без зупинки їх роботи, створити проект цієї мережі та виконати необхідні експерименти для визначення граничних характеристик, можливості розширення, зміни топології і модифікації мережного обладнання з метою подальшого його вдосконалення і розвитку. Для реалізацій функцій імітаційного моделювання у складі NetCracker передбачені засоби завдання характеристик трафіків різних протоколів; засоби візуального контролю заданих параметрів; засоби накопичення статистичної інформації та формування звітної документації про проведені експерименти. Все це говорить про можливість використання системи для дослідницьких і навчальних задач проектування, моделювання та аналізу комп'ютерних мереж.

ЛІТЕРАТУРА

1. Проектирование и диагностика компьютерных систем и сетей: уч. пособ. / [Бондаренко М.Ф., Кривуля Г.Ф., Рябцев В.Г. и др.]. – Киев: НМЦ ВО, 2000. – 306с.
2. Пономаренко Л.А. Инструментальные средства проектирования, имитационного моделирования и анализа компьютерных сетей: уч. пособ. для студ. вузов / Л.А.Пономаренко, В.И.Щелкунов, А.Я.Скляров. – 2-е изд., испр. и доп. – Харьков: Компания СМИТ, 2006. – 488с.
3. Бабенко М.В. Моделювання комп'ютерних мереж як засіб вивчання, проектування та оптимізації роботи мережі / Бабенко М.В., Алексеєва Ю.О. // Збірник наукових праць Дніпродзержинського державного технічного університету (технічні науки). – 2012. – Випуск 3 (20). – С.168-174.

Надійшла до редколегії 25.04.2016.

УДК 004.9

БОЖУХА Л.М., к.ф.-м. н., доцент
ЗІНЬКОВСЬКИЙ Д.В., студент

Дніпродзержинський державний технічний університет

ПРО НОВІТНІ ТЕХНОЛОГІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ТА ЇХ НЕДОЛІКИ ПРИ РОЗРОБЦІ ДОДАТКУ ТАЙМ-МЕНЕДЖЕРА

Вступ. Час є найбільш цінним ресурсом. Точність і правильність цього твердження особливо легко відчути в нашу соціально-онлайнову епоху, коли банальне бажання подивитися прогноз погоди непомітно перетворюється на присід за комп'ютерний стіл тривалістю кілька десятків хвилин. Дана проблема поширена настільки, що навіть встигла отримати спеціальну назву – прокрастинація. Прокрастинація описує процес, коли людина усвідомлює всю важливість та необхідність виконання певної справи, але натомість цього витрачає дорогоцінний час даремно, відволікаючись на різні дрібниці та розваги.

«Тайм-менеджмент» – управління своїм часом, досягнення поставлених цілей та вірна їх мотивація. На винахід цього терміну претендує компанія Time Management International. Її засновник, данець Клаус Меллер, в 70-ті роки винайшов Time Manager – складно влаштований блокнот-щоденник, який можна вважати прабатьком сучасного органайзера. Тайм-менеджмент – це технологія, яка дозволяє керувати часом у реальних ситуаціях повсякденного життя. Вирішення цієї проблеми викликало бажання написати time-manager під Android ОС, який включав би в себе засоби для продуктивного керування власними справами.

Аналізуючи ринок подібних пропозицій, можна дійти висновку, що вони всі або платні, або з малим функціоналом, або частково платні. Це й стало мотивом для створення власного тайм-менеджеру. Метою розробки додатку є створення механізму будильника для нагадувань про діяльність. Нагадування має оповіщати про діяльність звуко-

вим сигналом та вібрацією, якщо вона увімкнена у телефоні. При роботі з повідомленнями додатку може надаватися можливість вибору відстеження діяльності, видалення/ редагування. Створення механізму відстеження часу і таймерів реалізує вибір типу таймеру за обмеженістю. Робота кожного таймера повинна бути виділена в окремі потоки.

Постановка задачі. Мобільні програмні рішення дуже відрізняються від десктопних програм тим, що потрібно слідкувати за рівнем заряду батареї, відстежувати орієнтацію екрану, перезавантаження відбувається частіше, ніж у персональних комп'ютерів. Це дуже часто спонукає розробників проектувати варіанти роботи додатку для подальшого продовження роботи додатку після перезавантаження. Усі ці проблеми плануються бути вирішеними при розробці програмного додатку «Тайм менеджер під управлінням Android ОС» [1].

