Zbornik radova Fakulteta tehničkih nauka, Novi Sad

**UDK: 004.9**
**DOI: https://doi.org/10.24867/09BE09Ristanovic**

# ЈЕЗИК И ИНТЕРПРЕТЕР ЗА ИНТЕГРАЦИЈУ ВЕБ БАЗИРАНИХ АПЛИКАЦИЈА

# A LANGUAGE AND INTERPRETER FOR INTEGRATION OF WEB APPLICATIONS

Стефан Ристановић, *Факултет техничких наука, Нови Сад*

**Област – СОФТВЕРСКО ИНЖЕЊЕРСТВО И ИНФОРМАЦИОНЕ ТЕХНОЛОГИЈЕ**

**Кратак садржај –** *У овом раду описан је дизајн и имплементација језика специфичног за домен и интерпретера за интеграцију веб базираних апликација.*

**Кључне речи:** *ЈСД, аутоматизација, интеграција, интерпретер, textX*

**Abstract** – *This paper presents design and implementation of a domain-specific language and interpreter that is used for integration of web applications.*

**Keywords:** *DSL, automation, integration, interpreter, textX*

## 1. УВОД

У времену у којем живимо велики број веб апликација чини нашу свакодневицу. Без обзира на њихову намену, редак је случај да те апликације функционишу изоловано од остатка света. Ипак, постојање интеграција између веб апликација је прилично ограничено из више разлога, од којих је најчешћи онај који се тиче исплативости и одрживости таквих функционалности. Са повећањем броја веб апликација у свакодневној употреби, појавила се и потреба за системима чија је основна намена интеграција и аутоматизација процеса између њих. Међу сервисима који пружају овакву врсту услуге издвајају се *Zapier* [1] и *Microsoft Flow* [2]. Популарност *Zapier*-а и присуство компаније *Microsoft* са својим производом у овој сфери сведочи њеној валидности и реалној потреби за оваквим системима.

Тема овог рада је развој система који ће омогућити корисницима интеграцију веб апликација, односно аутоматизацију процеса кроз њихову интеграцију. За разлику од горепоменутих система, развијени *Calcifer*[1] систем представља софтвер отвореног кода, који ће настојати да корисницима олакша процес спецификације токова интеграције и аутоматизације пружањем језика специфичног за домен - *CalciferDSL*. Поред тога, посебна пажња биће стављена на аспект једноставне проширивости система.

## 2. ЈЕЗИЦИ СПЕЦИФИЧНИ ЗА ДОМЕН

У свим гранама науке и инжењерства могу се разликовати приступи који су општи и они који су специфични. Општи приступ пружа генеричка решења за велики број проблема, али та решења најчешће нису оптимална. Специфичан приступ пружа квалитетнија решења, али у доста ужој области. Ова два принципа решавања проблема у области софтверског инжењерства огледају се кроз језике опште намене (ЈОН) и језике специфичне за домен (ЈСД).

Од свог настанка, концепт језика специфичних за домен никада није имао строгу дефиницију и границе ових језика су још увек релативно флексибилне. Једну од дефиниција језика специфичних за домен пружа Мартин Фовлер (енг. *Martin Fowler*), који ову врсту језика дефинише као програмске језике ограничене експресивности фокусиране на одређену област [3]. Фовлер наглашава да фокусирање на узак домен заправо произилази из ограничења у експресивности. Да границе ове врсте језика нису стриктно дефинисане видимо из дефиниције: "Језици специфични за домен су програмски језици или извршиве спецификације језика које пружају, кроз прикладне нотације и апстракције, широку експресивност фокусирану, а често и ограничену, на специфичан домен проблема" [4]. Разлике између ове две дефиниције су суптилне али обе наглашавају да је кључна карактеристика језика специфичних за домен заправо њихова *фокусирана експресивна моћ,* која није специфична за језике опште намене.

