Stichting NIOC en de NIOC kennisbank

Stichting NIOC (www.nioc.nl) stelt zich conform zijn statuten tot doel: het realiseren van congressen over informatica onderwijs en voorts al hetgeen met een en ander rechtstreeks of zijdelings verband houdt of daartoe bevorderlijk kan zijn, alles in de ruimste zin des woords.

De stichting NIOC neemt de archivering van de resultaten van de congressen voor zijn rekening. De website www.nioc.nl ontsluit onder "Eerdere congressen" de gearchiveerde websites van eerdere congressen. De vele afzonderlijke congresbijdragen zijn opgenomen in een kennisbank die via dezelfde website onder "NIOC kennisbank" ontsloten wordt.

Op dit moment bevat de NIOC kennisbank alle bijdragen, incl. die van het laatste congres (NIOC2023, gehouden op donderdag 30 maart 2023 jl. en georganiseerd door NHL Stenden Hogeschool). Bij elkaar bijna 1500 bijdragen!

We roepen je op, na het lezen van het document dat door jou is gedownload, de auteur(s) feedback te geven. Dit kan door je te registreren als gebruiker van de NIOC kennisbank. Na registratie krijg je bericht hoe in te loggen op de NIOC kennisbank.

Het eerstvolgende NIOC vindt plaats op donderdag 27 maart 2025 in Zwolle en wordt dan georganiseerd door Hogeschool Windesheim. Kijk op www.nioc2025.nl voor meer informatie.

Wil je op de hoogte blijven van de ontwikkeling rond Stichting NIOC en de NIOC kennisbank, schrijf je dan in op de nieuwsbrief via

www.nioc.nl/nioc-kennisbank/aanmelden nieuwsbrief

Reacties over de NIOC kennisbank en de inhoud daarvan kun je richten aan de beheerder:

R. Smedinga kennisbank@nioc.nl.

Vermeld bij reacties jouw naam en telefoonnummer voor nader contact.

Een application framework voor educatieve software

Jan Herman Verpoorten
Instituut voor Informatica en Informatiekunde, Universiteit Utrecht
janherm@cs.uu.nl

Samenvatting

Een application framework kan de kosten voor de ontwikkeling van applicaties verlagen, omdat grote tijdsbesparing kan worden bereikt door hergebruik van zowel het ontwerp als de code. Om dit m.b.t. educatieve software te bereiken, is het ClassMate application framework ontworpen. Dit is uitbreidbaar, leerstrategie-neutraal en baseert zich op open standaarden (XML, Java).

De eigenschappen van dit framework, met name de conceptuele modelering, worden besproken.

1. Introductie

Educatieve software, verder aangeduid als LTS (Learning Technology System [1]; hiervoor gebruiken we in het nederlands een hele reeks termen zoals electronische leeromgeving, E-learning, online leren, teleleren, computerondersteund onderwijs/toetsen, interactieve leeromgeving, ict-onderwijs, courseware e.a.) wordt gekenmerkt door *interactie* tussen de leerder en het programma. Hierin onderscheidt het zich van de meer statische (web) content. Zulke interactie is doorgaans complexer van aard dan de standaard in programma-bibliotheken aanwezige 'text fields', 'list boxes' en dergelijke. Ook de huidig bestaande digitale leeromgevingen bieden weinig meer dan zulke standaard interactiviteit, om de simpele reden dat vrijwel allen gebaseerd zijn op HTML (al dan niet met script of applets 'verrijkt'). Wie meer wil krijgt in de regel te maken met software-ontwikkeling. Het lijkt redelijk te veronderstellen dat hierbij dezelfde ontwikkelmethodieken kunnen worden ingezet als bij de meeste andere software het geval is. Dit is voor onderwijskundige toepassingen maar gedeeltelijk juist vanwege twee essentiële factoren, een externe en een interne.

De externe factor is de omvangrijke behoefte aan educatieve software, onvergelijkbaar met andere software. Als LTSs dagelijkse praktijk worden is de omvangsbehoefte groot: denk aan het aantal vakken, leerniveaus en de variëteit tussen instellingen door verschillen in didactische opvattingen. Ook al wordt van een lage LTS penetratiegraad uitgegaan, dan nog zijn er enorme resources (mensen, tijd, budgetten) nodig om dit allemaal te ontwikkelen.