Ключовим моментом створення програмного забезпечення є аналіз функціональної взаємодії об'єктів автоматизації [2]. В основі функціонального аналізу лежить принцип декомпозиції дій. Результатом аналізу в цьому контексті є функціональна модель, що дає уявлення про предметну область у термінах функцій і груп даних, що супроводжують виконання цих функцій. У найбільш узагальненому вигляді функціональна модель – це опис предметної області, заснований на аналізі семантики об'єктів і явищ, виконаний без орієнтації на використання програмних або технічних комп'ютерних засобів, у якому акцентується функціональний аспект моделювання предметної області [3].

Результати роботи. В якості засобу розробки функціональної моделі предметної області обрана методологія IDEF0, яка передбачає представлення результатів аналізу предметної області у вигляді діаграм: контекстної діаграми (рис.1) та діаграми декомпозиції (рис.2).

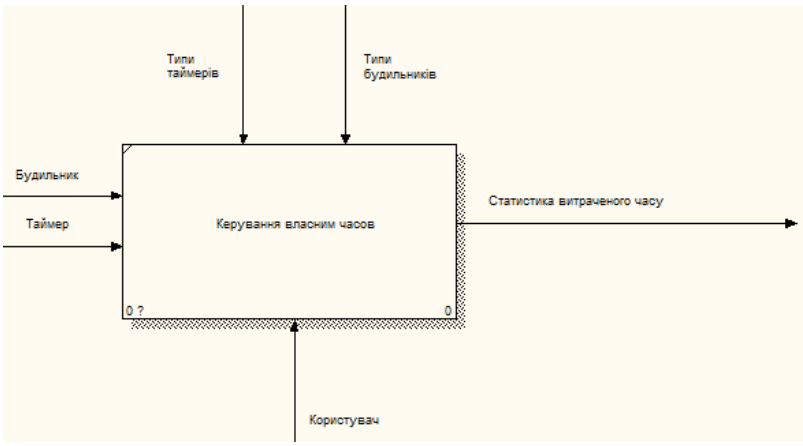


Рисунок 1 – Контекстна діаграма предметної області

Основним призначенням інформаційної системи (ІС) є оперативне забезпечення користувача інформацією про зовнішній світ шляхом реалізації питально-відповідного відношення. Питально-відповідні відношення дозволяють виділити для ІС певний її фрагмент – предметну область (ПрО), яка буде втілена в автоматизованій ІС. Інформація про зовнішній світ подається в ІС у формі даних, що обмежує можливості змістовної інтерпретації інформації й конкретизує семантику її подання в ІС.

Сукупність цих виділених для ІС даних, зв'язків між ними й операцій над ними утворить інформаційну й функціональну моделі ПрО, що описують її стан із певною точністю. Інформаційна й функціональна моделі ПрО є вхідними даними для процесу проектування БД [2].

Сутність ПрО є результатом абстрагування реального об'єкта шляхом виділення й фіксації набору його властивостей. Об'єкти взаємодіють між собою через свої властивості, що породжує ситуації. Ситуації – це взаємозв'язки, які виражають взаємини між об'єктами. Ситуації у предметній області описуються за допомогою висловлювань про ПрО з використанням виразами і обчисленнями предикатів, тобто формальної, ма-

