât şi conceptual.

Integrarea, respectiv cuplarea acestora, se realizează de la executabil la executabil pe baza unor interfețe standard de comunicare, deoarece atât componenta cât şi mediul de integrare beneficiază de fire de execuție. Rezultatul cuplării componentelor sau cuplarea acestora cu mediul de integrare constă în posibilitatea componentelor de a efectua schimb de funcționalități. Deşi această integrare se poate produce tehnic, background-ul este unul semantic-funcțional şi caracterizează domeniul pe care îl modelează aplicația software. Acest tip de software, fiind mai aproape de domeniul modelat şi automatizând partea tehnică a cuplării, facilitează inovarea, antrenând în acest proces specialişti, care nu sunt tehnicieni şi permițând testarea diverselor combinații de asamblare.

Dincolo de viteza crescută de dezvoltare a software-ului bazat pe componente, tehnologia oferă şi o soluție nouă problemei programării multilimbaj (prin compilarea într-un limbaj intermediar comun) şi a eterogenității platformelor de rulare (prin utilizarea unei maşini virtuale).

Software-ul orientat pe componente permite identificarea acelor componente a căror funcționalitate nu mai corespunde noilor cerințe şi înlocuirea punctuală şi reasamblarea lor. Modelul general este cel oferit de tehnologia MVC (Model- View- Controller), care permite separarea modelului de business de partea de vizualizare, respectiv interfața de cuplare, în ideea că pot avea grade de uzură diferite (Smeureanu & Isăilă, 2013).

Încapsularea accentuată, respectiv granularitatea mare, conduce la înlocuirea unor părți prea mari din software, deşi există porțiuni care nu ar

trebui schimbate. În acest caz, se recomandă crearea unor componente mai mici, care pot fi manipulate unitar şi integrat prin cuplare.

Pe de altă parte, aceste conglomerate pot fi descompuse şi recompuse într-o nouă formulă cu eventuale schimbări interne.

Aşa cum compunerea componentelor active se face automatizat (prin participarea activă a ambelor părți care se cuplează), tot aşa de uşor ar trebui să se facă şi descompunerea pe subcomponente, automatizând măcar parțial procesul de reinginerie (Smeureanu & Isăilă, 2013).

Stilul de arhitectură bazat pe componente se axează pe descompunerea în componente funcționale sau logice, care conțin metode, evenimente şi proprietăți şi expun interfețe de comunicare bine-definite. Acest mod de organizare oferă un nivel de abstractizare mai ridicat decât în cazul stilului orientat- obiect.

Principiul acestei arhitecturi se bazează pe caracteristicile componentelor, care sunt (Microsoft, 2009):

- reutilizabile în diferite aplicații;
- înlocuibile cu alte componente similare;
- utilizabile în diferite medii şi contexte de lucru;
- extensibile, obținându-se componente cu noi comportamente;
- încapsulate, în sensul că nu oferă detalii referitoare la procese sau variabile interne deşi permit apelantului să utilizeze funcționalitățile componentelor prin intermediul interfețelor pe care le expun;
- independente din punct de vedere al utilizării în orice mediu fără a afecta utilizarea altor componente sau sisteme.

În aplicații, de obicei, se folosesc componente pentru interfața cu utilizatorul numite şi controale, componente de ajutor şi utilitare, care expun un subset specific de funcțiile utilizate în alte componente.

Componentele depind de un mecanism în cadrul platformei, ca mediu în care se pot executa, astfel în Windows sunt Component Object Model (COM) şi Distributed Component Object Model (DCOM), iar pe alte platforme mai cunoscute sunt: Common Object Request Broker Architecture (CORBA) şi Enterprise JavaBeans (EJB).

Microsoft .NET Framework oferă suport pentru aplicațiile care folosesc o abordare bazată pe componente.

Stilul arhitectural bazat pe componente oferă o serie de avantaje deoarece (Microsoft, 2009):

• este un stil uşor de implementat permiţând înlocuirea versiunilor existente fără a afecta alte componente sau sistemul ca întreg;
• permite reducerea costurilor pentru dezvoltare şi întreţinere;
• arhitectura este uşor de dezvoltat;
• modelul permite reutilizarea componentelor în alte aplicaţii sau sisteme;
• complexitatea tehnică a acestui model arhitectural este redusă datorită componentelor şi a serviciilor pe care le oferă.