De interne factor betreft de inzet van LTSs. Deze vindt regelmatig plaats om redenen van onderwijsvernieuwing. Onvermijdelijk bevindt zich daarom een hoeveelheid onzekerheid in de ontwerpspecificaties, zowel wat betreft de juistheid van vakdidactische keuzes als het gebruik van technologie. Als in de praktijk de ideeën anders uitpakken worden wijzigingen en aanvullende wensen geformuleerd. Het gevolg is de behoefte aan updates, vaak al na korte tijd. Juist in de beginperiode zijn deze updates belangrijk om de ingezette onderwijsvernieuwing niet te frustreren, maar tegelijk is dit een verdere aantasting van de middelen. Willen LTSs op enige schaal in het onderwijs kunnen incorporeren, dan moet het hoge-onkostenprobleem *in ieder geval* opgelost worden.

Een in de software-ontwikkeling al langer bekende oplossingrichting heet 'application famework'. Dit artikel bepreekt de opzet van zo'n voor LTSs ontwikkeld application framework. Paragraaf 2 bespreekt algemene kenmerken en gaat in op enkele achtergronden. De kern van dit artikel is paragraaf 3 waarin de belangrijkste aspecten van de modelering behandeld worden. Paragraaf 4 eindigt met enkele conclusies.

2. Kenmerken en achtergronden

Een application framework kun je beschrijven als de implementatie van een herbruikbaar ontwerp. Grote tijdsbesparing kan bereikt worden omdat zowel het ontwerp als de code opnieuw gebruikt kunnen worden. Helaas is het ontwikkelen van een goed framework moeilijk. Voor een effectief gebruik moet de structuur niet te complex zijn, en het moet in staat zijn de variëteit in het toepassingsgebied op te kunnen vangen. Het huidige beschreven application framework is gebaseerd op veel praktijkervaring: Het heeft een voorloper waarmee in verschillende onderwijs-sectoren programma's zijn ontwikkeld, zowel voor onderwijs-onderzoek [3, 4] als voor computer-based training en toetsing [4, 5]. Dit eerdere framework is gemigreerd naar het hier besproken framework, *ClassMate* genoemd. Dit verschilt in meerdere opzichten van zijn voorganger. Zo wordt bijvoorbeeld een componenten-architectuur nagestreefd, die het mogelijk moet maken

uitbreidingen een aanpassingen beter te kunnen uitvoeren [6].

Het ClassMate framework verschilt op enkele punten van andere application frameworks. Zo is er geen ontwikkelomgeving nodig om de doelapplicatie te ontwikkelen: het volstaat een specificatie in een XML grammatica te leveren. Hierdoor is het framework ook bruikbaar voor instructie-ontwerpers met enige technische kennis. Wel is een goede kennis van de framework-modeling vereist.

Een prototype-realisatie van ClassMate is ingezet in twee multimediale e-leaning projecten, om het ontwerp te valideren. Hierna is de feitelijke programmering gestart. Op dit moment is de kern van het framework operationeel. Algemene uitgangspunten voor ClasMate zijn gebruik van open standaarden, platform- en distributie-onafhankelijk, ontworpen voor verandering, leerstrategie-neutraal, specificatie-gestuurd en een strikte scheiding tussen framework en lesinhoud.

3. Conceptueel model van het framework

Bij de modelering is uitgegaan van alle aspecten van een operationeel leersysteem. Hierin staan drie dingen centraal: de structuur van de lesinhoud, het sessieverloop en programma- of leerderssturing op beiden. Deze sturing veroorzaakt runtime dynamiek, die we in de werkelijke wereld aanduiden met begrippen zoals 'adaptief leer- of toetssysteem' en 'docent- of student centrale instructie'. Alle operationele kenmerken worden vastgelegd in de *specificatie*, opgesteld door een instructie-ontwerper. Een leerproces wordt beschouwd als het door de leerder *implementeren* van deze specificatie, met als uiteindelijk resultaat een *enkele score*. In ClassMate heet dit een sessie.

In de volgende paragrafen licht ik de modelering van de lesinhoud en het sessieverloop toe en komt ook de door programma- of leerderssturing veroorzaakte runtime dynamiek ter sprake.

