SPLIMBo – Developing and Evaluating a Software Product Line for Cross-Platform IM Bots

Victor Travassos Sarinho
Laboratório de Entretamento Digital Aplicado
Universidade Estadual de Feira de Santana
vsarinho@gmail.com

Abstract— Instant Messaging – IM platforms spread the communication instant among their users in a fast, low cost and multimedia way. This paper presents the development and evaluation process of SPLIMBo, an open source Software Product Line that allows the production of cross-platform IM bots in a "write once, run anywhere" perspective. It is based on a Product Line Architecture that provides adapters to interact with distinct IM platforms, bot sessions defined by IM feature configurations, and a relational database able to integrate IM data from deployed adapters and bot sessions. As evaluation process, the LibrasZap IM game development based on SPLIMBo was described, and a comparative analysis with available bot builders was performed. They confirmed the SPLIMBo ability to build local and configurable bots being available for multiple IM platforms.

Index Terms—Software Product Line, Instant Messaging, Cross-Platform IM Bots, IM Games

I. INTRODUCTION

Nowadays, many types of Instant Messaging – IM platforms, such as WhatsApp [1], Messenger [2] and Telegram [3], are available to provide a high quality instant communication to the final user. They offer important innovations in user interaction, such as real-time communication of multimedia messages and support of IM bot systems. For bots, they are “simply IM accounts operated by software that can do anything, such as teach, play, search, broadcast, remind, connect, integrate with other services, or even pass commands to the Internet of Things – IoT” [4].

Multiple IM APIs are available in a developer perspective [5]. They allow the production of IM systems able to provide automated responses for connected IM clients. Some of these APIs provide exclusive resources for bot development, such as inline keyboards with callback and URL buttons [6]. Others do not offer an official protocol documentation and, in case of WhatsApp, actively discouraged 3rd party implementation of their protocol [7].

Trying to support this IM API diversity, ChatFuel [8], Permabots [9], Flowxxo [10] and Gupshup [11] present cross-platform solutions in the cloud to develop IM bots. ChatFuel provides a bot interface that works as an initial wrapper for IM platforms, making distinct configurations for the same bot system. Permabots works with a limited API that redirects received IM from respective platforms. Flowxxo and Gupshup allow a programmable configuration of cross-platform IM bots, but with a limited number of interactions per month. In a web perspective, Imified [12] proposed a web service to host and run large scale IM bots. Moreover, in a stand-alone perspective, Zhou et al. [13] proposed a local method and system able to specify IM bots using state transitions and XML specifications.

This paper presents the production and evaluation process of SPLIMBo [14], an open source Software Product Line – SPLIMBo able to configure and deploy cross-platform IM bots in a “write once, run anywhere” perspective. To this end, section 2 describes SPLIMBo assets to deploy IM bots. Section 3 presents the configuration process of IM bots. Section 4 describes the obtained result of a deployed game using SPLIMBo and a comparative analysis with available bot builders. Finally, section 5 presents the conclusion and future work of this project.

II. THE SPLIMBO PLATFORM

Per Voelter and Groher [16], “the effectiveness of a SPL approach directly depends on how well feature variability within the portfolio is managed from early analysis to implementation and through maintenance and evolution”. Variability management is “the activity concerned with identifying, designing, implementing, and tracing flexibility in SPLs” [16], whose Product Line Architecture – PLA are “designed to support the variation needed by the products in product lines, and so making it reconfigurable makes sense” [15]. Beuche and Dalgarno [17] defined important Domain and Application steps based on features for SPL Engineering, such as “structure and selection for the solution elements of the product line platform” and the specification of “the needed platform elements (and additional application elements if required)”.

This section presents PLA elements able to be “structured and selected” by feature configurations to control the domain diversity of IM APIs. These PLA elements define the SPLIMBo platform, being responsible of monitoring IM platforms and performing IM actions for hosted bots.

A. Monitoring IM Platforms

There are different types of strategies and resources being used by IM APIs to work with IM data and services, such as: programming languages (Python, Java, Lua, etc.); IM data structures; user identification strategies (phone number, internal ID, Jabber ID); monitoring approaches of incoming messages (event-oriented, loop-oriented); available support for multimedia content; and so on. To integrate this collection of features, an adapter strategy was applied to offer a common set of generic IM
functions able to operate different types of IM APIs.

These generic IM functions are responsible to connect with IM platforms, handle received IMs and monitoring bot responses to be repassed to the respective IM platform. When executed, they follow the necessary steps to get credentials with the associated IM platform, and perform the necessary processing on sent/received IM data, which can be done by standard loop or event-driven approach according to the IM API monitoring strategy.

