УДК 004.4`2,004.75
DOI: 10.18523/2617-3808.2023.6.41-47

Давиденко А. М., Глибовець М. М.

ВЕББАЗОВАНА СИСТЕМА ГРУПОВОГО РОЗПОДІЛЕНОГО РОЗРОБЛЕННЯ ПРОГРАМ

У статті описано архітектуру, використані технології створеної веббазованої системи розподіленого розроблення програм з акцентом на ефективне й просте одночасне групове створення тексту програми та її редагування і відлагодження. Впровадження модульної архітектури та механізму збереження вмісту сесій редагування забезпечило злагоджену функціональність компонент системи та ефективну підтримку узгодженості, управління неблокуючим паралелізмом, а також підтримку застосування сторонніх компіляторів та їх інтеграції з вебсистемою для колаборативного редагування.

Для зручності спілкування користувачів у системі реалізовано чат. Вагомим доробком є підтримка окремих сесій редагування. Також імплементовано механізм прототипування клієнтських вебзастосунків, що дає змогу розробникам вебзастосунків швидко перевірити коректність програмного коду або верстки вебсторінки і поділитися цим з іншими розробниками.

Коротко представлено сфери застосування системи і подальшого вдосконалення.

Ключові слова: операційні перетворення, розподілені системи, веброзроблення, програмування, колаборативні середовища.

Вступ

Операційні перетворення (ОП) — це технологія для підтримки низки функціональних можливостей для спільної роботи в розподілених програмних системах. ОП було винайдено для підтримки узгодженості й управління неблокуючим паралелізмом під час розподіленого редагування (РР) текстових документів.

Уперше концепцію ОП розробили К. Елліс та С. Гіббс у системі GROVE (GRoup Outtie Viewing Edit) у 1989 р. [3]. Декілька років потому було виявлено деякі проблеми коректності, в результаті незалежно запропоновано кілька підходів щодо вирішення цих питань. Для розвитку взаємодії дослідників ОП і РР у 1998 р. було створено спеціальну групу за інтересами SIGCE (Special Interest Group of Collaborative Editing). Відтоді SIGCE проводить щорічні семінари в рамках основних конференцій CSCW (Computer Supported Cooperative Work).

Кожен учасник розподіленої системи з використанням операційних перетворень має підтримувати таку функціональність [4]:

1) генерування операцій: кожна дія користувача продукує операцію і має бути розіслана всім іншим учасникам;
2) прийом операцій: учасник повинен отримувати операції, що були згенеровані іншими учасниками;
3) виконання (застосування) операцій: учасник редагування повинен застосовувати отримані операції щодо локального контексту.

Колаборативні розподілені системи, що побудовані за ідеологією ОП, здебільшого використовують архітектуру, що базується на реплікації документів, які перебувають у сумісному розподіленому доступі. Це дає змогу досягти прийнятної швидкості відклику в середовищах із високою затримкою (таких, як Інтернет). Документи зі спільним доступом копіюються у локальне сховище кожного учасника РР. Таким чином, операції редагування можуть бути виконані локально відразу, а після цього поширені віддаленим учасникам. Віддалені операції редагування, що надходять учаснику, зазвичай спершу перетворюються і потім виконуються. Перетворення дає змогу гарантувати, що критеріїв узгодженості (які насамперед залежать від застосунку), буде досягнено у всіх учасників РР. Оскільки ОП має неблокуючу властивість, локальний час відгуку є нечутливим до мережевих затримок. З урахуванням вищезазначених особливостей і переваг, ОП особливо підходить для реалізації колаборативних функцій, як-от групове редагування у Вебі та Інтернеті, що і було реалізовано у нашій системі.

1. Опис веббазованої системи для розподіленого розроблення програм

Основну ідею операційних перетворень можна проілюструвати на прикладі сценарію РР тексту. Дано текстовий документ з рядком тексту "abc", який репліковано у двох учасників редагування, а також дві паралельні операції:

1. O_1 = *Insert* [0, "x"] (вставити символ "x" на позицію "0")
2. O_2 = *Delete* [2, "c"] (видалити символ "c" з позиції "2").

Ці операції згенеровано двома учасниками РР (рис. 1).

