НАЦІОНАЛЬНА АКАДЕМІЯ НАУК УКРАЇНИ
ІНСТИТУТ ПРОГРАМНИХ СИСТЕМ

ISSN 1727-4907

П Р О Б Л Е М И ПРОГРАМУВАННЯ

НАУКОВИЙ ЖУРНАЛ

P R O B L E M S IN PROGRAMMING

SCIENTIFIC JOURNAL

2021

№ 1

Теми випуску:

- ***Інструментальні засоби та середовища програмування***
- ***Теоретичні та методологічні основи програмування***
- ***Моделі та засоби систем баз даних і знань***

НАЦІОНАЛЬНА АКАДЕМІЯ НАУК УКРАЇНИ
ІНСТИТУТ ПРОГРАМНИХ СИСТЕМ

П Р О Б Л Е М И ПРОГРАМУВАННЯ

науковий журнал

Головний редактор
Андон Пилип Іларіонович
академік НАН України,
директор Інституту програмних систем
НАН України
✉ Інститут програмних систем
НАН України
проспект Академіка Глушкова, 40, корп. 5
03187, Київ-187
☎ Тел.+380 (44) 526 5507
E-mail: andon@isofts.kiev.ua
http://www.pp.isofts.kiev.ua

Адреса для кореспонденції

✉ Інститут програмних систем
НАН України
Проспект Академіка Глушкова, 40
03187, Київ-187

☎ Тел.: +380 (44) 526 5065
Факс: +380 (44) 526 6263
E-mail: iss@isofts.kiev.ua

Затверджено до друку вченою радою Інституту програмних систем НАН України.
Протокол № 2 від 04.02.2021 р.

Редактор *В.П. Замула*
Комп'ютерна верстка *В.П. Замула*

Підписано до друку 24.03.2021. Формат 60x84/8. Папір офс. Ум. друк. арк. 10,98.
Обл.-вид. арк.10,41. Тираж 120 прим. Ціна договірна. Замовл.

Віддруковано ВД «Академперіодика» НАН України
вул. Терещенківська, 4, м. Київ, 01004
Свідоцтво суб'єкта видавничої справи ДК № 544 від 27.07.2001

НАЦІОНАЛЬНА АКАДЕМІЯ НАУК УКРАЇНИ
ІНСТИТУТ ПРОГРАМНИХ СИСТЕМ

ПРОБЛЕМИ ПРОГРАМУВАННЯ

науковий журнал

№ 1 січень - березень 2021

Заснований у березні 1999 р.

ЗМІСТ

Свідоцтво про державну реєстрацію КВ № 7490 від 01.07.2003

Науковий журнал "Проблеми програмування" занесений до переліку наукових фахових видань України, в яких можуть публікуватися основні результати дисертаційних робіт.

ISSN 1727-4907

NATIONAL ACADEMY OF SCIENCES OF UKRAINE
INSTITUTE OF SOFTWARE SYSTEMS

PROBLEMS IN PROGRAMMING

scientific journal

№ 1 *January – March* *2021*

Founded in March, 1999

CONTENTS

УДК 004.4'24 https://doi.org/10.15407/pp2021.01.003

А.Ю. Дорошенко, Б.В. Бодак

МОДЕЛЮВАННЯ RESTFUL API ДЛЯ СИСТЕМИ АВТОМАТИЗАЦІЇ ПРИВАТНИХ ЕЛЕКТРОННИХ ЗАКУПІВЕЛЬ

Розроблено програмний засіб для автоматизації електронних закупівель на основі .NET Core RESTful API з використанням специфікацій OpenAPI v3.0. Реалізовано авторизацію користувачів системи, постачальників та замовників, за допомогою відкритого стандарту OAuth та Microsoft Identity Server. Для скорочення часу відгуку системи здійснено кешування даних на рівні репозиторію за підтримки розподіленного кешу. Створено підсистему для обробки та переходу між станами закупівель на основі скінченного автомату станів. Проведено випробування розробленого програмного засобу з використанням модульних та інтеграційних тестів.

Ключові слова: електронні закупівлі, RESTful API, OpenAPI v3.0, архітектура API, масштабовані системи автоматизації.

Вступ

Автоматизація процесів у системах закупівель набула широкого використання в багатьох країнах Європи та світу [1]. Як свідчить досвід європейських країн створення системи електронних торгів дозволяє суттєво зменшити витрати державних коштів замовниками торгів на придбання предметів закупівлі, організацію та проведення закупівельного процесу; дає змогу безперешкодного, щодобового он-лайн доступу до інформації всіх зацікавлених осіб; забезпечує відкритість та прозорість на всіх стадіях закупівель; скорочує паперовий документообіг у цій сфері; створює рівний доступ до накопичуваної інформації та рівні умови конкуренції між постачальниками; полегшує роботу контролюючих та правоохоронних органів; попереджує зловживання та корупційні прояви у сфері закупівель тощо. Зазвичай, подібні системи автоматизації тендерних закупівель складаються з декількох модулів, серед яких можна виділити кабінет замовника, постачальника, та модуль аукціону. Мета даної роботи – створення серверної частини системи автоматизації електронних закупівель для приватного бізнесу.

Основний фактор у системі електронних закупівель – аукціон, а точніше його зворотня форма. Подібний тип прийнято називати «редукціоном», оскільки це аукціон, у якому перемагає найбільш економічно вигідна пропозиція. Згідно сайту Business Dictionary, зворотній аукціон – це «Тип аукціону, у якому декілька постачальників пропонують свої товари та змагаються за ціну, яка буде прийнятою для замовника. Замовник зазвичай має можливість прийняти будь-яку пропозицію або відхилити всі пропозиції» [2]. Наявність даного типу аукціону головний критерій при аналізі існуючих рішень на ринку.

Станом на даний момент існує декілька систем, які пропонують готові рішення для веб-сайтів пов’язаних з аукціоном. На прикладі існуючих систем PHP Pro Bid [3] та iLance [4], ми порівняємо розроблений модуль аукціону та виділим переваги нашої системи автоматизації для електронних закупівель. PHP Pro Bid – це комплексна система, яка пропонує повноцінний функціонал для роботи з користувачами, аукціонами та різними методами оплати товарів чи послуг. При створенні аукціону користувачу надається можливість вказувати деталі, такі як назва товару, опис, ціна, зображення, мінімальна ставка та інше. Також аукціон автоматично продовжується якщо ставку зроблено в останній момент перед закриттям. Створення нового аукціону відбувається через панель адміністратора, що являється недоліком, оскільки у електронних закупівлях аукціони створюються замовником. У PHP Pro Bid не існує функціоналу для реєстрації користувачів різних ролей – замовник та

постачальник, що було реалізовано у нашій системі автоматизації електронних закупівель. Виділення ролей користувачів з певним функціоналом, створення єдиного сервісу для авторизації та реєстрації, дозволило зробити нашу систему масштабованою.

У свою чергу система iLance включає у себе різновидність модулів для автоматизації аукціонів. За допомогою шаблонів та широкого обсягу налаштувань у системі можливо створювати аукціони для різних видів товарів та послуг. Однак у даному фреймворку відсутня можливість оголошувати аукціон відкритого типу, що безумовно є ключовим у електронних закупівлях. На відміну від закритого аукціону, спостерігачам нашої платформи надається можливість переглядати інформацію за пропозиціями до закупівлі та постачальниками. Таким чином ми забезпечуємо найбільш можливу прозорість закупівлі та відкриту конкуренцію.

Аналіз різновидностей аукціонів показує що існує декілька типів зворотнього відкритого аукціону, а саме: Голанський, Японський, Американський та Англійський [5]. Останній тип – найбільш поширений у системах електронних закупівель, тому в нашій статті ми зосередимося на проектуванні та реалізації даного виду аукціону за допомогою кінцевого автомату станів Stateless [6]. Зворотній Англійський аукціон дозволяє постачальникам напряму взаємодіяти із замовником. Такий аукціон припиняється коли залишається лише один постачальник або час проведення завершується. Нарешті перемагає пропозиція з найбільш вигідною ціною для замовника, але замовник також матиме можливість обрати переможця за іншими нецiновими критеріями.

На сервері також використовується стандарт RESTFul API [7] забезпечує сумісність системи з більшістю сучасних клієнтів та дозволяє працювати з сутностями через обмін моделей даних у форматі JSON.

1. Формування вимог до системи

Система має надавати можливість замовнику оголошувати закупівлі на певні товари або послуги, запрошувати до участі постачальників. Замовник зможе створювати лоти та позиції товарів, критерії для відбору пропозицій (цінові та нецінові). До кожної позиції замовник може вказувати галузь закупівлі та місце поставки товарів.

У свою чергу, система також повинна автоматично запрошувати перевірених постачальників у галузі, якщо користувач обрав таку можливість. Електронний документообіг та укладання договорів через електронний цифровий підпис також мають бути передбачені у системі. Замовники зможуть завантажувати документи до закупівлі, а також переглядати документи завантажені постачальником.

Замовники та постачальники зможуть спілкуватися та обговорювати умови закупівлі через систему обговорень. Закупівлі мають проводитися відкрито, тобто будь-хто зі спостерігачів матиме можливість зайти до системи та переглянути активні тендери.

Діаграми використання системи для таких ролей користувача, як замовник, постачальник, та спостерігач (не авторизований користувач) показано на рис. 1–3.

Розроблена діаграма використання відображає базовий набір функцій, які доступні не авторизованому користувачу системи автоматизації електронних закупівель, а саме спостерігачу.

Оскільки система призначена для зареєстрованих користувачів, то функціонал спостерігача обмежено переглядом існуючих закупівель, пошуку тендера за номером, ключовими словами або назвою. Таким чином, спостерігач має доступ до перегляду усіх наявних у системі закупівель, які він може фільтрувати за такими критеріями як галузь, бюджет, ЄДРПОУ замовника, статус, дата початку та завершення.

У спостерігача наявна можливість зареєструватися у системі як замовник для подальшої роботи та отримання повноцінного функціоналу.

Система орієнтована на потреби замовника в процесі автоматизації електронних закупівель, та надає такі можливості, як оголошення закупівлі, перегляд власних закупівель, перегляд пропозицій, обговорення та обмін документами, вибір переможця та підписання договору.

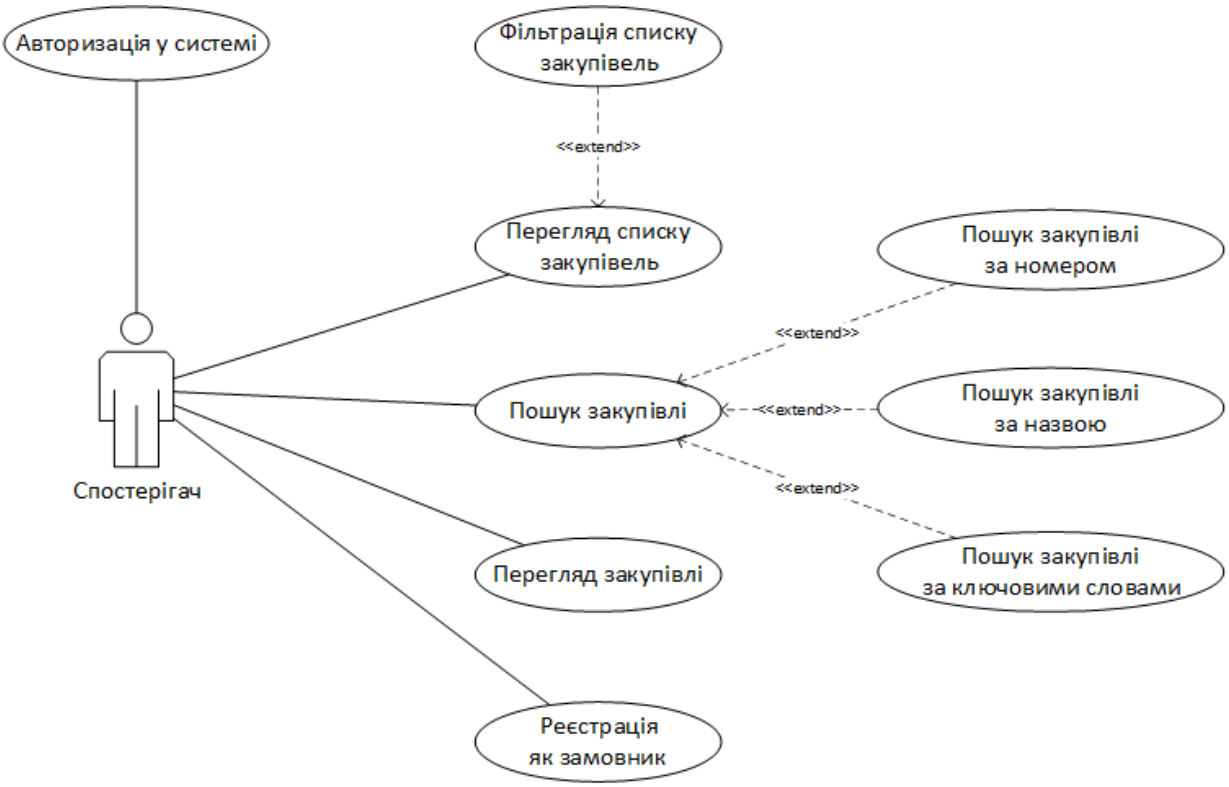


Рис. 1. Діаграма використання спостерігачем

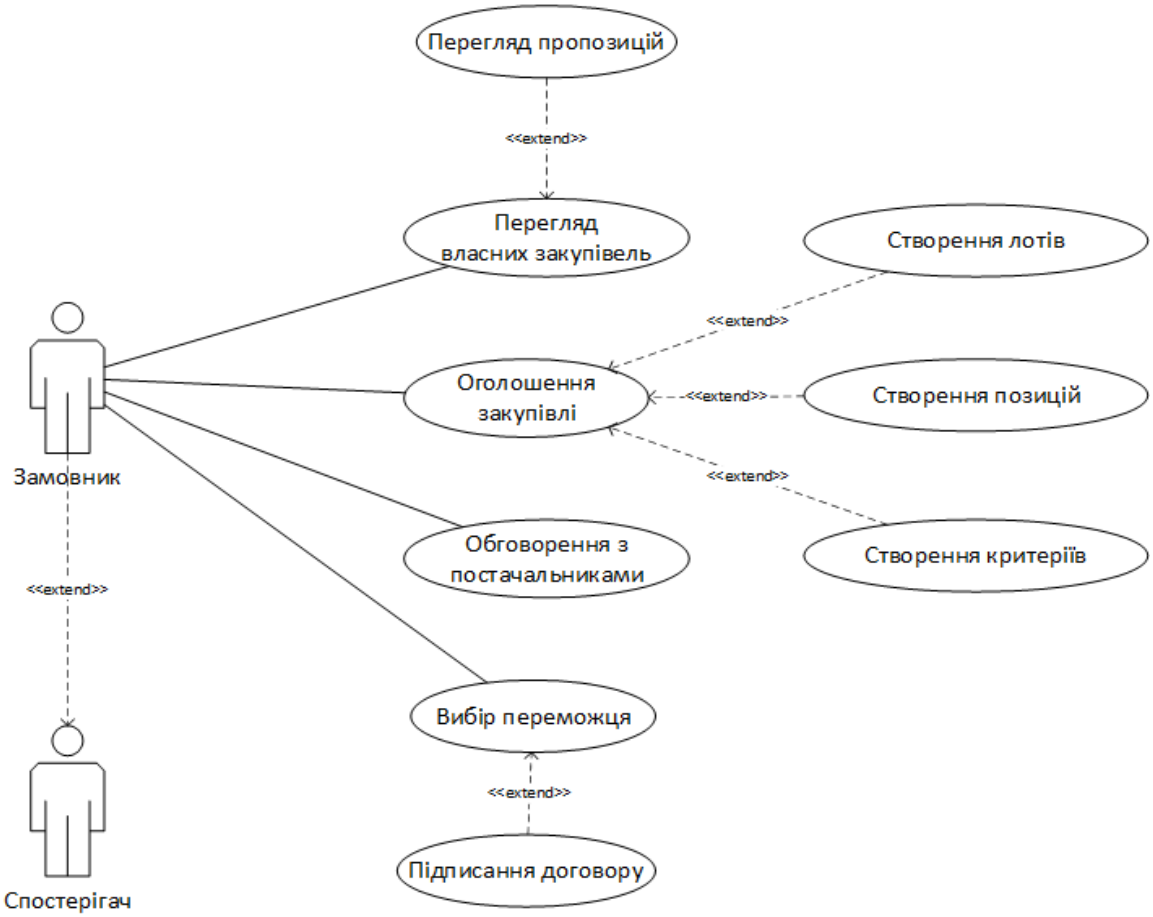


Рис. 2. Діаграма використання замовником

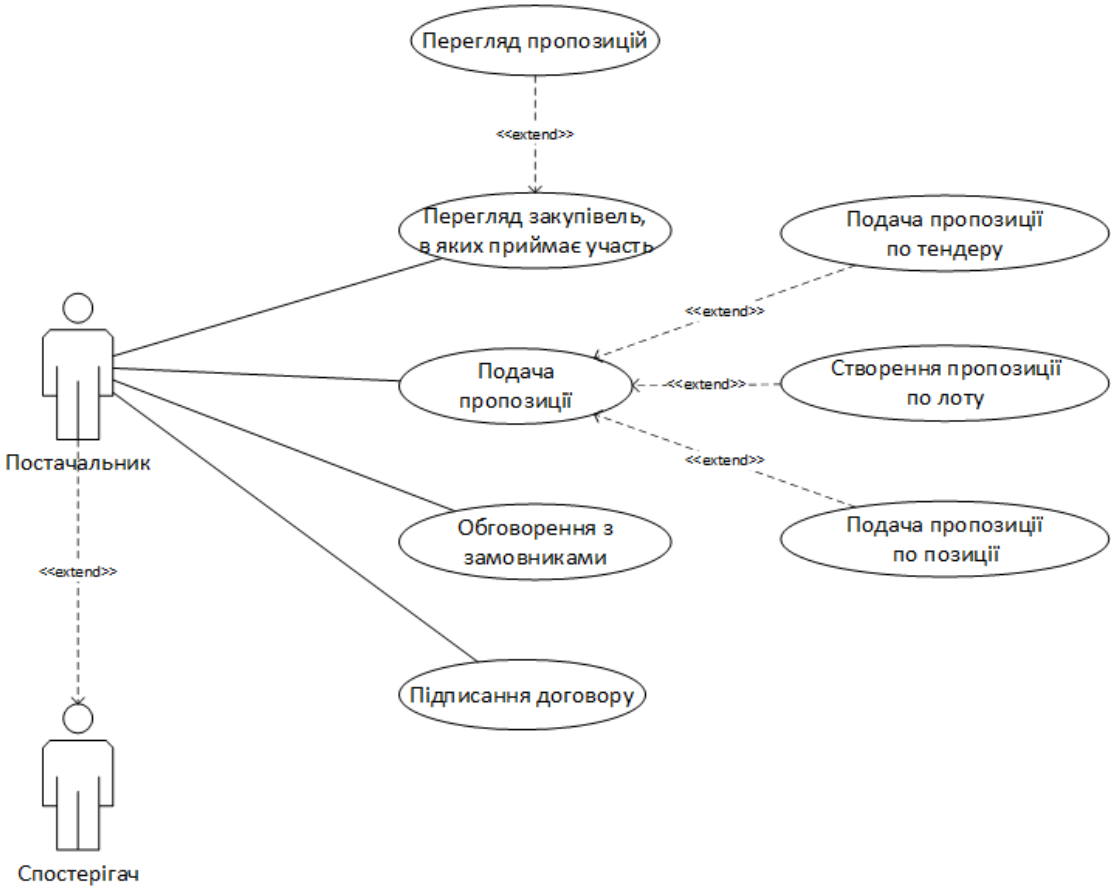


Рис. 3. Діаграма використання постачальником

Постачальник включає всі можливості спостерігача у системі та доповнює їх можливістю подавати пропозиції до закупівель, лоті та позицій, переглядати подані пропозиції, спілкуватися з замовником та підписувати контракт.

Після формування варіантів використання системи для різних ролей користувачів, було виділено основні сутності та атрибути для проектування рівня доступу до даних. Перелік сутностей та атрибутів наведено у табл. 1.

Таблиця 1. Сутності та атрибути

Сутність	Атрибути
Тендери	Статус, Назва, Опис, Бюджет, Валюта, ПДВ, Початок прийому пропозицій, Кінець прийому пропозицій, Інформація про компанію, Контактна особа, Тип закупівлі, Додаткові атрибути, Список нецінових показників, Список документів, Список критеріїв, Дата створення, Дата модифікації, Видалений
Лоти	Тендер, Етап, Назва, Опис, ПДВ, Статус, Додаткові атрибути, Список нецінових показників, Список документів, Список критеріїв, Дата створення, Дата модифікації, Видалений
Пропозиції	Тендер, Лот, Статус, Постачальник, Email, ЕДРПОУ, Назва компанії, Додаткові параметри, Видалена
Контракти	Пропозиція, Постачальник, Статус, Сума
Позиції	Лот, Назва, Опис, Кількість, Одиниці виміру, Додаткові атрибути, Список нецінових показників, Список документів, Список критеріїв, Дата створення, Дата модифікації, Виделено
Етапи	Номер етапу, JSON тендера

Проектування рівня доступу до даних починається з побудови логічної моделі БД. Незалежно від моделі даних, побудова логічної моделі БД на практиці виконується з урахуванням двох основних вимог: виключити надмірність і максимально підвищити якість даних. Ці вимоги випливають із вимоги колективного використання даних групою користувачів. На даному етапі потрібно перетворити об'єкти та зв'язки між ними у логічну модель даних – реляційну модель. В ній об'єкти та зв'язки між ними представлені у вигляді таблиць (відношень), які складаються із стовпців та рядків. Кожне поле має ім'я та тип. Тип задає спосіб представлення поля. Також визначимо для атрибутів чи обов'язково вони повинні містити дані, чи допускаються нульові значення. Для системи автоматизації електронних закупівель визначені наступні поля, значення, типи даних та обмеження, які наведені у таблицях 2–5.

Таблиця 2. Фізична модель закупівлі

Назва поля	Значення	Тип даних	Обмеження
1	2	3	4
Id	Ідентифікатор	BIGINT	PK, NN
Guid	Унікальний ідентифікатор	UNIQUEIDENTIFIER	NN
Status	Статус	INT	NN
Title	Назва закупівлі	NVARCHAR(1500)	NN
Description	Опис закупівлі	NVARCHAR(2500)	-
Budget	Бюджет закупівлі	MONEY DEFAULT (0)	-
CurrencyIso	Код валюти у форматі ISO	INT	NN
IsVat	Закупівля з ПДВ	BIT DEFAULT (0)	-
StartBid	Дата та час початку прийому пропозицій	DATETIMEOFFSET(7)	-
EndBid	Дата та час кінця прийому пропозицій	DATETIMEOFFSET(7)	-
InfoCompany	JSON інформації про компанію	NVARCHAR(MAX)	NN
ContactPerson	JSON інформації про контактну особу	NVARCHAR(MAX)	NN
TenderType	Тип закупівлі: аукціон, редукціон, запит цінових пропозицій	INT	NN
AdditionalAttributes	JSON будь-яких додаткових властивостей закупівлі (для підтримки нових підключень)	NVARCHAR(MAX)	-
ListFeatures	JSON списку нецінових показників	NVARCHAR(MAX)	-
ListDocuments	JSON списку документів до закупівлі	NVARCHAR(MAX)	-
ListCriteriaGroups	JSON критеріїв закупівлі, встановлених замовником	NVARCHAR(MAX)	-
DateCreate	Дата оголошення закупівлі	DATETIMEOFFSET(7) DEFAULT (SYSDATETIMEOFFSET())	NN
DateModified	Дата внесення змін до закупівлі	DATETIMEOFFSET(7) DEFAULT (SYSDATETIMEOFFSET())	-
IsRemoved	Чи було видалено закупівлю із системи	BIT DEFAULT (0)	-

Таблиця 3. Фізична модель лотів закупівлі

Назва поля	Значення	Тип даних	Обмеження
1	2	3	4
Id	Ідентифікатор	BIGINT	PK, NN
TenderId	Ідентифікатор пов'язаного тендера	BIGINT	FK, NN
Guid	Унікальний ідентифікатор	UNIQUEIDENTIFIER	NN
Status	Статус	INT	NN
Title	Назва лоту	NVARCHAR(1500)	NN
Description	Опис лоту	NVARCHAR(2500)	-
IsVat	Закупівля з ПДВ	BIT DEFAULT (0)	-
AdditionalAttributes	JSON будь-яких додаткових властивостей лота (для підтримки нових підключень)	NVARCHAR(MAX)	-
ListFeatures	JSON списку нецінових показників	NVARCHAR(MAX)	-
ListDocuments	JSON списку документів до лота	NVARCHAR(MAX)	-
ListCriteriaGroups	JSON критеріїв лота, встановлених замовником	NVARCHAR(MAX)	-
DateCreate	Дата створення лота	DATETIMEOFFSET(7) DEFAULT (SYSDATETIMEOFFSET())	NN
DateModified	Дата внесення змін до лота	DATETIMEOFFSET(7) DEFAULT (SYSDATETIMEOFFSET())	-
IsRemoved	Чи було видалено лот із системи	BIT DEFAULT (0)	-

Таблиця 4. Фізична модель пропозицій до закупівлі

Назва поля	Значення	Тип даних	Обмеження
1	2	3	4
Id	Ідентифікатор	BIGINT	PK, NN
StageId	Ідентифікатор пов'язаного етапу	BIGINT	FK, NN
TenderId	Ідентифікатор пов'язаного тендеру	BIGINT	FK, NN
LotId	Ідентифікатор пов'язаного лоту	BIGINT	FK, NN
Guid	Унікальний ідентифікатор	UNIQUEIDENTIFIER	NN
Status	Статус пропозиції	INT	NN
UserId	Ідентифікатор постачальника	BIGINT	NN
Email	Email постачальника	NVARCHAR(50)	NN
Edrpou	Код постачальника у реєстрі	NVARCHAR(15)	NN
CompanyName	Назва компанії постачальника	NVARCHAR(50)	NN
BidParameters	JSON списку додаткових параметрів закупівлі	NVARCHAR(MAX)	-

1	2	3	4
DateCreate	Дата створення пропозиції	DATETIMEOFFSET(7) DEFAULT (SYSDATETIMEOFFSET())	NN
DateModified	Дата внесення змін до пропозиції	DATETIMEOFFSET(7) DEFAULT (SYSDATETIMEOFFSET())	-
IsRemoved	Чи було видалено пропозицію із системи	BIT DEFAULT (0)	-

Таблиця 5. Фізична модель контрактів для закупівлі

Назва поля	Значення	Тип даних	Обмеження
Id	Ідентифікатор	BIGINT	PK, NN
StageId	Ідентифікатор пов’язаного етапу	BIGINT	FK, NN
TenderId	Ідентифікатор пов’язаного тендеру	BIGINT	FK, NN
LotId	Ідентифікатор пов’язаного лоту	BIGINT	FK, NN
ItemId	Ідентифікатор пов’язаної позиції	BIGINT	FK, NN
BidId	Ідентифікатор пов’язаної пропозиції	BIGINT	FK, NN
UserId	Ідентифікатор постачальника	BIGINT	FK, NN
Status	Статус договору	INT	NN
Rate	Сума договору	DECIMAL	NN

Додатково були розроблені збережені процедури для створення та оновлення пропозицій, отримання списку пропозицій по закупівлі, лоту, позиції, створення та редагування контрактів, отримання контрактів закупівлі. Первинні та зовнішні ключі забезпечують цілісність даних у БД. У таблицях 6 та 7 представлено фізичну модель товарів та етапів закупівлі відповідно.

Таблиця 6. Фізична модель товарів для закупівлі

Назва поля	Значення	Тип даних	Обмеження
1	2	3	4
Id	Ідентифікатор	BIGINT	PK, NN
LotId	Ідентифікатор пов’язаного лоту	BIGINT	FK, NN
Guid	Унікальний ідентифікатор	UNIQUEIDENTIFIER	NN
Title	Назва позиції	NVARCHAR(1500)	NN
Description	Опис позиції	NVARCHAR(2500)	-
Quantity	Кількість товарів у позиції	FLOAT DEFAULT (0)	NN
UnitId	Ідентифікатор одиниць вимірювання	INT	NN
AdditionalAttributes	JSON будь-яких додаткових властивостей позиції (для підтримки нових підключень)	NVARCHAR(MAX)	-
ListFeatures	JSON списку нецінових показників	NVARCHAR(MAX)	-
ListDocuments	JSON списку документів до позиції	NVARCHAR(MAX)	-

1	2	3	4
ListCriteriaGroups	JSON критеріїв лота, встановлених замовником	NVARCHAR(MAX)	-
DateCreate	Дата створення позиції	DATETIMEOFFSET(7) DEFAULT (SYSDATETIMEOFFSET())	NN
DateModified	Дата внесення змін до позиції	DATETIMEOFFSET(7) DEFAULT (SYSDATETIMEOFFSET())	-
IsRemoved	Чи було видалено позицію із системи	BIT DEFAULT (0)	-

Таблиця 7. Фізична модель етапів закупівлі

Назва поля	Значення	Тип даних	Обмеження
Id	Ідентифікатор	BIGINT	PK, NN
StageNumber	Номер етапу закупівлі	INT	NN
TenderId	Ідентифікатор пов’язаного тендеру	BIGINT	FK, NN
TenderJson	JSON інформації про закупівлю на минулому етапі	NVARCHAR(MAX)	-

2. Реалізація бізнес-логіки

У системі бізнес-логіка зосереджена в мікросервісах, які спілкуються з клієнтом у форматі обміну даних JSON. Мікросервіси викликаються через контролери-маршрутизатори, що робить її закритими від зовнішнього світу. Крім цього при запитах на модифікацію даних передаються відповідні заголовки авторизації для уникнення несанкціонованого доступу.

Сервіс TenderService призначено для роботи з закупівлями, наприклад, створення нової закупівлі, отримання існуючої закупівлі, отримання документів, видалення закупівлі. Клас сервісу складається з таких методів:

- UpsertTender. Метод виконує логіку створення нової закупівлі або оновлення вже існуючої. Закупівлю може створити лише замовник, який зареєстрований у системі та авторизований. Вхідним параметром є об’єкт типу TenderDTO, який містить дані для коректного створення закупівлі. Валідація параметру проводиться на контролері, тому сервіс лише викликає збережену процедуру для створення або оновлення закупівлі;
- DeleteTender. Метод виконує логіку видалення закупівлі за ідентифікатором. Для забезпечення можливості перегляду архівних закупівель, дані із бази не видаляються, замість цього у базі даних за допомогою збереженої процедури встановлюється поле isRemove для закупівлі у значення true;
- GetTenderById. Метод повертає закупівлю за ідентифікатором. Викликається збережена процедура, яка збирає із бази даних інформацію по тендеру та повертає результат у форматі JSON. Після чого метод за допомогою конверторів створює модель закупівлі TenderDTO та повертає її на контролер, який потім повертає цю модель клієнту;
- GetTenderDocumentsById. Метод містить логіку для отримання списку документів до закупівлі. Викликає збережену процедуру, що повертає документи за типом об’єкта (тендер, лот, позиція) з параметром тендер. Результат збереженої процедури оброблюється через конвертор і створюється список DocumentDTO, який містить інформацію по кожному документу: назва, тип, посилання, дата створення. Список повертається контролеру, який повертає його клієнтові.

За аналогічною схемою реалізовано класи для роботи з лотами (LotService) та позиціями (ItemService) закупівлі.

Патерн Dependency Injection [8] дуже широко використовується в даному проекті. Як ціль було прийнято відійти від

використання конкретними класами, а користуватися лише інтерфейсами, зменшити використання ключового слова new та полегшити тестування проекту. Для цих цілей використано стандартний механізм DI в .NET Core. Вибір обгрунтовується зрозумілим API, розгорнутою та повною документацією. В даному проекті використано стандартні механізми фреймворку .NET Core для реєстрації контролерів та підтримки режиму часу життя сутностей на протязі одного запиту до WebApi. Це дає нам можливість розробляти бізнес-логіку не звертаючи увагу на час життя сутності, тому що він визначається в реєстрації сутності.

Сервіс StateService призначено для роботи з статусами та етапами закупівель, переведення закупівлі на новий етап, зміну статуса. Клас складається з наступних методів:

- GetStateByTenderId. Метод містить логіку для отримання статуса закупівлі за ідентифікатором. Викликає відповідну збережену процедуру, яка знаходить закупівлю у БД і повертає її статус;
- ChangeStateTender. Метод виконує логіку по зміні статуса тендера на вказаний. Викликає збережену процедуру, яка змінює статус для закупівлі у БД на вказаний;
- MoveToNextStage. Метод переводить закупівлю за ідентифікатором на наступний етап (наприклад на підписання контракту). Викликається збережена процедура, яка перевіряє поточний етап тендера та змінює його на наступний.

Для реалізації станів закупівлі та переходу між ними використано фреймворк Stateless та Hangfire. Клас StateService використовується машиною станів TenderStateMachine. Спочатку створено базовий клас для кінцевого автомату BaseStateMachine:

```
internal abstract class BaseStateMachine: BaseService
  {
    public readonly IStateService _stateService;
    public BaseStateMachine(IStateService
stateService,IPublishService
publishService):base(publishService)
    {
      _stateService = stateService;
    }
    public abstract object ChangeState(object model);
  }
```

За допомогою механізму наслідування було створено реалізацію:

```
internal class TenderStateMachine : BaseStateMachine
  {
…
    // <summary>
    /// Configuration state
    /// </summary>
    private void ConfigureStateMachine()
    {
      _machine = new
StateMachine<TenderStateMachineStatus,
TenderStateMachineTrigger>(() =>
_TenderStateMachineStatus, s =>
_TenderStateMachineStatus = s);
      _triggerParams =
_machine.SetTriggerParameters<TenderSync>(Tender
StateMachineTrigger.ChangeStatus);


_machine.Configure(TenderStateMachineStatus.New)

.PermitIf(TenderStateMachineTrigger.ChangeStatus,
TenderStateMachineStatus.Planned);


_machine.Configure(TenderStateMachineStatus.Planne
d)

.Permit(TenderStateMachineTrigger.ChangeStatus,
TenderStateMachineStatus.AcceptanceOfOffers)
      .OnEntryFrom(_triggerParams, model =>
AcceptanceOfOffers(model));


_machine.Configure(TenderStateMachineStatus.Accep
tanceOfOffers)

.PermitTenderStateMachineTrigger.ChangeStatus,
TenderStateMachineStatus.DataProcessing)
      .OnEntryFrom(_triggerParams, model =>
AcceptanceOfOffers(model));


_machine.Configure(TenderStateMachineStatus.DataP
rocessing)

.Permit(TenderStateMachineTrigger.ChangeStatus,
TenderStateMachineStatus.Completed)
      .OnEntryFrom(_triggerParams, model =>
DataProcessing(model));
```

```
_machine.Configure(TenderStateMachineStatus.Compl
eted)
            .OnEntryFrom(_triggerParams, model =>
Task.Run(async () => await PushAwards(model)))
            .OnEntryFrom(_triggerParams, model =>
Task.Run(async () => await UpdateState(model)));

        }

        /// <summary>
        /// Changed Status
        /// </summary>
        /// <param name="state"></param>
        private async Task UpdateState(Tender model)
        {
            model.Status = (int)_machine.State;
            var result = await
_stateService.InsertOrUpdate(new StateTender()
            {
                TenderId = model.Id,
                StageNumber = model.StageNumber,
                StatusId = (int)_machine.State,
                JobId = _currentTenderState?.JobId,
                TenderJson =
JsonConvert.SerializeObject(model)
            });

            await
_stateService.UpdateStatusTender(model);
        }
…
}
```

Стани закупівлі можна побачити у перерахуванні TenderStateMachineStatus:

```
public enum SmartStateMachineStatus
    {
        New=0, // Нова закупівля
        Planned=1, // Заплановані торги
        AcceptanceOfOffers = 2, // Подача пропозицій
        DataProcessing = 3, // Аналіз пропозицій
        Completed =4 // Завершено
    }
```

Таким чином реалізовано кінцевий автомат станів для закупівлі.

3. Дизайн Web API

За допомогою онлайн інструменту для проектування та документації API – SwaggerEditor [9] згенеровано документацію основних методів та моделей. Інструмент призначений для конструювання API та автоматичної генерації шаблонного коду для основних мов програмування: C#, JavaSript, Python, Swift.

Web API реалізовано на основі технологій ASP .NET Core MVC [10], контролери виконують роль маршрутизаторів та перенаправляють запити на відповідні мікросервіси [11]. У табл. 8 наведено перелік основних методів дії для закупівлі.

Таблиця 8. Перелік основних методів Web API

Шлях	Опис	Повертаєме значення
-GET- Tender/GetList	Отримати список закупівель	StatusCode 200 – OK + список TenderDTO StatusCode 204 – NoContent StatusCode 500 – помилка сервера
-GET- Tender/GetTenderById/{tenderId}	Отримати закупівлю за ідентифікатором	StatusCode 200 – OK + TenderDTO StatusCode 404 – NotFound StatusCode 500 – помилка сервера
-POST-Tender/Upsert	Додати нову закупівлю або оновити існуючу	StatusCode 201 – Created StatusCode 403 – Forbidden (немає доступу) StatusCode 500 – помилка сервера
-DEL-Tender/Delete/{tenderId}	Видалити закупівлю	StatusCode 200 – OK StatusCode 403 – Forbidden (немає доступу) StatusCode 500 – помилка сервера
-PATCH-TenderState/ChangeState/{tenderId/{stateId}	Змінити статус тендера на заданий	StatusCode 200 – Created StatusCode 404 – Not Found StatusCode 500 – помилка сервера
-PATCH-TenderState/MoveToNextStage	Перевести закупівлю на наступний етап	StatusCode 200 – OK StatusCode 404 – Not Found StatusCode 500 – помилка сервера

В ході розробки було прийняте рішення викликати валідацію у методах контролерів на сервері – у місці, де дані першими приходять від клієнта. Таким чином забезпечується коректність даних з самого початку їх обробки.

