ПЛОВДИВСКИ УНИВЕРСИТЕТ „ПАИСИЙ ХИЛЕНДАРСКИ”
ФАКУЛТЕТ ПО МАТЕМАТИКА И ИНФОРМАТИКА
КАТЕДРА „СОФТУЕРНИ ТЕХНОЛОГИИ”

СТОЯН НИКОЛОВ ЧЕРЕШАРОВ

МОДУЛИ ЗА ИЗГРАЖДАНЕ НА УЕБ БАЗИРАНИ СОФТУЕРНИ СИСТЕМИ

АВТОРЕФЕРАТ

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

Научен ръководител: проф. д-р Христо Крушков

Пловдив, 2017 г.

Дисертационният труд е обсъден и насочен за защита на разширено заседание на катедра „Софтуерни технологии” при Факултета по математика и информатика на ПУ „Паисий Хилендарски“.

Дисертационният труд се състои от 158 страници, от които 121 основен текст. Използваната библиография включва 117 източника, от които 6 на български, 111 на английски език, от които 24 интернет източника.

Оформени са 3 приложения: Приложение 1. CoolCSN – проект с отворен код, прототип на модела; Приложение 2. Приложни и експериментални системи изградени с прототипа; Приложение 3. УБСС за обслужване на отдел „Развитие на академичния състав” към ПУ „Паисий Хилендарски”.

Приложени са списък с авторските публикации по темата съдържащ 5 заглавия, от които 3 на английски и 2 на български език, списък с изследвания изготвени в изпълнение на задачи по научни проекти, списък с изследвания докладвани на научни сесии и конференции..

Защитата на дисертационния труд ще се състои на 23 февруари 2018 г. от 14:00 часа в Заседателната зала на новата сграда на ПУ „Паисий Хилендарски“, гр. Пловдив.

Материалите по защитата са на разположение на интересуващите се в секретариата на ФМИ, нова сграда на ПУ, каб. 330, всеки работен ден от 8:30 до 17:00 часа.

Автор: Стоян Николов Черешаров

Заглавие: „Модули за изграждане на уеб базирани софтуерни системи“.

Университетско издателство „Паисий Хилендарски“
Тираж: 100 бр.
Пловдив, 2017 г.

Съдържание

Обща характеристика на дисертационния труд

Актуалност на проблема

Най-динамично развиващата се област на информационните технологии е именно уеб програмирането. Свидетели сме на Интернет революция [1]. Интернет вече се е превърнал в неотменна и абсолютно задължителна част от общественото развитие на човека. Той разширява възможностите на човешкия мозък.

Изграждането на софтуерни системи е сложен и отнемащ време процес. Проектирането на обектно-ориентиран софтуер е трудно, а проектирането на обектно-ориентиран софтуер за многократна употреба още повече [2]. Динамичното развитие на света и обществото поставят все по-големи изисквания и предизвикателства пред разработчиците на софтуер [3]. Все по-сложни системи трябва да бъдат създавани за все по-кратки срокове [4].

Пренаписването на код с една и съща функционалност за различни технологии и проекти е често срещан проблем [5]. Целта е кодът за реализиране на дадена функционалност да се напише веднъж и да се преизползва в колкото е възможно повече системи.

Бурното развитие на интернет на нещата (IoT – Internet of Things), изкуствения интелект (AI - Artificial Intelligence), облачните технологии (Cloud), уеб услугите (Web Services), безсървърната архитектура (Serverless Architecture), софтуерът като услуга (SaaS – Software as a Service) и като цяло изместването на приложенията в интернет са предпоставка за търсене на начини за ускоряване и облекчаване на процеса по изграждане на уеб базираните софтуерни системи без компрометиране на качеството им.

Цели и задачи на дисертационния труд

Основната цел на дисертационния труд е *да се създаде модел за изграждане на модулна система, с помощта на който процесът по създаване на уеб базирани софтуерни системи да се ускори, улесни и да се подобрят качеството и надеждността им.* Този модел трябва да може да се прилага във възможно най-голям брой различни технологии. С други думи, той трябва да е технологично независим (Technology Agnostic). С помощта на модулите трябва да стане възможно по-бързото и лесно изграждане на качествени уеб базирани системи, отговарящи на съвременните изисквания на софтуерните технологии.

Основните подцели (отбелязани с точки, напр. 1.) и задачи (отбелязани с подточки, напр. 1.1.), произтичащи от така поставената цел, са:

1. Да се изследват и анализират най-популярните технологии, работни рамки и платформи за разработване на модулни уеб базирани софтуерни системи и се направи обзор на модулни системи и решения.
   - 1.1. Да се създаде понятиен апарат.
   - 1.2. Да се анализират софтуерните технологии, работни рамки и платформи от гледна точка на модулни системи.
   - 1.3. Да се направи обзор на модулни системи и решения.
   - 1.4. Да се направи класификация на основните типове уеб базирани софтуерни системи.
2. Да се създаде концептуален модел на модулна система за бързо и качествено изграждане на уеб базирани софтуерни системи.

2.1. Да се дефинират изискванията към модела.
2.2. Да се създаде концептуален модел на база поставените изисквания.
2.3. Да се повиши абстракцията на модела, чрез използване на формални математически подходи и теории за моделиране на дискретни паралелни процеси. Това би довело до намаляване броя на необходимите модули и увеличаване обхвата на решаваните с тяхна помощ проблеми. Графичното математическо представяне на проблема би позволило създаване на приложения без писане на код, а изцяло с графични средства.
2.4. Да се намерят подходи и начини за абстракция, обобщаване и опростяване работата на модела с бази данни при боравене с предварително неизвестни и неопределени понятия.

3. Да се проектира, реализира и тества прототип на модулна система, по създадения концептуален модел с някоя от популярните софтуерни технологии.
3.1. Да се избере подходяща софтуерна технология и архитектура и да се проектира прототип на модулна система с използване на съвременните добри практики, шаблони за дизайн и методологии.
3.2. Да се реализират, документират и тестват базовите модули, съставящи модулната система. Някои от тях да се предложат като проекти с отворен код, за да се осигури поле за изследвания на ефекта от внедряването на модела.
4. Да се използва на практика създадения прототип на модулна система за учебни, изследователски и приложни цели. Да се използват модулите за изграждане на разнообразни уеб базирани софтуерни системи (УБСС).

Структура и обем на дисертационния труд

Дисертационният труд се състои от увод, три глави, заключение, три приложения и списък с използвана литература.

В първата глава се правят обзор и анализ на най-популярните софтуерни технологии и работни рамки, изградени на тяхна база. Направен е анализ и на готови софтуерни платформи. Разгледани са основните понятия и концепции, използвани в дисертационния труд. Обърнато е внимание на пакетните мениджъри (или мениджъри на зависимостите). Направен е преглед на съществуващи модулни решения. Обсъдени са работни рамки в най-популярните софтуерни технологии. Особено внимание е отделено на рамки, предлагащи упражняването на ORM подходи. Класифицирани са УБСС според начина на представяне на информацията, за да се определи обхватът на изследването.

Във втора глава е представен концептуален модел. Определят се специфичните изисквания на модела към изграждането на основните модули и базовата работна рамка. Разгледани са отделните логически слоеве в модулите. Набелязани са методите за инсталация и интеграция с основната софтуерна система.

Глава трета предлага архитектурата на модулите и тяхната конкретна реализация. Описани са приложенията им за бързо изграждане на уеб базирани системи. Демонстрира се как лесно и бързо може да се изгради отворена и гъвкава система, без необходимост от подробно познаване на определена софтуерна технология.

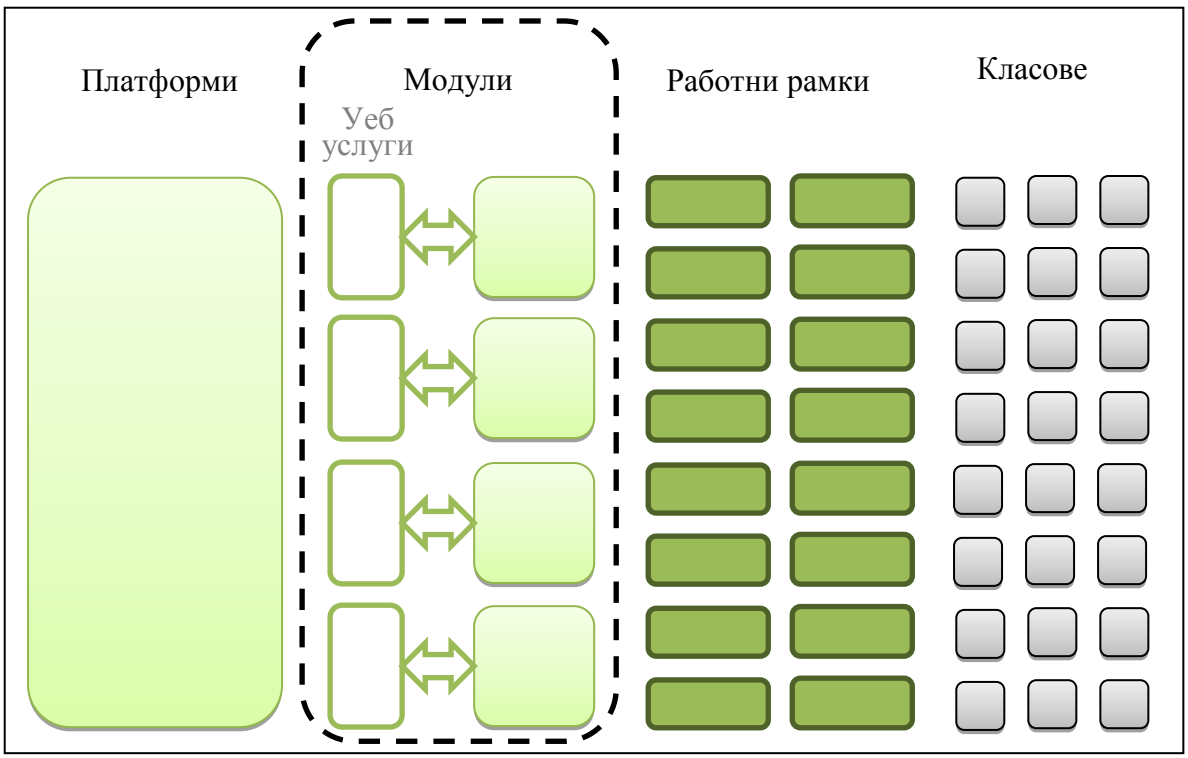
В заключението се прави обобщение на описания модел и са очертани възможностите за неговото бъдещо развитие.

Приложенията представят софтуерни системи изградени с помощта на описания модел.

Списъкът с използвана литература включва 117 източника от тях 111 на английски и шест на български език

Кратко съдържание на дисертационния труд

1. Технологии за изграждане на УБСС



Фигура 1 Модулна система от друго ниво

На Фигура 1 е представена предлаганата от нас модулна система от друго ниво (очертана с пунктир) сред другите модулни системи от ниво система за програмиране (класове на фигурата), работни рамки и платформи. Тя заема средно място между системите на ниво работни рамки и платформи.