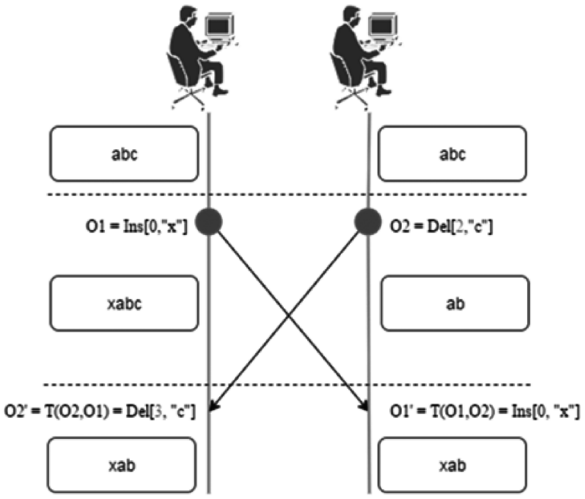

Рис. 1. Головний принцип операційного перетворення

Припустимо, що ці дві операції виконуються у такому порядку: спочатку O_1, потім O_2. Після виконання O_1 рядок у документі міститиме значення "xabc". Щоб виконати O_2 після O_1, O_2 має бути перетворено відносно O_1 і відповідно операція матиме такий вигляд: O'_2 = *Delete* [3, "c"]. Як можна побачити, перший параметр операції *Delete* було інкрементовано на одиницю у зв'язку зі вставкою символу "x" операцією O_1. Виконання O'_2 на рядку "xabc" має видалити коректний символ "c", і в результаті значення стане "xab". Варто зазначити, що у більшості систем для визначення порядку операцій використовується механізм векторного годинника [5].

У разі виконання операції O_2 без перетворення буде некоректно видалено символ "b" замість символу "c". Основною ідеєю ОП є перетворення (або налаштування) параметрів операцій редагування відповідно до ефектів попередньо виконаних паралельних операцій так, щоб перетворена операція досягала бажаного ефекту і зберігала узгодженість документа.

Розглянемо реалізацію поширеної функції будь-якого текстового редактора — Undo (відміна попередньої дії користувача). Маємо двох розподілених учасників редагування — Site 1 та Site 2. Також дано текстовий документ із рядком тексту "12" (рис. 2).

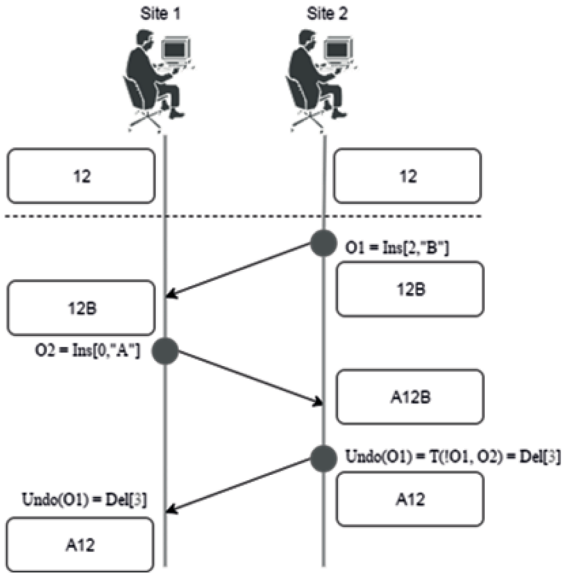

Рис. 2. Візуалізація відміни останньої дії (Undo)

Спершу користувач з Site 2 виконує вставку символу "B" на позицію 2, генеруючи наступну операцію: O_1 = *Insert* [2, "B"]. Ця операція також поширюється на Site 1 і застосовується без перетворень. У результаті маємо текст "12B" у обох учасників РР.

Далі користувач з Site 1 виконує вставку символу "A" на позицію 0, генеруючи наступну операцію: O_2 = *Insert* [0, "A"]. Ця операція також поширюється на Site 2 і застосовується без перетворень. У результаті маємо текст "A12B" у обох учасників РР.

Після виконання O_1 і O_2 користувач на Site 2 виконує відміну своєї попередньої дії, генеруючи команду *Undo* (O_1) яка власне не є останньою виконаною операцією. Система, що базується на підході ОП, спершу створить наступну інвертовану операцію:

$$!O'_1 = Inverse\ (O_1 = Insert\ [2, "B"]) = Delete\ [2].$$

Потім буде трансформовано $!O_1$ проти O_2: $!O'_1 = T(!O_1, O_2) = Delete$ [3]. Врешті-решт, після застосування $!O'_1$, буде отримано рядок "A12", що відповідає очікуваному ефекту операції Undo.