### 2.1. Предности и мане

Коришћење језика специфичних за домен није универзално решење за све проблеме које доносе језици опште намене. Израда и коришћење ЈСД-а има своје предности и мане. Да би се оно оправдало, мора се пронаћи баланс између добити и ризика које долазе са њим. Према [3] и [5] предности које језици специфични за домен пружају корисницима су повећана продуктивност, повећан квалитет кода и једноставнија валидација и верификација модела. Поред тога доменски језици пружају и побољшање комуникације између програмера и клијената. Изазови које доноси употреба ЈСД-ова нису занемариви и према [3] и [5] у пракси се најчешће јављају они који се тичу цене изградње, проблема око савладавања језика од стране корисника као и потребно време за дизајн, одржавање и еволуцију језика.

### 2.2. Категоризација

Најчешћи, и у струци најприхваћенији, начин поделе језика специфичних за домен је онај на **интерне** и **екстерне**. Та подела је базирана на начину имплементације језика, и односи се на то да ли је доменски језик имплементиран у оквиру неког језика опште намене (користећи конструкције матичног језика) или је потпуно самосталан. Аутор ове поделе је *Martin Fowler* [3]. Са развојем области, овим основним катего-

---

**НАПОМЕНА:**
**Овај рад проистекао је из мастер рада чији ментор је био др Игор Дејановић, ванредни проф.**

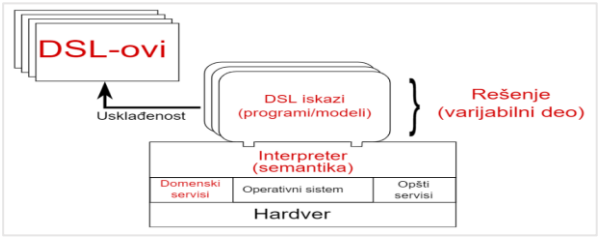
[1] https://github.com/stkeky/calcifer

ријама, које се односе на текстуалне ЈСД-ове, можемо додати и нетекстуалне (енг. *rich DSLs*) [6]. *Martin Fowler* такође препознаје ове посебне карактеристике и своју првобитну поделу проширује са додатном категоријом – **језичке радионице** [7]. У контексту овог рада, развијени *CalciferDSL*, може се сврстати у категорију екстерних језика специфичних за домен.

### 2.3. Интерпретер – шаблон имплементације

Када је доменски језик дефинисан, потребно је изабрати начин на који ће он бити имплементиран. Избор начина имплементације може имати огроман утицај на целокупан процес развоја ЈСД-а. Како за имплементацију језика опште намене тако и за језике специфичне за домен постоје одређени шаблони имплементације који стандардизују овај процес. Неки од најчешће коришћених шаблона приказаних у [8] су интерпретер, компајлер (генератор кода), препроцесирање, *embedding*, прошириви интерпретер/компајлер и хибридни шаблон.

Интерпретер и генерисање кода представљају два најчешћа шаблона имплементације који се срећу у пракси. Основни принцип који карактерише генерисање кода јесте да се конструкције језика специфичног за домен преводе у конструкције матичног језика опште намене и позиве различитих библиотека тог језика, док то није случај са интерпретер шаблоном (*Слика 2.1*) где се конструкције језика препознају и извршавају кроз стандардни циклус корака: *добави*, *декодирај* и *изврши*. Овај принцип карактерише одсуство генерисања кода јер се програм динамички евалуира у време извршавања.




*Слика 2.1 Архитектура интерпретер шаблона* [10]

Предности које интерпретер шаблон имплементације доноси у односу на генерисање кода су брже измене (енг. *roundtrip*), већа динамичност и флексибилност, једноставније увођење, портабилност као и способност дебаговања модела током извршавања уколико постоје специјализовани алати. Мане интерпретације у односу на генерисање кода су спорије извршавање и смањена ефикасност у поређењу са приступом генерисања кода.

