ПОСТИГАНЕ НА СЪБИТИЙНО-ОРИЕНТИРАНО ПОВЕДЕНИЕ ВЪВ ВИРТУАЛНОТО ОБРАЗОВАТЕЛНО ПРОСТРАНСТВО ЧРЕЗ МОДЕЛ ЗА ПРЕДСТАВЯНЕ И ОБРАБОТКА НА СЪБИТИЯ

Желян Гуглев, Емил Дойчев

Резюме: *Теоретичният модел на Виртуалното Образователно Пространство представляващо усъвършенствана версия на средата позната като Разпределен Център за Електронно Обучение (Distributed eLearning Center) предвижда събитийно-ориентиран подход, при взаимодействия между повечето нейни компоненти. За целта е необходим гъвкав модел за представяне и обработка на събития, описан в публикацията.*

Ключови думи: *Виртуално Образователно Пространство, събития, събитиен модел, събитийни параметри*

Въведение

Виртуалното Образователно Пространство наследник на средата за електронно обучение Distributed eLearning Center (DeLC), използвана във ФМИ на Пловдивския университет, представлява комплексна, кибер-виртуална структура, с множество опериращи в нея компоненти, крайната цел на съвкупността от които е доставка на електронни образователни услуги и учебно съдържание за подпомагане на образователния процес (Stoyanov, Ganchev, Popchev, O'Droma, 2005), (Stoyanov, Popchev, Doychev, Mitev, Valkanov, Stoyanova-Doycheva, Valkanova, Minov, 2010), (Стоянов, Попчев, 2014). За разлика от модела на DeLC, който разчита на виртуалната среда като единствена възможност за провеждане на образователен процес, ВОП отчита и физическата среда, като предвижда тясна интеграция между нея и виртуалната, обединявайки ги в единна Internet of Things екосистема (Стоянов, Орозова, Попчев, 2008), (Стоянов, Попчев, 2015), (Stoyanov, Zedan, Doychev, Valkanova, Stoyanova-Doycheva, Valkanov, 2016).

Цел

За долавяне, управление и синхронизация на взаимодействията между различните ВОП компоненти включващи интелигентни агенти, електронни услуги, онтологии и други модули (Stoyanov, Popchev, Doychev, Mitev, Valkanov, Stoyanova-Doycheva, Valkanova, Minov, 2010) е предвидено използване на събитийно-ориентиран подход. Постигането на ориентирано към събитията поведение изисква създаването на единен, гъвкав и преносим модел за представяне и обработка на събития, ключов за който са отчитането на факторите време и местоположение (Jain, 2008).

Задачи

Необходимо бе да се поставят основите на класификацията, към която различните събития възникващи в рамките на ВОП биха могли да се причислят, както и да се дефинират най-основните фактори, които определен клас събития трябва да могат да отчетат дискретно при възникването си.

Моделът за представяне и обработка на събития във ВОП трябва да може да работи като програмна библиотека и услуга. Първият режим на работа предоставя удобен начин за боравене със събития, а оперирането като услуга позволява управление

на информационния поток от събития генериран в пространството, абониране за даден вид събития, филтриране, разпращане към абонати и прочее.

Програмната библиотека трябва да може да предостави основни механизми за работа със събития, които да позволяват тяхната динамична промяна с цел редактиране или обогатяване, включвайки допълнителни, понякога и второстепенни за модела фактори, както и извършването на сравнителни операции с други събития.

Информацията за възникнало конкретно събитие трябва да може да се представя еднозначно в компактен, дискретен формат, който да е преносим през различни среди и обработваем от различни платформи.

Материал и методи

Представяне и класификация на събития във ВОП

Съществуват различни начини за представяне на събития. Често срещана класификация е според съдържанието на данните, които характеризират представеното събитие:

- *Атомарни събития* – основни, неделими структури без атрибути. По-сложни събития се представят като комплексни структури изградени от атомарните.
- *Атрибутирани събития* – събитията се представят на по-високо абстрактно ниво, като всяко събитие се характеризира с различни атрибути.

Вземайки предвид разнообразието от видове компоненти, които ще оперират и взаимодействат помежду си във ВОП бе приет смесен подход, който позволява на индивидуалните компоненти да работят с наличните събития и да детайлизират техни отделни аспекти.

