SwarmGen: a framework for automatic generation of semantic services in an IoT network

Pablo Calcina-Ccori  
Instituto de Matemática e Estatística  
Universidade de São Paulo  
Email: pcalcina@ime.usp.br

Laisa Caroline Costa De Biase  
Escola Politécnica  
Universidade de São Paulo  
Email: laisa@lsi.usp.br

Geovane Fedrecheski  
Escola Politécnica  
Universidade de São Paulo  
Email: geovane@lsi.usp.br

Flávio S. Corrêa da Silva  
Instituto de Matemática e Estatística  
Universidade de São Paulo  
Email: fcs@ime.usp.br

Marcelo Knorich Zuffo  
Escola Politécnica  
Universidade de São Paulo  
Email: mkzuffo@usp.br

Abstract—The growing adoption of Internet of Things technologies leverages the development of complex IoT applications. This situation is particularly visible in the consumer electronics realm, where integrating products from different manufacturers requires considerable effort. Currently the development of IoT applications is a time consuming task that demands expertise from different areas. This situation motivates the creation of tools that reduce the effort of creating IoT applications.

This work aims to simplify the creation of ready-to-run semantic services from a high-level description. We proposed and implemented the SwarmGen framework, and successfully tested the results in an existing configuration, with a reduced development effort.

I. INTRODUCTION

The increasing production and adoption of Internet of Things technology is permeating different areas of industry. This creates a fertile field of opportunities to create new innovative applications combining existing consumer electronics. An essential aspect for this development is the use of agile tools for rapid proof of concept creation.

Currently, the development of these applications requires strong skills in software development, hardware interfaces and communication protocols. The setup of a development environment is mostly a complex task that goes beyond the application business logic.

Previously [1][2], we proposed the Swarm IoT middleware as a communication platform for consumer electronics and IoT devices in general. A core element of the Swarm platform is the SwarmBroker, a software that is installed in every device that participates in the Swarm network. The SwarmBroker mediates the communication of devices, as shown in Figure 1.a, and provides platform services which include registry, discovery, mediation, access control, and contracting (Figure 1.b). Thus, the Swarm platform creates a network of heterogeneous devices, and provides security, access control, trust, and payment out-of-the-box. These characteristics allow the rapid creation of a network of consumer electronics, with minimum effort.

In order to take advantage of the Swarm infrastructure, in this paper we propose SwarmGen, a framework that automates the creation of a ready-to-execute source code project. The generation of software from a high-level specification reduces repetitive tasks, permitting that developers concentrate on the creative aspect of their applications, thus increasing productivity [3].

SwarmGen constitutes a complementary tool for the Swarm platform that speeds the creation of ready-to-use service source code based on a minimum declarative description. We propose an architecture and implementation, showing the synergies with the Swarm platform and the advantages for reducing programming effort for the application developer.

This paper is organized as follows. In Section II we explore related work. Section III presents the architecture of SwarmGen, including its internal modules. Next, Section IV provides more details about SwarmGen implementation. In Section V we show the results of our work and give some suggestions about future work that can derive from the present paper. Finally, Section VI concludes the paper.
II. RELATED WORK

Early advances in proposing development tools for home automation [4] recognize the necessity of an integrated environment to reduce development efforts and to reach market sooner. That work uses a CEBus network via powerline modules to exchange information. Although the technology then used is currently deprecated, this contribution provided us a confirmation of the importance of efforts in development tools for consumer electronics applications.

Later, a framework with similar purposes was created [5], however, it focused on the creation of a middleware for distributed objects, using CORBA [6]. Nowadays, the use of these technologies is deprecated and has been superseded by standard and lightweight communication protocols, such as RESTful services [7].

Also, authors in [8] recognize the importance of efficient tools for rapid development. That work focuses particularly on the development of a virtual environment for testing consumer electronics applications, based on open source software.

