ИНСТИТУТ ПО ИНФОРМАЦИОННИ И КОМУНИКАЦИОННИ ТЕХНОЛОГИИ – БЪЛГАРСКА АКАДЕМИЯ НА НАУКИТЕ

Контекстно-ориентирано управление на електронни услуги

Автореферат

на дисертационен труд за присъждане на образователна и научна степен "доктор"
в област *4. Природни науки, математика и информатика*,
професионално направление *4.6 Информатика и компютърни науки*,
докторска програма *Информатика*

Докторант: Владимир Николаев Вълканов

Научни ръководители: акад. проф. дтн. Иван Попчев

София

2013

Дисертацията е обсъдена и допусната до защита на разширено заседание на секция „Интелигентни системи” на ИИКТ-БАН, състояло се на 18.10.2013 г.

Дисертацията съдържа 140 стр., 33 фигури и 15 стр. литература, включваща 172 заглавия.

Защитата на дисертацията ще се състои на 17.12.2023 г. от _____ ч. в зала ___ на блок ___ на ИИКТ – БАН на открито заседание на научно жури в състав:

1. акад. д.т.н. Иван Попчев
2. проф. д.т.н. Тодор Стоилов – ИИКТ – БАН (рецензент)
3. проф. д-р Аврам Ескенази – ИМИ – БАН
4. проф. д-р Боян Бончев – СУ “Св. Кл. Охридски” (рецензент)
5. проф. д-р Асен Рахнев – ПУ „Паисий Хилендарски”

Материалите за защитата са на разположение на интересуващите се в стая 215 на ИИКТ – БАН, ул. „Акад. Г. Бончев”, бл. 25А.

Автор: Владимир Николаев Вълканов

Заглавие: Контекстно-ориентирано управление на електронни услуги

Съдържание

Индекс на фигурите

Увод

Средите за доставка на електронни образователни услуги стават все по-налагаща се интегрална част на съвременното обучение. В отговор на потребността за подпомагане на обучението посредством използване на съвременни информационни и комуникационни технологии, във ФМИ на Пловдивския университет се разработва инфраструктура, наречена Разпределен център за електронно обучение, познат като DeLC [20]. Предназначението на центъра е да доставя по един контекстно-зависим, адаптивен и персонализиран начин електронни образователни услуги и електронно учебно съдържание, разположени върху физически разделени сървъри [23]. Концептуално DeLC е динамична мрежова структура, състояща се от образователни възли и релации, специфициращи определени зависимости между тях [24,25,26,9]. Възлите могат да оперират самостоятелно или динамично да се свързват помежду си, образувайки комплексни виртуални структури, наречени **образователни клъстери**. В актуалната версия на DeLC са изградени два образователни клъстера. Първият клъстер, наречен MyDeLC, се използва за организация и провеждане на електронно обучение във ФМИ, като доставя фиксиран достъп до услугите и електронното съдържание посредством специализиран образователен портал [29]. Вторият клъстер, наречен InfoStation клъстер [22], предоставя трислойна архитектура, осигуряваща мобилен достъп до услуги и информационни ресурси посредством интелигентни безжични точки на достъп (наричани Information Stations - ISs), разположени около сградата на университета [19,21,22,10]. Цялостната концепция за DeLC, като контекстно-зависима и адаптивна инфраструктура за електронно обучение е представена в [18].

В съответствие с концепцията за развитие на центъра, следващ етап е изграждане на Виртуално Образователно Пространство (ВОП). В пространството, контекстно-зависимостта на образователните услуги ще се поддържа от автономни интелигентни компоненти с реактивно, интерактивно и проактивно поведение. Съществуващите образователни клъстера поетапно ще се трансформират в такива компоненти. Архитектурата на пространството ще бъде напълно децентрализирана. Трансформацията на DeLC във ВОП се предприема поради въздействието, което ще имат в недалечното бъдеще две глобални тенденции - *мрежа на нещата* (Internet of Things) [2,7] и *семантичен уеб* [3,1] - върху начина на опериране на Интернет-базирани приложения (каквото е DeLC).

Основната цел на дисертацията е да се ***разработи контекснтно-ориентирано управление на електронните услуги, доставяни във ВОП***. Съществено изискване е реализиращите такова управление софтуерни компоненти да могат лесно да се интегрират в архитектурата на пространството ВОП и същевременно да запазват нейната хомогенност.

За постигане на целта, в дисертацията се следва и адаптира **методологията** за изграждане на контекстно-зависима архитектура, представена в [18]. Основна роля играе понятието „*контекст*", което съгласно Дей [5,6] е всяка информация, използваема за характеризиране на ситуацията в една идентичност. Идентичност може да бъде човек, местоположение или обект, които са съществени (не могат да бъдат пренебрегвани) при взаимодействието между потребител и приложение (включително самият потребител и самото приложение). Контекстът може да бъде от различен тип, освен горепосочените, също и действие, време и т.н. В този смисъл, една система е контекстно-зависима, ако идентифицира и използва контексти при доставка на желана информация и услуги на потребителя, като степента на използваемост зависи от действията на потребителите. В [18], *адаптивността* и *персонализацията* са дефинирани като атрибути на *контекстно-зависимостта*. Целта на адаптивността е да се осигури безпроблемно, прозрачно и адекватно изпълнение на потребителските заявки за услуги и електронно съдържание, като се вземат предвид различните аспекти на контекста. Целта на персонализацията е, софтуерът да е в състояние динамично да се адаптира в зависимост от индивидуалните особености на конкретния потребител. Потребителите трябва да имат усещането, че той (софтуерът) е разработен специално за изпълнение на техните заявки. Така, когато едно приложение идентифицира определени промени в околната среда, то трябва да е в състояние да извърши специфични, зависещи от вида на промяната, компенсаторни действия, такива като напр., персонализация на услуги и съдържание, преформатиране на съдържание, превключване към подходящи хранилища на съдържание.

Съществен аспект на контексно-зависимото ВОП е осигуряване на контекстно-ориентирано управление на предлаганите електронни услуги. В разпределената архитектура на пространството промените могат да бъдат идентифицирани само локално, т.е. в обхвата (околната среда) на позиционирания там автономен интелигентен агент (който има само частичен контрол върху околната среда, т.е. само в своя обхват). Така, състоянието на пространството е **подредено** множество от локални

състояния. Решаваща за управлението в такава ситуация е възможността за синхронизация между отделните агенти. За да могат да се синхронизират, те се нуждаят от **изчислителен механизъм** за идентифициране и подредба на промените във ВОП. Тук не ни интересува реалното физическо време, а по-скоро моментите на промяна на локалните състояния на пространството. Добро решение е, ако този механизъм е **формален**.

