УДК 004.9

М. В. Арзубов, Н. Б. Шаховська
Національний університет “Львівська політехніка”,
кафедра інформаційних систем та мереж

РОЗШИРЕННЯ ДЛЯ ПОШУКУ І ВИДАЛЕННЯ ШКІДЛИВОЇ ЧИ НЕПОТРІБНОЇ ІНФОРМАЦІЇ В ІНТЕРНЕТ-БРАУЗЕРІ

Описано програмне забезпечення для пошуку і видалення шкідливої чи непотрібної інформації у інтернет-браузері. Визначено мету, завдання і сферу застосування розширення.

Ключові слова: Chrome Extension, Node.js, express, JavaScript I/O, ECMAScript6, Babel, WebPack, Redis, логістична регресія, метод найшвидшого спуску.

In this article software for searching and removing harmful or unnecessary information is described. Goals, objectives and scope of such an extension are defined.

Key words: Chrome Extension, Node.js, express, JavaScript I/O, ECMAScript6, Babel, WebPack, Redis, logistic regression, gradient descent method.

Вступ. Загальна постановка проблеми

В Інтернеті є дуже велика кількість різних веб-сайтів, які можуть містити інформацію будь-якого змісту. Веб-переглядачі практично не фільтрують вмісту сторінок. Дуже багато веб-сайтів містять рекламу, відео, банери різного змісту, які можуть блокувати чи відволікати користувача. Для забезпечення кращого перегляду інформації у веб-переглядачі потрібно її фільтрувати. До користувача може доходити будь-яка інформація. Це і пояснює таку велику кількість різноманітної реклами. Але потрібно контролювати вміст цих сторінок. Тому було вирішено розробити таке розширення, яке б фільтрувало цю інформацію у веб-переглядачі з найменшими затратами ресурсів, щоб не блокувати користувача. У 2013 р. у Великій Британії було проведено соціальне опитування на тему нав’язливості реклами в Інтернеті. Результати були такими:

- Близько 21 % опитуваних стверджували, що їх дратує вся реклама.
- Майже 38 % сказали, що ставляться нейтрально до реклами в Інтернеті.
- Решта відповіли, що вони позитивно ставляться до реклами в Інтернеті.

З’ясували також, яка саме реклама найбільше дратує користувачів Всесвітньої павутини. Результати ось такі:

- 80 % опитаних дратували вікна, які випливали.
- 6 % людей дратувала реклама на все вікно.
- 80 % опитаних дратувала реклама під час перегляду відео в Інтернеті.

Залишається близько 10 %, яких усе влаштовує або яким просто байдуже. А що ж робити решті 90 %, яких дратують певні аспекти веб-перегляду? Тому потрібно контролювати вміст цих сторінок.

Отже, метою цієї роботи є розроблення такого розширення, що покращило б веб-перегляд користувача, яке б фільтрувало цю інформацію у веб-переглядачі з найменшими затратами ресурсів, щоб не блокувати користувача. Також потрібно створити веб-сервер, який би міг зберігати вибірки для фільтрування.

1. Аналіз літературних та інших джерел

Для створення розширення використано технологію Google Chrome Extension, яка дає можливість інтегрувати ваше програмне забезпечення у веб-переглядач з доступом до всіх запитів і відкритих сторінок. Для серверної частини було вибрано Node.js сервер з використанням фреймворку Express.js. Використання цих технологій дає змогу асинхронно обробляти запити, що в

цьому випадку якраз і потрібно. Слід зауважити і про використання тільки однієї мови у розширенні й на сервері. Також вибрано базу даних Redis як сховище для зберігання списків вибірок. В ній дуже зручно зберігати різноманітні великі списки і працювати з ними. Redis зберігає дані у пам'яті, що дає нам можливість швидко отримувати дані.

Вибрано саме ці технології, щоб показати високу швидкість та продуктивність роботи з великими списками і фільтрацією даних.

Redis – структура даних з відкритим сирцевим кодом (BSD ліцензія), що використовується як база даних, кеш-пам'ять. Підтримує структури даних, такі як рядки, хеші, списки, набори, сортовані набори з різними запитами, растрові зображення, логи і геопросторові індекси з радіальними запитами. Redis має вбудовані реплікації, Lua скриптів, LRU, транзакції та різні рівні зберігання даних на диску і забезпечує високу доступність за допомогою Redis Sentinel і автоматичного розбиття з Redis Cluster [1].