Като уникално идентифициращи едно събитие атрибути бяха определени наредената тройка от параметри *(e_id, e_type, e_attr)* – идентификатор на събитие, тип събитие, множество от допълнителни характеризиращи атрибути в зависимост от типа. Подобен подход е описан в (Etzion O., Niblett P., D., L., 2010). Самите събитийни атрибути могат да представят разнообразни видове данни включително и цели други събития. Въпреки че гореописаната наредена тройка е достатъчна за уникално идентифициране, в практиката обикновено се изискват още данни, които да опишат по-подробно обстоятелствата при възникването на събитията. За тази цел бяха добавени още няколко атрибута, които покриват най-честите случаи при детайлизирането - време на възникване в милисекунди, приоритет, местоположение, описание, подсъбития изграждащи текущото, динамични параметри. Последните два вида атрибути позволяват лесното постигане на динамичност и използване на динамично-рекурсивни дефиниции при описанието.

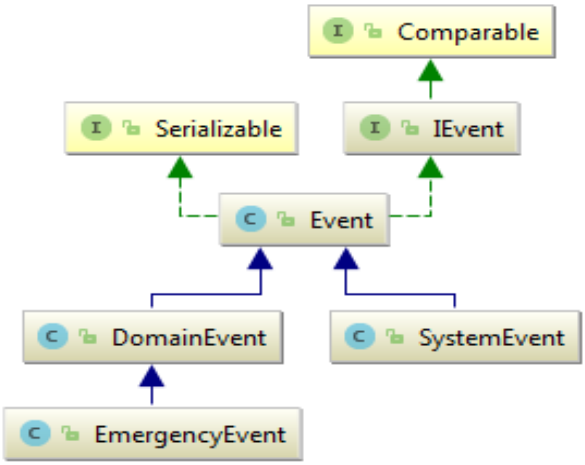
Съществуват множество възможности за класифициране на събития. В контекста на ВОП се отличават три основни групи събития – базови (атомарни, включващи гореописаните допълнителни атрибути), системни събития за представяне на явления, които засягат архитектурата на ВОП (напр. генериране или премахване на компонент) и домейн-събития свързани с конкретна приложна област (в контекста на ВОП лекции, упражнения, изпити и др.). Специално подмножество на домейн-събитията са emergency събитията използвани за отбелязването на изключителни ситуации, често с по-висок приоритет от останалите видове събития.

В зависимост от мястото на възникване събитията могат да се разделят на реални т.е. случили се във физическия свят и виртуални, отнасящи се само до виртуалното пространство. Поради спецификите на ВОП, съществува изискване и за

двата вида да е възможно тяхното виртуално представяне, тъй като може да съществуват различни отношения между тях.

Интерпретатор на събития „Event Engine“

Реализацията на изискванията на събитийния модел бе направена посредством създаването на така наречения интерпретатор на събития „Event Engine“. В основата на реализацията му, изцяло изпълнена с програмния език Java, стои обектно-ориентираната идеология за представяне на отделните видове събития чрез йерархия от специализирани класове, характерно за които е поддръжката на два основни вида събитийни атрибути дефинирани в интерпретатора като „вградени“ и „динамични“ параметри. За „вградени“ се считат онези параметри на събитието, които са предварително известни и представляват ясно обособена част от неговата дефиниция. Всички останали видове събитийни параметри се приемат за „динамични“ т.е. такива, които не винаги присъстват в обичайната дефиниция събитието, често налични само за конкретен представител на даден вид събитие, обикновено описващи нетипични, изключителни обстоятелства при възникването му.



Фигура 1 Събитийна йерархия имплементирана в Event Engine

На фигура 1 е представена основната йерархия използвана при класифициране на събития. Като основно изискване е всички класове да имплементират интерфейса „*net.uniplovdiv.fmi.cs.delc2.IEvent“*, посредством наследяване на някой от вече имплементираните в пакета „*net.uniplovdiv.fmi.cs.delc2.event*“ основни събития или чрез директна реализация на абстрактните методи от интерфейса. Самият *IEvent* от своя страна разширява Java интерфейса „*java.lang.Comparable*“, използван при сравняването на събития. Изисква се и имплементация на интерфейса „*java.lang.Serializable*“, който позволява на класа да бъде сериализиран, съответно пренасян през различни среди. Разчитайки само на стандартните Java механизми за сериализация на класове преносимостта на събитията се ограничава до Java платформи. В една типична IoT система обаче не е рядкост, някоя от страните да не поддържа Java. За справянето с този проблем Event Engine поддържа алтернативна сериализация използваща JavaScript Object Notation формат, имплементации, за който съществуват в болшинството от платформи.