От направения предварителен анализ за търсене на възможности за реализиране на поставената цел беше избрана системата Tempura (подробно представена във втората глава на дисертацията). Въпросът е как тази система да се направи използваема във ВОП. Съществено изискване е запазване хомогенността на архитектурата [18]. ВОП се изгражда от отделни интелигентни автономни компоненти, където електронните услуги са снабдени със съответни обслужващи агенти. Архитектурата на пространството е реализирана върху виртуалната машина на Java. Решението, предложено в дисертацията, е съществуващата архитектура да се разшири с *Tempura-базиран изчислителен механизъм*, който да идентифицира локално случващите се във времето промени в пространството и да генерира съответна **наредба**. Реализацията на този механизъм трябва да удовлетворява следните изисквания:

- Да запазва съществуващата функционалност на ВОП;
- Да запазва съществуващата хомогенност на ВОП;
- Да се вгражда безпрепятствено в съществуващата архитектура на ВОП.

Изводът е, че се нуждаем се от агентно-ориентирана версия на Tempura, имплементирана на езика за програмиране Java.

В съответствие с методологията, за постигане целта в дисертацията е предложен реинженерингов подход за реализиране на агетно-ориентирана версия на Tempura и са дефинирани следните основни задачи:

1) Разработване на подход за реинженеринг на оригиналния интерпретатор на Tempura при спазване на дадените по-горе изисквания;
2) Разработване на модели, поддържащи предложения реинженерингов подход;

3) Изграждане на целевата агентно-ориентирана архитектура;

4) Създаване на нова програмна версия на оригиналния интерпретатор на Tempura (реализиран на C/C++) в съответствие с подхода, предложен в дисертацията.

Дисертацията е структурирана както следва:

- Увод, в който са дефинирани целите и задачите на дисертацията.
- В първа глава е направен преглед на състоянието на научно-изследователски области, имащи отношение към изследването.
- Във втора глава се прави преглед на основните концепции на ITL. Представени са интерпретатора Tempura и неговата околна среда AnaTempura. Представен е също подход, чрез който съществуващите интерпретатор и околна среда (Tempura и AnaTempura), могат да бъдат преобразувани в система с подходяща архитектура, удовлетворяща нашите цели и желания.
- Трета глава представя съсздаването на обектно-ориентирана версия на Tempura – jTempura.
- Глава четвърта е посветена на следващата стъпка от разработката, а именно постигането на агетно-ориентирана версия на jTempura. Тук са представени модела за изграждане контекстно-зависими архитектури, наречен C3A и модела за трансформация на jTempura, наречен трансформационен.
- Заключението обобщава резултатите от дисертацията и дава някои перспективни насоки за продължаване на изследванията по темата на дисертационния труд.

Глава 1 Състояние на изследвания проблем

В тази глава е направен кратък преглед на състоянието на научно-изследователските области, свързани с проведеното в дисертацията изследване:

- Реинженеринг и рефакторинг на наследен софтуер.
- Интелигентни агенти и мулти-агентни системи.
- Виртуални интелигентни пространства.
- Контекстно-зависими системи.
- Формализми за контекстно-зависими системи.

Глава 2 Теоретични основи

Интервална темпорална логика (Interval Temporal Logic)

Interval Temporal Logic (ITL) е гъвкава нотация, както за съждителната логика, така и за логиката от първи ред. С нейна помощ се дефинират и моделират времево-зависими процеси; често се използва в описанията на хардуерни и софтуерни системи.

За разлика от повечето темпорални логики, ITL е подходяща за представяне и моделиране както на последователни, така също и на паралелни процеси и композиции. Тя предлага мощна и разширима спецификация и техники за верификация чрез аргументиране на свойства, като напр. сигурност, жизненост, планиране на времето. Темпоралните ограничения са лесно изразими. Голям брой императивни програмни конструкции могат да бъдат представени като формули в слабо променена версия на ITL. Едно подмножество на ITL, наречено Tempura, предоставя езикови конструкции, удобни за спецификация поведението на софтуерни системи, които могат да бъдат автоматично интерпретирани. В допълнение, ITL и Tempura се използват широко за специфициране свойствата на системи, работещи в реално време, където жизнения цикъл може да бъде представен директно чрез множество от прости темпорални

формули. Различни изследователски проекти използват Tempura за симулации, където времето е критичен фактор.

Темпоралната логика е полезен инструмент за аргументиране на процесите в конкурентни програми и хардуер [16]. В рамките на темпоралната логика могат да се изразят логически оператори обвързани с времезависещи концепции като „*винаги*" и „*понякога*".

Темпоралната логика е причислявана като инструмент за специфициране и доказване свойства на програми, използващи паралени процеси [14,15]. Различията между темпоралните логики и езиците за програмиране създават проблеми, когато трябва по формален начин да се изрази стойността на променлива, която се променя динамично по време на изпълнение на дадена програма. Програмни формализми, като напр. логиката на Hoare [13], динамична логика [11,17] или процесна логика [4,12] също отразяват това противоречие.

Tempura и AnaTempura

Съществува дебат до колко са приложими формалните модели при конструирането на съвременните компютърни системи, било то хардуерни или софтуерни. Някои твърдят, че използването на формални модели решава изцяло най-често срещаните проблеми при проектирането на тези системи, докато други са на мнението, че моделите на практика са слабо или трудно приложими при реални условия. Би било твърде смело да твърдим, че подходите за проектиране, използващи формални модели са универсално решение за софтуерните системи. От една страна, съществуват доста аспекти, които трудно биха били описани формално. От друга страна, използването на подходящи формални модели дават голямо предимство при проектиране и доказване коректността на софтуерните системи.

Един от основните проблеми при избора и прилагането на формализъм при моделирането на съвременните софтуерни системи е възможността за реализация на подходящ интерпретиращ формалната спецификация механизъм. Съществуването на такъв механизъм би направил възможно директно прилагане на формалните модели като част от системата, а не използването им само като теоретична постановка.

Interval Temporal Logic (описана в 2.1) е подходящ формализъм за описание на времево-зависими процеси, за който съществува интерпретиращ механизъм – софтуерната системата, наречена Tempura. Интерпретаторът Tempura представлява едно изпълнимо подмножество на ITL, което използва синтаксиса на ITL и се базира на неговата основна философия да възприема времето като крайна последователност от състояния, като на всяко състояние се съпоставя значещи (и налични) променливи, представящи интересуващите ни атрибути (свойства). На всяка променлива могат да бъдат присвоявани целочислени стойности. Подобна конструкция силно напомня поведението на софтуерна система по време на нейната работа и изменението на стойностите на съставящите я променливи.

Първата версия на интерпретатора Tempura е написан на езика Prolog от Бен Московски в началото на декември 1983 година. Съвременната версия на интерпретатора е написана на програмния език C през 1985 година от Роджър Хейл в университета в Кембридж, като негов докторантски тезис. През следващите години интерпретаторът е многократно дописван и разширяван с различни оператори и функционалности, като текущата версия Tempura 2.16 се поддържа от Антонио Кау, университет Де Монфорт, Лестър.