Redis написаний на ANSI C і працює у більшості систем POSIX, як Linux, * BSD, OS X без зовнішніх залежностей. Linux і OS X є двома операційними системами, де Redis розробляється і більш перевірений [2].

Як тільки ви почнете використовувати Redis, вас може зацікавити питання: "Наскільки багато ключів я можу мати ?". Вас також може зацікавити, скільки полів може містити хеш чи скільки елементів може зберігатись у списку. На практиці в базі даних Redis можуть зберігатись сотні мільйонів елементів [7].

Node.js – платформа виконання JavaScript, побудована на основі Chrome V8 JavaScript двигуна. Автором цього проекту є Раян Дал. Node.js використовує неблокувальну модель виконання подій, що робить його легким і ефективним. Npm – Node.js пакетний менеджер є найбільшим у світі [3].

Node.js є сервером JavaScript, який здатний підтримувати масштабовані веб-додатки з високою продуктивністю. З використанням асинхронного введення/виведення сервер може робити більше ніж одну річ одночасно – це ключова вимога для додатків реального часу, таких як чат, ігри і жива статистика. І, оскільки це JavaScript, ви можете використовувати її на всіх сторонах вашого проекту [8].

Express.js – це Node.js фреймворк, розроблений для створення односторінкових, багатосторінкових та гібридних веб-аплікацій. Вважається мінімалістичним і швидким. Також має багато плагінів [5].

Node Natural – бібліотека для роботи з природною мовою, з реалізованими загальними методами для роботи з природною мовою [4].

Node_redis – Redis клієнт для роботи з Node.js. Він підтримує всі команди Redis, серед яких і нещодавно додані команди, такі як EVAL з експериментальних віток Redis server [6].

Також створено бібліотеку hiredis для Node.js, яка надала зв'язок з офіційним hiredis C бібліотекою, яка є не блокувальною і швидкою, що дає змогу істотно підвищити продуктивність.

2. Основна частина

2.1. Сервер

Було створено Node.js сервер, який працював з базою даних Redis за допомогою node_redis клієнта. Це надало можливість зберігати статистику і потрібну інформацію не у розширенні веб-переглядача, а на сервері, що зробило проект гнучкішим. Завдяки серверу можна підтримати однакову версійність інформації, з якою працює розширення веб-переглядача для блокування шкідливої інформації.

Робота сервера складається з таких частин:

1. Зберігання даних у Redis DB та робота з ними.
2. Фільтрування інформації на вже завантаженій сторінці.
3. Перевірка посилання на веб-сторінку.

Також на сервері можна перевірити посилання на веб-сторінку. Це реалізовано на основі методу логістичної регресії. Для цього використано бібліотеку natural [8].

Для максимізації цієї функції правдоподібності, еквівалентної максимізації її логарифма застосовано метод градієнтного спуску.

Тренувальна вибірка складалась з близько 5000 тисяч посилань на веб-сайти, що дало змогу отримати доволі прийнятний результат перевірки.

Під час перевірки на тестовій вибірці досягнуто більш як 95 % правильних результатів валідації списку шкідливих ресурсів і 93 % за прийнятних ресурсів.

Тренування тривало до трьох хвилин на початковій вибірці, яка складалась з близько 5000 посилань на веб-ресурси. Після цього перевірка із запитом на сервер відбувалась до 50 мс, що є більш ніж прийнятним результатом.

Досягнуто високої швидкості та продуктивності, що дає змогу підтримувати велику кількість запитів, тобто сервер можна використовувати не тільки для зберігання даних і статистики, але й як сервіс для перевірки адрес посилань.