Оскільки перенавантажувати моделі даних логікою валідації вважається недоцільним, була розроблена власна система валідації на основі методів розширення для моделей та набору правил. Кожна модель у системі матиме метод IsValid, який буде викликати всі правила валідації для моделі, повертатиме булеве значення успішності валідації, а також масив помилок у форматі ключ-значення, де ключ – це назва поля моделі, а значення – опис помилки. У разі неуспішної валідації клієнту буде повертатися масив помилок, що буде оброблено та відповідним чином відображено у браузері.

Подібний підхід до процесу валідації дозволив розподілити відповідальність, очистити модель від непотрібних атрибутів або наслідування та покращити надійність програми.

4. Тестування

Тестування – це невд'ємна складова впровадження будь-якої програмної системи, тому під час розробки приділена значна увага покриттю коду тестами для швидкого виявлення та виправлення помилок, а також постійного контролю за роботоздатністю продукту. Під час тестування широко використовувались такі засоби, як unit- та mock-тести.

Для модульного тестування використовувалась бібліотека xUnit [12]. Це безкоштовний інструмент, що базується на відкритому до змін коді, який написано розробниками NUnit v2. Даний фреймворк призначено для тестування коду на C#, F#, VB.NET та інших мовах програмування .NET.

У фреймворку існує два поняття тестів: Fact та Theory. Fact – це окремий модульний тест, що не приймає параметрів. Theory – це тест, який приймає параметри та може містити декілька сценаріїв. Параметри до методу тесту можуть передаватися двома способами: через атрибути InlineData та MemberData.

За допомогою модульного тестування була перевірена робота системи валідації моделей, створені тестові методи для валідації кожної сутності у програмі. У кожному тесті на валідацію об'явлено два варіанта моделі: валідна та невалідна, а також два Theory-методи, які перевіряють валідацію. Також у даному тесті присутній Fact-метод, який перевіряє валідацію null-моделі.

За допомогою фреймворка Moq у системі розроблені тести для сервісів по роботі з закупівлями, лотами та позиціями. Для того, щоб перевірити CRUD-операції з основними сутностями системи створено мок-об'єкти відповідних сервісів-залежностей.

Moq – це фреймворк для створення моків для .NET, який розроблено з підтримкою специфічних можливостей .NET, таких як вирази LINQ та лямбди. Все це робить бібліотеку найбільш продуктивною, строго типізованою та дружньою до рефакторингу серед аналогів на ринку.

Наприклад, TenderController відповідає за основні операції для роботи із закупівлями, у нього є залежність – поле ITenderService, у якому знаходиться бізнес-логіка роботи з БД. Замість того, щоб при тестуванні створювати справжній об'єкт, ми створюємо мок ITenderService, який може працювати з тестовою базою даних або просто повертати потрібний результат. Після чого вже з використанням xUnit були написані модульні тести, які використовували створені у Moq моки сервісів.

Спираючись на результати проведених випробувань, можна зробити висновок про надійність системи та окремих її модулів. Відсоток покриття тестами становить 95 %, що підтверджує високу стійкість системи до збоїв.

Висновки

Реалізовано програмний засіб для автоматизації електронних закупівель з використанням новітніх технологій RESTful API та відкритих специфікацій

OpenAPI v3.0. Надано можливість авторизувати користувачів системи, а саме постачальників та замовників основуючись на стандарті Oauth. Бібліотека для роботи з розподіленим кешем Redis використовувалась для прискорення роботи системи та кешування даних на рівні доступу до БД. Для моніторингу стану закупівлі та переходу між станами реалізовано кінечний автомат станім за допомогою бібліотеки Stateless. Тестування розробленої системи проведено unit- та integration тестами. На відміну від комплексних систем автоматизації аукціонів по типу PHP Pro Bid, у яких клієнт є невідє'мною частиною сервера, що робить використання сервера неможливим для інших клієнтів (наприклад, мобільних додатків). У нашій системі електронних закупівель кожен модуль реалізовано як RESTFul API мікросервіс. Такий підхід до проектування серверної частини дозволив нам розподілити відповідальності між модулями та відділити сервер від клієнта. Використовуючи стандартний формат даних JSON, будь-який клієнт – веб-сайт або мобільний додаток, зможе повноцінно працювати з нашим API.

Література

1. Doroshenko A.Yu., Bodak B.V. (2019) .The impact and unforeseen challenges of E-procurement systems in Canada. Winter InfoCom Advanced Solutions. P. 9–10.
2. Business Dictionary, Reverse auction [Online] – Available from: https://financial-dictionary.thefreedictionary.com/Reverse+auction.
3. PHP Pro Bid Auction features [Online] – Available from: https://www.phpprobid.com/features/auctions.
4. iLance Auction Software [Online] – Available from: https://www.ilance.com/auction-software/.
5. Mubark A., Haggren A. (2017). Computer Science Optimization Of Reverse Auctions.
6. Stateless 3.0 – A State Machine library for .NET Core [Online] – Available from: https://www.hanselman.com/blog/stateless-30-a-state-machine-library-for-net-core/.
7. What is a RESTful API [Online] – Available from: https://searchapparchitecture.techtarget.com/definition/RESTful-API
8. Architectural Styles and the Design of Network-based Software Architectures [Online] – Available from: https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm.
9. About Swagger specification [Online] – Available from: https://swagger.io/docs/specification/about/.
10. Repository and UnitOfWork in ASP.NET Core [Online] – Available from: http://www.c-sharpcorner.com/article/repository-pattern-in-asp-net-core/
11. Doroshenko A.Yu., Bodak B.V. (2020) The implementation of RESTful API for social media events platform. Summer InfoCom Advanced Solutions. P. 11–12.
12. Build, test, and deploy .NET Core apps [Online] – Available from: https://docs.microsoft.com/en-us/azure/devops/pipelines/ecosystems/dotnet-core?view=azure-devops.

References

1. Doroshenko A.Yu., Bodak B.V. (2019) .The impact and unforeseen challenges of E-procurement systems in Canada. Winter InfoCom Advanced Solutions. P. 9–10.
2. Business Dictionary, Reverse auction [Online] – Available from: https://financial-dictionary.thefreedictionary.com/Reverse+auction.
3. PHP Pro Bid Auction features [Online] – Available from: https://www.phpprobid.com/features/auctions.
4. iLance Auction Software [Online] – Available from: https://www.ilance.com/auction-software/.
5. Mubark A., Haggren A. (2017). Computer Science Optimization Of Reverse Auctions.
6. Stateless 3.0 – A State Machine library for .NET Core [Online] – Available from: https://www.hanselman.com/blog/stateless-30-a-state-machine-library-for-net-core/.
7. What is a RESTful API [Online] – Available from: https://searchapparchitecture.techtarget.com/definition/RESTful-API

8. Architectural Styles and the Design of Network-based Software Architectures [Online] – Available from: https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm.
9. About Swagger specification [Online] – Available from: https://swagger.io/docs/specification/about/.
10. Repository and UnitOfWork in ASP.NET Core [Online] – Available from: http://www.c-sharpcorner.com/ article/repository-pattern-in-asp-net-core/
11. Doroshenko A.Yu., Bodak B.V. (2020) The implementation of RESTful API for social media events platform. Summer InfoCom Advanced Solutions. P. 11–12.
12. Build, test, and deploy .NET Core apps [Online] – Available from: https://docs.microsoft.com/en-us/azure/devops/pipelines/ecosystems/dotnet-core?view=azure-devops.

Одержано 20.01.2021

***Про авторів*:**

Дорошенко Анатолій Юхимович,
доктор фізико-математичних наук,
професор, завідувач відділу теорії
комп'ютерних обчислень Інституту
програмних систем НАН України,
професор кафедри автоматики і
управління в технічних системах НТУУ
"КПІ імені Ігоря Сікорського".
Кількість наукових публікацій в
українських виданнях – понад 190.
Кількість наукових публікацій в
іноземних виданнях – понад 80.
Індекс Гірша – 6.
http://orcid.org/0000-0002-8435-1451,

Бодак Богдан Вікторович,
аспірант кафедри автоматики і управління
в технічних системах НТУУ
"КПІ імені Ігоря Сікорського".
Кількість наукових публікацій в
українських виданнях – 2.

Місце роботи авторів:

Інститут програмних систем
НАН України,
03187, м. Київ-187,
проспект Академіка Глушкова, 40.
Тел.: (044) 526 3559.
E-mail: doroshenkoanatoliy2@gmail.com,
bohdan.bodak@outlook.com

УДК 004.4'22 https://doi.org/10.15407/pp2021.01.016

А.Ю. Дорошенко, І.З. Ашур

ЗАСТОСУВАННЯ ЗАСОБІВ НЕЙРОЕВОЛЮЦІЇ В ТЕХНІЧНИХ СИСТЕМАХ АВТОМАТИЗАЦІЇ КЕРУВАННЯ

В роботі запропоновано нове застосування техніки нейроеволюції наростаючих топологій для розв'язування задач автоматизації керування на одному з прикладів добре відомих стандартних та популярних задач керування технічними системами, де використовується навчання з підкріпленням. Використовується відкритий інструментарій для розробки та порівняння алгоритмів навчання з підкріпленням OpenAI Gym, повноцінна реалізація з відкритим програмним кодом генетичного алгоритму нейроеволюції NEAT під назвою SharpNEAT, та проміжне програмне забезпечення для оркестрації зазначених компонентів.

Ключові слова: штучні нейронні мережі, навчання з підкріпленням, генетичні алгоритми, автоматизація керування в технічних системах.

Вступ

В сучасній інженерній науці все більш важливими стають методи боротьби з невизначеністю, серед яких значне місце займають штучні нейронні мережі та методи навчання. Навчання з підкріпленням – галузь машинного навчання, що базується на тому, як програмним агентам слід виконувати дії у середовищі з метою максимізації поняття кумулятивної винагороди. Навчання з підкріпленням – це одна з трьох основних парадигм машинного навчання разом з контрольованим навчанням та навчанням без вчителя. Навчання з підкріпленням відрізняється від контрольованого навчання відсутністю необхідності розмічених вхідних та вихідних даних та відсутністю необхідності явного коригування субоптимальних дій. Натомість навчання з підкріпленням фокусується на пошуку балансу між дослідженням «невідомої» території та використанням набутих знань. Середовище для навчання з підкріпленням зазвичай описується у формі марківського процесу рішень, адже багато алгоритмів навчання з підкріпленням використовують техніки динамічного програмування. Головна різниця між класичними методами динамічного програмування та алгоритмами навчання з підкріпленням полягає у тому, що останні не оперують знаннями про чітку математичну модель марківського процесу рішень та орієнтуються на великі марківські процеси, в яких застосування точних методів стає неможливим. Завдяки рівню абстракції навчання з підкріпленням, воно використовується у ряді галузей знань, як теорія ігор, теорія керування, дослідження операцій, теорія інформації, оптимізація на основі симуляції, багатоагентні системи, колективний інтелект, статистика тощо. У літературі, присвяченій дослідженню операцій та теорії керування, навчання з підкріпленням зазвичай називається наближеним динамічним програмуванням або *нейродинамічним програмуванням* [1, 2].

У цій роботі використовується метод нейроеволюції наростаючих топологій (NEAT) [3], що являє собою генетичний алгоритм для генерації еволюціонуючих штучних нейронних мереж, заснований на трьох ключових принципах: спостереження (трекінг) генів з історичними відмітками для схрещування (кросинговеру) між топологіями, використання поняття еволюції видів для підтримання інновацій та інкрементальний розвиток топологій, починаючи з простих початкових структур. Алгоритм NEAT дозволяє використовувати генетичні алгоритми для визначення кращої і мінімально необхідної топології нейронної мережі.

У роботі використано також OpenAI Gym [4] – набір інструментів для розробки та порівняння алгоритмів навчання з підкріпленням. Gym являє собою колекцію тестових задач-середовищ, що можуть бути використані у користувацьких алгоритмах навчання з підкріпленням. Ці середо-

вища мають спільний інтерфейс, що дає змогу користувачам використовувати загальні, спільні підходи до розробки власних алгоритмів.

Суть даної роботи полягає у розробці й дослідженні можливості та ефективності застосування техніки нейроеволюції наростаючих топологій на прикладі однієї з стандартних та популярних задач навчання з підкріпленням з використанням OpenAI Gym, повноцінної імплементації з відкритим програмним кодом алгоритму NEAT під назвою SharpNEAT [5], та проміжного програмного забезпечення для оркестрації зазначених компонентів.

Навчання з підкріпленням

Навчання з підкріпленням – це галузь машинного навчання, де програмним агентам слід виконувати дії у середовищі з метою максимізації поняття кумулятивної винагороди. Навчання з підкріпленням є однією з трьох основних парадигм машинного навчання разом з контрольованим навчанням та навчанням без вчителя.

Навчання з підкріпленням відрізняється від контрольованого навчання у відсутності необхідності розмічених вхідних та вихідних даних та відсутності необхідності явного коригування субоптимальних дій. Натомість навчання з підкріпленням фокусується на пошуку балансу між дослідженням «невідомої» території та використанням набутих знань.

Середовище для навчання з підкріпленням зазвичай описується у формі марківського процесу рішень, адже багато алгоритмів навчання з підкріпленням використовують техніки динамічного програмування.

Типовий алгоритм навчання з підкріпленням моделюється марківським процесом рішень [6], де ми маємо:

- набір станів середовища та агента S;
- набір дій агента A;
- вірогідність переходу (у момент t) з одного стану до іншого при виконанні дії $a \in A$;
- негайну винагороду після переходу з одного стану до іншого при виконанні дії $a \in A$.

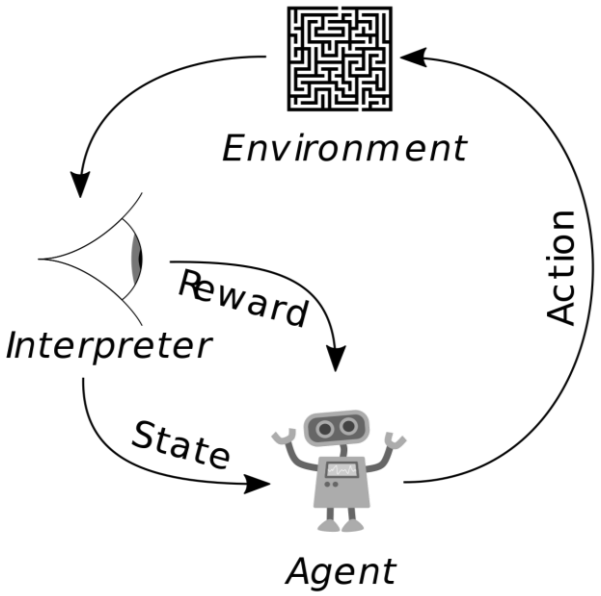


Рис. 1. Схема взаємодії компонентів в алгоритмі навчання з підкріпленням

Агент алгоритму навчання з підкріпленням взаємодіє з середовищем у дискретні проміжки часу. Для кожного проміжку часу агент отримує поточний стан та винагороду. Після цього, агент обирає дію з набору доступних дій (для дискретного простору дій) або значення (для неперервного простору дій), що, в свою чергу, надсилається до середовища. Середовище переходить у новий стан та визначається винагорода для переходу. Задача агента навчання з підкріпленням полягає у вивченні політики, що максимізує очікувану кумулятивну винагороду.

Застосування алгоритму NEAT

Нейроеволюція наростаючих топологій (NEAT) це – генетичний алгоритм для генерації еволюціонуючих штучних нейронних мереж, що встановлює параметри зв'язків між нейронами та структури мереж у спробах знаходження балансу між придатністю еволюціонуючих рішень та їх різноманіттям.

Традиційно, топологія нейромережі обирається людиною, що проводить експеримент, а вдалі значення вагів зв'яз-

ків між нейронами отримуються в процесі навчання. Це призводить до ситуації, де необхідно застосування підходу проб та помилок для знаходження вдалої топології для даної задачі. NEAT – приклад алгоритму, що еволюціонує і топологію, і значення вагів зв'язків штучної нейронної мережі одночасно.

Для кодування нейронної мережі у фенотип для генетичного алгоритму, NEAT використовує пряму схему кодування, в такій схемі явно представлений кожен зв'язок та нейрон.

Підхід NEAT починається з мережі, подібної до перцептрону, з лише вхідним та вихідним прошарком нейронів. З плином дискретних кроків еволюції, складність топології нейронної мережі може зростати за рахунок створення нових нейронів у шляхах зв'язків або за рахунок створення нових зв'язків між попередньо роз’єднаними нейронами.

Проблема договору про конкуренцію з'являється, коли існує декілька способів представлення інформації у фенотипі. Наприклад, якщо геном містить нейрони A, B та C, що представлені саме у цій послідовності, та схрещується з ідентичним за функціональністю, проте різним за представленням геномом [C B A], кросинговер створить виплід, якому бракує інформації, як [A B A] або [C B C]. NEAT вирішує цю проблему за рахунок трекінгу історії генів з використанням глобального числа інновацій, що збільшується з утворенням нових генів. З додаванням нових генів, глобальне число інновацій інкрементується та присвоюється цьому гену. Таким чином, новіші гени мають більші значення маркеру глобального числа інновацій. Для окремо взятого покоління, при виникненні ідентичної мутації у декількох геномах, всім таким геномам присвоюється одне і те ж значення маркеру глобального числа інновацій. Ці маркери числа інновацій дозволяють алгоритму обирати гени для кросинговеру [7].

SharpNEAT представляє собою імплементацію еволюційного алгоритму, а саме алгоритму еволюції нейронних мереж. Еволюційний алгоритм використовує еволюційні механізми мутації, рекомбінації та відбору для пошуку нейронних мереж, функціонал та поведінка котрих задовільнила б попередньо визначену задачу.

Фреймворк SharpNEAT надає набір вбудованих задач-прикладів для демонстрації роботи алгоритму еволюції. Імплементація SharpNEAT є модульною, що дозволяє легко вносити зміни, розширення та повторно використовувати її частини. SharpNEAT створений, зокрема, для задоволення інтересів питань біологічної еволюції та границь нейроеволюції у зрізі рівня складності проблем, рішення яких можуть запропонувати алгоритми нейроеволюції.

Існує дві основні концепції навчання з підкріпленням: середовище (навколишній світ) та агент (алгоритм). Агент надсилає команди до середовища, а середовище відповідає спостереженнями та винагородами. Головний інтерфейс gym стандартизований та універсальний, інтерфейси для агентів відсутні.

Основні методи середовища OpenAI Gym:

- *reset(self):*

скидання стану середовища. Повертає спостереження.

- *step(self, action):*

симулювати середовище на один часовий крок вперед. Повертає спостереження, винагороду та флаг закінчення епізоду.

- *render(self, mode='human'):*

повертає один кадр середовища.

OpenAI Gym не оперує будь-якими припущеннями щодо структури агенту.

У даній роботі розглядається середовище BipedalWalker-v3 [8], просте середовище з роботом, що крокує (див. рис. 2) Робот має чотири шарніри для його кінцівок.

Середовище існує у двох реалізаціях:

- “Normal”, з легкими нерівностями поверхні;
- “Hardcore”, зі сходами, пеньками та ямами.

Винагорода видається за рух вперед. Найвіддаленіші точки середовища відповідають 300 і більше балам винагороди. При падінні робот отримує штраф у 100 балів. Застосування скрутного моменту до шарнірів також має відносно невелику вартість у балах винагороди, а отже ефективніші агенти отримують більші винагороди.

Спостереження за середовищем складається з кутової швидкості корпусу, кутових швидкостей, горизонтальних швидкостей, вертикальних швидкостей, положень шарнірів, контакту ніг з землею, та десяти значень вимірювань лідару. Абсолютні координати у спостереженні за середовища відсутні.

Для успішного вирішення середовища необхідно набрати 300 балів винагороди не довше, ніж за 1600 часових ітерацій [9].

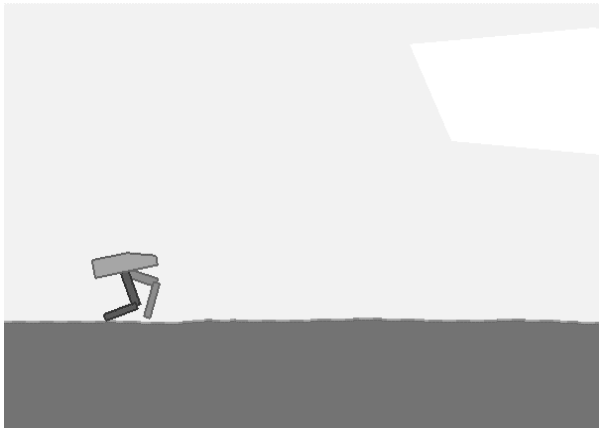


Рис. 2. Кадр з візуалізації середовища BipedalWalker-v3

Для поєднання середовища OpenAI Gym з реалізацією алгоритму Neat SharpNEAT, були застосовані засоби міжпроцесової взаємодії типу іменований канал.

Іменований канал – це один з механізмів обміну даними між процесами Unix та подібних операційних систем. На відміну від неіменованих каналів, доступ до яких мають лише процеси, породжені від спільного батьківського, доступ до іменованих каналів мають всі процеси, що знають ім'я цього каналу та мають відповідні права читання та/або запис в канал. Як і неіменований канал, іменований канал забезпечує обмін даними через оперативну пам'ять. В Unix-системах ім'я іменованого каналу зберігається у файловій системі як ім'я файлу каналу і доступне після перезавантаження системи. У MS Windows файл каналу створюється у пам'яті і знищується між завантаженнями [10].

Як реалізація іменованих каналів на стороні SharpNEAT було обрано сімейство класів .NET System.IO.Pipes: NamedPipeClientStream та NamedPipeServerStream [11].

Як реалізація іменованих каналів на стороні Python-обгортки середовища OpenAI Gym було обрано сімейство pywin32: win32file та win32pipe [12].

Експеримент SharpNEAT був сконфігурований наступним чином:

функція активації нейронів: Leaky rectified linear unit, Leaky ReLU, нещільний лінійний випрямляч, монотонна [13].

Випрямляч у контексті штучних нейромереж є передавальною функцією. Вона є аналогом напівперіодичного випрямляча у схемотехніці. Ця передавальна функція запроваджена для динамічних мереж Ганлозером (англ. Hahnloser) та іншими у 2000 році з біологічним підґрунтям та математичним обґрунтуванням. В 2011 році вперше продемонстровано, як забезпечити краще навчання глибинних мереж, на відміну від передавальних функцій, які широко використовувались до цього, а саме, логістичною функцією (яка була запозичена з теорії ймовірностей), і виявились більш практичними ніж гіперболічний тангенс. ReLU є, станом на 2018 рік, найбільш популярною передавальною функцією для глибинних нейронних мереж [14].

Нещільна ReLU (рис. 3) використовує невеликий додатній градієнт, коли передавач не активний, з рівнянням

$$f(x) = \begin{cases} 0,001x & x < 0 \\ x & x \geq 0 \end{cases}$$

і областю значень: $(-\infty, \infty)$.

До переваг ReLU відносяться.

1. *Біологічна правдоподібність* – одностороння, на відміну від центрально симетричного гіперболічного тангенса.

2. *Розріджена активація* – наприклад, у випадково ініціалізованій мережі, тільки близько 50 % прихованих елементів активуються (мають не нульове значення).

3. *Краще градієнтне поширення*: рідше виникає проблема зникнення градієнта у порівнянні з сигмоїдальною передавальною функцією, яка може виникнути в обох напрямках.

4. *Швидкість обчислення*: тільки порівняння, додавання та множення.

5. *Інваріантність відносно масштабування*: $\max(0, ax) = a\max(0, x)$ для $a \geq 0$.

ReLU використано для відокремлення специфічного збудження та неспецифічної заборони (інгібування) у піраміді з нейронною абстракцією (Neural Abstraction Pyramid), яка була навчена з учителем, щоб вирішувати декілька завдань комп'ютерного зору. У 2011 році, ReLU використовували як елемент нелінійності з метою показати, можливість глибокого навчання нейронної мережі без попереднього навчання без учителя. ReLU, на відміну від сигмоїда та подібних передавальних функцій, дозволяє швидше та ефективніше навчання глибоких нейронних мереж на великих та складних наборах даних.

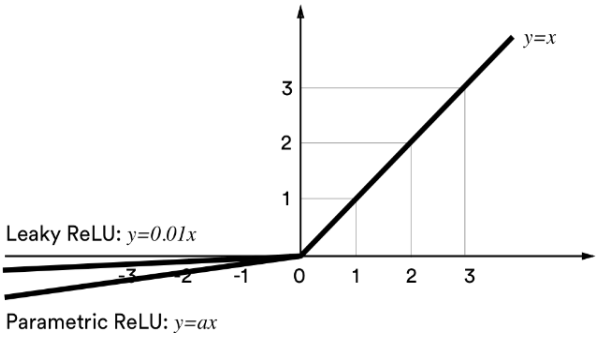


Рис. 3. Графік функції Leaky ReLU

Кількість видів: 10.

Відсоток елітарності: 20 %. Геноми різних видів відсортовані за успішністю, кращі з них, елітарні, зберігаються, інші ж вибраковуються щоб надати дорогу виплоду.

Розмір популяції: 2000. Кількість геномів у популяції.

Початкова частка зв'язків: 5 %.

Відсоток відбору: 20 %. Геноми різних видів сортуються за успішністю та обираються батьківські геноми для створення виплоду. Відбір відбувається до застосування правил елітарності.

Відсоток безстатевого виплоду: 50 %. Відсоток виплоду, шо створиться безстатевим шляхом, за рахунок мутацій.

Відсоток статевого виплоду: 49 %. Відсоток виплоду, що утворюється статевим шляхом.

Відсоток міжвидового розмноження: 1 %. Відсоток статевого розмноження, що задіє геноми різних видів.

Ймовірність мутації ваги зв'язків: 94 %.

Ймовірність мутації створення нового нейрону: 1 %.

Ймовірність мутації створення зв'язку: 2,5 %.

Ймовірність мутації видалення зв'язку: 2,5 %.

Результати експериментів

В результаті застосування алгоритму NEAT/SharpNAET з попередньо визначеною конфігурацією, середовищем та проміжними прошарками поєднуючого програмного забезпечення, поставлену задачу вдалося вирішити (рис. 6) за 10972000 епізодів симуляції. Під успішним вирішенням розуміється набір необхідних 300 балів винагороди найкращою еволюційною нейромережею у кожній із 100 послідовних тестових симуляцій.

На рис. 4 та рис. 5 зображені візуалізації топологій нейромереж геномів-чемпіонів у різні випадкові етапи еволюції.

На рис. 7–10 зображені графіки основних показників історичних даних експерименту: складності та придатності створених геномів.

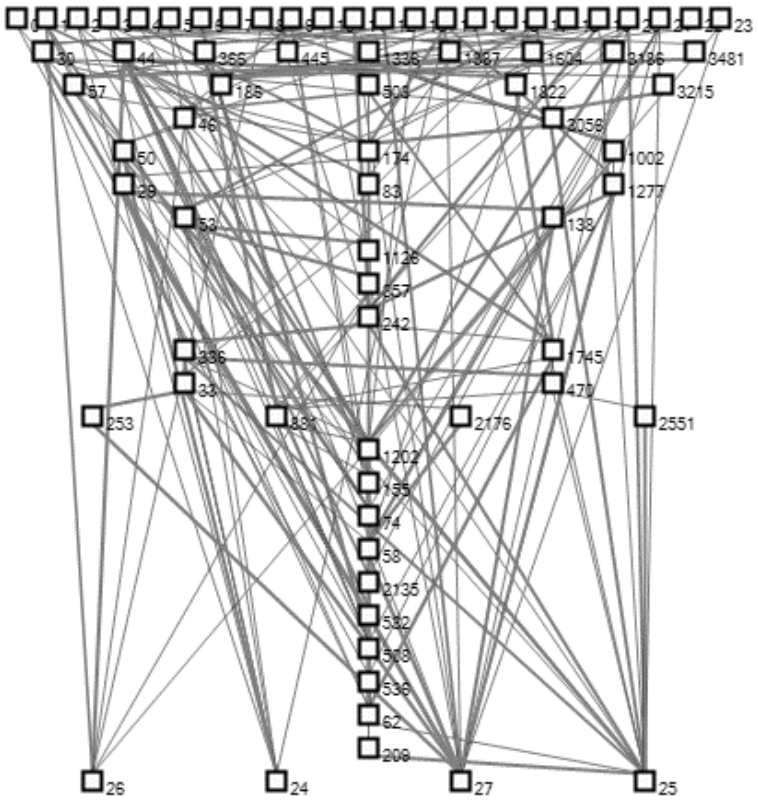


Рис. 4. Візуалізація випадкової топології нейромережі проміжних поколінь

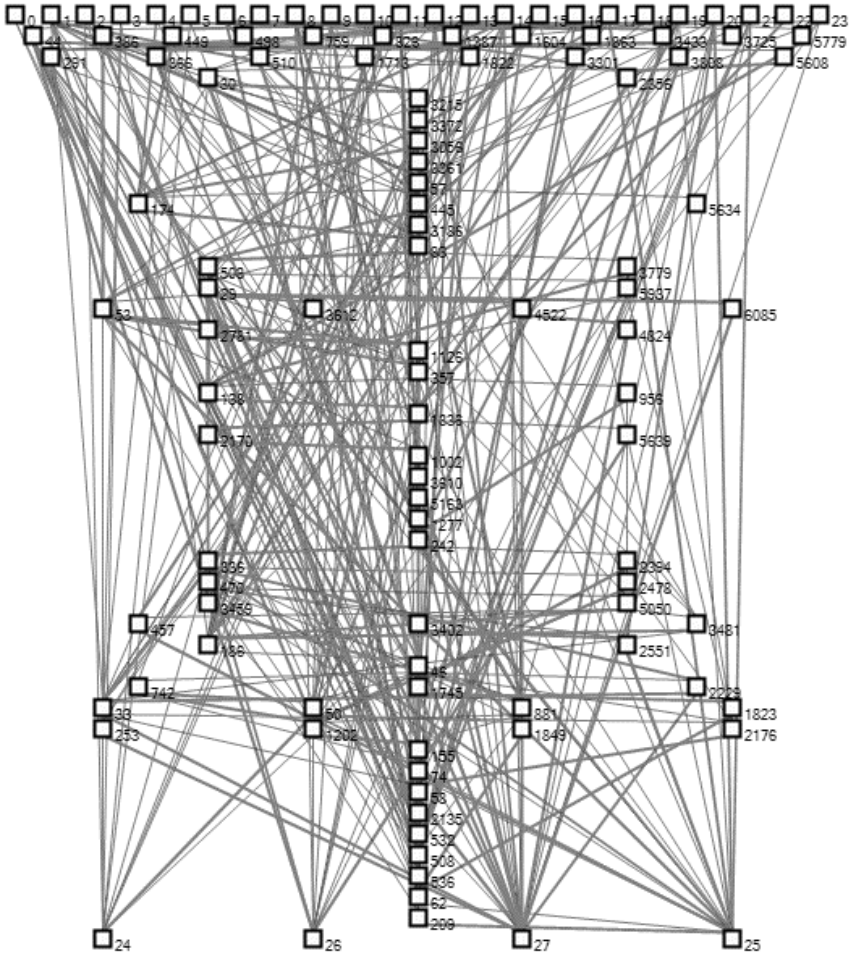


Рис. 5. Візуалізація однієї з найкращих топологій нейромережі

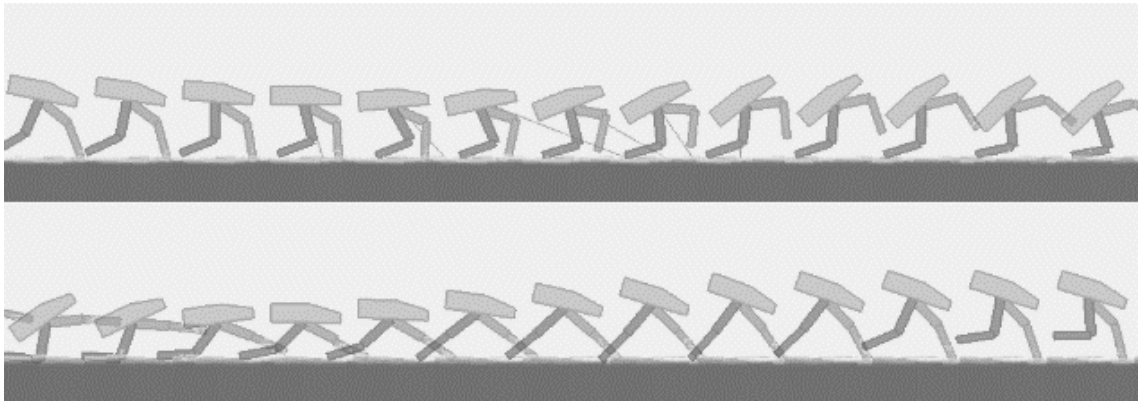


Рис. 6. Комбінована візуалізація кадрів симуляції середовища з використанням агента NEAT

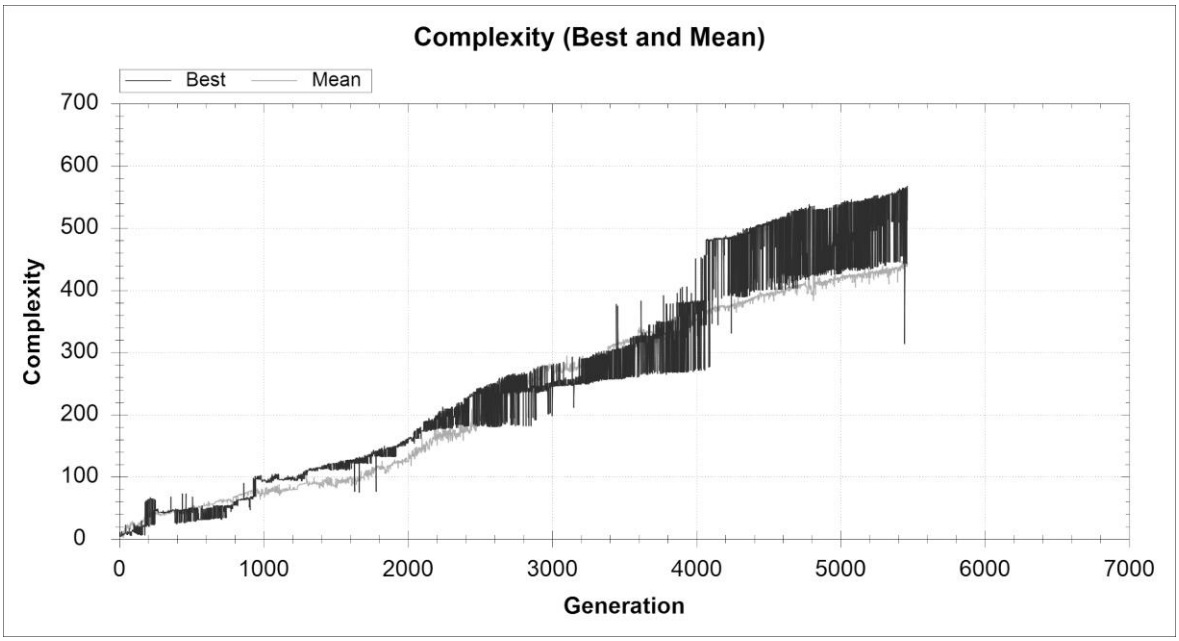


Рис. 7. Графік залежності складності кращої та середньої топології від покоління

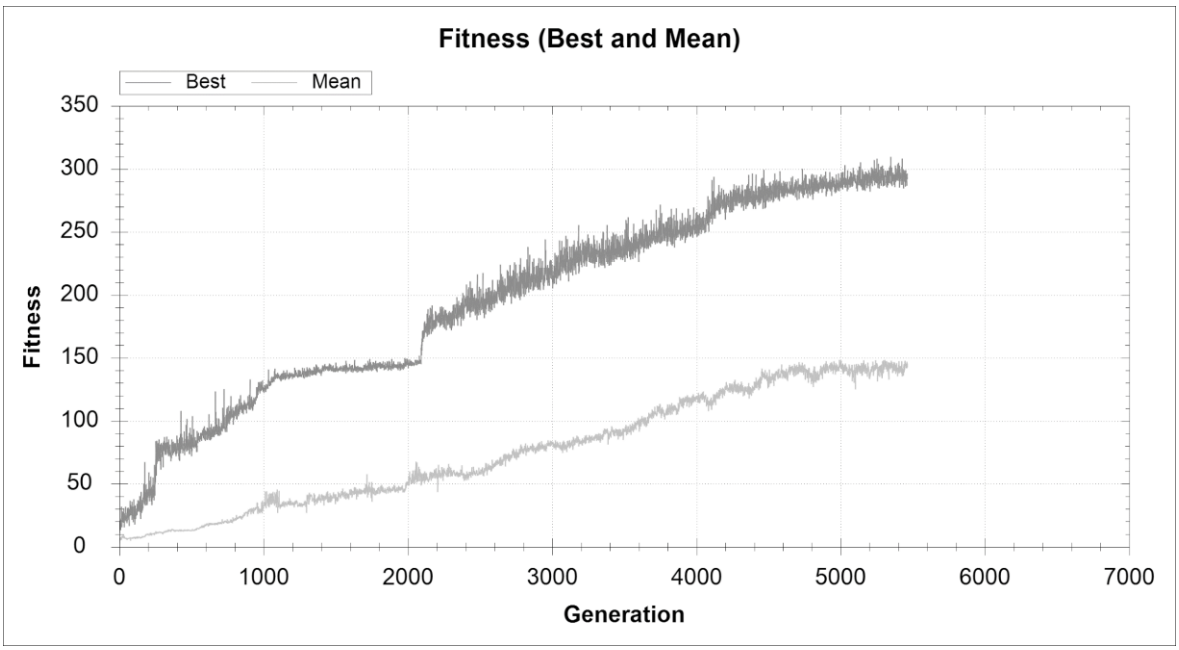


Рис. 8. Графік залежності найкращої та середньої винагороди агентів від покоління