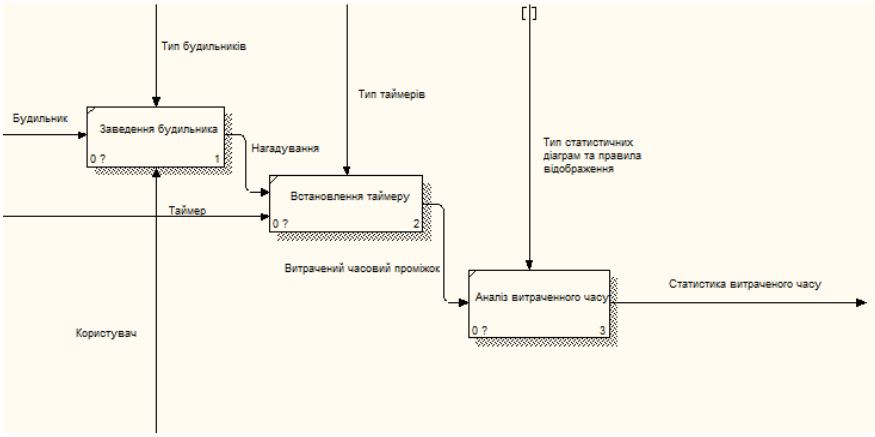


Рисунок 2 – Діаграма декомпозиції першого рівня предметної області

тематичної логіки. Розрізняють статичні й динамічні ситуації. Динамічні ситуації використовуються в одному циклі відтворення, наприклад, надходження нагадування на смартфон; статичні ситуації використовуються в багатьох циклах відтворення, наприклад, множинний відлік таймерів [4].

В якості інструментарію відображення об'єктної моделі предметної області обрано діаграму сутність-зв'язок. Враховуючи вимоги розробки діаграми сутність-зв'язок, описана специфікація об'єктів предметної області та специфікація атрибутів сутностей. Множинність зв'язку між сутностями «Будильник» та «Повідомлення будильника» становить 1:1.

Загальний вигляд діаграми «сутність-зв'язок» для описаної предметної області показано на рис.3.

Рисунок 3 – Загальний вигляд діаграми «сутність-зв'язок»

Проблеми ринку Android-додатків такої вузької спеціалізації, як time management територіальні та пов'язані з купівельною спроможністю власників смартфонів на території України. Ці проблеми взаємопов'язані, бо чималий відсоток від західних додатків є платними чи частково безкоштовними, які виражаються у обмеженому функціоналі, без якого або важко або взагалі обійтись неможливо.

Перевантаження функціями – є актуальною проблемою будь-якого програмного забезпечення. Н а сьогодні середній екран смартфона – це 4 дюйми; простота та мінімалістичність не в збиток продуктивності додатку є основними важелями з завоювання уваги клієнта. Але в той же час багато з розробників нехтують цим знанням.

Архітектура Android є фреймворк-орієнтованою (framework-based) у противагу вільній (free-style) архітектурі. Вільні додатки, написані на Java, починаються з класу, що має метод main (), і розробляються в повній відповідності до настрою розробника. На противагу цьому фреймворк-орієнтовані додатки ґрунтуються на існуючому фреймворку. Їх розробка зводиться до розширення деяких класів або реалізації інтерфейсів, наданих фрейморками. Такий додаток не може бути запущений поза фреймворком або без нього. Прикладом можуть бути веб-додатки на Java, в яких розробники реалізують інтерфейс Servlet або розширюють одну з його реалізацій, або програми Eclipse RCP, в якому розробник розширює один з класів Editor або View. Фреймворк-орієнтована архітектура обмежує свободу розробників, наказуючи їм, що і як слід робити. Але, в під-

сумку, зникають повторювані ділянки коду (boilerplate code) і розробникам доводиться ретельно слідувати шаблонами проектування.

Для Java існує безліч фреймворків. Тим не менш, команда Android вирішила створити свій власний. Можливо, однією з причин, по якій вони так вчинили, була необхідність підтримувати унікальну систему управління пам'яттю. В «звичайній» Java об'єкти знаходяться в пам'яті до тих пір, поки збирач сміття НЕ добереться до них. Відбувається це тільки тоді, коли на об'єкт немає жодного посилання від «живих» об'єктів. В Android це не так. Якщо якийсь інтерфейс користувача ховається (тобто його більше не бачимо на екрані), то немає ніякої гарантії, що він знаходиться в пам'яті, навіть є додаток, який надалі збирається використовувати цей інтерфейс. Якщо у ОС Android є достатньо вільної пам'яті, ці об'єкти можуть зберігатися в ній, але збирач сміття може знищити їх у будь-який момент, коли ОС вирішить, що пам'яті залишилося занадто мало. Те ж вірно і для процесів. Процес, який в даний момент часу не вказує користувачу ніякого графічного інтерфейсу, може бути знищений ОС Android на абсолютно законних підставах.

