JOURNAL OF INFORMATION SCIENCE AND ENGINEERING 34, 1187-1201 (2018)
DOI: 10.6688/JISE.201809_34(5).0005

Modularized and Flow-Based Approach to Chatbot Design and Deployment*

SHANG-PIN MA+ AND CHING-TING HO
Department of Computer Science and Engineering
National Taiwan Ocean University
Keelung, 202 Taiwan
E-mail: albert@ntou.edu.tw+

Chatbots are computer programs designed to chat with users via text or voice through the use of techniques of Web services, data analysis, and artificial intelligence (AI). Currently, the use of chatbot is becoming an important trend in the field of data science. An increasing number of chatbots are being built on social platforms, such as Facebook, LINE, and Slack. This has led to the development of numerous tools and online platforms for the construction of chatbots; however, most of these services do not provide comprehensive support for the visual representation and control of conversational flows, bi-directional Web service integration, or the systematic reuse of conversations. Developing a complex chatbot with external Web services requires the writing of extensive conversation scripts and additional coding. In this paper, we propose a visual, flow-based approach to the construction of chatbots on the Node-RED platform, referred to as FCF (Flow-based Chatbot Framework). This system is based on the newly-devised data format for Webhook, thereby allowing bidirectional service integration for software applications. Five chatbot dialogue patterns and three chatbot application scenarios are provided to be components for the construction of complex chatbot applications.

Keywords: chatbot, node-red, chatbot pattern, chatbot framework, service webhook

1. INTRODUCTION

Chatbots are computer programs designed to chat with users via text or voice. Existing chatbots allow the conversational user interface by using techniques of Web services, data mining and analysis, rudimentary artificial intelligence (AI), and/or the formulation of customized rules. Currently, chatbot is becoming an important trend in the field of data science [1, 2]. An increasing number of chatbots are being built on social platforms, such as Facebook, LINE, and Slack. The fact that smartphone users have begun limiting the number of apps they download and use is hindering the promotion of new apps and promoting the development of chatbots on social platforms. This has led to the development of numerous tools and online platforms aimed at facilitating the construction of chatbots.

Chatbot tools can be divided into three trends: (1) chatbot scripts, such as AIML [3, 4] and RiveScript [5], which provide dialog scripts or rules for the building of chatbots; (2) chatbot toolkits or online platforms, such as RedBot [6], Hubot [7], Cleverscript [8], and Chatfuel [9], which provide a programming framework or an online platform for the

Received September 13, 2017; revised October 23, 2017 & January 3, 2018; accepted January 30, 2018.
Communicated by Don-Lin Yang.
* This research was sponsored by Ministry of Science and Technology in Taiwan under grants MOST 105-2221-E-019-054-MY3.
* Special thanks to Mr. Gavin Chin for his valuable feedback to our research.

construction of chatbots; and (3) AI-based developmental tools, such as API.AI [10] (now named Dialogflow), Wit.ai [11], and LUIS.AI [12], which provide semantic analysis and machine learning capabilities. Unfortunately, these approaches do not provide comprehensive support for the visual representation and control of conversational flows, bi-directional Web service integration, or the systematic reuse of conversations. Integrating a complex chatbot with external Web services requires the writing of extensive conversation scripts and additional coding. In this paper, we propose a visual, flow-based approach to the construction of chatbots, referred to as the FCF (flow-based chatbot framework). We constructed FCF on the Node-RED platform to provide a visual, flow-based environment to facilitate the design and deployment of chatbots. We adopted the newly-devised data format for Webhook, thereby allowing bidirectional service integration for full-fledged conversational applications. Five chatbot dialogue patterns and three chatbot application scenarios are provided to be components for the construction of complex chatbot applications. Two chatbot applications using FCF are also presented to illustrate the feasibility and efficacy of the proposed approach.

The remainder of this paper is organized as follows: Section 2 presents a review of related research; Section 3 outlines the details of the proposed FCF approach; evaluations of the proposed system are presented in Section 4; and conclusions are drawn in the final section.

2. RELATED WORK

AIML (Artificial Intelligence Markup Language) [3, 4] is an XML-based language for the creation of natural language software agents (*i.e.*, chatbots). AIML provides several mechanisms by which to define conversation rules, such as patterns, strings of characters for matching one or more user inputs, and templates specifying the response to a matched pattern. AIML is the first widely used language developed specifically for chatbots. RiveScript [5] is a plain text, line-based scripting language commonly used to build chatbots. RiveScript is very similar to AIML, but superior to AIML on several aspects, such as pattern matching and control of conditions. AIML and RiveScript only provide dialog scripts or rules for the building of chatbots; *i.e.*, they are not flow-based and do not provide facilities to visualize the process.