Fig. 1 illustrates this adaptation life cycle, where the start method connects to an IM platform to get access credentials, the handle method receives new messages from IM clients, and the monitoring method verifies bot responses to send back to the IM client.

![IM Platform](image1)

**IM Platform**
- WhatsApp [1]
- Messenger [2]
- Telegram [3]
- ...

**IM API**
- ChatAPI [7]
- RestFB [18]
- Telebot [19]
- ...

**IM Adapter**
- start()
- handle(IM)
- monitoring()

**Fig. 1. Adaptation life cycle among IM Adapters, IM APIs and IM Platforms.**

A relational buffer was also designed to represent the data bus for transmitted IM between deployed adapters and available platforms. It is basically a Queue table (Fig. 1) composed with the following fields to be used by IM adapters and hosted IM bots:

- **status** – ‘R’ value for received IM and ‘S’ value for IM to be sent;
- **url** – indicates the current IM API for the sent/received message;
- **jidClient** and **jidServer** – identification codes for IM client and bot system;
- **message** – transmitted text during an IM conversation;
- **data and extension** – multimedia content and type of a transmitted IM;
- **dateTime** – indicates when the IM was sent from a bot or received by the adapter;
- **dateTimeToSend** – previous schedule to send an IM.

**B. Performing IM Actions**

The SPLIMBo PLA was designed to host different types of bot configurations able to provide automatic responses for incoming IMs. For this, each bot is configured by ZapML, an XML model derived from bot features that represents automated solutions for IM clients.

ZapML files are loaded by ZapServer, a bot container that decides which bot configuration must be loaded, hosted and performed for each received IM. ZapServer applies a continuous inspection of received IMs in the Queue, evaluating the conditions to create a new bot context and associate it to a new **Session**. For loaded bots, ZapServer verifies the destination IM number/id (**jidServer**) to get an available **Session** that will be responsible to evaluate the received IM.

Fig. 2 illustrates this ZapServer process of verifying received IMs from new clients, hosting new bots, and use available **Session** instance to perform received IMs.

![Fig. 2. Hosting process of new IM bots.](image2)

**Fig. 2. Hosting process of new IM bots.**

With an instantiated **Session** from a bot configuration, the next step is to define a representative context for each first IM received by ZapServer from the same client. For this, Execution [20] is a graph model of JBPIM based on interpretation of the process definition and the chain of command pattern. Each Execution is bounded to a ZapML configuration when a new bot **Session** is started. It allows the Execution instance to interpret a ZapML configuration referring to a current XML node in an IM conversation. For each received IM, Execution stores the current conversation status, performs the current XML node and decides the future XML node that will be performed in the next IM client response.

Fig. 3 illustrates the allocation of **Execution** resources (**Bot Workflow** and **Bot Context**) together with XML parsing, IM adapters and relational tables that model the data bus for transmitted IM. The combination of these structures represents the proposed SPLIMBo PLA able to support ZapML configs for multiple IM platforms.

**III. SPLIMBO CONFIGURATION**

**Textual User Interface – TUI** is a common approach to define human-computer interactions in operational systems and remote text terminals [21]. Based on prompts and menus [21], it is broadly applied in modern IM platforms in conjunction with multimedia and mobile resources.

Feature modeling is a technique for managing commonalities and variabilities within a product line [22]. It is used to capture the results of domain analysis, to facilitate scoping of product lines, and to provide a basis for automated configuration of concrete products [22].

Fig. 4 presents the proposed SPLIMBo feature model able to configure IM bot variants based on TUI interactions. It describes IM bots as a collection of Option features, such as Menu, Sequence, Command or HyperText. Each Menu contains lists of header and footer HyperText features, together with a list of sub-Option features. Sequence features represent loops of Option
features, being able to evaluate conditions when necessary. Command is a direct action that can be performed by a IM bot, such as: execute a database routine; send a web request to a server; or send a HyperText response to the IM client. Command features can be also defined as Prompt features, waiting for IM client responses to perform specific actions. Finally, HyperText features are responsible to send textual and multimedia content to the IM client, such as image, video, audio, document, etc.

![Fig. 3. The SPLIMBo Product Line Architecture.](image)

**A. The ZapML Language**

ZapML is an XML that specifies textual representations of TUI responses for IM bots. It is derived from SPLIMBo feature model, having Menu, Prompt, Command, Sequence, Exec and hypertexts tags as main XML elements.