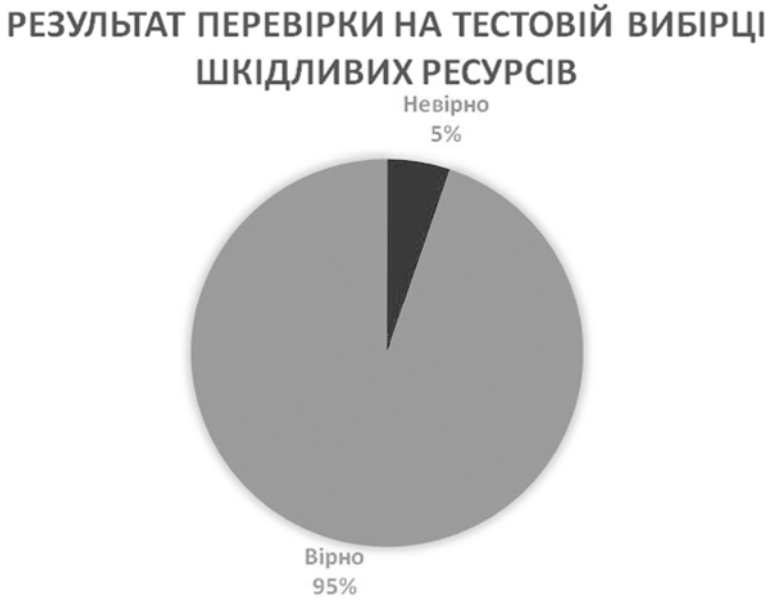

Рис. 1. Результат перевірки на тестовій вибірці шкідливих ресурсів

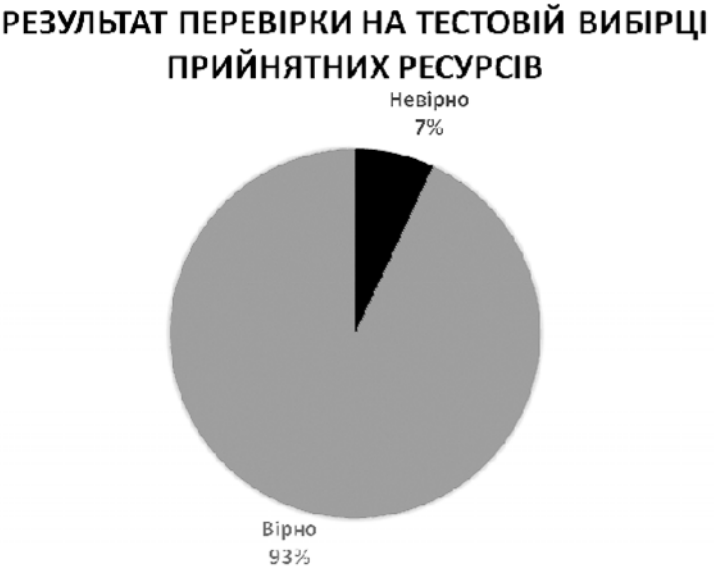

Рис. 2. Результат перевірки на тестовій вибірці прийнятних ресурсів

2.2. Розширення

Створено додаток до веб-переглядача Chrome, який працює на кожній сторінці, яку користувач відкриває. Програма складається з двох частин:

- Контент-скрипт.
- Фонова сторінка.
- Web Worker.

Робота розширення складається з кількох частин, а саме:

1. Перевірка назви хоста перед завантаженням сторінки.
2. Фільтрування інформації на вже завантаженій сторінці.
3. Перевірка запитів зі сторінки.

Контент-скрипт – це JavaScript файл, який запускається у контексті веб-сторінки. Також він працює в тому самому потоці. Тому одним з основних завдань було якнайменше навантажувати сторінку, щоб не блокувати користувача. Основні завдання контент-скрипту – це:

- створити зв'язок з фоновою сторінкою;
- швидко видобути інформацію про вміст сторінки і передати її;
- у разі надходження відповідного повідомлення з фонової сторінки видалити елемент з ймовірним шкідливим вмістом.

Фоновий скрипт не може змінювати вмісту сторінки, тому після валідації потрібно передавати відповідну команду до контенту, бо тільки він має доступ і можливість змінювати інтерфейс. Зв'язок з фоновою сторінкою створюється за допомогою long-lived chrome port connection. Це надає змогу швидко та асинхронно передавати повідомлення між частинами розширення. Отримавши повідомлення, завжди можна визначити, звідки воно прийшло, що дає нам змогу відправляти інформацію до конкретних сторінок. Приклад збирання інформації та її відправки до фонової сторінки:

```
//expand this to validate all doubtful blocks
$nodesToValidate = $('[href], [src], [data], [id],
[class]');
if ($nodesToValidate.length) {
  try {
    let data = [];
    Array.prototype.slice.call($nodesToValidate)
      .forEach(function (node, index) {
        data.push({
          href: node.href || '',
          src: node.src || '',
          id: node.id || '',
          data: node.data || '',
          className: node.className || '',
          nodeId: index
        });
      });
    port.postMessage({
      cmd: 'validateNode',
      data: data,
      windowName: windowName
    });
  }
  catch (e) {
    log(e);
  }
}
```