Another work [9] proposed a framework that generates device-specific code for IoT devices from a set of modeling languages. From a WSDL-based service description it generates a common intermediate representation with the Essential Meta-Object Facility. Although that framework generates code for specific platforms, such as Windows, Linux, and Symbian, it is still based on WS-* standards for implementing the gateway and the high-level description, which incurs in a performance issue in the gateway level.

Hydra [10], has in common the effort to create a semantic model for generic description of Web service API. Although it does not provide a specific code generator, the semantic model for Web API provides a high value that was partially adopted in our work, particularly the definitions of a service Operation.

In a previous work [11] we proposed a framework to generate code generation of services in the Internet of Things. Although that work generates the project for a Swarm service it is based on a non-semantic format, which makes it impossible to reuse type definitions from widely used external sources, such as the Schema ontology¹ [12]. The present work represents a significant improvement from [11].

III. FRAMEWORK ARCHITECTURE

The SwarmGen framework generates a project structure from a semantic service description file. Figure 4 provides a high-level architecture of the SwarmGen framework. The following sections show the details of the corresponding modules.

A. Semantic service description

The semantic service description format is an ontology-based model that is defined in [13]. This file contains the information necessary for the service to be discovered and invoked by other services in the network, such as operation

1http://schema.org

list, input and output parameters, service and operation types, geolocation, and pricing preferences.

The serialization format for the Swarm service model is JSON-LD [14][15], as it provides a human-friendly manner to describe a semantic graph. Older serialization formats such as WSML, RDF-XML and RDF-N3, did not have a large adoption by the Web development community due to the complex syntax and the so called semaphobia (phobia to semantics) [16]. The JSON-LD serialization permits the writing of a triple-based graph in a key-value format (duple-based), besides JSON being an already popular description language in Web development. Figure 2 shows a basic example of a semantic service description.

The use of semantics in the Swarm facilitates interoperability and flexible matching of equivalent services. The advantages of the use of semantics are described in detail in Section III-D.

B. SwarmGen modules

The Service Description Parser module parses a service description, as shown in Figure 2 and identifies the operations and data types. From this information it creates an internal representation which is then used by the Service Generator module.

The Dependency resolver module is responsible of determining the dependency packages necessary to execute the project. Additionally, a Code and File templates repository is used to produce the source code for a particular target. Thus, other template repositories can used to generate projects for different combinations of programming languages and platforms.

The Service Generator module uses the output of the previous module to create the project structure and provides an interactive interface to customize of the project generation.
C. Structure of the generated service

The Service file is the actual service file. It includes a functioning Web service ready to be used in the Swarm. While the service file is fully functional, only the actual business logic needs to be implemented. This way we simplify the development of Swarm services for an application developer.

Requirements is a separated file containing all the dependencies for the service, and Execution script provides the automation of common tasks to compile, test and run the generated service.

Broker Client is a software module that automates the interaction between the service and the corresponding broker. This interaction is depicted in Figure 3, and consists in two main steps: registry and service request. In the registry phase, the service sends its description to the associated broker, which in turn, spreads the document across the network. The discovery phase initiates when a service searches the network for a suitable service to consume, the search is based on a declarative semantic query format which is sent across the network.

Broker Client module implements the necessary messages to establish this communication among service and broker. Additionally, it provides a method to perform a service level agreement (SLA), and contracting the service provider, including payment and evaluation.

The Access Control policies file contains the default policies to access the service. All Swarm services must define this file to determine the rules and permission levels for other Swarm participants [17]. The access control policies create rules based on triples (object, permission, subject), which determine which service or group of services can access a specific operation.

Wallet information represents the private and public keys used by the service to sign transactions with other services. A transaction represents the interaction between a service consumer and service provider, through the payment for the service use and attribution of a reputation score at the end of the transaction [18]. Both payment and reputation are stored in a Blockchain structure.