Има основно три пътя, по които можем да поемем при изграждане на УБСС. Това са подходите с използване на трите вида модулни системи представени на Фигура 1:

- Използване директно средствата на езика за програмиране (означен с „класове“ на Фигура 1 Модулна система от друго ниво).
- Работни рамки.
- Платформи.

Директното използване на езика за програмиране е изключително труден и бавен подход: за да се внесе дисциплина и еднаквост при писането на кода ще трябва да се разработят и наложат стандарти и конвенции за имена и кодиране. Нужно е разработчиците да познават перфектно шаблоните за дизайн [2] [6] и добрите практики, да се създаде документация, тестове, стандарти отговарящи на съвременните изисквания и т.н. Неминуемо ще се наложи създаване на работна рамка. Това е огромен труд, отнемащ месеци и години.

При съвременните условия на бързо развиващ се бизнес няма кой да чака толкова дълго време само, за да тества бизнес идеята.

Ако поемем по другия път и използваме работна рамка то ще трябва да създадем цялата функционалност сами. Това е крачка напред в сравнение с директното използване на езика за програмиране, защото ще имаме наготово поне стандартите за имена и кодиране, компоненти, шаблоните за дизайн и тестовете. Но ще трябва да създадем схемата на базата данни, механизмите за оторизация, идентификация, навигация, CMS и т.н. Това е огромен труд, който изисква екипи от разработчици, време и ресурси. Нещо повече, ако искаме да изградим същата система, но с друга технология, ще трябва да търсим друга работна рамка и отново ще се наложи да преоткрием решенията на проблеми, които вече сме решили.

Ако използваме някоя готова платформа (Joomla, WordPress etc.) буквално за няколко минути, един разработчик ще доведе нещата до работеща CMS система. Но неминуемо ще се сблъскаме с проблема, че няма приставка, която да управлява работни процеси или да решава специфичните счетоводни проблеми. Ако дори намерим такива, те едва ли ще могат да работят заедно. Ще трябва да се пишат приставки с подходите и парадигмите на съответната платформа, които обикновено са остарели, сложни и носят белезите на специфичната платформа, няма начин да преизползваме написаното за друга платформа, стиковането на компонентите е трудно и мъчително, дори най-малката промяна в съществуващата вече функционалност е истинско предизвикателство.

Тук може да дойде на помощ предложеният от нас модел. Той заема мястото си между модулните системи на ниво работна рамка и платформа (Фигура 1). С помощта на модули изграждаме общата за всички системи функционалност и после дописваме само и единствено специфичната функционалност като модул. Нашето решение не е свързано със създаването на поредната работна рамка, която само увеличава броя им и прави избора по-труден – стремежът ни е да използваме вече готови рамки, които отговарят на поставените от модела изисквания и да се изгради система от ново ниво. Ние не описваме работна рамка, а използваме съществуващи.

Основни понятия

Софтуерна технология - Софтуерна технология или технология за създаване на софтуер представлява език за програмиране и съпътстващите го платформи, инструменти, библиотеки, приложни сървъри, среди за разработка и др., които ни позволяват да изграждаме софтуерни приложения.

Софтуерни работни рамки - Софтуерната работна рамка е преизползваемо „полузавършено“ приложение, което може да бъде специализирано за създаване на потребителски приложения [7] [8].

Софтуерни модули. Модулно програмиране - ***Какво ще се нарича модул, зависи от контекста, дизайна и съдържанието на проекта.*** *Модулът е концепция, която идва от парадигмата за модулно програмиране. Тя защитава идеята, че софтуерът трябва да се състои от отделни, заменими компоненти, наричани модули, като те съдържат отделна функционалност на системата. Понятието модул е концепция, значението на което се определя от контекста.*

Пакетен мениджър - *служи за дефиниция на модули, споделяне, тяхното зареждане и следене на зависимостите.*

Уеб базирани софтуерни системи (УБСС) - Липсва формално общоприето научно определение на понятието уеб базирана софтуерна система. От научни публикации, засягащи темата [9] [10] [11], става ясно, че *уеб софтуерна система или уеб базираната софтуерна система е информационна система, която използва интернет уеб технологии, за да предложи информация и услуги на потребителите си или на други информационни системи.*

Система за управление на работни процеси (Workflow System) - Системата за управление на работни процеси служи за автоматизирането изцяло или на част от бизнес процес, при което документи, информация или задачи се предават от един на друг участник за обработка, съгласно множество от процедурни правила [12].

Мрежи на Петри (Petri Nets) - Мрежи на Петри е формален и графичен език, подходящ за моделиране на системи за едновременно (конкурентно) използване от много потребители на споделени ресурси. Тя е обобщение на теорията на крайните автомати, позволяваща описание на едновременни събития.

Софтуерни платформи - *някои системи са се развили до степен, в която надхвърлят първоначалното си предназначение и са се превърнали в многофункционални платформи. Като пример можем да посочим WordPress, Joomla и др.*

Софтуерните технологии като модулни системи

В контекста на език за програмиране като модул се разглежда и възприема множество от функции, методи, класове, интерфейси и всички елементи на езика, които могат да се капсулират в преизползваема структура (Фигура 1 Модулна система от друго ниво, означени като „Класове“). Този тип модули предлагат завършена функционалност от ниско ниво. Обикновено не могат да бъдат разбити на по-малки градивни единици – те са атомите на системата и могат да се менажират от пакетните мениджъри на технологията, понеже пакетните мениджъри не поставят ограничение за размера и за съдържанието на модула.

Софтуерните работни рамки, като модулни системи

В контекста на работните рамки модулите обединяват групи модулни елементи (класове, методи и др.) от по-долно ниво, за да предложат ново качество. Модулите на ниво работна рамка предлагат завършена функционалност от по-висока степен (Фигура 1 Модулна система от друго ниво, означени като „Работни рамки“).

Софтуерните платформи като модулни системи

В контекста на софтуерните платформи модулите представляват инсталируеми приставки, написани и пакетирани по правилата на платформата, които се инсталират със специфични за платформата инсталатори.

Обзор на съществуващи модулни решения

В [13] Ju описва конвенциите за модулно програмиране на асемблерните езици. Неговата публикация предлага конвенция за последователност на повиквания и обработваща услуга за интермодулна комуникация. Схемата е лесна за използване и подобрява добрите програмни практики като простота, гъвкавост, разбираемост и цялост.

В [14] Busbee изследва модулния структуриран подход в C ++. Авторът описва възможен подход за създаване на модулна система в C ++ с елементите на програмния език. Това е модулна система на ниско ниво, а не за уеб базирани системи.

Модулните системи описани в [13] и [14] са на ниво програмни езици и не предлагат решения за уеб базирани системи. Те могат да ни послужат като концептуална основа.

Друг подход, базиран на модулна система за платформата NetBeans, е описан в [15]. Авторът заявява, че никой вече не пише софтуер само в един екип. Извън света на вградените системи (Embedded Systems) почти всеки разчита на библиотеки и рамки, написани от някой друг. Чрез използването им е възможно разработчикът да се съсредоточи върху действителната логика на приложението, като използва повторно инфраструктурата, рамките и библиотеките, написани и предоставени от други. Това намалява времето, необходимо за разработване на софтуер.

Реактивните блокове (Reactive Blocks) е модулна система за изграждането на IoT [16]. Тя използва модули наричани блокове (blocks) за изграждането на IoT приложения. Тази модулна система решава същите проблеми като нашите за преизползване на функционалност, ускоряване на разработването, повишаване на качеството, лесна инсталация и включване в проект, баланс между сложност и функционалност, графично изграждане на приложението. “Reactive Blocks” е приставка (Plugin) за Eclipse, която позволява систематично изграждане на приложение от изграждащи блокове. Изграждащите блокове могат да съдържат Java код и да имат специални описания на интерфейси, които надхвърлят традиционните интерфейси. Ето защо блоковете могат лесно да се споделят с други разработчици и лесно да се използват повторно. По принцип всичко, което се изпълнява в Java, може да бъде капсулирано в изграждащ блок. Това, което е изненадващо трудно в програмния код, е лесно да се изрази чрез графика [17].

Изследването на Parnas [18] е изключително важно и популярно в областта на модулното програмиране. То дава основа за разделяне на системата на модули като обяснява конвенционални и неконвенционални подходи при определяне обхвата на модула.

Изследвания и разработки в областта на работните процеси, системите за управлението и автоматизацията им, както и математическите теории за тяхното моделиране също са в обсега на вниманието ни понеже нашият модел трябва да предложи ново ниво на абстракция базирано на математически формализъм. Мрежите на Петри е първият опит да се моделират математически дискретни, паралелни процеси. Поради тяхната простота за възприемане и широко разпространение, ние ще използваме тях в модулната ни система. Това е причината, поради която ще бъде отделено място на изследвания и разработки в тази насока.

В [19] Zhen предоставя формализирана дефиниция на работните процеси, ограничени от входа и изхода. Предлага се модел, базиран на мрежи на Петри (Petri Nets).

Атанасов в [20] представя формални дефиниции на понятията "интуиционистки размити обобщени мрежи" от първи, втори, трети и четвърти тип и описва алгоритмите за прехвърляне на ядрата в отделните преходи и мрежите като цяло. Обобщените мрежи представляват разширение на мрежите на Петри (Petri Nets). Това е друг инструмент за моделиране и оптимизиране на дискретни паралелни процеси. В нашите модули търсим формален математически подход за моделиране на дискретни паралелни процеси – това изследване дава друг математически формализъм за реализиране на система за управление на работни процеси.

Интервалната темпорална логика (Interval Temporal Logic (ITL)) представена от Moszkowski в [21] е друг възможен математически формализъм, който може да се използва за описание на процеси. В [22] авторът разглежда връзката й с теорията на крайните автомати. Тези изследвания са свързани с нашата цел да формализираме широк кръг проблеми и намерим общ подход при решаването им чрез модули.

Основни видове уеб базирани информационни системи според начина на представяне на информацията

Обхватът на нашето изследване включва следните типовете системи, които могат да бъдат изградени с описания в дисертацията модел.