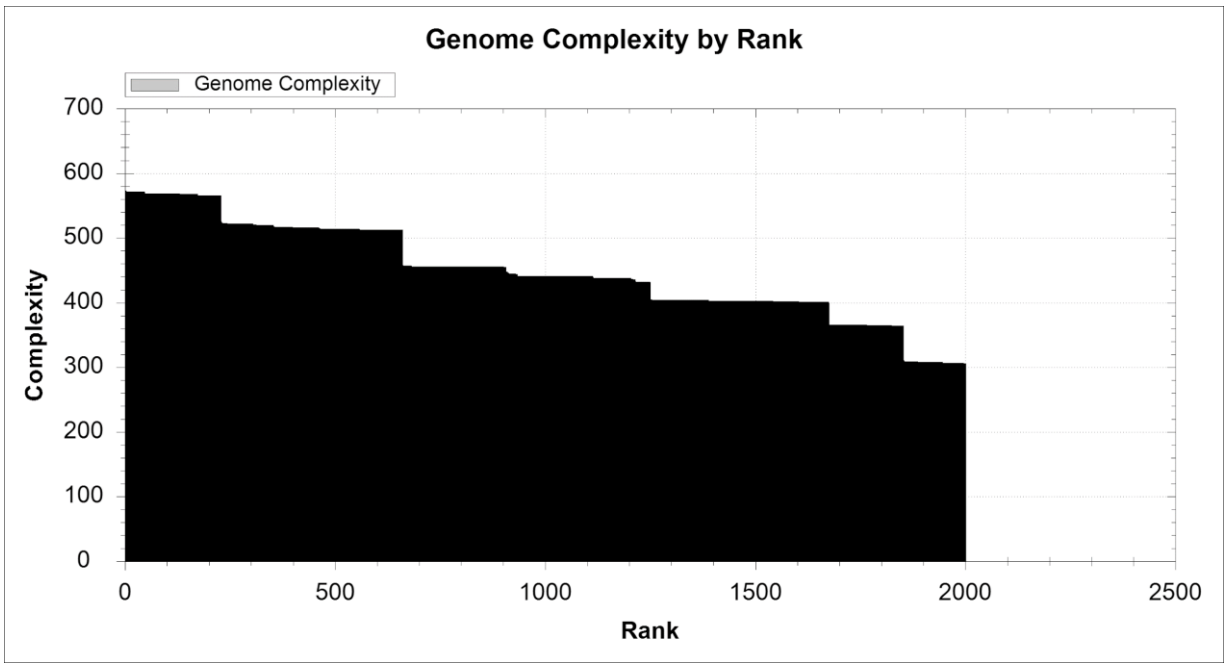


Рис. 9. Графік розподілу складності геному за його рангом

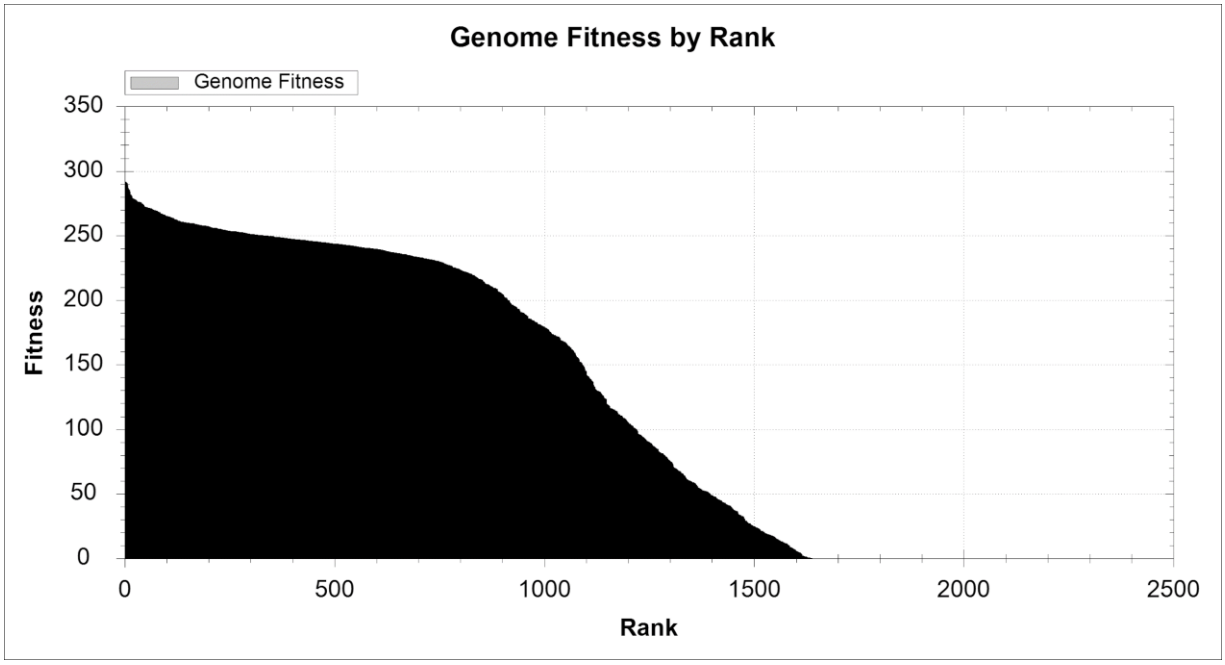


Рис. 10. Графік розподілу придатності геному за його рангом

Висновки

В роботі запропоновано нове застосування техніки нейроеволюції наростаючих топологій для розв'язування задач автоматизації керування на одному з прикладів моделювання задач керування технічними системами, де використовується навчання з підкріпленням. Використовується набір інструментів для розробки та порівняння алгоритмів навчання з підкріпленням OpenAI Gym, повноцінна реалізація з відкритим програмним кодом генетичного алгоритму нейроеволюції NEAT під назвою SharpNEAT та проміжне програмне забезпечення для оркестрації зазначених компонентів. Алгоритм нейроеволюції наростаючих топологій демонструє знаходження ефективних нейронних мереж на прикладі вирішення простих стандартних галузевих задач з

системами з неперервним керуванням OpenAI Gym. Наступними кроками може бути дослідження можливості та ефективності застосування розглянутої техніки та супровідних компонентів середовища на прикладі масштабних задач, задач інших напрямків, дослідження можливості та ефективності застосування паралельних, розподілених та апаратно-прискорених обчислень у NEAT.

Література

1. Хайкин С. Нейронные сети: полный курс, 2-е изд., Испр.: Пер. с англ. М.: ООО «И.Д. Вильямс », 2006. 1104 с.
2. Wilson, Halsey (2018). Artificial intelligence. Grey House Publishing. 184 p.
3. Kenneth O. Stanley. Ph.D. Dissertation: EFFICIENT EVOLUTION OF NEURAL NETWORKS THROUGH COMPLEXIFICATION [Електронний ресурс] / Kenneth O. Stanley // Department of Computer Sciences, The University of Texas at Austin. 2004. Режим доступу до ресурсу: http://nn.cs.utexas.edu/keyword?stanley:phd04.
4. https://gym.openai.com/
5. https://sharpneat.sourceforge.io/
6. Russell Stuart J.; Norvig Peter (2010). Artificial intelligence: a modern approach (Third ed.). Upper Saddle River, New Jersey. P. 830, 831.
7. Kenneth O. Stanley and Risto Miikkulainen (2002). "Evolving Neural Networks Through Augmenting Topologies". Evolutionary Computation 10 (2). P. 99–127.
8. https://gym.openai.com/envs/BipedalWalker-v2/
9. https://github.com/openai/gym/blob/master/gym/envs/box2d/bipedal_walker.py
10. https://tldp.org/LDP/lpg/node15.html
11. https://docs.microsoft.com/en-us/windows/win32/ipc/named-pipes?redirectedfrom=MSDN
12. https://github.com/mhammond/pywin32
13. https://medium.com/@himanshuxd/activation-functions-sigmoid-relu-leaky-relu-and-softmax-basics-for-neural-networks-and-deep-8d9c70eed91e
14. https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6

References

1. Haykin, Simon S. (1999). Neural networks: a comprehensive foundation. Prentice Hall.
2. Wilson, Halsey (2018). Artificial intelligence. Grey House Publishing. 184 pages.
3. Kenneth O. Stanley. Ph.D. Dissertation: EFFICIENT EVOLUTION OF NEURAL NETWORKS THROUGH COMPLEXIFICATION / Kenneth O. Stanley // Department of Computer Sciences, The University of Texas at Austin. – 2004. – http://nn.cs.utexas.edu/keyword?stanley:phd04.
4. https://gym.openai.com/
5. https://sharpneat.sourceforge.io/
6. Russell, Stuart J.; Norvig, Peter (2010). Artificial intelligence: a modern approach (Third ed.). Upper Saddle River, New Jersey. P. 830, 831.
7. Kenneth O. Stanley and Risto Miikkulainen (2002). "Evolving Neural Networks Through Augmenting Topologies". Evolutionary Computation 10 (2): 99-127
8. https://gym.openai.com/envs/BipedalWalker-v2/
9. https://github.com/openai/gym/blob/master/gym/envs/box2d/bipedal_walker.py
10. https://tldp.org/LDP/lpg/node15.html
11. https://docs.microsoft.com/en-us/windows/win32/ipc/named-pipes?redirectedfrom=MSDN
12. https://github.com/mhammond/pywin32
13. https://medium.com/@himanshuxd/activation-functions-sigmoid-relu-leaky-relu-and-softmax-basics-for-neural-networks-and-deep-8d9c70eed91e
14. https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6

Одержано 21.02.2021

***Про авторів*:**

Дорошенко Анатолій Юхимович,
доктор фізико-математичних наук,
професор, завідувач відділу теорії
комп'ютерних обчислень,
професор кафедри автоматики та
управління в технічних системах
НТУУ "КПІ імені Ігоря Сікорського".

Кількість наукових публікацій в
українських виданнях – понад 190.
Кількість наукових публікацій в
зарубіжних виданнях – понад 80.
Індекс Гірша – 6.
http://orcid.org/0000-0002-8435-1451,

Ашур Ілля Зін-Еддінович,
аспірант 3 курсу в
НТУУ “КПІ імені Ігоря Сікорського”.

Місце роботи авторів:

Національний технічний університет
України "КПІ імені Ігоря Сікорського",
проспект Перемоги 37.

Інститут програмних систем
НАН України,
03187, м. Київ-187,
проспект Академіка Глушкова, 40.

Тел.: (044) 526 3559.

E-mail: doroshenkoanatoliy2@gmail.com,
ilyaachour@gmail.com

УДК 004.4'22 https://doi.org/10.15407/pp2021.01.026

А.М. Покровський

ЗАСІБ ВИМІРЮВАННЯ МЕТРИК ВИХІДНОГО КОДУ FORTRAN ЗА ДОПОМОГОЮ СИНТАКСИЧНОГО АНАЛІЗУ

За умов стрімкого розвитку методик забезпечення якості програмного коду, все більшою стає потреба в інструментах, що можуть автоматизувати процес оновлення та реструктуризації текстів програм. Розроблено програмний засіб для вимірювання програмних метрик, що дозволяє провести оцінювання якості вихідного коду програм мовою Fortran. Для цього розроблено алгоритми обходу синтаксичного дерева програми та на їх основі реалізовано модуль для інтегрованого середовища програмування Photran. Модуль використовує синтаксичний аналізатор програмного коду та побудоване Photran на його основі структурне дерево. Проведено порівняння розробленого засобу з наявними інструментами аналізу вихідного коду.

Ключові слова: метрика вихідного коду, якість програмного забезпечення, рефакторинг.

Вступ

Забезпечення якості вихідного коду – це важлива проблема, що стоїть перед розробниками програмного забезпечення. Протягом життєвого циклу програмного продукту вимоги до нього видозмінюються, що призводить до необхідності редагування та реструктуризації коду. У деяких випадках, життєвий цикл програми (від початку проектування до виходу із вжитку) може складати десятки років. За такий час міняються не тільки функціональні вимоги, а й версії мови програмування, що використовується, стилістичні підходи при написанні текстів програм тощо.

Прикладом такого середовища є екосистема мови Fortran. Як перша високорівнева мова програмування, що активно використовується й нині, Fortran за час свого існування зазнала значних змін як у синтаксисі, так і у вимогах якості до вихідного коду [1]. Водночас, велика кількість програмних засобів та компонент вимагає підтримки та оновлення, оскільки задачі, що вирішуються за допомогою цих програм, не втрачають своєї актуальності. Для забезпечення підтримуваності та для спрощення задачі проектування Fortran – систем була створена значна кількість підручних засобів автоматизації розробки (Computer-Aided Software Engineering, CASE-інструментів). Процес підвищення якості програмного забезпечення (ПЗ) (реінжиніринг) полягає у виявленні та усуненні потенційних вад (технічних або стилістичних), що можуть ускладнити роботу над програмним продуктом. Для цього CASE-інструменти надають дві групи автоматизованих операцій – вимірювання програмних метрик, що дозволяють дати оцінку якості вихідного коду, та здійснення рефакторингів, тобто внесення у цей код комплексних змін, що призведуть до підвищення його якості. Існує багато інтегрованих середовищ програмування, які надають подібні можливості при роботі з Fortran, наприклад, Forcheck, fpt, плагіни для Visual Studio та CLion.

В даній роботі за основу програмного засобу було взято Photran – плагін для Eclipse IDE з відкритим вихідним кодом, що містить вбудований програмний інтерфейс для взаємодії та аналізу вихідного коду Fortran-програм. На його основі створено засіб для оцінювання якості вихідного коду за допомогою вимірювання програмних метрик. Також наведено порівняння різних інструментів, що надають подібний функціонал.

Матеріал даної роботи організовано наступним чином. В першому розділі описані програмні метрики, вимірювання яких було реалізовано у розробленому засобі. В другому розділі описано плагін Photran та реалізація інтерфейсу програмного засобу. В третьому розділі описано розроблені алгоритми вимірювання метрик з використанням синтаксичного дерева програми. В останньому розділі наведено порівняння розробленого програмного засобу зі схожими інструментами, що надають інші середовища програмування.

1. Програмні метрики

Метрики програмного забезпечення – множина алгоритмів, за допомогою яких можна отримати з вихідного коду програми числове значення, яке має свідчити про рівень складності, взаємозалежності та ефективності алгоритму. Більшість метрик фокусуються на вимірюванні таких показників, як розмір програми, складність логічного графу програми (структурні елементи та залежності між ними – класи, модулі, файли, функції тощо) або складність графу керування програми (послідовність виконання інструкцій, викликів підпрограм).

Різні класи метрик вимагають для їх обчислення взаємодії з різними форматами представлення програми. Для максимально повної демонстрації можливостей програмного інтерфейсу засобу доцільно вибирати метрики, які покривають більшість таких форматів.

В роботі реалізовано вимірювання трьох метрик, що представляють три класи взаємодій з програмним інтерфейсом.

Кількість рядків коду (Source Lines Of Code, SLOC) – примітивна метрика, що вимірює обсяг та зусилля, необхідні для написання програми рядками вихідного коду. Ця метрика складається з трьох різних значень:

- кількість фізичних рядків коду (LOC) – кількість рядків програми, що безпосередньо перетворюються компілятором/транслятором на машинний код;
- кількість рядків коментарів (comment LOC, CLOC) – кількість рядків коментарів, тобто тверджень, що ігноруються компілятором;
- кількість пустих рядків (blank LOC, BLOC) – кількість рядків, що не містять тексту та використовуються для візуального розбиття програми на блоки.

Вид взаємодії з програмним інтерфейсом – для вимірювання цієї метрики необхідно мати змогу отримати доступ до вихідного коду програми.

Обсяг зусиль по Холстеду (Halstead Effort) – метрика, що була введена в 1977 році Морісом Холстедом в числі групи метрик складності (Halstead Complexity Measures) як спосіб виміряти складність та обсяг вихідного коду програми. Ця метрика фокусується на оцінюванні зусиль, що будуть витрачені програмістом на написання або підтримку коду та розглядає текст програми як набір лексичних одиниць та зв'язків між ними [2].

Для обчислення обсягу зусиль за Холстедом необхідно виміряти такі початкові значення:

n_1 – кількість унікальних операторів,

n_2 – кількість унікальних операндів,

N_1 – загальна кількість операторів,

N_2 – загальна кількість операндів.

Ці значення характеризують розмір та різноманітність програмного лексикону, що був використаний при написанні вихідного коду, а відношення між загальною кількістю та кількістю унікальних елементів показує ступінь зв'язаності тексту (більша кількість використань символу – більша зв'язаність). На основі цих значень здійснюється обчислення похідних метрик:

- словник програми (n):

$$n = n_1 + n_2 . \quad (1)$$

- розмір програми (N):

$$N = N_1 + N_2 . \quad (2)$$

- обсяг (об'єм) програми (V):

$$V = N \times \log_2 n . \quad (3)$$

- складність програми (D):

$$D = \frac{n_1}{2} \times \frac{N_2}{n_2} . \quad (4)$$

- обсяг зусиль для написання програми (E):

$$E = D \times V . \quad (5)$$

Вимірювання цієї метрики на відміну від LOC вимагає аналізу вихідного коду та визначення залежностей між лексичними одиницями мови програмування. Вид взаємодії з програмним інтерфейсом – збір інформації про синтаксичні елементи ви-

хідного коду, їх тип та зв'язки між визначенням та використанням ідентифікаторів.

Цикломатична складність за Маккейбом – структурна метрика складності алгоритму, введена Томасом Маккейбом в 1976 році. Практичною проблемою, що призвела до створення цієї метрики, була необхідність вирахування кількості тестів за допомогою яких можна повністю покрити комп'ютерну програму. Цикломатична складність визначається як кількість лінійно незалежних шляхів через алгоритм [3]. Найпростіша програма, що не містить операторів умовного переходу, буде мати цикломатичну складність 1, бо існує лише один шлях проходу через неї. Один умовний перехід збільшить кількість маршрутів удвічі – один у випадку коли значення умови переходу є істинним, другий у випадку коли воно є хибним.

З математичної точки зору цикломатична складність вираховується за допомогою представлення програми у вигляді орієнтованого графа, де вузлами є групи нероздільних інструкцій, а ребрами – переходи потоку керування між цими групами. Позначимо:

E – кількість ребер у графі,

N – кількість вузлів у графі,

P – кількість компонентів зв'язності графа (для підпрограми дорівнює 1).

Тоді цикломатичну складність M можна обчислити за формулою:

$$M = E - N + 2 \times P .$$

Для вимірювання цієї метрики необхідно побудувати та обійти структурний граф програми та визначити, як впливають на цикломатичну складність різні мовні конструкції. Вид взаємодії з програмним інтерфейсом – збір інформації про потік керування та послідовність виконання інструкцій програми.

2. Середовище Photran

Photran – потужне інтегроване середовище програмування (IDE) для Fortran побудоване на базі Eclipse. В набір функцій Photran входять редагування вихідного коду, система пошуку та навігації, компіляція, виконання та налагодження програм. Для побудови проектів використовується make, що в сукупності з відкритим вихідним кодом та можливістю запускати Eclipse з усіх популярних ОС, робить Photran зручним та гнучким засобом, який можна швидко налаштувати під конкретні потреби.

Представлення Fortran-програми у Photran здійснюється за допомогою двох структур даних – абстрактного синтаксичного дерева (Abstract Syntax Tree, AST) та віртуальнного програмного графа (Virtual Program Graph, VPG). AST представляє собою структурне дерево програми, гілки якого закінчуються листками, які відповідають за синтаксичні елементи мови програмування (оператори, літерали, ідентифікатори тощо). VPG служить інтерфейсом між програмістом та AST, дозволяє отримувати та вивільнювати його екземпляри, проводити лексичний аналіз коду, отримувати інформацію про зв'язки між програмними структурами та їх ідентифікаторами [4].

Модуль вимірювання метрик необхідно пов'язати з інтерфейсом Eclipse для можливості взаємодії з вікном огляду проекту (для вибору файлів, з яких необхідно зібрати дані). Для цього у роботі було використано вбудовану у Photran механіку рефакторингів. Вона дозволяє за допомогою так званого «ресурсного рефакторингу» зареєструвати певну операцію, що має відповідний підпункт на навігаційній панелі та передає на вхід операції файли, що були виділені користувачем у вікні огляду проекту.

Для реєстрації рефакторингу необхідно виконати ряд дій:

- створити клас, що реалізує інтерфейс IResourceRefactoring. Цей інтерфейс містить методи для отримання назви операції, перевірки початкових умов (коректності програмного коду та файлової структури проекту) та внесення змін у код. Оскільки операція вимірювання метрик є пасивною, то останній метод не використовується.
- Зареєструвати нові рядки, що міститимуть назву операції та її класу, додати значення цих рядків до конфігураційного файлу.

– Зареєструвати саму операцію у файлі plugin.xml пакета org.eclipse.photran.ui.vpg.

Далі наведено приклад текста файла маніфеста для реєстрації рефакторингу:

```
<group>
<submenu name="Code Analysis"><!--
Refactorings to perform code analysis -->
<resourceRefactoring
class="org.eclipse.photran.internal.
corerefactoring.MeasureCodeMetrics-
Refactoring"/>
</submenu>
</group>
```

3. Реалізація модуля вимірювання метрик для Photran

Для отримання доступу до структури програми та вимірювання метрик використовується об'єкт AST. Програмні конструкції, що необхідно враховувати при цьому, у синтаксичному дереві представлені вузлами які, у свою чергу, представлені в програмному інтерфейсі Photran об'єктами класів, що реалізують базовий інтерфейс IASTNode. Обхід дерева здійснюється за допомогою патерна «відвідувач», для чого необхідно реалізувати інтерфейс IASTVisitor. Цей інтерфейс містить методи, в тілі яких має здійснюватися обробка «відвіданого» структурного вузла. Кожен вузол у свою чергу містить метод accept(IASTVisitor), який є реалізацією патерну та надає переданому об'єкту доступ до внутрішніх конструкцій та дочірніх вузлів. Наприклад, клас ASTIfConstructNode відповідає за конструкцію умовного переходу, та в реалізації методу accept дозволяє «відвідувачу» обійти тіло конструкції (код, куди здійснюється умовний перехід), конструкцію else if та конструкцію else.

Тривіальним прикладом структур дерева та відвідувача буде програма, що містить визначення певних даних (змінної, масиву тощо) та виклик підпрограми. Підпрограма, в свою чергу, міститиме конструкцію умовного переходу (if-elseif). Клас-відвідувач для обходу такого дерева має реалізувати ряд методів, по одному для кожного типу структури. Спрощену діаграму класів для такого випадку показано на рис. 1.

Алгоритми вимірювання метрик:

SLOC та BLOC – найпростіші можливі метрики, для вимірювання яких достатньо отримати текст програми за допомогою методу IASTNode.toString(), розбити його на рядки та порахувати їх загальну кількість та кількість порожніх рядків.

Для вирахування метрик LOC, CLOC, що залежать від вмісту рядка, було використано метод IFortranAST. findFirstTokenOnLine(). У випадку наявності «токена» (програмного виразу) рядок є рядком коду, інакше – рядком з коментарем.

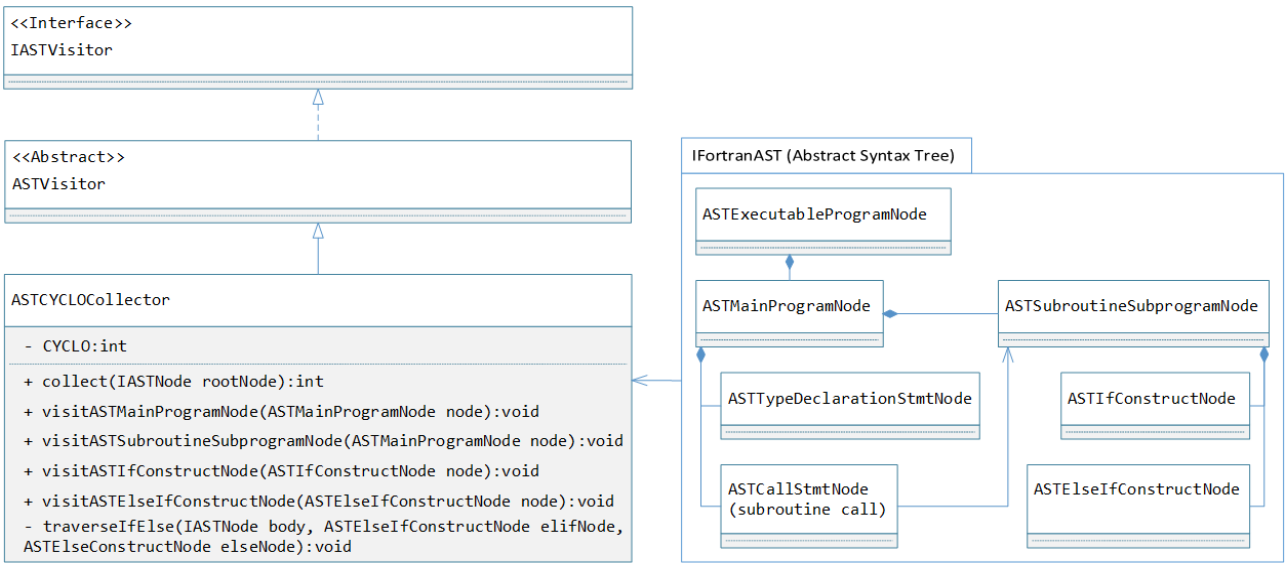


Рис. 1. Спрощена діаграма класів для патерна «відвідувач» у Photran

Для вимірювання **зусиль за Холстедом** необхідно отримати чотири базові значення – загальну та унікальну кількості операндів та операторів (N_1, N_2, n_1, n_2). Кожному операнду та оператору в синтаксичному дереві відповідає певний об'єкт класу Token. У кожного токена, у свою чергу, є тип (представлений об'єктом класу Terminal), що унікально ідентифікує синтаксичні одиниці мови.

Операнди у Fortran діляться на два типи: «ідентифікатори» (змінні та імена функцій або підпрограм), що отримують тип токена T_IDENT та літерали або константи, що отримують тип у залежності від типу константи (T_ICON для цілочисельного літерала, T_SCON для строкового літерала тощо). Літерали, навіть однакові за значенням, завжди вважаються унікальними, тому їх кількість просто сумується і додається до N_2 та n_2. До N_2 також додається сумарна кількість ідентифікаторів, але оскільки одного визначення (однієї змінної) можуть стосуватися багато токенів, то для підрахунку кількості унікальних ідентифікаторів необхідно відслідкувати ці визначення. Для цього VPG надає метод Token.resolveBinding() що вертає унікальний об'єкт визначення (класу Definition), що стосується даного ідентифікатора. Загальна кількість цих об'єктів після обходу всіх токенів додається до n_2.

Всі токени, що не є ідентифікаторами або константами, вважаються операторами. Їх загальна кількість записується в N_1, а кількість унікальних – в n_1. Після цього здійснюються обчислення, описані в розділі 1.

Збір інформації для вирахування **цикломатичної складності** вимагає повного обходу синтаксичного дерева та зміни значення метрики в залежності від типу конструкції. Конструкції, що вимагають обходу «відвідувачем» при обчисленні цикломатичної складності, наведені в табл. 1.

Таблиця 1

Дії при виявленні мовних конструкцій

Назва конструкції	Приклад конструкції	Дія при виявленні
if statement	if () ! action	– збільшення значення метрики на 1

if construct	if () then
	! body
	end if
		– збільшення значення метрики на 1
		– обхід тіла конструкції

else if construct	else if () then ! body end if	– збільшення значення метрики на 1 – обхід тіла конструкції

else construct	else
	! body
	end if
		– обхід тіла конструкції

loop construct	do [label] [while ()] ! body [label | end do]	– збільшення значення метрики на 1 – обхід тіла конструкції
assigned/computed goto statement	goto (label[, label]), value	– збільшення значення метрики на кількість аргументів label
case construct	select case () ! body end select	– обхід тіла конструкції
case statement	case ()	– збільшення значення метрики на 1
forall statement	forall () ! action	– збільшення значення метрики на 1
forall construct	forall () then ! body end forall	– збільшення значення метрики на 1 – обхід тіла конструкції

Особлива ситуація виникає з конструкцією where. Ця конструкція призначена для скороченого запису фільтрації масивів за певною умовою. В найпростішому випадку (один «рівень» фільтрації) препроцесор перетворить твердження where на один цикл do з одним набором умовних переходів всередині. При додаванні додаткового рівня фільтрації, препроцесор вимушений виконати «розгортання» конструкції для того щоб другий рівень фільтрації виконувався після звершення першого (на оновлених даних) [5]. Приклад таких конструкцій наведено в табл. 2.

З цієї причини, звичайний обхід конструкцій where не дасть виміряти коректного значення метрики. Цикломатична складність вузла в цьому випадку залежить не тільки від типу, але й від рівня фільтрації на якому перебуває вузол, загальної вкладеності рівнів фільтрації та їх кількості всередині вузла. Оскільки конструкції where можуть містити в собі тільки інші конструкції where, то зручно розглядати where найвищого рівня як окреме дерево.

Позначимо:

n – кількість рівнів фільтрації,

m_i – кількість вузлів на i-ому рівні, $i = \overline{1,n}$,

w_{ij} – j-ий вузол -го рівня, $i = \overline{1,n}$, $j = \overline{1,m_i}$,

s_{ij} – кількість дочірніх вузлів w_{ij}, $i = \overline{1,n}, j = \overline{1,m_i}$,

c_{ijk} – k-ий дочірній вузол w_{ij}, $i = \overline{1,n}, j = \overline{1,m_i}$,

$L(x)$ – кількість викликів вузла x.

Таблиця 2

Перетворення конструкцій where препроцесором

К-сть шарів фільтрації	Вихідний код	Фактичний код	Значення метрики

1	where (condition1)
	  ! body1
	elsewhere (condition2)
	  ! body2
	elsewhere
	  ! body3
	end where
		do
		  if (condition1) then
		  ! body1
		  else if (condition2) then
		  ! body2
		  else
		  ! body3
		  end if
		end do
			4

2	where (condition1)
	  where (condition3)
	  ! body3
	  end where
	  ! body1
	elsewhere (condition2)
	  ! body2
	end where
		do
		  if (condition1) then
		  ! body1
		  else if (condition2) then
		  ! body2
		  end if
		end do
		
		do
		  if (condition1) then
		  if (condition3) then
		  ! body3
		  end if
		  end if
		end do
			7

Проаналізувавши приклади, наведені в табл. 1, видно що вузол w_{ij} буде задіяно один раз для -го рівня і по одному разу для кожного рівня, в якому лежать його нащадки. З цього виходить, що кількість викликів вузла $L(x)$ на одиницю більша за його висоту у дереві і її можна обчислити за рекурсивною формулою:

$$L(w_{ij}) = \max_k \{L(c_{ijk})\}_{k=1}^{s_{ij}} + 1. \qquad (6)$$

За допомогою формули (6) можна порахувати кількість конструкцій if, які буде створено препроцесором для певної конструкції where. Водночас, оскільки для кожного рівня необхідно окремо обійти масив, кількість циклів, що буде створено, дорівнює висоті дерева. Тому загальну цикломатичну складність такої конструкції можна обчислити як:

$$\sum_{i=1}^{n}\sum_{j=1}^{m_i} L(w_{ij}) + \max_j \{L(w_{1j})\}_{j=1}^{m_1}. \qquad (7)$$

Для демонстрації вимірювання метрик використано проект-приклад, завантажений з інтернету. Приклад виводу програми показано на рис. 2.

filename	SLOC	LOC	CLOC	BLOC	CYCLO	EFFORT
...	...	...	...	...	...	...
tavg.f90	1599	806	478	315	115	5046464,17
test_prognostic.f	121	53	39	29	1	25533,89
time_manageme	4319	2018	1370	931	329	12638544,90
timers.f90	876	367	306	203	47	1815981,12
topostress.f90	399	172	138	89	28	913376,87
vertical_mix.f90	1545	749	456	340	107	9218730,17
vmix_const.f90	242	97	85	60	12	292245,27
vmix_kpp.f90	2072	1014	645	413	168	15959973,99
vmix_rich.f90	403	168	136	99	25	1057082,47
xdisplay.f90	215	73	92	50	11	105203,90
Total	64670	32622	18789	13259	5412	314670533,81
Execution took PT5.946404S						

Рис. 2. Приклад виводу програми

При багаторазовому запуску програми середня швидкість оброблення вихідного коду складає ~1000 строк/сек. Вимірювання проводилися з використанням комп'ютера оснащеного процесором Intel Core i7-9750H 2.6GHz з обсягом оперативної пам'яті 16GB.

4. Порівняння розробленого засобу з аналогами

На початку тисячоліття хороший інструмент для рефакторингу має містити швидкий статичний аналізатор синтаксису, потужний модуль пошуку та заміни. На сьогоднішній день цей список буде надто коротким щоб задовольнити всі потреби розробника. При порівнянні сучасних інструментів рефакторингу мова йдеться про їх інфраструктуру, про можливість інтеграції з сучасними інтегрованими середовищами розробки, про перевірений та зручний інтерфейс користувача, гарну масштабованість для можливості підтримки різних розмірів проекту, компіляторів та середовищ виконання, часті оновлення, хорошу документованість, інструменти для роботи з командою, тобто мова йдеться вже про екосистему мови програмування.

За час існування Fortran створена велика кількість засобів для автоматизації навігації та аналізу програмного коду (CASE-інструментів). Вони здебільшого діляться на чотири категорії:

технічні аналізатори вихідного коду. Це програмні засоби, призначені для перегляду тексту програми та пошуку застарілого або «метрвого» коду та синтаксичних помилок [6];

стилістичні аналізатори вихідного коду. Вказують на заплутаність тексту програми, недостатню кількість коментарів та шкідливі стилістичні або архітектурні практики [7];

засоби навігації вихідним кодом. Дозволяють здійснювати переміщення графом потоку даних (між змінними та їх визначеннями) та потоку керування (деревом викликів підпрограм), а також здійснювати пошук за ключовими словами, типами даних тощо [8];

засоби рефакторингу вихідного коду. Надають інструментарій для автоматичної реорганізації тексту програми з метою покращення структури, швидкодії, полегшення внесення змін та подальшої підтримки продукту.

Засоби для вимірювання програмних метрик належать до першої категорії, але оскільки сучасна парадигма інтегрованого середовища програмування передбачає об'єднання більшості необхідних розробникам інструментів в один продукт, ці засоби зазвичай містять функціонал з інших категорій. Найбільш розвинуті CASE-інструменти для Fortran (крім Photran) наведені у табл. 3.

В порівнянні з перерахованими засобами, Photran має ряд переваг. Eclipse IDE в його основі робить його однаково простим та гнучким у налаштуванні на Windows, Linux та MacOS. Відкритий вихідний код та ліцензування за Eclipse Public License означає що будь-хто може переглядати, змінювати та використовувати готові компоненти для реалізації нового функціоналу. Можливість прямої взаємодії з AST означає порівняну легкість реалізації на його основі систем переписувальних правил [9] або графічних візуалізацій [10]. В сукупності з навігацією вихідним кодом Fortran, здійсненням рефакторингів та вимірюванням метрик, Photran є потужним та доступним засобом автоматизації підвищення якості програмного забезпечення в галузях, які найбільше цього потребують.

Таблиця 3

Порівняння CASE-інструментів для Fortran

Назва	Стан підтримки	Загальний опис	Збір програмних метрик
Understand	В активній розробці	Інтегроване середовище, що містить функціонал для навігації вихідним кодом та сфокусоване на створенні візуальних звітів по структурі проекту.	Збирає метрики LOC та інші кількісні метрики (кількість файлів, підпрограм, викликів функцій тощо).
SCSE	В активній розробці	Потужний інструмент для навігації вихідним кодом, основним функціоналом є виконання пошукових запитів на великих об'ємах даних.	Збирає метрики LOC, цикломатичну складність та зусилля за Холстедом.
FPT	Остання активність 07.2020	Технічний аналізатор вихідного коду з фокусом на міграціях програм між різними середовищами виконання та операційними системами.	Збирає метрики LOC та цикломатичну складність.
Forcheck	В активній розробці	Технічний аналізатор вихідного коду, що підтримується вже більше 30 років [11], з фокусом на пошуку неявних помилок у тексті програм.	Збирає метрики LOC та цикломатичну складність.
Fortran Analyzer	Остання активність 09.2019	Стилістичний аналізатор вихідного коду з фокусом на альтернативних метриках, що вказують на «чистоту» коду, таких як глибина вкладених циклів, документованість програмних елементів (файлів, підпрограм, модулів).	Збирає метрику SLOC та інші кількісні метрики (кількість підпрограм, викликів функцій,).

Висновки

В роботі розроблені алгоритми для обходу синтаксичного дерева, на основі яких реалізовано додаток до плагіну Photran для оцінювання якості вихідного коду мови Fortran за допомогою вимірювання програмних метрик. Використання розробленого засобу є більш дієвим у поєднанні з іншими функціями Photran для визначення, усунення та відслідковування динаміки росту програми. Проведено порівняння поточного стану та перспектив розвитку інфраструктури середовища Photran з іншими CASE-інструментами. До подальших напрямків розробки належать створення інструментів для візуалізації структури вихідного коду, рефакторингів для оптимізації паралельних обчислень, засобів для стилістичного аналізу програм.

