Smarter Mobile Apps through Integrated Natural Language Processing Services

Bahar Sateli[1], Gina Cook[2], and René Witte[1]

[1] Semantic Software Lab, Department of Computer Science and Software Engineering, Concordia University, Montréal, Canada
[2] iLanguage Lab, Montréal, Canada

Abstract. Smartphones are fast becoming ever-present personal assistants. Third-party 'apps' provide users with nearly unlimited customization options. A large amount of content read on these devices is text based – such as emails, web pages, or documents. Natural Language Processing (NLP) can help to make apps smarter, by automatically analyzing the meaning of content and taking appropriate actions on behalf of their users. However, due to its complexity, NLP has yet to find widespread adoption in smartphone or tablet applications. We present a novel way of integrating NLP into Android applications. It is based on a library that can be integrated into any app, allowing it to execute remote NLP pipelines (e.g., for information extraction, summarization, or question-answering) through web service calls. Enabling a separation of concerns, our architecture makes it possible for smartphone developers to make use of any NLP pipeline that has been developed by a language engineer. We demonstrate the applicability of these ideas with our open source Android library, based on the Semantic Assistants framework, and a prototype application 'iForgotWho' that detects names, numbers and organizations in user content and automatically enters them into the contact book.

1 Introduction

The hand-held devices market has never been a quiescent one. With fierce competition among big mobile companies and rapid advancements in the hardware industry, customers are now in possession of mobile devices with relatively large amount of memory and processing resources that are powerful enough for a wide range of tasks, from social networking to document editing and sharing. With fast, ubiquitous Internet connections, smartphone users have access to an ever growing amount of information available in web pages or email boxes. Unfortunately, they are still left alone to deal with this information overload on their usually small-screen devices. While state-of-the-art techniques from the Natural Language Processing (NLP) domain have proven to be useful in dealing with such situations, it is not yet feasible to deploy a complete NLP solution on smartphones, mainly due to its resource-intensive nature. This major limitation prompts the need for novel approaches that can bring NLP power to smartphones, taking into account their limited input and processing capabilities.

Towards this end, efforts have been made to develop various applications for mobile platforms, such as Google's Android[3] or Apple's iOS,[4] which target specific productivity issues, like retrieving information about places or events with question-answering (QA) applications like Siri[5] or context-sensitive information extraction (IE) like Google Now.[6] However, this means that users must install and use different apps for their various information needs and the market is still lacking a generic NLP app that can offer several text processing capabilities, such as summarization or entity detection, in a unified manner. Preferably, while offering various sophisticated NLP techniques, such an app should be simple enough so that customers, as well as other app developers with no technical NLP background, can make use of its capabilities.

In this paper, we present the first open source NLP library for the Android platform that allows various applications to benefit from arbitrary NLP services through a comprehensive, service-oriented architecture. This work is significant since we offer an application-independent software layer for Android that can be integrated into any existing app in need of NLP support, rather than enhancing a single application. Our approach introduces a novel Human-AI collaboration pattern that can be leveraged to aid mobile users with information-intensive tasks across various domains, such as health care, law, engineering, e-learning, e-business, among others [1–3].

2 Background

In this section, we briefly explain the conceptual and technical foundations for our work, namely, natural language processing and the web service-based Semantic Assistants framework with its NLP API.

2.1 Natural Language Processing (NLP)

The history of Natural Language Processing (NLP) goes far back a few decades. One of the ultimate goals of NLP is to derive intelligence from unstructured content expressed in a natural language, such as English or German, using a variety of techniques from the Artificial Intelligence and Computational Linguistics domains. *Text Mining* is an immensely popular application of NLP that aims at extracting patterns and structured information from textual content. Due to its importance, many frameworks have been developed to facilitate the development of text mining applications, such as the open source *General Architecture for Text Engineering* (GATE) [4], designed both for language experts who need to implement concrete NLP pipelines, as well as software developers seeking to embed NLP capabilities in their applications.

[3] Google Android, http://www.android.com/
[4] Apple iOS, http://www.apple.com/ios/
[5] Siri, http://www.apple.com/ios/siri/
[6] Google Now, http://www.google.com/landing/now/

Mobile Applications of NLP. In what follows, we present a number of standard NLP tasks, with a focus on those relevant for mobile applications. However, this list is by no means exhaustive; many other tasks exist, in particular, for domain-specific contexts (e.g., e-health, e-learning, or e-business).

Automatic Summarization. In the presence of mass information, there is a need to allocate the attention of the target efficiently among the overabundance of information sources [5]. Consider a mobile user that quickly needs to get the main content out of a number of emails, documents, or web pages. Browsing through large amounts of textual content on a small-screen device is not only tedious and time-consuming, but important information can be easily overlooked. In such a situation, where a user's information need is dispersed over a single long or multiple documents, NLP techniques can provide him with a *summary*, a compressed version of the original document(s) that preserves the main information as good as possible. While *generic summaries* are probably the most widely known, other summary types are even more interesting for mobile applications: *Focused summaries* start from a topic (such as a question stated by the user) and generate the summary targeted at this question. *Update summaries* take the user's reading history into account, summarizing only the new information that was not seen before [6].