- Тип брошура
- Блог
- WiKi
- Форум
- За електронна търговия
- Социална мрежа
- Системи за управление и автоматизация на работни/бизнес процеси
- Специализирани

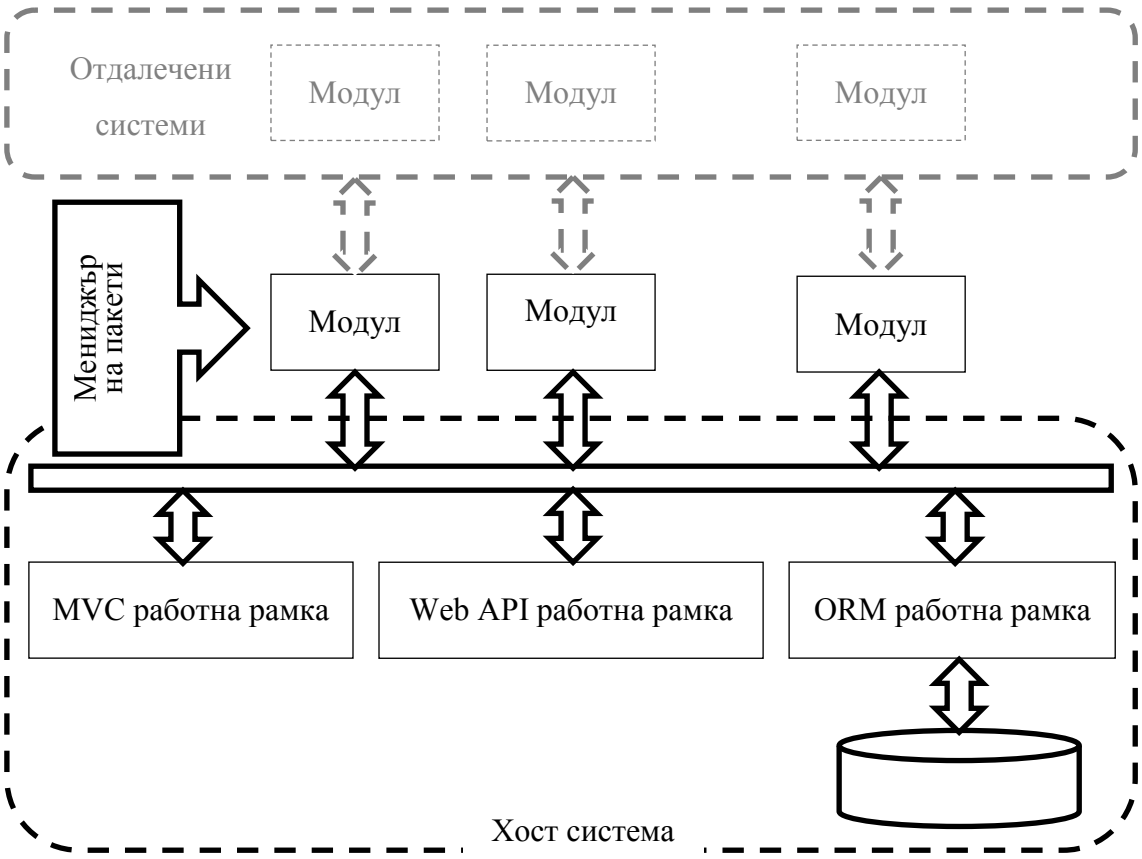
Специализираните системи остават извън обхвата на изследването, макар някои от тях също да е възможно да бъдат изградени по модела.

Видовете УБСС попадащи в обхвата на изследването, имат някои общи функционалности, които са характерни за повечето от тях. Така например функционалностите идентификация, оторизация, навигация и управление на съдържанието, могат да бъдат реализирани като отделни модули. *Макар че за нашето изследване е важен моделът за изграждане на модулна система, а не толкова видът на конкретните модули. Ако променим обхвата на изследването и включим други видове УБСС, единственото, което би могло да се промени, са само видът на нужните модули.* Някои системи, се нуждаят от система за управление на работни процеси. Тази система е по-специфична поради нейния абстрактен характер, който позволява комбинирането й с останалите, при което се създава ново качество. Тя разширява кръга от проблеми, които могат да се решат с такава УБСС. Могат да се открият и други общи функционалности, които имат отношение към техническото функциониране на системата и имат повече или по-малко спомагателен характер, като механизми за кеширане, сесия, локализация и превод, управление на файлове. Тези функционалности не са абсолютно задължителни, но е добре да бъдат добавени.

От по-горе описаните общи функционалности можем да заключим, че наборът от модули, който позволява с негова помощ да се изгради всяко от цитираните типове приложения включва *основните модули за идентификация, оторизация, навигация, управление на съдържанието, управление на работни процеси и спомагателните модули за кеширане, управление на сесиите, локализация и превод, управление на файлове.* Не е задължително при изграждане на всяка УБСС да се използва всеки един от описаните модули. Използват се само тези, които са нужни за постигане на поставената цел. Моделът позволява добавяне на други модули при необходимост за увеличаване на видовете функционалности.

2. Концептуален модел на модулна система

В тази глава е описан концептуален модел на модулна система от различно ниво за бързо и качествено изграждане на уеб базирани софтуерни системи. Използвани са идеи, характерни за широк набор от стилове за програмиране като процедурно, обектно-ориентирано, аспектно-ориентирано, модулно програмиране, както и математически методи от теорията на мрежи на Петри (Petri Nets). Моделът се обляга на работни рамки и мениджър на зависимости, като позволява и отдалечена комуникация между модулите (Фигура 2).



Фигура 2 Концептуален модел на модулна система

На Фигура 2 всеки от модулите е изграден с помощта на утвърдени и доказали качествата си работни рамки. Моделът не предлага поредна работна рамка, която се конкурира с подобни такива, преоткривайки по малко по-различен начин решения на вече решени проблеми. Той предлага да се стъпи на вече утвърдени работни рамки като създаде ново качествено по-високо ниво на автоматизация на програмисткия труд. MVC и Web API работните рамки на практика предлагат презентационни слоеве за комуникация с потребители или други приложения. ORM работната рамка предлага виртуален слой за комуникация със слоя за постоянно запазване на данните (Persistence Layer). От гледна точка на модула това са все входно-изходни канали за комуникация на данни.

MVC работната рамка позволява на модула да се включи в УБСС и при нужда да комуникира с потребителя, като част от уеб приложение. Обикновено тази работна рамка предлага мениджър на събития (Event Manager), реализация на принципа обръщане на

контрола (IoC) чрез контейнер за инжектиране на зависимости (DiC) или локатор на услуги (Service Locator) и др. Ако MVC работната рамка не предлага тези услуги, то моделът допуска включването на допълнителни работни рамки за удовлетворяване на това изискване.

Web API работната рамка дава възможност на модула да предлага функционалността си посредством уеб услуги (Web Services). По този начин модулът може да бъде използван при изграждане на приложения с безсървърна архитектура (Serverless Architecture), едностранични приложения (Single Page Application (SPA)), мобилни приложения и при комуникация между приложения. Ако модулите на модулната система са разположени на отдалечени възли в Интернет, то това е основният тип комуникация между тях. Моделът не поставя ограничение за начина на комуникация, но в настоящия момент това основно се осъществява с RESTful уеб услуги (Web Services) базирани на JSON. Възможността за отдалечена комуникация между модулите позволява те да бъдат създадени с различни технологии и разположени на отдалечени системи в Интернет (Фигура 2).

ORM работната рамка дава възможност на модула да се изолира от слоя за запазване на данните. По този начин модулът работи с една и съща виртуална база данни, независимо от вида и начина на реалното запазване на данните. Този абстрактен слой позволява модулите да работят на различни платформи без промяна на кода им. ORM работните рамки се грижат за правилното изпълнение на транзакциите и прихващане на изключенията.

Друга характерна особеност на модела е, че инсталацията на отделните модули става със стандартния мениджър на пакети (Фигура 2) за технологията, с която е създаден модулът. Това позволява всеки модул лесно да се добави към съществуващи проекти на технологията и евентуално да служи като свързващо звено към функционалности на други модули, създадени с други технологии. От своя страна модулът трябва да е изграден в съответствие с изискванията и правилата на мениджъра на пакети за съответната технология. Лесната инсталация със стандартни и популярни средства би позволила преизползване на функционалността, предлагана от отделните модули и тяхното популяризиране сред разработчиците.

Изисквания към модела

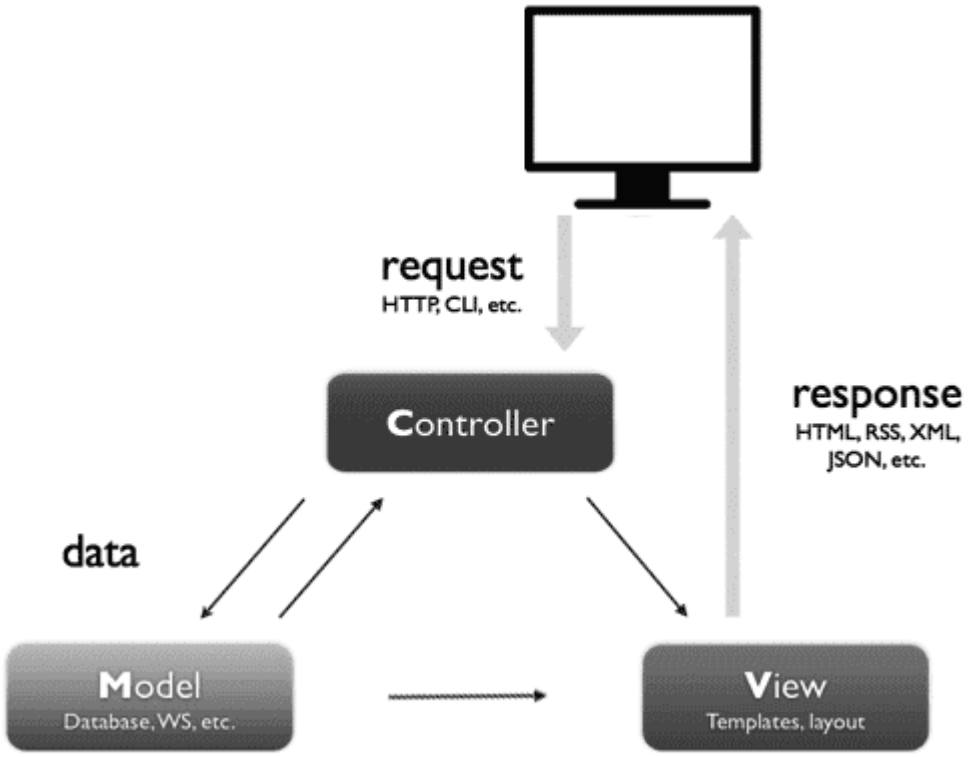
Моделът трябва да предлага изграждането на модулна система, притежаваща следните характеристики.

1. Отвореност за разширяване и затвореност за промяна.
2. Гъвкавост.
3. Разпределеност и хетерогенност.
4. Ясно дефинирани отговорности на модулите в модела.
5. Централизиране на комуникацията между слоевете.
6. Лесна инсталация на модулите.
7. Независимо управление на данни.
8. Динамично управление на ресурси и потребители.
9. Отдалечена комуникация между отделните модули и други системи.
10. Универсално управление на данни.

11. Управление и автоматизация на работни/бизнес процеси.
12. Независимост от технологиите – моделът е технологично независим.
13. Независимост от схемата на базата данни.
14. Моделът да е базиран на утвърдени и доказали качествата си шаблони за дизайн и работни рамки.
15. Лесно разполагане (Deploy) на хост система.
16. Много разработчици и използване на гъвкави методологии (Agile Software Development, Extrime Programming).

Описание на модела

Многослойна приложна архитектура (MVC)



Фигура 3 Model View Controller за едно уеб приложение

Моделът използва утвърден в практиката шаблон MVC (Фигура 3). Той позволява независима работа на слоевете и лесната им замяна. Всеки един от тях може да се разглежда като черна кутия. Този модел предоставя лесна промяна на потребителския интерфейс. Слоят с модела на предметната област (Domain Model) се преизползва от всички заинтересувани слоеве в приложението. Бизнес логиката се изнася в слой с услуги, които също се преизползват. Комуникацията между тези логически слоеве може да се реализира на базата на интерфейси като се гарантира слабата свързаност (Loose Coupling) и силна кохезия (Strong Cohesion).