Розглянемо приклад. Нехай наш додаток має екрани A і B. Користувач спочатку відкриває екран A, а потім екран B; з цього моменту екран A став невидимий. Це означає, що екран A і вся логіка, що міститься в ньому, може перебувати в пам'яті, а може і не перебувати. Так, немає гарантій, що об'єкти, пов'язані з екраном A знаходяться в пам'яті, поки видно екран B. Логіка екрану B не повинна зав'язуватися на знаходження об'єктів екрану A в пам'яті. Побічний ефект полягає в тому, що архітектура Android примушує до використання архітектурного стилю «shared nothing». Це означає, що різні частини Android програми можуть викликати один одного і взаємодіяти між собою тільки формально; жоден з них не може звернутися до іншого безпосередньо. А що трапиться, якщо користувач вирішить повернутися на екран A? Напевно, він захоче побачити його в тому ж стані, в якому він залишив його, вірно? Фреймворк Android вирішує цю проблему: для кожного стану життєвого циклу існують методи, кожен з яких може бути перевизначений розробником. Ці методи викликаються фреймворком в заздалегідь певні ключові моменти, наприклад, коли користувальницький інтерфейс показується на екрані, ховається і т.п. У цих методах розробник може реалізувати логіку для зберігання і відновлення стану об'єктів.

Подібна обробка приховування користувальницьких інтерфейсів і існування кнопки «назад» на Android-пристроях призводять до необхідності наявності стека користувальницьких інтерфейсів, в якому поточний видимий інтерфейс поміщається на вершину, а всі інші зсуваються вниз (стекова операція «push»). Подібний стек існує в Android. У документації він називається «activity stack» або іноді «back stack».

У плані обробки взаємодії між інтерфейсом і його логікою Android можливе використання архітектурного шаблону «Model-View-ViewModel» (MVVM – найкраща архітектура GUI-додатків на даний момент).

Архітектура MVVM була створена з метою поділу праці дизайнера і програміста, неможливого, коли Java-розробник намагається побудувати GUI в Swing або розробник на Visual C ++ намагається створити користувальницький інтерфейс в MFC. Розробники мають безліч навичок, але створення зручних і привабливих інтерфейсів вимагає абсолютно інших талантів, ніж ті, якими володіють розробники. Ця робота більше підходить для дизайнерів інтерфейсів. Хороші дизайнери інтерфейсів краще знають, чого хочуть користувачі, ніж експерти в галузі проектування та написання коду. Зрозуміло, буде краще, якщо дизайнер інтерфейсів створить інтерфейс, а розробник напише код, який реалізує логіку цього інтерфейсу, але технології типу Swing або MFC не дозволяють діяти таким чином.

Архітектура MVVM вирішує цю проблему ясним поділом відповідальності:

- розробка користувальницького інтерфейсу здійснюється дизайнером інтерфейсів за допомогою технології, більш-менш природної для такої роботи (XML);

- логіка користувача інтерфейсу реалізується розробником як компонент ViewModel;
- функціональні зв'язки між інтерфейсом і ViewModel реалізуються через Біндінг (bindings), які є правилами типу «якщо кнопка А була натиснута, повинен бути викликаний метод onButtonAClick () з ViewModel». Біндінг можуть бути написані в коді або визначені декларативним шляхом (Android використовує обидва типи).

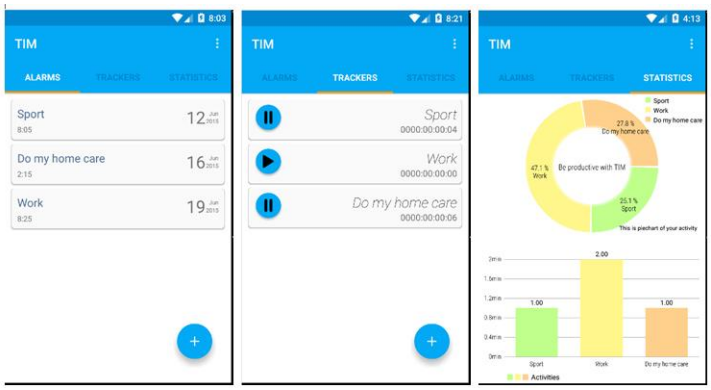


Рисунок 4 – Екранні форми (AlarmsFragment.java – Екранна форма будильників, TrackersFragment.java – Екранна форма таймерів, StatisticsFragment.java – Екранна форма графічного представлення)

Архітектура MVVM використовується в тому чи іншому вигляді усіма сучасними технологіями, наприклад, Microsoft WPF і Silverlight, Oracle JavaFX, Adobe Flex, AJAX.

