Combinatorial Service Rendezvous Middleware (CSRM)

Benjamin Gooder BS*, Rasib Khan PhD*, Paul-Rus Adrian BS†, Kokou Sossoe BS*
*Northern Kentucky University, †Babes Bolyai University, Romania
Email: *gooderb1@nku.edu, *khanr2@nku.edu, †rusp1@nku.edu, *sossoek1@nku.edu

***Abstract*—Coordination between IoT devices is a significant challenge hindering the realization of a truly smart world. We propose a system that implements a Combinatorial Service Rendezvous Middleware (CSRM) that matches subscriber devices' requests with publisher devices' services. Our system extends that of a simple matching system by coordinating the combination of published services to create a new, unique service that best meets the subscriber device's original request. We propose and test a matching algorithm to be used by the CSRM.**

I. Introduction

The Internet of Things (IoT) is a highly connected ecosystem of connected physical objects. Information and communication systems are invisibly embedded in the environment around us. IoT objects "speak" to each other in interoperable data formats [1]. Data is stored, processed and presented in a seamless, efficient and easily interpretable form. IoT has direct applications and involves new paradigms of software development in fields, such as home-automation, ambient assisted living, healthcare, smart-cities, industrial management and artificial intelligent computing [2].

A basic IoT system architecture is composed with: (i) the physical layer that contains embedded devices, (ii) the gateway layer that provides the mechanism and protocols for devices to expose their sensed data to the Internet (e.g. Wi-Fi, Ethernet, GSM, etc.). (iii) The middle-ware layer that facilitates and manages the communication between the real world sensed activities and the application layer. (iv)The application layer that assemble applications that can be used by the consumer to send commands to devices over the Internet via mobile applications, web apps, etc. [3]. A classical distributed software model does not fit well with IoT network of heterogeneous devices which collaborate with each other to deliver valuable functionalities to end users.

A Service Oriented Architecture (SOA) on the other side offers an adequate framework to enhance connectivity, interoperability and integration in IoT systems, it is the most used architecture for IoT frameworks. But with the expansion of today's IoT systems, SOA framework becomes limited. The adoption of microservices help manage easily systems extensibility and is a solution for poor service integration, scalability and fault tolerance in IoT. Because the true value of IoT lies in the interaction of several services from physical things, answers to this question are essential to support a rapid creation of new IoT smart and ubiquitous applications. The problem is known as service composition [4].

Our project's aim is to create a middleware that can handle this data, make the necessary combinations based on user's queries and send the resulting data back to the user. This project can then be used to combine 2 or more services for the clients to use later on, either as a continuous stream of data with the server as a middle point, or just give the IP addresses to the user for him to decide what to do later on.

One of the major limitations of this study is that the proposed architecture was not tested in the real world with a specific use case.

Our contributions in this paper are as follows:

- We propose an architecture based on microservices where a service assembler is introduced to manage consumer request and services coupling
- We provide an algorithm to accomplish the matching of services and then their combination to provide a unique service to the requester/user

The paper is organized as follows: the next session summarizes the related works. Section 3 presents a detailed discussion of the proposed model and architecture. In Section 4, we present the proposed matching algorithm. Section 5 shows the result of the tests conducted with our proposed middleware. Finally, we conclude and propose future works.

II. Background

Realization of a smart world comprised of pervasive IoT devices in a distributed network requires an increidble degree of coordination, standardization, standardization, and flexibility. As the number of IoT devices increases, so do the number and variety of services that they offer. This quickly introduces the need of an automatic and scalable approach for services' discovery, selection, and combination to respond to consumer requests.

The rendezvous of publishers' services and a subscriber request is achieved by the two meeting at an intermediate node. In the context of webservices, there are many systems that allow for matchmaking between user's needs and published services at a semantic level using ontological frameworks such as those employed by OWL-S [5] and WSMO [6]. To address the problems encountered in semantic matchmaking such as high heterogeneity between the various semantic models used across service providers, ontology matching techniques exist (based on syntactic, structural or semantic features) to align semantic models with each other [8].

