УДК: 004.451.45, 004.451.8

М. І. Ковалик, Р. М. Камінський
Національний університет “Львівська політехніка”,
кафедра інформаційних систем та мереж

ОСОБЛИВОСТІ ВЗАЄМОДІЇ КОМПОНЕНТІВ У МОБІЛЬНІЙ ПЛАТФОРМІ ANDROID

Описано головні способи взаємодії між сервісами та активностями у системі Android. Описано переваги і недоліки кожного з підходів, а також ситуації, коли конкретний підхід найоптимальніший.

Ключові слова: активність, інтент, багатоканальний приймач, міжпроцесна взаємодія.

The main ways of interaction between Services and Activities in Android are described in the article. Advantages and disadvantages of each approach are described. The situations where a particular approach is most appropriate are dealt with.

Key words: Service, Activity, Intent, BroadcastReceiver, Interprocess Communication.

Вступ. Загальна постановка проблеми

Стрімкий розвиток телекомунікацій зумовлений, з одного боку, значними успіхами в нанотехнології, а з іншого, можливостями, які дістають визнання користувачів. Забезпечити широкі можливості засобів телекомунікацій вдається шляхом використання високопродуктивних систем, серед яких найбільшу популярність має платформа Android. Широка розповсюдженість цієї систем зумовлена низкою особливостей, зокрема: відкритість вихідних кодів, гнучкість та надійність вищевказаної операційної системи. Хоча платформа Android є порівняно новою платформою, однак вона вже встигла завоювати значне коло прихильників та користувачів, проте для розробників ця платформа характеризується недостатньою кількістю досліджень у галузі загальноприйнятих термінів та технік, які описують правильну організацію взаємодії компонентів у цій системі. Ця проблема є актуальною, оскільки містить систематизацію та узагальнення вже існуючих досліджень у цій галузі, характеристику кожної з технік взаємодії, а також виділяє переваги і недоліки відповідного варіанта взаємодії.

Аналіз останніх досліджень та публікацій

Будь-яка програма у Android – це система взаємопов’язаних компонентів, таких як активностей (англ. *Activity*), сервісів (англ. *Service*), провайдерів даних (англ. *Content Provider*) та багатоканальних приймачів (англ. *Broadcast Receiver*). Для коректної співпраці цих компонентів часто необхідно здійснювати обмін даними або результатами виконання між ними. Між активностями і сервісами взаємодія є найбільш трудомісткою, оскільки кожен з них має свій власний «життєвий» цикл і може діяти як самостійний компонент незалежно один від одного. Схематично взаємодію компонентів зображено на Рис. 1. Для фонової роботи у системі Android найбільш розповсюдженими варіантами є використання потоків (англ. *Thread*), асинхронних завдань (англ. *AsyncTask*) та сервісів. Сервіс, на відміну від потоку і асинхронного завдання, є самостійним компонентом, може виконуватись навіть після завершення основної програми та може працювати і в тому самому процесі, що й основна програма (локально), і в окремому. Для кожного варіанта існує відповідна система внутрішньо-процесної або міжпроцесної роботи.

Активність (англ. *Activity*) – це візуальна компонента системи Android, яка являє собою одиничний екран програми і відповідає за пряму взаємодію з користувачем. Програма може складатись з декількох активностей[1].