Information Extraction (IE) is one of the most popular applications of NLP. IE identifies instances of a particular class of events, entities or relationships in a natural language text and creates a structured representation of the discovered information. A number of systems and APIs have been developed for this task, such as OpenCalais,[7] which can extract named entities like *Persons* and *Organizations* within a text and present them in a structured language, e.g., XML or RDF.[8] These techniques are also helpful for mobile applications, where users often deal with large amounts of textual content. For example, using an IE service in a mobile app can help users to automatically find all the occurrences of a specific type of entity, such as 'company', and gather complementary information in form of metadata around them.

Content Development. NLP techniques can also be helpful when developing new content: Typing (or dictating) longer texts on a mobile device is typically cumbersome and time-consuming. Here, NLP services such as summarization or IE can support users by generating part of the new content. For example, when replying to an email in response to a question, a focused summary can provide the main body of information that needs only editing and verification by a user.

Question Answering (QA) has become one of the most prominent applications of NLP on mobile devices, in particular due to Apple's Siri app. In contrast to focused summaries, which also address an information need of a user, but are typically answering open-ended questions in an essay-style (e.g., *"What is the importance of refrigeration for ice cream deliveries?"*), QA aims at providing answers to factual, closed questions (e.g., *"Where do I find the nearest ice cream*

[7] OpenCalais, http://www.opencalais.com/

[8] Resource Description Framework (RDF), http://www.w3.org/RDF/

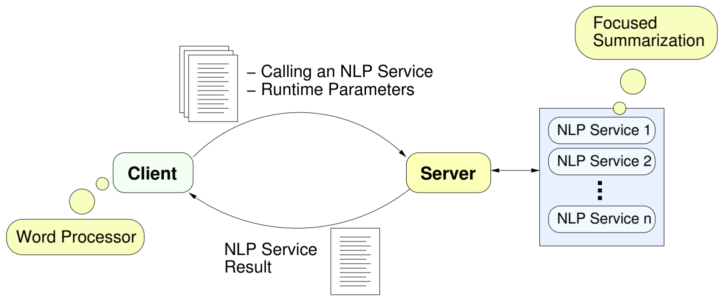


Fig. 1. The Semantic Assistants service execution workflow [7]

parlor?"). QA is a significant improvement to simple (keyword-based) information retrieval, which simply presents a list of possibly relevant documents to the user, because it allows users to pose questions against its knowledge base in natural language. Then, after "understanding" the question, an answer formulation step brings back the extracted information to the user. While systems like Siri apply to a general domain, QA systems on mobile devices have much broader potential: For example, a company might want to develop an app that can answer questions about its products or services; or a university could provide students and staff with an app that is capable of answering questions about its courses or buildings.

2.2 The Semantic Assistants Framework

As we described in the previous section, NLP pipelines are diverse in nature and in some scenarios, more than one NLP technique might be useful in respect to the user's task at hand. The Semantic Assistants framework [7] is an existing open source architecture that provides a unified manner of offering NLP capabilities to clients, ranging from desktop applications to web information systems, in form of W3C standard web services.[9] The goal of the Semantic Assistants architecture is to wrap concrete analysis pipelines, developed based on existing NLP frameworks, and broker them to connected clients through a service-oriented architecture.

Corresponding to the definition of service-oriented architectures, the Semantic Assistants architecture has a repository of NLP pipelines that are formally described using the Web Ontology Language (OWL).[10] Such an infrastructure allows for dynamic discovery of NLP services in the architecture and reasoning capabilities over pipelines before recommending them to clients. For example, based on the user's context or the language of source content, Semantic Assistants can recommend a subset of NLP services to the user. This way, any NLP service deployed in the Semantic Assistants repository becomes available to all connected clients through a WSDL[11] interface description.

[9] Web Services Architecture, http://www.w3.org/TR/ws-arch/
[10] Web Ontology Language (OWL), http://www.w3.org/2004/OWL/
[11] Web Services Description Language (WSDL), http://www.w3.org/TR/wsdl

As shown in Fig. 1, an NLP execution workflow is initiated by the client through sending textual content (e.g., the URL of a document or literal text) to a Semantic Assistants server. Optionally, clients can customize the pipelines' behaviour by setting a number of runtime parameters, e.g., output formats or special thresholds. These requests trigger the execution of an actual NLP pipeline on the provided content, and if successful, return the results to the client in form of a unified XML document. The receiving client is then responsible for parsing the results and providing the user with a proper presentation, for instance, highlighting annotations in a text or opening up a new document.