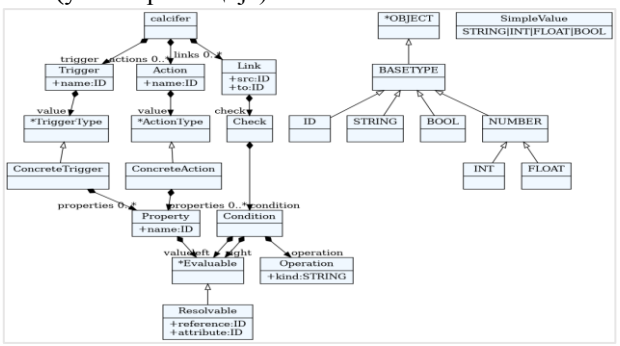
У контексту овог рада, развијени *CalciferDSL* је имплементиран коришћењем интерпретер шаблона, а развијен је уз помоћ *textX* алата [9].

## 3. СПЕЦИФИКАЦИЈА СИСТЕМА

Циљ *Calcifer* система јесте да се кориснику омогући једноставна интеграција између више различитих веб апликација. Основно средство за остварење горепостављеног циља представља дизајн и креирање посебног језика специфичног за домен – *CalciferDSL*.

### 3.1. CalciferDSL

Основна намена овог језика јесте да се кориснику олакша креирање комплексних токова интеграције између више веб апликација. Идеја на којој се заснива јесте да се читав ток представи уз помоћ ацикличног графа са кондиционалним гранањем, алтернативним и паралелним путањама. Овај концепт у *Calcifer* екосистему се назива *Spark*. Главни концепти *CalciferDSL*-а (*Слика 2.2*) су ***Trigger*** (једини улаз у граф тока, односно догађај од интереса на некој од интегрисаних апликација), ***Action*** (радња коју је потребно извршити) и ***Link*** (грана графа којом су чворови повезани, омогућава условну транзицију). *CalciferDSL* чине и: ***SimpleValue*** (појединачна константна вредност која се додељује приликом дефинисања), ***Resolvable*** (вредност коју је потребно добавити у току извршавања, референцира атрибут другог чвора), ***Evaluable*** (апстракт који обједињује *SimpleValue* и *Resolvable*), ***Property*** (атрибут чвора) и ***Gate*** (услов транзиције).




*Слика 2.2 Приказ основних концепта CacliferDSL-а*

### 3.2. Синтакса језика

Спецификација *Spark*-а се састоји из три основне секције које започињу јединственом кључном речи коју прати двотачка (`:`). Прва секција захтева навођење јединог тригера за читав *Spark* и ова секција започиње кључном речи `trigger` коју прати дефиниција тригера. Наредна секција, која је представљена кључном речи `actions`, служи за навођење свих акција које су саставни део овог тока. Последњу секцију, која се започиње кључном речи `flows`, чине дефиниције линкова, тј. грана графа. Ова секција садржи сву логику тока јер повезује све чворове графа и дефинише услове прелаза из једног у други чвор.

```
trigger:
  release GithubRelease
actions:
  sendEmail SendEmail(
    apiKey: "<API_KEY>",
    title: "New release just in!",
    header: $release.repository,
    content: $release.value,
    to: "st.keky@gmail.com",
    subject: "New Release")
flows:
  release -> sendEmail  [$release.type == "created"]
```

*Листинг 3.1 Приказ једноставног Spark-а*

Дефиниције тригера и акција су сличне, и обе се састоје се из јединственог назива (који служи за идентификацију) и типа. Акције могу поседовати и атрибуте које је потребно навести у оквиру заграда. Нотација за исказивање атрибута је `name:value`. Сваки чвор графа се, уз помоћ посебног оператора за селекцију - `'$'`, може референцирати. Захваљујући овоме могуће је искористити вредност атрибута из било којег претка тренутног чвора. Референцирање је могуће и током дефинисања услова за транзицију. Нотација за референцирање вредности је `$node_name.property`. За дефинисање линкова

користи се нотација `in_node->out_node`. Опционо сваки линк може садржати услов неопходан за транзицију из улазног у излазни чвор. Услов се наводи унутар средњих заграда коришћењем нотације `[leftOperand operator rightOperand]`. И леви и десни операнди могу бити или константне вредности или референце на вредности атрибута неког од претходних чворова. На *Листинг 3.1* приказана је синтакса *CalciferDSL*-а којом је описан један *Spark*.