Література

1. Overbey J.L., Negara S., Johnson R.E. Refactoring and the evolution of Fortran. Workshop on Software Engineering for Computational Science and Engineering, Vancouver, BC. 2009. P. 28–34.
2. Hariprasad T., Vidhyagaran G., Seenu K., Thirumalai C. Software complexity analysis using halstead metrics. International Conference on Trends in Electronics and Informatics (ICEI), Tirunelveli. 2017. P. 1109 –1113.
3. McCabe T.J. A Complexity Measure. IEEE Transactions on Software Engineering. 1976. N 4. P. 308–320.
4. Photran Developer's Guide [Електронний ресурс]. Режим доступу до ресурсу: https://git.eclipse.org/c/ptp/org.eclipse.photran.git/plain/org.eclipse.photran-dev-docs/dev-guide/dev-guide-specialized.pdf.
5. Metcalf M., Reid J.K. Fortran 90/95 explained. USA: Oxford University Press, Inc. 1999. 341 p. (2).
6. fpt – Tools for Fortran Engineering [Електронний ресурс]. Режим доступу до ресурсу: http://www.simconglobal.com/fpt_summary.html.
7. García-Rodríguez M., Añel J., Foujols M., Rodeiro J. FortranAnalyser: A software tool to assess Fortran code quality [Електронний ресурс]. 2016. Режим доступу до ресурсу: http://fortrananalyser.ephyslab.uvigo.es/docs/access.pdf.
8. Source Code Search Engine [Електронний ресурс]. Режим доступу до ресурсу: http://www.semanticdesigns.com/Products/SearchEngine/?site=SoftwareRecommendations.
9. Дорошенко А.Ю., Жереб К.А., Туліка Є.М. Розпаралелювання програм на фортрані з використанням техніки переписувальних правил. *Проблеми програмування*. 2012. № 2-3. С. 388–397.
10. Fortran 77 Flowcharting Utility [Електронний ресурс]. Режим доступу до ресурсу: http://www.deater.net/weave/vmwprod/f77_diagram/.
11. Lawden M.D. FORCHECK. A Fortran Verifier and Programming Aid. Lawden. Starlink User Note. 1990. N 73. P. 1–3. Top 10 Digital Transformation Trends For 2019 [Електронний ресурс]. Режим доступу до ресурсу: https://www.forbes.com/sites/danielnewman/2018/09/11/top-10-digital-transformation-trends-for-2019/#17d55d1e3c30.

References

1. Overbey J.L., Negara S., Johnson R.E. Refactoring and the evolution of Fortran. Workshop on Software Engineering for Computational Science and Engineering, Vancouver, BC. 2009. P. 28–34.
2. Hariprasad T., Vidhyagaran G., Seenu K., Thirumalai C. Software complexity analysis using halstead metrics. International Conference on Trends in Electronics and Informatics (ICEI), Tirunelveli. 2017. P. 1109–1113.
3. McCabe T.J. A Complexity Measure. IEEE Transactions on Software Engineering. 1976. N 4. P. 308–320.
4. Photran Developer's Guide [online] Available at: <https://git.eclipse.org/c/ptp/org.eclipse.photran.git/plain/org.eclipse.photran-dev-docs/dev-guide/dev-guide-specialized.pdf> [Accessed 12 February 2021].
5. Metcalf M., Reid J.K. Fortran 90/95 explained. USA: Oxford University Press, Inc. 1999. 341 p. (2).
6. fpt – Tools for Fortran Engineering [online] Available at: <http://www.simconglobal.com/fpt_summary.html> [Accessed 12 February 2021].
7. García-Rodríguez M., Añel J., Foujols M., Rodeiro J. FortranAnalyser: A software tool to assess Fortran code quality [online]

Fortrananalyser.ephyslab.uvigo.es. Available at: <http://fortrananalyser.ephyslab.uvigo.es/docs/access.pdf> [Accessed 12 February 2021].
8. Semanticdesigns.com. n.d. Source Code Search Engine. [online] Available at: <http://www.semanticdesigns.com/Products/SearchEngine/?site=SoftwareRecommendations> [Accessed 12 February 2021].
9. Doroshenko A., Zhereb K. and Tulika Y. Parallelization of Fortran programs using rewriting rules. *Problems of programming*. 2012. Vol. 2-3. P. 388–397.
10. Deater.net. n.d. 10. Fortran 77 Flowcharting Utility. [online] Available at: <http://www.deater.net/weave/vmwprod/f77_diagram/> [Accessed 12 February 2021].
11. Lawden M., 1990. FORCHECK – A Fortran Verifier and Programming Aid. Starlink User Note, 73(1). P. 1–3.

Одержано 13.02.2021

***Про автора*:**

Покровський Андрій Максимович,
студент 2 курсу магістратури в
НТУУ “КПІ імені Ігоря Сікорського”

Місце роботи автора:

Національний технічний університет України "КПІ імені Ігоря Сікорського", кафедра автоматики і управління в технічних системах.
проспект Перемоги 37.

E-mail: a.m.pokrovskyi@gmail.com

УДК 004.62 https://doi.org/10.15407/pp2021.01.036

Ю.А. Дивак

АНАЛІТИЧНИЙ ОГЛЯД ПІДХОДІВ ДО ІНТЕГРАЦІЇ ПРОГРАМНИХ СИСТЕМ

Виконаний порівняльний аналіз існуючих підходів до інтеграції програмних систем, з метою подальшого розроблення нових альтернативних підходів для вирішення задач інтеграції і композиції веб сервісів. Проведено інтеграцію платіжної системи в Інтернет-магазин одним із зазначених методів для більш глибокого розуміння проблем які виникають при інтеграції програмних систем. Розглянуті виклики, які стоять перед розробниками під час інтеграцій систем, та методи вирішення поставлених задач. Визначено ключові моменти на які слід звернути уваги про інтеграції програмних систем і взаємодії між ними. Об'єкт вивчення потребує нового погляду на проблему і нових альтернативних підходів, які принесуть більшу гнучкість, можуть підвищити продуктивність, знайти своє застосування у сфері інтеграції програмних систем і композиції веб-сервісів.

Ключові слова: Інтеграція програмних систем, композиція веб-сервісів.

Вступ

Великі компанії мають екосистеми, які включають у себе більше ніж одну програмну систему для підтримки бізнес процесів [1]. Наявність декількох систем працюючих разом дає багато переваг по відношенню до відокремлених незалежних монолітних систем: вони можуть працювати узгоджено розподіляючи ресурси, мати більш широкий функціонал, працювати незалежно та об'єднувати результати роботи. Тому, в корпоративному середовищі в сфері інформаційних технологій існує проблема інтеграції декількох відокремлених систем. Основне питання цієї проблеми, це як змусити відокремлені системи працювати разом та забезпечити об'єднаний набір функцій? Деякі програми можуть розроблятися власноруч вдома або в гаражі, тоді як інші купуються у сторонніх постачальників. Додатки працюють на декількох комп'ютерах, які можуть представляти кілька платформ і можуть бути географічно розподіленими. Деякі програми можуть працювати поза межами підприємства зі сторони партнерів або замовників. Доводиться інтегрувати програми які не були розроблені для інтеграції та взаємодії з іншими системами, та не можуть бути змінені, що ускладнює завдання. Ці проблеми та інші подібні до них ускладнюють інтеграцію додатків і взаємодію між системами.

Метою аналітичного огляду є розгляд існуючих підходів до інтеграції, для більш глибокого аналізу проблематики та розуміння потреб тих кого ця проблема стосується – це розробники корпоративних застосунків та інтеграційних рішень, корпорації метою яких є інтеграція з партнерами та колаборація з іншими учасниками ринку для досягнення економічних показників і конкурентоспроможності, наукове товариство яке зацікавлене в розробці нових підходів до існуючих проблем та їх вирішення. Ми розглянемо декілька підходів та визначимо позитивні та негативні сторони кожного з них.

Критерії оцінки та визначення підходу для інтеграції програмних систем

Інтеграція декількох систем потребує завжди різний набір властивостей і функцій, тому підходів з інтеграції існує декілька і кожен підхід задовольняє різні її потреби. Однак, як і будь-які складні технологічні зусилля, інтеграція додатків включає цілий ряд міркувань та наслідків, які слід враховувати під час вибору підходу до інтеграції.

Критерії інтеграції мають бути враховані під час вибору та розробки інтеграційного підходу. Який підхід інтеграції

корпоративних застосунків більш відповідає вказаним критеріям, такий і треба застосувати в конкретному випадку.

Перший критерій – це сама інтеграція програми. Чи потрібна інтеграція взагалі? Якщо можливо розробити окремий додаток, який не потребує взаємодії з будь-якими іншими програмами, то можемо повністю уникнути всіх проблем пов'язаних з інтеграцією застосунків. Однак, навіть у простого підприємства є декілька програмних продуктів, які повинні працювати разом, щоб забезпечити взаємодію для працівників підприємства, партнерів та клієнтів [3].

Зв'язність застосунків. Всі застосунки які приймають участь в інтеграції повинні мінімізувати свою залежність один від одного, щоб кожен міг розвиватися незалежно, не створюючи проблем для інших. Тісно зв'язані програми роблять численні припущення про те, як працюють інші програми (зазвичай на основі інтерфейсів, контрактів). Але коли програми змінюють і порушують ці припущення (інтерфейси, контракти), інтеграція порушується. Інтерфейс для інтеграції програм повинен бути достатньо конкретним для реалізації корисної функціональності, але достатньо загальним, щоб дозволити цій реалізації змінюватися за необхідності [3].

Простота інтеграції. При інтеграції програми в корпоративному середовищі розробники повинні мінімізувати зміни в програмі та мінімізувати необхідну кількість коду для інтеграції. Вирішення повинно бути простим і зрозумілим. Проте, як правило, необхідні зміни та новий код, щоб забезпечити хорошу функціональність інтеграції, і підходи з найменшим впливом на програму можуть не забезпечити найкращої інтеграції на підприємстві. Інтеграція повинна також забезпечувати гнучкість, що не завжди забезпечуються простими рішеннями [3].

Технологія інтеграції. Різні методи інтеграції вимагають різного обсягу спеціалізованого програмного та апаратного забезпечення. Ці спеціальні інструменти можуть бути дорогими і призвести до блокування постачальників (сервісів) та збільшити навантаження на розробників, які повинні розуміти як використовувати інструменти для інтеграції [3].

Формат даних. Інтегровані програми повинні узгоджувати формат даних, якими вони обмінюються, або повинні мати проміжний транслятор для уніфікації програм, які використовують різні формати даних. З часом формат даних може еволюціонувати, змінюватись та розширюватись, що може вплинути на функціональність програми. При чому узгодження повинно відбуватися як на загальному рівні (XML, JSON, BLOB), так і на рівні структур даних, які передаються між системами [3].

Своєчасність даних. Інтеграція повинна мінімізувати проміжок часу між програмами, коли одні відправляють дані, а інші отримують і можуть їх використовувати. Даними слід часто обмінюватися невеликими порціями, а не чекати обміну великим набором непов'язаних елементів, але канал передачі даних є вузьким місцем будь-якої системи, тому іноді краще зменшити кількість запитів між інтегрованими системами та відправляти дані одним набором, що підвищить швидкість взаємодії між системами. Інтегровані програми повинні бути проінформовані, як тільки спільні дані будуть готові до споживання. Затримка обміну даними повинна враховуватися в інтеграційному дизайні; чим довше триває доступ, тим більша вірогідність що дані застаріли і тим складнішою стає інтеграція [3].

Дані або функціональні можливості. Інтегровані програми можуть не обмінюватися даними, а використовувати функції, щоб кожна програма могла використовувати функціональність інших програм. Викликати віддалену функцію важко, і хоча це може здатися таким самим, як виклик локальної функції, воно працює зовсім по-іншому, що суттєво впливає на ефективність роботи інтеграції.

Прикладом такої інтеграції між двома відокремленими програмами, застосунком та базою даних, які можуть бути розташовані на різних серверах є можливість виклику віддалених процедур у базах даних Oracle та MySQL [3].

Асинхронність. Комп'ютерна обробка, як правило, синхронна, це значить що процедура чекає, поки виконується її підпроцедура, але можливий варіант виконання процедури асинхронно коли процедура не буде чекати закінчення виконання підпроцедури, а виконає її у фоновому режимі. Це особливо стосується інтегрованих програм, де віддалена програма може не працювати, або мережа може бути недоступною, у такому випадку вихідна програма може просто зробити доступними спільні дані або записати запит на виклик підпроцедури, але потім перейти до іншої роботи впевнена, що віддалений виклик буде викликано пізніше [3].

Існуючі підходи до інтеграції корпоративних застосунків

Представлено декілька підходів для інтеграції застосунків. Кожен з них у більшій чи в меншій мірі відповідає критеріям інтеграції. Як стверджено G. Hohpe та B. Woolf [3] інтеграція програмних систем може бути виконана чотирма підходами які називаються: Передача файлів, Спільна база даних, Виклик віддаленої процедури, Обмін повідомленнями.

1. Передача файлів (by Martin Fowler). Кожна програма виробляє файли з даними якими вона ділиться з іншими застосунками. Та отримує файли з даними, які були вироблені іншими застосунками.

Уявляючи, як організація працює з єдиного цілісного програмного забезпечення, розробленого з самого початку, щоб працювати в єдиній і цілісній формі. Звичайно, навіть найменші операції не працюють так. Кілька програмних продуктів обробляють різні аспекти діяльності підприємства. Це пов'язано з низкою причин.

– Корпорації купують програмні системи, розроблені сторонніми організаціями.

– Різні системи будуються в різний час, що призводить до різного вибору технологій.

– Будь-які системи будуються особами, досвід та уподобання яких приводить до різних підходів до побудови додатків.

Розроблення програмного продукту та надання цінності компанії та бізнесу є більш важливим, ніж забезпечення вирішення питань інтеграції з іншими системами, особливо коли ця інтеграція не додає ніякої цінності програмі, що розробляється. Як результат, будь-яка організація повинна турбуватися про обмін інформацією між різними програмами. Вони можуть бути написані різними мовами, на основі різних платформ і з різними припущеннями про те, як працює бізнес. Зв'язування таких додатків вимагає великих знань про розуміння того, як пов'язати програми як на бізнес-рівні, так і на технічному рівні. Щоб налагодити взаємодію, треба мінімізувати те, що потрібно знати програмі про те, як працюють інші програми. Потрібен загальний механізм передачі даних, який може використовуватися різними мовами та платформами. Для цього потрібно мати мінімальну кількість спеціалізованого обладнання та програмного забезпечення, використовуючи те, що є у наявності підприємства. Файли – це універсальний механізм зберігання, вбудований в будь-яку корпоративну операційну систему, доступний на будь-якій мові підприємства. Найпростішим підходом було б якось інтегрувати програми за допомогою файлів (рис. 1).

Нехай кожна програма створює файли, що містять інформацію, яку повинні використовувати інші програми. Інтегратори несуть відповідальність за перетворення файлів у різні формати.

Важливим рішенням щодо файлів є формат, який використовувати. Дуже рідко результат однієї програми буде саме

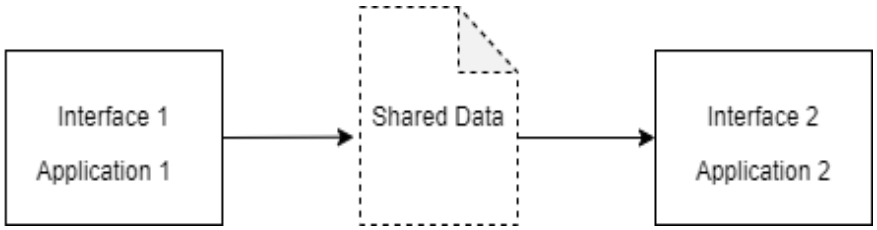


Рис. 1. Інтеграція через Передачу файлів

тим, що потрібно для іншої, тож доведеться досить дорого обробляти та перетворювати файли. Всі програми, які використовують файл, повинні його прочитати, та вміти використовувати на ньому інструменти обробки. Як результат, із часом стандартні формати файлів виросли. Системи мейнфреймів зазвичай використовують канали даних на основі форматів файлових систем COBOL. Системи Unix використовують текстові файли. Сучасність полягає у використанні XML та JSON. Навколо кожного з цих форматів склалася галузь читачів файлів, письменників файлів та засобів трансформації. Інше питання, пов'язане з файлами, – це коли їх створювати та споживати. Оскільки для створення та обробки файлу потрібні певні зусилля, але не має бажання надто часто працювати з ними. Зазвичай є якийсь регулярний діловий цикл, який визначає рішення: щоночі, щотижня, щокварталу тощо. Програми пристосовують до конкретно часу в яких інформація у файлі доступна та є актуальною, і обробляють його у свій час. Великою перевагою файлів є те, що інтегратори не потребують знання внутрішніх елементів програми. Команда заявок зазвичай надає файл. Вміст та формат файлу узгоджуються з інтеграторами. В результаті різні додатки які приймають участь в інтеграції досить добре відокремлені один від одного. Кожна програма може вільно вносити внутрішні зміни, не впливаючи на інші програми, за умови, що вони все одно створюють однакові дані у файлах у тому ж форматі. Файли фактично стають інтерфейсом кожного додатка. Частина того, що спрощує передачу файлів, полягає у тому, що не потрібні додаткові інструменти чи пакети інтеграції, але це також означає, що розробники повинні зробити багато роботи самостійно. Програми повинні узгодити правила іменування файлів та каталоги, в яких вони відображаються. Автор файлу повинен реалізувати стратегію, щоб зберегти імена файлів унікальними. Програми повинні узгодити, як будуть видалятися, і визначатися старі файли. Додатки повинні впровадити механізм блокування або дотримуватися норми синхронізації, щоб гарантувати, що одна програма не намагається прочитати файл, а інша все ще пише його. Якщо всі програми не мають доступу до одного диска, тоді деяка програма повинна взяти на себе відповідальність за передачу файлу з одного диска на інший. Однією з найбільш очевидних проблем передачі файлів є те, що оновлення, як правило, зустрічаються нечасто, в результаті чого системи можуть вийти з синхронізації. Система керування клієнтами може обробляти зміну адреси та видавати файл витягу щовечора, але система виставлення рахунків може відправити рахунок на стару адресу того ж дня. Іноді відсутність синхронізації не є великою проблемою. Часто очікується певне відставання в отриманні інформації, навіть від комп'ютерів. В інших випадках використання несвіжої інформації є катастрофою для роботи програм. Вирішуючи, коли створювати файли, потрібно враховувати потреби споживачів у актуальності даних. Якщо клієнт того самого дня змінює свою адресу за допомогою двох різних систем, але одна з них помиляється і отримує неправильну назву вулиці, то буде дві невідповідні адреси клієнта. Тоді знадобиться якийсь спосіб на вирішення цього питання. Чим довший період між передачею файлів, тим більш імовірною і нестерпнішою стане ця проблема. Звичайно,

файли можна створювати частіше для підтримки актуальності даних. Інша проблема полягає в керуванні всіма створеними файлами. Повний перелік можливостей підходу основаного на файловій системі набагато ширший, але при обробці файлу витрачається багато ресурсів, що є надмірним, якщо хочемо швидко створити багато файлів [3].

2. Спільна база даних. Застосунки зберігають дані якими вони хочуть ділитися в спільній базі даних.

Для сучасного бізнесу хочемо, щоб усі мали якомога більше актуальних даних. Це не просто зменшення помилок, а збільшення довіри до даних. Швидке оновлення також дозволяє краще усунути невідповідності. Чим частіше синхронізуємось, тим менше шансів отримати невідповідності і тим менше зусиль витрачаємо на вирішення. Але якими б швидкими не були зміни, все одно будуть проблеми.

Існує багато прикладів семантичного дисонансу при застосуванні спільної бази даних, які набагато складніше розглядати, ніж суперечливі формати даних. (Для набагато глибшого обговорення цих питань варто прочитати Data and reality, a timeless perspective on perceiving and managing information in our imprecise world, William Kent [4]).

Якщо потрібна централізована, узгоджена база даних, до якої мають спільний доступ усі додатки, щоб кожен із них мав доступ до будь-якої спільної інформації, і не було б за потрібне інтегрувати програми, через збереження своїх даних в одній спільній базі даних (рис. 2).

Якщо сімейство інтегрованих додатків покладається на одну і ту ж базу даних, то можемо бути впевнені, що вони завжди постійно узгоджуються. Якщо отримуємо одночасне оновлення окремого фрагмента даних з різних джерел, щоб уникнути колізій для цього існують системи керування транзакціями, які керуються принципами ACID [5]. Використання спільної бази даних значно полегшується завдяки широкому розповсюдженню реляційних баз даних на базі SQL. Практично всі платформи для розробки додатків можуть працювати з SQL, часто з досить складними інструментами. Тому не доведеться турбуватися про декілька форматів файлів. Оскільки будь-яка програма в будь-якому випадку повинна використовувати SQL, це уникає додавання іншої технології, якою кожен може володіти. Оскільки всі використовують одну і ту ж базу даних, це вирішує проблеми семантичного дисонансу [3].

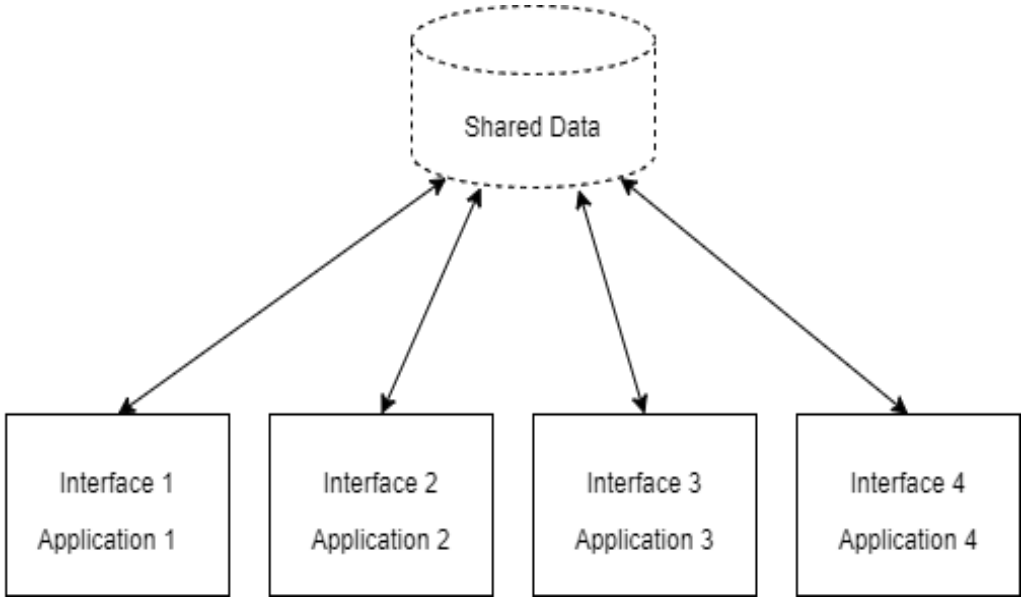


Рис. 2. Інтеграція через Спільну базу даних

Однією з найбільших труднощів спільної бази даних є створення відповідного дизайну (мається на увазі архітектура системи). Інтеграційна база даних повинна мати схему, яка задовольнить потреби всіх клієнтських систем, ця схема завжди більш загальна і більш складна. База даних розробляється декількома організаціями тому змінити щось у базі складно, через те, що зміни мають погоджуватися між клієнтськими застосунками. Розробка уніфікованої схеми, яка може задовольнити потреби багатьох додатків, є дуже складною справою, що часто призводить до створення схеми, з якою програмістам важко працювати. Якщо критична програма, ймовірно, зазнає затримок для роботи з уніфікованою схемою, то часто виникає непереборний тиск для відокремлення. Конфлікти між відділами часто посилюють цю проблему. Іншим, більш жорстким обмеженням для спільної бази даних є зовнішні програмні системи. Частіше вони не працюватимуть з іншою схемою, окрім власної. Навіть якщо є якийсь простір для адаптації, це, швидше за все, буде набагато обмеженішим, ніж хотіли б інтегратори. Ця проблема також поширюється на інтеграцію після розробки. Навіть якщо впорядкувати всі свої програми, все одно виникне проблема інтеграції, якщо відбудеться злиття компаній. Кілька програм, які використовують Спільну базу даних для частого читання та модифікації одних і тих самих даних, можуть спричинити вузькі місця у роботі та навіть тупикові ситуації, оскільки кожна програма блокує інші дані. Коли програми розподіляються між кількома комп'ютерами, база даних повинна бути розподілена також, щоб кожна програма могла отримати доступ до бази даних локально, що заплутує проблему, на якому комп'ютері слід зберігати дані. Розподілена база даних із конфліктами блокування може легко стати перешкодою продуктивності [3].

Існує два підходи до застосування баз даних. База даних може бути або прив'язаною до конкретної системи і належати їй, або вона може бути інтеграційною, і зберігати спільні дані з декількох систем. Різниця між ними полягає у тому хто контролює потік даних. В останні часи розвивається сервісно-орієнтований підхід до розробки корпоративних програмних систем з власною базою даних, які взаємодіють через сервісні інтерфейси, ефективно замінюючи інтеграцію через спільну базу даних інтеграцією на основі виклику віддаленої процедури або обміном повідомлень.

Плюс інтеграційної бази даних – це інтеграція, яка не потребує відокремленого шару інтеграційних сервісів у застосунку. Всі зміни в базі даних стають доступними для всіх клієнтських застосунків одночасно, що робить спільні дані більш синхронізованими.

З іншого боку спільна база даних веде до значних проблем тому, що вона стає точкою зв'язності застосунків. Зазвичай це дуже глибока зв'язність, яка збільшує ризик пов'язаний із змінами системи та її розвитком. Більшість спеціалістів вважає, що треба уникати цього підходу для інтеграції.

3. Виклик віддалених процедур (by Martin Fowler) – кожен застосунок має процедури, які можна викликати віддалено і програми таким викликом запускають процес обміну даними.

Парадигма RPC була використана для впровадження багатьох повсякденних систем. Від програм нижчого рівня, таких як Мережеві файлові системи [6] та віддаленого прямого доступу до пам'яті [7], до протоколів доступу до розвитку екосистеми мікросервісів, RPC використовується скрізь. RPC має різноманітні програми – SunNFS [6], Twitter Finagle (Eriksen, 2013), Apache Thrift (Prunicki, 2009), Java RMI [8], SOAP, CORBA (Group, 1991) та gRPC від Google (Google, n. d.).

Існує декілька підходів до віддаленого виклику процедур (RPC): CORBA, COM, .NET Remoting, Java RMI і т. д. Вони відрізняються за кількістю підтримуваних систем та простотою використання. Часто такі середовища мають додаткові можливості, такі як транзакції.

Передача файлів та спільна база даних дозволяють програмам обмінюватися своїми даними, що є важливою частиною інтеграції програм, але просто обміну даними часто буває недостатньо. Часто зміни даних призводять до того, що доводиться робити зміни в різних додатках. Наприклад, зміна адреси може бути простою зміною даних, або це може спричинити реєстрацію та юридичні процеси з урахуванням різних правил у різних юридичних юрисдикціях. Наявність однієї програми для виклику таких процесів у іншій вимагало б від програм занадто багато знання про внутрішні функції інших програм. Ця проблема відображає класичні проблеми у розробці додатків. Одним з найпотужніших механізмів структурування при розробці додатків є механізм інкапсуляції – де модулі приховують свої дані через інтерфейс виклику функції. Таким чином, вони можуть перехоплювати зміни в даних, щоб виконувати різні дії, які їм потрібно робити, коли дані змінюються. Спільна база даних забезпечує велику, неінкапсульовану структуру даних, що значно ускладнює контроль даних. Передача файлів дозволяє програмі реагувати на зміни під час обробки файлу, але процес затримується. Той факт, що спільна база даних має некапсульовані дані, також ускладнює ведення сімейства інтегрованих додатків. Багато змін у будь-якій програмі можуть спричинити зміни в базі даних, які мають значний вплив на кожен додаток. Як результат, системи, які використовують спільну базу даних, часто дуже неохоче змінюють базу даних, а це означає, що робота з розробки додатків набагато менше реагує на зміни потреб бізнесу [3].

Необхідний механізм який дозволить одному застосунку викликати функції в інших застосунках, передавати дані, які потрібні для спільного використання та викликати функцію, яка інформує програму-отримувач як обробляти дані [3].

Розробка кожної програми – це величезний проект з інкапсульованими даними. Забезпечення інтерфейсу, який дозволяє іншим програмам взаємодіяти з цим застосунком [3].

Віддалений виклик процедур застосовує принципи інкапсуляції до інтеграції застосунків (рис. 3). Якщо програма потребує деякої інформації, яка належить іншій програмі, вона напряму посилає запит до застосунку, який володіє інформацією. Якщо одній програмі необхідно змінити дані іншої, вона посилає запит для зміни інформації. Кожний застосунок підтримує цілісність власних даних, а також може змінювати дані незалежно від інших застосунків.

На сьогодні лідерами у використанні є веб служби, які використовують такі стандарти як: SOAP і XML. Цінною особливістю є те, що вони легко працюють з HTTP, який проходить через брандмауери.

Той факт, що існують методи які обробляють дані, спрощує семантичний дисонанс. Застосунки можуть забезпечувати декілька інтерфейсів для одних і тих же даних, дозволяючи одним користувачам бачити один стиль, а іншим – інший. Оновлення можуть використовувати декілька інтерфейсів, що дає більшу можливість для підтримки декількох методів взаємодії. Однак інтеграторам не зручно

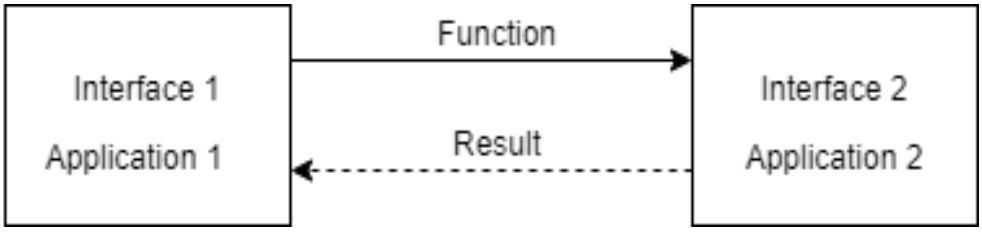


Рис. 3. Віддалений виклик процедур

додавати конвертери, тому важливо узгоджувати інтерфейси взаємодії з іншими застосунками.

Оскільки розробники звикли до виклику процедур, RPC добре співпрацює з тим до чого вони звикли. Насправді це скоріше недолік. Між віддаленими та локальними викликами існує велика різниця у швидкодії та надійності. Якщо розробники не розуміють як використовувати віддалені процедури правильно то це може призвести до розробки повільних та ненадійних систем [8].Інкапсуляція допомагає зменшити зв'язок програм, усуваючи велику спільну структуру даних, але програми ще досить тісно пов'язані між собою. Віддалені виклики, які підтримує кожна система, мають тенденцію пов'язувати різні системи у зростаючий вузол. Зокрема, послідовність – виконання певних дій у певному порядку, яка може ускладнити самостійну зміну систем. Це проблеми, які не важливі у межах однієї програми, а стають важливі при інтеграції декількох програм.

Спеціалісти, розробляючи інтеграцію як єдину програму, не підозрюють про правила зміни.

4. Відправлення повідомлень – кожна програма приєднана до спільної системи повідомлень, яка ділиться даними та запускає будь-які процедури використовуючи повідомлення (рис. 4).

Асинхронні повідомлення – це принципово прагматична реакція на проблеми розподілених систем. Надсилання повідомлення не вимагає одночасної роботи і готовності обох систем.

Існує багато прикладів застосування підходу передачі повідомлень до інтеграції застосунків через Enterprise Service Bus (ESB), це такі як Dell Boomi [9], Informatica [10], JitterBit [11], MuleSoft [12], Oracle [13], SnapLogic [14], as well Guarana [15].

Крім того, розгляд асинхронної комунікації змушує розробників визнати, що робота з віддаленим додатком відбувається повільніше, що заохочує розробку компонентів з високою згуртованістю (багато роботи локально) та низькою адгезією (вибіркова робота віддалено). Системи обміну повідомленнями також дозволяють роз'єднання, яке можливо отримати під час використання передачі файлів. Повідомлення можуть бути трансформовані під час передачі, але ні відправник, ні отримувач не знають про трансформацію. Дійсно, роз'єднання дозволяє інтеграторам транслювати повідомлення на кілька приймачів, підтримує вибір одного з багатьох потенційних приймачів та інші топології, які

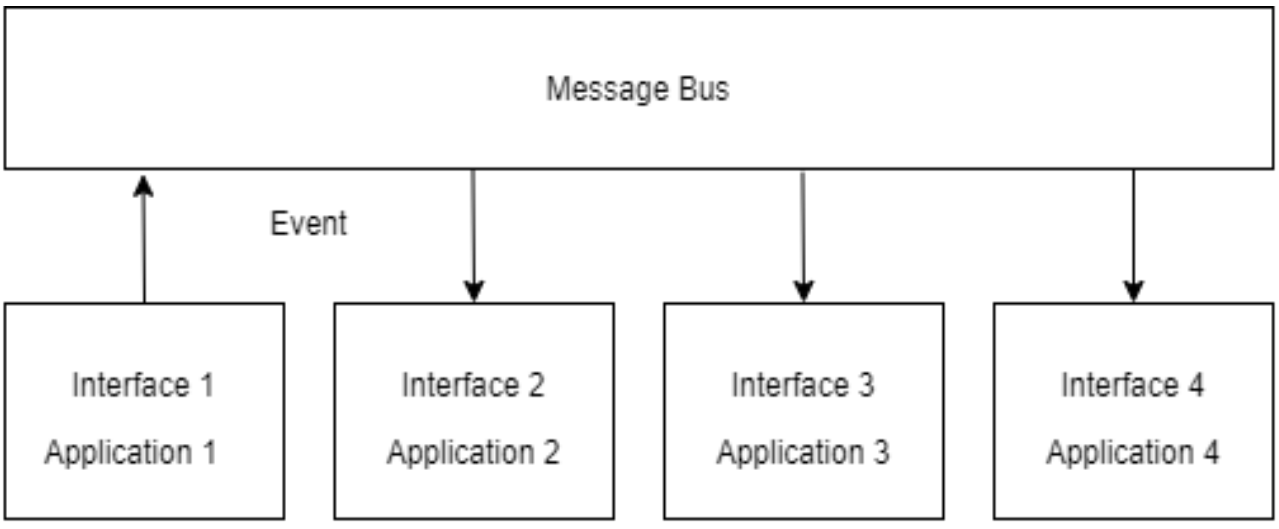


Рис. 4. Інтеграція через відправку повідомлень

дозволяють інтеграцію відокремлювати від розробки додатків. Оскільки дані проблеми, як правило, відокремлюють розробку додатків від інтеграції програм, цей підхід працює скоріше з людською природою, а не проти. Трансформація означає, що окремі програми можуть мати досить різні концептуальні моделі. Звичайно, це означає, що відбудеться семантичний дисонанс, але з точки зору обміну повідомленнями. Міра, яку вживає **Спільна база даних**, щоб уникнути семантичного дисонансу, занадто складна, щоб працювати на практиці, і не може бути виконана після об'єднання пакетів [3].

При надсиланні невеликих повідомлень, дозволяється програмам взаємодіяти між собою, а також обмінюватися даними. Можна запросити інформацію та швидко відповісти. Хоча така співпраця не буде такою швидкою, як віддалений виклик процедури, абоненту не потрібно зупинятися, поки обробляється повідомлення та повертається відповідь. І обмін повідомленнями не такий повільний, як думають багато людей – багато рішень для обміну повідомленнями зароджується у галузі фінансових послуг, де тисячі біржових котирувань або торгів повинні проходити через систему обміну повідомленнями щосекунди.

Висока частота повідомлень у програмі обміну повідомленнями зменшує багато проблем із невідповідністю, що спричиняє неприємну передачу файлів, але не повністю їх усуває. Виникатимуть проблеми із запізненням, оскільки системи не оновлюються одночасно.

Асинхронний дизайн – це не те, що вивчає більшість програмістів, і в результаті існує ціла низка різних правил і методів характерних вивченню такого дизайну. Тестування та налагодження є суттєво складнішим у середовищі обміну повідомленнями.

Можливість перетворення повідомлень має приємну перевагу, дозволяючи додаткам набагато відокремитись один від одного, ніж у віддаленому виклику процедур та передачі файлів. Але ця незалежність насправді означає, що інтеграторам часто залишається писати багато безладного коду, щоб все поєднати.

Вирішивши використовувати **Повідомлення** для системної інтеграції, існує низка нових питань, які слід розглянути і практики, які використовуватимемо. **Як передавати пакети даних?** Відправник надсилає дані повідомлення отримувачу через канал повідомлення, що з'єднує відправника та отримувача. **Як дізнатися куди надсилати дані?** Якщо відправник не знає, куди направити дані, він може надіслати дані маршрутизатору повідомлень, який направить дані до належного отримувача. **Як дізнатися який формат даних використовувати?** Якщо відправник та отримувач не погоджуються щодо формату даних, відправник може направити дані до **Перекладача** повідомлень, який перетворить дані у формат отримувача, а потім перенаправить дані отримувачу. Якщо ви розробник додатків, то виникає питання **як підключити програму до системи обміну повідомленнями?** Додаток, який бажає використовувати обмін повідомленнями, реалізує кінцеві точки повідомлень для фактичного надсилання та отримання.