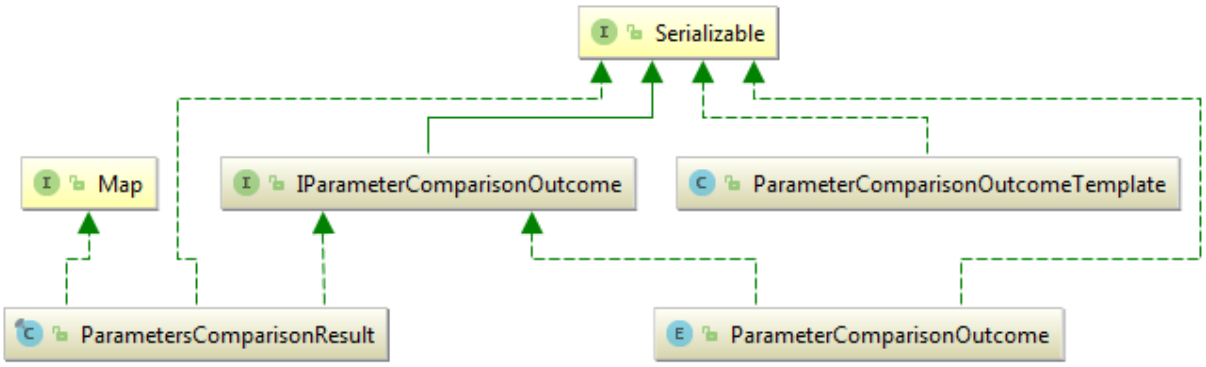
За отбелязването на вградените параметри на едно събитие се използва специално разработената за целта Java анотация *“net.uniplovdiv.fmi.cs.delc2.event.annotations.EmbeddedParameter“*. Представянето на динамични параметри в едно събитие става посредством специализирани структури имплементиращи Java интерфейса *„java.util.Map“* и по-конкретно класовете *„ParametersContainer“* и *„EventsContainer“* от пакета „*net.uniplovdiv.fmi.cs.delc2.event.parameters“*. Съществува

изискване за всички събитийни параметри да разполагат с уникално име под формата на низ, което се използва за идентифициране, при сериализация, при автоматизирано „смесване“ на вградени и динамични параметри в общи структури с цел масова обработка. Динамичните параметри изпълняват условието чрез класовете имплементиращи *„java.util.Map“*. При вградените параметри за това се грижи анотацията *„EmbeddedParameter“*, която приема като задължителен параметър низ специфициращ уникалното име. Съществуването ѝ също така прави възможно отличаване и включване на „служебни“ данни като част от събитийния клас, от които зависи инстанцирането на събитието, но нямат общо със събитийните параметри, проверка на типовете данни и дали отговарят на изискванията на Event Engine по време на компилиране на кода, възможности за смесване на вградени и динамични параметри.

Сравнителни операции върху събития чрез Event Engine

Упражняването на сравнителни операции върху събития е от изключително значение при анализа и интерпретацията им. Съществуват два начина за тяхното прилагане – императивен използвайки наличните в езика Java оператори и методи за сравнение и декларативен, специално разработен за да осигури контролирано изпълнение чрез данни, които могат да идват от външен източник. Самите данни представляват описание на благоприятните изходи от сравнението и се представят в специализирани, сериализируеми структури-шаблони (клас *ParameterComparison OutcomeTemplate*). Конкретните сравнения се извършват спрямо общи събитийни параметри, като самата операция се базира на метода *„java.lang.Comparable:: compareTo“*, но разширява множеството от възможни резултати до 8 (дефинирани в енумератора *ParameterComparisonOutcome*) – стандартните LESS (по-малко), EQUAL (равно) и GREATER (по-голямо) заедно с още 3 специализирани вида - NOTCOMPARED (не е сравнен), UNKNOWN (неизвестен), INCOMPARABLE (несравним), както и 2 помощни комбинации – NOTCOMPARED_UNKNOWN (не е сравнен или неизвестен), NOTCOMPARED_UNKNOWN_INCOMPARABLE (не е сравнен или неизвестен или несравним). Допълнителните изходи позволяват връщане на краен резултат дори при сравняване на събития без общи атрибути. Очакваните изходи се асоциират със съответните събитийни параметри чрез класа *ParamatersComparisonResult.*

Изграждането на по-сложни логически условия също е възможно посредством поддръжката на полета в класа *ParametersComparisonOutcomeTemplate* за конюнкция и дизюнкция, съхраняващи наредени множества от класове от същия тип, както и булево поле за отрицание.