Unfortunately, most matchmaking techniques require human input in order to register the published service or to explicitly map the relationship between users and the individual service. Practically, this is potentially a major bottleneck in the scalability of IoT. Additionally, due to the amount of IoT devices, the lack of interface standardization, and the general heterogeneity between IoT devices, no system currently gives participating publishers and subscribers the ability to combine and request services dynamically. As IoT matures as a field and transitions from theory to actualization, a system such as this will be needed more and more.

III. CSRM SYSTEM

A. System Components

1) Publisher Device (P): A publisher device is any IoT device that is collecting and publishing data. A publisher device has at least one capability by which it collects said data. A publisher device connects with the Combinatorial Service Rendezvous Middleware, provides it with a list of its capabilities' requirements, and can publish the capability's data asked of it by the middleware.

2) Capability (C): A capability is the medium by which the publisher device collects data. Therefore, the capability often defines the type of data it collects (e.g. video, audio, temperature, etc.) A capability should have at least one requirement. A capability's role is to collect data on behalf of the publisher device.

3) Requirement (R): A requirement is a specific characteristic of data. Requirements are used to uniquely identify either the data that is being collected by each individual publisher device's capability. They are also used to specify the data that comprises the user requested service. A requirement could be the location of the publisher device or a timestamp of the capability's collection of data. Requirements are used to uniquely identify either the data that is being collected by each individual publisher device's capability or the data that comprises the user requested service.

4) Middleware (CSRM): Middleware for IoT devices acts as a bond joining the heterogeneous domains of applications communicating over heterogeneous interfaces [20]. The Combinatorial Service Rendezvous Middleware (CSRM) goes beyond the stated definition of middleware in that it can match the requirements of a user device's service request to the requirements of one or more publisher devices' capabilities. This matching would then determine which publisher devices' capabilities' data sets could be combined to create the requested service. The CSRM would then combine the previously stated data sets to provide a single, unique service. Finally, the CSRM will deliver this unique service to the user device by establishing a session between the user and the service or services.

5) User Device (U): A user device is the device on which the user can both request and receive the requested service. A requested service has at least one requirement. It requests a service from the CSRM by specifying the requirements of the data it wants to receive. It must also be capable of receiving the CSRM's new service's data.

B. System Architecture

On a global level, our system is composed of different devices D where: D = (D1, D2, . . ., Dm) connected with each other and which have different capabilities C where: C = [C1, C2, . . ., Co]. Capabilities are composed of requirements R where: R = [R1, R2, . . ., Rn] which can be expressed as XML files. Capabilities are properties of Publisher devices which are advertised by different nodes. The user can send requests to the middleware through an internet connection.

If a user wants to experience a particular service characterized by specific requirements, it would make a request to the CSRM. The CSRM will search for and locate this user node which is requesting a particular service, and then the CSRM will combine devices to have a combination of capabilities to provide the service back to the user. This type of system which combines capabilities from IoT publisher devices to provide a unique service is best realized within the context of a distributed system architecture. To attain a scalable and fault-tolerant IoT framework we propose the adoption of microservices and a service-oriented architecture. Interactions between different services are conveniently handled in microservices architecture.

A Service Assembler is introduced to handle consumer request and service coupling (combining two or more services to generate a new functional service X which adheres to the service contract of the consumer based on the implementation details) and will be beneficial in reducing the call time [21]. Fig. 1 represents our system implementing a microservice architecture.

C. Communication Protocols

Fig. 3 depicts how a new publisher device interacts with the CSRM. At first, the new device connects to the CSRM. The server asks the new publisher device for its capabilities and requirements. These are then stored inside the CSRM's database alongside a unique publisher device ID. At the end, the server sends an acknowledgment to the device to confirm its successful registration and connection to the system. Fig. 4 shows the protocol for how a subscriber device would request a combination of services from the CSRM. The subscriber device connects to the CSRM and then sends it the capabilities and requirements that characterize the requested service. The CSRM queries its internal database for a list of publisher devices that could fulfill the request. For each viable publisher device, the CSRM requests the publisher's serivce, whether continuous or for based on a time-stamp, and combines the resulting data into one service. This combined service is then sent to the user.