Приклад інтеграції двох застосунків методом виклику віддалених процедур (RPC)

Для більш глибокого розуміння проблем, які виникають під час інтеграції застосунків розглянемо практичний приклад інтеграції додатків через віддалений виклик процедур. Нам потрібно інтегрувати два застосунки один з яких є сервісом, який надає постачальник сервісу, у нашому випадку постачальником сервісу є процесінговий центр, а сервіс це платіжна система, яка приймає платежі з банківських карт VISA та MASTERCARD та розподіляє платіж у будь-який банк утримуючи процент комісії.

Споживачем сервісу є Інтернет-магазин з продажу будь-чого. Задача ускладняється тим, що взаємодія застосунків повинна бути налагоджена з двох сторін – із сторони кінцевого користувача

(frontend), а також із сторони застосунку (backend). Кожна позитивна відповідь з платіжної системи буде реєструватися в системі інтернет-магазину для подальшої обробки замовлення та відслідковування коштів на рахунках (рис. 5).

Треба чітко розуміти, що ми маємо дві абсолютно незалежні системи у яких є фронтенд частина і бекенд частина які повинні взаємодіяти між собою.

Зі сторони кінцевого користувача інтеграція здійснюється за допомогою вбудованого айфрейму, якщо сервіс платіжної системи побудований правильно з можливістю адаптації візуальної частини під потреби споживача сервісу, то кінцевий користувач може не відрізнити вбудований сервіс і не зрозуміє, що водночас він працює з сервісом платіжної системи.

Проблема CORS (крос-доменних запитів) та зберігання cookies сесії вирішуються за допомогою реверс проксі, який можна налаштувати, наприклад, за допомогою веб серверу nginx.

Зі сторони серверної частини застосунку інтеграція проводиться за допомогою підходу виклику віддалених процедур через REST API.

Рис. 5. Інтеграція платіжної системи в Інтернет-магазин

Висновки

Передача файлів дозволяє програмам обмінюватися даними, але це може бути не своєчасною, а своєчасність може бути критичною проблемою в інтеграції. Якщо зміни не працюють швидко через декілька додатків, то, можливо допустити неправильні дії через застарілість даних. **Передача файлів** також може недостатньо застосовувати формат даних. Багато проблем при інтеграції виникають через несумісні способи обробки даних. Часто вони представляють тонкі бізнес-проблеми, які можуть мати величезний ефект. **Передача файлів** дозволяє програмі реагувати на зміни під час обробки файлу, тримати програми дуже добре відокремленими, але ціною своєчасності даних. Системи просто не встигають одна за одною. Поведінка співпраці занадто повільна.

Спільна база даних зберігає спільні дані, але ціною зв'язку всіх застосунків з базою даних. Цей підхід теж не справляється зі спільною поведінкою.

Передача файлів та **Спільна база** даних дозволяють програмам ділитися своїми даними, але не своїми функціональними можливостями.

Віддалений виклик процедур дозволяє програмам обмінюватися функціоналом, але тісно поєднує їх процеси. Часто завдання інтеграції полягає у тому, щоб зробити співпрацю між окремими системами якомога своєчасною, не поєднуючи системи між собою, таким чином робить їх ненадійними, як з точки зору виконання програми, так і розробки програми.

Зіткнувшись з цими проблемами, **віддалений виклик процедури** видається привабливим вибором. Але розширення моделі, яка використовується для одного додатка, при інтеграції додатків попадає на безліч інших слабких місць. Ці слабкі сторони починаються з основних проблем розподіленого розвитку. Незважаючи на те, що виклики віддалених процедур виглядають як місцеві виклики, вони не діють однаково. Віддалені виклики відбуваються повільніше і можуть не вдатися. Працюючи з кількома програмами, які обмінюються даними на одному підприємстві, не бажано, щоб збій однієї програми призвів до збою в усіх інші програмах. Крім того, не бажано розробляти систему, вважаючи, що виклики швидкі (але вони можуть бути досить повільними), і щоб кожна програма розпізнавала деталі інших програм, навіть якщо це лише деталі про їх інтерфейси.

Нам потрібно дещо на зразок **передачі файлів**, де багато невеликих пакетів даних можна швидко створити, легко передати, а програма-отримувач автоматично отримує повідомлення, коли новий пакет доступний для споживання. Передача потребує механізму повторної спроби, щоб переконатися, що вона успішна. Деталі будь-якої структури диска або бази даних для зберігання даних потрібно приховувати від програм, щоб, на відміну від **Спільної бази даних**, схема зберігання та деталі могли бути легко змінені, щоб відобразити зміни потреб підприємства. Одна програма повинна мати можливість надіслати пакет даних іншій програмі, щоб викликати поведінку в іншій програмі, наприклад, **віддалений виклик процедури**, але без схильності до відмови. Передача даних повинна бути асинхронною, щоб відправнику не потрібно було чекати на приймачі, особливо коли необхідна повторна спроба.

Проблема інтеграції програмних систем залишається відкритою. Крім існуючих підходів до інтеграції, предмет потребує нових альтернативних підходів до інтеграції, які відкриють нові шляхи для налагодження взаємодії, запропонують нові можливості та покриють потреби корпоративних програмних систем [2]. Інтеграція декількох систем завжди потребує різний набір властивостей і функцій (таблиця), тому системи відрізняються за обсягом, за підходами і технологіями. Інтеграція корпоративних застосунків не проста річ, яка потребує зважених технічних а також бізнес рішень.

Таблиця. Порівняння властивостей існуючих підходів

	Передача файлів	Спільна база даних	RPC	Повідомлення
Зв'язність	+	+	+	–
Швидкість	–	+	–	–
Інтеграція функціональності	–	–	+	+
Асинхронність	–	–	–	+
Можливість відмови	–	–	+	+
Гнучкість	+	–	+	+

Література

1. Rosa-Siqueira F.J., Basto-Fernandes V. Frantz R.Z. (2017) *Enterprise application integration: Approaches and platforms to design and implement solutions in the cloud.*
2. Soomro T.R., Awan A.H. (2012) Challenges and Future of Enterprise Application Integration, UAE, International Journal of Computer Applications.
3. Hohpe G., Woolf B. (2003) Enterprise integration patterns: Designing, building, and deploying messaging solutions, Addison-Wesley Professional.
4. Kent W. (2012) Data and reality, a timeless perspective on perceiving and managing information in our imprecise world. USA Technics Publications, LLC.
5. Kleppman M., (2017) Designing Data-Intensive Applications: The big ideas Behind Reliable, Scalable, and Maintainable Systems, USA, O'Reilly Media, Inc.
6. Sandberg R., Goldberg D., Kleiman S., Walsh D., & Lyon B. (1985). Design and implementation of the Sun network filesystem. *In Proceedings of the Summer*, USA CA, Sun Microsystems, Inc.
7. Kalia A., Kaminsky M., & Andersen D.G. FaSST: Fast, Scalable and Simple Distributed Transactions with Two-Sided (RDMA) Datagram RPCs. *In 12th USENIX Symposium on Operating Systems Design and Implementation*, (OSDI 16: Proceedings of the 12th USENIX conference on Operating Systems Design and Implementation) (P. 185–201).
8. Wollrath A., Riggs R., & Waldo J. (1996). A Distributed Object Model for the Java System, *Conference on Object-Oriented Technologies Toronto*, Ontario, Canada, June 1996.
9. Dell Boomi Home. https://boomi.com, 2017.
10. Informatica Home. https://www.informatica.com/products/cloud-integration, 2017.
11. Jitterbit Home. http://www.jitterbit.com, 2017
12. Mulesoft Home. https://www.mulesoft.com/, 2017.
13. Oracle Cloud Home. https://cloud.oracle.com/integration, 2017.
14. SnapLogic Home. https://www.snaplogic.com, 2017.
15. Frantz R.Z., Corchuelo R. (2016) On the design of a maintainable software development kit to implement integration solutions. *The Journal of Systems and Software, Journal of Systems and Software.* Vol. 111, January 2016. P. 89–104.

References

1. Rosa-Siqueira F.J., Basto-Fernandes V. Frantz R.Z. (2017) *Enterprise application integration: Approaches and platforms to design and implement solutions in the cloud.*
2. Soomro T.R., Awan A.H. (2012) Challenges and Future of Enterprise Application Integration, UAE, International Journal of Computer Applications.
3. Hohpe G., Woolf B. (2003) Enterprise integration patterns: Designing, building, and deploying messaging solutions, Addison-Wesley Professional.

4. Kent W. (2012) Data and reality, a timeless perspective on perceiving and managing information in our imprecise world. USA Technics Publications, LLC.
5. Kleppman M., (2017) Designing Data-Intensive Applications: The big ideas Behind Reliable, Scalable, and Maintainable Systems, USA, O'Reilly Media, Inc.
6. Sandberg R., Goldberg D., Kleiman S., Walsh D., & Lyon B. (1985). Design and implementation of the Sun network filesystem. *In Proceedings of the Summer*, USA CA, Sun Microsystems, Inc.
7. Kalia A., Kaminsky M., & Andersen D.G. FaSST: Fast, Scalable and Simple Distributed Transactions with Two-Sided (RDMA) Datagram RPCs. *In 12th USENIX Symposium on Operating Systems Design and Implementation*, (OSDI 16: Proceedings of the 12th USENIX conference on Operating Systems Design and Implementation) (P. 185–201).
8. Wollrath A., Riggs R., & Waldo J. (1996). A Distributed Object Model for the Java System, *Conference on Object-Oriented Technologies Toronto*, Ontario, Canada, June 1996.
9. Dell Boomi Home. https://boomi.com, 2017.
10. Informatica Home. https://www.informatica.com/products/cloud-integration, 2017.
11. Jitterbit Home. http://www.jitterbit.com, 2017.
12. Mulesoft Home. https://www.mulesoft.com/, 2017.
13. Oracle Cloud Home. https://cloud.oracle.com/integration, 2017.
14. SnapLogic Home. https://www.snaplogic.com, 2017.
15. Frantz R.Z., Corchuelo R. (2016) On the design of a maintainable software development kit to implement integration solutions. *The Journal of Systems and Software, Journal of Systems and Software.* Vol. 111, January 2016. P. 89–104.

Одержано 20.01.2021

Про автора:

Дивак Юрій Андрійович,
аспірант
Інституту програмних систем
Національної академії наук України.

Місце роботи автора:

Integration Full-stack software engineer,
Mediastream company.

E-mail: yurii.dyvak@ukma.edu.ua

УДК 004.434:004.75 https://doi.org/10.15407/pp2021.01.049

В.О. Ларін, О.І. Провотар

ПОРІВНЯННЯ ЕФЕКТИВНОСТІ ПІДХОДІВ MAP-REDUCE І АКТОРНОЇ МОДЕЛІ ПРИ РОЗВ'ЯЗАННІ ЗАДАЧ ІЗ ВИСОКОЮ ЗВ'ЯЗНІСТЮ ВХІДНИХ ДАНИХ НА ПРИКЛАДІ ЗАДАЧІ ОПТИМІЗАЦІЇ РОЮ ЧАСТОК

Показані приклади класу розподілених паралельних задач зі зв'язаними компонентами вхідних даних в рамках моделі Map-Reduce. Виконано порівняння ефективності подібної задачі на прикладі задачі рою часток в рамках моделі Map-Reduce (на основі фреймворку Spark) і акторної моделі з підтримкою спільної пам'яті (на основі Strumok DSL). Оцінено перспективи використання гібридної акторної моделі для інших подібних задач.

Ключові слова: акторна модель, Map-Reduce, оптимізація методом рою частинок, паралельні системи із загальною пам'яттю.

Вступ

В даний час модель Map-Reduce, набула широкого поширення в різних областях – бізнес аналітика, розподілені обчислення, обробка даних для прикладних задач (медицина, комерція, фінанси і т. д.). В основі цієї моделі лежить принцип паралельної обробки і агрегації лінійних масивів даних. Для обробки масиву даних необхідно задати дві функції: map, яка для кожного вхідного елемента співставить від 0 до n вихідних пар ключ-значення, і reduce, яка агрегує всі значення, що відповідають одному ключу [1–3]. При цьому допускається що дана модель – це універсальний примітив паралельних обчислень [4]. Альтернативний підхід – це модель акторів, винайдена в 1973 році, представляючи математичну модель, яка трактує поняття «актор» як універсальний примітив для паралельних обчислень: у відповідь на повідомлення, які він отримує, актор може змінити свій локальний стан, створити нові актори, і відправити повідомлення тим акторам яких він знає [5]. Дана модель залишалася в обмеженому застосуванні до появи фреймворків Akka [6], Akka .NET, Orleans, які значно спростили її практичне використання. При більшій гнучкості моделі акторів (контроль життєвого циклу кожного актора; маршрутизації повідомлень; контрольований супервізор для нештатних ситуацій) залишається питання – наскільки доцільно застосування набагато більш комплексного підходу при вирішенні прикладних задач. Далі розглядатимемо обмеження моделі map-reduce і представлені рішення як для моделі акторів (на основі раніше запропонованої предметно-орієнтованої мови Strumok [7]), так і у вигляді ациклічно направленого графа з серій викликів map-reduce використовуючи фреймворк Apache Spark на прикладі задачі оптимізації методом рою часток.

Задачі зі зв'язаними компонентами вхідних даних і визначення їх в термінах Map-Reduce і акторів

Структура підходу map-reduce передбачає наявність всіх вхідних параметрів перед викликом безпосередньо map-reduce алгоритму. Але не для кожного завдання таке визначення всіх вхідних елементів є можливим. Припустимо, у нас є задача, задана в такий спосіб:

$$S = F(V), V = \{v_i\}_{(i=1)}^{N}.$$

В даному вираженні V вектор великої розмірності $\mathcal{R}^N$. Нехай $O(f) \gg O(F)$, тобто основна обчислювальна складність полягає у знаходженні V, знаючи який пошук $S = F(V)$ не складає великої обчислювальної складності. Масив компонент V

представлений функціонально залежними елементами, які можна обчислити ітераційно:

$$v_i = \{a_{(i,j)}\}_{(j=0)}^{M}, a_{(i,k+1)} = f(a_{(i,k)}),$$

$$a_{(i,0)} = a_i, \text{ де } f: \mathcal{R} \to \mathcal{R}.$$

Така ітерація не може бути представлена одним викликом map-reduce, так як кожен наступний елемент ітерації залежить від попереднього, що в свою чергу порушує головну вимогу операції map – незалежність вхідних елементів один від одного. Але водночас, ми можемо її реалізувати як серію викликів окремих map-reduce підзадач.

Таким чином, в разі якщо обчислення f може бути представлено як незалежне обчислення кожної компоненти вектора $a_{(i,k)}$ з простору $\mathcal{R}^{\mathcal{M}}$, то задача знаходження S у цілому зводиться до ациклічного направленого графу (дерева) глибини N, де елемент буде являти собою задачу map-reduce вигляду $a_{(i,k+1)} = f(a_{(i,k)})$, $\forall i = 1..N$ для вершин із ступенем 1, і $S = F(V)$ для крайньої вершини.

Задача оптимізації рою часток (Particle Swarm Optimization) як приклад класу задач з пов'язаними компонентами вихідних даних

Метод рою часток (МРЧ) – метод чисельної оптимізації, для використання якого не потрібно знати точного градієнта функції що оптимізується [8–10]. МРЧ спочатку призначався для імітації соціальної поведінки [11, 12]. Алгоритм був спрощений, і було помічено, що він придатний для виконання оптимізації. МРЧ оптимізує функцію, підтримуючи популяцію можливих рішень, які називаються частками, і переміщує ці частинки в просторі рішень згідно простої формули. Переміщення підкоряються принципу найкращого знайденого в цьому просторі положення, яке постійно змінюється при знаходженні частками більш вигідних положень. Розглянемо структуру обчислень в цьому алгоритмі. Ітераційно оновлюється набір часток – «рій», які поступово наближаються до екстремумів досліджуваної функції рухаючись з певною швидкістю щодо своїх минулих позицій. Неважко помітити, що така структура співвідноситься з обчисленням масиву залежних елементів і поданням його як ациклічного дерева обчислень у моделі map-reduce. Але, додатково до цього «рій» частинок вносить ще один елемент оптимізації – швидкість руху частинок, а таким чином і кількість ітерацій, залежить від поточного кращого знайденого екстремуму. Це, в свою чергу, створює додатковий вид залежності між вхідними даними (в рамках ітерацій). Спробуємо представити даний параметр апроксимовано нашій моделі. Припустимо наявність зовнішнього для системи параметра L, який дозволяє знайти відразу наступний елемент ітерації:

$$a_{(i,k+2)} = f(a_{(i,k)}), a_{(i,k)} < \text{L}.$$

З цього співвідношення видно, що при хорошій оцінці глобального екстремуму L, рішення може бути знайдено при дворазовому скороченні кількості ітерацій. Таким чином для отримання оптимальної паралельної ефективності методу важливо якомога швидше оновлювати значення L. В реальних задачах рою частинок прискорення залежить від значення параметрів функції швидкості частинки і впливає на стабільність алгоритму. Розглянемо додавання параметра L у систему ациклічного графа map-reduce і акторної моделі. В процесі ітерацій рою часток кожне обчислення $a_{(i,k)}$ дозволяє оновити оцінку L. Оцінимо час синхронізації параметра в різних моделях.

У разі акторної моделі, параметр L може бути негайно оновлений для всіх акторів, які займаються ітеруванням часток. Розглянемо конфігурацію з одним обчислювальним вузлом. Витрати складуть час на доставку й обробку повідомлення будь-кому акторам. Цей час можна оцінити як час обробки одного значення $a_{(i,k)}$ кожним актором. При додаванні спільної пам'яті (Strumok) час оновлення стає

дуже малим і можна говорити про негайне оновлення, так як інші актори будуть безпечно звертатися до області загальної пам'яті, значення в якій можна оновити за кілька тактів процесорного часу.

У разі розподіленої конфігурації час оновлення складатиме часу затримки всередині кластера, і механізм спільної пам'яті дозволить зменшити кількість трафіку що пересилається, і прискорити оновлення в рамках кожного вузла кластера.

У моделі map / reduce, в свою чергу, кожен елемент map повинен зберігати всі дані необхідні для обчислення, таким чином оновлення параметра L можливо тільки між ітераціями, і, крім того, в розподіленої конфігурації час затримки кластера також накладається на швидкість оновлення, тому для подальшого порівняння цей параметр можна опустити, при цьому потенційно акторна модель дозволяє максимально оптимізувати кластерний трафік.

Загальний час оновлення необхідний системі на map-reduce можна оцінити в залежності від співвідношення кількості пар паралельних вузлів / процесорів до кількості часток в ітерації. Якщо співвідношення наближається до рівності (тобто при невеликій кількості часток у популяції) час оновлення map reduce стає відповідним з акторною моделлю. Водночас, при великій кількості часток, раннє оновлення L при перших обчисленнях $a_{(i,k)}$ не відбудеться до повного завершення ітерації, що, по суті, робить оновлення параметра неефективним для більшості часток в ітерації, і призводить до втрати прискорення, яке досягається оновленням L (дворазове в апроксимованому випадку).

Реалізація задачі рою частинок використовуючи підходи map-reduce (Spark) і акторної моделі (Strumok / Akka)

Тепер розглянемо практичну реалізацію даної задачі в рамках акторної моделі. Залежно від величини $\mathcal{N}$ можна делегувати по одному актору на кожен елемент v_i, або ж у разі великих значень $\mathcal{N}$ кожен актор може обробляти певний сегмент значень $v_i..v_{(i+l)}$ заданий при конфігуруванні акторної моделі. У першому (більш тривіальному) випадку логіку акторів можна визначити наступним псевдокодом:

```
Message "Update":
 If k == K then send "Compute
S" to root with Ai and i
 Else Ai = f(Ai); k = k + 1;
send "Update" to self
```

Даний псевдокод дозволяє відразу звернути увагу що проміжне значення $a_{(i,k)}$ зберігається всередині актора, завдяки чому для виконання кожній ітерації досить лише порожнього повідомлення-тригера, пересилати додаткові дані, в тому числі між кластером (в разі розподіленої архітектури) не потрібно. Крім того, кожен актор оновлюється асинхронно, додатковий бар'єр на очікування однієї повної ітерації V_k не потрібен. У той же час в парадигмі Map-Reduce, запуск кожної ітерації вимагає копіювання вхідних даних, ініціалізації завдання на виконавчих вузлах (executors), а також блокування сесії до повного завершення ітерації.

При використанні предметно орієнтованої мови Strumok завдання оптимізації методом рою часток можна представити таким чином:

```
package strumok.pso;

import
strumok.pso.TestFunction;

actor Particle {
  actorref GMinimum handler

  local double minimum
  local double x
  local double value
  local double velocity

  message update() {
    var approxG = ~handler.g
    if
(approxG<=handler.threshold)
    {
      return
    }
```

```
          var rp = Random.value
          var rg = Random.value
          velocity = velocity * 0.3 +
0.7 * rp * (minimum - value) + 1.3 *
rg * (approxG - value)

          x += velocity

          value = TestFunction(x)

          if (value < minimum) {
              minimum  = value
          }

          if (value < approxG) {

handler.updateMinimum(value)
          }
          update()
        }
      }
      actor GMinimum {
        actorref Swarm swarm

        shared double g
        shared double threshold

        message updateMinimum(double
newValue)
        {
          if (newValue < g) {
            g = newValue

            if (g <= threshold) {
               swarm.complete(g)
            }
          }
        }
      }

      actor Swarm {
          @AkkaRoutedPool(strategy =
RoutingStrategy.RoundRobin, size =
1000)
          actorref
Selection<Particle> swarm

          actorref GMinimum handler

          Swarm() {
            handler = new GMinimum()
            broadcast
swarm.initialize(handler)
            broadcast swarm.update()
          }

          message complete(double
result) {
            println("Result is " -
result);
          }
      }
```

Даний лістинг визначає 3 класи акторів: Swarm – керуючий, GlobalMinimum – зберігає поточний глобальний екстремум (параметр L у моделі), Particle – клас обчислювача в рамках ітерації. Частки розміщуються в акторному пулі (розміром 1000), який ініціалізується початковими значеннями часток, а потім оновлює їх ітераційно до досягнення зазначеного threshold (оцінки глобального мінімуму).

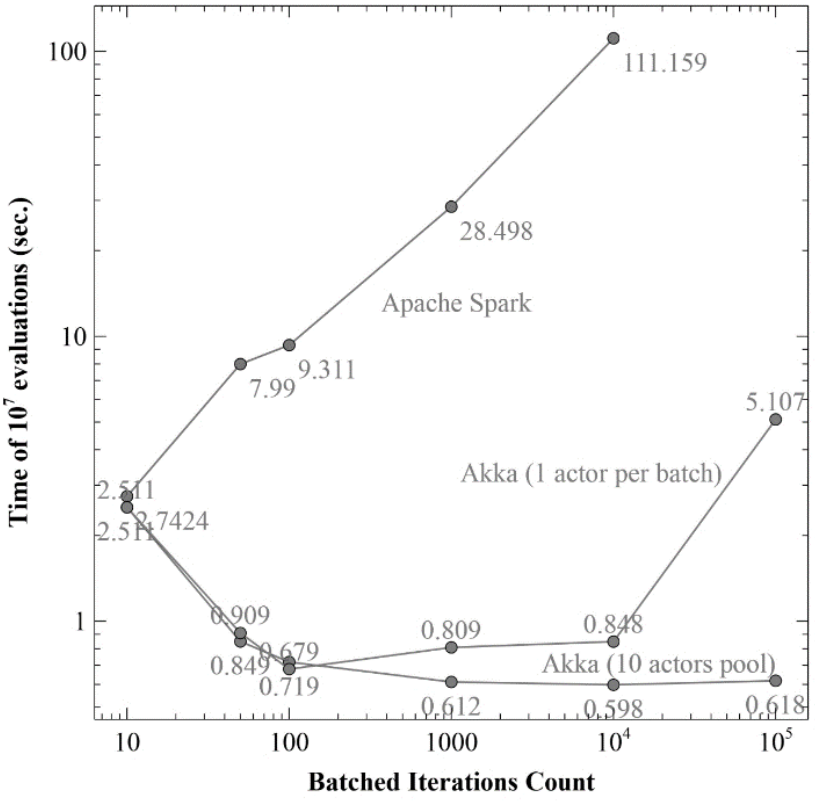
```
val minX = 0
val maxX = 10
val amount = 1000000
val startTime = System.currentTimeMillis()
var particles =
sc.parallelize(generateParticles(minX,
     maxX, amount)).repartition(8)
val globalMinimumValue: Double = parti
   cles.map(_.localMinimum).min()
val globalMinimum = new Minimum
   sc.register(globalMinimum, "minimum")
   globalMinimum.add(globalMinimumValue)

def iterate() = {
 val g = sc.broadcast(globalMinimumValue)
 particles = particles.map(p => {
  val rp = r.nextDouble()
  val rg = r.nextDouble()
  var velocity = p.velocity * 0.7 + 0.7 * rp *
(p.localMinimum - p.current) + 0.3 * rg * (g.value
- p.current)

  //handle out of bounds..
  if (p.x < minX && velocity < 0 || p.x > maxX
&& velocity > 0) {
```

```
    velocity = -velocity
   }

   val x = p.x + velocity
   val current = f(x)
   val minimum = if (current < p.localMinimum)
current
    else p.localMinimum
   if (current < g.value)
globalMinimum.add(current)
   Particle(x, velocity, current, minimum)
  }).localCheckpoint()

  particles.foreach(_ => {})
  globalMinimum.value
}
```

Лістинг вище визначає реалізацію ітерації методу рою часток для map-reduce фреймворка Apache Spark. Дана ітерація визначає глобальний лічильник globalMinimum, який оновлюється між ітераціями, і ітерацію map-reduce, де в map частини оновлюється значення частки, а reduce частина опущена (в реальному алгоритмі може бути вибір максимуму, реалізація якого тривіальна і може бути опущена в тестовій реалізації). Отриманий проміжний екстремум повертається назовні для повторних ітерацій.

Оцінка впливу зв'язності компонентів вхідних даних на ефективність Map-Reduce в порівнянні з акторним підходом

Для кожної версії реалізації задачі була проведена оцінка ефективності роботи. Зважаючи на велику чутливість алгоритму МРЧ до параметрів і недетермінованого характеру алгоритму в цілому, за основу оцінки було взято час обчислення 10 мільйонів часток при збільшенні сумарної кількості ітерацій. Більш складні задачі вимагають більшої кількості ітерацій для вирішення, таким чином ефективність на великих кількостях ітерацій дозволяє оцінити продуктивність системи в реальних умовах. З характеристиками стендової машини: Intel Core i7 3770K 3.5GHz, 16GB 2133 MHz RAM, 128 GB SSD на десяти тестових прогонах отримані наступні усереднені результати (шкали логарифмічні) (рисунок).



Рисунок

Як видно з графіку, Apache Spark дуже важко працює з довгими ациклічними графами виконання (на значеннях ітерацій в 1000 і вище, прискорення акторної моделі стає більш ніж на 2 порядки). Акторна модель реалізована на Strumok / Akka спочатку локально прискорюється (заповнюючи вільні обчислювальні потоки тестової машини), а після цього зберігає продуктивність і при високих розмірах задачі, а при використанні оптимізованої версії (коли кожен актор зберігає відразу кілька часток для поновлення) швидкість роботи зберігається стабільною при будь-якому зростанні ітерацій, які були в рамках експерименту.

Висновки

Перетворення над масивами незалежних наборів даних добре вирішуються в підході map-reduce, але при цьому при появі залежності між вхідними даними альтернативи, такі як модель акторів, а також модель акторів із спільною пам'яттю дозволяють отримувати вкрай істотні прирости в ефективності паралельних обчислень. Клас задач з пов'язаними вхідними даними добре представлений задачами глобальної оптимізації. Поведінка і ефективність такого класу задач були розглянуті на прикладі задачі оптимізації методом рою часток. На окремих конфігураціях прискорення може досягати десятків і сотень разів, що доводить значимість альтернативних підходів для подання паралельних обчислень. Крім того, залишається важливим вивчення залежностей в даних для кроку reduce, що так само може виявити клас задач, які ефективно вирішуються в рамках акторної моделі або подібних альтернатив, але при цьому помітно втрачають в ефективності в рамках map-reduce парадигми.

Література

1. Борис Т.В., Алєксєєв М.О. Порівняльний аналіз технології паралельного обчислення великих масивів даних MapReduce. *Proceedings of Second International Conference "Cluster Computing"*. Lviv , 2013. С. 54–57.
2. Гладкий М.В. Модель распределенных вычислений MapReduse. Труды БГТУ. 2016. Т. № 6. С. 194–198.
3. MapReduce – краткое руководство: веб-сайт. URL: https://coderlessons.com/tutorials/bolshie-dannye-i-analitika/izuchit-kartu-umenshit/mapreduce-kratkoe-rukovodstvo (дата звернення 15 07 2020).
4. Mateu Zaharia. An Architecture for Fast and General Data Processing on Large Clusters. Association for Computing Machinery and Morgan & Claypool. 2016. P. 89.
5. Глибовець М.М., Зінчук С.О. Використання моделі акторів для реалізації розподілених обчислень. *Системні дослідження та інформаційні технології*. 2015. № 2. С. 16–25.
6. Akka: веб-сайт. URL: https://akka.io/ (дата звернення 19.07.2020).
7. Ларін В.О., Бантиш О.В., Галкін О.В., Провотар О.І. Предметно-ориентированный язык Strumok для описания акторных систем с общей памятью. *Кибернетика и системный анализ*. 2018. Т. 54. С. 170–180.
8. Карпенко А.П, Селиверстов Е.Ю. Обзор методов роя частиц для задачи глобальной оптимизации (Particle Swarm Optimization). Наука и образование: электрон. науч.-тех. изд. URL: http://www.technomag.edu.ru (дата звернення 04 08 2020).
9. Курейчик В.В., Запорожец Д.Ю. Роевой алгоритм в задачах оптимизации. *Известия Южного федерального университета.* Технические науки. 2010 р. № 7. С. 28–32.
10. Kennedy J., Eberhart R. Particle Swarm Optimization. *Proceedings of the IEEE International Conference on Neural.* NJ : IEEE Press, 1995. P. 1942–1948.
11. Олійник О.О., Субботін С.О. Оптимізація на основі колективного інтелекту рою часток з керуванням зміною їхньої швидкості. *Радіотехніка. Інформатика. Управління*. 2009. № 2. С. 96–101.
12. Hoorfar A. Evolutionary programming in electromagnetic optimization: a review. *IEEE Trans. Antennas Propag*. 2007. Vol. 55. Iss. 3. P. 523–537.

References

1. Boris T.V., Alekseev M.O., Comparative analysis of MapReduce - technology for parallel computation of large data sets. *Proceedings of the Second International Conference "Cluster Computing"*. Lviv, 2013. P. 54–57.
2. Gladkiy M.V., Model of distributed calculations MapReduse. *Proceedings of BSTU*. 2016. N 6. P. 194–198.
3. MapReduce – a quick guide: a website. URL: https://coderlessons.com/tutorials/bolshie-dannye-i-analitika/izuchit-kartu-umenshit/mapreduce-kratkoe-rukovodstvo (accessed 15 07 2020).
4. Matthew Zechariah, An Architecture for Fast and General Data Processing on Large Clusters. Association for Computing Machinery and Morgan & Claypool. 2016. P. 89.
5. Hlybovets M.M., Zinchuk S.O., Using the model of actors for the implementation of distributed computing. *System research and information technology*. 2015. N 2. P. 16–25.
6. Akka: website. URL: https://akka.io/ (accessed 19 July 2020).
7. Larin V.O., Bantysh O.V., Galkin O.V., Provotar O.I., Object-oriented Strumok language for describing actor systems with shared memory. *Cybernetics and systems analysis.* 2018. Vol. 54. P. 170–180.
8. Karpenko A.P. and Seliverstov E.Y. "Review of particle swarm methods for the global optimization problem (Particle Swarm Optimization)," Nauka i obrazovanie: digital scientific and technical ed. URL: http://www.technomag.edu.ru (access date 04 08 2020).
9. Kureychik V.V., Zaporozhets D.Yu. Swarm algorithm in optimization problems. *Izvestiya Yuzhnogo federalnogo universiteta*. Technical sciences. 2010. Vol. 7. P. 28–32.
10. Kennedy J., Eberhart R. Particle Swarm Optimization. *Proceedings of the IEEE International Conference on Neural*. NJ: IEEE Press, 1995. P. 1942–1948.
11. Oliynyk O.O., Subbotin S.O., Optimization on the basis of collective intelligence of a swarm of particles with control of change of their speed. *Radiotechnics. Informatics. Management*. 2009. Vol. 2. P. 96–101.
12. Hoorfar A. Evolutionary programming in electromagnetic optimization: a review. *IEEE Trans. Antenna Propag*. 2007. Vol. 55. Iss. 3. P. 523–537.

Одержано 24.02.2021

Про авторів:

Провотар Олександр Іванович,
доктор фізико-математичних наук,
професор, завідувач кафедри
інтелектуальних програмних систем.
Кількість наукових публікацій в
українських виданнях – 150.
Кількість наукових публікацій в
зарубіжних виданнях – 45.
Н-індекс – 5,
http://orcid.org/0000-0002-6556-3264,

Ларін Владислав Олегович,
аспірант.
Кількість наукових публікацій в
українських виданнях – 4.
Кількість наукових публікацій в
зарубіжних виданнях – 1.
Тел.: +380972712208.
E-mail: vlarinmain@gmail.com.

Місце роботи авторів:

Київський національний університет
імені Тараса Шевченка,
03187, Київ-187,
Проспект Академіка Глушкова, 4д.
Факультет комп'ютерних наук
та кібернетики.

UDC 004.62 https://doi.org/10.15407/pp2021.01.056

I.S. Chystiakova

MAPPING OF THE DESCRIPTIVE LOGIC INTO RDF USING BINARY RELATIONAL DATA MODEL

This paper is dedicated to the data integration problem. The descriptive logic and the relational data model are at the heart of a study. They have been used to create a mapping method on the theoretical level. The previous studies are continued in this paper to prove on practice a mapping creation method between the descriptive logic and the binary relational data model, which is a part of a mapping method. The method uses the binary relational data model as an integrating model. The task to prove the theoretical mapping method on practice was formulated. A question how to map the binary relational data model into RDF-triples was considered. A brief overview of the R2R ML conversion tool was given. Triple maps were created to convert a conceptual information model of descriptive logic into RDF triplets with the help of R2R ML. Also, triples maps are described to convert basic mapping mechanisms into RDF with the help of R2R ML.
Key words: binary relational data model, description logic, mapping, RDF, DL, RM^2, ALC, OWL.

Introduction

A complex problem of data integration in the semantic web exists in the modern scientific field of research. An analysis of this problem can be found in the [1]. A task to establish an interaction between a descriptive logic (DL) and a relational data model (RDM) arises as a part of solution of the data integration problem. Such interaction is called mapping. To establish an interaction means to create mapping mechanisms between the DL and the RDM. A series of studies [1–7] is dedicated to the analysis and solution of the mapping creation problem. **A binary relational data model (RM^2)** [6] was created as a result of this series. The main task of RM^2 is to be an integrating model for creating mappings. How to map the descriptive logic ALC and its main components into RM^2 was described in the study [8] as well as how to map the classical RDM into RM^2. This approach was described purely on a theoretical level. Until now, the lack of any practical approbation was a significant drawback of the proposed results.

A method to test mappings between DL and RM^2 with the help of RDF graphs is proposed in this paper. The main idea is to map DL-to-RDM conversion formulas into RDF, and then to test them for workability within the unified RDF framework. The results of mapping DL expressions into RDF using OWL 2 were published in [9]. This study focuses on creation of mappings for RDM expressions into RDF using R2R ML.

Section 1 is dedicated to the problem statement. Section 2 provides a short overview of the R2R ML. Section 3 summarizes the main theoretical aspects of the DL-RDM mapping method. Section 4 describes the RM^2 to RDF mapping rules using R2R ML. Section 5 contains conclusions.

Problem statement

A theoretical approach how to describe mappings between DL and RDM is presented in a series of studies [1-9]. A binary relational data model (RM^2) is proposed as an integrating model. The approbation task of this approach is based on a number of facts.

Firstly, it is known that DL is the mathematical basis of any ontology description language. Thus, all the constructors of concepts and roles of the underlying DL are reflected in the toolbox of the corresponding language. OWL 2 is not an exception.

Secondly, only binary connections are allowed in RM^2. Both binary and n-ary connections are allowed in a classical RDM. A result of [10] seems to indicate that any n-ary relation can be represented by a set of binary ones. Thus, any classic RDM can be expressed with RM^2. A way how to convert RDM into RM^2 is described in [6].

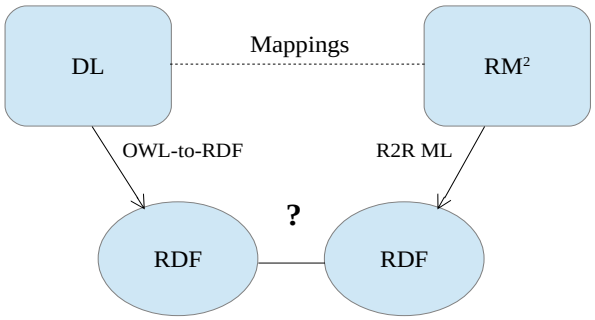
The idea of testing is not new. The statements of the theory being proved are transformed into statements of the established theory. The converted expressions are then checked for truth within the well-established

theoery using its own methods and properties. If the final expression is true in the existing theory, then the original expression is also true in the study area.

Thus, the problem statement ti test mappings between DL and RDM is formulated as follows. On the one hand, DL statements (expressed in OWL 2) are mapped to the RDF triplets using OWL 2-to-RDF conversion rules. On the other hand, relational database (RDB) expressions are mapped to the RDF triplets using R2R ML. The resulting RDF triplets constitute a set of RDF graphs. The resulting graphs are compared for equivalence.