Фоновий скрипт працює в окремому потоці, тому він не блокує користувача. Основна логіка роботи програми якраз розміщена у цій частині. У фоновому скрипті перевіряється назва хоста перед завантаженням будь-якої інформації з сервера. Це єдина частина, яка повинна виконуватись синхронно. Основні завдання фонової сторінки такі:

- створити зв’язок зі всіма контент-скриптами;
- слухати до оновлення адреси сторінки;
- обробляти всі запити зі сторінок ще до їхнього виконання;
- швидко й асинхронно обробляти повідомлення;
- виконувати саму перевірку у Web Worker, який запускається в іншому потоці, на основі вибірок, отриманих з сервера за допомогою регулярних виразів;
- зберігати потрібну інформацію у локальному Chrome Storage

```
chrome.tabs.onUpdated.addListener(function (tabId, changeInfo, tab){
  //validate location synchronously
  var location = _urlParser.getHost(tab.url);
  if (_hostValidation.test(location)) {
    log('blocking tab:' + tab.url);
    localStorage.setItem('lastForbiddenPage', tab.url);
    //redirect tab with bad resource to noizy-ads forbidden page
    chrome.tabs.update(tabId, {url: 'views/forbidden.html'});
  }
});
```

Після цього запускається перевірка всіх запитів зі сторінки, що дає змогу блокувати завантаження непотрібної інформації, ще до їх виконання.

```
chrome.webRequest.onBeforeRequest.addListener(
  (details)=>{
    var blocked,
        tabId = details.tabId,
      reqType = details.type;

    //validate url synchronously
    var location = _urlParser.getHost(details.url);
    blocked = _hostValidation.test(location);
    return { cancel: blocked };
  },
  {urls: ['http://*/*', 'https://*/*']}, ['blocking']
);
```

Далі контент-скрипт, який прикріпляється до сторінки, витягує всю інформацію одним запитом і передає її на фоновий скрипт, який працює у іншому потоці.

```
port.onMessage.addListener(function (request, sender, sendResponse)
{
  if (request) {
    var cmd = request.cmd,
      windowName = request.windowName,
      data = request.data;
    worker.postMessage({
      cmd: cmd,
      data: data,
      windowName: windowName,
      sender: sender
    });
  } else {
    log('Error: empty request');
  }
});
```

Фоновий скрипт передає інформацію для перевірки у Web Worker, який працює у іншому потоці. Там відбувається асинхронна перевірка всіх запитів зі сторінки, що дає змогу заблокувати шкідливу інформацію ще до завантаження. Web Worker виконує найбільш трудомістку роботу, працює у додатковому потоці, що приводить до зменшення навантаження на основні частини розширення та дає можливість не блокувати роботу програмного забезпечення і веб-переглядача. Також можна створювати кілька Web Worker, тобто в разі розширення перевірок чи збільшення навантажень можна розділити функціонал і винести у різні Web Worker.

Далі на UML діаграмі (рис. 3) схематично зображено роботу розширення.