In order to facilitate the integration of new clients, the Semantic Assistants architecture offers a Client-Side Abstraction Layer (CSAL) library. Essentially, the CSAL library is a Java archive of common communication and data transformation functionalities that can be reused by different clients. Such an abstraction layer eases the implementation of the client-server communication process in new clients, as well as the transformation of NLP results to other useful data types.

What distinguishes the Semantic Assistants framework from other approaches is its emphasis on a *separation of concerns*: Users who interact with the NLP pipelines are not concerned with how the analysis is performed on their provided content; Application developers can use the CSAL library to easily connect to the back-end server and invoke services and retrieve the results; and Language engineers can develop sophisticated NLP pipelines without worrying about how they are going to be offered to the end user. Currently, the Semantic Assistants architecture supports NLP pipelines developed based on the GATE[12] framework, but also provides for the execution of pipelines based on OpenNLP[13] and UIMA[14] through their GATE integration.

3 Design of the Semantic Assistants Android NLP Layer

Following the description of how different NLP techniques can aid mobile users and the Semantic Assistants framework that provides such capabilities to its connected clients, we now describe a high-level design overview of our novel Android-NLP integration process. The ultimate goal we are trying to achieve in this integration is to provide a software layer to the Android platform that can be used by various applications in need of NLP support.

Our research hypothesis is that numerous applications running on a smartphone can benefit from natural language processing support: interactions with the user will become faster through automation; tasks that are currently difficult to perform on small-screen devices will be significantly improved.

Robust, open source implementations for tasks like information extraction or automatic summarization are readily available, in the form of NLP pipelines running within a text analysis framework. However, these frameworks cannot be executed on an Android device, as NLP is very resource-intensive and mobile

[12] GATE, http://gate.ac.uk/
[13] OpenNLP, http://opennlp.sourceforge.net
[14] UIMA, https://uima.apache.org/

devices have both hard- and software limitations [8]. Furthermore, rather than just enhancing a single application with NLP capabilities, we aim to offer an application-independent software layer that can be integrated into any existing application requiring NLP support. Similarly, we do not want to restrict the type and number of possible NLP analysis services in advance: rather, it should be possible to dynamically add new services and have them discovered by the mobile device, i.e., we aim to design a service-oriented architecture (SOA). The vision is that applications on a mobile device can dynamically request NLP services, such as entity extraction, summarization [9, 10], or question-answering [11], to support the user in knowledge-intensive tasks, just like a human assistant would.

3.1 Requirements

We start by defining a set of requirements for our integration.

Remote Execution of Services (R1). In order to tackle the limited memory and processing capacities of mobile devices, the integration must execute the NLP services on a remote machine and bring the results back to the mobile device.

NLP Service Independence (R2). Mobile users deal with various types of information, ranging from news articles to personal documents, for which generic or domain-specific NLP pipelines may be useful. Irrespective of how the NLP pipelines are concretely implemented, the integration must offer them within a single unique interface.

App Independence (R3). The integration must be realized in such a way that various apps can seamlessly benefit from NLP services, rather than enhancing a single app only.

Flexible Response Handling (R4). Depending on the application area, NLP pipelines can generate various output formats, e.g., annotations for IE or new documents for summarization. The integration must accordingly represent and transform the NLP pipeline results in a mobile device.

Inter-App Communication (R5). Various apps must be able to share content with the integration layer for analysis and receive the results, where applicable. It should also be possible to receive content from one app and write the results to another.

3.2 Developed Solution

Here, we describe our design decisions to meet the requirements enumerated in the previous section. Our goal is to design an architecture that allows Android apps to connect to a remote Semantic Assistants server and benefit from its NLP services, either directly within their environment or through another *service app*, i.e., an application that performs the invocation process and returns the results to the app that originally requested the NLP service.

Fig. 2 illustrates a high-level overview of our integration architecture, where the client is always an Android app requesting an NLP service, and the server side is

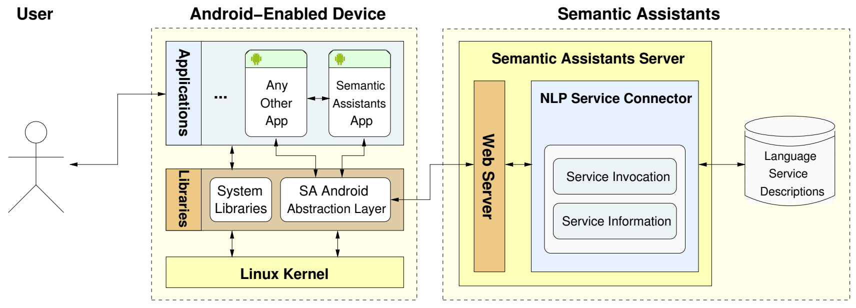


Fig. 2. The Semantic Assistants-Android integration high-level architecture

the Semantic Assistants server brokering these pipelines as Web services (R1). The client-server communication is facilitated through an Android-specific client-side abstraction layer, designed as an extension to the Semantic Assistants framework. Because of the absence of Simple Object Access Protocol (SOAP) [12] libraries in Android, both components in our architecture have to communicate in a RESTful way. To provide a RESTful communication over the HTTP protocol, we extended the Semantic Assistants server tier with a REpresentational State Transfer (REST) [13] interface that additionally provides a secure HTTPS channel.