Pentru a gestiona dependenţele între componente şi pentru a promova cuplajul slab şi reutilizarea pot fi folosite modele precum *Dependency Injection* sau *Service Locator* (Microsoft, 2009).

Aceste modele se utilizează pentru a construi aplicaţii mixte bazate pe combinarea refolosirii componentelor în mai multe aplicaţii. Devine posibilă astfel crearea unei arhitecturi conectabile sau compozit, care să permită înlocuirea rapidă şi actualizarea componentelor individuale (Microsoft, 2009).

## 3. Cuplarea componentelor active la nivel de platformă

Abordarea bazată pe componente se axează pe reutilizarea componentelor software deja create, ceea ce favorizează realizarea de aplicaţii personalizate în condiţii de timp şi efort scăzut de realizare şi cu costuri mult diminuate.

Un alt avantaj major al utilizării componentelor create, care rulează în medii distribuite, se referă la faptul că nu este necesară duplicarea codului.

Importantă devine realizarea conexiunilor între componente, respectiv gradul de dependenţă între module. Pentru măsurarea gradului de dependenţă se utilizează numărul parametrilor de intrare - ieşire, numărul de variabile globale şi numărul de module care apelează sau cele care sunt apelate.

După cauza care stă la baza cuplării componentelor, se pot realiza următoarele tipuri de cuplări între componente (Shingade, 2007):

- *Cuplarea prin schimb de mesaje (fără parametri)*, cazul componentelor interdependente din arhitectura orientată pe servicii, care folosesc metode publice pentru a schimba mesaje fără parametri. Acest tip de cuplare susţinut de arhitectura orientată pe servicii asigură nivelul cel mai scăzut de cuplare.
- *Cuplarea prin intermediul datelor sau argumentelor*, caz în care schimbul se face în mod direct;

- *Cuplarea structurilor de date* la nivel de obiect;
- *Cuplarea prin includerea codului sursă* în altă componentă;
- *Cuplarea de conținut*, în care cuplarea implică modificarea datelor interne ale unei componente de către cealaltă componentă; este cel mai puternic tip de cuplare.
- *Cuplarea de control*, bazată pe controlul, prin comenzi sau indicatori, a acțiunii procedurilor apelate; acest tip de cuplare prezintă dezavantajul schimbării procedurii apelate la fiecare adaugare a unei noi comenzi de către apelator;
- *Cuplarea globală* (prin variabile globale), în care dependența componentelor se realizează prin setul de date pe care le partajează ;
- *Cuplarea prin structuri*, care este asemănătoare cuplării globale cu diferența că se partajează doar tipurile de date definite global;
- *Cuplarea prin importarea unei componente*, care implică declararea importării acesteia de către cealaltă componentă; acest tip de cuplare este mai slabă decât cuplarea prin includerea codului sursă;
- *Cuplarea externă*, caz în care o componentă software este dependentă de o orice altă componentă realizată sau este dependentă de un anumit tip de hardware.

O cuplare slabă între componente se realizează atunci când complexitatea lor şi numărul de relații sunt reduse.

Sunt considerate componente active acele obiecte care pe parcursul instanțierii sau în interacțiunea cu utilizatorul pot coopera, iar prin cuplare dinamică permit realizarea unor acțiuni complexe, care nu pot fi executate de către componente în mod individual.

Potrivit lui Signer & Norrie (2009) cuplarea componentelor se poate face de la aplicație la aplicație, de la aplicație la obiecte de date sau de la aplicație la diferite periferice client.

Din punct de vedere al mecanismelor folosite pentru cuplarea componentelor cele mai utilizate sunt: „*drag & drop*" şi „*point & shoot*", cu diferența că cel de-al doilea mecanism nu implică deplasarea simbolurilor grafice asociate.

Vizualizarea tuturor legăturilor (existente sau posibil de stabilit datorită cooperării posibile între componentele sursă şi destinație) reprezintă o opțiune oferită utilizatorului de către unele platforme de lucru.