Отвореност за разширяване и затвореност за промяна

Модулите изградени по модела могат да съществуват отделно един от друг. Единствената им зависимост е от услугата ”Event Manager” или слоя за комуникация. Това превръща системата, изградена по този начин, в отворена за добавяне на нови модули. Всеки модул може да бъде заменен с подобен, който слуша за същите събития в системата, но е в състояние да реагира по различен начин. Можем да направим извод, че за постигане на отвореност на системата трябва да са налице следните възможности:

- Добавяне на модули с нова функционалност без това да налага каквито и да било промени в съществуващите модули.
- Разширяване на съществуващи модули без тяхната промяна – като разширяването може да включва всеки един от слоевете MVC.
- Модулът трябва да позволява промяна на своя презентационен слой, без да се налага да се правят промени в самия модул.
- Добавяне или разширяване на елементите в слоя модел на предметната област без промяна на оригиналните елементи в модула.
- Промяна в бизнес логиката чрез разширяване функционалността в слоя с услуги без това да налага промени в оригиналния модул. Слоят с бизнес логика съдържа описанието на предметната област и бизнес процесите.

Гъвкавост

Моделът предлага всеки модул да позволява конфигурационните му параметри да бъдат променяни от модули, които са заявени по-късно в конфигурационния файл на приложението. Хост системата разбира за модула по конфигурационния файл. Това дава възможност всеки от модулите да бъде разширяван с допълнителна функционалност като код от него се преизползва. Например може да се замени презентационния слой на някой модул като просто се създаде нов модул, съдържащ новия презентационен слой и после се добави в приложението след оригиналния модул. За да се постигне това мениджърът на модулите на работната рамка трябва да позволява тази функционалност: гъвкавостта е обикновено следствие от отвореността на системата [23].

Разпределеност и хетерогенност

Моделът предлага възможност отделните модули да са разпределени в уеб пространството. Всеки модул е възможно да е изграден с различна технология и разположен на различна отдалечена компютърна система. Комуникацията се осъществява посредством уеб услуги или сокети.

Ясно дефинирани функционалности на модулите в модела

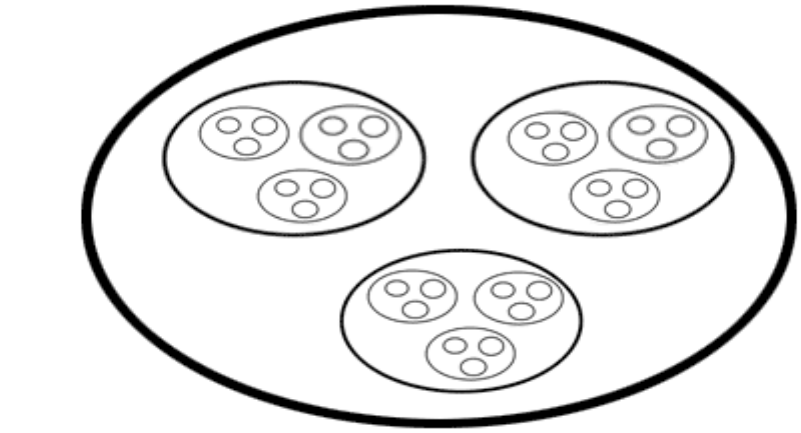
Преизползването на код е важен аспект в създавания модел. То позволява намаляване на ресурсите, необходими за създаване на системата. Видовете УБСС, които влизат в обхвата на изследването, имат общи функционалности, които определят и видовете модули, от които се нуждае моделът, а именно:

- Модул за идентификация.
- Модул за оторизация.
- Модул за навигация.
- Модул за управление на съдържанието.

- Модул за управление на процеси.
- Модул за управление на сесиите.
- Модул за локализация и превод.
- Модул за кеширане.
- Модул за управление на файлове.

Централизиране на комуникацията между слоевете

Софтуерните системи са сложни с много взаимовръзки. Изграждането на такава система все още е изкуство. Има различни шаблони за дизайн, и практики, които непрекъснато се увеличават. Те набелязват само някаква насока за това как архитектурата на системата може да изглежда. Изграждането на конкретна система е оставено в ръцете на разработчика [24-29].



Фигура 4 Безкрайност в двете посоки. Всяка система се състои от подобни на нея самата системи като границата между тях се определя от обща среда за разпространение на сигнали.

Подходът със специализиран слой за разпространение на сигнали се среща в някои софтуерни системи и рамки. Класът или групата класове, които осигуряват комуникационна среда се нарича обикновено ”Event Manager”. Други термини, които могат да се срещнат за този слой са “Connection, Environment, Events etc.”, просто за да се наблегне на факта, че този слой е предназначен за разпространение на съобщения. В дисертационния труд е използван терминът “Event Manager”, защото той е широко разпространен.

Моделът изисква задължително използване на среда за комуникация или “Event Manager” при изграждането на модулите и комуникацията им. Този подход позволява висша степен на независимост и модулност. Системата би продължила да работи, независимо от това, дали някой от модулите съществува или не. Разбира се, функционалността би се намалила при липса на модули, но това няма да попречи на останалите елементи да изпълняват функциите си. Класът за връзка имплементира няколко интерфейса, дефинирани в базовата работна рамка или използва Event Manager обект, предоставен от нея.

Лесна инсталация на модулите

Моделът предвижда инсталацията на модулите да се реализира с помощта на стандартния за технологията мениджър на пакети/зависимости. Което поставя още едно ограничение при избора на технология. Тя трябва да притежава стандартен или най-малкото широко приет мениджър на пакети/зависимости. Използването на стандартен за технологията

мениджър на зависимости разтоварва разработчиците на проекта от отговорността сами да изграждат подобна система. Делегирането на това задължение на друга група разработчици извън проекта, които се занимават с решаването само на този проблем, позволява концентриране на усилията за решаването на по-малък кръг проблеми. *Съществуването на мениджър на зависимостите за избраната технология е задължително*.

Абстрактна връзка с източниците на данни

За удовлетворяване на това изискване моделът предвижда използване на подходящи ORM работни рамки. За да се запази независимостта на модела от конкретната реализация и версия на ORM работната рамка, е нужно използването на подходящи похвати и шаблони за дизайн. Може да се постави адаптер [2] между приложението и ORM работната рамка – така то става независимо от нея. ORM работните рамки позволяват изолиране на модулите от специфичния източник на данни и абстрактна работа с него. Използването на ORM работна рамка, която представлява отделен проект, гарантира, че екипът, разработващ модулите, ще се концентрира тясно в решаване само на специфичните проблеми за модулите. С разработването, документирането и тестването на ORM работната рамка се занимава друг екип разработчици, които са тясно специализирани в решаването на този проблем, което предполага по-високо качество и надеждност на продукта.

Динамично управление на ресурси и потребители

За управлението на системните права е нужно да се използва механизъм, необвързан с конкретна схема от роли или права за достъп. Описанието на достъпа на потребители до определени ресурси може да стане едва след като системата е била разположена (Deploy) на даден хост. Тази възможност за отлагане на взимането на решения или за лесната им промяна е изключително предимство при управление на бизнес процеси. Използването на списък за контрол на достъпа (ACL), предлага възможно най-висока абстракция при работа с подобни системи, което от своя страна предполага възможност за прилагане на модела във възможно най-голям брой ситуации. Cau в [30] описва как политиката на права може да се формализира с помощта на синтезирана логика (Fusion Logic), темпорална логика (Temporal Logic) за специфициране поведението на системи. Авторът демонстрира с примери проверка за конфликти, динамично разпределение на задължения и налагане на политики за достъп до ресурси с помощта на прототипна реализация (FLCheck). Това изследване ни въоръжава с теоретични и приложни инструменти за работа с ACL.

Отдалечена комуникация между отделните модули и други системи

За да даде възможност за изграждане на уеб базирани софтуерни системи без наличие на централизирани сървъри (Serverless), моделът предвижда използване на уеб услуги за комуникация на модулите помежду си, както и за комуникация с други системи. Моделът предвижда комуникацията между модулите да се осъществява локално посредством интерфейси и споделени обекти и ресурси (EventManager), ако модулите са разположени на един и същи хост. За отдалечена комуникация между модулите, а така също и с други системи, трябва да се използват уеб услуги или сокети. Това позволява модулите да бъдат създадени с помощта на различни технологии.

Управление на данни от страна на крайния потребител

За да отговори на това изискване моделът предвижда създаването на два базови модула. Единият модул е система за управление на съдържанието (СУС) (CMS). Този модул се грижи за управлението на всякакъв вид съдържание, под формата на хипер текст (HTML).

Другият модул, предназначен да удовлетвори това изискване, е модул за управление на файлове (File Manager), способен да извършва всички действия, свързани с качване, сваляне, изтриване и др. на файлове. Съвместната работа на тези два модула задоволява огромен кръг изисквания за управление на всякакъв вид съдържание.

Възможност за управление и автоматизация на работни/бизнес процеси

За удовлетворяване на това изискване моделът предвижда изграждане на модул за управление и автоматизация на работни процеси, базиран на теорията на Петри. Разбира се, модулът може да бъде базиран на друг формален математически апарат.

Моделът да позволява изграждане на модулите с различни софтуерни технологии

Моделът позволява комуникация между модулите не само чрез интерфейси, но и посредством уеб услуги. Използването на уеб услуги за комуникация като допълнителна възможност, позволява на всеки модул да предлага микроуслуга и да е независим от останалите части на системата. Ако уеб услугите не задоволяват дадена функционалност, то моделът допуска използването на други протоколи и средства за комуникация, например сокети. Това позволява модулите да бъдат създадени на различни програмни езици. На практика предложеният модел позволява самите модули да съдържат произволен код. Той може дори да не е обектен. Допустимо е в модула да се затворят ресурси, които дори не са код като например изображения или код от друга технология, тоест моделът е приложим за различни технологии.

Независимост на модулите от схемата на базата данни

Макар ORM работните рамки да осигуряват независимост от слоя за запазване на данните, модулите са все още зависими от схемата на базата данни. Промяната в схемата на базата данни предизвиква верижна реакция от промени във всички софтуерни слоеве. Използването на бази данни без схема (NoSQL) премахва тези ограничения и разкрива нови хоризонти и възможности. Но тези бази данни идват със собствен набор от проблеми. Моделът трябва да предлага гъвкаво решение на тези нови проблеми, като комбинира предимствата на SQL и NoSQL базите данни чрез използване на NoSQL подходи в SQL бази данни.