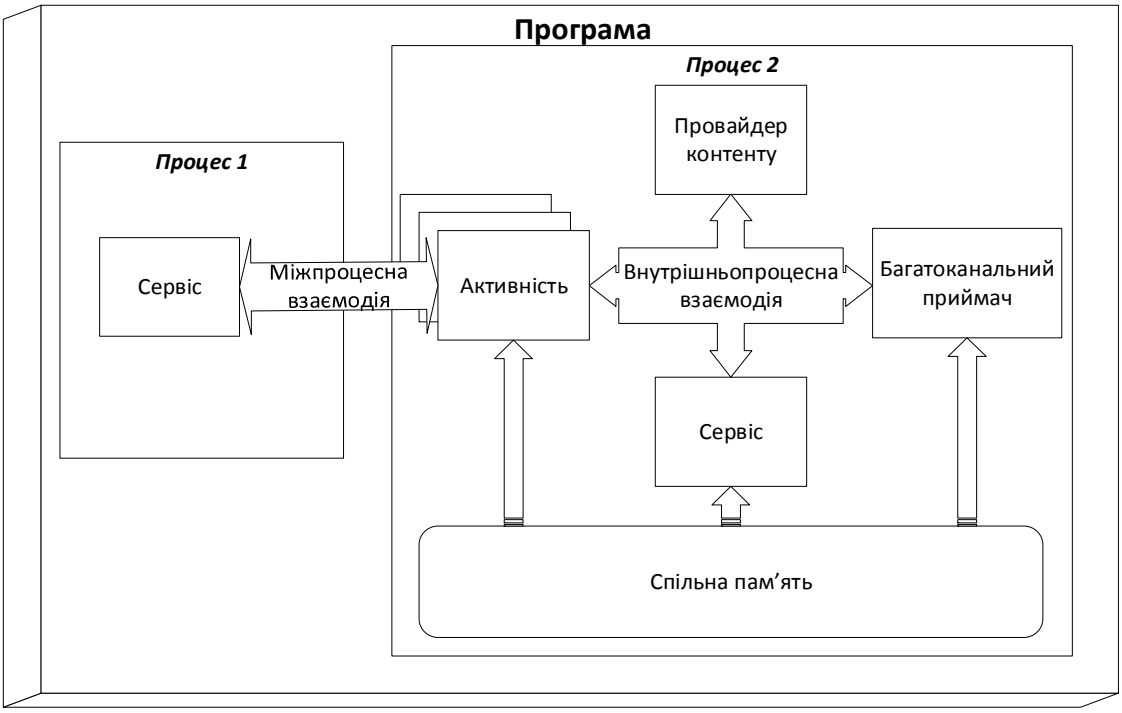

Рис. 1. Схематичне зображення взаємодії компонентів

Сервіс (англ. *Service)* – це компонента системи Android, що виконується у фоні та не має прямої взаємодії з користувачем[2]. Як правило, призначена для виконання будь-якої довготривалої роботи[3]. Сервіс може бути у двох можливих станах:

- **Запущений** (англ. *Started*). Сервіс може бути у даній формі, якщо запуск відбувається за допомогою методу *startService().* У цьому стані сервіс перебуватиме доти, доки не буде явно завершений за допомогою методів *stopService()*або *stopSelf().*
- **Прив'язаний** (англ. *Bound*). Сервіс знаходиться у цьому стані, якщо викликається метод *bindService()* та існує доти, доки він зв’язаний хоча б з одним компонентом системи.

Кожна програма у системі Android запускається і працює у окремому процесі. Для одного процесу є так звана спільна пам’ять (англ. *Shared memory*), яка використовується як загальний спосіб для передавання інформації між потоками (рис. 2).