Фигура 2. Помощни структури за извършване сравнителни операции

Резултати

В IoT среда, където една от основните форми на комуникация ще бъде чрез събития е от изключително значение скоростта, която тяхната обработка отнема. За измерване на средната скорост при сериализация и десериализация бяха използвани 1 000 000 еднотипни събитийни обекти, наследници на домейн-събитията от базовата йерархия използваща обичайните за нея събитийни атрибути, дефиниращи в себе си два допълнителни атрибута – един низ и едно събитие. Измерванията бяха направени без използване на паралелизация, върху Java среда версия 1.8.0_144-b01, под операционна система Windows 7 64bit върху двуядрен процесор Intel Core i5 460M 2,8 GHz с включен hyper threading и оперативна памет от 8 GB DDR3-1066.

Вид операция	Входящ формат	Изходящ формат	Общо време милисекунди	Време за обект микросекунди
Сериализация	Java Object	JOSS	17360	17.36
Сериализация	Java Object	JSON	24689	24.689
Десериализация	JOSS	Java Object	86057	86.057
Десериализация	JSON	Java Object	27584	27.584

Таблица 1. Време за сериализация и десериализация на събитийни обекти

Измерването на скоростта при сериализация и десериализация на обекти за изграждане на шаблони от сравнителни операции за събития бе направена върху същата платформа като тази предоставила резултатите от **Error! Reference source not found.**. Използвани бяха 1 000 000 еднотипни обекти, всеки един от които съдържа по едно сравнение за конкретен атрибут плюс още едно в конюнктивната структура.

Вид операция	Входящ формат	Изходящ формат	Общо време милисекунди	Време за обект микросекунди
Сериализация	Java Object	JOSS	6701	6.701
Сериализация	Java Object	JSON	15213	15.213
Десериализация	JOSS	Java Object	43471	43.471
Десериализация	JSON	Java Object	21021	21.021

Таблица 2. Време за сериализация и десериализация на шаблони за сравнение на събитийни обекти

По отношение на разликата в размерите на сериализираните обекти бяха установени:

Тип обект	Сериализиращ формат	Размер байта
Събитиен – Таблица 1	JOSS	1618
Събитиен – Таблица 1	JSON	1408
Шаблон за сравнение на събития – Таблица 2	JOSS	759
Шаблон за сравнение на събития – Таблица 2	JSON	1484

Таблица 3. Размер на сериализирани събитийни и шаблонни обекти

Обсъждане

Представените на таблица 1 и таблица 2 резултати показват приемливи разлики в производителността, които взаимно се компенсират при работа в двете посоки. По този начин идеологичните различия между Java Object Serialization Specification и JavaScript Object Notation Format по-скоро влияят на избора на формат по отношение на удобство и приложимост и по-малко върху скоростта, както бе заложено в оригиналната идея при създаването на тази функционалност.

Причините за разликите в скорост и размер на изхода при сериализация и десериализация се дължат на особеностите на механизмите на работа на вградената в Java „JOSS“ имплементация и използваната за работа с JSON библиотека „GSON“, като някой по-основни са:

- JOSS е проектиран да бъде сигурен, като за целта включва в себе си допълнителна валидация на сериализираните данни.
- JSON е проектиран да бъде опростен и по-отворен, което ограничава валидацията на данни.
- JOSS използва двоичен формат и разполага с повече правила за представяне на различните видове данни.
- JSON разполага само с общи правила за представяне на основополагащи видове данни. Това прави представянето на другите по-неефективно (например двоични данни, сложна йерархия от класове и др.).
- При сериализация JOSS изгражда пълно описание на обекта включително и за NULL атрибути.
- Повечето JSON имплементации като GSON позволяват пропускане на NULL атрибути.
- За постигане на надеждна работа с конкретни имплементации на Java интерфейси, абстрактни класове и йерархии от гореописаните при сериализация до JSON, Event Engine извършва допълнителна работа за установяване на връзките между обектите.
- С цел по-висока устойчивост към грешки и безопасност Event Engine се грижи за динамичното добавяне и премахване на някой допълнителни полета по време на JSON сериализация и десериализация, като например версия на обекта и по-детайлна информация за произхода му.