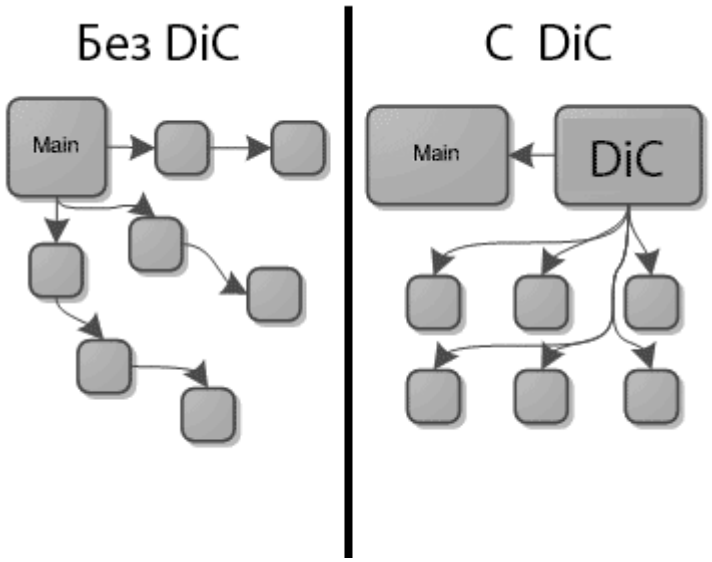
Моделът да е базиран на утвърдени и доказали качествата си шаблони за дизайн и работни рамки

Работните рамки дават стабилна база, документация и тестове. Налагат стандарти за кодиране и именуване, които улесняват работата на разработчиците и ускоряват разработката на модулите. Стандартите, налагани от работната рамка и от правилата на проекта, гарантират еднаквост и последователност в кода и в решенията на проблемите. За намаляване на зависимостите на модулите от работните рамки могат да бъдат използвани подходящи шаблони за дизайн, като например „Адаптер“ (Adapter) [5]. Модулите не само трябва да позволяват части от тях да се заменят, но и допълване на съществуващата функционалност. Това може да се реализира чрез използване на слой за услуги, а не чрез пренасяне на кода във входните контролери на работната рамка.

Ако модулите, които ще създаваме, няма да използват някоя от готовите работни рамки, ние трябва да създадем своя собствена такава [31] [23] и тя да предлага среда за разпространение на сигнали “Event Manager”. Изграждането на работна рамка или използването на такава е абсолютно задължително условие за изграждането и за

функционирането на модела. Без работна рамка работата е хаотична и обикновено се стига да ниво на сложност, което става непреодолимо на даден етап от разработката [32].

Fowler в [33] обяснява принципа за обръщане на контрола (Inversion of Control (IoC)), въвежда понятия и дава примери за него, описва шаблон за дизайн, наречен контейнер за инжектиране на зависимости (Dependancy Injection Container (DiC)) - Фигура 5. Създаването на обекти, от които други обекти зависят, става натоварващо за разработчика, затова създаването на обекти и инжектирането на техните зависимости се делегира на специалната система за инжектиране на зависимости (DiC). Наличието на имплементация на шаблона за дизайн "DiC" в работната рамка за уеб приложения е предимство. В някои технологии този шаблон за дизайн е реализиран като отделна работна рамка.



Фигура 5 Използване на принципа IoC с помощта на Dependancy Injection Container

При избора на работна рамка наличието на мениджър на модулите (Module Manager) е предимство. Този мениджър може да се грижи за това да проверява за модули в дадена папка или в конфигурационен файл, след което да настройва модулите за съвместна работа и да ги инстанциира. Той трябва да е достатъчно гъвкав и интелигентен, за да може например да избира измежду противоречиви конфигурационни параметри в отделните модули. Една възможна стратегия е да се позволи на конфигурационни параметри на модули, които се зареждат по-късно, да пренапишат конфигурация, обявена по-рано.

Лесно разполагане (Deploy) на хост система

За да отговори на това изискване, моделът предвижда, за изграждането на модулите да се избере технология, която може да работи на колкото се може по-голям брой операционни системи, уеб сървъри и сървъри за бази данни и може да използва както специализирани, така и споделени сървъри. Трябва да се избере някоя от най-широко разпространените и използвани технологии за уеб системи, технология, която се предлага често от хостинг компаниите, което снижава цената и улеснява разполагането на модулите. Избраната технология трябва да позволява разнообразно комбиниране на операционна система, уеб сървър и сървър за бази данни.

Много разработчици и използване на гъвкави методологии (Agile Software Development, Extrime Programming)

Моделът предвижда едновременна работа на много разработчици над отделните модули. Това изискване може да се постигне чрез използване на гъвкавите методологии за изграждане на софтуер [34]. От архитектурна гледна точка това се постига чрез използване на интерфейси, разделяне на системата, използвайки принципите на обектно-ориентираното, аспектно-ориентираното и модулното програмиране. Използването на работни рамки като основа на изградените модули е ключов момент при удовлетворяване на изискването за работа на много разработчици. Наличието на документация и на тестове за работните рамки е задължително условие за ефективна, бърза и едновременна работа на много разработчици и екипи. Създаването на документация и тестове за самите модули също е от съществено значение.

За изграждането на модулите е добре да се избере технология с отворен код и с голямо и отзивчиво общество от разработчици. Езикът за програмиране, който избраната технология предлага, трябва да е лесен за усвояване и за употреба. Това би позволило включването в проекта и на разработчици без голям опит и умения, например студенти.

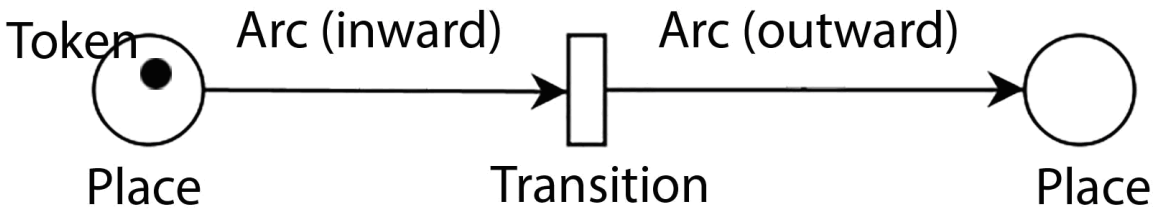
Модул за работни процеси, базиран на мрежи на Петри

В тази подточка се описва подробно един от множеството модули за бързо изграждане на уеб базирани софтуерни системи, който използва математическия апарат за моделиране, наречен мрежи на Петри (Petri Nets) [35] [36] [37]. Неговата цел е да ускори процесът на проектиране, създаване и тестване на такива системи. Модулът допълва останалите модули като добавя допълнителен слой на абстракция и ускорява процеса по създаване на приложение чрез въвеждане на математически формализъм. Той помага при решаване на широк кръг проблеми, които могат да бъдат описани с помощта на мрежи на Петри. Модулът за автоматизация на работни процеси, базиран на мрежи на Петри, може да се ползва като универсален инструмент за решаване на огромен кръг задачи. Например работен процес или дори софтуерна програма могат да бъдат описани с математическия апарат на Петри и имплементирани директно почти без нужда от писане на програмен код. Ако даден проблем може да се опише с помощта на математическия инструментариум на мрежите на Петри, то описанието му може да служи за вход на модула.

Ще опишем използвания в модула математически формализъм. Той включва обектите, които могат да се използват, правилата на които се подчиняват тези обекти, тригерите за преходи, маршрутизацията и начина, по който се реализират разделянето и обединяването.

Обекти в мрежите на Петри

- Позиции (Places) – те са пасивни и могат да играят ролята на кутии за получаване на съобщения (Inboxes) в една офис система. Представят се с кръгове в диаграмата на мрежите на Петри. Всяка мрежа на Петри има една входна и една изходна позиция, но неограничен брой междинни позиции.
- Преходи (Transitions) – те са активни и представляват задачите, които трябва да се изпълнят. Представят се с правоъгълници в диаграмата.
- Дъги (Arcs) – всяка дъга свързва една позиция с един преход. Представят се със стрелки в диаграмата. „Входяща дъга“ излиза от позиция и влиза в преход. „Изходяща дъга“ излиза от преход и влиза в позиция.
- Ядра/Пулове/Жетони (Tokens) – те представляват текущото състояние на даден работен процес. Представят се като черни точки в позициите на диаграмата. Една позиция може да съдържа 0 или повече ядра в даден момент от времето.



Фигура 6 Обекти в мрежите на Петри

Правила за обектите в мрежите на Петри

- Позициите не правят нищо друго, освен да съдържат ядра, които представят състоянието на процеса. Позицията може да съдържа 0 или повече ядра в даден момент.
- Една дъга свързва една позиция с един преход.
    - Позиция P се нарича входна за преход T, ако съществува директна дъга от P към T.
    - Позиция P се нарича изходна за преход T, ако съществува директна дъга от T към P.
- Когато един разрешен преход се изпълни, то той премества ядра от всичките си входни позиции до всичките си изходни позиции.
    - Преход T се казва, че е разрешен, ако всяка входна позиция P за T съдържа поне едно свободно ядро.
    - Как един разрешен преход се изпълнява, зависи от типа на тригера.
    - Когато преход T се изпълни, консумира по едно ядро от всяка входна позиция P за T и произвежда по едно ядро във всяка изходна позиция P за T.
- Всеки работен процес има само една позиция за стартиране. Трябва да има поне една входяща дъга, влизаща в преход. Може да има една изходяща дъга, идваща от преход с цел рестартиране на процеса.
- Всеки работен процес има само една крайна позиция. Трябва да има поне една изходяща дъга, идваща от преход (може да има повече от една). Но не може да има никакви входящи дъги, влизащи в преход.

Тригери в мрежите на Петри

Моментът, в който един преход се разрешава, и моментът, в който се изпълнява, обикновено не съвпадат. Нещото, което кара един разрешен преход да се изпълни, наричаме тригер. Различаваме четири вида тригери:

- Автоматичен (Automatic) – преходът се изпълнява незабавно, щом се разреши.
- Потребител (User) – преходът се изпълнява в резултат от човешка намеса, в една система за управление на процеси всеки потребител има входяща кутия, в нея попадат задачите, които трябва да се изпълняват (Workitems).
- Време (Time) – един разрешен преход може да се изпълни в резултат от изтичане на определен интервал от време.
- Съобщение (Message) – външно събитие, включващо изпълнението на прехода. Като пример може да се посочи телефонно обаждане, изпращане на писмо и т.н.

NoSQL подходи в SQL бази от данни

С развитието на облачните технологии ролята на така наречените NoSQL бази от данни нараства все повече. Това се дължи на удобствата, които предлагат при работа в реално време в уеб среда с големи обеми от данни (Big Data). Те предлагат независимост на приложението от схемата на базата данни, което е едно от поставените пред модела изисквания. Разработчиците, се опитват да изградят приложенията така, че промяната в един елемент да не предизвиква верижна реакция от промени в други части на софтуера. Използването на ORM техниките също способства вълните на промените да бъдат ограничени до определени слоеве. Промяната в схемата на една релационна база от данни води до промяна на всички слоеве над нея, а обикновено базата с данни е в ядрото на приложението, което значи, че то цялото ще претърпи промени. Липсата на схема в една NoSQL база от данни позволява вълната на промяната да бъде ограничена. Но NoSQL базите от данни не поддържат транзакции и не следват ACID принципите, което ограничава техните възможности. От друга страна една релационна база от данни поддържа транзакции и следва ACID принципите, но се нуждае от ясни изисквания и предварително заложена схема, което не позволява гъвкавост при промени. В дисертационния труд се описват подходи, чрез които някои от предимствата на NoSQL базите данни могат да се симулират в релационна база от данни, което позволява оптимизирането на приложения, които са изградени над такъв тип бази от данни. Подходите включват използване на таблици само с полета ключ и стойност, таблици с полета идентификатор, ключ и стойност, таблици с полета идентификатор, външен ключ, ключ и стойност, а така също добавяне на полета с общо предназначение и добавяне на таблици подмножества.