Mai mult chiar, recunoaşterea capacității de cooperare între componente se poate face pe baza metadatelor, stabilite la definirea claselor aferente

componentelor active sau dinamic în baza unor ontologii specifice domeniului de lucru în care activează componentele. Pe de altă parte, componentele active pot proveni din diferite clase de programare deoarece rulează sub o maşină virtuală.

Pentru a simula interacţiuni complexe, o platformă cu componente active dispune de un software cu execuţie continuă în background (fundal) asistând utilizatorul la cuplarea de componente.

În cazul platformelor de e-learning acesta asistă profesorul sau studentul în realizarea de lecţii complexe prin combinarea componentelor active de învăţare (Smeureanu & Isăilă, 2012).

Mecanismul bazat pe componente active asigură simplificarea procesului de reutilizare deoarece componentele sunt în măsură:

- să-şi expună capabilităţile printr-o interfaţă specifică;
- să identifice în mediul de lucru alte componente prin care să-şi extindă propriile funcţionalităţi;
- să propună variante de cuplare pe care utilizatorul să le accepte sau nu, diminuând astfel efortul de programare.

Astfel, un utilizator fără cunoştinţe de programare poate să creeze aplicaţii informatice complexe, care din punct de vedere al e-learning-ului se referă la posibilitatea realizării de lecţii complexe, punând astfel în valoare experienţa individuală şi particularizarea învăţării.

Prin capacitatea de identificare a altor componente, componentele active devin context- sensitive. Expunerea capabilităţilor proprii ar trebui să fie atât de bună încât să permită cuplarea unor componente complet noi, care să nu mai fi fost folosite pe platformă. Provocarea constă în a anticipa nevoile de programare ale utilizatorului, folosind eventual template-uri (şabloane) de utilizare. Aşadar şi mediul de utilizare trebuie să fie activ, în sensul sintetizării unor template-uri pe măsura învăţării prin experiment.

În termenii învăţării asistate acest lucru corespunde distincţiei dintre formularea problemei şi rezolvarea problemei pe baza unor modele.

Spaţiul de lucru devine astfel, unul îmbogăţit în informaţie (Robertson et al., 1993) capabil să furnizeze acces rapid la facilităţile oferite prin componentele disponibile. În funcţie de obiectivul învăţării template-urile de utilizare preiau rolul structurării şi ierarhizării dinamice a informaţiei.

Din punct de vedere al cunoaşterii informaţiei despre existenţa componentelor, putem ierarhiza componentele astfel (Smeureanu & Isăilă, 2012):

- componente cunoscute de utilizator, deoarece le-a mai folosit şi le poate identifica uşor;
- componente vag cunoscute de utilizator, dar identificate uşor de către mediul de programare deoarece se înscriu obiectivelor contextuale de lucru; selectarea lor se face de către utilizator la propunerile oferite de mediu;
- componente necunoscute de către utilizator şi inexistente în mediu, dar posibil de identificat prin căutarea după funcționalități în baza unor ontologii specifice domeniului de lucru.

Aşadar, în sistemele active cererile de căutare a componentelor software pot aparține utilizatorului, mediului de programare sau unor componente deja activate care ştiu despre existența altor componente cu care pot colabora. De fiecare dată trebuie găsit un echilibru în a lăsa utilizatorului libertatea de gândire şi inițiativa mediului de a propune alternative, astfel încât utilizatorul să nu se simtă invadat de inițiativele mediului (Smeureanu & Isăilă, 2012).

## 4. Folosirea tehnicii „point & shoot" în realizarea de obiective complexe

Programarea orientată obiect oferă posibilitatea moştenirii comportamentului unui control şi dezvoltarea acestuia pentru a îndeplini funcționalități noi specifice doar obiectelor derivate.

Când obiectivul dorit este unul complex, provenind din compunerea comportamentelor mai multor controale, derivarea nu mai este suficientă, preferându-se crearea aşa numitelor controale de utilizator.