The Semantic Assistants Android abstraction layer encompasses common functionalities for Android apps to communicate with the Semantic Assistants server by creating RESTful requests for service inquiry, execution, user authentication and eventually, parsing the retrieved results. It also contains pre-defined Android *intents*[15] for remote execution of NLP services. In the Android platform, intents are system-wide messages that contain abstract descriptions of operations to be performed. Intents are the Android's platform facility for communication between components within one or multiple applications. The Semantic Assistants Android abstraction layer library has an extensible structure for NLP intents that can be extended as more frequently-used NLP services are made available in the Semantic Assistants repository. For example, the Semantic Assistants Android abstraction layer has a built-in entity detection intent that extracts *Person* and *Location* named entities from a given text and returns the extracted results as series of annotations to the invoking application. We will demonstrate a use case for this intent within a demo app in Section 4.

4 Implementation

Our two novel components in the Android-NLP integration are the *Semantic Assistants App* and the *Semantic Assistants Android Abstraction Layer* library shown in Fig. 2. The Android abstraction layer plays the role of a system-wide library that can be referenced by all Android apps installed on the application

[15] Intents and Intent Filters, http://developer.android.com/guide/components/intents-filters.html

layer of the Android OS architecture. The Semantic Assistants App[16] is an implementation of a general-purpose NLP app that can offer various NLP services to users both within its user interface, as well as other apps using system messages.

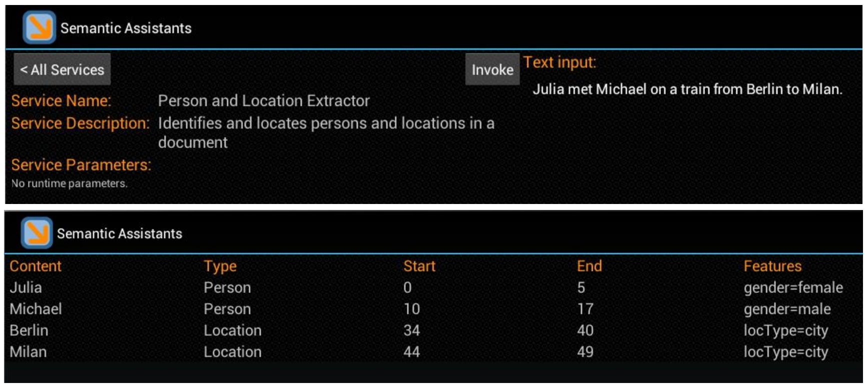
4.1 The Semantic Assistants Android Abstraction Layer

Technically, the Semantic Assistants Android abstraction layer is an Android library[17] project that holds shared code and resources for apps that need NLP support in their workflow (R3). The abstraction layer classes implementing the SA-Android communication and data transformation are compiled by the Android development tools into a Java archive (JAR) file and installed on the system. Thereby, various apps can refer to the Semantic Assistants abstraction layer as a *library* dependency in their code. Using the library classes, other apps can directly connect to a given Semantic Assistants server interface and invoke the NLP services available in the server's repository without the hassle of implementing connection handlers to a remote server. The library also helps developers in preparing requests conforming to the Semantic Assistants server endpoint description file and parsing the resulting XML document. The current implementation of the Semantic Assistants Android library supports NLP service inquiry, invocation, and user authentication on the Semantic Assistants server over HTTP and HTTPS protocols.

4.2 The Semantic Assistants App

As a part of our contribution and in order to demonstrate a general-purpose app offering arbitrary NLP services to Android mobile users, we have implemented an Android app, called the *Semantic Assistants App*, that offers a unique user interface to inquire and invoke NLP services on a user-provided content. The Semantic Assistants App's main activity allows users to authenticate themselves and configure the app to connect to a specific Semantic Assistants server endpoint. Once the user settings are stored, the Semantic Assistants App inquires about the available assistants from the server and generates a dynamic list containing service names (R2), their descriptions and possible further configuration options, such as applicable runtime parameters, as shown in the top part of Fig. 3. When the user selects a service from the list, the Semantic Assistants App features a text field for the user to enter content for analysis. For example, in our screenshot taken from a tablet running the Semantic Assistants App, we have provided a short sentence for a sample analysis scenario and chosen the "Person and Location Extractor" as the NLP service. Pressing the "Invoke" button sends a request to the remote Semantic Assistants server through the Semantic Assistants Android abstraction library, which in turn triggers the execution of the actual pipeline on the server. The Person and Location entities found in the text are first received by the Semantic Assistants Android abstraction library in form a typical Semantic

[16] Semantic Assistants App is available at http://www.semanticsoftware.info/sa-android
[17] Android Library Project, http://developer.android.com/tools/projects/index.html



Service Name: Person and Location Extractor
Service Description: Identifies and locates persons and locations in a document
Service Parameters:
No runtime parameters.
Text input:
Julia met Michael on a train from Berlin to Milan.