RedBot [6] is a suite of expansion nodes developed on the Node-RED platform as a chatbot building tool. RedBot is meant to facilitate the construction and/or management of chatbots without the need for extensive programming skills. RedBot facilitates the construction of simple chatbots; however, putting together complex chatbots requires the use of other Node-RED nodes and additional coding. Hubot [7] is an open source chatbot framework maintained by the GitHub team. It uses Node.js as the development language for the creation of ChatOps, which is a collaboration model used to connect people, chatbots, and tools within an automated workflow featuring systems for monitoring and sharing the overall status of projects. Hubot does not support the visualizations required for the efficient design and rendering of chatbot processes. Cleverscript [8] is an online chatbot platform for the construction of chatbots using spreadsheets of conversation rules. Chatfuel [9], one of most popular chatbot tools, allows users to create Facebook or Telegram chatbots without any coding experiences by setting dialog blocks.

Both Cleverscript and Chatfuel support convenient ways for the construction of chatbots, but do not provide facilities to visualize the process.

API.AI [8] is a chatbot platform that supports natural language processing (NLP) and basic machine learning (ML). This system can be used to convert a user-entered message into usable information, analyze its intent, and convert it into actions by enabling the resulting chatbot to comprehend human languages. Wit.ai also allows the chatbot construction using entities, intents, contexts, and actions, with the support of NLP. In other words, Wit.ai [11] provides natural language interface for applications to turn sentences into structured data. LUIS.ai [12] is capable to take the sentences the user sends and interpret them in terms of the intention to allow further analysis. Basically, API.AI, Wit.ai, LUIS.ai provide similar and efficient NLP and ML features. However, extra coding effort and form-based development style of these methods considerably hinder their applicability.

Based on the above examination of existing chatbot frameworks and development tools, we identified the following key requirements for the proposed approach.

1. The ability to construct a chatbot by designing the chatbot process.
2. Visualized process for the chatbot process.
3. Support of the extraction of keywords and establishment of dialog branches.
4. Straightforward integration of external RESTful services in a bi-directional way.
5. Conversational interface using existing social platforms.
6. Modularize chatbot processes for easy reuse of components in subsequent applications.

3. FCF: FLOW-BASED CHATBOT FRAMEWORK

In this research, the proposed FCF was built on Node-RED by installing a suite of FCF nodes to realize the core functionalities of chatbots. Node-RED [13] is a flow-based visual development tool used in the connection of hardware devices, APIs, and online services. Node-RED features (1) browser-based flow construction and editing; (2) programmable modules using node.js; and (3) support for social development. Multiple frameworks in various domains, such as IoT (Internet of Thins) [14, 15] and Health Care [16], are built atop Node-RED.

The overall architecture of the proposed FCF framework is shown in Fig. 1. In this research, we focus on the newly-devised Node-RED nodes (chat nodes, IM bridge, and Web service bridge), the data format for the proposed Service Webhook, the chatbot dialogue patterns, and the chatbot scenarios. For the usage of FCF, the application developer may use the FCF nodes or the chatbot dialogue patterns to construct and deploy chatbot applications in a flow-based way using the visual development environment. The end users could use the published chatbot applications on the social platforms such as Facebook.

3.1 Node-RED-based Chatbot Framework

The proposed FCF (flow-based chatbot framework) provides the following three features.

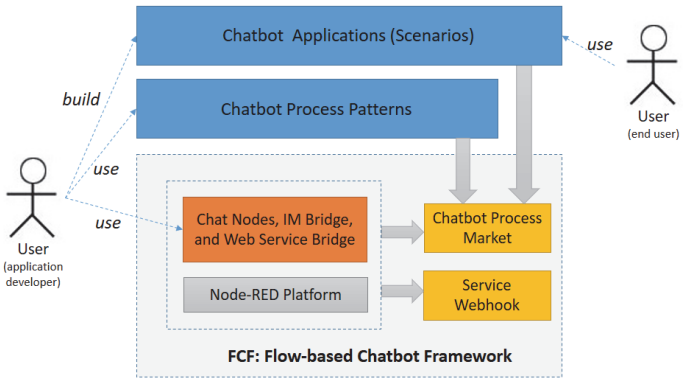

Fig. 1. Overall architecture of FCF.

(1) **Chat Nodes:** FCF provides visual nodes to handle the basic functionalities of conversation-related dialogs. In Node-RED, a node is created by packing three files: (1) JavaScript code, providing core functions written in node.js; (2) HTML code, including the edit template to let users fill out configuration parameters; and (3) package.json, specifying the metadata for the node. In this study, we devised five types of Node-RED nodes: *Message*, *Dispatcher*, *Keyword Extraction*, *Data Collection*, and *Frame*. The objectives of these nodes are outlined in the following:
- **Message:** Transmission of plain text messages.
- **Dispatcher:** Analyzing the intent of the incoming messages from users to identify the branch for the following chatbot process.
- **Keyword Extraction**: Retrieval of keywords from incoming messages for subsequent processing.
- **Data Collection:** Collection of necessary data from the user.
- **Frame**: Storage of data produced by *Keyword* and *Pull Service* nodes to serve as memory in the chatbot process. This node plays the role of memory for the whole conversation, which is a crucial part for a chatbot [2].