Интеграция на модулите с основната хост система

Интеграцията на модул с основната хост система се осъществява само чрез една зависимост - EventManager. Това е ключов момент. Целият модел почива на тази идея – намаляване на зависимостите до краен предел. Модулът познава от главната система само и единствено средата за комуникация, предоставена му по време на присъединяване към системата. За създаването на самия модул се използва контейнер за инжектиране на зависимости (DiC). Мениджърът на модули (Module Manager) е натоварен със задължението да свърже модула с главната система. За наличието на модула се научава от конфигурационния файл.

Включването към системата се изразява в инжектирането на обект от клас ”EventManager” в модула. Този обект от клас ”EventManager” трябва да е съобразен с нивото, на което се включва модулът.

Взаимодействието със системата се осъществява само чрез инжектирания в модула обект от клас ”EventManager”. Модулът трябва да „слуша“ за събитията, които получава през комуникационната среда и, ако има достатъчно знание за тях, трябва да реагира адекватно. Самият модул трябва да излъчва в средата за връзка събития като информира системата (останалите модули) за това какво се случва в него. Ако има заинтересувани, то те ще реагират адекватно. Ако няма подслушвачи за излъчваните събития, просто нищо няма да се случи (”Loos Coupling”, ”Strong Cohesion” Principles).

Начини за разширяване функционалността на модули

Макар базовата функционалност на модулите да е създадена по такъв начин, че да задоволи основните потребности на огромен кръг потребители, може да се наложи нейната промяна или разширяване. Два са основните подходи при решаване на този проблем.

Разширяване чрез създаване на разширяващ модул

Ако някой от логическите слоеве на модулите трябва да бъде променен или да бъде разширена неговата функционалност, то е възможно създаването на нов разширяващ модул. В разширяващия модул се дефинират само новите елементи за разширяване или промяна, които или изцяло заместват съществуващите в оригинала, или преизползват и разширяват оригинала. Съществено при този подход е, че в една система, изградена с помощта на модулите, трябва да съществуват и двата модула. Базовият модул трябва да се зареди преди разширяващия и в последствие да се зареди разширяващият. За да може моделът да функционира правилно, технологията, с която са изградени и се зареждат модулите трябва да позволява последователно зареждане като редът на зареждане има съществено значение.

Разширяване чрез клониране и отклоняване от базата

Може да съществуват ситуации, при които нужната функционалност е толкова различна и нестандартна, че почти нищо или нищо от базовата не може да се преизползва. Дори и в такива ситуации моделът предвижда възможно решение. В подобни ситуации модулът може да бъде използван като база за преизползване на код и вече изградена структура, тоест отказваме се от преизползването на функционалност, но преизползваме код и структура от базовия модул. Това се реализира чрез клониране на базовия код (Code Base) и създаване на нов модул, който няма функционална зависимост от съществуващ модул. Съществено при този подход е, че в една система, изградена с помощта на модулите, трябва да съществува само новосъздаденият модул. Базовият модул не е нужно да се зарежда или ако се зареди, то той трябва да бъде изцяло заместен от новосъздадения модул, евентуално преизползващ само кода му чрез клониране.

3. Реализация на модела – прототип, приложение, експерименти

За изграждането на прототип по описания в 2 модел първо трябва да се намери подходяща технология. Най-широко разпространената технология за изграждане на уеб базирани информационни системи е PHP [38]. Други възможни кандидати за изграждане на прототип са C# .NET и Java. Те отстъпват на PHP по разпространение за изграждане на уеб системи и възможности за разполагането (Deploy) на хостове. По-долу са изброени причините за избора на технологията PHP. Тя е предназначена специално за изграждане на уеб базирани системи, с отворен код е, има голямо и отзивчиво общество от разработчици. Тази технология разполага с широко разпространен мениджър на зависимостите и множество работни рамки за изграждане на уеб базирани приложения, ORM работни рамки, както и работни рамки за създаване на уеб услуги. Голяма част от тези проекти са с отворен код. Разполагането на хост система е изключително лесно, евтино и достъпно. Могат да се използват както споделени, така и специализирани хостове. PHP работи на голям брой операционни системи, уеб сървъри и сървъри за бази данни. Езикът за програмиране PHP е изключително лесен за усвояване и работа. Това би позволило дори разработчици без много опит и умения, дори и студенти, да се включат ползотворно в проекта. Това са характеристики, които отговарят на изискванията на модела, описан в 2. Един възможен недостатък е свободата, която PHP дава на разработчиците

да реализират една и съща функционалност по много различни начини. За да се избегне този недостатък е нужно да се използват работни рамки, стандарти за кодиране и именуване, документация, тестове, гъвкави методологии, които да наложат строга дисциплина и последователност в проекта. Поради голямата свобода има огромно разнообразие от работни рамки. Езикът PHP не е типизиран (Loosly Typed, Dynamic Language), което не позволява улавянето на грешки още при писане (Type Save), но той използва интерпретатор, което позволява тестване на приложението без нужда от компилация и изграждане (Build).

След анализа на работните рамки в технологията PHP, заключението е, че работната рамка Zend отговаря най-пълно на изискванията на модела, описан в 2. Тази работна рамка взаимства най-много и стои най-близо до идеите, шаблоните за дизайн и структурата на лидерите в това отношение C# ASP.NET MVC и Java Spring MVC. Приликите на Zend с работните рамки ASP.NET и Spring MVC биха позволили лесно адаптиране на разработчици, идващи от тези технологии, към нуждите на проекта за изграждане на модули. Използването на Zend е съчетание на качества на технологията PHP с качества и предимства на технологиите C# и Java. Съчетават се лекотата на използване, разполагане и широко разпространение на PHP с идеите, шаблоните за дизайн, добрите практики и т.н. идващи от типизирани езици като C# и Java.

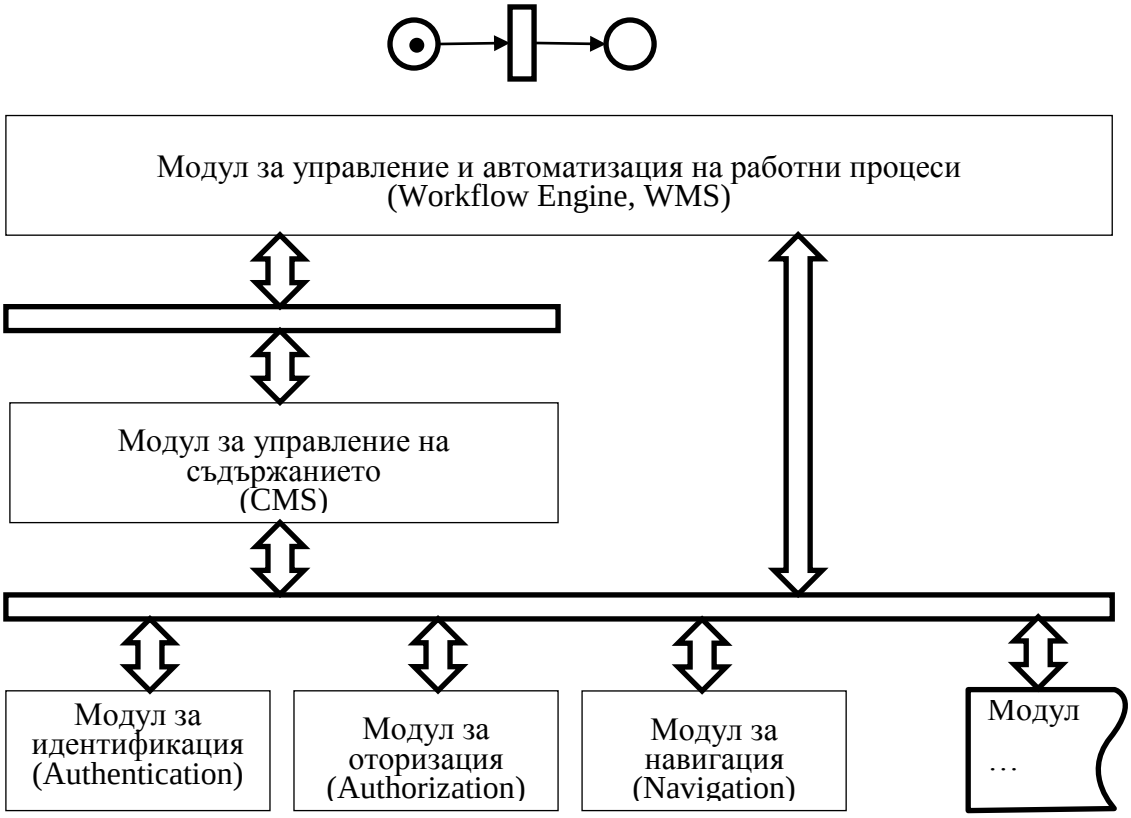
Zend всъщност е библиотека от компоненти (Component Library), а не работна рамка. Компонентите се използват при желание (At Will). Те са много слабо свързани и могат да се използват самостоятелно. Това, което превръща Zend в работна рамка, е компонентът MVC, който единствен от компонентите налага структура на проекта и прилага принципа обръщане на контрола (IoC). Ако се използва MVC компонента, то лесно и бързо може да се изгради приложение с ясно дефинирана структура.

Zend притежава като отделни компоненти мениджър на събития (Event Manager), мениджър на модули (Module Manager), контейнер за инжектиране на зависимости (DiC), както и друга имплементация на принципа обръщане на контрола (IoC), а именно локатор на услуги (Service Locator). Тези компоненти отговарят напълно на изискванията на модела, описан в 2.

Изискването на модела за използване на ORM работна рамка се удовлетворява напълно от работната рамка Doctrine в избраната по-горе технология PHP. Doctrine отново е взаимствала идеи, идващи от работните рамки Hibernate и Entity Framework, съответно в технологиите Java и C#. Тя съчетава голямата популярност, лекота за използване, разработка и разполагане на PHP с идеите, структурата, добрите практики, шаблоните за дизайн, използвани в C# и Java. Работната рамка Doctrine е идеално интегрирана за съвместна работа със Zend и Apigility.

Изискването на модела за използване на работна рамка за уеб услуги в технологията PHP се удовлетворява от работната рамка Apigility. Тя е естествен избор, понеже използва много от компонентите, създадени в работната рамка Zend. Apigility позволява на модулите да предлагат функционалност чрез уеб услуги.

В модулите се използва добре утвърдената трислойна архитектура MVC, но моделът не ограничава броя на слоевете до три – при нужда могат да се добавят още логически слоеве.



Фигура 7 Архитектура на модулна система

Базовите модули в архитектурата на модулната система Фигура 7 са:

- Модул за идентификация.
- Модул за оторизация.
- Модул за навигация.
- Модул за управление на съдържанието.
- Модул за управление и автоматизация на процеси.

В прототипа са включени още няколко незадължителни модула, които подобряват и разширяват техническата функционалност на системата. Това са модулите за локализация и превод, управление на сесиите, кеширане и управление на файлове.