The implementation idea is schematically shown in Figure 1.

Figure1. Approbation scheme of the mapping



method between DL and RM^2

OWL 2-to-RDF conversion rules have official W3C status [11]. R2R ML also has official W3C status [12].

The algorithm for testing mappings between DL and RM^2 is as follows:

Step 1. There is a DL expression. It is mapped into the RM2 statement.

Step 2. The statement is mapped intto an OWL 2-expression from the DL side.

Step 3. OWL 2-expression (step 2) is converted to RDF-triples that form an RDF-graph. OWL 2-to-RDF rules are used for mapping.

Step 4. The statement is formulated in terms of RDB from the RM2 side.

Step 5. RDB expression is converted to the RDF-triples that form an RDF-graph. R2R ML is used to create mappings.

Step 6. The RDF-graph (step 3) is compared for equivalence with another RDF-graph (step 5). If they are equivalent, then the DL-to-RM2 mapping formula is true.

The mappings from step 1 are described in [8] on the theoretical level. Also, they are briefly described in section 3. The mappings from steps 2 and 3 are described in article [9]. The current work will present a way to perform transformations from steps 4 and 5. The comparison from step 6 remains in the field of future research.

It is known that OWL 2 is based on the SROIQ descriptive logic. Thus, the OWL 2-to-RDF mapping area is limited to only those operations that are present in DL SROIQ. DL SROIQ syntax include the following: DL ALC syntax, DL axiomatics, numerical constraints, nominals, and inverse roles. The theoretical part of DL-to-RM^2 mappings has been worked out for several role constructors. The issue of mapping some of the role constructors in RDF (except the inverse role) remains open.

There are a number of features in R2R ML. It allows you to transform the structure and integrity constraints of an RDB into RDF triplets. However, there are some features of mapping the manipulative part of RDM into RDF. Any operation can be mapped only as part of an SQL query. Each SQL query is represented as a logical table within a triples map generated for such table. Thus, there are no mechanisms to transform directly each individual operation of relational algebra (RA) within R2R ML itself. A small overview of R2R ML is presented in section 2.

A key question of the approbation task is to prove the equivalence of graphs, obtained as a result of pairwise mapping of the DL and RM^2 statements. The results of [13] demonstrate that RDF-graph is a special case of a regular graph. This means that the question of equivalence is reduced to proving their isomorphism. In the work [13] RDF-graph is analyzed as a special case of a usual graph. Several criteria for graph isomorphism in the general case are also studied. Based on these criteria, three necessary and sufficient conditions for the RDF-graphs equivalence of are formulated. They are as follows:

1. *Equal number of vertices*. Both graphs must contain the same number of vertices, otherwise they are not isomorphic.

2. *Vertices equivalence*. Each vertex of one graph must have an equivalent in the other graph in a pairwise comparison. Otherwise, such graphs are not isomorphic.

3. *Ribs equivalence*. Each edge of one graph must have an equivalent in the other graph in a pairwise comparison. Otherwise, such graphs are not isomorphic.

Reducing a graph to a self-isomorphic remains the last question in the problem framework. As a result of mappings, the following situation may arise at the RDF level. The vertex of one graph will semantically correspond to a subgraph of the comparable graph. Such a subgraph can consist of several vertices connected by edges. This situation is possible, since a large number of anonymous (empty) nodes appears during the mapping process. Such nodes have their own semantic purpose. Thus, the question of reducing an RDF-graph to the self-isomorphic remains open.

Preliminaries: R2R ML Overview

Here is a brief overview of the R2R ML. It will be used in Section 4 to map RM^2 expressions into RDF.

The R2R ML has a development history. Tim Berners-Lee published an article [14] in 1998. It was entitled as "Relational Databases on the Semantic Web". This paper discusses the concept of presenting any database in the semantic web. Its main idea is to use RDF as an ER model. The ER model establishes a correlation between relational database elements and RDF-triples. The author of the concept proposes to use an XML-format of serialization of RDF-triples. The paper also presents the "bottlenecks" of mapping of the RDB information into RDF.

Then, in 2007, the conference "W3C Workshop on RDF Access to Relational Databases" was held [15]. It was dedicated to the presentation of ordinary relational data in RDF format, as well as the use of RDF in RDB queries. The W3C RDB2RDF Incubator Group was established in 2009 and operated until 2012. It declared the following goals:

- to study and classify existing approaches of mapping the relational data in RDF;
- to determine the need for standardization of this area;
- to determine the mechanisms of generation the RDF-triples from one or more relational databases without the loss of information;
- to study the possibility of mapping OWL classes into relational data, taking into account the developed approach.

As a result, a document with official R2R ML recommendations was received and published. Figure 2 shows the chronology of R2R ML development [16].

Now it is necessary to consider the main standings of R2R ML. It`s important to understand how RDF-triples are formed from relational data. Let's turn to the official documentation [12].

R2R ML – is a language for expressing customized mappings from relational databases to RDF datasets. It defines the mapping of the relational database into the RDF. An R2RML mapping refers to *logical tables* to retrieve data from the *input database*. The input to an R2R ML mapping is called **the input database**. Figure 3 highlights a UML diagram of the R2R ML language overview. The picture is taken from the official R2R ML website.

An R2R ML mapping – is a structure that consists of one or more *triples maps*. **A triples map** is a rule that maps each row in the logical table to a number of RDF triples. The rule has two main parts: *a subject map* and multiple *predicate-object map*.

A subject map generates the subject of all RDF triples that will be generated from a logical table row. The subjects are often IRIs that are generated from the primary key column(s) of the table.

Predicate-object maps in turn consist of *predicate maps* and *object maps*. Sometimes predicate-object map additionally has *referencing object maps*.

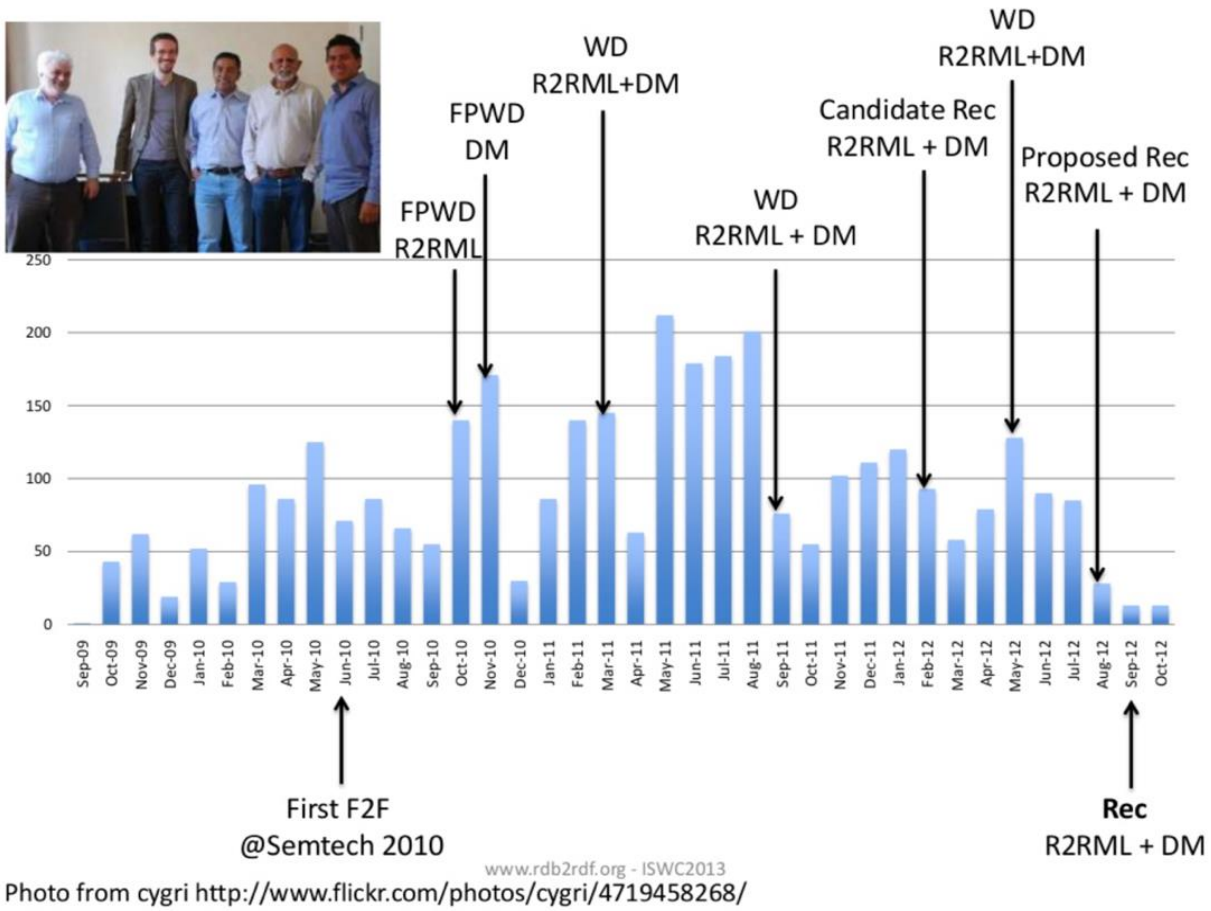


Figure 2. Chronology of the R2R ML development

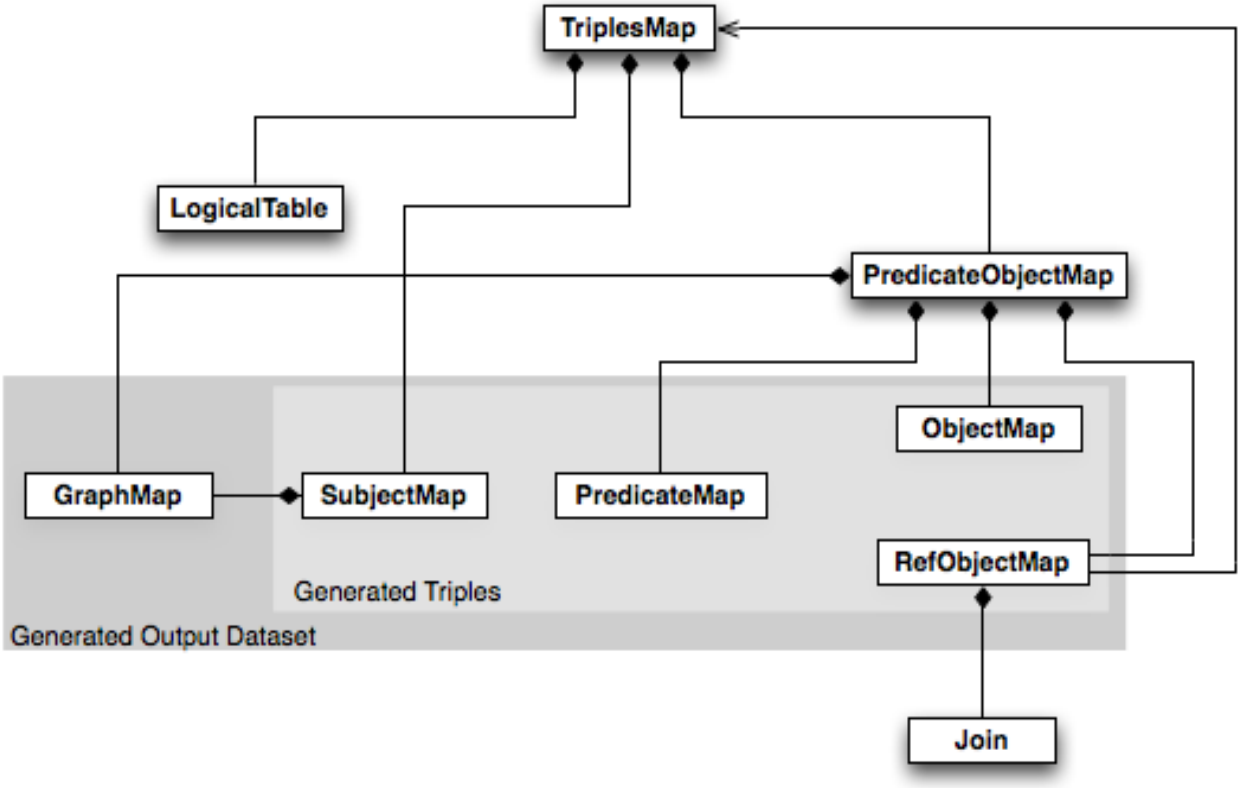


Figure 3. R2R ML overview

Triples are produced as follows. Subject map is combined with a predicate map and object map. These three are applied to each logical table row. By default, all RDF triples are in the default graph of the output dataset. A triples map can contain graph maps. Such graph maps place some or all of the triples into named graphs instead of the default graph.

Triples map (fig. 2) consists of the three main components: *a logical table*, *a subject map*, *a predicate-object map*. In detail each of them is as follows.

A logical table is a tabular SQL query result that is to be mapped to RDF triples. In simple words a logical table is **what** will be displayed. It can take one of the following three forms:

- SQL base table;
- SQL view;
- R2R ML view (a valid SQL query). It got its name because it only emulates a SQL view without modifying the database.

A **logical table row** is a row in a logical table. It can be a row of a base SQL table or SQL view. It can also be a row of an R2R ML view obtained with an SQL query.

A **column name** is the name of a column of a logical table. A column name must be a *valid SQL identifier*. For example, the name of a SQL object, such as a column, table, view, schema, or catalog. Column names do not include any qualifying table, view or schema names.

The logical table in the triples map is written using one of the properties:

- *rr:tableName* – specifies the table or view name of the base table or view. Its value must be a valid schema-qualified name. It is a sequence of one, two or three valid SQL identifiers, separated by the dot character (“.”). The three identifiers name, respectively, a catalog, a schema, and a table or view. If no catalog or schema is specified, then the default catalog and default schema of the SQL connection are assumed.

- *rr:sqlQuery* and *rr:sqlVersion* – defines R2R ML view and SQL query version. R2R ML view is a logical table whose contents are the result of executing a SQL query against the input database. If rr:sqlVersion property is absent, then the rr:sqlQuery property value conforms to Core SQL 2008.

Before considering the subject maps and the predicate-object maps, it’s necessary to give a number of definition.

An **RDF term** is either an IRI, or a blank node, or a literal.

A **term map** is a function that generates an *RDF term* from a *logical table row*. The result of that function is a generated RDF term. Term maps are used to generate the subjects, predicates and objects of the RDF triples. In turn, RDF triples are generated by a triples map. Consequently, there are several kinds of term maps, depending on where in the mapping they occur: subject maps, predicate maps, object maps and graph maps. The *referenced columns* of a term map are the set of column names referenced in the term map. They depend on the type of term map.

A **subject map** is a term map. It specifies a rule for generating the subjects of the RDF triples.

In the triples map, the subject map is specified as follows:

- *rr:subjectMap* is a property, which value must be a specific subject map;

- *rr:subject* is a shortcut constant property whose value is IRI.

The subject map can contain the *rr:class* property. Its value must be an IRI, which is called class IRI. In this case, the RDF expression generated by the subject map will look like this. For each subject, a triple is created with the *rdf:type* predicate and the *rr:class* property as object value. A subject map can contain several class IRIs at the same time. There are cases when the class IRI must be computed based on the contents of source database. In such situations, a predicate object map is used. The predicate value is indicated by rdf: type. The value of an object is set through a non-constant object map.

A **predicate-object map** is a function that creates one or more predicate-object pairs for each logical table row of a logical table. It is used in conjunction with a subject map to generate RDF triples in a triples map. A predicate-object map is represented by a resource that references: one or

more predicate maps and one or more object maps.

A **predicate map** is a term map. It can be defined in two ways:

- *rr:predicateMap* is a property, whose value must be a predicate map;
- *rr:predicate* is a constant shortcut property whose value is IRI.

An **object map** is a term map. It can be defined in two ways:

- *rr:objectMap* is a property, whose value must be either an object map, or a referencing object map;
- *rr:object* is a constant shortcut property whose value is IRI or literal.

A **referencing object map** allows using the subjects of another triples map as the objects generated by a predicate-object map. Since both triples maps may be based on different logical tables, this may require a join between the logical tables. However, the join condition (one or more joins) is optional.

A referencing object map is represented by the following resources:

- *rr:parentTriplesMap* is a property, whose value must be a triples map. Such triples map is known as the referencing object map's parent triples map. The value of object will be extracted exactly from the parent triples map.
- *rr:joinCondition* is a property whose values must be join conditions options.

A **join condition** is represented by a resource that has exactly **one value** for each of the following two properties:

- *rr:child* is a property, whose value is known as the join condition's child column. It must be a column name that exists in the logical table of the triples map (that contains the referencing object map).
- *rr:parent* is a property whose value is known as the join condition's parent column. It must be a column name that exists in the logical table of parent triples map (of the referencing object map's). The name of the parent triples map was specified in the rr:parentTriplesMap property.

Here is one of the examples given in the description of the R2R ML standard [12]. The following example database consists of two tables, EMP (table 1) and DEPT (table 2), with one row each:

Table 1

EMP

EMPNO Integer primary key	*EName Varchar (100)*	*Job Varchar (20)*	*DEPTNO Integer references DEPT (DEPTNO)*
7369	*Smith*	*Clerk*	*10*

Table 2

DEPT

DEPTNO Integer primary key	*DName Varchar (30)*	*Loc Varchar (100)*
10	*Appserver*	*New York*

The following R2R ML mapping document will produce the desired triples from the EMP table:

```
@prefix rr: <http://www.w3.org/ns/r2rml#>.
@prefix ex: <http://example.com/ns#>.
<#EmpMap>
rr:logicalTable [ rr:tableName "EMP" ];
rr:subjectMap [
rr:template "http://data.example.com/employee/{EMPNO}";
rr:class ex:Employee;
];
rr:predicateObjectMap [
rr:predicate ex:name;
rr:objectMap [ rr:column "EName" ];
];
rr:predicateObjectMap [
rr:predicate ex:job;
rr:objectMap [ rr:column "Job" ];
];
rr:predicateObjectMap [
rr:predicate ex:department;
rr:objectMap [
rr:parentTriplesMap <#DeptMap>;
rr:joinCondition [
```

rr:child "DEPTNO";
rr:parent "DEPTNO";
];
].

The definition of a triples map that generates the desired DEPT triples follows.

<#DeptTableView> rr:sqlQuery """
SELECT DEPTNO, DName, Loc,
(SELECT COUNT(*) FROM EMP WHERE EMP.DEPTNO=DEPT.DEPTNO) AS Staff
FROM DEPT;
""".
<#DeptMap>
rr:logicalTable <#DeptTableView>;
rr:subjectMap [
rr:template "http://data.example.com/department/{DEPTNO}";
rr:class ex:Department;
];
rr:predicateObjectMap [
rr:predicate ex:name;
rr:objectMap [rr:column "DName"];
];
rr:predicateObjectMap [
rr:predicate ex:location;
rr:objectMap [rr:column "Loc"];
];
rr:predicateObjectMap [
rr:predicate ex:staff;
rr:objectMap [rr:column "Staff"];
].

The desired RDF triples to be produced from this database are as follows:

<http://data.example.com/employee/7369> rdf:type ex:Employee.
<http://data.example.com/employee/7369> ex:name "Smith".
<http://data.example.com/employee/7369> ex:job "Clerk".
<http://data.example.com/employee/7369>ex:department<http://data.example.com/department/10>.
<http://data.example.com/department/10> rdf:type ex:Department.
<http://data.example.com/department/10> ex:name "Appserver".
<http://data.example.com/department/10> ex:location "New York".
<http://data.example.com/department/10> ex:staff 1.

Preliminaries: mapping DL into RM^2

The approbation task of testing mappings between DL and RDM is based on a series of theoretical studies [1–7]. Here is a brief summary of them. This summary will be used in Section 4 to create mappings from RM^2 to RDF.

Binary Relational Data Model (RM^2) [6] was developed to address the issue of establishing relations between DL and RDM. It has several advantages over the classic RDM by Codd [17]. For example, RM^2 contains support for the open world assumption, while classical RDM works according to epy closed world assumption. Unlike Codd's RDM, RM^2 supports the implementation of DL constructors and concepts and roles axioms. The RM^2 is described in detail in [6].

To describe mappings, the first step is to build a conceptual information model of descriptive logic. The main task of the conceptual information model of any subject area is to define the basic concepts and to describe their properties and relations. The ER language is one of the most used for this purpose. It operates with the concepts of entity, attribute and relation. The Barker dialect [18] of the ER language was used to describe the conceptual information model of descriptive logic (Fig. 4).

There are three basic entities in the model:

- Concept – to present DL concepts
- Role – to present DL roles
- CIndividual – to present DL individuals

Each entity has a single attribute that is called “Name”. This attribute uniquely identifies the entities. The rest of entities are relation entities. They represent binary relations between basic entities. Link entities do not have their own attributes and are uniquely identified only by their links.

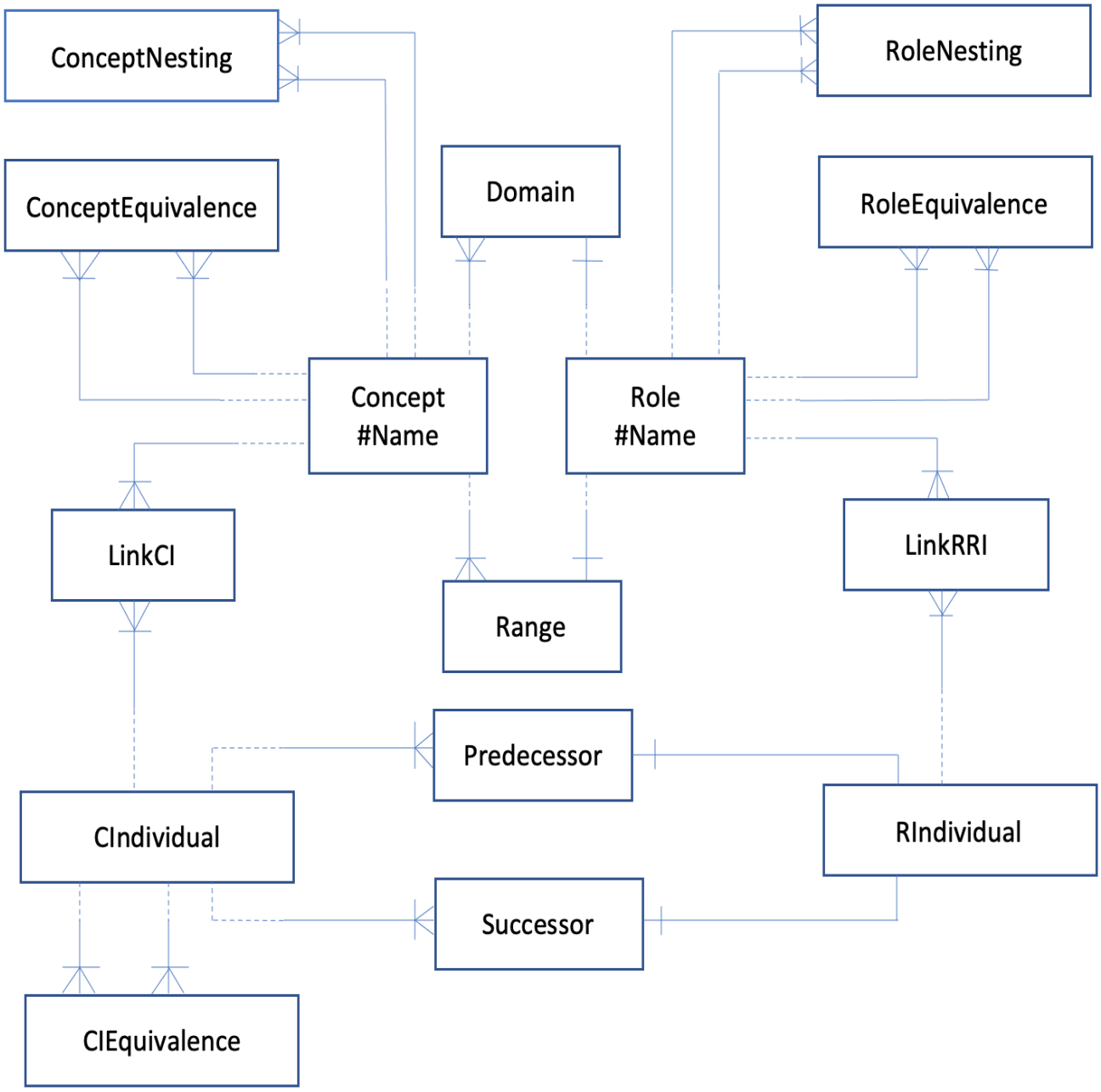


Figure 4. Conceptual information model of descriptive logic

The RM^2 scheme was built according to the given ER-model. The transformation algorithm described in [2, 6, 8] was used to construct the scheme. Additionally, the idea of placeholder attributes was used to represent the primary keys. This idea firstly was proposed by E. Codd [17], the founder of RDM. Each entity is represented as a RM^2 relationship, since the constructed ER-scheme fits the 3NF requirement. The name of the relationship is the same as the name of the entity. Table 3 highlights all the ER-model entities with their descriptions and the corresponding relationships with the list of attributes.

To describe mappings, the second step is to express DL ALC in RM^2. ALC is the simplest variant of DL. It is included in all widely used dialects of other DLs. It should be recalled, that description logics uses constructs that have semantics given in predicate logic. The ALC semantics is defined through the concept of interpretation.

Table 3

ER-model entities with their descriptions and corresponding relationships

Entity (relationship) name	Entity description	Relationship attributes with description
Concept	Presents concepts	CPK – primary key Name – concept name
Role	Present roles	RPK – primary key Name – role name IsTransitive – is role transitive
CIndividual	Present individuals	CIPK – primary key Name – individual name
RIndividual	Pesent role individuals	RIPK – primary key
Domain	Present role domain	CFK – foreign key on Concept RFK – foreign key on Role
Range	Present role range	CFK – foreign key on Concept RFK – foreign key on Role
LinkCI	Allows many-to-many relations between concepts and individuals	CFK – foreign key on Concept CIFK – foreign key on CIndividual
LinkRRI	Allows many-to-many relations between roles and their instances	RFK – foreign key on Concept RIFK – foreign key on RIndividual
Predecessor	Presents the first individual of a role individual	CIFK – foreign key on CIndividual RIFK – foreign key on RIndividual
Successor	Presents the second individual of a role individual	CIFK – foreign key on CIndividual RIFK – foreign key on RIndividual
Concept Nesting	Represents the concept inclusion (hierarchy) axiom	CInFK – foreign key on Concept («child») COutFK – foreign key on Concept («parent»)
Concept Equivalence	Represents the concept equivalence axiom	CForFK – foreign key on Concept («which is equal») CIsFK – foreign key on Concept («equals to which»)
Role Nesting	Represents the role inclusion (hierarchy) axiom	RInFK – foreign key on Role («child») ROutFK – foreign key on Role («parent»)
Role Equivalence	Represents the role equivalence axiom	RForFK – foreign key on Role («which is equal ») RIsFK – foreign key on Role («equals to which »)
CIEquivalence	Represents the individual equivalence axiom	CIForFK – foreign key on CIndividual («which is equal ») CIIsFK – foreign key on CIndividual («equals to which »)

Interpretation is a pair $I = (\Delta, \bullet^I)$, where

- Δ – a non-empty set called the domain of interpretation,
- $\bullet I$ – interpretation function.

Interpreter function assigns each atomic concept A a set $A^I \subseteq \Delta$, and each atomic role R a binary relationship $R^I \subseteq \Delta \times \Delta$.

Further, $C^E_{RM^2}$ will denote the RM^2 relationship extensional. This relationship corresponds to the interpretation of an arbitrary concept C.

Table 4 shows the DL ALC syntax and semantics, as well as their corresponding mapping formulas.

Table 4

ALC syntax and semantics and their corresponding mapping formulas

Syntax	**Semantics**	**Mapping**
1	**2**	**3**
ALC Syntax Concepts		
$\top$	$\top^I = \Delta$	$\top^E_{RM^2} = \pi_{Name}(CIndividual)$
$\bot$	$\bot^I = \emptyset$	Empty(Name)
C	$C^I \subseteq \Delta^I$	$C^E_{RM^2} = = \pi_{CIndividual.Name}(\sigma_{Concept.Name='C'} (CIndividual \bowtie_{CIPK=CIFK} (LinkCI \bowtie_{CFK=CPK} Concept)))$
R	$R^I \subseteq \Delta^I \times \Delta^I$	$R^E_{RM^2} = \pi_{First,Second}(\rho_{CIdividual.Name/Second}(CIndividual \bowtie_{CIPK=CFPK} (Successor \bowtie_{RIFK=RIPK} (\rho_{CIndividual.Name/First}(CIndividual \bowtie_{CIPK=CIFK} (Predecessor \bowtie_{RIFK=RIPK} (\sigma_{Role.Name='R'}(RIndividual \bowtie_{RIPK=RIFK} (LinkRRI \bowtie_{RFK=RPK} Role)))))))))$
$\neg C$	$\Delta^I \backslash C^I$	$(\neg C)^E_{RM^2} = \pi_{Name}(CIndividual) - C^E_{RM^2}$
$C \sqcap D$	$(C \sqcap D)^I = C^I \cap D^I$	$(C \sqcap D) = C^E_{RM^2} \cap D^E_{RM^2}$
$C \sqcup D$	$(C \sqcup D)^I = C^I \cup D^I$	$(C \sqcup D) = C^E_{RM^2} \cup D^E_{RM^2}$
$\exists R.C$	$\exists R.C = \{a \in \Delta \mid \exists b \in \Delta((a,b) \in R^I \wedge b \in C^I)\}$	$(\exists R.C)^E_{RM^2} = \pi_{First}(R^E_{RM^2} \bowtie_{Second=Name} C^E_{RM^2})$
$\forall R.C$	$\forall R.C = \{a \in \Delta \mid \forall b \in \Delta((a,b) \in R^I \rightarrow b \in C^I)\}$	$(\forall R.C)^E_{RM^2} = \pi_{First}(R^E_{RM^2}) - \pi_{First}(R^E_{RM^2} \cap (\pi_{First}(R^E_{RM^2}) \times (\pi_{Second}(R^E_{RM^2}) - \pi_{Name}(C^E_{RM^2}))))$
Number restrictions, nominals		
$(\geq nR)$	$(\geq nR)^I = \{e \in \Delta \mid \lvert R^I(e)\rvert \geq n\}$	$(\geq nR)^E_{RM^2} = \pi_{First}(\underset{1\leq i<j\leq n}{\sigma} R_i.Second_i \neq R_j.Second_j(\underset{1\leq i\leq n}{*} \rho_{R_i(First,Second_i)}(R^E_{RM^2})))$
$(\leq nR)$	$(\leq nR)^I = \{e \in \Delta \mid \lvert R^I(e)\rvert \leq n\}$	$(\leq nR)^E_{RM^2} = \pi_{Name}(Cindividual) - \pi_{First}(\underset{1\leq i<j\leq n+1}{\sigma} R_i.Second_i \neq R_j.Second_j (\underset{1\leq i\leq n+1}{*} \rho_{R_i(First,Second_i)}(R^E_{RM^2})))$
$(\geq nR.C)$	$(\geq nR.C)^I = \{e \in \Delta \mid \lvert R^I(e) \cap C^I\rvert \geq n\}$	$(\geq nR.C)^E_{RM^2} = \pi_{First}(\underset{1\leq i<j\leq n}{\sigma} R_i.Second_i \neq R_j.Second_j(\underset{1\leq i\leq n}{*} \rho_{R_i(First,Second_i)}(R^E_{RM^2} \bowtie_{Second=Name} C^2_{RM^2})))$
$(\leq nR.C)$	$(\leq nR.C)^I = \{e \in \Delta \mid \lvert R^I(e) \cap C^I\rvert \leq n\}$	$(\leq nR.C)^E_{RM^2} = \pi_{Name}(CIndividual) - \pi_{First}(\underset{1\leq i<j\leq n}{\sigma} R_i.Second_i \neq R_j.Second_j (\underset{1\leq i\leq n}{*} \rho_{R_i(First,Second_i)}(R^E_{RM^2} \bowtie_{Second=Name} C^E_{RM^2})))$
$\{a\}$	$\{a\}^I$	$\{a\}^E_{RM^2} = \{a^E_{RM^2}\}$
Role Constructors		
R^-	$(R^-)^I = \{(e,d) \in \Delta \times \Delta \mid (d,e) \in R^I\}$	$(R^-)^E_{RM^2} = (\rho_{R(Second,First)}(R^E_{RM^2}))$

1	2	3
¬R	$(\neg R)^I = \Delta \times \Delta \backslash R^I$	$(\neg R)^E_{RM^2} = (\rho_{Name/First}(\pi_{Name}(CIndividual)) \times \rho_{Name/Second}(\pi_{Name}(CIndividual))) - R^E_{RM^2}$
R ⊓ S	$(R \sqcap S)^I = R^I \cap S^I$	$(R \sqcap S)^E_{RM^2} = R^E_{RM^2} \cap S^E_{RM^2}$
R ⊔ S	$(R \sqcup S)^I = R^I \cup S^I$	$(R \sqcup S)^E_{RM^2} = R^E_{RM^2} \cup S^E_{RM^2}$
R ∘ S	$(R \circ S)^I = \{(e,d) \in \Delta \times \Delta \mid \exists c \in \Delta((e,c) \in R^I \wedge (c,d) \in S^I)\}$	$(R \circ S)^E_{RM^2} = \pi_{R.First,S.Second}(R^E_{RM^2} \bowtie_{R.Second=S.First} S^E_{RM^2})$
id(C)	$(id(C))^I = \{(e,e) \in \Delta \times \Delta \mid e \in C^I\}$	$(id(C))^E_{RM^2} = (\rho_{Name/First}(\pi_{Name} C^E_{RM^2})) \bowtie_{First=Second} (\rho_{Name/Second}(\pi_{C.Name} C^E_{RM^2}))$
R^+	$(R^+)^I = \bigcup_{n \geq 1} (R^I)^n$	$(R^+)^E_{RM^2} = (R^E_{RM^2})^+$
R^*	$(R^*)^I = \bigcup_{n \geq 0} (R^I)^n$	$(R^*)^E_{RM^2} = (\rho_{Name/First}(\pi_{Name} CIndividual)) \bowtie_{First=Second} (\rho_{Name/Second}(\pi_{Name} CIndividual)) \cup (R^+)^E_{RM^2}$

The DL axiom mapping has been shown in the conceptual ER scheme. Each axiom has its own entity. Each axiom has its own binary relationship in RM^2. Each relationship has two foreign keys. Each of the keys refers to concepts, roles or individuals about which the axiom is formulated.

ConceptNesting (CInFK, COutFK)

$C \sqsubseteq D$

ConceptEquivalence (CForFK, CIsFK)

$C \equiv D$

RoleNesting (RInFK, ROutFK)

$R \sqsubseteq S$

RoleEquivalence (RForFK, RIsFK)

$R \equiv S$

CIEquivalence (CIForFK, CIIsFK)

a = b

Role(RPK, Name **IsTransitive**)

TR(R)

Mapping RM^2 into RDF

RM^2 to RDF mappings can be meaningfully divided into several parts. Firstly, how to transform each RDB relationship of an RM^2 will be shown. All RDB relationships can be divided into the following groups: basic relationships (concepts, roles, individuals), connective relationships and axiom relationships. The mapping of all ALC constructs will be shown next. The description of mapping mechanisms for number restrictions, roles restrictions and nominals completes this section.

Since empty relationships map to an empty RDF graph, there are a number of rows in each relationship to be an example. These strings will be mapped to the RDF triples using R2R ML triple maps. These triple maps are the mechanism for mapping RDB relationships into RDF.

Examples of mapping are present for only one row of each logical table. This is done to save space and to emphasize the rules themselves, not just their use.

Turtle syntax was used to describe triple maps, as well as the following notation:

@prefix rr:<http://www.w3.org/ns/r2rml#>

@prefix ex: http://example.com/Ch#

1. Basic concepts

1.1. Concept

SQL table Concept

CPK	Name
1	C
2	D
3	E
4	F

R2R ML Triple Map

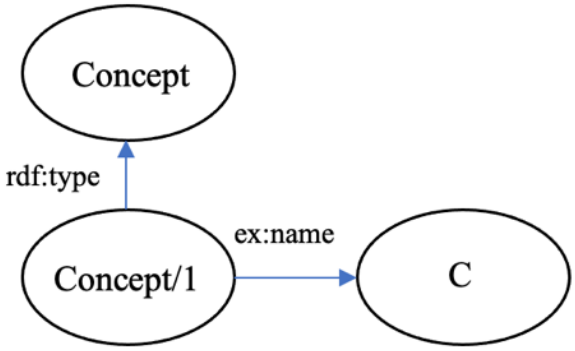
<#TriplesMap1>

rr:logicalTable [rr:tableName «Concept»];
r:subjectMap [
rr:template «http://example.com/Ch#/Concept/{CPK}»;
rr:class ex:Concept;
]
rr:predicateObjectMap [
rr:predicate ex:name;
rr:objectMap [rr:column «Name»];
]

RDF output example

<http://example.com/Ch#/Concept/1> rdf:type ex:Concept.
<http://example.com/Ch#/Concept/1>ex:name"C

RDF output example graph



1.2. Role

SQL table Role

RPK	Name	IsTransitive
56	R	No
67	S	No
89	T	No
34	Z	No
23	U	Yes

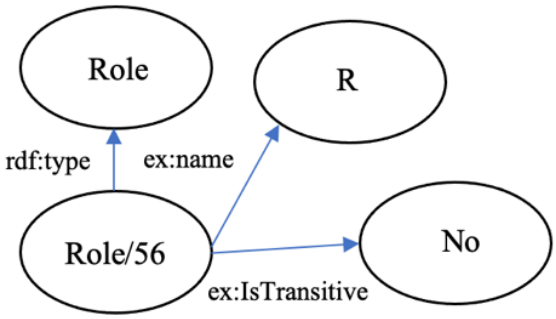
R2R ML Triple Map

<#TriplesMap2>
rr:logicalTable [rr:tableName «Role»];
r:subjectMap [
rr:template «http://example.com/Ch#/Role/{RPK}»;
rr:class ex:Role;
]
rr:predicateObjectMap [
rr:predicate ex:name;
rr:objectMap [rr:column «Name»];
]
rr:predicateObjectMap [
rr:predicate ex:IsTransitive;
rr:objectMap [rr:column «IsTransitive»];
]

RDF output example

<http://example.com/Ch#/Role/56> rdf:type ex:Role.
<http://example.com/Ch#/Role/56> ex:name "R".
<http://example.com/Ch#/Role/56> ex:IsTransitive "No".