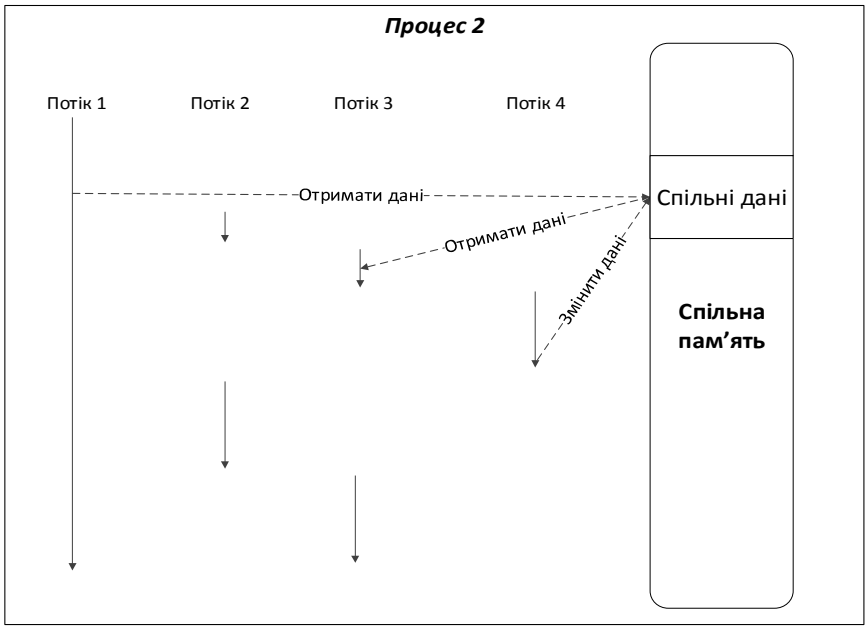

Рис. 2. Схематичне зображення взаємодії потоків за допомогою спільної пам'яті

Всі потоки у програмі можуть мати доступ до одного адресного простору у межах одного процесу. Потік може зберігати локальні змінні, які доступні лише йому, і жоден інший потік не має доступу них. У спільній пам'яті зберігаються члени-екземпляри, члени класу та об'єкти, оголошені у методах. Посилання на об'єкти зберігаються локально у стеку (англ. *stack*) потоку, але самі об'єкти розміщуються у спільній пам'яті. Об'єкти доступні з багатьох потоків лише тоді, коли посилання на об'єкти оголошені поза межами методів потоку.

Для внутрішньопроцесної роботи обмін даними та інформацією здійснюється через спільну пам'ять процесу і тому не потрібно додаткових операцій для організації роботи. Міжпроцесна робота у системі Android реалізовується за допомогою механізму байндер (англ. *Binder*), який керує обміном даними, коли немає спільної пам'яті.

Найпоширеніші випадки міжпроцесної взаємодії керуються високорівневими компонентами у Android, такими, як інтент (англ. *Intent*), системні сервіси та контент провайдери. Вони можуть бути використані у програмі, навіть, не знаючи чи взаємодіє програма всередині процесу, чи поза процесом. Інколи для програми необхідно визначити більш явну модель взаємодії і глибше взаємодіяти у комунікації [4]. Для цього у системі Android використовується спеціальний механізм віддалених викликів процедур.

Формулювання мети

Метою роботи є проведення порівняльного аналізу найпоширеніших варіантів кооперації між основними компонентами системи Android.

Аналіз отриманих наукових результатів

Для міжпроцесної та внутрішньопроцесної роботи існують такі можливі механізми взаємодії між активністю та сервісом: інтент, багатоканальний приймач, прив'язка активності до локального сервісу, обробник і месенджер та AIDL для міжпроцесної взаємодії.

Інтент (Intent)

Інтент – це абстрактний опис операції або дії, який може бути використаний для передавання даних між компонентами. Цей варіант призначений для непрямої однонапрямленої взаємодії з сервісом. Всі дані, які необхідні для роботи сервісу, передаються через інтент під час запуску. Для запуску сервісу створюється інтент та за допомогою методу *putExtra()* додаються дані, які будуть використані у сервісі. Для запущеного сервісу отримати вхідні дані можна у методі *onStartCommand (Intent intent, int flags, int startId)* за допомогою *getExtra()*. Для прив'язаного сервісу цей механізм не працює, оскільки під час прив'язування метод *onBind(Intent intent)* викликається лише один раз. Більше того, механізм прив'язування на відміну від механізму запуску, не призначений для багаторазового виклику, а орієнтований на пряму роботу з екземпляром інтерфейсу *IBinder*.

Недоліком цього підходу є неможливість прямої двосторонньої взаємодії. Перевагою є простота реалізації і непотрібність створення будь-яких додаткових класів та об'єктів.

Найоптимальнішим варіантом використання є реалізація подібних операцій\механізмів з різними початковими параметрами як, наприклад, завантаження даних. Вхідними параметрами у такому випадку виступатимуть: *посилання*, по якому відбувається завантаження, та *назва файла* для збереження на пристрої пам'яті. Цей підхід працює і для локального сервісу, і для сервісу, що виконується у окремому процесі.

Багатоканальний приймач

Багатоканальний приймач (англ. *BroadcastReceiver*) – компонент системи, що дозволяє опрацьовувати повідомлення або дії, передані з інших компонентів системи. Він дуже часто використовується для опрацювання повідомлень або дій безпосередньо від системи Android, таких, як повідомлення про низький рівень батареї, загасання екрана тощо[5].