Структура інтерфейсної частини даної АІС (рис. 4) як сукупність засобів для обробки та відображення інформації, максимально пристосованих для зручності користувача, складається із програмної назви сторінки та опису сторінки. У графічних системах інтерфейс користувача реалізовується багатовіконним режимом, змінами кольору, розміру, видимості вікон, їхнім розташуванням, сортуванням елементів вікон, доступністю багатокористувацьких налаштувань.

Функції додатку представлені на діаграмі варіантів використання (рис.5).

Рисунок 5 – Діаграма варіантів використання та статистка діяльності

Після додавання декількох таймерів можна відстежувати їх роботу (рис.5). Користувач може скористуватися таймерами для відстеження діяльності без використання будильника. Будильник у даному додатку розроблений лише для нагадування про ту роботу, яку потрібно відстежити.

Як було описано вище, операційна система Android використовує компоненти BroadcastReceivers, Service, Intent для обміну повідомленнями між системою.

Це мабуть було б і все, якби з додатку не виходили, і пристрій ніколи не перезавантажувався і був завжди ввімкнений. Для вирішення цих проблем у маніфесті додат-

ку треба вказати дозвіл відстежувати системні повідомлення, які розсилає система, а саме потрібна інформація перезавантаження девайсу. При виході з додатку та вказавши потрібний дозвіл, програмне забезпечення підписано на повідомлення перезавантаження. Після отримання повідомлення перезавантаження запускається сервіс, який перевіряє статус будильників та встановлює їх в початковий рівень з обчисленою різницею часу.

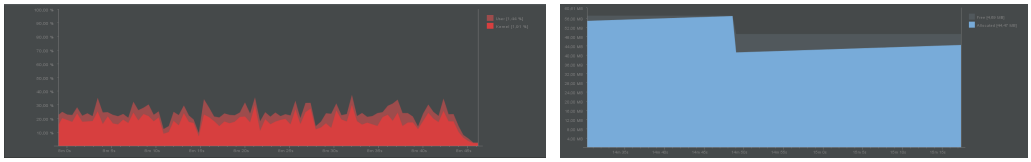
У випадку, коли девайс вимкнений, для роботи будильника потрібний ще один дозвіл на пробудження телефону зі сну on/off режиму (навіть при розрядці батареї).

Опираючись на механізм потоків Java, робота кожного таймера виноситься в окремий потік Java(Thread). Android звичайно надає свої засоби для вирішення таких проблем, як винесення важкої роботи в інші потоки, це AsyncTask, Loader, IntentSevice та ін. Але жоден з них не призначався з ціллю постійної роботи протягом кількох годин або днів, якщо користувач не зупинив додаток (пауза у роботі).

AsyncTask виконує роботу в іншому потоці і має дуже зручний механізм доступу до головного потоку, але більше призначений для відносно неважких операцій (завантаження зображення з сервера та іншої діяльності, яка має логічний кінець). Поганою практикою вважається мультизапуск AsyncTask. Loader не можна використовувати ідеологічно, бо він лише конфліктував би з іншими Loader'ами, які використовуються для завантаження таймерів у список на екрані. IntentService – окремий потік – теж не найкраще рішення, бо багато IntentSevice в системі спровокує Android звільнити ресурси та повбивати екземпляри інтент-сервісів, та ж проблема і з AsyncTask. Жодному з цих компонентів не можна задати пріоритет виконання. Thread підходить для ролі потоку, який буде працювати невідомий проміжок часу, до перезавантаження системи. Є можливим запуск багатьох екземплярів класу Thread.

Було зроблено тестування розробленого додатку в звичайних та екстремальних ситуаціях (рис.6, 7):

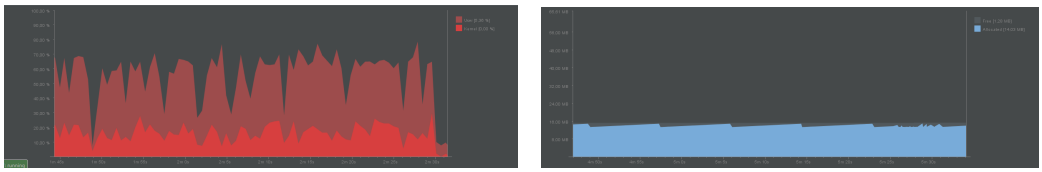
- навантаження на процесор смартфона з 50 сторонніми потоками та постійною прокруткою списку ListView з максимальним API 22 та з мінімальним API 17;
- об'єм оперативної пам'яті, яка використана під час виконання 50-ма потоками роботи на відлік часу API 22 та API 17.