RDF output example graph



1.3. CIndividual

SQL table CIndividual

CIPK	Name
100	abc
101	def
102	aaa
103	bbb
104	ccc
105	ddd

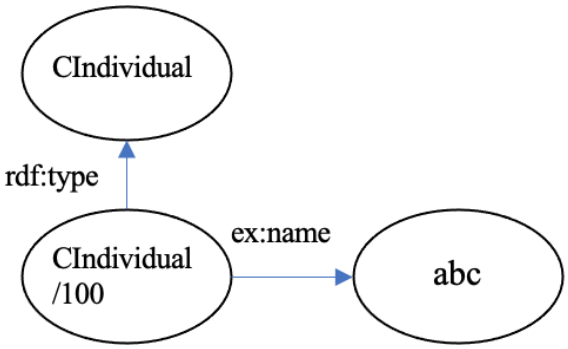
R2R ML Triple Map

<#TriplesMap3>
rr:logicalTable [rr:tableName «CIndividual»];
r:subjectMap [
rr:template «http://example.com/Ch#/CIndividual/{CIPK}»;
rr:class ex:CIndividual;
]
rr:predicateObjectMap [
rr:predicate ex:name;
rr:objectMap [rr:column «Name»];
]

RDF output example

<http://example.com/Ch#/CIndividual/100> rdf:type ex:CIndividual.
<http://example.com/Ch#/CIndividual/100> ex:name "abc".

RDF output example graph



1.4. RIndividual

SQL table RIndividual

RIPK
10
11
12
13
14
15

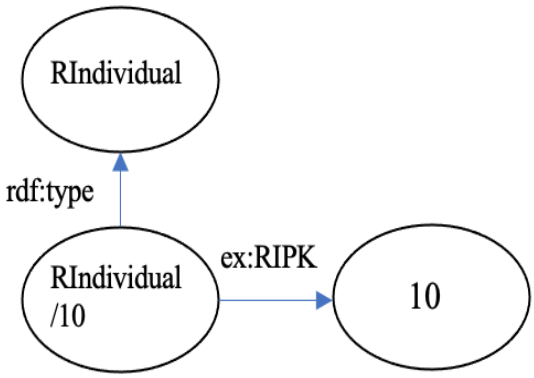
R2R ML Triple Map

```
<#TriplesMap4>
rr:logicalTable [ rr:tableName «RIndividual»];
r:subjectMap [
        rr:template
«http://example.com/Ch#/RIndividual/{RIPK}»;
        rr:class ex:RIndividual;
]
rr:predicateObjectMap [
        rr:predicate      ex:RIPK;
        rr:objectMap      [rr:column «RIPK»];
]
```

RDF output example

<http://example.com/Ch#/RIndividual/10> rdf:type ex:RIndividual.
<http://example.com/Ch#/RIndividual/10> ex:RIPK 10.

RDF output example graph



2. Relationship-bundles

2.1. Domain

SQL table Domain

CFK	RFK
1	56
2	67

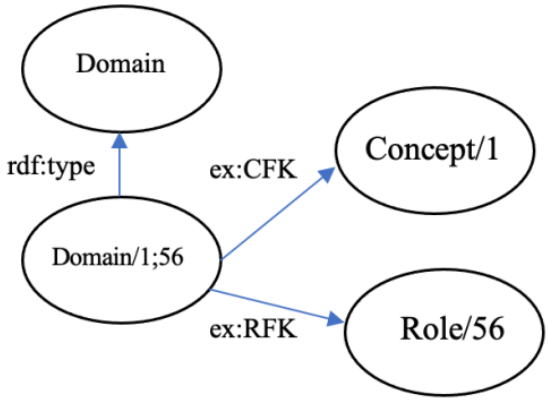
R2R ML Triple Map

```
<#TriplesMap5>
rr:logicalTable [ rr:tableName «Domain»];
r:subjectMap [
        rr:template
«http://example.com/Ch#//Domain/{CFK};{RFK
}»;
        rr:class ex:Domain;
]
rr:predicateObjectMap [
        rr:predicate      ex:CFK
        rr:objectMap      [
        a rr:RefObjectMap ;
rr:parentTriplesMap <#TriplesMap1>
        rr:joinCondition [
                rr:child «CFK»;
                rr:parent «CPK»;
]]]
rr:predicateObjectMap [
        rr:predicate      ex:RFK
        rr:objectMap      [
        a rr:RefObjectMap ;
rr:parentTriplesMap <#TriplesMap2>
        rr:joinCondition [
                rr:child «RFK»;
                rr:parent «RPK»;
]]]
```

RDF output example

<http://example.com/Ch#/Domain/1;56>rdf:type ex:Domain.
<http://example.com/Ch#/Domain/1;56>ex:CFK <http://example.com/Ch#/Concept/1>.
<http://example.com/Ch#/Domain/1;56>ex:RFK <http://example.com/Ch#/Role/56>.

RDF output example graph



2.2. Range

SQL table Range

CFK	RFK
2	56
1	67

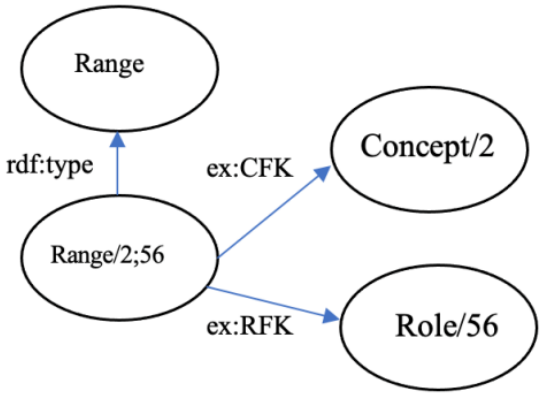
R2R ML Triple Map

```
<#TriplesMap6>
rr:logicalTable [ rr:tableName «Range»];
r:subjectMap [
        rr:template
«http://example.com/Ch#/Range/{CFK};{RFK}»;
        rr:class ex:Range;
]
rr:predicateObjectMap [
        rr:predicate      ex:CFK
        rr:objectMap    [
        a rr:RefObjectMap ;
rr:parentTriplesMap <#TriplesMap1>
        rr:joinCondition [
                rr:child «CFK»;
                rr:parent «CPK»;
]]]
rr:predicateObjectMap [
        rr:predicate      ex:RFK
        rr:objectMap    [
        a rr:RefObjectMap ;
rr:parentTriplesMap <#TriplesMap2>
        rr:joinCondition [
                rr:child «RFK»;
                rr:parent «RPK»;
]]]
```

RDF output example

<http://example.com/Ch#/Range/2;67>rdf:type ex:Range.
<http://example.com/Ch#/Range/2;67>ex:CFK <http://example.com/Ch#/Concept/2>.
<http://example.com/Ch#/Range/2;67>ex:RFK <http://example.com/Ch#/Role/67>.

RDF output example graph



2.3. LinkCI

SQL table LinkCI

CFK	CIFK
1	100
1	101
2	102
2	103
2	104
2	101

R2R ML Triple Map

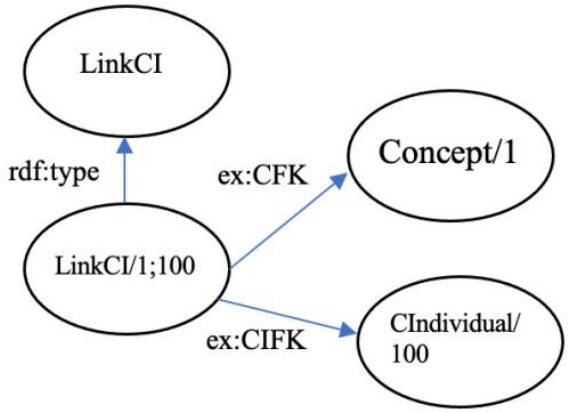
```
<#TriplesMap7>
rr:logicalTable [ rr:tableName «LinkCI»];
r:subjectMap [
        rr:template
«http://example.com/Ch#/LinkCI/{CFK};{CIFK}
»;
        rr:class ex:LinkCI;
        ]
rr:predicateObjectMap [
        rr:predicate      ex:CFK
        rr:objectMap    [
        a rr:RefObjectMap ;
rr:parentTriplesMap <#TriplesMap1>
        rr:joinCondition [
                rr:child «CFK»;
                rr:parent «CPK»;
]]]
rr:predicateObjectMap [
        rr:predicate      ex:CIFK
        rr:objectMap    [
```

a rr:RefObjectMap ;
rr:parentTriplesMap <#TriplesMap3>
rr:joinCondition [
rr:child «CIFK»;
rr:parent «CIPK»;
]]]

RDF output example

<http://example.com/Ch#/LinkCI/1;100>rdf:type ex:LinkCI.
<http://example.com/Ch#/LinkCI/1;100>ex:CFK <http://example.com/Ch#/Concept/1>.
<http://example.com/Ch#/LinkCI/1;100>ex:CIFK <http://example.com/Ch#/CIndividual/100>.

RDF output example graph



2.4. LinkRRI

SQL table LinkRRI

RFK	RIFK
56	10
56	11
67	12
67	13
67	14
56	15

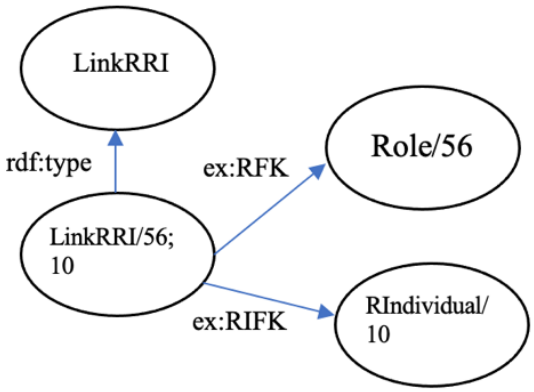
R2R ML Triple Map

<#TriplesMap8>
rr:logicalTable [rr:tableName «LinkRRI»];
r:subjectMap [
rr:template
«http://example.com/Ch#/LinkRRI/{RFK};{RIFK}»;
rr:class ex:LinkRRI;
]
rr:predicateObjectMap [
rr:predicate ex:RFK
rr:objectMap [
a rr:RefObjectMap ;
rr:parentTriplesMap <#TriplesMap2>
rr:joinCondition [
rr:child «RFK»;
rr:parent «RPK»;
]]]
rr:predicateObjectMap [
rr:predicate ex:RIFK
rr:objectMap [
a rr:RefObjectMap ;
rr:parentTriplesMap <#TriplesMap4>
rr:joinCondition [
rr:child «RIFK»;
rr:parent «RIPK»;
]]]

RDF output example

<http://example.com/Ch#/LinkRRI/56;10> rdf:type ex:LinkRRI.
<http://example.com/Ch#/LinkRRI/56;10> ex:RFK <http://example.com/Ch#/Role/56>.
<http://example.com/Ch#/LinkRRI/56;10> ex:RIFK <http://example.com/Ch#/RIndividual/10>.

RDF output example graph



2.5. Predecessor

SQL table Predecessor

CIFK	RIFK
100	10
101	11
100	15
102	12
103	13
104	14

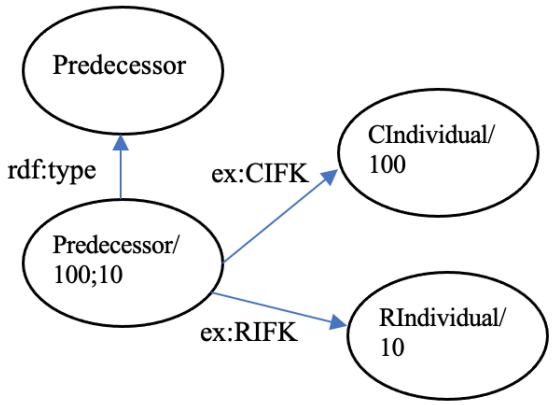
R2R ML Triple Map

```
<#TriplesMap9>
rr:logicalTable [ rr:tableName «Predecessor»];
r:subjectMap [
        rr:template
«http://example.com/Ch#/Predecessor/{CIFK};{R
IFK}»;
        rr:class ex:Predecessor;
]
rr:predicateObjectMap [
        rr:predicate      ex:CIFK
        rr:objectMap    [
        a rr:RefObjectMap ;
rr:parentTriplesMap <#TriplesMap3>
        rr:joinCondition [
                rr:child «CIFK»;
                rr:parent «CIPK»;
]]]
rr:predicateObjectMap [
        rr:predicate      ex:RIFK
        rr:objectMap    [
        a rr:RefObjectMap ;
rr:parentTriplesMap <#TriplesMap4>
        rr:joinCondition [
                rr:child «RIFK»;
                rr:parent «RIPK»;
]]]
```

RDF output example

<http://example.com/Ch#/Predecessor/100;10> rdf:type ex:Predecessor.
<http://example.com/Ch#/Predecessor/100;10>ex :CIFK <http://example.com/Ch#/CIndividual/100>.
<http://example.com/Ch#/Predecessor/100;10>ex :RIFK <http://example.com/Ch#/RIndividual/10>.

RDF output example graph



2.6. Successor

SQL table Successor

CIFK	RIFK
102	10
103	11
104	15
100	12
101	13
101	14

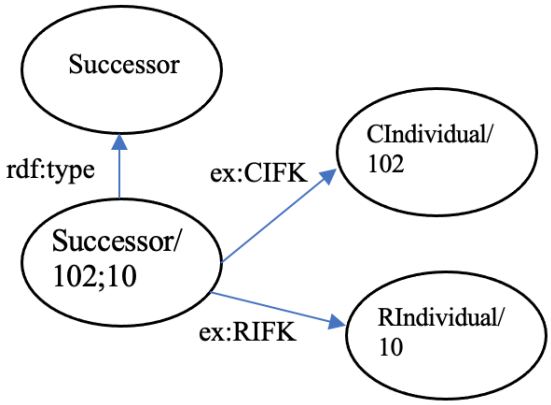
R2R ML Triple Map

```
<#TriplesMap10>
rr:logicalTable [ rr:tableName «Successor»];
r:subjectMap [
        rr:template
«http://example.com/Ch#/Successor/{CIFK};{RI
FK}»;
        rr:class ex:Successor;
]
rr:predicateObjectMap [
        rr:predicate      ex:CIFK
        rr:objectMap    [
        a rr:RefObjectMap ;
rr:parentTriplesMap <#TriplesMap3>
        rr:joinCondition [
                rr:child «CIFK»;
                rr:parent «CIPK»;
]]]
rr:predicateObjectMap [
        rr:predicate      ex:RIFK
        rr:objectMap    [
        a rr:RefObjectMap ;
rr:parentTriplesMap <#TriplesMap4>
        rr:joinCondition [
                rr:child «RIFK»;
                rr:parent «RIPK»;
]]]
```

RDF output example

<http://example.com/Ch#/Successor/102;10> rdf:type ex:Successor.
<http://example.com/Ch#/Successor/102;10> ex:CIFK <http://example.com/Ch#/CIndividual/102>.
<http://example.com/Ch#/Successor /102;10> ex:RIFK <http://example.com/Ch#/RIndividual/10>.

RDF output example graph



3. Axioms

3.1. ConceptEquivalence

SQL table ConceptEquivalence

CForFK	CIsFK
3	4

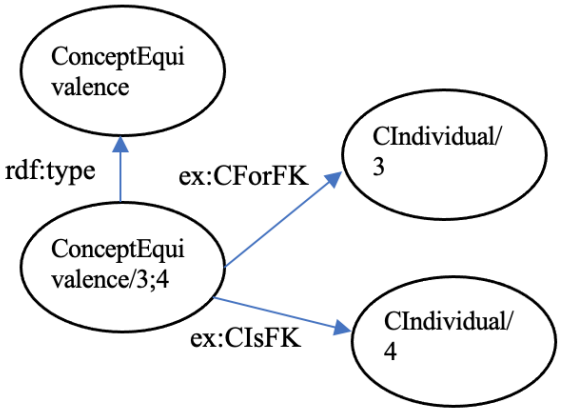
R2R ML Triple Map

```
<#TriplesMap11>
rr:logicalTable [ rr:tableName «ConceptEquivalence»];
r:subjectMap [
        rr:template «http://data.example.com/ConceptEquivalence/{CForFK};{CIsFK}»;
        rr:class ex:ConceptEquivalence;
]
rr:predicateObjectMap [
        rr:predicate    ex:CForFK
        rr:objectMap    [
        a rr:RefObjectMap ;
rr:parentTriplesMap <#TriplesMap1>
        rr:joinCondition [
                rr:child «CForFK»;
                rr:parent «CPK»;
]]]
rr:predicateObjectMap [
        rr:predicate    ex:CIsFK
        rr:objectMap    [
        a rr:RefObjectMap ;
rr:parentTriplesMap <#TriplesMap1>
        rr:joinCondition [
                rr:child «CIsFK»;
                rr:parent «CPK»;
]]]
```

RDF output example

<http://example.com/Ch#/ConceptEquivalence/3;4> rdf:type ex:ConceptEquivalence.
<http://example.com/Ch#/ConceptEquivalence/3;4>ex:CForFK <http://example.com/Ch#/Concept/3>.
<http://example.com/Ch#/ConceptEquivalence/3;4>ex:CIsFK <http://example.com/Ch#/Concept/4>

RDF output example graph



3.2. ConceptNesting

SQL table ConceptNesting

CInFK	COutFK
3	1

R2R ML Triple Map

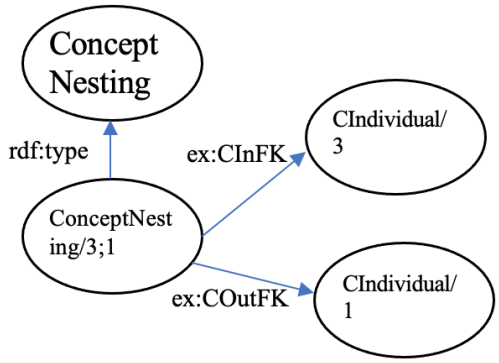
```
<#TriplesMap12>
rr:logicalTable [ rr:tableName «ConceptNesting»];
r:subjectMap [
        rr:template «http://data.example.com/ConceptNesting/{CInFK};{COutFK}»;
        rr:class ex:ConceptNesting;
]
rr:predicateObjectMap [
        rr:predicate    ex:CInFK
        rr:objectMap    [
        a rr:RefObjectMap ;
rr:parentTriplesMap <#TriplesMap1>
        rr:joinCondition [
                rr:child «CInFK»;
                rr:parent «CPK»;
]]]
rr:predicateObjectMap [
        rr:predicate    ex:COutFK
        rr:objectMap    [
        a rr:RefObjectMap ;
```

rr:parentTriplesMap <#TriplesMap1>
rr:joinCondition [
rr:child «COutFK»;
rr:parent «CPK»;
]]]

RDF output example

<http://example.com/Ch#/ConceptNesting/3;1> rdf:type ex:ConceptNesting.
<http://example.com/Ch#/ConceptNesting/3;1>ex :CInFK <http://example.com/Ch#/Concept/1>.
<http://example.com/Ch#/ConceptNesting/3;1>ex :COutFK <http://example.com/Ch#/Concept/1>

RDF output example graph



3.3. RoleNesting

SQL table RoleNesting

RInFK	ROutFK
89	56

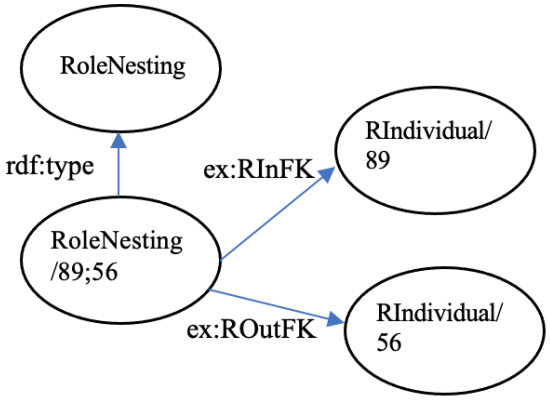
R2R ML Triple Map

<#TriplesMap13>
rr:logicalTable [rr:tableName «RoleNesting»];
r:subjectMap [
rr:template «http://example.com/Ch#/RoleNesting/{RInFK};{ROutFK}»;
rr:class ex:RoleNesting;
]
rr:predicateObjectMap [
rr:predicate ex:RInFK
rr:objectMap [
a rr:RefObjectMap ;
rr:parentTriplesMap <#TriplesMap2>
rr:joinCondition [
rr:child «RInFK»;
rr:parent «RPK»;
]]]
rr:predicateObjectMap [
rr:predicate ex:ROutFK
rr:objectMap [
a rr:RefObjectMap ;
rr:parentTriplesMap <#TriplesMap2>
rr:joinCondition [
rr:child «ROutFK»;
rr:parent «RPK»;
]]]

RDF output example

<http://example.com/Ch#/RoleNesting/89;56> rdf:type ex:RoleNesting.
<http://example.com/Ch#/RoleNesting/89;56> ex:RInFK <http://example.com/Ch#/RoleNesting/89>
<http://example.com/Ch#/RoleNesting/89;56> ex:ROutFK <http://example.com/Ch#/RoleNesting/56>

RDF output example graph



3.4. RoleEquivalence

SQL table RoleEquivalence

RForFK	RIsFK
89	34

R2R ML Triple Map

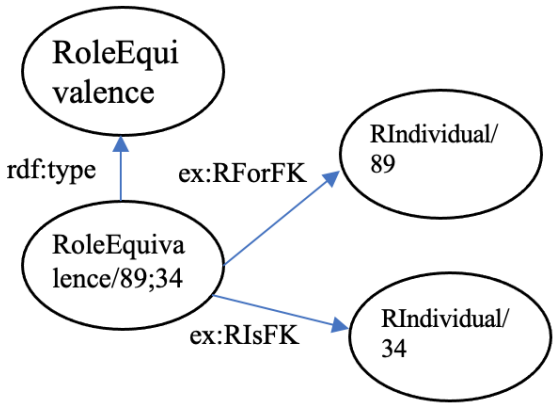
<#TriplesMap14>
rr:logicalTable [rr:tableName «RoleEquivalence»];
r:subjectMap [
rr:template «http://example.com/Ch#/RoleEquivalence/{RForFK};{RIsFK}»;
rr:class ex:RoleNesting;
]

```
rr:predicateObjectMap [
        rr:predicate      ex:RForFK
        rr:objectMap    [
        a rr:RefObjectMap ;
rr:parentTriplesMap <#TriplesMap2>
        rr:joinCondition [
                rr:child «RForFK»;
                rr:parent «RPK»;
]]]
rr:predicateObjectMap [
        rr:predicate      ex:RIsFK
        rr:objectMap    [
        a rr:RefObjectMap ;
rr:parentTriplesMap <#TriplesMap2>
        rr:joinCondition [
                rr:child «RIsFK»;
                rr:parent «RPK»;
]]]
```

RDF output example

<http://example.com/Ch#/RoleEquivalence/89;34> rdf:type ex:RoleEquivalence.
<http://example.com/Ch#/RoleEquivalence/89;34>ex:RForFK
<http://example.com/Ch#/RoleEquivalence/89>
<http://example.com/Ch#/RoleEquivalence/89;34>ex:RIsFK
<http://example.com/Ch#/RoleEquivalence/56>

RDF output example graph



3.5. Transitive role

SQL table Role

RPK	Name	IsTransitive
56	R	No
67	S	No
89	T	No
34	Z	No
23	U	Yes

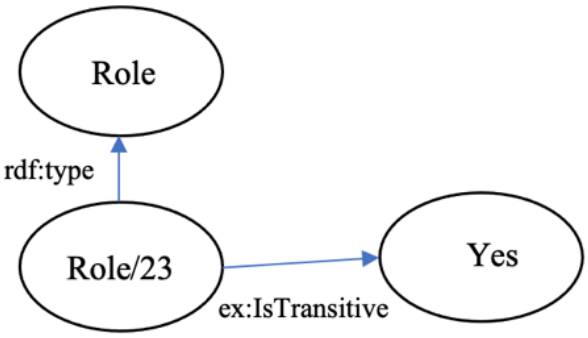
R2R ML Triple Map

```
<#TriplesMap15>
rr:logicalTable [ rr:tableName «Role»];
r:subjectMap [
        rr:template
«http://example.com/Ch#/Role/{RPK}»;
        rr:class ex:Role;
]
rr:predicateObjectMap [
        rr:predicate ex:IsTransitive;
        rr:objectMap [rr:column «IsTransitive»];
]
```

RDF output example

<http://example.com/Ch#/Role/23> rdf:type ex:Role.
<http://example.com/Ch#/Role/23> ex:IsTransitive "Yes".

RDF output example graph



3.6. CIEquivalence

SQL table CIEquivalence

CIForFK	CIIsFK
105	104

R2R ML Triple Map

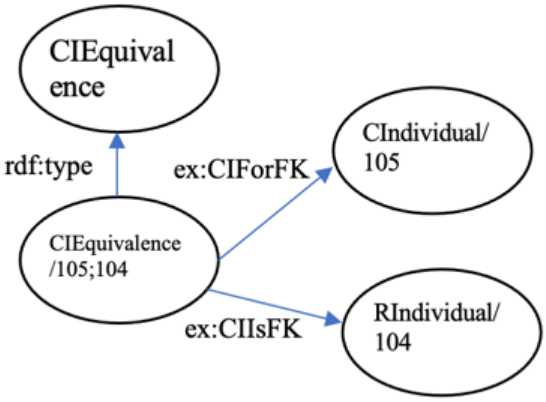
```
<#TriplesMap16>
rr:logicalTable [ rr:tableName «CIEquivalence»];
r:subjectMap [
        rr:template
«http://example.com/Ch#/CIEquivalence/{CIForFK};{CIIsFK}»;
        rr:class ex:RoleNesting;
]
rr:predicateObjectMap [
        rr:predicate      ex:CIForFK
        rr:objectMap    [
        a rr:RefObjectMap ;
```

rr:parentTriplesMap <#TriplesMap3>
rr:joinCondition [
rr:child «CIForFK»;
rr:parent «CIPK»;
]]]
rr:predicateObjectMap [
rr:predicate ex:CIIsFK
rr:objectMap [
a rr:RefObjectMap ;
rr:parentTriplesMap <#TriplesMap3>
rr:joinCondition [
rr:child «CIIsFK»;
rr:parent «CIPK»;
]]]

RDF output example

<http://example.com/Ch#/CIEquivalence/105;104 > rdf:type ex:CIEquivalence
<http://example.com/Ch#/CIEquivalence/105;104 >ex:CIForFK
<http://example.com/Ch#/CIndividual/105>.
<http://example.com/Ch#/CIEquivalence/105;104 >ex:CIIsFK
<http://example.com/Ch#/CIndividual/104>.

RDF output example graph



4. ALC syntax mapping

4.1. Concept

RA[2] term

$C^{E}_{RM^2} = \pi_{CIndividual.Name}(\sigma_{Concept.Name='C'}(CIndividual \bowtie_{CIPK=CIFK} (LinkCI \bowtie_{CFK=CPK} Concept)))$

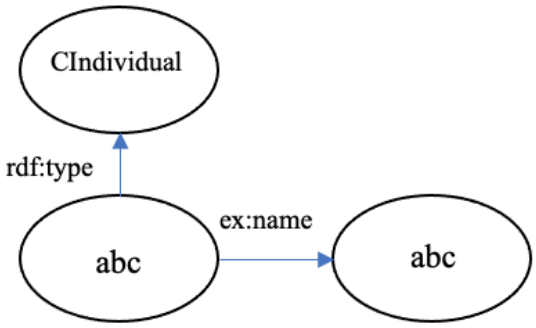
R2R ML Triple Map

<#TriplesMap17>
rr:logicalTable [rr:sqlQuery
“”” SELECT ci.Name
FROM CIndividual ci, Concept c, LinkCI lci,
WHERE ci.CIPK = lci.CIFK. AND lci.CFK = c.CPK AND c.Name = ‘C’ “”””];
r:subjectMap [
rr:template «http://example.com/Ch#/{Name}»;
rr:class ex:CIndividual;
]
rr:predicateObjectMap [
rr:predicate ex:name
rr:objectMap [rr:column: Name;]
]

RDF output example

<http://example.com/Ch#/CIndividual/abc> rdf:type ex:CIndividual.
<http://example.com/Ch#/CIndividual/abc> ex:name “abc”.

RDF output example graph



4.2. Role

RA[2] term

$R^{E}_{RM^2} = \pi_{First,Second}(\rho_{CIdividual.Name.Second}(CIndividual \bowtie_{CIPK=CFPK} (Successor \bowtie_{RIFK=RIPK} (\rho_{CIndividual.Name.First}(CIndividual \bowtie_{CIPK=CIFK} (Predecessor \bowtie_{RIFK=RIPK} (\sigma_{Role.Name='R'}(RIndividual \bowtie_{RIPK=RIFK} (LinkRRI \bowtie_{RFK=RPK} Role)))))))))$

R2R ML Triple Map

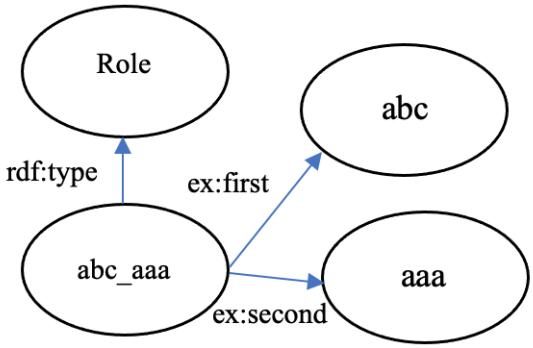
<#TriplesMap18>
rr:logicalTable [rr:sqlQuery
“”” SELECT first.Name AS First, second,Name AS Second
FROM Role r, RIndividual ri, LinkRRI lrri , Predecessor p, Successor s
CIndividual first, CIndividual.second

WHERE r.RPK = lrri.RFK AND lrri.RIFK = ri.RIPK AND ri.RIPK = p.RIFK AND
p.CIFK = first.CIPK AND ri.RIPK = s.RIFK AND s.CIFK = second. CIPK AND r.Name='R" """];
r:subjectMap [
rr:template
«http://example.com/Ch#/{First}_{Second}»;
rr:class ex:RIndividual;
]
rr:predicateObjectMap [
rr:predicate ex:first
rr:objectMap [rr:column: First;]
]
rr:predicateObjectMap [
rr:predicate ex:second
rr:objectMap [rr:column: Second;]
]

RDF output example

<http://example.com/Ch#/abc_aaa>rdf:type ex:Role;
<http://example.com/Ch#/abc_aaa>ex:first "abc"
<http://example.com/Ch#/abc_aaa>ex:second "aaa"

RDF output example graph



4.3. Concept negation

RA^2 term

$$(\neg C)^{E}_{RM^2} = \pi_{Name}(CIndividual) - C^{E}_{RM^2}$$

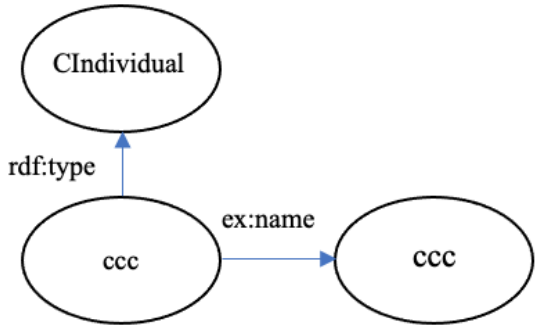
R2R ML Triple Map

<#TriplesMap19>
rr:logicalTable [rr:sqlQuery
""" SELECT ci.Name
FROM CIndividual ci
EXCEPT
SELECT ci.Name
FROM CIndividual ci, Concept c, LinkCI lci,
WHERE ci.CIPK = lci.CIFK. AND lci.CFK = c.CPK AND c.Name = 'C' """];
r:subjectMap [
rr:template
«http://example.com/Ch#/{Name}»;
rr:class ex:CIndividual;
]
rr:predicateObjectMap [
rr:predicate ex:name
rr:objectMap [rr:column: Name;]
]

RDF output example

<http://example.com/Ch#/CIndividual/ccc> rdf:type ex:CIndividual.
<http://example.com/Ch#/CIndividual/ccc> ex:name "ccc".

RDF output example graph



4.4. Concept union

RA^2 term

$$(C \sqcup D) = C^{E}_{RM^2} \cup D^{E}_{RM^2}$$

R2R ML Triple Map

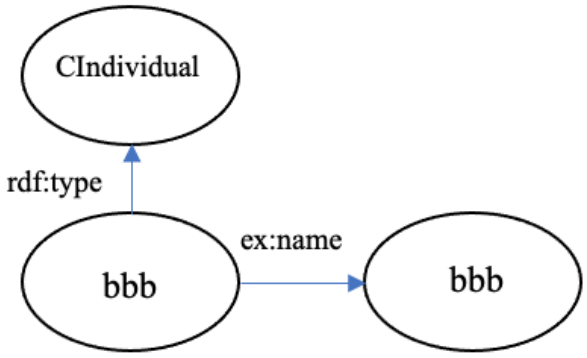
<#TriplesMap20>
rr:logicalTable [rr:sqlQuery
""" SELECT ci.Name
FROM CIndividual ci, Concept c, LinkCI lci,
WHERE ci.CIPK = lci.CIFK. AND lci.CFK = c.CPK AND c.Name = 'C'
UNION
SELECT ci.Name
FROM CIndividual ci, Concept c, LinkCI lci,
WHERE ci.CIPK = lci.CIFK. AND lci.CFK = c.CPK AND c.Name = 'D' """];
r:subjectMap [
rr:template

«http://example.com/Ch#/{Name}»;
rr:class ex:CIndividual;
]
rr:predicateObjectMap [
rr:predicate ex:name
rr:objectMap [rr:column: Name;]
]

RDF output example

<http://example.com/Ch#/CIndividual/bbb> rdf:type ex:CIndividual.
<http://example.com/Ch#/CIndividual/bbb> ex:name "bbb".

RDF output graph



4.5. Concept intersection

RA[2] term

$$(C \sqcap D) = C^{E}_{RM^2} \cap D^{E}_{RM^2}$$

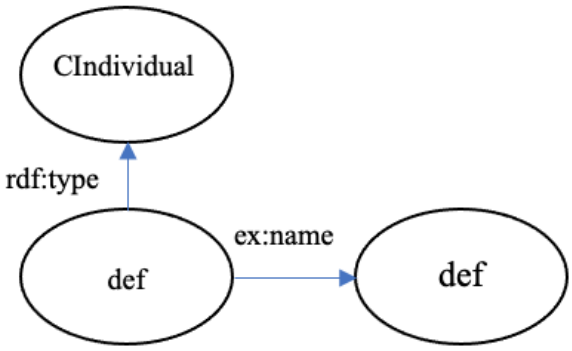
R2R ML Triple Map

<#TriplesMap21>
rr:logicalTable [rr:sqlQuery
""""
SELECT ci.Name
FROM CIndividual ci, Concept c, LinkCI lci,
WHERE ci.CIPK = lci.CIFK. AND lci.CFK = c.CPK AND c.Name = 'C'
INTERSECT
SELECT ci.Name
FROM CIndividual ci, Concept c, LinkCI lci,
WHERE ci.CIPK = lci.CIFK. AND lci.CFK = c.CPK AND c.Name = 'D'
""""];
r:subjectMap [
rr:template
«http://example.com/Ch#/{Name}»;
rr:class ex:CIndividual;
]
rr:predicateObjectMap [
rr:predicate ex:name
rr:objectMap [rr:column: Name;]
]

RDF output example

<http://example.com/Ch#/CIndividual/def> rdf:type ex:CIndividual.
<http://example.com/Ch#/CIndividual/def> ex:name "def".

RDF output example graph



The following notation is introduced:

- a table RE(First, Second) was get after mapping $R^{E}_{RM^2}$ (<#TriplesMap18>);
- a table CE(Name) was get after mapping $C^{E}_{RM^2}$(<#TriplesMap17>).

4.6. Existential quantification

RA[2] term

$$(\exists R.\, C)^{E}_{RM^2} = \pi_{First}(R^{E}_{RM^2} \bowtie_{Second=Name} C^{E}_{RM^2})$$

R2R ML Triple Map

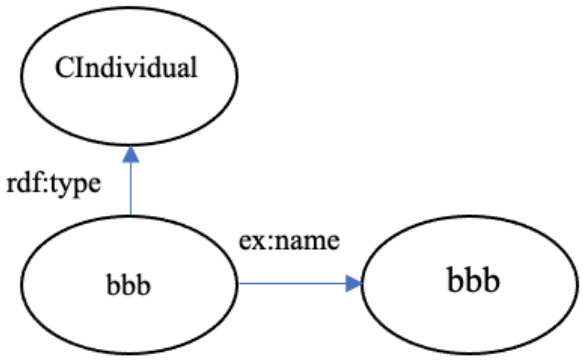
<#TriplesMap22>
rr:logicalTable [rr:sqlQuery
"""" SELECT RE.First
FROM RE, CE
WHERE RE.Second = CE.Name """"];
r:subjectMap [
rr:template
«http://example.com/Ch#/{First}»;
rr:class ex:CIndividual;
]
rr:predicateObjectMap [
rr:predicate ex:first

rr:objectMap [rr:column: First;]
]

RDF output example

<http://example.com/Ch#/CIndividual/bbb> rdf:type ex:CIndividual.
<http://example.com/Ch#/CIndividual/bbb> ex:name "bbb".