3.1. Modelering van lesinhoud

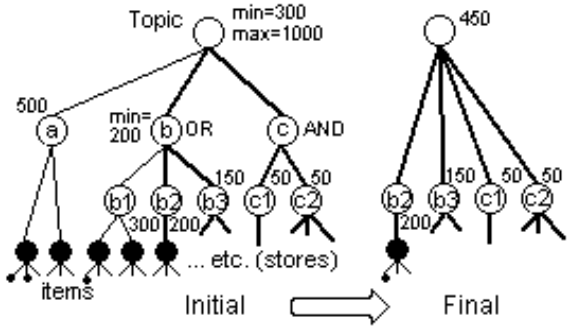
De structuur van de lesinhoud wordt vastgelegd in een boom, *content tree* genoemd. Deze boom bevat drie typen nodes: *topic, store* en *item*. Een topic is alleen een *label* voor een bepaalde lesinhoud. Een topic node heeft topic of store kinderen. Store is een abstractie van opslag. De kinderen van store zijn item nodes. Een item node is een representatie van de kleinste zelfstandige educatieve eenheid. Het bevat geen educatieve inhoud, maar de location en enkele metadata zoals het ID van een item object. De modelering van item objects behandel ik in 3.4.

De hoofdreden om de structuur van inhoud te beschrijven in een boom is de noodzakelijke *dominantie* van een node op zijn kinderen: hiermee kan een content profiel gerealiseerd worden. Een voorbeeld van dominantie is een store die reeks bij elkaar horende vragen (vervolgvragen) bevat. Keuze van de store of van één item selecteert alle items: de alles_of_niets-dominatie, die plaats vindt door de AND-waarde van het nodeType attribuut. Andere waarden zijn XOR en OR. De uiteindelijke dominantie wordt gespecificeerd door een samenspel van de topic en store attribuutwaarden weight, min/maxWeight, isSelected en nodeType. Deze resulteren uiteindelijk in een bepaalde selectie van topics, stores en items. Bij de aanvang van een sessie hoeft de content tree echter niet geheel vast te liggen: Hiervoor wordt in het framework onder meer een 'bootstrap' onderscheiden.

3.2 Bootstrap

Bootstrap heeft op initiële sessie-eigenschappen betrekking. Het is ook een van de aspecten van het standaard applicatiegedrag van het ClassMate application framework. In principe staan alle specificatie-aspecten onder programma- of leerderssturing. Als bijvoorbeeld de ontwerper de keuze van bepaalde topics aan de leerder overlaat, is de content tree nog niet af: voor de aanvang van de sessie moet de feitelijke content tree door de leerder bepaald worden. In dit geval levert de instructie-ontwerper een raamspecificatie die tijdens bootstrap afgemaakt wordt. De content tree beperkt leerderssturing tot topic selectie.

De attributen nodeType en min/maxWeight domineren de sturing die een leerder op topics heeft. Ik maak dit duidelijk aan de hand van een denkbeeldige basisspecificatie (figuur 1) waarin tijdens bootstrap door de leerder keuzes gemaakt zijn (aangegeven met dikke lijnen). De figuur geeft enkele van de dominerende randvoorwaarden weer. Bijvoorbeeld impliceert keuze van alleen topic c, c1 of c2 automatisch selectie van c=c1+c2 vanwege het AND nodeType van node c.



Figuur 1. Initial and final content tree

Na het keuzeproces verwijdert de boom overbodige topics en is de definitieve specificatie van de content tree totstand gekomen (figuur 1). Met deze specificatie wordt in de sessie verder gewerkt. Uiteraard bevat de bootstrap nog meer eigenschappen, van betrekking op

het sessieverloop. Deze blijven op deze plaats onbesproken.

3.3 Modelering van sessie verloop

De kleinste implementatie-eenheden zijn item objecten, kortweg items genoemd (zie verder 3.4). In de specificatie worden items gerepresenteerd door item nodes. In een sessie hoeven niet alle gespecificeerde items ook noodzakelijk gepresenteerd te worden. Er is sprake van een task list die afgewerkt wordt. Het koppelen van een task list entry aan een nieuw item gebeurt door een samenspel van z.g. node shufflers van content tree. Elke node van content tree heeft een gespecificeerde 'shuffler'. Alle shufflers tesamen bepalen het pad naar een item.

Selectie van een task list entry gebeurt door een selector, die in de specificatie aangegeven wordt. Ook deze kan een shuffle actie uitvoeren. Zeer eenvoudige voorbeelden zijn een menu waarin een item gekozen / herkozen wordt (leerderssturing), of een niet-visuele selector die items sequentieel presenteert en na implementatie voor herpresentatie blokkeert (programmasturing).