(2) **IM Bridge:** It is used for connecting to the IM (instant messaging) function of existing social platforms – it currently contains three types of FCF nodes for supporting the integration of Facebook: *Facebook In*, *Facebook Out*, and *Facebook Notification*.
- **Facebook In:** Receiving messages sent from the user's Facebook Messenger.
- **Facebook Out:** Sending messages to Facebook Messenger by following incoming messages.
- **Facebook Notification:** Proactively pushing messages to the user's Facebook Messenger.

(3) **Web Service Bridge:** It links external services – currently contains two types of FCF nodes: *Pull Service* and *Push Service.*
- **Pull Service:** Requesting data from external RESTful services. The external service produces data in the format defined by FCF.
- **Push Service:** Pushing data to external applications. FCF automatically generates a service URL for a Push Service node to allow external applications to invoke the service endpoint.

3.2 Service Webhook

Webhook [17] is a user-defined HTTP callback, which is usually triggered by an event. When an event occurs, the source website sends an HTTP request for a URI configured in Webhook. Social platforms, such as Facebook, LINE, and Slack are using Webhook to connect external services. When a social platform receives a message from a user, it sends the message via Webhook to the configured URI for subsequent actions. Based on Webhook, in FCF, we devised our Service Webhook mechanism to allow the integration of a chatbot process and external services. We devised the Pull/Push Service node, a series of Web APIs (Web service) in Webhook form, and specified the data exchange format using JSON (JavaScript Object Notation) to satisfy most business situations.

(1) Pull Service

As mentioned above, the *Pull Service* node requests data from external RESTful services. We achieved this by devising a JSON-based data format applicable to the requests issued by FCF as well as the response received by FCF. As shown in Fig. 2, the data format of the requests and responses is the same, including *Query*, *UserData*, and *Results*.

The *Query* and most of the *UserData* are derived from the *Data Collection* node, whereas UserID (a username on Facebook or another social platform) is obtained automatically by FCF. In the request to the external service, the *Result* field is left blank. When the response is returned, the *Result* field is filled out by the service. In the example in Fig. 2, *Query* data in the *Data Collection* node includes the brand and the price expected by the user as well as the name and phone number entered by the user. Invoking the services specified in the *Pull Service* node leads to the filling out of the *Result*. The collected data is stored in the *Frame* node for subsequent use.


```
{
  "Query": {
    "brand": "HTC",
    "price": 20000
  },
  "UserData": {
    "UserID": "asdfghjkl",
    "name": "SOSELab",
    "phone": "0912345678"
  },
  "Result": {
    "ItemID": "H2017",
    "ItemNumber": 2
  }
}
```


Fig. 2. Example of data collected from Pull Service node.

(2) Push Service

The *Push Service* node is responsible for passively receiving data from external applications. As shown in Fig. 3, the external application submits to the *Push* node JSON-based data containing *UserID* and *Message* created by the application developer. When the *Push Service* node receives the data, FCF may send the message to other *Pull Service* nodes to perform subsequent tasks or to the *Facebook Notification* node for directly dis-

playing the message in the user's Facebook Messenger. The example in Fig. 3 displays an acknowledgement for an issued order.

```
{
  "UserID": "asdfghjkl",
  "Messsage": "Your order has been sent out."
}
```

Fig. 3. Example showing transmission of Push Service node data.

3.3 Chatbot Dialogue Patterns and Scenarios

Based on FCF nodes, we devised five types of common dialog pattern, including *Branch Dialogue*, *Data Collection Dialogue*, *Keyword Extraction Dialogue*, *Service Connection Dialogue*, and *Service Push Dialogue*. These patterns allow users to construct a variety of complex chatbot processes to satisfy the requirements of specific chatbot applications.

(1) Pattern 1: Branch Dialogue

The *Branch Dialogue* includes the three basic nodes (*Facebook In*, *Facebook Out*, and *Message*) as well as a *Dispatcher* node for use in analyzing the user's intentions by examining the incoming message. This enables the chatbot to provide different response messages in accordance with the user's intentions. In the current version of FCF, the *Dispatcher* node relies on API.AI for intention analysis; *i.e.*, FCF sends the received message to the API.AI and obtains the "Action" of the user intent. The value of the returned action indicates the appropriate branch for the conversation. Fig. 4 illustrates the flow of this pattern, which can be used in the construction of basic chatbot processes.

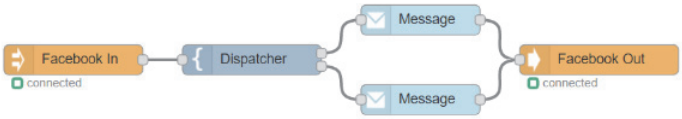

Fig. 4. Branch dialogue pattern.