Основният принцип на работа на интерпретатора Tempura е обработването на текстови низове, които по своята същност представляват ITL твърдения или формули. Този принцип на работа, макар и коректно работещ, на база математическия модел на ITL, е труден за директно прилагане в реални системи. Поради тази причина е разработен съпътстващ модул, наречена AnaTempura. Той играе ролята на околна среда на интерпретатора, като събира и доставя информация, която да бъде интерпретирана.

Принципния механизъм на взаймодействие между Tempura и AnaTempura може да бъде описан по следния начин:

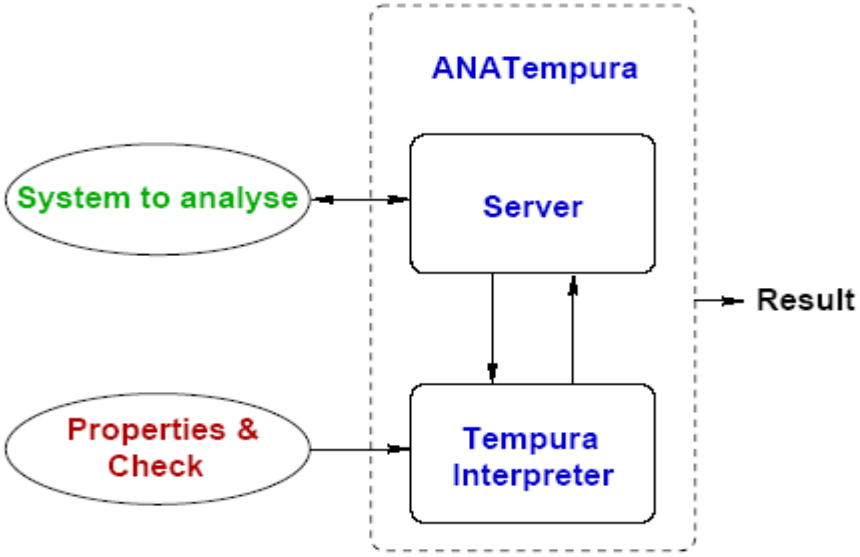
- Въз основа на анализа, който искаме да направим на дадена софтуерна или хардуерна система, се дефинира набор от променливи. Изменението на стойностите на тези променливи се следи по време на изпълнението на програмата с помощта на предварително дефинирани ITL изрази;

- AnaTempura служи като интерфейс между Tempura и системата, която се анализира. Нейната задача е да събира стойностите на следените

променливи и да реализира присвоявания на променливите в предварително дефинираните ITL изрази, които впоследствие ще бъдат обработени от Tempura;

- AnaTempura подава като текстови низ съставените изрази, които Tempura интерпретира на база теоретичния модел на ITL и връща получения резултат;
- AnaTempura анализира интерпретирания резултат и прави съответните изводи.

Архитектурата на този механизъм за анализ и верификация на различни системи може да бъде онагледена с принципната схема:



Фиг. 1: Принципна схема на взаимодействието Tempura - AnaTempura

Подход за разработване на архитектурата

Голямата сложност и комплексност на проблема изискват ясно дефиниран систематичен подход за решаването му. За създаване на механизъм за управление на времево-зависими процеси и вграждането му във вече изграждащата се архитектура на ВОП, ние предлагаме подход, включващ следните основни стъпки:

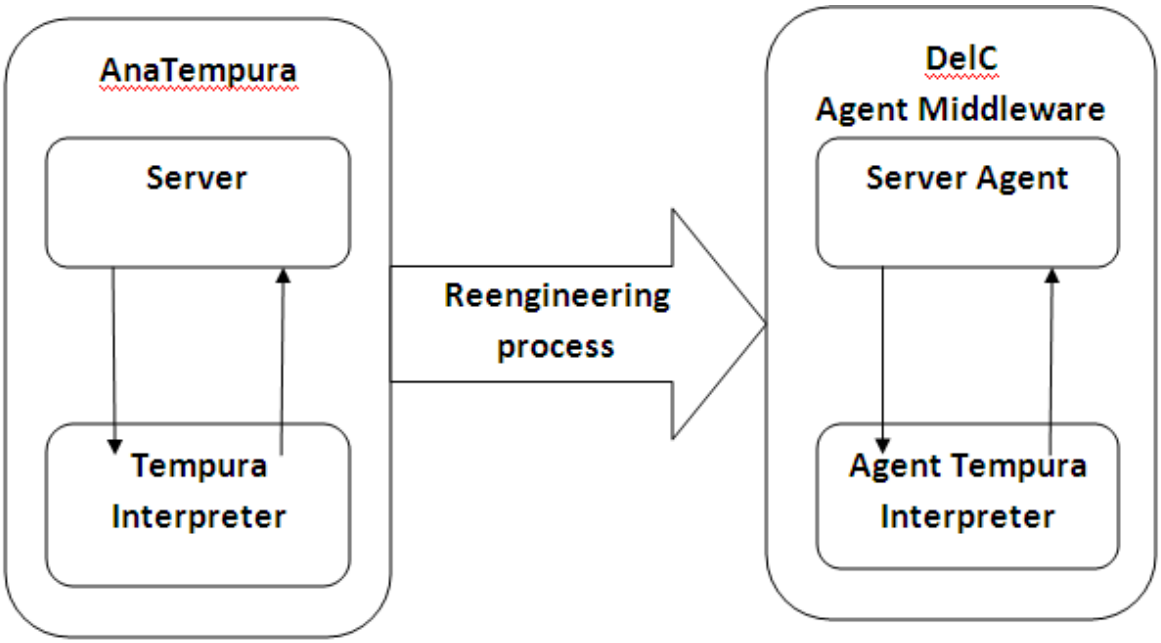
- Избор на формализъм за описание на времево-зависими процеси;
- Избор на интерпретиращ механизъм;

- Реинженеринг на съществуваща система;
- Интеграция в архитектурата на ВОП.

Някои от съществуващите механизми за дефиниране и описание на времево-зависими процеси бяха разгледани в Глава 2. Избраният от нас формализъм е специализирана темпорална логика, наречена Interval Temporal Logic (ITL). Този формализъм разглежда времето като крайна последователност от времеви интервали, което много се доближава до начина на обработка на данни в софтуерните системи и би довело до по-лесната му програмна реализация.

За постигането на нашите цели ние ще преминем през няколко последователни стъпки. Ще бъде извършен реинжинеринг на интерпретатора Tempura за преминаване към езика Java, с цел да се запази хомогенността на системата. Освен това разработването на новата версия ще премине през няколко ключови етапа, които да направят процеса по лесно проследим и коректен:

- Първо ще бъде направен директен превод от програмния език C на Java, за да се запази оригиналния синтаксис на една вече изпитана система, което ще спомогне и за тестването на новата версия.
- Следва промяна на архитектурата на Java интерпретатора, за да се получи обектно ориентирана версия. Това би направило системата по-лесна за използване и в други бъдещи проекти.
- Последната стъпка е създаване на агентно-ориентирана версия на Tempura, която да бъде вградена в съществуващата архитектура на ВОП. По този начин околната среда AnaTempura, която работи с оригиналния C интерпретатор, ще бъде заменена с интелигентни агенти опериращи във ВОП.