Цей механізм є доволі простим у реалізації і водночас гнучким та потужним.

Для реалізації цього підходу використовується така послідовність дій:

1. Для компонента, якому необхідно передати дані або повідомлення, оголошується клас, похідний від класу BroadcastReceiver.

2. У похідному класі реалізується абстрактний метод *onReceive (Context context, Intent intent),* в якому безпосередньо здійснюються дії при отриманні інтенту.

3. Визначити дії, на які повинен реагувати приймач, тобто створити **фільтр інтентів** (англ. *IntentFilter*) і вказати його під час реєстрації приймача у методі *registerReceiver (BroadcastReceiver receiver, IntentFilter filter).* Для активності найпоширенішим місцем для реєстрації та скасування багатоканального приймача є методи відповідно *onResume()* та *onPause().*

4. У компоненті, з якого необхідно передати дані, створюється інтент з відповідною дією, яка вказана у фільтрі інтентів, і викликається метод *sendBroadcast(Intent intent).*

Варто зауважити, що за відсутності потреби у міжпроцесній взаємодії між компонентами, для роботи з багатоканальним приймачем краще використовувати допоміжний клас *LocalBroadcastManager*. Цей клас призначений для локальної роботи, тобто роботи в процесі програми. Перевагами використання цього класу є:

- ✓ гарантія, що дані, які передаються, не можуть вийти за межі процесу, в якому працює програма і, отже, не існує можливості «витоку» приватних даних;
- ✓ багатоканальний приймач не отримує повідомлення від інших програм, тому не потрібно хвилюватись за «діри» у безпеці програми;
- ✓ багатоканальний приймач є ефективнішим, ніж глобальне передавання повідомлень[6].