### 3.3. Семантичка валидација

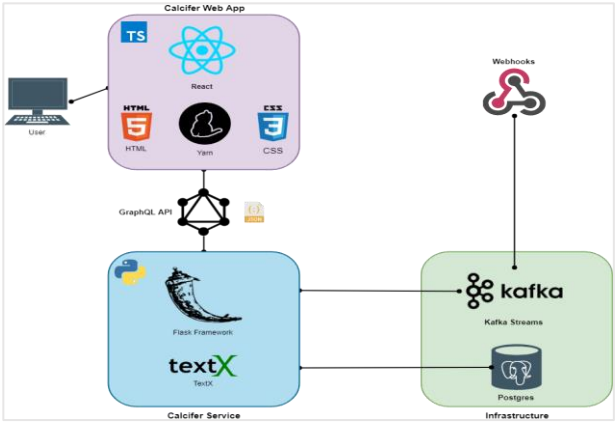
Сваки синтаксно валидан израз у оквиру *CalciferDSL*-а не мора бити семантички исправан. Како би се одржала семантичка исправност моделоване аутоматизације, *CalciferDSL* са собом доноси одређена семантичка правила. Та правила укључују ограничење на тачно један тригер, превенцију спецификације акције које нису повезане са главним током, ограничење јединствености назива свих ентитета, ограничење да претходно референцирана акција мора постојати и бити један од предака акције која је референцира и превенција цикличних токова којим се спречава потенцијално бесконачно извршавање тока.

## 4. ИМПЛЕМЕНТАЦИЈА СИСТЕМА

У овом поглављу описана је архитектура и детаљи имплементације *Calcifer* система .

### 4.1. Архитектура система и коришћени алати

*Calcifer* систем сачињавају три целине (*Слика 4.1*): сервер са интерпретером *CalciferDSL* језика, компонента за процесирање *webhook*-ова и клијентска апликација.

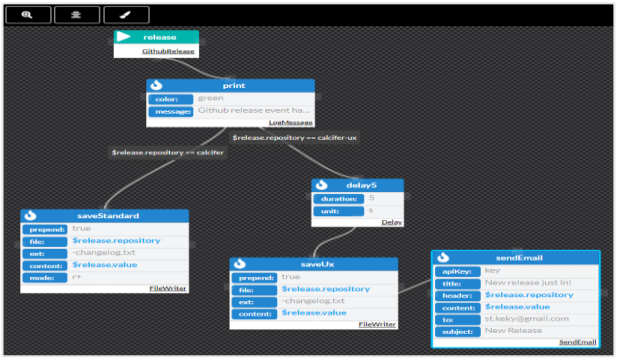



*Слика 4.1 Архитектура Calcifer система*

Серверска страна *Calcifer* система изграђена је уз помоћ *Flask microframework*-а коришћењем *Python* програмског језика. У оквиру серверске стране налази се спецификација и интерпретер *CalciferDSL*-а изграђеног уз помоћ *textX* алата. Серверска страна пружа *GraphQL* API, који клијентска страна конзумира ради управљања ентитетима.

Перзистенција свих података о ентитетима врши се у *PostgreSQL* релациону базу података. У оквиру серверске стране, као посебна компонента, налази се процесор свих долазних захтева (*webhook*) од веб апликација које су чиниоци интеграције. Када HTTP захтев пристигне, ова компонента га обрађује и шаље на за то предвиђен *Kafka* топик. Са овог топика га касније парсира конзумер који покушава да препозна да ли је захтев везан за неки од активних токова аутоматизације. Уколико је захтев повезан са неким током, он бива прослеђен интерпретеру на даљу обраду. Интерпретер уз помоћ *textX* алата парсира спецификацију изражену кроз *CalciferDSL* и покреће одговарајући ток аутоматизације.