Content	Type	Start	End	Features
Julia	Person	0	5	gender=female
Michael	Person	10	17	gender=male
Berlin	Location	34	40	locType=city
Milan	Location	44	49	locType=city

Fig. 3. The Semantic Assistants App Service User Interface

Assistants XML document. The library then parses the embedded entities to an array of Java objects and sends them back to the Semantic Assistants App, where a list is automatically populated from the entities' literal content, their semantic types, their exact character offsets in the original text, as well as any additional features provided by the pipeline (Fig. 3, bottom).

Of course, manually pasting content into the Semantic Assistants App's text field is still not convenient for most mobile users, especially on small screens. That's why the Semantic Assistants App also *listens* to the system's *sharing* intents, broadcasted from other existing apps on the device (R5). Sharing intents are fired in the system whenever an app provides the user with the ability to share content directly with another app, e.g., sending a paragraph from a web page to an email application as a newly composed message. Because the Semantic Assistants App registers itself in the Android system as a listener component for sharing intents, its name pops up in the contextual menu of sharing actions where applicable. Hence, users can choose the Semantic Assistants App as the content receiver application and thereby automatically populate the text field of the app, preparing it for a new NLP analysis session. Based on the NLP pipeline's result format, the Semantic Assistants App can eventually present the extracted annotations in a list format, or open up the generated output file in the Android's configured default browser (R4).

Listening to system-wide sharing intents provides Semantic Assistants App users with a convenient method to invoke arbitrary NLP pipelines using the app's graphical user interface. However, another important target user group of our integration are Android app developers who need to embed NLP capabilities in their own application. In order to suppress the need to interact with the Semantic Assistants App's GUI for each analysis session, the Semantic Assistants App is designed to offer its "*Semantic Assistants Open Intents*" to all applications, once it has been installed on a device. The idea behind open intents is to allow other apps to invoke specific NLP services in a headless manner by sending a broadcast message across the system asking for a specific intent. If the intent is

```
<service
  android:name="info.semanticsoftware.semassist.android.service.
      SemanticAssistantsService"
  android:process=":semassist_service"
  android:label="semassist">
    <intent-filter  android:label="Semantic Assistants Open Intents">
      <action android:name="org.openintents.action.PERSON_LOCATION_EXTRACTOR" />
      <category android:name="android.intent.category.DEFAULT" />
    </intent-filter>
</service>
```

Fig. 4. A Semantic Assistants open intent example

recognized by the Semantic Assistants App, it is received and transformed into a corresponding NLP service execution request. Subsequently, the results of a successful execution are returned to the app that sent the broadcast message. Fig. 4 shows an example Semantic Assistants open intent to extract person and location named entities from a given text.

5 Application

Following a thorough description of the Android-NLP integration, in this section we explain how Android app developers can now embed NLP capabilities in their own apps using our novel architecture. We begin this section by an example scenario, in which an app developer wants to write a *smart* app for his client. Our developer's client frequently travels to different conferences and meets new people. He tries to remember and keep in touch with his new connections by creating contact entries in his Android phone. However, every once in a while, he goes back to his contact book, though by just reading a name he doesn't remember in which context he met that specific person. So he asks our developer to create an app for him that can automatically extract useful metadata and related context information from the emails his connections send him and create new contacts or add notes to his contact book entries.

In a typical Android development process, our developer starts by designing an Android app that receives shared content from an email app on the device. Once he receives the message content in his app, he has to apply various heuristics, such as regular expressions or string matching, on the text to find useful entities for his client. Such an approach is computationally expensive and does not usually yield an acceptable precision or recall when implemented by someone unfamiliar with the text mining domain. Therefore, after a short research, our developer finds a *Semantic Assistants open intent* that can extract various entities, such as person, organization, location and address entities, from a given text. The developer then embeds the Semantic Assistants Android abstraction library in his project and delegates the entity extraction responsibility to the Semantic Assistants App via a simple Java method call, provided by the abstraction library classes. In this case, the processing flow of this new application, called the 'iForgotWho' app, follows the sequence shown in Fig. 5.