Підсумовуючи, зазначимо, що основною ідеєю операційних перетворень для операції відміни останньої дії користувача є інвертування операції O з урахуванням операцій, що були виконані після O задля досягнення ефекту відміни

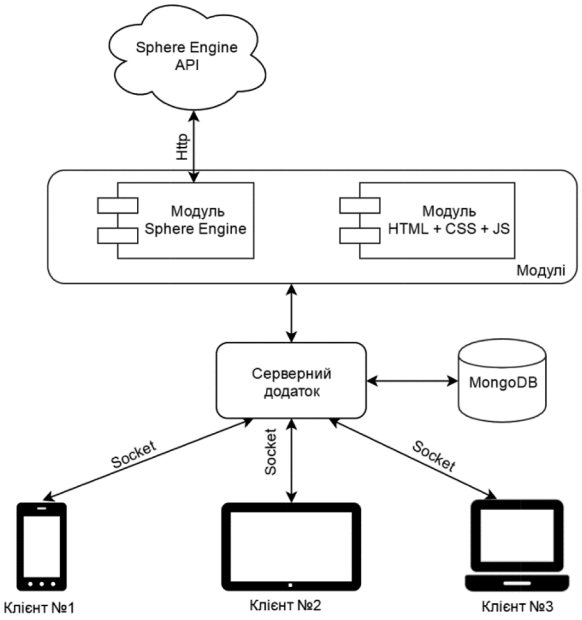

Рис. 3. Компонентна архітектура веббазованої системи для розподіленого розроблення програм

(Undo effect). Коректний ефект відміни досягається виключенням ефекту операції *O*, проте зі збереженням ефектів інших операцій.

2. Опис веббазованої системи для розподіленого розроблення програм

Порівняно з попередньо реалізованим прототипом [1], представлена тут система РР отримала нову та покращену функціональність. Також у зв'язку зі впровадженням модульної архітектури та механізму збереження вмісту кімнат було оновлено компонентну архітектуру застосунку (рис. 3).

Насамперед, вагомим доробком є підтримка окремих сесій редагування. Після відкриття головної сторінки вебзастосунку користувача буде перенаправлено на сторінку зі згенерованим випадковим чином префіксом. Цей префікс і відповідатиме id кімнати, у якій перебуває користувач. Одночасно до кімнати може бути підключено багато учасників РР. Для того щоб запросити до кімнати іншого учасника, достатньо скопіювати поточну адресу в браузері та поділитися нею, що робить процес РР найбільш простим та інтуїтивним. Після завантаження сторінки необхідно ввести ім'я користувача і натиснути кнопку "Join". Після входу інтерфейс системи має такий вигляд (рис. 4).

Для збереження внесених змін програмного коду та можливості відновлення втрачених даних у разі технічного збою сервера було використано нереляційну базу даних MongoDB [6]. Окрім збереження самого документа, що редагується у колаборативному розподіленому середовищі, зберігається вся історія правок документа, що дасть змогу за необхідності відтворити зміни.

Для зручності користувачів у правій частині середовища розміщено чат, де користувачі можуть спілкуватися між собою у поточній кімнаті.

Також було імплементовано механізм для прототипування клієнтських вебзастосунків.

Рис. 4. Інтерфейс веббазованої системи для розподіленого розроблення програм

Часто у розробників вебзастосунків виникає ситуація, коли необхідно швидко перевірити коректність програмного коду або верстки вебсторінки і поділитися цим з іншими розробниками. Саме для таких випадків новий механізм стане у пригоді. Після вибору режиму "HTML+CSS+JS" перед учасниками редагування з'являється чотири секції: HTML, JavaScript, CSS та Result (результат компіляції).

Структура скомпільованого вебзастосунку матиме такий вигляд:

```
<html>
    <head>
          <style>
          <!-- Вміст секції CSS -->
          </style>
    </head>
    <body>
    <!-- Вміст секції HTML -->
          <script>
          <!-- Вміст секції
                      JavaScript -->
          </script>
    </body>
</html>
```

Після компіляції вебзастосунку скомпоновану сторінку буде відображено в розділі Result, а JavaScript скрипти буде виконано (рис. 5).

В оновленій версії системи PP було розроблено гнучку модульну архітектуру для підтримки використання сторонніх компіляторів та їх інтеграції з вебсистемою для колаборативного редагування.

За програмним контрактом кожен створений адаптер компілятора повинен мати такі методи:

- `beforeCompile(obj)` — викликається перед виконанням компіляції та призначений для попереднього оброблення даних (наприклад, видалення спецсимволів, попередня валідація тощо). Як вхідний параметр приймає масив документів колаборативного редагування;
- `compile(obj)` — виконує компіляцію програмного коду. Як вхідний параметр приймає результат методу beforeCompile. У середині цього методу має бути здійснено виклик до стороннього компілятора або ж проведено компіляцію «на місці». Цей метод має бути асинхронним і повертати об'єкт `promise` одразу, а після виконання компіляції — результат компіляції;
- `processResponse(obj)` — готує результат компіляції до відображення у вікні виведення графічного інтерфейсу користувача. Як вхідний параметр приймає результат методу `compile`. Повертає рядок символів.