Фиг. 2: Трансформация на AnaTempura в арх. на ВОП

Глава 3 Обектно-ориентирана jTempura

Въведение

За създаването на обектно-ориентираната версия jTempura е избран реинженерингов подход върху C/C++ версията на интерпретатора и пренаписването му на езика за програмиране Java като най-подходящ. Нашето решение може да бъде обосновано със следните причини:

- Всички софтуерни модули на ВОП са реализирани на езика за програмиране Java и от тук, най-подходящо е интерпретаторът да бъде написан на същия език – по този начин ще бъде осигурена необходимата хомогенност, което от своя старна дава допълнителни възможности за бъдещо разширяване и развитие на системата при добра ефективност на run-time;

- Също така една Java версия би дала възможност за по-лесно създаване на агентно-ориентирана версия на интерпретатора, която да бъде лесно вградена в мулти-агентната архитектура на ВОП;

- От гледна точка на развитието на самия интерпретатор, версия, реализирана на езика за програмиране Java, може да се използва в бъдещи проекти, използващи съвременни обектно и агентно-ориентирани архитектури;

- При този подход изцяло се използва проверената и доказала своята ефективност в много приложения функционалност на Tempura. Оригиналната версия е използвана в реални системи и разполага с много тестови примери, които ще бъдат добра основа за първоначални тестове на новата версия ***jTempura***.

Обобщавайки, целта ни на този етап е пренасяне на пълната функционалност на Tempura в нова обектно-ориентирана архитектура, реализирана на езика за програмиране Java, която да може:

- Да съществува самостоятелно и да се интегрира в обектно-ориентирани приложения, реализирани на Java;

- В следваща стъпка, да се трансформира в агентно-ориентирана версия, която ще бъде интегрирана в изграждащата се мулти-агентна архитектура на ВОП.

Взаимствайки от добрите практики за разработване на обекно-ориентиран софуер, реинжинерингът на интерпретатора Tempura бе извършен чрез итеративен подход, по подобие на стандартните модели, като бяха реализирани две ясно различими итерации. Итерациите бяха предхождани от стъпка, при която беше възпроизведена (от първичния код) и описана архитектурата на оригиналната Tempura.

Като резултат от разработващата фаза получихме аналогична на оригиналната C-Tempura версия, написана на програмния език Java. Тестовете в третата фаза на процеса бяха използвани за проверка коректността на Java-версията на интерпретатора и запазване функционалността на оригинала. При реинжинеринговия процес искахме да запазим оригиналния синтаксис на интерпретатора, така че тестовете се проведоха, чрез изпълняване на скриптове от оригиналната C-Tempura. Паралелното изпълнение на тестови примери и сравняването на резултатите ни даде възможност да подтвърдим

коректността на новата версия и съответсвието й като формат на входно-изходните данни с оригиналната Tempura.

Въпреки директения превод на кода бяха направени някои промени по структурата, като в Java версията се опитахме да обособим логически близки групи от класове. Това ще ни даде възможност в следващите етапи на разработка да постигнем така важната и характерна за Java пакетна структура на интерпретатора. Основните предимства от това са следните:

- Постигане на по-добре структуриран код, което ще даде възможност за бъдещо развитие на интерпретатора, лесното му използване в други проекти и от други работни групи;
- Създаването на пакети може да позволи части от кода да бъдат дефинирани като библиотеки, които да се вградят във външни системи, които се нуждаят от контрол на времево-зависими процеси – начин, по който се използва оригиналния интерпретатор в практиката;
- От гледна точка на създаването на агентно-ориентирана версия, пакетите са много по-лесни за трансформация. Теорията казва, че агентите трябва да имат проста функционалност и не бива да изпълняват много и разнородни задачи. От тази гледна точка е логично да се търси архитектура, която преобразува всеки логически обособен пакет в агент.

Глава 4 Агентно-ориентирана AjTempura

При изграждане на агентно-ориентираната версия на Tempura (AjTempura), която ще бъде интегрирана в архитектурата на ВОП, трябва да бъдат спазвани две съществени изисквания:

- Запазване хомогенността на пространството;
- Осигуряване на лесна интеграция на нови компоненти в съществуващите архитектури.

С реализиране на jTempura първото условие е изпълнено, т.е. новата версия на Tempura може да работи върху Интегрираната Технологична Платформа (ИТП) [30] на пространството, използваща виртуална машина на Java. Това свойство ще бъде автоматично пренесено в агентно-ориентираната версия, понеже при реинженеринга на jTempura не се предвижда промяна на езика за имплементация.

По дефиниция ВОП се състои от автономни контекстно-зависими компоненти, което предполага агентно-ориентирана архитектура на пространството. Така jTempura, със своята обектно-ориентирана структура, не удовлетворява второто изискване. По тази причина е необходима нова реинженерингова стъпка, където обектно-ориентираната структура на интерпретатора ще бъде трансформирана в агентно-ориентирана, която ще наричаме AjTempura, при запазване пълната функционалност на jTempura.

При търсене на решение за успешно провеждане на трансформацията възникват различни въпроси, като напр. следните:

- Как може да бъде пренесена функционалността на класовете във функционалност на кореспондиращи агенти, отчитайки съществуващите релации между класовете в изходната (обектно-ориентираната) архитектура?
- Как ще бъде трансформирана вътрешната структура на класовете във вътрешна архитектура на кореспондиращите агенти?
- Как ще се изградят елементи в агентно-ориентираната архитектура, които нямат аналог в обектно-ориентираните архитектури, като напр. ментални състояния, околна среда?

За целите на дисертацията от съществено значение е отговорът на първия въпрос. Решения на останалите два проблема могат да се търсят в рамките на конкретна работеща агентно-ориентирана архитектура. Така напр., околната среда на AjTempura е ВОП, като за връзка с тази околна среда ще бъдат разработени специализирани интерфейсни агенти, както е обяснено по-долу в главата.

Модел на контекстно-зависима архитектура на ВОП

По дефиницията ВОП се състои от автономни контекстно-зависими компоненти, които могат да оперират в променяща се околна среда. Тези компоненти ще бъдат реализирани като интелигентни агенти с „ограничена рационалност". Предполагайки използването на такива агенти приемаме следните допускания:

- Функционалността на пространството в даден момент е променливо множество от информационни ресурси (агенти, услуги, структури данни, и т.н.);
- Без ограничаване на общовалидността върху пространство, приемаме, че функционалността се доставя от агенти;
- При настъпване на промяна в околната среда определени компонентите от пространството трябва да извършат определени компенсаторни действия, отразяващи тази промяна;
- Съществува минимална функционалност, без която пространството не може да оперира.