а) б)

а) навантаження процесора з 50 запущеними сторонніми потоками та прокрученням списку;
б) об'єм оперативної пам'яті, яка використана 50-ма потоками

Рисунок 6 – API 22



а) б)

а) навантаження процесора з 50 запущеними сторонніми потоками та прокрученням списку;
б) об'єм оперативної пам'яті, яка використана 50-ма потоками

Рисунок 7 – API 17

Висновки. Проведено аналіз новітніх технологій та розглянуто проблеми їх використання у межах предметної області розробки програмного забезпечення додатку «Тайм-менеджер» до операційної системи Android. За результатами проведених досліджень виконано аналіз структури та методів побудови програмних додатків для операційної системи Android.

За допомогою Intellij IDEA та Android Studio створено програмний додаток, який містить у собі наступний функціонал: механізм будильника для нагадувань про діяльність; механізм відстеження часу та таймерів з вказівкою типу обмеження; графічний механізм відображення витраченого часу.

При створенні програмного продукту вирішено проблеми перезавантаження пристрою, спрацювання таймеру під час сну телефону, відновлення роботи будильників після випадкового перезавантаження, багатопоточного запуску таймерів, обробки зміни орієнтації екрану, роботи у фоновому режимі.

ЛІТЕРАТУРА

1. Sovelluksen Valmistaja Create an elegantly designed Reminder/Alarm clock application [Электронный ресурс] / Learn Android Development | Download Free Apps. – 2014. – Режим доступу: http://www.appsrox.com/android/tutorials/remindme/.
2. Отзывчивое Android-приложение или 1001 способ загрузить картинку [Электронный ресурс] / Блог компании EastBanc Technologies, Разработка под Android. – 2013. – Режим доступу: http://habrahabr.ru/company/eastbanctech/blog/ 192998.
3. Сухоруков И. Многопоточность в Java / Программирование, Параллельное программирование, JAVA. – 2012. – Режим доступу: http://habrahabr.ru/post/164487.
4. Material Design: на Луну и обратно [Электронный ресурс] / Блог компании REDMADROBOT. – 2015. – Режим доступу: http://habrahabr.ru/company/redmadrobot/blog/252773/.

Надійшла до редколегії 26.04.2016.

004.9:004.912

ДРАНИШНИКОВ Л.В., д.т.н., професор
ШКУРКО О.А., магістр

Дніпродзержинський державний технічний університет

АНАЛІЗ ТА ВИЛУЧЕННЯ ІНФОРМАЦІЇ З ТЕКСТІВ

Вступ. Комп’ютерна лінгвістика (також відоме поняття "обробка природної мови" від дослівного перекладу з англійської naturallanguageprocessing) – це надзвичайно важлива область комп’ютерних наук яка ґрунтується на знаннях інформатики, лінгвістики і, все частіше, на теорії ймовірності та статистики.

На високому рівні абстракції комп’ютерна лінгвістика використовує комп’ютери при обробці людської (природної) мови (рис.1). З одного боку цієї проблеми ми маємо те, що часто називають розумінням природної мови, де ми беремо текст в якості вхідних даних для комп’ютера, а потім він обробляє текст і робить щось корисне. З іншого боку, ми маємо те, що часто називається naturallanguagegeneration, де комп’ютер, у деякому сенсі, виробляє мову у спілкуванні з людиною (користувачем системи).

Одна з найстаріших програм і проблема великої важливості – це машинний переклад. Це проблема зіставлення речень однією мовою з реченнями іншою мовою. І це дуже-дуже складне завдання. Помітний швидкий прогрес в останні 10-20 років у цій області. Наприклад, подивимось на результат перекладу фрази, яка багатьом знайома, Googletranslate з української на англійську. Хоча переклад далекий від ідеалу, ви все одно зможете зрозуміти багато з того, що було сказано в оригіналі.