Each ZapML response to IM clients is based on tag evaluation, which defines the current state of a bot in an IM conversation. For example, when a Menu tag is evaluated, the IM bot shows the menu options and wait for an IM client response. If the IM client chose a sub-Menu option, it executes a new ZapML tag that can be a Command, a Sequence, or even a new Menu. If a back option is selected, it returns to the previous tag content. Finally, if the IM client send an invalid option, the current Menu is executed again showing the same menu options of the current node.

```
<zapp path="../DemoZap/">
<menu includeBackOption="false">
<textOptions/>
<textDescription="Hello World!" keycode="System.order">
<text>Hello World!!!</text>
</textDescription>
</menu>
</zapp>
```

Fig. 5. ZapML configuration of the DemoZap example.

Fig. 5 and 6 describes a ZapML example called DemoZap [20]. It is based on one Menu and two Command tags (the Menu options) able to send “Hello World!!!” and an echo as IM response. The initial Menu removes the return option (includeBackOption = ”false”), shows the “Options:” text, and present the “I” and “2” options according to System.order values. Command submits the “Hello World!!!” text when the option “1” is selected, returning to the initial Menu after its execution. Prompt waits the user message and creates a new local variable called echo with the received text. The confirmationMessage attribute prepares a message with System.echo value and sends back to the IM user.

![Fig. 6. DemoZap execution on Telegram platform.](image)

**B. ZapML Interpretation Process**

The startup actions of the ZapML interpretation process can be resumed in three: 1) instantiate an Execution when an IM conversation is started; 2) add default variables to the Execution context (System.jidServer, System.jidClient, etc); and 3) perform the main Node [20] of the Execution instance.

The main Node refers to a default graph model able
to interpret ZapML tags, that: 1) loads the proposed bot configuration via ZapML Parser; 2) evaluates the initial tag (the currentOption) of the bot configuration; 3) sends an IM based on currentOption and context values (currentMessage, currentExtension, etc); and 4) wait for user response or directly decides the new currentOption to be evaluated by the main Node.

The new currentOption is obtained by the tag reference available in the current Node, which could be the same tag, a parent tag, a sub-Option tag from a Menu, or a next tag from a Sequence. After selecting the new currentOption, the process decides which IM response will be sent, and whether the “ok” event will be trigger or not. The “ok” event restart the evaluation process of the current Node without receiving a new message from the IM client. Fig. 7 illustrates a partial state chart of this Menu tag evaluation process.

IV. RESULTS AND DISCUSSION

Interesting IM bots have been developed in recent years. Nombot [24] is a bot that collects data about the nutrition to simplify the food tracking. @dawebot [25] is a Telegram bot for training students in any subject using multiple choice question quizzes. Teleboyarin [26] is an instant messaging bot for rapid delivery of annotation processes powered by a mechanized labor engine over the Telegram messaging system. Tcarenko et al. [27] presented an IoT-enabled fall detection system with a messenger-notification method over the Telegram messaging system. Art-bots [28] is a Messenger chatbot that interacts with visitors through chat and convey information about the museum artifacts in the form of short stories. Finally, for chatbot platforms, Kar and Haldar [29] introduced a conceptual system design which will aid in building Chatbot systems for IoT.

As a SPLIMBo bot example, LibrasZap [30] is a quiz game that assesses knowledge in Libras, the Brazilian sign language used by deaf (Fig. 8). The game consists of selecting the correct answer among available options in continuous game rounds. Each game round provides one Libras word in a video and four word options to be chosen by the player. If the player selects the correct video word, the player gets one point and goes to the next round. If the player selects a wrong word, the game ends and the player result is compared with the best game results, allowing or not the player to put your name in the hall of fame.

Fig. 8. Telegram, WhatsApp and Messenger versions of LibrasZap game.

Fig. 9. Partial description of LibrasZap configuration.

In a ZapML perspective, Fig. 9 illustrates a partial description of the LibrasZap configuration for game rounds. It shows a combination of Sequence tags that prepare the next game round for each correct answer and control the game play asking for new attempts. Menu and Prompt tags are also applied to perform game interactions, such as: send the video with the Libras word to be assessed, show options to be chosen by the player, and ask textual questions according to the game flow (“Try again (Y/N)?”, for example).

Regarding of available bot builders, the number of tools and services have grown significantly in recent years. In most cases, they are proprietary solutions available in the cloud that offer free services to build and support multi-IM bots (tables I and II). Most of them provides a flexible API to define simple workflows for bot conversations. Some of them also provide support for Javascript programming resources, communication
API for external services, and bot templates to build common bots for specific categories.