Situația se complică atunci când obiectivul complex îl stabileşte utilizatorul la momentul execuției, neputând fi anticipat la momentul link-editării. Soluția constă în cuplarea mai multor componente care cooperează la realizarea obiectivului complex; problema echivalează cu legarea întârziată a unor instanțe de obiecte de clase diferite, ce pot comunica printr-o interfață simplă, schimbând date de obicei în format extern sau în formate standard cu acceptabilitate mai largă (XML, LaTeX, etc).

Legarea dinamică aplicată în domeniul e-learning permite educatorilor să asambleze şi să distribuie obiecte pedagogice complexe fără nici un efort de programare. Obiectele care se cuplează sunt aduse într-un container comun dintr-un repository (depozit de componente) propriu sau din Internet.

Cuplarea lor se face frecvent prin acțiunea de „drag & drop”, prin care un obiect sursă este plasat peste un obiect destinație, pentru a stabili o conexiune între cele două obiecte; după stabilirea conexiunii, poziția fizică a obiectului sursă revine însă la cea originală.

O altă modalitate de indicare a modului de cuplare a obiectelor este tehnica de „point & shoot”; lucrul cu mouse-ul este similar acțiunii de „drag & drop”. Clic-ul de mouse pe obiectul sursă este urmat de o deplasare cu butonul mouse-ului apăsat până la poziția obiectului destinație.

Spre deosebire de „drag & drop” obiectul sursă nu se deplasează; la începutul operațiunii (dragării), contextul este vizibil grafic printr-o linie blinking (pâlpâire), care începe în obiectul sursă şi se termină în poziția curentă a mouse-ului. La finalul operației de „point & shoot”, linia va puncta (pointa) pe obiectul destinație, dispărând odată cu eliberarea butonului mouse-ului. Aşadar, „point & shoot” are propria semantică, diferită de cea a tehnicii „drag & drop”, stabilind o conexiune permanentă între componentele de învățare.

Decuplarea obiectelor se poate face în aceeaşi manieră doar că în faza finală butonul mouse-lui va fi undeva în afara oricărui obiect capabil să comunice cu obiectul sursă. Pe de altă parte, decuplarea se realizează automat la stabilirea unei noi conexiuni permanente între obiectul sursă şi un alt obiect destinație de acelaşi tip. Prin urmare, sunt permise mai multe conexiuni de la acelaşi obiect sursă către obiecte destinație dar care sunt de tipuri diferite.

## 5. Studiu de caz: învățarea matematicii

Pentru învățarea matematicii importanță deosebită are folosirea tehnologiei componentelor deoarece obiectele sunt disponibilizate sub formă de componente şi pot fi interconectate.

În învățarea asistată de calculator se pot realiza aplicații complexe şi aceasta datorită faptului că arhitectura bazată pe componente distribuie funcționalități pe componente (Fig.3).

Colaborarea dintre componentele: *Equation- MathTTS- Sintetizator vocal* pentru redarea vocală a expresiilor matematice în limba română reprezintă un exemplu în acest sens. Componenta *Equation*, permite introducerea de expresii matematice, prin manipularea simbolurilor grafice şi generarea codului text asociat simbolurilor şi funcțiilor matematice uzuale.