Заключение

Текущите нужди на Виртуалното Образователно Пространство за гъвкаво представяне на събития се удовлетворяват напълно от разработения модел. Особеностите на практическата му реализация наречена Event Engine с помощта на програмния език Java и създадените специализирани контейнери за съхраняване на метаданни и данни успяха да комбинират изискванията за строга типизираност от страна на езика с нуждата от динамичност в модела. Предвижданото на нуждата от представяне на събития в алтернативен формат още преди модела да бъде имплементиран, насочи реализацията му да фаворизира максимално опростени и в същото време лесни за използване структури от данни. В резултат постигнатата скорост при сериализация и десериализация на съответните Java класове до стандартен или алтернативен формат е задоволителна, а изходните данни адекватно отразяват съдържанието на оригинала без излишно дублиране.

Възможностите за прилагането на първичен анализ на събития по отношение на определени критерии, включително и логически съждения за тях, изцяло представени с данни разширяват теоретичните възможности на Event Engine да взаимодейства с други системи.

Широката разпространеност на Java платформата гарантира преносимостта на Event Engine, а поддръжката на алтернативен формат за представяне на събития разширява възможностите за разпространение и работа с ВОП събития извън Java екосистемата.

Благодарности

Изследването е частично финансирано от НПД – Пловдивски Университет „П. Хилендарски“ – Проект No. FP17-FMI-008 „Иновационни софтуерни инструменти и технологии с приложения в научни изследвания по математика, информатика и педагогика на обучението“, 2017–2018.

Литература

Stoyanov, S., I. Ganchev, I. Popchev and M. O'Droma, From CBT to e-Learning, *B: Information Technologies and Control*, Vol. III, № 4, 2005, стр. 2–10.

Jain, R., EventWeb: Developing a Human-Centered Computing System, *Computer*, Vol. 41, № 2, February 2008.

Stoyanov, S., I. Popchev, E. Doychev, D. Mitev, V. Valkanov, A. Stoyanova-Doycheva, V. Valkanova and I. Minov, DeLC Educational Portal, *Cybernetics and Information Technologies*, Vol. 10, № 3, 2010, 49–69.

Стоянов, С. и И. Попчев, DeLC – минало, настояще, бъдеще, *„From DeLC to VelSpace“*, пленарен доклад, Пловдив, 2014.

Стоянов, С. и И. Попчев, Инфраструктури за електронно обучение, *Техносфера*, БАН, 4(30)/2015, ISSN 1313-38612015, 38–45.

Stoyanov, S., H. Zedanl, E. Doychev, V. Valkanova, A. Stoyanova-Doycheva and V. Valkanov, Context-Aware E-Learning Infrastructure, *The ICT Age*, Cambridge Scholars Publishing, ISBN (10): 1-4438-8714-5, ISBN (13): 978-1-4438-8714-4, 2016 г., стр. 221–280.

Стоянов, С., Д. Орозова и И. Попчев, Виртуално образователно пространство – настояще и бъдеще, *Юбилейна научна конференция с международно участие „Новата идея в образованието“*, 20–21 септември, Бургас, 2016, 410–418.

Etzion, O. and P. Niblett, *Event Processing in Action, Greenwich: Manning Publications*, 2011.

Java Object Serialization Specification, 2010, https://docs.oracle.com/javase/8/docs/platform/serialization/spec/serialTOC.html, 27 Август 2017

Факултет по математика и информатика,

Пловдивски университет „Паисий Хилендарски“,

Пловдив 4003, бул. „България“ № 236,

Имейли: e.doychev@isy.dc.com, jelian_g@mail.bg

ACHIEVING AN EVENT-ORIENTED BEHAVIOR IN THE VIRTUAL EDUCATION SPACE USING A MODEL FOR REPRESENTATION AND PROCESSING OF EVENTS

Zhelyan Guglev, Emil Doychev

Summary: *The theoretical model of the Virtual Education Space, improved version of the environment known as Distributed eLearning Center requires an event-oriented approach as a basic way for interaction between the most of its components. For that purpose there is a requirement for a flexible model allowing event representation and manipulation. This publication describes some of the underlying technical aspects and implementation details of the model.*

Key words: *Virtual Education Space, events, event model, event parameters*