TABLE I

<table>
<thead>
<tr>
<th>Project</th>
<th>Host</th>
<th>Open Source</th>
<th>Write Once Run Anywhere</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChatFuel</td>
<td>Cloud</td>
<td>No</td>
<td>No</td>
<td>Active</td>
</tr>
<tr>
<td>Pernambots</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Active</td>
</tr>
<tr>
<td>Floxvolc</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Active</td>
</tr>
<tr>
<td>FuzzyPlay</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Active</td>
</tr>
<tr>
<td>Motion.ai</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Active</td>
</tr>
<tr>
<td>ManyChat</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Active</td>
</tr>
<tr>
<td>Sequel</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Active</td>
</tr>
<tr>
<td>Imifed</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Active</td>
</tr>
<tr>
<td>Zhou et al.</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Active</td>
</tr>
<tr>
<td>SPLIMBo</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Active</td>
</tr>
</tbody>
</table>

TABLE II

<table>
<thead>
<tr>
<th>Project</th>
<th>Cost</th>
<th>Implementation</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChatFuel</td>
<td>Free</td>
<td>Custom API</td>
</tr>
<tr>
<td>Pernambots</td>
<td>Free</td>
<td>Redirector API</td>
</tr>
<tr>
<td>Floxvolc</td>
<td>Free</td>
<td>Javascript + Custom API</td>
</tr>
<tr>
<td>FuzzyPlay</td>
<td>Free</td>
<td>Javascript + Custom API</td>
</tr>
<tr>
<td>Motion.ai</td>
<td>Free</td>
<td>Node.js + Custom API</td>
</tr>
<tr>
<td>ManyChat</td>
<td>Free</td>
<td>Custom API</td>
</tr>
<tr>
<td>Sequel</td>
<td>Free</td>
<td>Bot Templates + Custom API</td>
</tr>
<tr>
<td>Imifed</td>
<td>Free</td>
<td>HTTP/REST API</td>
</tr>
<tr>
<td>Zhou et al.</td>
<td>Patent</td>
<td>XML</td>
</tr>
<tr>
<td>SPLIMBo</td>
<td>Free</td>
<td>XML + Custom Script</td>
</tr>
</tbody>
</table>

In contrast with them, SPLIMBo presents a local and open source solution to deploy multi-IM bots. The idea is to provide a SPL structure that supports the fast configuration of customized IM services for dedicated systems. As a result, simple and advanced workflows can be developed to integrate IM bots and platforms with desired information systems available in a local environment. Moreover, by the possibility of extending open source IM adapters, SPLIMBo can communicate with distinct IM platforms, such as web services and IoT for example, in order to send/receive desired IM content.

V. CONCLUSIONS AND FUTURE WORK

This paper presented SPLIMBo, an open source SPL for cross-platform IM bots. For this, SPLIMBo assets based on cross-platform IM resources were defined and integrated. The interpretation process of received IM and the monitoring process of IM platforms were shown and explained. Finally, the configuration of SPLIMBo bots was described and exemplified by DemoZap and LibrasZap bots.

In a brief comparison with related work [8-14, 31-33], SPLIMBo presents a configurable, local hosting and open source cross-platform SPL solution that works with current resources of IM platforms. It describes a possible solution to the variability problem of IM bots, allowing the configuration of advanced TUI in a “write once run anywhere” perspective by feature-based XML configurations. SPLIMBo also enables the production of dynamic and interoperable bots, by the execution of programming scripts and dedicated routines that share database structures and ZapML configurations in local environments.

Regarding the SPLIMBo usage perspective, there is a recent rise in popularity of messaging bots: chatterbot-like agents with simple, textual interfaces that allow users to access information, make use of services, or provide entertainment through online messaging platforms” [34]. It is a consequence of the bot interface paradigm that “makes use of context, history, and structured conversation elements for input and output in order to provide a conversational user experience while overcoming the limitations of text-only interfaces” [34]. Therefore, there is a trend in the development of “Botplications” [34] in next years, and SPLIMBo can be defined as a candidate to support this development demand in the future.

As identified limitations, SPLIMBo does not offer dedicated resources from specific bot platforms to provide advanced IM interactions, such as in-line buttons and webviews [35]. For IM adapters, the loop-driven monitoring approach improves the server consumption of computational resources available in the local structure, limiting the SPLIMBo scalability to offer a cloud service for bot support. Moreover, the representation of SQL routines in ZapML configurations opens security holes in the current SPLIMBo version for bot configs provided by distinct users.

As future work, new bots will be developed to improve the validation and evolution process of SPLIMBo assets. For game bots, an IM game engine based on SPLIMBo resources is currently in production [36]. Finally, other IM adapters and ZapServer optimizations will be developed to extend the collections of communication platforms and bot interaction possibilities, such as dedicated IM bots and SPLIMBo cloud services for example.

REFERENCES