(2) Pattern 2: Data Collection Dialogue

In the *Data Collection Dialogue*, the *Data Collection* node is responsible for prompting the user to enter required information in accordance with the hint messages specified by the *Facebook In* node. When a user message is received, pre-defined response messages are shown in Facebook Messenger using the *Message* node and the *Facebook Out* node. Meanwhile, the collected information is stored in the *Frame* node. Fig. 5 illustrates the flow of this pattern in FCF.

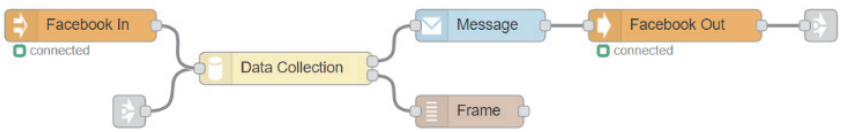

Fig. 5. Data collection dialogue pattern.

(3) Pattern 3: Keyword Extraction Dialogue

In the *Keyword Extraction Dialogue*, the *Keyword Extraction* node receives messages sent by the *Facebook In* node, filters out unnecessary content, and forwards the qualified content to API.AI for keyword analysis. The returned keywords are stored in the *Frame* node for further processing. Fig. 6 illustrates the flow of this pattern in FCF.

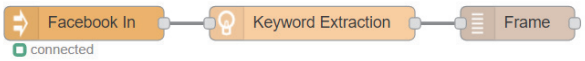

Fig. 6. Keyword extraction dialogue pattern.

(4) Pattern 4: Service Integration Dialogue

The *Service Integration Dialogue* is meant to leverage external RESTful services in answering user questions. The *Keyword Extraction* pattern is used to capture the core information in the user question. The *Pull Service* node invokes a specified RESTful service using keywords stored in the *Frame* node as input parameters. The response of the service is also stored in another *Frame* node to allow access to the *Message* node. The wrapped response message is sent to the user's Facebook Messenger by the *Message* node and *Facebook Out* node. Fig. 7 illustrates the flow of this pattern in FCF.

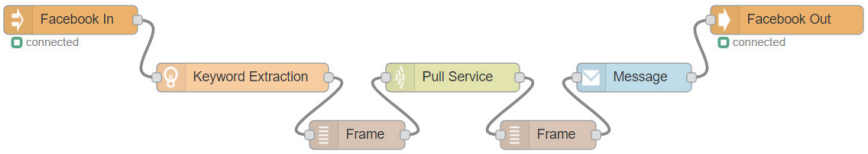

Fig. 7. Service integration dialogue pattern.

(5) Pattern 5: Service Push Dialogue

FCF also supports the integration of RESTful services with converse direction. External applications or services can transmit data to the chatbot by invoking *Push Service* node and triggering the subsequent nodes. As shown in Fig. 8, when the *Push Service* node is called, the message with service response is sent to the Facebook Messenger by the *Facebook Notification* node. Fig. 8 illustrates this pattern in FCF.

Fig. 5. Service push dialogue pattern.

These FCF nodes and chatbot dialogue patterns enable users to build a variety of application scenarios for their chatbots. We also designed three typical application scenarios as sample chatbots for use as references, including the Q&A chatbot, transaction chatbot, and hybrid chatbot with transaction and Q&A features.

(1) Scenario 1: Simple Q&A Chatbot

The *Simple Q&A Chatbot* applies the *Branch Dialogue* pattern to build a chatbot

with a common FAQ (frequently asked questions) feature (shown in Fig. 9). The user inputs a question and the chatbot responds. The *Dispatcher* node is used to identify the type of question, and assign answers to the *Message* nodes of each branch to provide replies for each type of questions. Obviously, this chatbot can only answer specific questions. Nonetheless, it could be made more powerful by including *Service Pull* nodes to provide answers based on the knowledge base behind the binding RESTful service. Static information, such as product Q&A, service Q&A, or even regulations or laws, could be wrapped as chatbots using this type of chatbot process.

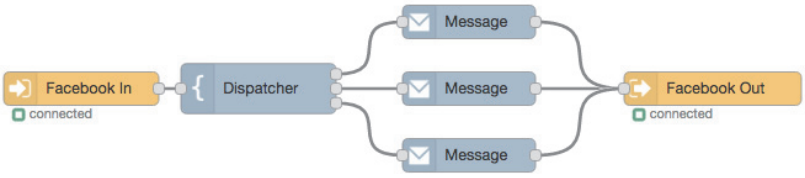

Fig. 9. Q&A chatbot.

(2) Scenario 2: Simple Transaction Chatbot

The *Simple Transaction Chatbot* combines the *Data Collection* pattern and *Service Integration* pattern to build chatbot providing services for ordering items or services (shown in Fig. 10). The chatbot prompts the user to fill out required user and order-related information, and invokes external services to actually place the order. Existing simple business Web services, such as those used for reservations in restaurants and booking train tickets, could be wrapped as chatbots using this type of chatbot process.