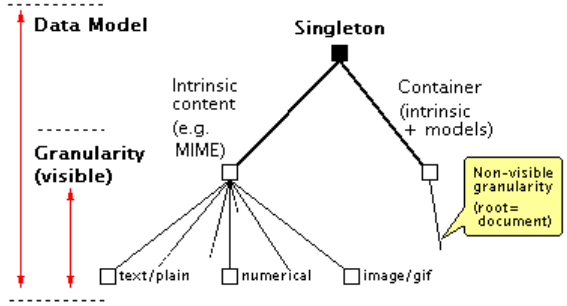
Een sessie kan bestaan uit meerdere opeenvolgende iteraties, elk met een verschillende run strategy. Run strategy is een bundeling van sessie-eigenschappen zoals het soort item manager, de item mode, UI-instellingen (panelcompositie, menubar, toolbar en navigatie), init en exit item, iteration control modes, property filters en meer. Zeer eenvoudige bekende run strategy voorbeelden zijn 'browsen', 'drill & practice' of 'zelftoetsing'.

De hierboven beschreven kenmerken zijn voornamelijk de meer statische aspecten van een sessie. Hiernaast zijn er ook dynamische aspecten: dingen die tijdens een sessie veranderen onder invloed van bepaalde gebeurtenissen, zoals de vorderingen van de leerder, of op initiatief van de leerder zelf. De sturing die een leerder op het sessieverloop kan uitvoeren is grotendeels zichtbaar in de user interface. Elke run strategy kan een aantal menu items bevatten; de specificatie van elk menu item bevat een action list. Programmasturing op het sessieverloop wordt gespecificeerd d.m.v. rules. Bijvoorbeeld om op een topic in te zoomen vanwege een matig voortgangsresultaat. De rules bestaan uit een boolean expressie en een actionlist. Een expressie wordt geëvalueerd als er veranderingen zijn in de state waarvoor de regel listener is. Deze sturing bevat zoveel details zodat verdere bespreking achterwege blijft.

3.4 Item

Item is reeds gedefinieerd als de kleinste zelfstandige educatieve eenheid. In de modelering behandel ik het als een transactie object. Dit object is een container voor *viewer*s, *interaction*s, *rule*s en [inner] *items* (recursieve definitie). Verder bevat het een *response* object dat het transactie-resultaat bevat. De visuele specificatie vindt plaats in een *view tree*.

Viewers zijn voor het zichtbaar maken van niet-interactieve data. In de modelering wordt onderscheid gemaakt tussen het data model, de in het model zichtbare granulariteit en het data type (zie figuur 2).



Figuur 2. Data model, granularity and type

Drie elementaire data modellen worden onderscheiden: *singleton*, *list* en *tree*. Een singleton is een enkele entiteit. ClassMate heeft een aantal viewers die allen geschikt zijn voor singletons met intrinsieke content. Custom types kunnen geregistreerd worden in het framework o.b.v. het MIME type (Multipurpose Internet Mail Extensions). Een GIF bitmap plaatje bijvoorbeeld heeft een 'singleton' data model en intrinsieke content van het GIF data type.

Complexiteit ontstaat wanneer modellen andere modellen bevatten (die andere …. etc.). Er is dan sprake van een container voor intrinsieke content en andere datamodellen. De wortel van zo'n hierarchie heet document. Ik acht het niet doenlijk om voor een dergelijke complexiteit een generieke modelering te vinden. In de specificatie wordt dit opgelost door declaratie van de betreffende viewer class (een implementatie van een abstracte class in het framework). Dit principe herhaalt zich bij het *list* en *tree* model, welke beiden singletons bevatten.