Архитектура

Архитектурата на всички модули в модулната система има сходни характеристики. Всички базови модули обикновено имат абстрактно ядро, конфигурация, слой за достъп до данните, презентационен слой, уеб услуги, бизнес логика и др. Общите характеристики ще бъдат разгледани в настоящата подточка.

Абстрактно ядро (логически слой с услуги)

В реализирания прототип, ядрото на модула е изолирано в слой с услуги (Service Layer). Входният контролер познава и използва само интерфейса на услугата. По този начин

всички аспекти от функционирането на модула могат да бъдат променяни независимо. Промяната може да се реализира, както беше описано в модела, от по-късно зареждащи се модули, които заменят или допълват даден логически слой. Абстрактното ядро се състои от набор интерфейси, с които класовете в слоя с услуги сключват договор. Това ядро се заявява като налична функционалност на всички потребители. В случая, потребители основно са входни контролери от слоевете за презентация. По принцип ядрото може да се изолира в отделен логически слой и пакет, а не да се намира в слоя с услуги. Така се реализира шаблонът за дизайн разделен интерфейс (Separated Interface) [5]. В прототипа интерфейсите са оставени в слоя с услуги.

Конфигурация

За наличието на модула, системата разбира от декларация в конфигурационния файл. Възможен е друг подход с проверка за наличие на модул в определена папка или динамично зареждане. Вторият подход не е използван в прототипа поради факта, че при всяка заявка към сървъра трябва да се преминава през процес на сканиране на директории за наличие на определени файлове, които разкриват наличието на модула. Много по-опростен е подходът с явно заявяване на модула. Този подход позволява много по-висока скорост на изпълнение, но натоварва разработчиците със задължението явно да декларират наличието на модула в конфигурационния файл. Конфигурационният файл използва обикновен PHP асоциативен масив. Този подход не изисква допълнително преобразуване на XML, ini, YAML и т.н. формати към нативна PHP структура данни, но не позволява съвместимост с други технологии. За да се постигне съвместимост и при реализация с .NET или Java трябва да се придържаме към използване на XML, ini или YAML.

Слой за достъп до данните (Data Access Layer)

За комуникация със слоя за запазване на данните е използвана ORM работната рамка Doctrine. В прототипа не е използван адаптер за разделяне на ORM работната рамка от бизнес логиката. Това опростява реализацията, но прави слоя зависим от конкретната ORM работна рамка. Doctrine позволява комуникация с NoSQL бази данни, което напълно удовлетворява изискванията на модела. Достъп до мениджър на единици (Entity Manager) се получава през единствената зависимост на модула – мениджър на събития (Event Manager) като моделът на предметната област (Domain Model) остава пакетиран в модула.

Презентационен слой

Презентационният слой включва презентационните елементи, които в случая на уеб приложение се създават с помощта на HTML, CSS и JavaScript. Този код е капсулиран в отделни файлове според тяхното предназначение (Separateion of Concern). В случая с използването на работната рамка Zend, за всяко входно действие от страна на клиента е създаден скрипт за изглед (View Script). Поради факта, че трябва да има разделение на отговорностите (Separation of Concern), методите (действията) на входните контролери предават към скриптовете само данни. Тези данни се интерпретират в скрипта за изглед с помощта на някакъв език (Template Language). В прототипа е използван PHP като език за интерпретация на данните и изграждане на потребителския интерфейс. В отделни файлове се съхраняват HTML с PHP, CSS и JavaScript. За изграждането на цялостния завършен интерфейс, се използва шаблонът за дизайн, „двустъпков изглед“ (Two Steps View) [5]. Този шаблон, както подсказва името му, предполага, че сглобяването на финалния HTML, CSS и JavaScript код става на две стъпки. Първата стъпка е създаване на изгледа за конкретното действие, предприето от потребителя. Втората е съвместяването на кода, генериран от първата

стъпка с друг, оформящ постоянните елементи на интерфейса. Този подход позволява създаване на последователен интерфейс, централизира и намалява работата върху него.

Уеб услуги

Презентационният слой може да представлява, мобилно приложение, SPA (Single Page Application) или друг вид приложение. Тогава комуникацията с приложението се реализира чрез уеб услуги (Web Services). За реализацията на уеб услуги в прототипа е използвана работната рамка Apigility. Тя позволява бизнес логиката, обикновено затворена в логическия слой за услуги (Service Layer), да бъде лесно и бързо предложена за използване под формата на уеб услуги. Тези услуги обикновено са RESTful. Възможно е предлагане на услуги под формата на RPC (Remote Procedure Call) евентуално с използване на WSDL (Web Services Descripton Language) и XML или JSON. Възможно е използването на потребителски формати за комуникация, които евентуално повишават бързодействието, но намаляват съвместимостта с други системи. Моделът не изключва възможността от използване при нужда на други протоколи за връзка освен HTTP. В такава ситуация обикновено се използват сокети (Sokets), които позволяват лесна, бърза комуникация и много възможности, но отново намаляват съвместимостта. Прототипът в текущия си вариант не предлага слой за подобна комуникация, но той лесно и бързо може да бъде изграден благодарение на предложената в модела гъвкава архитектура.

Бизнес логика

Бизнес логиката в прототипа е изнесена в отделен слой с услуги (Service Layer). Тя имплементира интерфейсите от абстрактното ядро. Този подход позволява преизползването ѝ в различни други слоеве. Слоевете, в които обикновено се използва, са входните контролери за предлагане на функционалност под формата на HTML, уеб услуги и др. Бизнес логиката на всеки модул лесно може да бъде разширена или изцяло заменена чрез използване на разширяващ модул, описан в модела. Това позволява изключителна гъвкавост и свобода при употребата на модулите.

Начини за инсталация и употреба

Всеки модул се създава и пакетира по правилата на мениджъра за пакети Composer. Този мениджър е най-популярния в технологията PHP. Именно затова е използван в прототипа. Мениджърът позволява пакетиране на ресурси от всякакъв вид. Това може да са както файлове с код, така също и файлове с графична и всякакъв друг вид информация. Единственото изискване е наличието на JSON файл, съдържащ мета данни, описващи съдържанието и зависимостите на пакета от други пакети. Пакетите могат да са разположени в публични или в частни хранилища за код (Repositories). Composer следи за версиите и ревизиите. Избраният мениджър на пакети отговаря напълно на изискванията на модела и позволява изключителна свобода и гъвкавост. Начините за инсталация на пакетите, съдържащи модули, са два:

- Ръчно.
- Автоматично.

Ръчната инсталация не е за препоръчване, понеже за зареждането, следенето и управлението на зависимостите трябва да се грижи разработчик. При ръчната инсталация не се използват инструментите, предоставени от мениджъра на зависимости Composer. Подобен вид инсталация обикновено предполага желание за разширяване на функционалността чрез промяна на кода на модулите. Така на практика се създават нови модули за конкретния

проект. Разположението на модулите във файловата структура на проекта е произволно в зависимост от нуждите.

Автоматичната инсталация е нормалният начин за инсталация и употреба на модулите. Composer поставя пакетите в специализирана папка на проекта и се грижи за всички зависимости и за обновяванията автоматично.

Както при ръчната, така и при автоматичната инсталация, се изисква наличието на модула да бъде заявено в конфигурационния файл на приложението. Всички други специфични конфигурационни параметри се задават в допълнителни конфигурационни файлове.

Разширяване

В прототипа са спазени изискванията на модела и е реализиран подход, разрешаващ два вида разширяване:

- Разширяване с отклоняване от базовия код.
- Разширяване чрез използване на разширяващи модули.

При разширяване с отклоняване от базовия код се предполага, че ще се наруши принципът за затвореност на модула и той ще бъде вътрешно променен. Както беше посочено вече, инсталацията допуска ръчен и автоматичен подход. При ръчната инсталация се допуска промяна на самия код на модула. Този вид разширяване е с отклоняване от базовия код (Code Base).

При разширяването с надграждане или със замяна на функционалност в разширяващи модули се създават локално за проекта модули, които заменят или разширяват функционалност вече предложена в автоматично инсталирания модул.

Модули

В прототипа на модулната система са включени следните видове модули:

- Модул за идентификация (Authentication)
- Модул за оторизация (Authorization)
- Модул за навигация (Navigation)
- Модул за управление на съдържанието
- Модул за управление на сесиите
- Модул за локализация и превод
- Модул за кеширане
- Модул за управление на файлове (File Manager)
- Модул за управление на процеси (система за управление и автоматизация на работни процеси)

Софтуерна методология за изграждане на прототипа и създаване на проект с отворен код

Изключително важен момент при изграждането на прототипа, както и на всеки софтуерен проект е създаване на екип от разработчици и управлението им за довеждането му

до успешен край. Приложимостта на модела в условията на разработване на софтуер с отворен код също представлява интерес.

Приложения и експерименти с прототипа

В тази точка са разгледани приложенията на модулите за бързо изграждане на УБСС (RADWIS Modules). Те демонстрират приложимостта на разработения модел, както и неговите предимства, позволяват с прототипа на модела да се проведат експерименти по изграждане и тестване на различни УБСС. Използваната софтуерна методология (Agile Software Development и Extreme Programming) проповядва идеята за постигане на целта по най-прагматичен и директен начин, а качественият код сам по себе си служи като документация. *Изграденият по модела прототип на модулна система (модулите) е инструмент за изграждане на УБСС. Функциониращ ясен и структуриран код е на разположение и доказва правотата на идеите в модела. Самите модули и кодът в тях са на разположение, като проекти с отворен код за тези, които искат да ползват функционалността или разгледат реализацията.*

Заключение

Главният извод, който можем да направим е, че поставените цели в настоящия дисертационния труд са постигнати. Определени са изискванията към модела за изграждане на модули. Създадени са модули по представения модел, доказващи неговата приложимост. Изградени са УБСС с използването на описаните модули.

Основните приноси на дисертационния труд са от научно-приложен и приложен характер:

1. Подробно описание на концептуален модел на модулна система от ново ниво, разположена между работните рамки и платформите.
2. Създаване на прототип на модулна система по описания модел.
3. Изграждане на реални експериментални системи с модулите.
4. Изграждане на модул, базиран на теорията на мрежи на Петри, за решаване на широк кръг проблеми. Това води до повишаване на абстракцията на системата.
5. Откриване и използване на NoSQL подходи в SQL бази данни.

Връзките между приносите, целите, задачите, мястото на описание в дисертационния труд и публикациите са описани в следната таблица:

Принос	Цел	Задачи	Секция	Публикация
1	2	2.1, 2.2	2.1, 2.2	1,4,5
2	3	3.1, 3.2	3.1 – 3.8	1,4,5
3	4	4	3.12	1,4,5
4	2	2.3	1.1.7, 11, 2.2.11, 2.3, 3.10	2
5	2	2.4	13, 2.2.13, 2.4	3

Перспективи за развитие

По описания модел могат да бъдат създадени модули чрез други технологии, като проекти с отворен код. Така хора от цял свят могат да се включат в изграждането им.

Към модела, освен описаните модули, могат да бъдат добавени още други такива, които допълнително да обогатят предлаганите функционалности: по-този начин се увеличава обхватът на научното изследване.