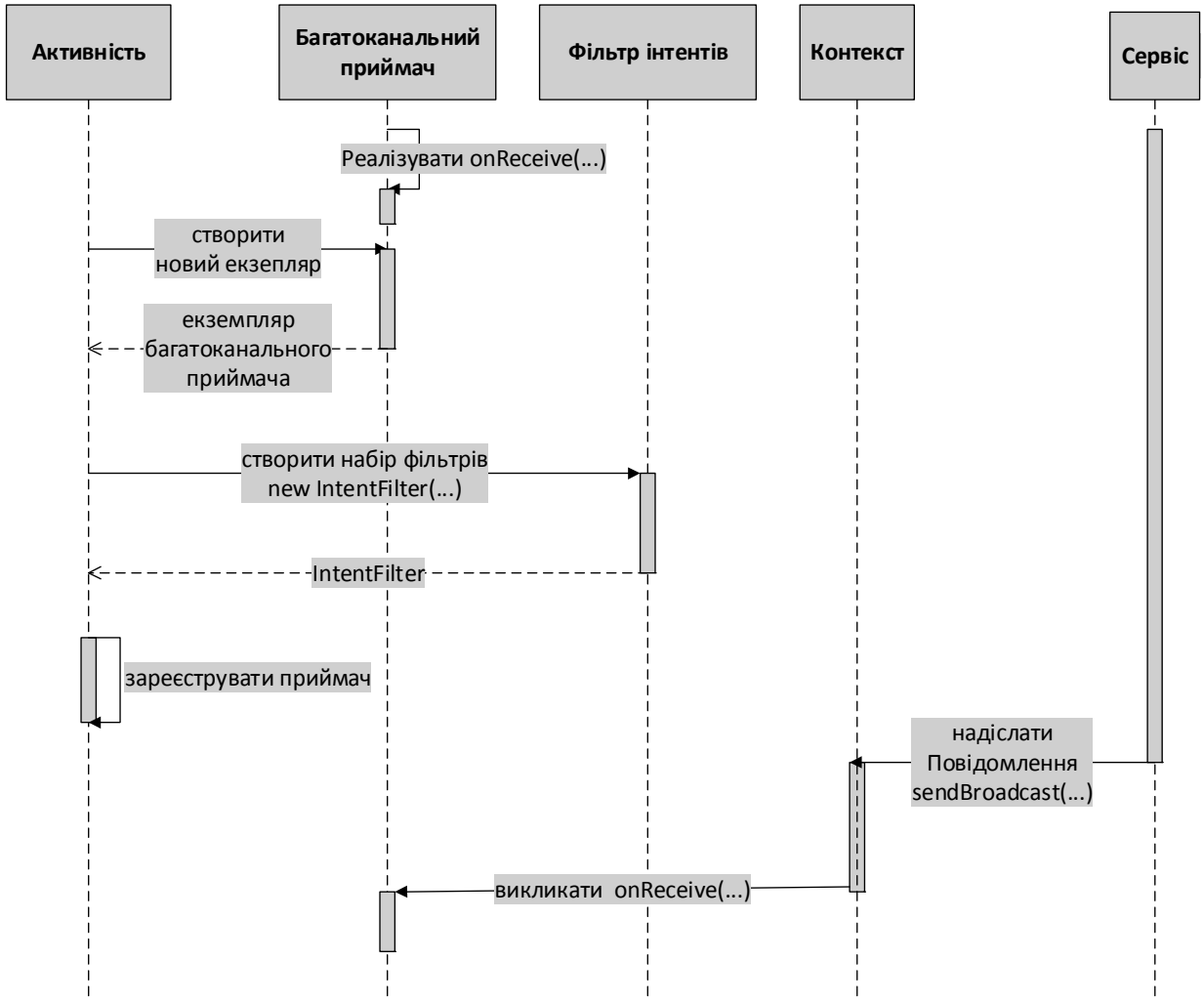

Рис. 3. Послідовність викликів при роботі з багатоканальним приймачем

Найоптимальнішим є використання такого підходу для організації взаємодії «сервіс-активність». Також багатоканальний приймач підтримує можливість міжпроцесної взаємодії.

Прив'язка активності до локального сервісу

Цей механізм доступний лише для локального сервісу. Під час реалізації цього механізму використовується прямий зв'язок з сервісом, тобто після зв'язування з сервісом методом *bindService(Intent intent, ServiceConnection serviceConnection, int flags)* викликається метод *onServiceConnected (ComponentName name, IBinder binder),* в якому відбувається «приведення» *IBinder* до конкретного типу сервісу. Після «приведення», робота відбувається безпосередньо з конкретним екземпляром сервісу, звідки і здійснюються всі виклики конкретних методів. Екземпляр класу повертається у методі *onBind(Intent intent)* Для формування двостороннього зв'язку, у сервісі можливо створити інтерфейс для передавання результатів виконання у інший компонент. У цьому випадку, інтерфейс – це набір методів, які будуть викликані у сервісі. У компоненті, в який передаватимуться дані, необхідно реалізувати відповідний інтерфейс і передати на нього посилання. Це посилання передається або параметром у конкретний метод, або шляхом створення окремого методу для встановлення значення посилання.

Зобразимо схематично взаємодію активності та локального сервісу:

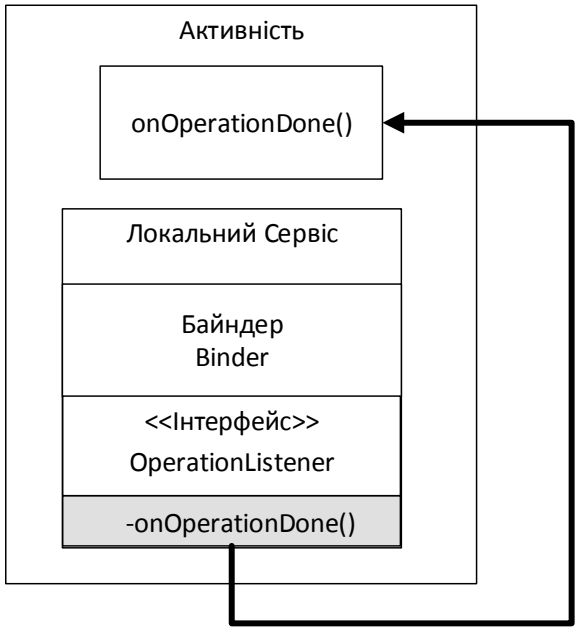

Рис. 4. Схематичне зображення взаємодії сервісу та активності

Варто пам'ятати, що при даному варіанті будь-який довготривалий виклик слід здійснювати у окремому потоці, щоб не блокувати головний потік. Аналогічно діємо у випадку, якщо у інтерфейсі відбувається внесення певних змін до візуальних компонентів системи, їх необхідно здійснювати у головному потоці, тобто викликати метод *runOnUiThread(…).*