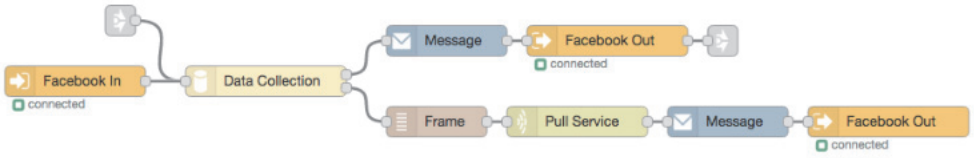

Fig. 10. Transaction chatbot.

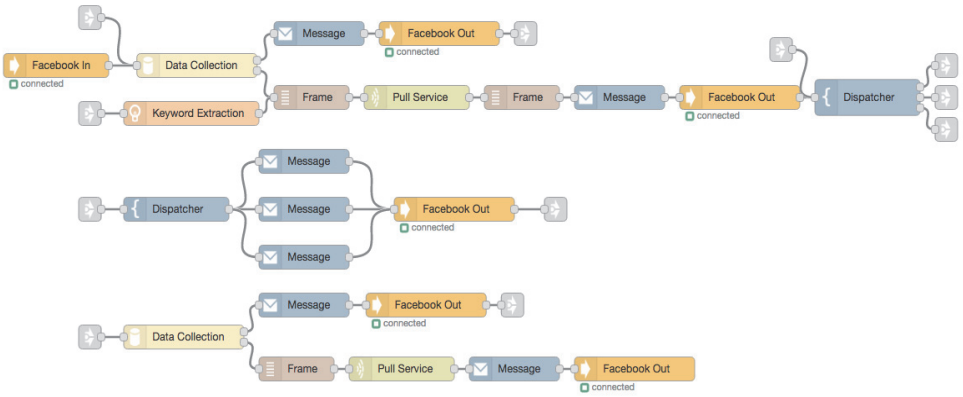

Fig. 11. Chatbot with hybrid functions.

(3) Scenario 3: Hybrid Chatbot with Transaction and Q&A Features

The *Hybrid Chatbot* combines the previous two scenarios to enable hybrid functions using the *Keyword Extraction* node, *Frame* node, and *Dispatcher* nodes (shown in Fig. 11). This chatbot process enables users to query product information and place an order. This includes three implicit sub-flows: (1) The chatbot first prompts the user to enter relevant information pertaining to the product the user is seeking. The user then decides whether to accept the product recommendation (first sub-flow); (2) If the user does not accept the suggestion, the chatbot allows the user to ask further questions about the target product (second sub-flow), or asks the user to refine their requirements and then modifies the product parameters accordingly (back to the first sub-flow); (3) When the user accepts the suggestion, the chatbot prompts the user to enter information pertaining to order placement (third sub-flow). Existing business Web services, such as online shopping, hotel reservations, flight bookings, and car rentals, could be wrapped as chatbots using this type of chatbot process.

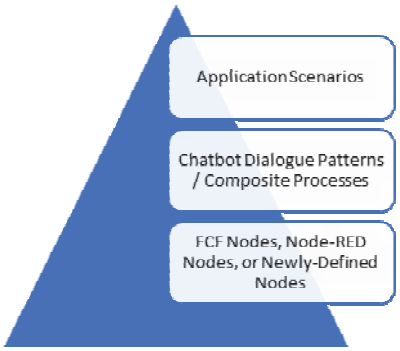

Fig. 12. Development stack for FCF.

Fig. 12 summarizes a development stack for FCF that could be used by developers in the formulation of their own development style for the construction of chatbots.

1. **FCF Nodes:** FCF Nodes is the core of the FCF framework. Developers can simply use FCF nodes to design and publish chatbot processes directly.
2. **Node-RED Nodes:** Node-RED provides a large number of previously published nodes. A developer with specific needs can first search for published nodes meeting their requirements and integrate those that meet their criteria within the chatbot process.
3. **Newly-Defined Nodes:** When no suitable nodes can be found, the developer can design a new node according to their requirements.
4. **Chatbot Dialogue Patterns / Composite Processes:** Developers can combine one to more chatbot dialogue patterns to facilitate the construction of chatbots. Developers can also combine multiple nodes to create a new composite flow, which can then be saved as a pattern for further use.
5. **Application Scenarios:** A complete chatbot process includes one to more chatbot dialogue patterns (composite flows). Developers could modify sample scenarios to create applications or combine multiple scenarios to construct chatbots of greater complexity.

4. ILLUSTRATIVE EXAMPLES

To demonstrate the feasibility and efficacy of the proposed FCF framework, we developed two illustrative examples, parking lots finder and smartphone shopping. Meanwhile, as mentioned, chatbot tools can be divided into three trends: (1) chatbot scripts; (2) chatbot toolkits or online platforms; and (3) AI-based developmental tools. Since chatbot scripts only provide dialog scripts or rules for the building of chatbots without supporting development environments, we decide to compare FCF with the second and third types of chatbot tools. Accordingly, we use the second example as the benchmark to conduct comparison with RedBot, a typical toolkit in the second trend, and API.AI, a representative service in the third trend.