Конструктор компілятора може приймати необхідну кількість параметрів (наприклад, токен для компіляції тощо).

Для того щоб створити власний модуль адаптера до компілятора, необхідно виконати такі кроки:

1. Створити файл із розширенням `*.js`, що має міститися у папці `compilers` на тому самому рівні, що й запущений вебзастосунок.

Рис. 5. Режим "HTML+CSS+JS" у веббазованій системі для розподіленого розроблення програм

2. Створити модуль адаптера до компілятора. У загальному випадку модуль має таку структуру:

```
var q = require('q');

function compilerModule(param1,
param2) {
    this.output = «compose»;
    this.compile = function(obj) {
        var deferred = q.defer();
        deferred.resolve(obj);
        return deferred.promise;
    },

    this.processResponse =
                function(response) {
        return response;
    },

    this.beforeCompile =
                function(obj) {
        return obj;
    }
};
module.exports = function(module_
                          holder) {
    module_holder['compiler_module']
= compilerModule;
};
```

3. Зареєструвати компілятор у конфігураційному файлі. Для цього необхідно відкрити конфігураційний файл `config.js` та у масив `config.compilers` додати запис про новостворений адаптер у такому форматі:

```
{
    id: 'compiler_module', // назва
                компілятора, що була
                          // вказана
                  у експорті модуля
    params: ['param', 40] // масив
                     параметрів для
                        //ініціалі-
          зації об'єкту компілятора
}
```

4. У конфігураційному файлі у масиві `config.langs` додати або змінити компілятор, що використовується для певної мови (поле `compilerId`), наприклад:

```
{
id: 3,
    name: «Java»,
    compilerId: «compiler_name»,
    style: «text/x-java»,
    nested: «false»,
    view: «partial-single-document.
                                html»,
    placeholder: «code-area»,
    output: «compile-output»,
    visible: «true»
}
```

Опис значень параметрів конфігурації:

- **id** — унікальний ідентифікатор мови;
- **name** — назва мови (режиму), яка виводиться у списку графічного інтерфейсу користувача;
- **compilerId** — назва компілятора, що використовується для цієї мови;
- **style** — стиль текстового редактора CodeMirror для стилізації програмного коду;
- **nested** — позначає, чи підтримує ця мова (режим) багатодокументне редагування. Якщо значення цього параметра дорівнює true, то необхідно вказати ідентифікатори мов у масиві inner;
- **view** — назва html-документа, який буде відображено для поточної мови. Має містити секції для введення коду і для виведення результату;
- **placeholder** — ідентифікатор div для розміщення коду;
- **output** — ідентифікатор div для розміщення результату;
- **visible** — чи показувати цю мову у списку мов на клієнті.

5. Перезапустити серверний застосунок. Під час запуску сервер динамічно підвантажить усі наявні адаптери з папки `compilers`.

Така архітектура є досить гнучкою і дає змогу використовувати різноманітні сторонні компілятори. За необхідності нові адаптери до компіляторів можна додавати без правки коду серверного застосунку, що є якісною ознакою гарної архітектури вебсистеми.

Висновки

У рамках цієї роботи представлено створену веббазовану систему, що базується на операційному підході до розподіленого редагування, а також значно розширено гнучкість модульної архітектури та функціональність попередньо розробленого прототипу [1].

У статті описано архітектуру, використані технології створеної веббазованої системи розподіленого розроблення програм з акцентом на ефективне й просте одночасне групове створення тексту програми та її редагування і відлагодження. Впровадження модульної архітектури та механізму збереження вмісту забезпечило злагоджену функціональність компонент систе-

ми й ефективну підтримку узгодженості, управління неблокуючим паралелізмом. Оскільки система має неблокуючу властивість, локальний час відгуку є нечутливим до мережевих затримок. Для збереження внесених змін програмного коду та можливості відновлення втрачених даних у випадку технічного збою сервера використано нереляційну базу даних MongoDB.

Важливою особливістю системи є використання гнучкої модульної архітектури для підтримки ефективного застосування сторонніх компіляторів та їх інтеграції з вебсистемою за колаборативного редагування.

Для зручності спілкування користувачів у системі реалізовано чат. Вагомим доробком є підтримка окремих сесій редагування. Також імплементовано механізм прототипування клієнтських вебзастосунків, що дає змогу розробникам вебзастосунків швидко перевірити коректність програмного коду чи верстки вебсторінки і поділитися цим з іншими розробниками. Саме для таких випадків новий механізм стане у пригоді.