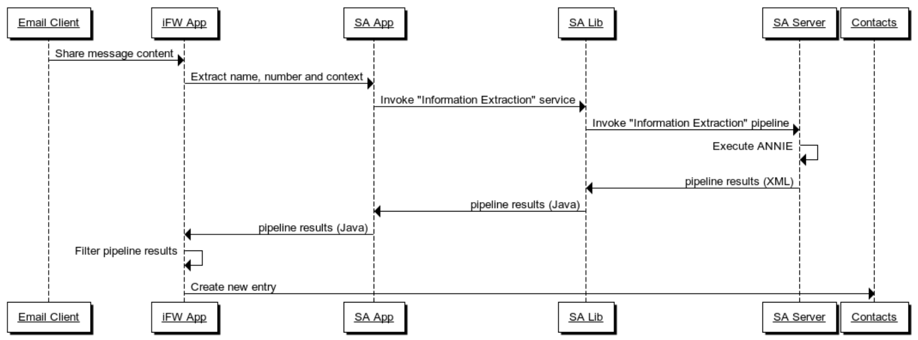


Fig. 5. The iFW app workflow sequence diagram

```
 1 <!-- Transparent activity bound to the system share intent -->
 2 <activity android:name=".activity.ContactFinderActivity" android:theme="@style/Theme.
       Transparent">
 3         <intent-filter  android:label ="iForgotWho">
 4                 <action android:name="android.intent.action.SEND" />
 5                 <category android:name="android.intent.category.DEFAULT" />
 6                 <data android:mimeType="text/*"/>
 7         </intent-filter>
 8 </activity>
 9
10 <!-- Broadcast receiver for the Semantic Assistants App messages -->
11 <receiver android:name=".services.NotifBroadcastReceiver">
12         <intent-filter>
13                 <action android:name="info.semanticsoftware.semassist.android.BROADCAST"/
                       >
14         </intent-filter>
15 </receiver>
```

Fig. 6. The iForgotWho app manifest file snippet

To better demonstrate this use case, we implemented the iForgotWho (iFW) Android app and used its NLP capability on an example email message. The iFW app's manifest file[18] (Fig. 6) denotes two important features about this app: The iFW app accepts textual content shared from other apps (lines 1–8) and listens to broadcast messages sent from the Semantic Assistants App (lines 10–15).

The iFW app contains only two activity classes[19] – one transparent activity that delegates user requests to the Semantic Assistants Android library and another one that lists the extracted entities and asks for the user's permission before adding them to his contact book. Fig. 7 contains the complete source code of the iFW's service-requesting activity, showing how the complicated task of entity detection can be simply achieved through using a Semantic Assistants open intent.

[18] The AndroidManifest.xml File, http://developer.android.com/guide/topics/manifest/manifest-intro.html

[19] Activity classes provide user interface components that users interact with.

```
public class ContactFinderActivity extends Activity {
        @Override
        public void onCreate(Bundle savedInstanceState) {
                super.onCreate(savedInstanceState);
                Intent service = new Intent("org.openintents.action.INFORMATION_EXTRACTOR");
                /*The share intent carries the user selected text.
                 * We simply pass the text to the designated Semantic Assistants service. */
                String input = getIntent().getExtras().getString(Intent.EXTRA_TEXT);
                service.putExtra(Intent.EXTRA_TEXT, input);
                /* True silent mode means that iForgotWho app
                 * will take care of the results presentation. */
                service.putExtra(Constants.SILENT_MODE, "true");
                // call the service
                startService(service);
                // close the activity
                finish();
        }
}
```

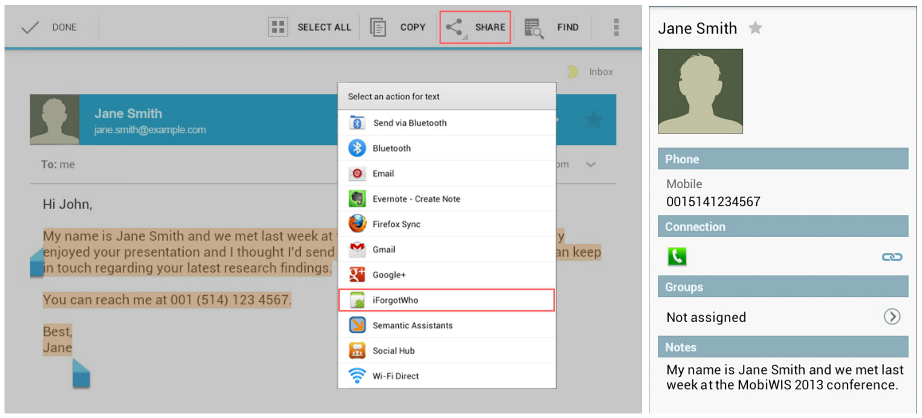
Fig. 7. The iForgotWho activity requesting an open intent

In our example scenario, our developer's client uses the iFW information extraction feature to automatically create a new contact entry from an email message he has received from one of his connections. The client selects the content of his email message and shares it with the iFW app. The iFW app then calls the `INFORMATION_EXTRACTOR` service of the Semantic Assistants App to extract useful information, such as the person's name or phone number from the email message, through executing GATE's ANNIE pipeline [4] on the server side. Upon receiving the results, iFW prompts the user with the detected entities and lets our client verify the results. Once approved by the user, iFW automatically creates a new contact entry and populates the corresponding fields, such as the contact's name, phone number and a sentence from the (email) message containing a named entity, like the sender's name, as contextual information to help our user remember the contact. Fig. 8 shows the sharing process and the contact entry that is eventually created by the iFW app in the user's contact book.