Хоча цей підхід є доволі поширеним, але він має низку недоліків:

✓ можливий лише для локальних сервісів;

✓ потребує додаткових витрат на опрацювання міжпотокової роботи;

✓ не є достатньо гнучким, оскільки при зв'язуванні з сервісом здійснюється приведення до конкретного типу, що змушує працювати лише з цим типом сервісу. І для здійснення будь-яких змін потрібно змінювати кожне «місце дотику» до сервісу. Інакше кажучи, з'являється ще одна залежність від конкретного типу класу.

Обробник та месенджер

Цей механізм дає змогу і односторонньої, і реверсивної комунікації. Ґрунтується на роботі двох основних елементів:

- *Обробник* (англ. *Handler*) – елемент, що здійснює опрацювання повідомлень.
- *Месенджер* (англ. *Messenger*) – елемент, за допомогою якого відбувається передавання даних.

Для втілення цього механізму у компоненті, що прийматиме результати від сервісу, створюється обробник і реалізовується його метод *handleMessage(Message message, int what).* Оскільки прямого передавання його у сервіс не існує, тому створюється об'єкт, що міститиме обробник, і який можна було б помістити у екземпляр інтенту.Таким об'єктом є клас месенджер. Він реалізує інтерфейс *Parcelable*, тобто його можна помістити у інтенту та запустити сервіс. Месенджер створюється на основі обробника передаванням його у конструктор. Отже, месенджер виступає в ролі своєрідної «обгортки» для класу обробника. Для передавання результатів у об'єкті класу *Messenger* викликається метод *send(Message).*

Створення двосторонньої взаємодії, як правило, реалізується для прив'язаного сервісу. Для цього сервісу аналогічно створюється обробник, «обгортається» його месенджер і у методі *onBind*(…) повертається *Messenger.getBinder().* Після опрацювання повідомлень у компоненті *onServiceConnected(…)* і отримання месенджера, вони надсилаються вже у сервіс.

Перевагами цього механізму є:

- ✓ можливість організації двосторонньої кооперації;
- ✓ простота реалізації міжпотокової взаємодії;
- ✓ міжпроцесна робота.

Послідовність роботи з сервісом за допомогою обробника зображено на Рис. 5.