За реализация на пространството предлагаме абстрактен модел за контекстно-зависима агентна архитектура, наречен C3A (Context-Aware Agent Architecture). C3A оперира по следния начин:

- С *Agents* означаваме множеството на всички потенциално опериращи в простарнството агенти;
- $Agents = PA \cup OA$, така че $PA \cap OA = \varnothing$ – в множеството на всички агенти различаваме две дизюнктивни групи агенти;
- *PA* – наречени *персистентни агенти*, представят минималната функционалност на пространството;
- *OA* – наречени оперативни (компенсиращи) агенти, определена специфична условна функционалност;

- С *Activities* означаваме всички възможни действия на агентите;

- *Activities* = *Fuct* $\cup$ *Gen*, така че *PA* $\cap$ *OA* = $\varnothing$ – различаваме две дизюнктивни групи действия. В първата група се включват дейности, които реализират функционалността на пространството. Във втората група са включени така наречените генератори, с помощта на които динамично могат да се генерират и премахват оперативни агенти;

- $\forall$ *oa* $\in$ *OA* ($\exists$ $Act_{oa} \subseteq 2^{Funct}$) - всеки оперативен агент реализира определена специфична условна функционалност, която е необходима като реакция на идентифицираната промяна в околната среда. Този вид агенти се генерират динамично, при необходимост, и обикновено са с кратък жизнен цикъл;

- С *Events* означаваме всички възможни събития в пространството;

- *Events* = $Ev(pa_1) \cup Ev(pa_2) \cup ... \cup Ev(pa_k)$, където $Ev(pa_i)$ е множеството на локални събития, случващи се в околната среда (обсега) на персистентния агент pa_i , $i = 1, ..., k$ (k е броят на персистентните агент);

- $\forall$ *pa* $\in$ *PA* ($\exists$ $Act_{pa} \subseteq 2^{Funct} \times \{ Gen(e_i) \mid e_i \in Events \}$ - персистентните агенти имат две различни предназначения. Първо, те реализират сравнително стабилна малка функционалност (принадлежаща към минималната), която трябва да се изпълнява безусловно. Второ, използвайки локалния си контрол върху околната среда (обикновено агентите нямат пълен контрол върху околната среда, а само ограничен) те могат да установяват определен вид промени в околната среда. В зависимост от вида промяна те генерират динамично определен вид оперативен агент, който от своя страна извършва необходимото компенсаторно действие. Впоследствие оперативният агент се премахва. Персистентните агенти винаги са налични.

```
repeat
        running(pa1), running(pa2), …, running(pan), pan ∈ PA, i = 1, …, n;
        anytime ∀ pai {
                ∃ ej , ej ∈ Ev(pai) ⇒
                        { GENERATE (pai , oai );
                          REQUEST (pai , oai );
                          if INFORM (oai , pai , 'done' )
                          then REMOVE( pai , oai );
                        }
         }
        endanytime
forever
```

Фиг. 3: Жизнен цикъл на СЗА

На Фиг. 3 е даден (като псевдокод) принципен жизнен цикъл на СЗА. Представена във времето една контекстно-зависима агентно-ориентирана архитектура изглежда като пулсиращо ядро (реализиращо минималната функционалност на пространството), което периодично се „разширява" в различни посоки (в зависимост от промяната в околната среда) и отново се „свива" до обичайните си размери (минималната функционалност).

Трансформационен модел на AjTempura

В този раздел представяме модел, наречен *трансформационен*, който служи като технологична рамка за програмната реализация на AjTempura [28]. Съществото за *трансформационния модел* е разделянето на агентите на два типа – *абстрактни* и *реални*. Едно от предимствата на това разделяне е, че впоследствие моделът (абстрактните агенти) може да бъде реализиран в различни развойни среди (в дисертацията е реализиран в JADE). За спецификацията на абстрактните агенти използваме пакетния механизъм на Java. За да подчертаем разликата между

абстрактните агенти в трансформационния модел и реалните агенти в реализацията, използваме различни имена. В реализацията е дадена кореспондираща таблица.

При първата стъпка на реинженеринга, обектно-ориентираният код на jTempura може да се групира в няколко основни пакета според функционалните зависимости на класовете. Създадени са четири отделни пакета (IOPackage, LexerPackage, ITLElements, ITLAlgorithms) и самостоятелно е оставен един базов клас (TInterpreter) (**Error! Reference source not found.**). При интегриране на AjTempura в разпределената агентно-ориентирана архитектура на ВОП, този клас ще бъде заменен с подходящи интерфейсни агенти.

Реализация на AjTempura

Програмната реализация на AjTempura е извършена на основата на *трансформационния модел* и *C3A модела*. AjTempura се състои от следните два основни модула:

- Клиентски модул - означен като bg.uniplovdiv.fmi.delc.aot.client, реализиращ LifeCycleAgent абстрактния агент и базовия клас TInterpreter;
- Сървърен модул - означен като bg.uniplovidv.fmi.delc.aot.server, реализиращ абстрактните агенти LPAgent и ReducerAgent.

В клиентския модул е реализиран агент, който служи като посредник между клиентското устройство и агентите от сървърния модел, реализиращи интерпретиращия механизъм. В съответствие със C3A модела, този агент (наречен *IOAgent*) е специфициран като *персистентен*.

В сървърния модул са реализирани два агента (наречени *TempuraAgent* и *InterpreterAgent*), чиято работа е пряко свързана с обработката на ITL твърдения и формули. В съответствие със C3A модела, *TempuraAgent* е *персистентен*, а *InterpreterAgent* е *оперативен*, т.е. генерира се динамично при необходимост.

Клиентски модул

Агентът *IOAgent* събира данни и конструира ITL твърдение или формула, отразяваща моментното състояние на околната среда (ВОП), която се изпраща за

анализ на сървърните агенти. В съответствие с теорията за агентни архитектури и следвайки технологичните особености на JADE, в реалния агент IOAgent се изгражда от два базови Java класа, реализиращи съответно вътрешната архитектура и поведение (функционалността) на агента. С цел улесняване тестване на прототипната версия на AjTempura беше допълнително реализирана тестова среда, симулираща ВОП (като графична среда), в която автоматично се генерират тестови ITL формули и твърдения.

Класът, реализиращ агента, IOAgent, наследява класа Agent - предоставен от развойната среда JADE. Логиката на класа е сравнително проста; реално той служи като рамка, в която се изпълнява поведението на агента. Тежестта на комуникацията и вземането на решения е реализирана в класа, реализращ поведението на агента.

Поведението на IOAgent е реализирано в класа LifeCycleBehavior. Този клас е създаден като наследник на класа Behaviour. В JADE се поддържат различни типове поведения. Класът Behaviour е от най-високо абстрактно ниво; той реализира *най-общия тип* поведение, такова, което се изпълнява *еднократно* - за сметка на това предоставя различни механизми за блокиране или рестартиране на поведението. В JADE е възможно специфициране на повече от десет различни типа поведения, като всички те се реализират като класове, наследници на базовия клас Behaviour. Някои от останалите типове се използват в реализацията на сървърниет агенти на AjTempura.