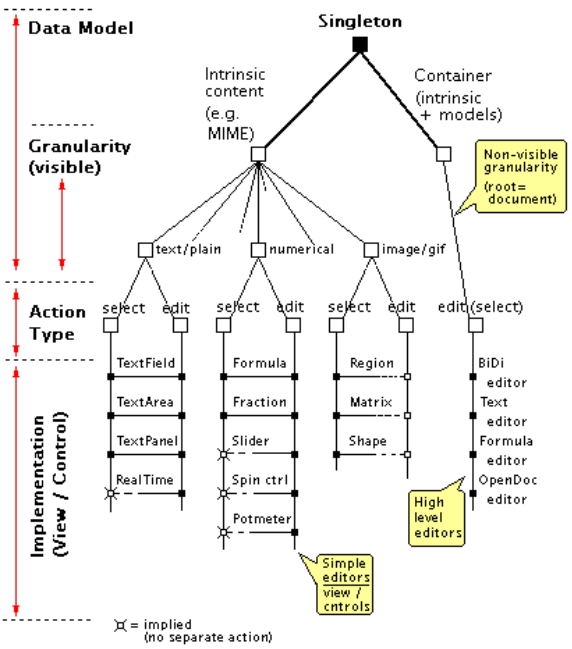
Interactions zijn een belangrijk deel van de framework modelering, wellicht het belangrijkste: interactie is een essentieel kenmerk van een leeromgeving. Een ontwerp-uitdaging ligt in het vinden van interacties die specifiek zijn voor educatieve software (onderwijsvernieuwing!), en niet alleen een schaduwganger zijn van interactie in de klaspraktijk of uitsluitend terugvallen op de interacties in het computer besturingssysteem. Een andere uitdaging ligt in het vinden van een modelering voor een conceptueel eindige interactie-verzameling. De uitgevoerde modelering representeert zo'n eindige verzameling. Het belang hiervan is helder. Alleen op deze manier kunnen interfaces, abstracte objecttypen en hulpobjecten ontworpen worden, noodzakelijk voor de consistentie en ontwikkelsnelheid in geval van toekomstige uitbreidingen.

In de uitgevoerde modelering wordt onderscheid gemaakt tussen *model*, *action* en *view/control*. Deze scheiding komt overeen met het bekende MVC (Model View Controller) paradigma, oorspronkelijk ontworpen voor SMALLTALK bij Xerox PARC. Anders dan in MVC wordt 'action' gebruikt voor een meer abstracte betekenis van 'controller' gedrag. Vanwege de hechte relatie tussen view en control zijn deze twee gebundeld in diverse concrete view/control implementaties van abstracte acties. De twee basis-acties zijn *selecteren* en *editen*. De precieze betekenis hiervan is afhankelijk van het data model waarop een actie werkzaam is en de view/control implementatie. Ik onderscheid (uiteraard analoog aan viewers) drie verschillende basis data models: *singleton*, *list* en *tree* waarop acties werkzaam kunnen zijn.

Bij een bepaald data model en een bepaalde actie hierop zijn dus meerdere actie-implementaties mogelijk. Eenvoudige bestaande voorbeelden zijn 'text field' en 'text area', beiden een edit-actie op een 'text/plain' data type (singleton model). Maar er kunnen meer implementaties zijn die qua view/control functionaliteit van deze twee voorbeelden verschillen. Bijvoorbeeld tekstinvoer waarbij 'fout' ingetypte karakters geweigerd worden of door juiste tekens worden vervangen.

Figuur 3. Selecteer-interactie voorbeeld

Figuur 3 geeft een voorbeeld van een selecteer-interactie: het selecteren van een hot spot door het plaatsen van een image (hier een wortel). Figuur 4 geeft een beeld van de interactie modelering bij het singleton data model. Uiteraard is de figuur niet volledig ingevuld vanwege de grote mate van detaillering in de praktijk. Wel is te zien hoe bij een enkele actie meerdere view/control implementaties mogelijk zijn. Daarbij is het ook mogelijk dat een enkele implementatie op meerdere data types tegelijk betrekking heeft. Bijvoorbeeld de 'image selector' in figuur 3, die op image/gif, image/png en image/jpg kan functioneren.



Figuur 4. Interacties bij singleton data model

Acties op een lijst of boom kunnen zowel betrekking hebben op het data model als op een element hierbinnen. In de modelering wordt dit onderscheid aangehouden. Een edit-actie bijvoorbeeld op het model betekent zoveel als elementen invoegen, verwijderen of ordenen. Een selecteer-actie op een element in het model kan tot doel hebben een singleton actie te starten, dus om *in* het element een selecteer- of edit-actie uit te voeren. Op deze wijze zijn ook complexere interacties mogelijk.