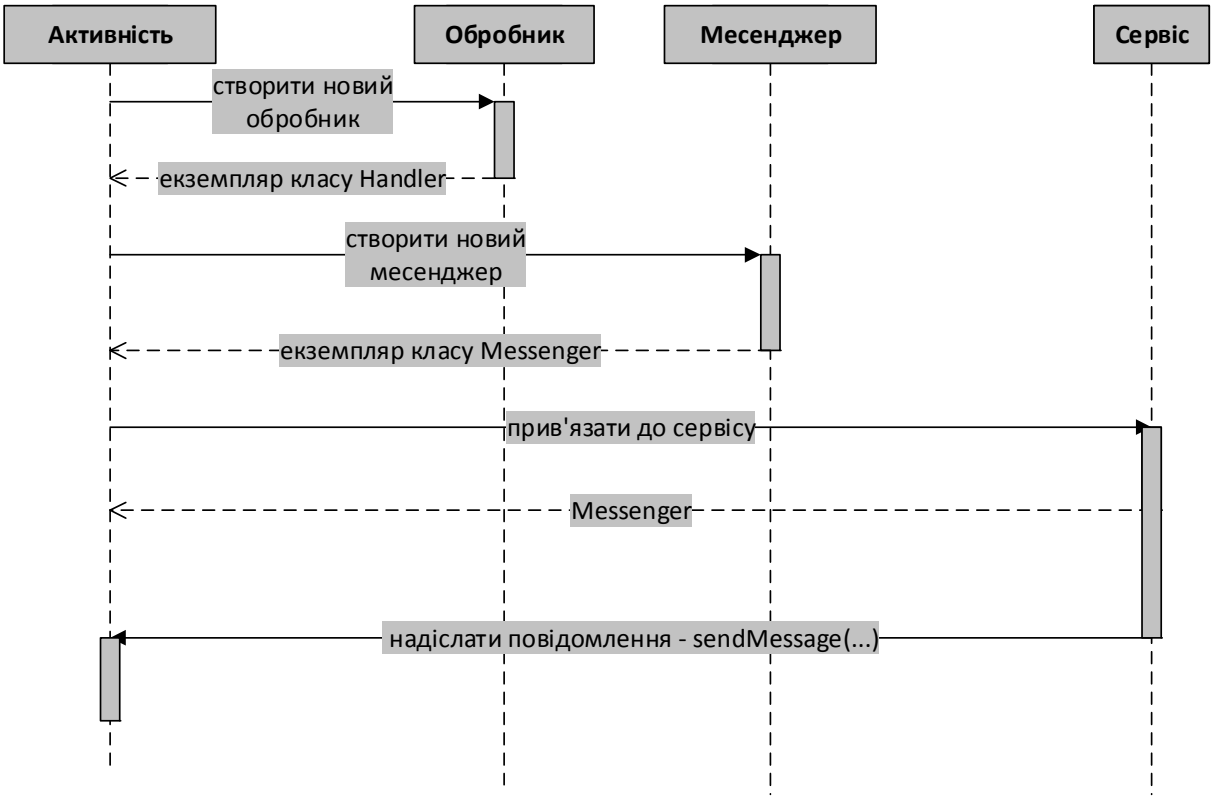

Рис. 5. Послідовність роботи з обробником

AIDL для міжпроцесної кооперації

Система Android використовує механізм байндер (англ. Binder) для міжпроцесної взаємодії. Цей механізм дозволяє здійснювати віддалені виклики процедури (remote procedure calls - RPC). Це означає, що можливо здійснювати виклики методів з інших процесів так, ніби вони знаходяться у поточному. Для здійснення таких викликів необхідно оголосити інтерфейс, тобто сигнатуру методів (тип вхідних\вихідних параметрів, тип взаємодії і т.д.). Для опису інтерфейсів використовується Android Interface Definition Language (AIDL). Інтерфейси оголошуються у файлах з розширенням *.aidl і під час компіляції перетворюються у згенеровані Java-інтерфейси. Як видно з нижче-наведеної схеми (Рис. 6), згенерований Java файл передається у програму клієнта та у програму сервера для уніфікації системи взаємодії і запобіганню конфліктів.