Поведението е съставено от няколко базови метода, които се изпълняват в определена последователност, в зависимост от входните данни и комуникацията с другите AjTempura агенти. Основните методи, използвани в поведението на IOAgent са следните:

- getTempuraAgentName - това е метод, чрез който IOAgent се опитва да получи името на агент, към който да се обръща за обработката на ITL формула или твърдение;

- sendCreateRequestMessage - чрез този метод се изпраща съобщение, за генериране на оперативен агент, който да отговаря за обработката на конкретно твърдение/формула;

- sendExpressionRequestMessage - методът изпраща съобщение към вече генериран оперативен агент за обработка на твърдение или фомула, дадени като част от съдържанието на съобщението;

- sendDoneMessage - метод, чрез който се изпраща съобщение за приключена работа. Това предизвиква унищожаването на оперативния агент.

Поведението на агентите в мулти-агенти системи (като напр. AjTempura) зависи пряко от комуникацията с останалите агенти в системата. В съответствие с ACL [8] съществуват различни типове съобщения, дефинирани чрез така наречените *перформативи*. Тъй като това е съществен момент от разработката на системата, по нататък ще бъде отделено специално внимание на съобщенията и комуникацията между агентите.

Последният клас от клиентския пакет е *ExpressionGui* - той представлява малка графична форма, която ни дава възможност да проведем тестове със системата преди да бъде интегрирана във ВОП. Формата предоставя текстово поле, където тестващия може да въвежда различни ITL изрази. По този начин можем да използваме оригиналните Tempura Test Suites (вече използвани за тестване на jTempura). Формата предоставя два бутона, като всеки от тях задейства определена част от поведението на агента. Първият бутон изпраща съобщение с ITL израз, който предизвиква изпълнение на метода *sendExpressionRequestMessage*. Вторият бутон служи като изход, т.е. предизвиква обработка на описания по-горе s*endDoneMessage* метод, който от своя страна изпраща съобщение за край на комуникацията.

Друг интересен ефект, реализиран чрез този елементарен графичен интерфейс, е че той реално замества околната среда (ВОП), от която се събират данни и генерират трърденията за интерпретация. От гледна точка на оригиналния интерпретатор, тази графична форма симулира AnaTempura. От гледна точка на бъдещата интеграция на AjTempura, формата временно замества околната среда от агенти, изграждана като ВОП.

Сървърен модул

В сървърния модул са реализирани два агента – като вътрешни архитектури и вградени в тях поведения. Първият агент, наречен *TempuraAgent*, е *персистентен* (в съответствие с C3A модела) и е първата комуникационна цел на *IOAgent*. Задачата на *TempuraAgent* е да отговаря на заявките, получени от IOAgent, като генерира интерпретиращ оперативен агент, който от своя страна да интерпретира ITL израза, представящ актуалното състояние на околната среда. Вторият сървърен агент, наречен *InterpreterAgent*, по своята същност е *оперативен* агент, т.е. генерира се динамично, разширявайки базовата архитектура. Веднъж генериран, този агент отговаря за обработката на конкретно твърдение и след приключване на работата си се самоунищожава. Това е интересна ситуация, произтичаща от C3A модела, където обстоятелствата налагат определен агент да може да се самоунищожи (една от демонстрациите на огромните възможности, предоставяни от C3A модела, за изграждане на гъвкави и мощни софтуерни архитектури).

И двата агента, реализирани в сървърния модул, следват базовата архитектура и предоставените от средата JADE средства и са реализирани чрез наследяване на базови конструкции, като напр. класовете *Agent* и *Behaviour*.

Подобно на *IOAgent*, архитектурата на всеки от агентите се състои от два класа. Единият е наследник на базови клас *Agent*, реализиран в библиотеките на JADE. Вторият клас е реализиран като наследник на един от подкласовете на общия клас за поведение *Behaviour*. Типът на поведението на агента е *CyclicBehaviour*. Характерно за този тип поведения е цикличното изпълнение, което означава повторение на едно и също действие, докато агентът не бъде унищожен. Тази логика се налага от спецификата на комуникацията между агентите, която ще бъде коментирана по-подробно впоследствие.

Архитектурата на агентите в този модул не се различава особено от тази на *IOAgent*, с едно малко изключение - тъй като в нашата архитектура *IOAgent* е инициатор на комуникацията, той е агентът, който търси връзка с останалите агенти (за да изпрати израз за интерпретиране) и той е по-лесен за идентификация. От своя страна сървърните агенти са коренно различни един от друг – единият (*TempuraAgent*) е *персистентен* и осъществява началната връзка с *IOAgent*. Другият (*InterpreterAgent*) е

оперативен агент и отговаря за обработката на конкретен ITL израз. Това налага, при декларацията на агентите, да се дефинира типа на агента. Типизирането е стандартна процедура, залегнала в спецификацията на JADE. Типовете служат за разпознаване на логически еднотипни агентни в една мулти-агентна система. Много често в реални ситуации съобщенията не се изпращат до конкретен агент, а до опредлен тип агенти и първия свободен отговаря на заявката. Реално типът е символен низ, който играе роля на идентификатор. Агентите, които се генерират като инстанции на класа *TempuraAgent* са дефинирани с тип „tempura", а тези, производни от класа *InterpreterAgent* са с тип „tempura-interpreter".

Поведението на *TempuraAgent* е реализирано в клас TempuraProxyBehaviour, който е наследник на CyclicBehaviour, което означава, че това поведение ще се изпълнява многократно. Логиката на поведението се поддържа от следните методи:

- sendConfirmMessage - метод, чрез който *TempuraAgent* изпраща потвърждение към *IOAgent*, че, в отговор на искането за интерпретатор, е генерирал съответния *InterpreterAgent*. В съдържанието на това съобщение също стои името на конкретния *InterpreterAgent* - за да знае *IOAgent* къде да бъде изпратен актуалният ITL израз;

- sendRefuseMessage - методът се активира, ако към агента бъде подадено съобщение, което е с грешен перформатив. Както е известно всяко съобщение в JADE се конструира като перформатив (стриктура на езика ACL) и притежава определен семантичен тип;

- createInterpreter - методът се извиква, когато *TempuraAgent* трябва да генерира нов агент от тип *TempuraInterpreter*. Методът не само генерира нов агент, но генерира и уникален идентификатор на агент, отличаващ го от другите инстанции на този тип агент. Този идентификаторът се изпраща на IOAgent за комуникация с интерпретиращия агент. Този метод е конструиран така, че чрез него всеки *TempuraAgent* може да генерира до хиляда копия (напълно достатъчни за нашата цел) на *InterpreterAgent.*

- Класът *MessageInterpreterBehaviour* реализира поведението на *InterpreterAgent* и всички негови копия. Поведението е реализирано

отново като наследник на CyclicBehavioura. Базовите функционалности, които изпълнява поведението са описани, чрез методите на класа;

- sendRefuseMessage - методът е аналогичен на този, използван при *TempuraAgent*. При получаване на съобщение с грешен перформатив или такъв, който не се разпознава от сензорите на агента, се изпраща съответното съобщение;

