Juno: An Intelligent Chat Service for IT Service Automation

Jin Xiao(✉), Anup K. Kalia, and Maja Vukovic

IBM T.J. Watson, Yorktown Heights, NY, USA
{jinoaix,anup.kalia,maja}@us.ibm.com

Abstract. Juno is a chat-based service that interacts with user through natural language, in order to understand, assist and execute the user's service request with a IT Service Management (ISM) system.

1 The Need for Juno

IT service request processing and execution makes up a significant portion of any large enterprise's IT operations. Traditionally, IT service request processing is quite labor-intensive involving many personnel of varied IT expertise. A helpdesk support person is needed to understand and determine the type of IT service required, generates a IT service ticket and dispatching it to appropriate systems operation team. Before the systems operation team can operate on the ticket, the required action or change is scrutinized by various security expert, client manager and sometimes the client's operations team, to ensure that the actions to be performed adhere to the security policies and infrastructure consistencies. Once approved, the systems operation team will pickup the tickets and attempt to perform the requested actions. Sometimes, the ticket may not contain sufficient information to fulfill all of the parameters of a change (e.g., didn't mention which drive a Disk should be mounted on) or contain incidental errors (e.g., ask to patch a DB2 database "dbinst1" that does not reside on the specified machine x, although DB2 database "dbinst0" does reside on x). The ticket then have to be discussed with the requester for further clarity and correction. This process overall is labor-intensive, error-prone and slow. Over the past few years, significant strides have been made in making IT service management process simpler and more automated [1–4]. Automation platforms supports large variety of change automata ranging from OS support, to databases, hardware configurations, etc. Typically these automata take inputs on the parameters and options to suite the specific target environment and the particular change request. IT service management (ISM) systems are created to help streamlining and expediting the request creation, dispatching and approval process. IBM Control Desk (ICD) is such a representative system IBM Global Technology Service uses for IT service request management. Moreover, helpdesk support is replaced by a self-service catalog user interface. A service catalog contains a listing of pre-approved services a service requester can use to facilitate a service request action. Each catalog service takes on a web-based form that contains the required parameters

X. Liu et al. (Eds.): ICSOC 2018 Workshops, LNCS 11434, pp. 486–490, 2019.
https://doi.org/10.1007/978-3-030-17642-6_49

and configuration options for automata execution. Furthermore, these catalogs are designed as end-system-centric, whereby a requester should first specify the end system to be acted on. Over time with the use of these self-service catalogs, some challenges have emerged: (1) requester may not necessarily know what the change action they are looking for is called or where it's located in the catalog. For instance, "grant user access to mailbox" could be located in User Account Management, Mailbox Management, or Security Services. This sometimes causes frustrating browsing experience; (2) requester may not know the name/value of a parameter that is valid for the change action. Frequently, the requester needs to look up fully qualified domain names for servers, the exact database name on a server, etc.; (3) Sometimes, an action's parameter list and configuration items can be sizable, and it is difficult for a requester to look up all of the configuration values and entity names. For instance, to add a MQ channel, the parameter includes: server name, MQ application name, Queue manager name, channel name, queue type, etc. The net effect being even with a self-service user interface, requesters (especially new users) tend to have a lot of questions and need to do many lookups. Juno is designed and created to be an intelligent self-service chatbot to address these challenges. Its aim is to be a virtual helpdesk support agent for a user that can directly engage Juno in a conversational format that does not require the user to browse the service catalog and to assist the user in parameter fulfillment through recommendation, auto-fill and validations. Our experience suggest that Juno serves as a much smoother and guided user experience to the service requesters.

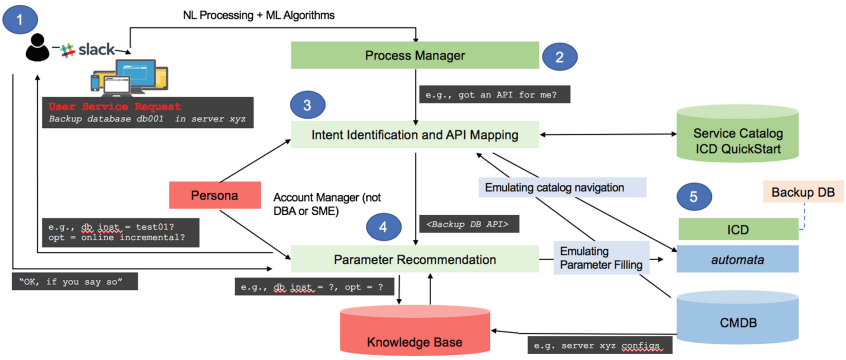

Fig. 1. The Juno architecture

2 The Juno Architecture

Juno is a collection of micro-services that communicates through APIs. The chat front-end of Juno (i.e., the chatbot) is built for a specific chat application, and is independent of the back-end reasoning and execution engines of Juno. The entire Juno service can operate across multiple chat applications, and service

multiple users at the same time. As depicted in Fig. 1, Juno consists of process manager, intent identifier, parameter recommender, execution engine, knowledge base, and persona.

Process manager handles the authentication and authorization process, as well as session management. It is the interface to the chatbot. When a user initiates a conversation with Juno via chat, the process manager first authenticates the user. The authentication process involves establishing and linking a user's chat identity to his/her back-end ISM identity. Secondly, based on the user's ISM identity, the user's role and access rights to the target infrastructure and the authorized set of actions are also established. These access rights will be used throughout the session to ensure the user's interaction with Juno and the backend ISM are compliant with security policies. The process manager also keeps track of ongoing requests the user may have initiated during the session and is able to therefore bridge the chatbot's dialogue state to Juno's internal processing state.