D. The role of semantics

The Swarm platform uses a semantic model for services to allow interoperability in different levels [13]. First, by creating a taxonomy of service and operation categories, which facilitates the flexible discovery of services in the network. Second, by creating a hierarchy of data types for input and output service parameters, which permits to identify the possibilities of service composition by input/output compatibility. Third, it leverages semantic data exchange, due to the use of ontology defined data types. Finally, it encourages the reuse of data types from existing ontologies, such as Schema.org2, Friend of a Friend3, The Music Ontology4, among others.

Fig. 3. High level sequence diagram of the life cycle of a service in the Swarm [13]

The concrete instances of the generated files for the current implementation are briefly described as follows. The service.py file is a standalone Web service based on the Flask framework [20]. The generated file includes a function definition for each declared operation, with annotations for the corresponding HTTP methods and access control policies, as shown in Figure 5.

Accordingly, requirements.txt contains the necessary packages to execute the service, imported from the Python Package Index5. Accordingly, the execution script corresponds to a properly configured Makefile.

The access control policies are written in a triple-based JSON file, whose format is specified in [17]. Finally, the key.json file stores the private and public key for service, according to [18].

Fig. 4. Architecture of the Swarm service generator

IV. IMPLEMENTATION

The proposed framework was implemented using the Python programming language, which presents several advantages for Internet of Things development [19], such as being easy to learn, code and debug; fast; wide library support; embeddable; interpreted; extensible; portable; and open source.

The concrete instances of the generated files for the current implementation are briefly described as follows. The service.py file is a standalone Web service based on the Flask framework [20]. The generated file includes a function definition for each declared operation, with annotations for the corresponding HTTP methods and access control policies, as shown in Figure 5.

Accordingly, requirements.txt contains the necessary packages to execute the service, imported from the Python Package Index5. Accordingly, the execution script corresponds to a properly configured Makefile.

The access control policies are written in a triple-based JSON file, whose format is specified in [17]. Finally, the key.json file stores the private and public key for service, according to [18].

2http://schema.org
3http://www.foaf-project.org/
4http://musicontology.com/
5https://pypi.org
V. RESULTS AND FUTURE WORK

The generated service was tested in a functioning configuration, with a SwarmBroker running in the same network. The sequence diagram of the execution of service registry and discovery are shown in Figure 3.

Comparing the necessary lines of code to describe the service (25), to the 350 generated lines of code (excluding the libraries), we estimate a ratio of 14:1. Although this value alone does not summarize exactly the quantitative relationship between the effort of writing a services, it provides a clear and undeniable saving of development time. The real benefits appear when an IoT application requires the interaction of many services; in that case, this ratio scales with the number of participating devices, making the saved effort even more considerable.

The current implementation of the SwarmGen framework constitutes an important aid to the developer interested in creating Swarm services. In order to enrich and facilitate the development experience we identify some improvements.

Although the generated project provides communication capabilities with other services and permits custom implementation of business logic, a modification of the declared methods is not automatically reflected in the original service description. We call this feature two-way conversion between high-level description and implementation. The absence of this feature does not hinder the correct functioning, nonetheless, it is a desirable new feature.

Aside from the generation of a skeleton for application development, generated a ready-to-deploy project, that targets several hardware architectures and containerized environments, such as Docker [21].

The current use case is related to home automation, using a wrapper service for a printer. In the future we aim to expand our use cases to other consumer electronics realms.

VI. CONCLUSION

In this paper we presented SwarmGen, a framework for automatic generation of semantic services for the Swarm IoT platform, which is capable of creating consumer electronics applications.

Through the use of SwarmGen, developers can focus on the high-level interface definition of their applications and rely on the code generation by SwarmGen. As a consequence, a higher level of decoupling is achieved between the API definition and business logic. Furthermore, our framework contributes in reducing the development time, using human-friendly formats and tools.

Future works include the implementation of two-way conversion, to reflect method updates in the service description, as well as expanding support to different hardware architectures, execution environments, and applications.

REFERENCES