Modelmatig onderscheid ik maar twee basis acties, *selecteer* en *edit*. Een selecteer-actie is een enkelvoudige handeling, maar een edit-actie is vaak een combinatie van verschillende handelingen. Zoals eerst het selecteren van een positie, gevolgd door een handeling die iets verandert aan de situatie op die positie: invoegen, wissen, vervangen of de bestaande ordening veranderen (door verplaatsen). Bijna elke edit-actie impliceert een initiële selectie vanwege de interne granulariteit. Sommige implementaties hebben geen interne granulariteit, bijvoorbeeld een slider of spin control. In deze gevallen kun je de waarde wel editen, maar er is maar één initiële positie (dus geen selecteer-handeling). Editen is dus complexer dan selecteren alleen, maar ook minder nauwkeurig bepaald. Het wordt gebruikt in de betekenis van 'wijzigen', anders dan selecteren alleen. Op deze wijze kan volstaan worden met twee basistypes (zie figuur 4) die tevens aansluiten op het huidige spraakgebruik (mentale

modellen). De concrete uitwerking van een actie ligt vast in de view/control implementatie.

Een item bevat ook z.g. rules, waarin de responses van de student bevraagd kunnen worden. Ze zijn bedoeld voor het berekenen van waarden en het conditioneel uitvoeren van programma-acties. Eén zo'n actie is het lanceren van een z.g. inner item. Dit is het eenvoudigst te vergelijken met de zogenaamde 'feedback' in de oudere computerondersteunde training: een presentatie die betrekking heeft op het zojuist geleverde antwoord. Een inner item is een volwaardig item, dat al dan niet exclusief door item wordt beheerd. Het toepassingsgebied van inner items is heel breed: van feedback of hulpvragen tot en met 'adventure games'. Ieder item kan (recursief) meerdere inner items bezitten of refereren.

Het laatste aspect dat van item beschreven wordt is de visuele presentatie die in een view tree vast ligt. Alle viewers en interacties in een item worden uiteindelijk op een beeldscherm gevisualiseerd. Dit scherm wordt beschouwd als de wortel van een visuele boom, waarin de diverse knopen (bundelingen van) viewers en interacties bevatten. De boomstructuur is onderdeel van elke item specificatie. Hiermee zijn individueel verschillende presentaties mogelijk, maar uiteraard kan ook worden verwezen naar een enkele 'stylesheet' die voor meerdere items geldig is. De feitelijke visualisering wordt door een (te specificeren) 'layout manager' uitgevoerd. Zoals veel eigenschappen van het framework is ook de visuele boom dynamisch. De structuur kan at runtime gewijzigd worden. Een voorbeeld van een toepassing hiervoor is een 'formuliervraag' waarin nog maar één interactie moet worden beantwoord, die dan als enige nog in beeld verschijnt.

4. Conclusie

ClassMate is geen geïntegreerde omgeving die tevens studentengroepen, profielen en voortgangsgegevens beheerd. Het kan wel als zodanig in een 'digitale leeromgeving' worden ingezet (als applet op een webpagina of als applicatie). Men zal echter zelf de koppelingen naar de beheersdelen moeten uitvoeren.

In ClassMate ligt het accent op het primaire proces: onderwijs. De uitgevoerde modelering is van toepassing op een grote variëteit aan instructie. Een specificatie kan in korte tijd uitgevoerd worden. Op basis hiervan biedt het application framework direct beschikbare functionaiteit. M.b.t. het creëren van een specificatie wordt gerekend op instructie-ontwerpers, omdat dit enige technische kennis en een goede kennis van de modelering vereisen. Door de inherente complexiteit hiervan ligt grootschalige inzet door 'echte' eindgebruikers (docenten) nog niet voor de hand, maar wel dichtbij.

Uitbreidingen, zoals eigen vraagtypen moeten worden uitgevoerd door programmeurs, maar kunnen relatief eenvoudig tot stand komen.

Referenties

[1] IEEE Learning Technology Standards Committee, *Draft Standard for Learning Technology – Learning Technology Systems Architetcture*, IEEE, http://ieee.ltsc.org/wg1, 2000.

[2] Verpoorten, J.H., *A Model Based Approach to Courseware Development* [thesis], ECCO Expertise Centrum voor Computerondersteund Onderwijs, Utrecht, 1995.

[3] Graaff, R. de, *Differential Effects of Explicit Instruction on Second Language Acquisition* [thesis], HIL Dissertations, Leiden, 1997.

[4] Slagter, P.J., *Learning by Instruction* [thesis], Rodopi, Amsterdam, 2000.

[5] J.H. Verpoorten, "Informatietechnologie voor studenten letteren". In: Mirande, M.J.A., *De Kwaliteiten van Computerondersteund Onderwijs*, Coutinho, Bussum, 1994, pp. 318-331.

[6] Szyperski, C., *Component Software - Beyond Object-Oriented Programming*, Addison-Wesley, New York, 1999.