The intent identifier employs natural-language processing, feature extraction and classification techniques to map a requester's requested action to a service catalog entry if there exists one. The service catalog entries are a priori onboarded to Juno as a meta catalog. The meta catalog entry for a service entry consists of: API descriptions of the automata associated with the service entry, the action utterances, the parameter key-values, parameter utterances, and types of the parameters. This design facilitates a dynamic approach to onboarding new service catalog entries to Juno. The Intent identifier creates a weighed feature map of the API parameters and actions. When a request is received, for example, "backup database db001 on server xyz". The intent identifier does NL-based preprocessing, tokenization and lemmatization to generate list of likely action and parameter related phrases (e.g., "backup", "database", "db001", etc.). The phrases are further searched with Intent identifier's word dictionary to identify normalized tokens for them (e.g., "back up", "backup" maps to "backup"; "db", "database" maps to "database", etc.). The normalized tokens are used to fit the weighed feature maps of APIs for classification. Furthermore, likely parameter value word phrases such as "db001" are also searched in Juno's knowledge base to determine if they exist, and what type of parameter they are. This information is also fed into the classifier. The resulting top ranked matches are then returned to the chatbot.

The parameter recommender is involved for further interaction with the requester via process manager. Based on what the requester selects, the parameter recommender takes the features generated by the intent identifier and loads the service API corresponding to the selected service entry. Different from the meta catalog entry, the service API associated with the entry contains further technical details on the parameter key-values, types of parameters, associated concepts in the knowledge base, value constraints (e.g., Integer) and defaults, as well as mapping information to generate backend data payload to the backend ISMs

(this demo uses ICD as the ISM that takes in a structured XML payload). The parameter recommender first fits the word tokens into parameters required, it does verification with the knowledge base to ensure the word tokens are entities exists in the target infrastructure, the requester have access rights to it, have the right concept (e.g., "db001" is indeed a database), and the right system dependencies (e.g. "xyz" is indeed a server and "db001" is hosted on "xyz"). The outcome of this step produces a valid partially complete API payload. Then the recommender does auto-fulfillment on parameters where it can conclusively determine the value for. In the instance of "backup database db001 on server xyz", the API also requires a database application name. Since the recommender is able to determine the existence and validity of "db001" and "xyz", also "db001" is hosted on "xyz", it performs a lookup in the knowledge base for the name of the database application that hosts "db001" on server "xyz", and fulfills that parameter. In the case, where there's an inconsistency on the requester's provided values. For example, "xyz" is not a valid server, or "xyz" does not host any database called "db001", then the user is provided recommendations for the valid candidate server that hosts "db001", and the valid databases that is hosted on "xyz". The parameter recommender interacts with the user via process manager to systematically recommend parameters (as we'll see in the demo), until all parameters of the API is fulfilled.

The execution engine serves as the interface to the backend ISM system. It takes a completed service API and translates into the native payload the backend ISM expects. For this demo, XML payloads are generated to create and process requests in ICD. The execution engine can also pull request status from the ISM, and pull configuration items from Configuration Management Database (CMDB). The CMDB is the ISM's representation of the target infrastructure.

The persona service is persistent across sessions. It records a requester's identity, access rights (cached from ISM), request history, ongoing request status, as well as any linguistic utterance associations specific to the user (e.g., server "xyz" is referred to by the requester as "ICD test server").

The knowledge base is a service that is heavily used by the intent identifier and parameter recommender service to: search for existence of an entity, understand the concept type of the entity, and how the entity is situated in the target infrastructure. The CMDB of a ISM is injested to automatically generate/update the knowledge base.

3 What Is in the Demo

In this quick demo of Juno we have a Slack chatbot built that communicates with the Juno service. The dialogue management for the Slack chatbot is minimal, mostly passing process manager's response directly to the user.

First, users issue browsing command to retrieve the list of target systems they have access to. Notice that the user "Anup" and user "Jin" have access to different systems. Then, Juno guides user through a series of commands, many of which requires Juno to auto-fill parameters, correct user errors, and/or

recommend parameter values. Because ISM system interaction takes time, we only show the ISM path once during this demo.

4 How to View This Demo

The demo file is titled "Juno_DEMO_ICSOC_2018.mp4". It is a MP4 video format, no sound, any MP4 viewer should be able to play this video. Thank you.

References

1. Alès, Z., Duplessis, G.D., Şerban, O., Pauchet, A.: A methodology to design human-like embodied conversational agents. In: International Workshop on Human-Agent Interaction Design and Models, Valencia, pp. 1–16 (2012)
2. Ayachitula, N., et al.: IT service management automation - a hybrid methodology to integrate and orchestrate collaborative human centric and automation centric workflows. In: Proceedings of the 4th International Conference on Services Computing, Salt Lake City, pp. 574–581. IEEE (2007)
3. Kalia, A.K., Telang, P.R., Xiao, J., Vukovic, M.: Quark: a methodology to transform people-driven processes to chatbot services. In: Maximilien, M., Vallecillo, A., Wang, J., Oriol, M. (eds.) ICSOC 2017. LNCS, vol. 10601, pp. 53–61. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-69035-3_4
4. Kalia, A.K., Xiao, J., Bulut, M.F., Vukovic, M., Anerousis, N.: Cataloger: catalog recommendation service for IT change requests. In: Maximilien, M., Vallecillo, A., Wang, J., Oriol, M. (eds.) ICSOC 2017. LNCS, vol. 10601, pp. 545–560. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-69035-3_40