4.1 Parking Lots Finder

The first illustrative example is a chatbot that could help users find parking lots nearby based on open data in Taiwan. The RESTful service linked by the chatbot collects the static open data of parking lots, such as basic information, location, and the parking fee for each parking lot, performs statistical analysis for parking lots in each region, and continuously updates the dynamic open data, such as available parking spaces. The expected chatbot conversations include:

1. The user asks the chatbot about three kinds of questions:
   (1) The available and cheapest parking lot in the current region and its parking fee.
   (2) The average parking fee in the current region.
   (3) The available parking lot with highest ranking in the current region.
2. The chatbot provides the answers for the user questions. For questions #1 and #3, the chatbot also displays the map for the recommend parking lot.

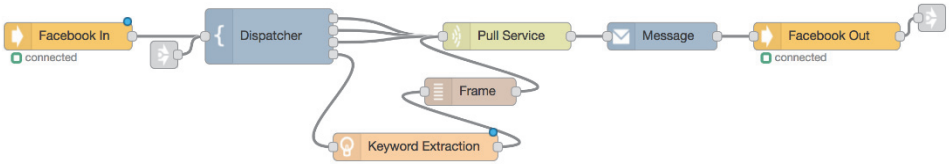

Fig. 13. Process for the parking lots finder.

For the detailed process that realizes the chatbot (shown in Fig. 13), initially, the chatbot receives the user request and collects the user location by the *Facebook In* node, and uses the *Dispatcher* node and *Keyword Extraction* node to analyze which question the user wants to raise. Next, the question keyword is stored in the *Frame* node. Based on the keyword, the chatbot invokes an external RESTful service by using the current user location as the input parameter to obtain answers. The external service has three endpoints to provide services about the cheapest parking lot, the average parking fee, and the top-rated parking lot for the region the user resides. Finally, the answer sends to the user's Facebook Messenger through the *Facebook Out* node.

From the viewpoint of development, the user, *e.g.*, the application developer, should

develop the RESTful service firstly and design the chatbot process using FCF in a visualized and flow-based way. Since designing the chatbot process is simple, the core task of this application is developing the RESTful service providing various information of parking lots by integrating open data of parking lots in Taiwan and leveraging techniques of statistic data analysis. In other words, domain-specific services in the field of data science could be wrapped as chatbot applications easily using the proposed FCF.

4.2 Smartphone Shopping

The second illustrative example is a chatbot that could help users to select smartphones and place orders. The expected chatbot conversations include:

1. The chatbot prompts the user to enter a series of information, such as brand and budget, for the user requirement.
2. The chatbot gives the user the information of recommended smartphones.
3. The user can ask the chatbot to modify the requirements or provide more detailed parameters for the smartphone.
4. The chatbot recommends another smartphone to the user. If the user is dissatisfied with the recommendation, back to step 3.
5. If the user is satisfied with the recommendation, the chatbot asks the user to provide additional order information, such as name, payment details, and delivery address, to place an order.
6. The external service directly pushes the status of order information to the user.

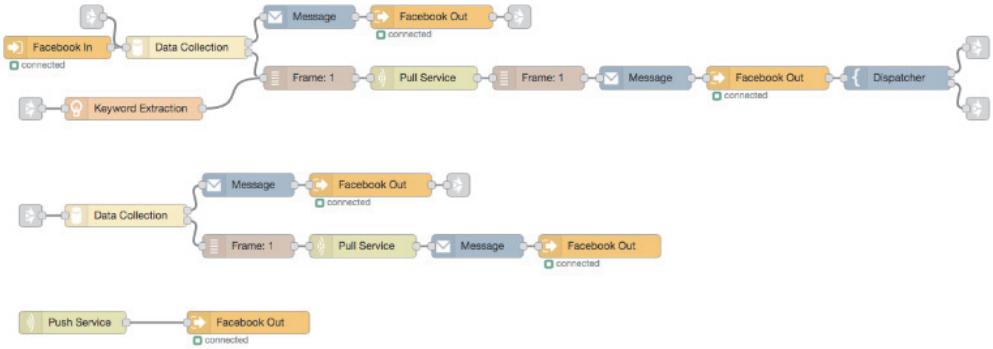

Fig. 14. Process for the smartphone shopping chatbot in FCF.