Прикладом застосування розробленої системи може бути використання у електронному навчанні. З огляду на стрімке поширення засобів електронного навчання у вищій школі [2], а також дистанційний характер навчання протягом останніх років, запропонована система може сприяти покращенню рівня співпраці учасників навчального процесу.

Подальшим розвитком цієї роботи може бути дослідження неконфліктуючих реплікованих типів даних (CRDT) як альтернативи операційним перетворенням для забезпечення синхронізації даних за колаборативного редагування.

Список літератури

1. Бублик В. В. Колаборативні методи в електронному навчанні програмування / В. В. Бублик, А. М. Давиденко // Наукові записки НаУКМА. — 2016. — Т. 190. Комп'ютерні науки. — С. 41–45.
2. Глибовець М. М. Застосування Semantic WEB до створення колаборативного освітнього простору / М. М. Глибовець // Збірник праць П'ятої Міжнародної конференції «Нові інформаційні технології для всіх». — Київ : Академперіодика, 2010. — С. 179–192.
3. Ellis C. A. Concurrency control in groupware systems / C. A. Ellis, S. J. Gibbs // Proceedings of the 1989 ACM SIGMOD international conference on Management of data / C. A. Ellis, S. J. Gibbs. — New York, NY, USA : ACM, 1989. — Pp. 399–407.
4. Copies Convergence in a Distributed Realtime Collaborative Environment / N.Vidot, M. Cart, J. Ferrie, M. Suleiman // Proceedings of the ACM Conference on Computer-Supported Cooperative Work / N.Vidot, M. Cart, J. Ferrie, M. Suleiman., 2000. — Pp. 171–180.
5. Lamport L. Time, Clocks, and the Ordering of Events in a Distributed System / L. Lamport // Communications of the ACM 21. — 1978. — No. 21. — Pp. 558–565.
6. MongoDB Manual [Electronic resource]. — Mode of access: https://docs.mongodb.com/manual/.

References

Bublyk, V. V., & Davydenko, A. M. (2016). Kolaboratyvni metody v elektronnomu navchanni prohramuvannia. *Naukovi zapysky NaUKMA. Kompiuterni nauky*, *190*, 41–45 [in Ukrainian].

Hlybovets, M. M. (2010). Zastosuvannia Semantic WEB do stvorennia kolaboratyvnoho osvitnoho prostoru. In *Zbirnyk Prats Piatoi Mizhnarodnoi Konferentsii "Novi Informatsiini Tekhnolohii Dlia Vsikh"* (pp. 179–192). Akademperiodyka [in Ukrainian].

Ellis, C. A., & Gibbs, S. (1989). Concurrency control in groupware systems. *Sigmod Record*, *18* (2), 399–407. https://doi.org/10.1145/66926.66963.

Vidot, N., Cart, M., Ferrie, J., & Suleiman, M. (1999). Copies Convergence in a Distributed Realtime Collaborative Environment. *Proc. ACM Conf. Computer-Supported Cooperative Work (CSCW '00)*, 171–180.

Lamport, L. (1978). Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, *21* (7), 558–565. https://doi.org/10.1145/359545.359563.

What is MongoDB? — MongoDB Manual. (n.d.). https://docs.mongodb.com/manual/.

A. Davydenko, M. Glybovets

WEB-BASED SYSTEM FOR DISTRIBUTED GROUPWARE SOFTWARE DEVELOPMENT

The article describes the architecture and technologies used to create a web-based distributed software development system with an emphasis on efficient and simple simultaneous grouped creation of program text, its editing, and debugging. The introduction of a modular architecture and a content storage mechanism ensured the coordinated functionality of the system components and effective support for consistency and non-blocking parallelism management. Since the system has a non-blocking property,

the local response time is insensitive to network delays. The non-relational MongoDB database is used to save the changes made to the program code and to recover lost data in the event of a technical server failure.

An important feature of the system is the use of a flexible modular architecture to support the effective use of third-party compilers and their integration with the web-based system for collaborative editing.

To facilitate user communication, the system has a chat feature. A significant improvement is the support of separate editing sessions. A mechanism for prototyping client web applications has also been implemented, which allows web application developers to quickly check the correctness of the program code or web page layout and share it with other developers. It is for such cases that the new mechanism will come in handy.

The areas of its application and further improvement are briefly presented. Further development of the system may include the study of the implementation of nonconflicting replicated data types (CRDTs) as an alternative to operational transformations to ensure data synchronization during collaborative editing.

Keywords: operational transformations, distributed systems, groupware, web-development, programming, collaborative environments

Матеріал надійшов 27.11.2023