Обхватът на приложение би могъл да се разшири:

- Като вместо математическата теория на мрежи на Петри, се използва друг математически формализъм, на чиято база да се изгради модул за управление на работни процеси. Например – посредством теорията на обобщените мрежи, която е разширение на мрежите на Петри.
- Чрез използване на интервална темпорална логика [39] като математически формализъм за описание на работни процеси. Комбинирането на теорията на мрежи на Петри и интервална темпорална логика е възможно и многообещаващо като подход.

Друга научна работа би могла да бъде изучаването на ефекта от изграждане на модулите като проекти с отворен код. Биха могли да се използват строги научни критерии като интервюта, допитвания, анкети за оценка на използването на модулите от различни екипи в различни проекти. Интерес би представлявало и участието на много разработчици в създаването и поддръжката на модулите.

Апробация

Части от дисертационния труд са изготвени в изпълнение на задачи по следните проекти:

1. Научен проект НИ13 ФМИ-002 (2013-2014) към фонд "Научни изследвания" при ПУ: ", тема: „Интеграция на ИТ в научните изследвания по математика, информатика и педагогика на обучението” – участник.
2. Проект НИ15-ФМИ-004 (2015-2016) към фонд "Научни изследвания" при ПУ: "Иновативни фундаментални и приложни научни изследвания по компютърни науки, математика и педагогика на обучението" – участник.
3. Проект ИТ15-ФМИИТ-004 (2015-2016) към фонд "Научни изследвания" при ПУ: "Изследвания в областта на иновационни ИКТ с ориентация към бизнеса и обучението" – участник.

Части от дисертационния труд са докладвани на следните научни сесии и конференции:

1. *Distributed Cloud Based System for Active/E-Learning*, International Conference "FROM DELC TO VELSPACE", Plovdiv, 26–28 March 2014.
2. *Модул за работни процеси, базиран на мрежа на Петри*, научна конференция на ФМИ на ПУ "Иновационни ИКТ: Изследвания, разработка и приложения в бизнеса и обучението", гр. Хисар, 11-12 ноември 2015 г.
3. *NoSQL подходи в SQL бази от данни*, научна конференция на ФМИ на ПУ „Иновационни ИКТ в бизнеса и обучението: Тенденции, приложения и разработване“, Пампорово, 24-25 ноември, 2016 г.

По описания модел е създаден и внедрен софтуер за нуждите на ПУ:

1. Разработка на уеб базирана информационна система за обслужване на отдел „Развитие на академичния състав” към ПУ „Паисий Хилендарски”.

Публикации по темата на дисертационния труд:

1. Cheresharov, St., Hr. Krushkov, *Distributed Cloud Based System for Active/E-Learning*, Proceedings of the International Conference From DeLC to VelSpace, Plovdiv, 26–28 March 2014, pp. 327-334, Third Millennium Media Publications, ISBN: 0-9545660-2-5.
2. Черешаров, Ст., Хр. Крушков, *Модул за работни процеси, базиран на мрежа на Петри*, Сборник с доклади на научна конференция "Иновационни ИКТ: Изследвания, разработка и приложения в бизнеса и обучението", гр. Хисар, 11-12 ноември 2015 г., стр. 73, ISBN: 978-954-8852-56-7.
3. Черешаров, Ст., Хр. Крушков, *NoSQL подходи в SQL бази от данни*, Сборник с доклади на научна конференция „Иновационни ИКТ в бизнеса и обучението: Тенденции, приложения и разработване“, Пампорово, 24-25 ноември, 2016 г., стр. 79, ISBN: 978-954-8852-72-2.
4. Cherecharov, St, Hr. Krushkov, M. Krushkova, *NLP module for Bulgarian text processing*, CBU International Conference Proceedings 5: INNOVATIONS IN SCIENCE AND EDUCATION, 2017, pp.1113-1117.
5. Cheresharov, St, Hr. Krushkov, St. Stoyanov, I. Popchev, *Modules for Rapid Application Development of Web-Based Information Systems*, Cybernetics and Information Technologies (CIT), Sofia, 2017, Volume 17, No 3, ISSN: 1314-4081.

Благодарности

Изказвам сърдечна благодарност на научния си ръководител проф. д-р Христо Крушков за проявената подкрепата по време на работата върху дисертацията на проф. д-р Станимир Стоянов за оказаните професионална помощ и на всички колеги за съветите, препоръките и съдействието.

Библиография

[1] J. Moffett, „Chapter 9 - Looking Toward the Horizon.,“ в *Bridging UX and Web Development*, Morgan Kaufmann, 2014, pp. 163-188.

[2] E. Gamma, R. Helm, R. Johnson и J. Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software, Chicago: AddisonWesley, 1994.

[3] R. C. Martin, Clean Code: A Handbook of Agile Software Craftsmanship, Prentice Hall, 2008.

[4] Microsoft, „Application Architecture Guide,“ 26 March 2015. [Онлайн]. Available: https://msdn.microsoft.com/en-us/library/ff650706.aspx.

[5] M. Fowler, D. Rice, M. Foemmel, E. Hieatt, R. Mee and R. Stafford, Patterns of Enterprise Application Architecture, Boston: Addison Wesley, 2002.

[6] E. Freeman, Head First Design Patterns, O’Reilly, 2004.

[7] D. Schmidt, „Applying Patterns and Frameworks to Develop Object-Oriented Communication

Software,“ *Handbook of Programming Languages,* том 1, 1997.

[8] R. Johnson и B. Foote, „Designing Reusable Classes,“ *Journal of Object-Oriented Programming,* том 1, p. 22–35, 6/7 1988.

[9] S. Murugesan, Y. Deshpande, S. Hansen и A. Ginige, „Web Engineering: a New Discipline for Development of Web-Based Systems,“ *Lecture Notes in Computer Science,* том Volume 2016, pp. 3-13, 02 May 2001.

[10] C. McCormack и D. Jones, Building a Web-Based Education System, New York: John Wiley & Sons, 1997.

[11] P. De Bra, L. Aroyo и V. Chepegin, „The Next Big Thing: Adaptive Web-Based Systems,“ *Journal of Digital Information,* том Vol 5, № 1, 2004.

[12] „Workflow Management Coalition,“ 16 November 2015. [Онлайн]. Available: http://www.wfmc.org/.

[13] S. Ju, „Modular programming conventions in assembly languages,“ в *AFIPS '77 Proceedings of the June 13-16, 1977, national computer conference*, Dallas, 1977.

[14] K. Busbee, Programming Fundamentals - A Modular Structured Approach using C++, Houston: CONNE XIONS, 2008.

[15] T. Boudreau, J. Tulach и G. Wielenga, Rich Client Programming, Santa Clara: Prentice Hall, 2007.

[16] Bitreactive, „Reactive Blocks,“ 13 06 2017. [Онлайн]. Available: http://www.bitreactive.com/.

[17] A. Kraemera, „Block by Block Towards IoT Applications.,“ 13 06 2017. [Онлайн]. Available: http://reference.bitreactive.com/papers/bitreactive-towards-iot-applications.pdf.

[18] D. L. Parnas, „On the criteria to be used in decomposing systems into modules,“ *Communications of the ACM,* том 15, № 12, pp. 1053-1058, 12 1972.

[19] C. Zheng, Y. Yao, S. Huang и Z. Ren, „Modeling Workflow Systems Constrained by Inputs and Outputs − An Approach Based on Petri Nets,“ *Cybernetics and Information Technologies (CIT),* том 15, № 4, pp. 27-41, 2015.

[20] K. Atanassov, D. Dimitrov и V. Atanassova, „Algorithms for Tokens Transfer in Different Types of Intuitionistic Fuzzy Generalized Nets,“ *Cybernetics and Information Technologies (CIT),* том 10, № 4, pp. 22-35, 2010.

[21] B. Moszkowski и Z. Manna, „Reasoning in Interval Temporal Logic,“ в *ACM/NSF/ONR Workshop on Logics of Programs*, Stanford, 1983.

[22] B. Moszkowski, „An Automata-Theoretic Completeness Proof for Interval Temporal Logic,“ в *27th International Colloquium on Automata, Languages and Programming (ICALP 2000)*, Geneva, 2000.

[23] А. Пенев, *Отворени хибридни системи за геометрично моделиране,* Пловдив: Дисертационен труд, Пловдивски университет "Паисий Хилендарски", 2013.

[24] M. Souza и M. Ferreira, „Designing reusable rule-based architectures with design patterns,“ *Expert Systems with Applications,* том 23, № 4, pp. 395-403, 11 2002.

[25] B. Douglass, „Software Design Architecture and Patterns for Embedded Systems,“ в *Software Engineering for Embedded Systems*, 2013, pp. 93-122.

[26] . A. Ampatzoglou, A. Kritikos, G. Kakarontzas и I. Stamelos, „An empirical investigation on the reusability of design patterns and software packages,“ *Journal of Systems and Software*, № 84 (12), pp. 2265-2283, 2011.

[27] M. Paasivaara и C. Lassenius, „Communities of practice in a large distributed agile software development organization – Case Ericsson,“ *Information and Software Technology*, том 56 (12), pp. 1556-1577 , 2014.

[28] X. Yu и S. Petter, „Understanding agile software development practices using shared mental models theory,“ *Information and Software Technology*, том 56 (8), pp. 911-921, 2014.

[29] D. Kirk и E. Tempero, „A lightweight framework for describing software practices,“ *Journal of Systems and Software*, том 85 (3), pp. 582-595, 2012.

[30] A. Cau, H. Janicke и B. Moszkowski, „Verification and enforcement of access control policies,“ *Formal Methods in System Design*, том 43, № 3, pp. 450-492, 2013.

[31] Н. Вълчанов, *Моделиране и изграждане на разширяеми модулни информационни системи,* Пловдив: Дисертационен труд, Пловдивски университет "Паисий Хилендарски", 2011.

[32] E. Evans, Domain-Driven Design Tackling Complexity in the Heart of Software, Addison-Wesley, 2003.

[33] M. Fowler, „Inversion of Control Containers and the Dependency Injection pattern,“ 23 01 2004. [Онлайн]. Available: https://martinfowler.com/articles/injection.html.

[34] Х. Т. Христов, *Методика на преподаване на съвременни технологии за създаване на софтуер,* Пловдив: Дисертационен труд, Пловдивски университет "Паисий Хилендарски", 2016.

[35] M. Diaz, Petri Nets: Fundamental Models, Verification and Applications, ISTE, 2009.

[36] E. W. Mayr и J. Weihmann, „A Framework for Classical Petri Net Problems: Conservative Petri Nets as an Application,“ в *International Conference on Applications and Theory of Petri Nets and Concurrency*, 2014.

[37] P. Pawlewski, Petri nets - manufacturing and computer science, InTech, 2012.

[38] W3Techs, „W3Techs,“ 13 3 2017. [Онлайн]. Available: https://w3techs.com/.

[39] A. Cau, H. Zedan, N. Coleman и B. Moszkowski, „Using ITL and Tempura for large-scale specification and simulation,“ в *Proceedings of 4th Euromicro Workshop on Parallel and Distributed Processing*, Braga, Portugal, 1996.