- sendConfirmMessage - този метод се използва, за да се изпрати обратно към *IOAgent* резултата от вече интерпретиран ITL израз. В съдържанието на съобщението влизат, както обработената информация, така също и индикация в отговор на коя заявка е този резултат;

- processCommand - този метод е вътрешен за класа, реализиращ поведението. Той не е свързан с между-агентната комуникация, както другите методи. Неговата задача е да предаде постъпилото за обработка ITL твърдение към класовете, извършващи интерпретацията. Тези класове са пакетирани като библиотека под формата на .jar файл. Те бяха описани по-рано в тази глава, като част от ITLAlgorithms пакета на jTempura;

Заключение – резюме на получените резултати

В дисертацията са изследвани проблеми, свързани с контекснтно-ориентирано управление на електронните услуги, доставяни във виртуално образователно пространство. Особено внимание е обърнато на времевите аспекти на това управление и по-специално на идентифициране и подредба във времето на събития и промени, случващи се в разпределената инфраструктура на пространството. За реализиране на това управление е избран формализъм, базиращ се на интервална модална логика, познат като ITL (Interval Temporal Logics) и поддържащата го софтуерна среда Tempura.

Приносите на автора, представени в дисертационния труд, са научно-приложни и приложни.

Научно-приложните могат да бъдат обобщени както следва:

1) Предложен е подход за реинженеринг на оригиналния интерпретатор на Tempura;

2) Разработени е теоретичен модел, наречен C3A, за контекстно-зависими софтуерни архитектури, който се използва за реализиране на контекстно-зависима мулти-агентна версия на Tempura интерпретатора, наречен AjTempura. В C3A се различават два типа агенти – *персистентни* и *оперативни*. Персистентните агенти предоставят минималната функционалност на едно приложение, която включва също възможности за идентифициране на случващи се промени в околната среда. В съответствие с класическата теория за агенти, тези агенти са винаги налични. В отклонение от класическата теория, оперативните агенти не са винаги налични – в зависимост от вида на идентифицираните промени в околната среда (в нашия случай ВОП) те се генерират динамично (от кореспондиращи персистентни агенти). Тяхната задача е да извършват компенсиращи действия, необходимостта от които е породена от случващите се промени. След окомплектоване на съответните компенсации, тези агенти се унищожават (или самоунищожават). Моделът е независим от конкретната приложна област и може има различни приложения;

3) За реализиране на AjTempura е предложен също втори модел, наречен *трансформационен*, който представя възможен начин за реинженеринг на обектно-ориентираната версия jTempura в AjTempura. Съществото за трансформационния модел е използването на пакетния модел на Java за спецификация на *абстрактни агенти*. Едно от предимствата на това разделяне – абстрактни и реални агенти - е, че впоследствие моделът (абстрактните агенти) може да бъде реализиран в различни развойни среди (в дисертацията са реализирани в JADE).

Приложните приноси на автора са следните:

4) Разработена е нова обектно-ориентирана версия на интерпретатора, наречена jTempura. В съответствие с реинженерингов подход, jTempura е междинна стъпка към крайната цел – създаване на AjTempura. jTempura може да се използва също като самостоятелен софтуерен продукт и да се вгражда в обектно-ориентирани архитектури, използващи JVM;

5) Реализиран е контекстно-зависим мулти-агентен интерпретатор (AjTempura). AjTempura се интегрира в изграждащото се виртуално образователно пространство (ВОП) за усилване неговата контекстно-зависимост;

6) Създадена е програмна реализация на С3А модела, вградена в AjTempura. Първите тестове на AjTempura демонстрират големия потенциал на С3А модела за създаване на гъвкави и мощни софтуерни архитектури.

Междинни резултати и отделни части от работата, освен в цитираните публикации, са представени и в отчетите на международни и национални научно-изследователски проекти, в които е участвал авторът (виж Приложения).

За бъдещото развитие на получените в дисертацията резултати предвиждаме следните насоки:

- *jTempura* – усъвършенстване на обектно-ориентираната структура и стандартизиране на интерфейсните класове, с цел по-лесно интегриране в различни обектно-ориентирани приложения;

- *AjTempura* – продължава работата по интегриране във ВОП. Разработване на различни типове интерфейсни агенти, които напълно да заменят AnaTempura и да разширят възможностите за интеграция на интерпретатора;

- *Усъвършенстване на двата модела* - провеждане на експерименти, анализ и оценка на резултатите, които ще се използват за усъвършенстване на С3А модела и трансформационния модел;

- *Формални средства за моделиране на ВОП* – за моделиране и изследване на виртуалното пространство предвиждаме изграждане на моделираща среда, в която ще бъде възможно комбиниране на AjTempura с други формални средства, като напр. CCA [27] и CS-Flow. Така ще могат да се изследват различни характеристики на пространството.

Цитирани публикации на автора

1) Stoyanov, S., I. Popchev, E. Doychev, D. Mitev, V. Valkanov, A. Stoyanova-Doycheva, V. Valkanova , I. Minov. DeLC Educational Portal. *// Cybernetics and Information Technologies (CIT),* vol. 10, No 3, 2010, pp. 49-69. <http://www.cit.iit.bas.bg/CIT_2010/CIT_10-3.html>
2) Valkanov, V., D. Mitev. Tempura Reengineering. // *REMIA 2010* : *Anniversary International Conference,* 2010, pp. 311-320.
3) Stoyanov, Stanimir, Ivan Ganchev, Damyan Mitev, Vladimir Valkanov, Martin O`Droma. Service-oriented and Agent-based Architecture Supporting Adaptable, Scenario-based and Context-aware Provision of Mobile E-learning Services. //*IJCISIM,* vol. 3, 2011, pp. 771-779. ISSN 2150-7988
4) Stoyanov, S., H. Zedan, E. Doychev, V. Valkanov, I. Popchev, G. Cholakov, and M. Sandalski. Intelligent Distributed eLearning Architecture. // *Intelligent Systems, InTech March*, *V.M.Koleshko (Ed.)*, Hand Cover, 2012, 366 pages, pp. 185-218, ISBN 978-953-51-0054-6
5) S. Stoyanov, E. Doychev, A. Stoyanova-Doycheva, V. Valkanova, V. Valkanov, Education Cluster Supporting eTesting and eLearning in Software Engineering, 2nd Annual International Conference on Web Technologies & Internet Applications (WebTech 2012), 7- 8 May 2012, Bali, Indonesia, 23-28.

6) Владимир Вълканов, „Темпура реинжинеринг", Докторантски семинар по проект "Изграждане на висококвалифицирани млади изследователи по съвременни информационни технологии за оптимизация, разпознаване на образи и подпомагане вземането на решения" - BG051PO001-3.3.04/40 , 16 февруари 2010, ИИТ-БАН, София;

7) Владимир Вълканов, „Context-aware management of e-services (Tempura reengineering)", 13th Workshop on "Software Engineering, Education and Reverse Engineering", DAAD, 24th August – 31th August 2013, Банско, България

Библиография

[1] Allemang, D. и J. Hendler, *Semantic Web for the Working Ontologist*.: Elsevier, 2011, ISBN 978-0-12-385965-5.