As we showed in our iFW app, our Android-NLP integration architecture provides the possibility for developers to enhance their applications' capabilities with state-of-the-art techniques from the natural language processing domain, implementing novel use cases of human-AI collaboration patterns within the context of mobile devices at the convenience of calling open intents.

6 Related Work

Using natural language processing techniques in the context of mobile devices has recently gained attention, both among academic and industry communities. The lavish launch of Apple's Siri as a voice-controlled personal assistant was a major demonstration of how state-of-the-art NLP techniques can aid mobile users with productivity tasks, like finding a nearby restaurant. This has motivated other frontier mobile development companies to increase their efforts in adopting



Jane Smith
jane.smith@example.com

To: me

Hi John,

My name is Jane Smith and we met last week at … y
enjoyed your presentation and I thought I'd send … an keep
in touch regarding your latest research findings.

You can reach me at 001 (514) 123 4567.

Best,
Jane

Jane Smith

Mobile
0015141234567

Not assigned

My name is Jane Smith and we met last week at the MobiWIS 2013 conference.

Fig. 8. Sharing content with the iFW app (left) and the resulting contact entry (right)

NLP techniques for mobile devices. Google Now integrates additional techniques, such as personalization, on top of NLP. However, many of these applications are still largely limited to "speak to command" use cases, which merely provides a natural language interface for users to issue spoken, intuitive commands to a mobile device for tasks like question-answering or sending text messages in a hands-free manner [14]. Moving beyond speech recognition, the apparent need for other more complex tasks requiring sophisticated NLP analysis is being satisfied through various apps for information-intensive areas like personalized news recommendation [15] or web page summarization [9]. Unlike these rapidly emerging task-specific apps, our mobile-NLP integration aims at providing a general NLP solution that can offer various techniques through a unified interface. On the foundations of the Semantic Assistants' service-oriented architecture, a multitude of generic and domain-specific NLP pipelines can be deployed and utilized within Android apps. In addition, in contrast to [16] and [17], where authors try to adapt resource-intensive NLP techniques like information extraction and machine translation to mobile contexts, our integration architecture offers the full power of NLP applications to mobile users.

Alongside the efforts to develop NLP-enabled apps that target specific productivity issues in mobile contexts, others are providing more general solutions to embed NLP capabilities within mobile apps by providing software APIs. AlchemyAPI[20] offers an Android SDK for app developers to make use of its various text mining techniques, such as keyword extraction or concept tagging, within their apps through embedding a Java archive file that makes remote web service calls to the AlchemyAPI text mining platform. Maluuba[21] is another NLP API for mobile apps that offers two separate interfaces for "interpreting" spoken commands and "normalizing" textual content for Android and Windows Phone devices. In

[20] AlchemyAPI, http://www.alchemyapi.com
[21] Maluuba, http://www.maluuba.com/

contrast, our approach not only provides a unique interface to offer various NLP techniques, but also provides ready-to-use NLP *intents*. This is an advantage over merely publishing an API that first needs to be understood by developers and then employed in their apps. Moreover, unlike these commercial offerings, both the Semantic Assistants framework and our Semantic Assistants-Android integration are open source software[22] that allow the developers to deploy their own custom NLP pipelines (e.g., for healthcare, biomedical research, e-learning, or entertainment) and extend our integration's open intents, as their need arises. Finally, by providing a clear separation of concerns, mobile developers with no NLP background can now make use of a multitude of existing open source pipelines, in particular those developed based on the popular GATE framework, as well as custom pipelines that language engineers will develop without the concern on how these pipelines are going to be integrated in mobile apps.

7 Conclusion

For many users, smartphones have become their go-to-device for industry news feeds, checking email, and scheduling their agenda. The sheer quantity of text which is read on these devices, and the number of screen-taps needed to get things done, could be significantly reduced by applying existing Natural Language Processing (NLP) pipelines, such as automatic summarization or information extraction, to news feeds, emails, or attachments. Summarized text can also be consumed in an eyes-free manner using the smartphone's Text-To-Speech capabilities. Dates, locations and people can be automatically detected using named entity recognition and integrated in the creation of new events in a user's agenda and entries in the contact book as we demonstrated with the iFW app. We developed the first open source API for Android that provides an easy way to integrate NLP capabilities into an application. It relies on interactions familiar to mobile application developers, without requiring any background in computational linguistics. With our framework, application developers can easily integrate complex text mining tasks into a smartphone application, either based on existing open source NLP tools, or by delegating the pipeline development to a language engineer. We believe that the resulting architecture has significant potentials to make mobile apps 'smarter' across a wide range of domains.