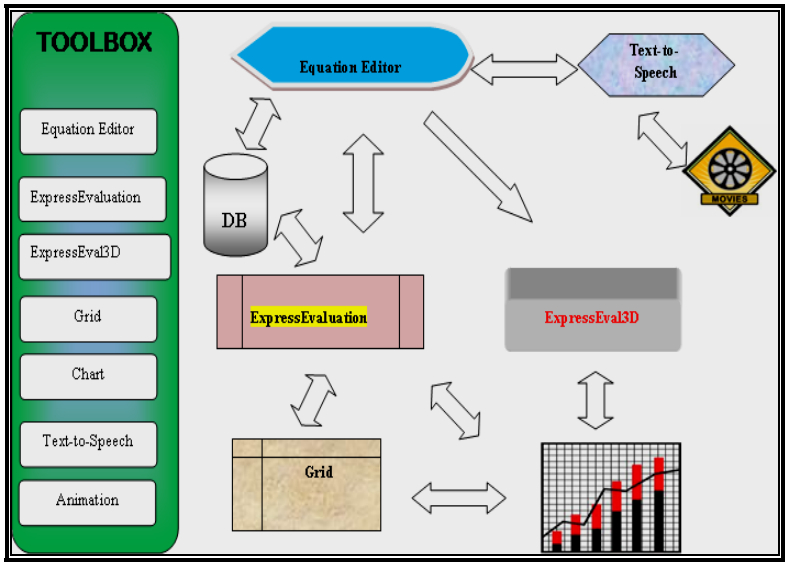


*Figura 3. Varianta de interconectare a componentelor în învățarea matematicii*

Expresiile matematice, ca obiecte, pot fi stocate în *baza de date*, de unde pot fi uşor recuperate şi folosite pentru reconstruirea formei vizuale a ecuației sau generarea formei matematice uzuale.

Forma textuală generată de componenta *Equation* va fi preluată de componenta *MathTTS* şi convertită într-un text în limba română (inclusiv pronunția simbolurilor şi elementele de prozodie), care poate fi încărcat în *sintetizatorul vocal* şi astfel expresiile matematice sunt redate vocal. Această abordare a interacțiunii între componente este importantă pentru realizarea unor aplicații utile de accesibilizare a informației în instruirea asistată de calculator pentru persoanele cu dizabilități vizuale.

O altă colaborare poate fi între componentele *Equation- Function-EvalExpress*. Evaluatorul de expresii matematice EvalExpress preia funcția ca formulă matematică, o compilează, face corecțiile necesare şi generează cod pentru stocarea obiectului *Function* într-un format intern, eficient în evaluarea de către calculator. Componenta *Function* poate fi cuplată cu controale de tip *Chart* (pentru reprezentarea grafică a valorilor uneia sau mai multor funcții) sau de tip *Data GridView*. Formei numerice a funcției îi corespunde intern un obiect *DataSet*, iar ca vizualizare numerică acestei componente îi poate corespunde în aplicație o componentă *GridView*, care permite interacțiunea cu utilizatorul pentru diferite ajustări (Smeureanu &

Isaila, 2011). Datorită capacității de cuplare, componentele pot fi accesate ca servicii web pe platforme e-learning, ceea ce explică importanța utilizării lor în învățarea asistată de calculator.

## 6. Concluzii

Cuplarea componentelor software active la nivelul unei platforme de învățare, prin folosirea tehnicilor de cuplare a componentelor, asigură obținerea rapidă de produse noi cu funcționalități noi.

În domeniul educației, utilizarea tehnologiei componentelor reprezintă soluția la schimbările privind cerințele de software iar uşurința în manipularea şi compunerea obiectelor de învățare de către profesori le deschide calea spre inovare şi creativitate în cadrul procesului didactic.

## References

Garlan, D., Shaw, M. (1993). An Introduction to Software Architecture, Advances in Software Engineering and Knowledge Engineering, Vol. I, 1- 39, New Jersey.

Microsoft Corporation (2009). Microsoft Application Architecture Guide, 2nd Edition, accessed to http://www.microsoft.com/en-us/download/.

Robertson, G. G., Card, S. K. & Mackinlay, J. D. (1993), Information Visualization Using 3D Interactive Animation, Communications of the ACM 36(4), 57–71.

Shingade, M. (2007). Design loosely coupled modules- Object oriented systems, http://www.codeproject.com/Articles/19783/Design-Loosely-Coupled-Modules-Object-Oriented-Sys

Signer, B., Norrie, C., M. (2009), Active Components as a Method for Coupling Data and Services-A Database Driven Application Development Process, Proceedings of ICOODB 2009, Zurich, Switzerland, 59-76.

Smeureanu, I., Isăilă, N. (2011). New information technologies for an innovative education, World Journal on Educational Technology, vol.3 issue 3, 177- 189.

Smeureanu, I., Isăilă, N. (2012). Learning environment based on active components, Proceedings of IE 2012, Bucharest Romania, 464- 468.

Smeureanu, I., Isăilă, N. (2013). Re-engineering of assisted learning software, Proceedings of IE 2013, Bucharest Romania, 275 – 279.