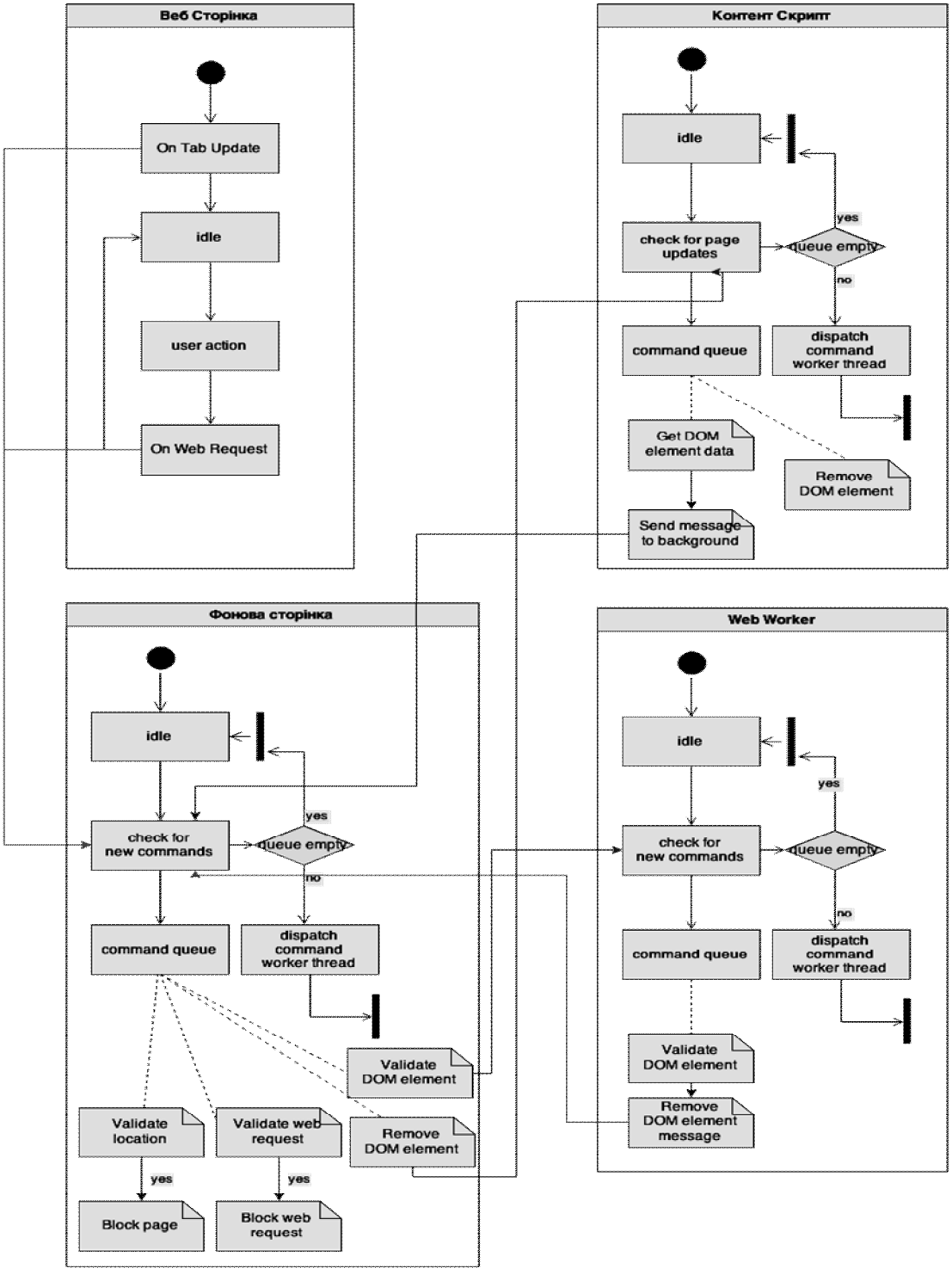

Рис. 3. UML діаграма роботи розширення

Досягнуто високої швидкості та продуктивності, що дає змогу підтримувати велику кількість сайтів. Контент-скрипт мінімально навантажує сторінку, що не блокує користувача.

Тепер вибірка складається з близько 2200 шкідливих сайтів і 2200 прийнятних сайтів, у разі збільшення їх кількості відповідно підвищуватиметься кількість блокувальної інформації на веб-сайтах. Протягом 1–2 секунд відбувається асинхронна перевірка DOM елементів на шкідливість без блокування користувача. На рис. 4 зображено меню розширення, яке містить назву та статистику автора і репозиторію.

Рис. 4. Меню з інформацією про розширення

У разі блокування якогось сайта, для прикладу www.liveinternet.ru, користувача перенаправляють на сторінку розширення, де відображена інформація перевірки та є можливості різних дій, як показано на рис. 5.

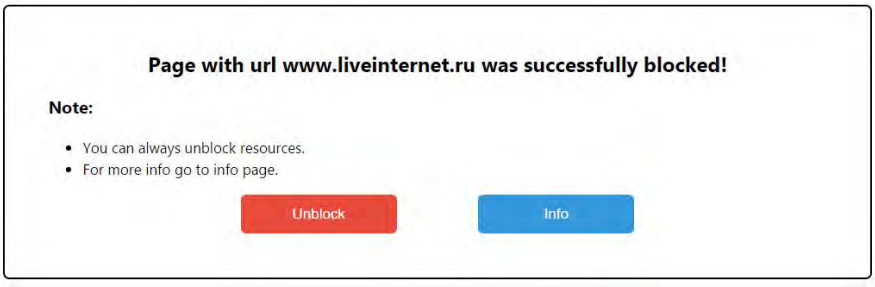

Рис. 5. Приклад блокування шкідливої сторінки

Також розширення показує кількість заблокованої інформації з моменту відкриття сторінки (рис. 6).