For the detailed process that realizes the chatbot (shown in Fig. 14), initially, the *Data Collection* node prompts the user to enter the expected brands and budget for the smartphone. The information is then sent out to the fist external RESTful service by the *Frame* node and the *Pull Service* node to give the user a preliminary recommendation result. In this study, we devised a RESTful service based on the our previous work, BRSD (business-rule-based service discovery) [18], to offer the functionality of recommendation. The result is stored in a *Frame* node for further processing. Using the *Dispatcher* node, the chatbot can analyze whether the user accepts the recommendation or

not. If not, the user can ask the chatbot to modify the requirements or provide more detailed parameters for the smartphone, including the phone memory capacity, the screen size, and the camera level by using the *Keyword Extraction* node. It is noted that the *Frame* node continues to update the data provided by the user so that the user does not need to input the same information again when the recommendation is rejected. Once the user accepts the recommendation, the *Data Collection* node prompts the user to give required information for the order, and then sends all collected information to the external RESTful service by the *Frame* node and the *Pull Service* node to place an order. It is also noted that the external service can actively push the message to the user using the *Push Service* node since the user's Facebook ID is already collected by the *Facebook In* node in the early stage of the conversation.

4.3 Comparison with RedBot and API.AI

Building chatbots by RedBot is similar to the proposed FCF, however, in order to meet the requirements of the experimental scenario, developers have to rely on other nodes of Node-RED and write additional code. The Node-RED flow using RedBot is shown in Fig. 15. Using RedBot, users have to use different ways to realize the functionality of FCF nodes, as follows:

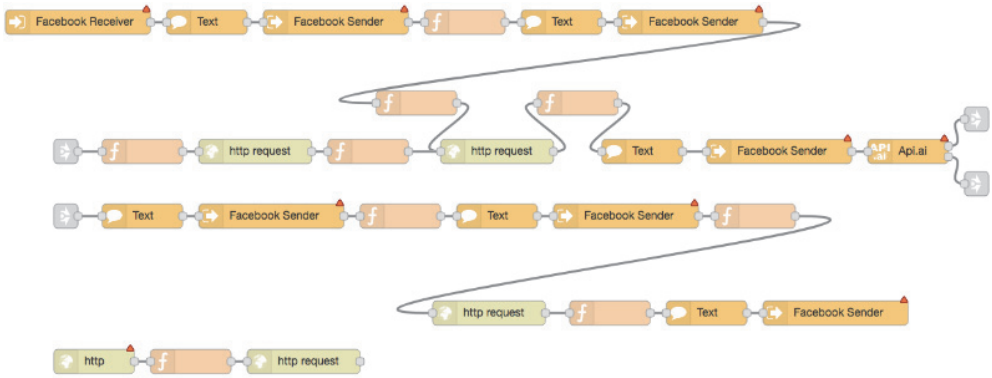

Fig. 15. Process for the smartphone shopping chatbot in RedBot.

- Data Collection: During the collection of multiple data items from a user, RedBot must include multiple sub-flows comprising of *Text*, *Facebook Sender*, and *Function* nodes.
- Frame: RedBot needs to use *Function* nodes with customized code to store and adapt the data collected from the user for further use.
- Keyword Extraction: RedBot requires at least two *Function* nodes to send and receive data from API.AI, and an *HTTPRequest* node.
- Push Service: RedBot needs to establish an http server to receive external service requests.

For API.AI, at least three intents need (as shown in Fig. 16) to be established by filling out the web forms to achieve the requirements of the experimental scenario while

the user data are obtained by the slot filling mechanism of API.AI. The first intent prompts users to provide data on the expected brands and budget and sent out the requirement to the first external service to get the recommendation result. If the user is satisfied with the result, the chatbot enters the second intent to prompt the user to provide required data for the order and send out the data to the second external service to complete the order. If the user is dissatisfied with the result, the chatbot switches to a third intent for changing requirements. Because API.AI cannot retain previously obtained data, the chatbot may prompt the user to provide all data that he/she has given. If we plan to keep user data automatically, extra coding effort is required.

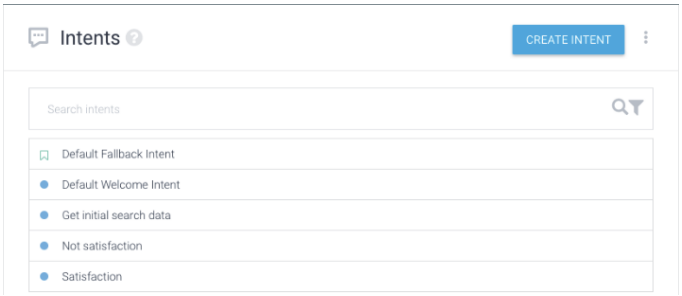

Fig. 16. Intent setting.

To evaluate the proposed FCF approach more formally, we compare RedBot, API.AI, and FCF based on three key quality indicators of ISO/IEC 25010 [19]:

1. Functional suitability, the degree to which a product provides functions that meet needs.
2. Usability, the degree to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use.
3. Maintainability, the degree of effectiveness and efficiency with which a product can be modified to improve it, correct it or adapt it to changes.

We adopt the three indicators firstly to conduct comparison since these indicators are closely relevant to the identified requirements in Section 2 than other ones in ISO/IEC 25010. The comparison is shown in Table 1. From the comparison results, it is obvious that FCF is superior to RedBot and API.AI from the viewpoint of functional suitability, usability, and maintainability.