IV. EXPERIMENTS

A. Prototype Implementation

The prototype of our project has 2 main components: the networking component and the "Search and Match" component. These 2 components would later be merged into one project that would fulfill our aims for the final demonstration.

The first component, the networking one, deals with creating the necessary implementation of a working server-client program. This implementation would contain 3 sub-programs: the server, the user client and the IoT device client.

The IoT device client would collect data from an IoT device, however in this case the data is hard-coded by the programmer. This data would then be sent to the server over the network so that the server may store it in a local database. In our implementation, the data is an object called “Element”. This object contains 2 lists that contain all the capabilities and metadata of the IoT device and the Network-IP of the device, which would act as a unique identifier. The server, also called the middleware, acts as a connecting hub between the IoT devices and the user. The server would receive the necessary data from all IoT devices connected to it and store it in its local database. The user client would then send over an “Element” object of its own, through which the server would try and find close enough matches and insert them into a list of elements, along with a percentage of each of the elements’ matching the original specifications given by the client. This list is later sent on to the user client.

The user client would create the “Element” object, which would act as specifications for the “Search and Match” algorithm run on the server. After the server has sent back its results, the user client would present these results on the screen for the user to choose 1 or more of the given results. The second component deals with the “Search and Match” algorithm implemented in the CSRM. This function simply iterates through the user’s requirements and searches for any available published services with the same requirements. Partial matching is allowed for in this function.

B. Experimentation Environment

The success of this CSRM solution and its surrounding architecture is largely based on the specific use case of the user. Because of this, determining the specific test environment was difficult to do without allowing too much bias towards a specific anticipated use case. It is the authors’ opinions that case studies based on the parameters inherent to a specific use case be tested in future research. Due to the generality of this study in regard to the proposed architecture’s use case, tests in this study were run on a simulated environment.

A single User device’s service request was initialized with between 1 and 100 requirements. This number was chosen randomly at the beginning of each simulated test. Each specific requirement was randomly initialized from a list of 500 unique requirements. 100,000 Publisher devices were each initialized with a random number of requirements between 1 and 100 requirements. Like the requirements that make up the user request, the Publisher devices’ specific requirements were randomly initialized from the same list of 500 unique requirements.

C. Results, Evaluation, and Analysis

These tests measured the coverage of the User device’s request (i.e. the number of matched requirements divided by the number of requested requirements). Each test also measured the speed of the matching algorithm. Though each test created an entirely new environment, the algorithm speed metric was only measured from the beginning of the search and match algorithm to the point where the CSRM had made a final list of responses to the final user requested service requirements.

An average coverage percentage of 25.25% and an average matching algorithm speed of 543 milliseconds was found over the course of 1000 simulated tests. It’s important to note the limitations of these tests. Since the tests were simulated and all of the requirements were randomly selected, this simulated test environment does not likely reflect the way in which a real user device would choose the requirements for its requested service. In addition to this, the speed of the matching algorithm does not account for the time taken by the CSRM to combine the publishers’ services.

V. Conclusion

The development of IoT software infrastructures encounter new problems related to networking and composition aspects. In this paper, we proposed an architecture based on microservices which integrate a service assembler in charge of managing consumer request and services coupling. The proposed architecture facilitates service identification, classification, combination and choreography and allows for ease of deployment, inter-domain communication and lightweight implementation. A middleware was proposed for the matching of services as well as their combination to provide a unique service that satisfies the original request of the user.

Future work ought to extend the findings of this study by applying the proposed architecture to a specific use case. One possible use case could include the combination of multiple video feeds to create 3 dimensional models. Another use case would be querying multiple environmental monitoring services to determine when the optimal time would be to interact with the environment in a specific way (e.g. watering, fertilizing, weed-killing, crop-rotation, etc.). Our work could be extended by implementing a service auditor that will control the quality of the service proposed to the user and how it will cooperate with the service assembler to propose a new combination of services. Finally, more efficient matching algorithms should be introduced into the CSRM in order to optimize service discovery and coupling.