References

1. Aanensen, D.M., Huntley, D.M., Feil, E.J., al Own, F., Spratt, B.G.: EpiCollect: Linking Smartphones to Web Applications for Epidemiology, Ecology and Community Data Collection. PLoS ONE **4**(9) (09 2009) e6968
2. Doukas, C., Pliakas, T., Maglogiannis, I.: Mobile healthcare information management utilizing Cloud Computing and Android OS. In: Engineering in Medicine and Biology Society (EMBC), 2010 Annual International Conference of the IEEE. (Sept. 4 2010) 1037–1040

[22] Available at http://sourceforge.net/projects/semantic-assist/

3. Kamel Boulos, M.N., Wheeler, S., Tavares, C., Jones, R.: How smartphones are changing the face of mobile and participatory healthcare: an overview, with example from eCAALYX. Biomedical Engineering Online **10**(1) (2011) 1–24
4. Cunningham et al., H.: Text Processing with GATE (Version 6). University of Sheffield, Deptartment of Computer Science (2011)
5. Simon, H.A.: Designing organizations for an information rich world. In Greenberger, M., ed.: Computers, communications, and the public interest. The Johns Hopkins Press (1971) 37–72
6. Witte, R., Bergler, S.: Next-Generation Summarization: Contrastive, Focused, and Update Summaries. In: International Conference on Recent Advances in Natural Language Processing (RANLP 2007), Borovets, Bulgaria (September 27–29 2007)
7. Witte, R., Gitzinger, T.: Semantic Assistants – User-Centric Natural Language Processing Services for Desktop Clients. In: 3rd Asian Semantic Web Conference (ASWC 2008). Volume 5367 of LNCS., Springer (2008) 360–374
8. Park, S.Y., Byun, J., Rim, H.C., Lee, D.G., Lim, H.: Natural language-based user interface for mobile devices with limited resources. Consumer Electronics, IEEE Transactions on **56**(4) (november 2010) 2086–2092
9. Alam, H., Hartono, R., Kumar, A., Rahman, F., Tarnikova, Y., Wilcox, C.: Web Page Summarization for Handheld Devices: A Natural Language Approach. 7th International Conference on Document Analysis and Recognition (ICDAR'03) **2** (2003) pp.1153
10. Buyukkokten, O., Garcia-Molina, H., Paepcke, A.: Seeing the whole in parts: text summarization for web browsing on handheld devices. In: Proceedings of the 10th international conference on World Wide Web. WWW '01, New York, NY, USA, ACM (2001) 652–662
11. Jilani, A.: Mobile Phone Text Processing and Question-Answering. Future Technologies in Computing and Engineering: Proceedings of Computing and Engineering Annual Researchers' Conference 2010: CEARC10 (2010) 130–135
12. Box, D., Ehnebuske, D., Kakivaya, G., Layman, A., Mendelsohn, N., Nielsen, H.F., Thatte, S., Winer, D.: Simple Object Access Protocol (SOAP) 1.1. W3C Note, World Wide Web Consortium (May 2000) See http://www.w3.org/TR/SOAP/.
13. Fielding, R.T.: Architectural styles and the design of network-based software architectures. PhD thesis (2000)
14. Zhou, L., Shaikh, M., Zhang, D.: Natural Language Interface to Mobile Devices. In Shi, Z., He, Q., eds.: Intelligent Information Processing II. Volume 163 of IFIP International Federation for Information Processing. Springer US (2005) 283–286
15. Tavakolifard, M., Gulla, J.A., Almeroth, K., Ingvaldsen, J.E., Nygreen, G., Berg, E.: Tailored News in the Palm of Your HAND: A Multi-Perspective Transparent Approach to News Recommendation. In: Proceedings of the 22nd International World Wide Web Conference. WWW '13, Rio de Janeiro, Brazil (May 13–17 2013)
16. Seon, C.N., Kim, H., Seo, J.: Information extraction using finite state automata and syllable n-grams in a mobile environment. In: Proceedings of the ACL-08: HLT Workshop on Mobile Language Processing, Columbus, Ohio, Association for Computational Linguistics (June 2008) 13–18
17. Homola, P.: A Distributed Database for Mobile NLP Applications. In: Proceedings of the ACL-08: HLT Workshop on Mobile Language Processing, Columbus, Ohio, Association for Computational Linguistics (June 2008) 27–28