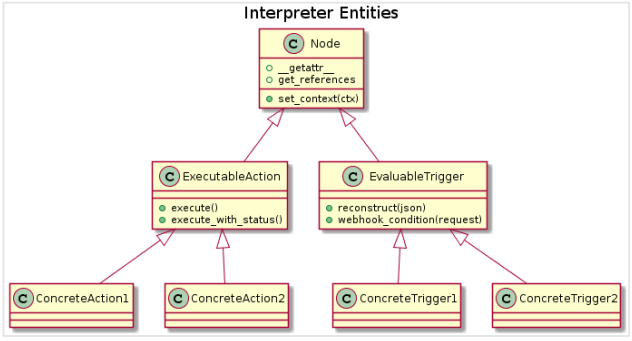
Клијентску страну *Calcifer* система чини *single-page* веб апликација написана уз помоћ *ReactJS* радног оквира коришћењем *TypeScript*-а. Клијентска страна пружа једноставан графички кориснички интерфејс који конзумира *GraphQL* API, и омогућава корисницима једноставну манипулацију ентитетима. У оквиру клијентске стране налази се текстуални едитор прилагођен *CalciferDSL*-у (бојење синтаксе, комплетирање кода итд.) изграђен помоћу *react-monaco-editor* библиотеке, као и компонента за визуелизацију читавог тока аутоматизације (*Слика 4.2*) изграђена помоћу *storm-react-diagrams* библиотеке.




*Слика 4.2 Визуелизација Spark-а*

### 4.2. Интерпретер

*Spark* је представљен структуром графа и његово извршавање ефективно представља обилазак графа. Један чвор овог графа представљен је апстрактном класом `Node`. Ову апстрактну класу наслеђују друге две апстрактне класе `ExecutableAction` и `EvaluableTrigger`, које представљају акцију и тригер респективно. Ове две класе ће бити наслеђене од стране конкретних тригера и акција (*Слика 4.3*).




*Слика 4.3 Модел основних ентитета интерпретера*

`EvaluableTrigger` класу наслеђују сви конкретни тригери. Наслеђивање ове класе од стране конкретних тригера захтева редефинисање две методе: `webhook_condition` која врши проверу долазног захтева и упарује захтев са конкретним тригером и метода `reconstruct` која на основу садржаја пристиглог захтева реконструише све атрибуте тригера. `ExecutableAction` представља родитељску класу сваке акције и захтева редефинисање методе `execute`. Позив методе `execute` представља извршавање ефекта акције која је описује. Приликом

интерпретирања *Spark*-а, када буде посећен чвор који представља конкретну акцију биће позвана ова метода и њено извршавање представљаће извршавање ефекта повезаног са њом. Проширивање *Calcifer* система је једноставно и додавање нових интеграција заснива се на имплементирању конкретних класа које наслеђују ове две апстрактне класе, и оне ће динамички бити учитане и доступне за коришћење у оквиру *CalciferDSL*-а и целокупног *Calcifer* система уопште. Једноставности доприноси и редефинисана метода `__getattr__` (приступ атрибутима објекта у *Python* језику у *dot* нотацији) основне `Node` класе, чија редефиниција омогућава приступ дубоко енкапсулираним `Property`-има конкретних акција и тригера у *dot* нотацији уз разрешавање оних који представљају референце на атрибуте других чворова.

Када се иницијални тригер деси и компонента за обраду *webhook*-ова препозна да је тригер повезан са активним *Spark*-ом, подаци се прослеђују интерпретеру. Интерпретер на основу мета-модела и спецификације *Spark*-а изражене кроз *CalciferDSL* врши парсирање употребом *textX* алата, а затим и изградњу конкретне инстанце графа тока употребом горепоменутих класа. Евалуација се врши обиласком графа у смеру од тригера ка крајњим чворовима позивањем рекурзивне методе `_eval_node`, која врши евалуацију тренутног чвора графа, потенцијално извршава везану акцију и проверава услове за прелазак у наредни чвор. Избор наредног чвора се своди на добављање свих излазних грана из тренутног чвора и провером услова за прелазак у наредни чвор.

Скуп излазних грана може бити празан, што значи да се тренутно евалуира лист графа и у том случају се интерпретер враћа корак уназад и процесира следеће подстабло уколико такво постоји. Када више не постоји ниједно подстабло које интерпретер, уз поштовање свих услова, није обишао, извршавање тока аутоматизације се завршава и врши се перзистенција статуса извршавања уз временске маркере.