RDF output graph



4.7. Value restriction

RA^2 term

$$(\forall R.C)^{E}_{RM^2} = \pi_{First} R^{E}_{RM^2} - \pi_{First}(R^{E}_{RM^2} \cap (\pi_{First} R^{E}_{RM^2} \times (\pi_{Second} R^{E}_{RM^2} - \pi_{Name} C^{E}_{RM^2})))$$

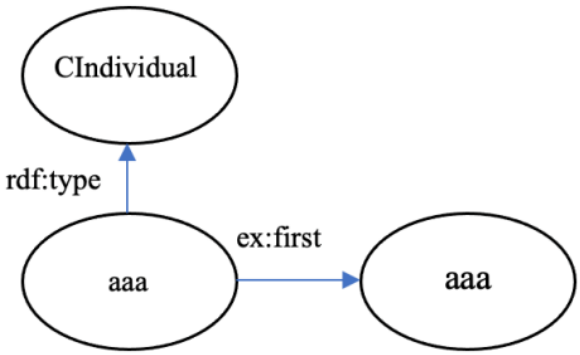
R2R ML Triple Map

```
<#TriplesMap23>
rr:logicalTable [ rr:sqlQuery
“””SELECT RE.First
FROM RE
EXCEPT
SELECT First
FROM (SELECT
        FROM RE
        INTERSECT
        (SELECT *
        FROM (SELECT RE.First
                FROM RE),
                (SELECT RE.Second
                FROM RE
                EXCEPT
                SELECT CE.Name
                FROM CE))) “””];
r:subjectMap [
rr:template «http://example.com/Ch#/{First}»;
rr:class ex:CIndividual;
]
rr:predicateObjectMap [
        rr:predicate    ex:first
        rr:objectMap    [ rr:column: First;]
]
```

RDF output example

<http://example.com/Ch#/CIndividual/aaa> rdf:type ex:CIndividual.
<http://example.com/Ch#/CIndividual/aaa> ex:first "aaa".

RDF output graph



5. ALC extensions

The mappings to only several number restrictions are shown in the paper. Mappings for the rest of extensions uses the recursive SQL.

5.1. Functional restrictions

RA^2 term

$$(\geq 2R)^{E}_{RM^2} = \pi_{First}(\sigma_{Second1 \neq Second2} (\rho_{Second/Second1}(R^{E}_{RM^2}) \bowtie_{First=First} (\rho_{Second/Second2}(R^{E}_{RM^2}))))$$

R2R ML Triple Map

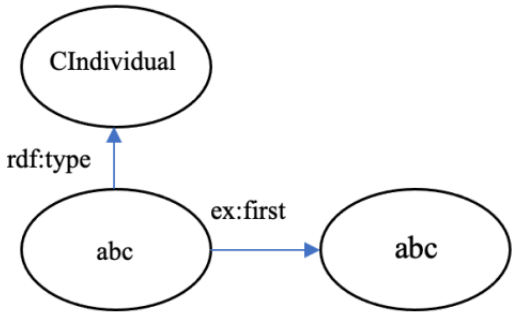
```
<#TriplesMap24>
rr:logicalTable [ rr:sqlQuery
“””SELECT one.First
FROM RE one, RE two
WHERE one.First = two.First
AND one.Second <> two.Second “””];
r:subjectMap [
rr:template «http://example.com/Ch#/{First}»;
```

rr:class ex:CIndividual;
]
rr:predicateObjectMap [
rr:predicate ex:first
rr:objectMap [rr:column: First;]
]

RDF output example

<http://example.com/Ch#/CIndividual/abc> rdf:type ex:CIndividual.
<http://example.com/Ch#/CIndividual/abc> ex:name "abc".

RDF output graph



RA^2 term

$$(\leq 1R)^{E}_{RM^2} = \pi_{Name}(CIndividual) - (\geq 2R)^{E}_{RM^2}$$

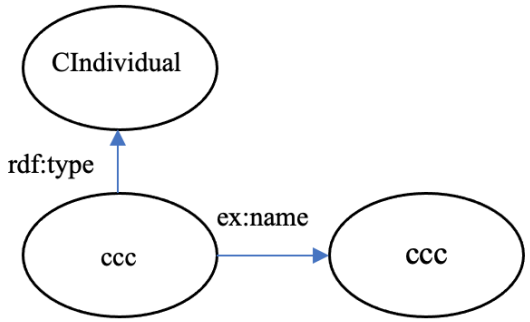
R2R ML Triple Map

<#TriplesMap25>
rr:logicalTable [rr:sqlQuery
""""SELECT ci.Name
FROM CIndividual ci
EXCEPT
SELECT one.First
FROM RE one, RE two
WHERE one.First = two.First
AND one.Second <> two.Second """"];
r:subjectMap [
rr:template «http://example.com/Ch#/{Name}»;
rr:class ex:CIndividual;
]
rr:predicateObjectMap [
rr:predicate ex:name
rr:objectMap [rr:column: Name;]
]

RDF output example

<http://example.com/Ch#/CIndividual/ccc> rdf:type ex:CIndividual.
<http://example.com/Ch#/CIndividual/ccc> ex:name "ccc".

RDF output graph



5.2. Several quality restrictions

RA^2 term

$$(\geq 2\mathrm{R}.\mathrm{C})^{\mathrm{E}}_{\mathrm{RM}^2} = \pi_{\mathrm{First}}(\sigma_{\mathrm{Second}_1 \neq \mathrm{Second}_2}(\rho_{\mathrm{Second}/\mathrm{Second}_1}(\mathrm{R}^{\mathrm{E}}_{\mathrm{RM}^2}) \bowtie_{\mathrm{Second=Name}}(\rho_{\mathrm{Second}/\mathrm{Second}_2}(\mathrm{R}^{\mathrm{E}}_{\mathrm{RM}^2} \bowtie_{\mathrm{Second=Name}} \mathrm{C}^{2}_{\mathrm{RM}^2}))))$$

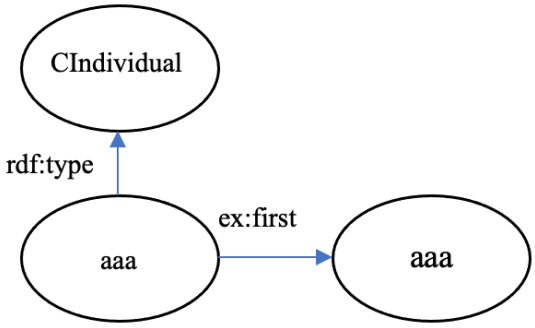
R2R ML Triple Map

<#TriplesMap26>
rr:logicalTable [rr:sqlQuery
""""SELECT one.First
FROM
(SELECT First, Second
FROM RE, CE
WHERE RE.Second = CE.Name) one,
(SELECT First, Second
FROM RE, CE
WHERE RE.Second = CE.Name) two
WHERE one,First = two.First
AND one.Second <> two.Second""""];
r:subjectMap [
rr:template «http://example.com/Ch#/{First}»;
rr:class ex:CIndividual;
]
rr:predicateObjectMap [
rr:predicate ex:first
rr:objectMap [rr:column: First;]
]

RDF output example

<http://example.com/Ch#/CIndividual/aaa> rdf:type ex:CIndividual.
<http://example.com/Ch#/CIndividual/aaa> ex:first "aaa".

RDF output graph



RA^2 term

$$(\leq 1R)^{E}_{RM^2} = \pi_{Name}(CIndividual) - (\geq 2R.\,C)^{E}_{RM^2}$$

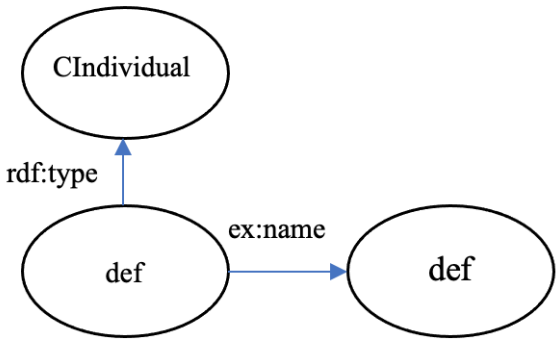
R2R ML Triple Map

```
<#TriplesMap27>
rr:logicalTable [ rr:sqlQuery
“””SELECT ci.Name
FROM CIndividual ci
EXCEPT
        SELECT one.First
        FROM
        (SELECT First, Second
        FROM RE, CE
        WHERE RE.Second = CE.Name) one,
                (SELECT First, Second
        FROM RE, CE
        WHERE RE.Second = CE.Name) two
WHERE one,First = two.First
AND one.Second <> two.Second “””];
r:subjectMap [
rr:template «http://example.com/Ch#/{Name}»;
rr:class ex:CIndividual;
]
rr:predicateObjectMap [
        rr:predicate      ex:name
        rr:objectMap    [ rr:column: Name;]
]
```

RDF output example

<http://example.com/Ch#/CIndividual/def> rdf:type ex:CIndividual.
<http://example.com/Ch#/CIndividual/def> ex:name "def".

RDF output graph



6. Role restrictions

Despite the number of role restrictions only role inverse mapping into RDF is considered in the paper. As known [19] the OWL 2 is based on the DL SROIQ. This logic includes only role inverse through all the role restrictions amount. So, the mapping for other role restrictions is out of scope of this research.

6.1. Role inverse

RA^2 term

$$(R^{-})^{E}_{RM^2} = (\rho_{R(Second,First)}(R^{E}_{RM^2}))$$

R2R ML Triple Map

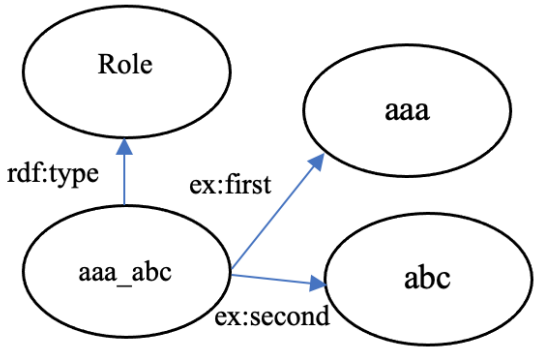
```
<#TriplesMap28>
rr:logicalTable [ rr:sqlQuery
“””SELECT second.Name AS First, first.Name AS Second
        FROM Role r, RIndividual ri, LinkRRI lrri, Predecessor p, Successor s
        CIndividual first, CIndividual.second
        WHERE r.RPK = lrri.RFK AND lrri.RIFK = ri.RIPK AND ri.RIPK = p.RIFK AND p.CIFK = first.CIPK AND ri.RIPK = s.RIFK AND s.CIFK = second. CIPK AND r.Name=’R” “””];
r:subjectMap [
```

```
rr:template «http://example.com/Ch#/{Name}»;
rr:class ex:Role;
]
rr:predicateObjectMap [
        rr:predicate      ex:name
        rr:objectMap      [ rr:column: Name;]
]
```

RDF output example

<http://example.com/Ch#/aaa_abc>rdf:type ex:Role;
<http://example.com/Ch#/aaa_abc>ex:first "aaa"
<http://example.com/Ch#/aaa_abc>ex:second "abc"

RDF output example graph



Conclusions

The article outlines a method how to check mappings between the descriptive logic and the binary relational data model using mappings into RDF. The task is set. The description of the theoretical aspects is presented. The publication provides mappings for the binary relational data model into an RDF triples using the R2R ML language. The paper also outlines the rules for converting the DL-to-RM^2 mapping formulas into RDF.

The issue of converting a number of role constructors into RDF remains open.

References

1. Chystiakova, I. Ontology-oriented data integration on the Semantic Web. *Problems in Programming.* 2014. N 2-3. P. 188–196.
2. Reznichenko, V. and Chystiakova, I. Mapping of the Description Logics ALC into the Binary Relational Data Structure. *Problems in Programming.* 2015. N 4. P. 13–30.
3. Reznichenko, V. and Chystiakova, I. Integration of the family of extended description logics with relational data model. *Problems in Programming*. (2016). N 2-3. P. 38–47.
4. Chystiakova, I. Integration of the description logics with extensions into relational data model. *Problems in Programming*. 2016. N 4. P. 58–65.
5. Chystiakova, I. Integration of the Descriptive Logic Axiomatics with the Relational Data Model. *Problems in programming*. 2017. N 1. P. 51–58.
6. Reznichenko, V. and Chystiakova, I. Binary Relational Data Model. *Problems in Programming*. 2017. N 2 (4). P. 96–105.
7. Chystiakova I.S. (2018). Mapping of the Relational Algebra into Descriptive Logic. *Problems in Programming*. 2017. N 2–3. P. 214 – 225.
8. Andon P. and Reznichenko V. and Chystiakova I. Mapping of Description Logic to the Relational Data Model. Cybernetics and Systems Analysis. 2017. N 53 (6). P. 963–978.
9. Chystiakova I.S. Implementation of mappings between the description logic and the binary relational data model on the RDF level. *Problems in programming.* 2020. N 4. P. 41 – 54.
10. Heath I.J. Unacceptable file operations in a relational data base. SIGFIDET '71. 1971. P. 19–33.
11. OWL 2 Web Ontology Language. Mapping to RDF Graphs (Second Edition). [Online] December 2012. Available from: https://www.w3.org/TR/owl2-mapping-to-rdf/#Translation_of_Axioms_without_Annotations. [Accessed: 20 February 2021].
12. R2RML: RDB to RDF Mapping Language. [Online] September 2012. Available from: https://www.w3.org/TR/r2rml/. (last access 20 February 2021).
13. Caroll J.J. Matching RDF Graphs I. Horrocks and J. Hendler (Eds.): ISWC. 2002. LNCS 2342. P. 5–15.
14. Berners-Lee, T. Relational Databases on the Semantic Web [Online] September 1998. Available from: https://www.w3.org/DesignIssues/RDB-RDF.html. (last access 20 February 2021).
15. W3C Workshop on RDF Access to Relational Databases URL:

https://www.w3.org/2007/03/RdfRDB/ (last access 20 February 2021)
16. RDB2RDF Tutorial (R2RML and Direct Mapping) at ISWC 2013 URL: https://www.slideshare.net/juansequeda/rdb2-rdf-tutorial-iswc2013 (last access 20 February 2021)
17. Codd E.F. Extending the database relational model to capture more meaning. ACM Transactions on Database Systems (TODS). 1979. Vol. 4. Issue 4. P. 397–434.
18. Barker R. Case* method: entity relationship modelling. Addison-Wesley. 1990. P. 240.
19. The description logic handbook: theory, implementation, and applications. Baader F., Calvanese D., McGuinness D., Nardi D., Patel-Schneider P. (Eds.) Cambridge University Press. 2003. 555 p.

Література

1. Чистякова І.С. Онтолого-ориентированная интеграция данных в семантическом вебе. *Проблеми програмування*. 2014. № 2-3. С. 188–196.
2. Резниченко В.А., Чистякова И.С. Отображение дескриптивной логики ALC в бинарную реляционную структуру данных. Проблеми програмування. 2015. № 4. С. 13–30.
3. Резниченко В.А., Чистякова И.С. Интеграция семейства расширенных дескриптивных логик с реляционной моделью данных. *Проблеми програмування*. 2016. № 2-3. С. 38–47.
4. Чистякова И.С. Интеграция логик с операциями над ролями с реляционной моделью данных. *Проблеми програмування*. 2016. № 4. С. 58–65.
5. Чистякова И.С. Интеграция аксиоматики дескриптивных логик с реляционной моделью данных. *Проблеми програмування*. 2017. № 1. С. 51 – 58.
6. Резниченко В.А., Чистякова И.С. Бинарная реляционная модель данных. *Проблеми програмування.* 2017. № 2. С. 96 – 105.
7. Чистякова И.С. Отображение реляционной алгебры в дескриптивную логику. *Проблеми програмування.* 2018. № 2–3. С. 214 – 225.
8. Andon P. and Reznichenko V. and Chystiakova I. Mapping of Description Logic to the Relational Data Model. Cybernetics and Systems Analysis. 2017. N 53 (6). P. 963–978.
9. Chystiakova I.S. Implementation of mappings between the description logic and the binary relational data model on the RDF level. *Problems in programming.* 2020. N 4. P. 41 – 54.
10. Heath I.J. Unacceptable file operations in a relational data base. SIGFIDET '71. 1971. P. 19–33.
11. OWL 2 Web Ontology Language. Mapping to RDF Graphs (Second Edition). [Online] December 2012. Available from: https://www.w3.org/TR/owl2-mapping-to-rdf/#Translation_of_Axioms_without_Annotations. [Accessed: 20 February 2021].
12. R2RML: RDB to RDF Mapping Language. [Online] September 2012. Available from: https://www.w3.org/TR/r2rml/. (last access 20 February 2021).
13. Caroll J.J. Matching RDF Graphs I. Horrocks and J. Hendler (Eds.): ISWC. 2002. LNCS 2342. P. 5–15.
14. Berners-Lee, T. Relational Databases on the Semantic Web [Online] September 1998. Available from: https://www.w3.org/DesignIssues/RDB-RDF.html. (last access 20 February 2021).
15. W3C Workshop on RDF Access to Relational Databases URL: https://www.w3.org/2007/03/RdfRDB/ (last access 20 February 2021)
16. RDB2RDF Tutorial (R2RML and Direct Mapping) at ISWC 2013 URL: https://www.slideshare.net/juansequeda/rdb2-rdf-tutorial-iswc2013 (last access 20 February 2021)
17. Codd E.F. Extending the database relational model to capture more meaning. ACM Transactions on Database Systems (TODS). 1979. Vol. 4. Issue 4. P. 397–434.
18. Barker R. Case* method: entity relationship modelling. Addison-Wesley. 1990. P. 240.
19. The description logic handbook: theory, implementation, and applications. Baader F., Calvanese D., McGuinness D., Nardi D., Patel-Schneider P. (Eds.) Cambridge University Press. 2003. 555 p.

Received 04.02.2021

About the author:

Inna Chystiakova,
junior researcher at the Institute of software systems of NASU.
The number of publications in Ukrainian journals is 11.
The number of publications in foreign journals is 1.
Hirsh index is 5.
https://orcid.org/0000-0001-7946-3611.

Affiliation:

Institute of software systems of NASU
03187, Kyiv,
pr. Glushkova, 40, build 5.
Tel.: +38(066)8477784.
E-mail: inna_islyamova@ukr.net.

УДК 681.3

Моделювання RESTful API для системи автоматизації приватних електронних закупівель / А.Ю. Дорошенко, Б.В. Бодак. – С. 3 – 15.

Розроблено програмний засіб для автоматизації електронних закупівель на основі .NET Core RESTful API з використанням специфікацій OpenAPI v3.0. На сервері використовується стандарт RESTful API забезпечує сумісність системи з більшістю сучасних клієнтів та дозволяє працювати з сутностями через обмін моделей даних у форматі JSON. Реалізовано авторизацію користувачів системи, постачальників та замовників, за допомогою відкритого стандарту OAuth та Microsoft Identity Server. закупівель. Виділення ролей користувачів з певним функціоналом, створення єдиного сервісу для авторизації та реєстрації, дозволило зробити нашу систему масштабованою. У системі бізнес-логіка зосереджена в мікросервісах, які спілкуються з клієнтом у форматі обміну даних JSON. Мікросервіси викликаються через контролери-маршрутизатори, що робить її закритими від зовнішнього світу. Такий підхід до проектування серверної частини дозволив нам розподілити відповідальності між модулями та відділити сервер від клієнта. Крім цього при запитах на модифікацію даних передаються відповідні заголовки авторизації для уникнення несанкціонованого доступу. Для скорочення часу відгуку системи здійснено кешування даних на рівні репозиторію за підтримки розподіленого кешу. Створено підсистему для обробки та переходу між станами закупівель на основі скінченного автомату станів. В ході розробки було прийняте рішення викликати валідацію у методах контролерів на сервері, таким чином забезпечується коректність даних з початку їх обробки.

UDC 681.3

Designing RESTful API for the e-procurement system in private sector / A.Yu. Doroshenko, B.V. Bodak. – P. 3 – 15.

The software for the e-procurement system was developed based on .NET Core RESTful API with Open API specifications. The server side uses RESTful API which ensures compatibility with the majority of clients and enables them to exchange information in JSON format. The authentication and authorization flow was implemented using OAuth open standard paired with Microsoft Identity Service. User roles and functionality were handled with a standalone service for authentication and registration that made our system efficient and scalable. Business logic was designed to be split into micro-services accessible through routing controllers. This approach allowed us to separate the responsibilities between the server and the client side. Special authorization headers passed during modification queries allowed us to control and restrict access to particular resources for unauthorized users. The distributed cache mechanism inside the data repository level was used in order to increase the responsiveness of the system. The state handling subsystem was designed utilizing Finite State Machine concepts. The developed system was verified using unit and integration tests.

Key words: e-procurement, RESTful API, OpenAPI v3.0, API architecture, scalable software systems.

Проведено випробування розробленого програмного засобу з використанням модульних та інтеграційних тестів.

Ключові слова: електронні торги, RESTful API, OpenAPI v3.0, архітектура API, масштабовані системи автоматизації.

УДК 004.4'22

Застосування засобів нейроеволюції в технічних системах автоматизації керування / А.Ю. Дорошенко, І.З. Ашур. – С. 16 – 25.

Навчання з підкріпленням – галузь машинного навчання, що базується на тому, як програмним агентам слід виконувати дії у середовищі з метою максимізації поняття кумулятивної винагороди. В даній роботі запропоновано нове застосування техніки машинного навчання з підкріпленням у формі нейроеволюції наростаючих топологій для розв'язування задач автоматизації керування на одному з прикладів моделювання задач керування технічними системами. Використовується набір інструментів для розробки та порівняння алгоритмів навчання з підкріпленням OpenAI Gym, повноцінна реалізація з відкритим програмним кодом генетичного алгоритму нейроеволюції NEAT під назвою SharpNEAT, та проміжне програмне забезпечення для оркестрації зазначених компонентів. Алгоритм нейроеволюції наростаючих топологій демонструє знаходження ефективних нейронних мереж на прикладі вирішення простих стандартних галузевих задач з системами з неперервним керуванням з набору OpenAI Gym.

Ключові слова: штучні нейронні мережі, навчання з підкріпленням, генетичні алгоритми, автоматизація керування в технічних системах.

UDC 004.4'22

Application of neuroevolution tools in automation of technical control systems / A.Yu. Doroshenko, I.Z. Achour. – P. 16 – 25.

Reinforced learning is a field of machine learning based on how software agents should perform actions in the environment to maximize the concept of cumulative reward. This paper proposes a new application of machine reinforcement learning techniques in the form of neuroevolution of augmenting topologies to solve control automation problems using modeling control problems of technical systems. Key application components include OpenAI Gym toolkit for developing and comparing reinforcement learning algorithms, full-fledged open-source implementation of the NEAT genetic algorithm called SharpNEAT, and intermediate software for orchestration of these components. The algorithm of neuroevolution of augmenting topologies demonstrates the finding of efficient neural networks on the example of a simple standard problem with continuous control from OpenAI Gym.

Key words: artificial neural networks, reinforced learning, genetic algorithms, control automation in technical systems.

УДК 004.4'22

Засіб вимірювання метрик вихідного коду Fortran за допомогою синтаксичного аналізу / А.М. Покровський, А.Ю. Дорошенко. – С. 26 – 35.

В умовах стрімкого розвитку методик забезпечення якості програмного коду, все більшою стає потреба в інструментах, що можуть автоматизувати процес оновлення та реструктуризації текстів програм. Розроблено програмний засіб для вимірювання програмних метрик, що дозволяє провести оцінювання якості вихідного коду програм мовою Fortran. Для цього розроблено алгоритми обходу синтаксичного дерева програми та на їх основі реалізовано модуль для інтегрованого середовища програмування Photran. Модуль використовує синтаксичний аналізатор програмного коду та побудоване Photran на його основі структурне дерево. Проведено порівняння розробленого засобу з наявними інструментами аналізу вихідного коду. Результати показують, що розроблений засіб особливо ефективний у поєднанні з вбудованою в Photran системою рефакторінгу, а програмний інтерфейс самого Photran дозволяє легко масштабувати існуючу інфраструктуру, додаючи підтримку інших видів аналізу.

Ключові слова: метрика вихідного коду, якість програмного забезпечення, рефакторінг.

UDC 004.4'22

A tool to measure Fortran source code metrics using syntax analysis / A.M. Pokrovskyi, A.Yu. Doroshenko. – P. 26 – 35.

The rapid development of software quality measurement methods, the need in efficient and versatile reengineering automatization tools becomes increasingly bigger. This becomes even more apparent when the programming language and respective coding practices slowly develop alongside each other for a long period of time, while the legacy code base grows bigger and remains highly relevant. In this paper, a source code metrics measurement tool for Fortran program quality evaluation is developed. It is implemented as a code module for Photran integrated development environment and based on a set of syntax tree walking algorithms. The module utilizes the built-in Photran syntax analysis engine and the tree data structure which it builds from the source code. The developed tool is also compared to existing source code analysis instruments. The results show that the developed tool is most effective when used in combination with Photran's built-in refactoring system, and that Photran's application programming interface facilitates easy scaling of the existing infrastructure by introducing other code analysis methods.

Key words: source code metric, software quality, refactoring.

УДК 004.62

Аналітичний огляд підходів до інтеграції програмних систем / Ю.А. Дивак. – С. 36 – 48.

Виконаний порівняльний аналіз існуючих підходів до інтеграції програмних систем, з метою подаль-

UDC 004.62

Analytical review of approaches to integration of software systems / Y.A. Dyvak. – P. 36 – 48.

Conducted comparison analysis of existing approaches to integrations of software systems, with the purpose of

шого розроблення нових альтернативних підходів для вирішення задач інтеграції і композиції веб сервісів. Проведено інтеграцію платіжної системи в Інтернет-магазин одним із зазначених методів для більш глибокого розуміння проблем, які виникають при інтеграції програмних систем. Розглянуті виклики які стоять перед розробниками під час інтеграцій систем, та методи вирішення поставлених задач. Визначено ключові моменти на які слід звернути увагу про інтеграції програмних систем і взаємодії між ними. Об'єкт вивчення потребує нового погляду на проблему і нових альтернативних підходів які принесуть більшу гнучкість, можуть підвищити продуктивність і знайти своє застосування в сфері інтеграції програмних систем і композиції веб-сервісів.

Ключові слова: інтеграція програмних систем, композиція веб-сервісів.

continuous design of new alternative approaches for resolving integration tasks and web-service composition tasks. Conducted integration of payment system with online store with one of the mentioned approaches for a deeper understanding of problems that caused during the integrations of a software system. Were considered challenges that actual for integration developers during the integrations and methods for resolving raised issues. Defined the key moments of integrations of software systems and interactions between them that we should pay attention to. The subject of research needs a new point s of view to the issue and new alternative approaches that might bring more flexibility, increase performance, and they will find the right place for applying in the field of integrations of software systems and the composition of web-services.

Key words: Integrations of software systems, web-service composition.

УДК 004.434:004.75

UDC 004.434:004.75

Порівняння ефективності підходів Map-Reduce і акторної моделі при розв'язанні завдань з високою зв'язністю вхідних даних на прикладі задачі оптимізації рою часток / В.О Ларін, О.І. Провотар. – С. 49 – 55.

Comparison of the effectiveness of the Map-Reduce approach and the Actor model in solving problems with high boundness of input data based on Particle Swarm Optimization problem / V.O Larin, O.I. Provotar. – P. 49 – 55.

Показані приклади класу розподілених паралельних задач зі зв'язаними компонентами вхідних даних в рамках моделі Map-Reduce. Виконано порівняння ефективності подібної задачі на прикладі задачі рою часток в рамках моделі Map-Reduce (на основі фреймворку Spark) і акторної моделі з підтримкою спільної пам'яті (на основі Strumok DSL). Оцінено перспективи використання гібридної акторної моделі для інших подібних задач.

Ключові слова: акторна модель, Map-Reduce, оптимізація методом рою частинок, паралельні системи із загальною пам'яттю.

The paper defines the notion of distributed problems with bounded input components. Particle Swarm Optimization problem is shown to be an example of such a class. Such a problem's implementation based on the Map-Reduce model (implemented on the Spark framework) and an implementation based on an actor model with shared memory support (implemented on Strumok DSL) is provided. Both versions' performance assessment is conducted. The hybrid actor model is shown to be an order of magnitude more effective in time and memory efficiency than Map-Reduce implementation. Additional optimization for the hybrid actor model solution is proposed. The prospects of

using the hybrid actor model for other similar problems are given.

Key words: actor model, map-reduce, particle swarm optimization, parallel shared memory systems.

УДК 004.62

Відображення дескриптивної логіки у RDF з використанням бінарної реляційної моделі даних / І.С. Чистякова. – С. 56 – 83.

Робота присвячена комплексній проблемі інтеграції даних в семантичному вебі. Однією з ключових задач її вирішення є встановлення взаємозв'язків між моделями даних. Основою досліджень обрано дескриптивну логіку та реляційну модель даних. Метод створення відображень між цими моделями даних було розроблено на теоретичному рівні. Для створення механізмів відображення, цей метод використовує бінарну реляційну модель даних у якості інтегруючої. З метою його практичної перевірки, у роботі продовжено попередні дослідження способу реалізації відображень між дескриптивною логікою та бінарною реляційною моделлю даних. Було сформульовано задачу перевірки теоретичного механізму відображень за допомогою RDF. Розглянуто питання перетворення бінарної реляційної моделі даних у RDF-трійки. Дано стислий огляд інструменту перетворення R2R ML. За допомогою R2R ML, створено карти трійок для перетворення концептуальної інформаційної моделі дескриптивної логіки в RDF-трійки. Описано карти трійок для перетворення основних механізмів відображень у RDF-трійки за допомогою R2R ML.

Ключові слова: бінарна реляційна модель даних, дескриптивна логіка, відображення, RDF, DL, RM^2, ALC, R2RML.

UDC 004.62

Mapping of the descriptive logic into RDF using binary relational data model / I.S. Chystiakova. – P. 56 – 83.

This paper is dedicated to the data integration problem. To establish relationships between data models is one of the key tasks in this solution. The descriptive logic and the relational data model are at the heart of a study. They have been used to create a mapping method on the theoretical level. The binary relational data model has been developed as a part of a mapping method. The previous studies are continued in this paper to prove on practice a mapping creation method between the descriptive logic and the binary relational data model. The method uses the binary relational data model as an integrating model. This paper continues the previous research of practical implementation of the mapping creation between the descriptive logic and the binary relational data model. The task to prove the theoretical mapping method on practice was formulated. A question how to map the binary relational data model into RDF-triples was considered. A brief overview of the R2R ML conversion tool was given. Triple maps were created to convert a conceptual information model of descriptive logic into RDF triplets with the help of R2R ML. Also, triples maps are described to convert basic mapping mechanisms into RDF with the help of R2R ML.

Key words: binary relational data model, description logic, mapping, RDF, DL, RM^2, ALC, R2RML.

ДО УВАГИ АВТОРІВ!

У журналі "Проблеми програмування" публікуються наукові матеріали, які раніше не публікувалися в інших виданнях.

Мова статті: українська, англійська.* Обсяг статті - від 6 до16 сторінок формату А4.

Документ зберігається у форматі doc або docx. Ім'я подається транслітерацією, як прізвище автора (авторів), наприклад, “Petrenko.doc”.

Автори можуть користуватися електронною поштою і також телефаксом для ділової переписки та передачі до редакції тексту статті та правки при коректурі. E-mail редакції: alengoro@isofts.kiev.ua. FAX: +380 (44) 526 6263, Телефон: 526 5065.

1. Оформлення файлу з текстом статті.

При підготовці файлу використовуються: стиль нормальний (звичайний) або normal; шрифт Times New Roman, розмір шрифта 12 пт.; міжрядковий інтервал – 1,0; абзацний відступ -1,25 см; вирівнювання – по ширині. У тексті не допускається вирівнювання пропусками; розстановка переносів – автоматична. Формат паперу А4, розміри полів документа – 20 мм. Текст статті після анотації має бути оформлений **у 2 колонки,** ширина яких – 7,86 см, а пробіл між ними – 1,27 см.

2. Послідовність розміщення та оформлення матеріалу статті.

УДК: індекс за універсальною десятковою класифікацією.

Автори: ініціали та прізвища авторів, курсив (світлий).

Заголовок 1 **(назва статті):** не містить абревіатур та строго відповідає змісту статті. Шрифт 15 пт, напівжирний, регістр верхній.

Анотація **(мовою статті):** 50-100 слів, не містить абревіатур, зрозумілих із змісту статті. Шрифт 10 пт, звичайний.

Ключові слова **(мовою статті):** не більше10 слів, не містить абревіатур, зрозумілих із змісту статті, подаються в називному відмінку, розділені комами. Шрифт 10 пт, звичайний.

Заголовок 2 **(назва розділу):** шрифт 14 пт, напівжирний; абзац із центральним вирівнюванням, без переносів. Заголовки нижчого рівня (пункти і т.п.) у самостійний абзац не виділяються і проходять першим реченням текстового абзацу, шрифт 12 пт, напівжирний.

Основний текст статті має такі необхідні елементи:

постановка проблеми в загальному вигляді і її зв'язок з важливими науковими або практичними завданнями;

аналіз останніх досліджень і публікацій, у яких розпочато рішення даної проблеми і на які спирається автор, виділення невирішених раніше частин загальної проблеми, яким присвячується дана стаття;

формулювання цілей статті (постановка задачі);

виклад основного матеріалу дослідження з повним обґрунтуванням отриманих наукових результатів;

висновки з даного дослідження і перспективи подальших розробок у даному напрямку;

подяка (за наявності такої).

Формули створюються в редакторі Microsoft Equation 3.0 або MathType. Формули, на які є посилання в тексті, повинні мати наскрізну нумерацію. Номер формули друкується в круглих дужках біля краю правого поля. Розмір основного шрифту редактора формул – 12 пт. Розміри символів у формулах: звичайний – 12 пт, великий індекс – 9 пт, дрібний індекс – 7 пт, великий символ – 18 пт, дрібний символ – 11пт. Не допускається масштабування формульних об´єктів.

Рисунки мають бути створені вбудованим редактором Word Picture або експортовані з прикладних програм Windows у графічних форматах (bmp, pcx, gif, jpg або tif). Рисунки розташовуються по центру. Нумерація рисунків здійснюється відповідно до порядку

згадування у тексті. Нумеровані підписи розміщуються під рисунком з позначенням «Рис. », далі вказується номер рисунка і текст підпису.

Таблиці мають бути підготовлені стандартним вбудованим в Word інструментарієм "Таблиця". Таблиці нумеруються за порядком згадування. На номер таблиці повинно бути посилання в тексті. Номер таблиці вказується в окремому рядку з вирівнюванням по правій стороні (наприклад, «Таблиця 1»). Назви таблиць розміщуються над таблицею з вирівнюванням по центру. Мінімальний розмір шрифту в таблицях – 11 пт.

Література**:** нумерований список джерел згідно ДСТУ 8302:2015 від 01.07.2016 р., шрифт 11 пт, відступ: спеціальний, навислий, 0,63 см. Джерела з заголовками на латиниці наводяться без перекладу. Інші джерела подаються мовою оригіналу. Приклади оформлення бібліографічних посилань згідно з вимогами ***Harvard Style*** наведені в багатьох публікаціях, наприклад: http://www.staffs.ac.uk/assets/harvard_referencing_examples_tcm44-39847.pdf

Дані про авторів**:** мають починатися рядком "Про авторів:", напівжирний курсив. Далі вказуються для кожного з авторів ПІБ повністю, наукове звання, посада, адреса, кількість публікацій в українських виданнях (приблизна), кількість публікацій в зарубіжних індексованих виданнях (приблизна), індекс Хірша (за наявності), обов'язково номер ORCID (сайт ORCID http://orcid.org/).

Дані про місце роботи авторів**:** починаються рядком "Місце роботи авторів:", напівжирний курсив. Далі вказуються місце роботи, адреса, телефон, факс, електронна пошта, контактний телефон.

3. Оформлення файлу з анотаціями.

Файл з анотаціями містить інформацію двома мовами (наприклад, якщо стаття написана українською мовою, то анотація та ключові слова – англійською і українською мовами) та має бути оформлений у дві колонки: УДК (шрифт – 8 пт); назва статті (шрифт – 12 пт, напівжирний); прізвища та ініціали авторів (шрифт – 12 пт); текст анотації, ключові слова (шрифт – 10 пт).

Вимоги до анотації англійською мовою: обсяг від 100 до 250 слів, інформативність, оригінальність (не є калькою української анотації), змістовність (відображає основний зміст статті і результати досліджень), структурованість (дотримується логіки опису результатів у статті).

Документ зберігається у форматі doc або docx. Ім´я подається транслітерацією, як прізвище автора (авторів), наприклад, "Petrenko_Annot.doc".

*16.07.2020 р. набули чинності положення Закону України «Про забезпечення функціонування української мови як державної». Відповідно до статті 22 «Державна мова у сфері науки» у наукових виданнях не повинно бути вміщено матеріалів іншими мовами, окрім державної, англійської та мов ЄС.

Примітка: Підписний індекс журналу "Проблеми програмування" – **90853.**