[2] Ashton, K. , "That 'Internet of Things' Thing", *RFID Journal*, 2009.

[3] Berners-Lee, T. , J. Handler и O. Lassila, "The Semantic Web", *Scientific American* , no. 284, pp. 34-43, May 2001.

[4] Chandra, A. , J Halpern и A. Meyer, "Equation between regular terms and an application to process logic" в *Thirteenth Annual ACM Symposium on Theory of computing* , Milwaukee, Wisconsin,May, 1981, pp. 384-390.

[5] Dey, A. K., "Understanding and Using Context", *Personal and Ubiquitous Computing Journal*, vol. 5, no. 1, pp. 4-7, 2001.

[6] Dey, A.K. и G.D. Abowd, "Towards a better understanding of context and context-awareness" в *Proceedings of the Workshop on the What, Who, Where, When and How of Context-Awareness*, New York, ACM Press, 2000.

[7] Dieter, U. , H. Mark и M. Florian, "Architecting the Internet of Things", *Springer*, 2011, ISBN 978-3-642-19156-5.

[8] FIPA. (2013, May) FIPA ACL. <http://www.fipa.org/repository/aclspecs>

[9] Ganchev, I. , S. Stoyanov и M. O'Droma, "Consumer-oriented DeLC Service Architecture" в *In Proc. of the 3rd International Conference on Education and Information Systems, Technologies and Applications (EISTA 2005)*, Orlando, Florida, USA, ISBN 980-6560-34-5 , 2005, pp. 213-218.

[10] Ganchev, I. , S. Stoyanov, V. Valkanova и M. O'Droma, "Service-oriented and Agent-based Architecture Supporting Adaptable Context-Aware Provision of Mobile e-Learning Services" в *IADIS e-Learning 2010 Conference*, Freiburg, Germany, 26 - 29 July 2010, pp. 97-104.

[11] Harel, D. , "First-order dynamic logic", *Lecture notes in computer science*, vol. Number 68, Berlin, 1979.

[12] Harel, D. и D Kozen, "Process logic: Expressiveness, decidability, completeness", *Journal of computer and System science*, vol. vol 25, pp. 144-170, October 1982.

[13] Hoare, C. A.R., "An axiomatic basis of computer programming.", *Communications of the ACM* , pp. 576-580, Octomber 1969.

[14] Hoare, C. A.R., *Communicating sequential processes*. London: Printise Hall International, 1985.

[15] Hoare, C. A.R., *Towards a theory of parallel programming in C*, C.A.R. Hoare and R.H. Perrot, Ed.

London: Academin Press London , 1972.

[16] Manna, Z. и A. Pnueli, "Verification of concurent programs: Temporal framework", *The Correctnes Problem in Computer Science*, pp. 215-273, 1981.

[17] Pratt, V. , "Semantical considerations on Floyd-Hoare logic" в *Proceedings of the Seventeenth Annual IEEE symposium on Foundations of computer science*, Houston,Texas, Octomber, 1976, pp. 109-121.

[18] Stoyanov, S. , "Context-Aware and Adaptable eLearning Systems", STRL, De Montfort University, Leicester, UK, PhD Thesis 2012.

[19] Stoyanov, S.; Ganchev, I.; O'Droma, M.; Zedan, H.; Meere, D.; Valkanova, V., "Semantic Multi-Agent mLearning System" in *Semantic Agent Systems: Foundations and Applications*, A. Elci, M. T. Kone, and M. A. Orgun, Eds.: Springer Verlag, 2011, vol. 344, ISBN: 978-3-642-18307-2.

[20] Stoyanov, S.; Popchev, I.; Doychev, E.; Mitev, D.; Valkanov, V.; Stoyanova-Doycheva, A.; Valkanova, V.; Minov, I., "DeLC Educational Portal", *Cybernetics and Information Technologies (CIT)*, vol. 10, no. 3, pp. 49-69, 2010.

[21] Stoyanov, S. , I. Ganchev, I.Popchev, M.O'Droma и V. Valkanova, "Agent-Oriented Middleware for InfoStation-based mLearning Intelligent Systems" в *5th IEEE International Conference on Intelligent Systems IS'10*, London, IEEE Catalog Number: CFP10802-CDR, ISBN:978-1-4244-5164-7, Library of Congress:2009934065 , 07.07 – 09.07.2010, pp. 91-95.

[22] Stoyanov, S. , I. Ganchev, D. Mitev, V. Valkanov и M. O'Droma, "Service-oriented and Agent-based Architecture Supporting Adaptable, Scenario-based and Context-aware Provision of Mobile e-Learning Services", *International Journal of Computer Information Systems and Industrial Management Applications*, vol. 3, pp. 771-779, 2011, ISSN 2150-7988.

[23] Stoyanov, S. , I. Ganchev, I. Popchev и M. O'Droma, "An Approach for the Development of a Context-Aware and Adaptive eLearning Middleware" in *Intelligent Systems: From Theory to Practice*, V. Sgurev et al., Ed. Berlin Heidelberg: Springer-Verlag, 2010, pp. 519-535, ISBN: 978-3-642-13427-2.

[24] Stoyanov, S. , I. Ganchev, I. Popchev и M. O'Droma, "An Approach for the Development of InfoStation-Based eLearning Architectures", vol. 61, no. 9, pp. 1189-1198, 2008.

[25] Stoyanov, S. , I. Ganchev, I. Popchev, M. O'Droma и R. Venkov, "DeLC – Distributed eLearning Center" в *1st Balkan Conference in Informatics BCI'2003*, Thessaloniki, Greece, ISBN 960-287-045-1 , 2003, pp. 327-336.

[26] Stoyanov, S. и I. Popchev, "Evolutionary Development of an Infrastructure Supporting the Transition from CBT to e-Learning", *Cyberbetics and Information Technologies (CIT)*, vol. 2, pp. 101-114, 2006, Bulgarian Academy of Sciences.

[27] Stoyanov, S. и др., "Intelligent Distributed eLearning Architecture" in *Intelligent Systems*, V. M. Koleshko, Ed.: ИнТецх, 2012, pp. 185-218, Hard cover, 978-953-51-0054-6.

[28] Valkanov, V. , "Context-aware management of e-services (Tempura reengineering)", 13th Workshop on "SOftware Engineering, Education and Reverse Engineering", Банско, България, 2013.

[29] Дойчев, Емил , *Среда за електронни образователни услуги - дисертация*. Пловдив: Пловдивски университет "П. Хилендарски", 2013.

[30] Орозова, Д. , С. Стоянов и И. Попчев, "Виртуално образователно пространство" в *Научна конференция с международно участие „Знанието – традиции, иновации, перспективи”*, Бургас, 2013, приета за печат.