Показано високу сумісність і продуктивність у разі одночасної роботи з AdBlock (рис. 7). Блокування інформації для цього і попереднього прикладу відбувалось на сайті sport.ua.

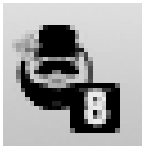

Рис. 6. Приклад зображення кількості заблокованої інформації

Рис. 7. Приклад зображення кількості заблокованої інформації одночасно з AdBlock

У фоновій сторінці під час блокування виводиться в консоль інформація про шкідливий ресурс. Приклад зображено на рис. 8.

Рис. 8. Приклад логування інформації про заблоковані ресурси у фоновій сторінці

Висновки та перспективи подальших розвідок

У статті описано та реалізовано розширення до веб-браузера Google Chrome та NodeJS сервера з базою даних Redis для перевірки на шкідливість веб-посилань. Технологією реалізації програмного продукту вибрано сучасні технології розроблення застосунків для браузера на основі HTML5, CSS3 і JavaScript з експериментальними можливостями. Коректність програмної реалізації алгоритму логістичної регресії підтверджено статистичними результатами. Досягнуто більш ніж задовільної швидкості перевірки у реальному часі. На основі отриманих результатів можна зробити

висновок про доцільність використання цього розширення у веб-переглядачі та методу логістичної регресії для валідації посилань на сервері. Варто зауважити, що цей додаток можна використовувати спільно з додатком AdBlock, вони не конфліктують і захищають від більшої кількості загроз. Також можна використовувати веб-сервер для перевірки шкідливості того чи іншого ресурсу.

1. VMware: the new Redis home [Електронний ресурс]. – Режим доступу: http://antirez.com/post/vmware-the-new-redis-home.html. 2. Документація Redis "Virtual Memory" [Електронний ресурс]. – Режим доступу: http://redis.io/documentation. 3. Nodejs [Електронний ресурс]. – Режим доступу: https://nodejs.org/. 4. NaturalNode документація [Електронний ресурс]. – Режим доступу: https://github.com/NaturalNode/natural/blob/master/README.md. 5. Express.js [Електронний ресурс]. – Режим доступу: http://expressjs.com/. 6. Node redis документація [Електронний ресурс]. – Режим доступу: https://github.com/mranney/node_redis. 7. The Little Redis Book [Електронний ресурс]. – Режим доступу: https://github.com/karlseguin/the-little-redis-book/blob/master/en/redis.md. 8. Mike Cantelon, Marc Harter, T. J. Holowaychuk and Nathan Rajlich. Node.js in Action // Manning Publications October. – 2013 – 416 с.

УДК 004.89;004.048;004.416.3

Є. В. Буров, В. В. Пасічник
Національний університет "Львівська політехніка",
кафедра інформаційних систем та мереж

ПРОГРАМНІ СИСТЕМИ НА БАЗІ ОНТОЛОГІЧНИХ МОДЕЛЕЙ ЗАДАЧ

Розглянуто застосування онтологічних моделей задач для побудови програмних систем, здатних адаптуватися до зміни стану предметної області. Розроблено математичну формалізацію подання та опрацювання знань у системі, архітектуру та принципи функціонування програмної системи на базі онтологічних моделей. Розглянуто методи використання онтологічних моделей для розв'язання практичних задач. Проаналізовано переваги застосування онтологічних моделей порівняно з традиційним підходом до побудови програмних систем та розроблено формули для оцінювання їх переваг. Розроблено програмний інструментальний засіб для побудови та моделювання програмних систем на основі онтологічних моделей задач.

Ключові слова: онтологія, онтологічна модель, програмна система, адаптація.

The increased mobility of business processes today relies on extensive use of software and in turn puts high demands on the quality of software and its ability to be quickly and accurately adapted to the changes in the business environment. A promising approach to solving the problem of adapting the software to changes in its operational environment is the use of ontological modeling. In the article the theoretical principles of knowledge representation and processing in software based on ontological task models were developed. A formal model of knowledge representation was built. The method of using ontological task models for automated testing of nightly software builds was developed. This method also demonstrates the organization of models interaction in the multistage business process of automated testing.

Key words: ontology, ontological model, software system, adaptation.

Вступ та постановка проблеми

Аналіз тенденцій розвитку програмних систем показує, що темпи їх змін та складність продовжують зростати. Це відповідає загальним трендам глобалізації у світовій економіці та необхідності швидко реагувати на зміни у бізнесовому середовищі. Підвищення мобільності бізнес-