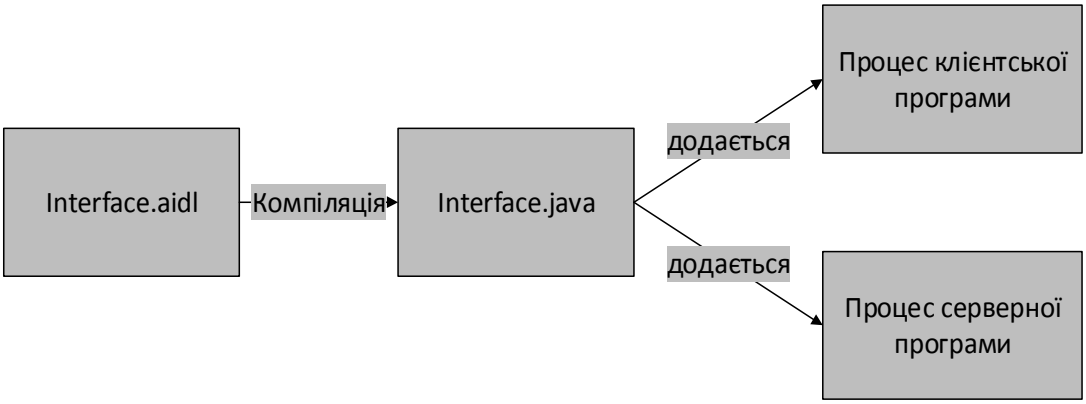

Рис. 6. Схема інтерфейсу віддаленої взаємодії

Згенерований інтерфейс визначає допоміжні класи – Proxy та Stub, що відповідають за маршалізацію та демаршалізацію даних і методів. Маршалізація – це процедура перетворення даних з оригінального формату у послідовність байтів для передавання її між процесами. Демаршалізація – зворотна процедура перетворення з лінійної впорядкованої послідовності байтів у природне Java-представлення. Тобто, процедури маршалізації і демаршалізації використовуються для простого та швидкого обміну даними між процесами і можливі лише для типів, що реалізовують інтерфейс *android.os.Parcelable*.

За замовчуванням всі виклики є синхронними, тобто після виклику методу з поточного процесу, його потік блокується доти, поки метод не завершиться або доки не повернуться результати роботи методу у іншому процесі. Асинхронний виклик реалізовується за допомогою двох окремих oneway-інтерфейсів. Ключове слово *oneway* означає, що потік після виклику не буде очікувати результат, а продовжить виконуватись. У такий спосіб інтерфейс клієнта призначений для початку роботи (запуску методів), а інтерфейс сервера призначений для надсилання результатів або повідомлень про завершення роботи. Варто пам'ятати, що синхронні виклики блокують викликаючу програму на час виконання методу і тому треба всі виклики реалізовувати у фоновому потоці. Переваги цього механізму такі:

- ✓ надає механізм для об’єктно-орієнтованої міжпроцесної взаємодії;
- ✓ є строго типізованим механізмом і створює менше помилок під час роботи;
- ✓ надає можливість і синхронного, і асинхронного викликів;

Висновки і перспективи подальших наукових розвідок

Система Android надає доволі широкий спектр для організації взаємодії між активностями та сервісами залежно від поставленого завдання. Взаємодія може бути і міжпроцесною, і внутрішньопроцесною. У загальному випадку потреба у міжпроцесній взаємодії з’являється рідко і

лише у доволі великих проектах. Більшість міжпроцесної взаємодії у програмі опрацьовується високорівневими компонентами. Якщо потрібно, є можливість переходу на низькорівневу роботу за допомогою RPC та месенджера. Перевага надається RPC, коли необхідно збільшити продуктивність через опрацювання вхідних запитів одночасно. У разі, коли такої потреби немає – доцільніше і простіше використовувати месенджер.

1. Activities[Electronic Resource]:[site]// Official documentation. – Mode of access : URL: http://developer.android.com/guide/components/activities.html. – Title from the screen. – Last access:10.09.2014. 2. Android Services – Tutorial[Electronic Resource]:[site]. – Mode of access : URL: http://www.vogella.com/tutorials/AndroidServices/article.html#service_overview. – Title from the screen. – Last access:10.09.2014. 3. Services[Electronic Resource]:[site]// Official documentation. – Mode of access : URL: http://developer.android.com/guide/components/services.html. – Title from the screen. – Last access:10.09.2014. 4. Göransson Anders. Efficient Android Threading / Anders Göransson. – Sebastopol:O'Reilly Media, 2014. – 260 p. – ISBN: 978-1-449-36413-7. 5. Application Fundamentals[Electronic Resource]:[site]// Official documentation. – Mode of access : URL: http://developer.android.com/guide/components/fundamentals.html. – Title from the screen. – Last access:10.09.2014. 6. LocalBroadcastManager[Electronic Resource]:[site]// Official documentation. – Mode of access: URL:http://developer.android.com/reference/android/support/v4/content/LocalBroadcast Manager.html. – Title from the screen. – Last access:10.09.2014. 7. Activity and Service communication[Electronic Resource]:[videolecture]// Pattern-Oriented Software Architectures: Programming Mobile Services for Android Handheld Systems. – Mode of access: URL: https://class.coursera.org/posa-002/lecture/159.- Title from the screen. – Last access:13.09.2014. 8. Using LocalBroadcastManager in Service to Activity Communications[Electronic Resource]:[site]. – Mode of access: URL: http://www.intertech.com/Blog/using-localbroadcastmanager-in-service-to-activity-communications/ – Title from the screen. – Last access:13.09.2014.