## 5. ЗАКЉУЧАК

У овом раду је описан реализовани систем за интеграцију и аутоматизацију процеса између различитих веб апликација – *Calcifer*, при чему су приказани сви детаљи његове имплементације. Мотивација за овај рад пронађена је у практичним недостацима постојећих система прегледом и анализом стања у области. Детаљно је описан модел, како графа којим је представљен појединачни ток аутоматизације, тако и језгра интерпретера, али и читавог система. Пружен је преглед теоријских основа иза изабраног принципа имплементације коришћењем језика специфичног за домен, као и све предности и мане једног таквог решења. Темељно је описана синтакса креираног *CalciferDSL*-а, његова семантика и начини коришћења. Наведене су све технологије које су коришћене у процесу израде овог решења, као и разлози који су утицали на њихов избор. Стављен је нагласак на флексибилност овакве спецификације *CalciferDSL*-а, имплементацију интерпретера, лакоћу његовог коришћења и проширивости. Конструкција *Calcifer* система укључујући посебно развијени доменски језик - *CalciferDSL*, интерпретер токова аутоматизација и интеграција између веб апликација, као и посебан *CalciferUX* који олакшава коришћење овог језика специфичног за домен кроз коришћење сугестија, комплетирања кода, бојења синтаксе и визуелизације представљају главне доприносе овог рада.

Тренутна имплементација *Calcifer* система представља солидну основу за даљи развој и додавање нових функционалности. Неки од наредних праваца развоја могу укључивати додавање подршке за парцијално извршавање токова, подршке за више од једног корисника по активној инстанци уз побољшање безбедносних аспеката, поширивање система визуализације креирањем графичког доменског језика и додавање подршке за аутоматско покретање токова аутоматизација у одређеним временским интервалима као додатак тренутном приступу базираном на *webhook*-овима.

## 6. ЛИТЕРАТУРА


[1] Zapier, „Zapier,“ [На мрежи]. Available: https://zapier.com. [Последњи приступ 15. јул 2019.].

[2] Microsoft, „Microsoft Flow,“ [На мрежи]. Available: https://flow.microsoft.com. [Последњи приступ 11. јануар 2020.].

[3] M. Fowler, Domain-specific languages, Addison-Wesley, 2010.

[4] A. van Deursen, P. Klint и J. Visser, „Domain-Specific Languages: An Annotated Bibliography,“ *ACM SIGPLAN Notices*, т. 35, 2000.

[5] M. Voelter, DSL Engineering, 2013.

[6] G. Debasish, DSLs in Action, Manning Publications, 2010.

[7] M. Fowler, „Language Workbenches: The Killer-App for Domain Specific Languages?,“ 2005. [На мрежи]. Available: https://www.martinfowler.com/articles/languageWorkbench.html. [Последњи приступ 15. август 2019.].

[8] M. Mernik, J. Heering и A. M. Sloane, When and how to develop Domain-Specific Languages, Maribor, 2005.

[9] I. Dejanović, „textX,“ [На мрежи]. Available: https://textx.github.io/textX/stable/. [Последњи приступ 28. јул 2019.].

[10] I. Dejanović, „Generisanje programskog koda,“ 20. децембар 2019.. [На мрежи]. Available: http://www.igordejanovic.net/courses/jsd/04-generisanje-programskog-koda. [Последњи приступ 23. јануар 2020.].


**Кратка биографија:**


**Стефан Ристановић** рођен је у Лозници 1994. године. Основну школу „Боривоје Ж. Милојевић“ завршио је у Крупњу 2009. године, као носилац Вукове дипломе. Након тога уписује, а 2013. године завршава гимназију – општи тип у Средњој школи у Крупњу. Исте године уписује Факултет техничких наука у Новом Саду, смер Софтверско инжењерство и информационе технологије. Полаже све испите предвиђене планом и програмом са просечном оценом 9.83. Године 2017. завршава основне студије и уписује мастер академске студије на истом факултету. Полаже све испите мастер студија предвиђене планом и програмом са просечном оценом 10.00.