5. CONCLUSIONS

In this research, we developed a novel chatbot framework, referred to as FCF (Flow-based chatbot Framework). FCF has two main features: (1) We provided a visual, flow-based environment for the design and deployment of chatbots; and (2) FCF is based on the newly-devised data format for Service Webhook, which allows bidirectional ser-

vice integration for chatbots created using this system. Besides, we also devised five chatbot dialogue patterns and three chatbot application scenarios to be components for the construction of complex chatbot applications. In a nutshell, we provided a visualized and composable chatbot framework to let building chatbot is like stacking bricks.

In the future, we will be developing an independent machine learning module to replace the API.AI service in order to improve the efficacy of keyword extraction and enhance the accuracy of branch determination.

Table 1. Comparison of RedBot, API.AI, and FCF.

	Functional suitability	Usability	Maintainability
RedBot	RedBot cannot satisfy all the requirements for chatbot development directly. Users must use other NoderRed nodes and write additional code.	Although RedBot also supplies a visualized, flow-based environment, extra coding effort decreases its usability.	Because RedBot needs to rely on additional Node-RED nodes and related code to build chatbots, numerous codes are possibly required to be changed when the requirement change comes.
API.AI	API.AI also cannot satisfy all the requirements for chatbot development directly. Extra coding effort is required to collect and keep user data since API.AI is not able to store collected data cross different intents.	API.AI provide a form-based development environment, not in a visualized and flow-based way. The usability of FCF and RedBot is better than API.AI.	Because API.AI also need to rely on additional nodes to build chatbots, numerous code could be possibly required to be changed when the requirement change comes.
FCF (The proposed approach)	FCF can satisfy all the requirements for chatbot development.	FCF provides a visualized and flow-based development environment with a set of straightforward nodes.	FCF could only need to adjust the chatbot process to satisfy the requirement changes. Coding effort is minimized using the flow-based way.

REFERENCES

1. R. J. L. John, N. Potti, and J. M. Patel, "Ava: From data to insights through conversations," in *Proceedings of the Biennial Conference on Innovative Data Systems Research*, 2017.
2. F. Radlinski and N. Craswell, "A theoretical framework for conversational search," in *Proceedings of Conference on Human Information Interaction and Retrieval*, 2017, pp. 117-126.
3. F. A. Mikic, J. C. Burguillo, M. Llamas, D. A. Rodriguez, and E. Rodriguez, "CHARLIE: An AIML-based chatterbot which works as an interface among INES and humans," in *Proceedings of EAEEIE Annual Conference*, 2009, pp. 1-6.
4. M. S. Satu, M. H. Parvez, and M. Shamim Al, "Review of integrated applications with AIML based chatbot," in *Proceedings of International Conference on Computer and Information Engineering*, 2015, pp. 87-90.
5. N. Petherbridge, "RiveScript," https://www.rivescript.com/.
6. G. Bellomo, "RedBot," http://red-bot.io/.
7. "Hubot," https://hubot.github.com/.

8. "Cleverscript," http://www.cleverscript.com/.
9. "Chatfuel," https://chatfuel.com/
10. "API.AI," https://dialogflow.com/
11. "Wit.ai," https://wit.ai/
12. "LUIS.ai," https://www.luis.ai/.
13. "Node-RED," https://Node-Red.org/.
14. M. Blackstock and R. Lea, "Toward a distributed data flow platform for the web of things (Distributed Node-RED)," in *Proceedings of the 5th International Workshop on Web of Things*, 2014, pp. 34-39.
15. K. Klawon, P. Ryan, and J. Gold, "Using OSUS and node red to integrate IoT devices based on events," in *SPIE (the international society for optics and photonics) Defense + Security*, 2017.
16. J. Olsson and J. Asante, "Using node-red to connect patient, staff and medical equipment," Department of Electric Engineering, Linköping University, 2016.
17. "About Webhooks," https://help.github.com/articles/about-webhooks/.
18. S.-P. Ma, K. Y. Chang, J.-H. Lin, C.-C. Ma, and J.-H. Lin, "QoS-aware query relaxation for service discovery with business rules," *Future Generation Computer Systems*, Vol. 60, 2016, pp. 1-12.
19. S. Wagner, *Software Product Quality Control*, Springer, Berlin, 2013.

Shang-Pin Ma (馬尚彬) received his Ph.D. degree in Computer Science and Information Engineering from National Central University, Taiwan, in 2007. Dr. Ma is currently an Associate Professor in the Department of Computer Science and Engineering at National Taiwan Ocean University. His research interests include service-oriented computing, software engineering, mobile computing, and semantic web.

Ching-Ting Ho (何靖霆) received his Bachelor (2015) and Master's (2017) degrees from the Department of Computer Science and Engineering, National Taiwan Ocean University, Taiwan. His research interests include software engineering, service-oriented computing, and chatbot technology.