A Middleware for Automatic Composition and Mediation in IoT Systems

Abdessalam Elhabbash
School of Computing and Communications, Lancaster University, United Kingdom

Yehia Elkhatib
School of Computing Science, University of Glasgow, United Kingdom

Georgios Bouloukakis
Télécom SudParis, Institut Polytechnique de Paris, France

Maria Salama
Department of Mathematics and Statistics, Lancaster University, United Kingdom

October 14, 2022

Abstract

This paper presents `Hetero-Genius`, a middleware architecture that enables construction and mediation in Internet of Things (IoT) systems. IoT systems are deployed across physical spaces such as urban parks, residential areas, and highways. The services provided by such IoT deployments are constrained to specific devices and deployment contexts. While existing interoperability solutions enable the "design time" development and deployment of IoT systems, it is often essential to dynamically compose systems that consist of other "small scale" IoT systems. To achieve this, *post-deployment composition is needed*, *i.e.,* runtime composition of diverse IoT devices and capabilities. `Hetero-Genius` supports system and service discoverability, as well as automatic composability. We demonstrate this using a real-world Internet of Vehicles (IoV) scenario. Our experimental evaluation shows that developers can save up to 47% of their time when using `Hetero-Genius`, as well as improve code correctness by 55% on average.

1 Introduction

Due to technological advances in system-on-chip design and manufacturing, sophisticated devices are highly available, affordable, and of minimal size and power requirements. Network connectivity also has greatly improved, with a myriad of cellular technologies such as WiFi, 5G, Zigbee, LoRaWAN, etc. These advancements have facilitated great progress in building applications to operate over various fabrics made of in situ sensing and actuating devices, and support back-ends in the cloud and edge.

In effect, the current state consists of a huge va-

riety of constituent systems that are independently deployed but co-existing in shared physical spaces. Each system was designed to satisfy certain user requirements, given its physical properties. For example, a parking system provides bookable spaces that are associated with its location and spatial capacity. Each system exposes its physical properties and (usually, proprietary) APIs to enable integration with other systems and applications. This breeds great heterogeneity, which contributes to the complexity of integrating systems, as system developers are overwhelmed with the amount of knowledge they need to acquire to develop such *integrated systems*.

The Service Oriented Architecture (SOA) computing paradigm has been used to integrate functionalities of independent systems [14] by exposing them as services that enable data exchange between providers and consumers. A number of protocols could be used with SOA (*e.g.,* MQTT [2], CoAP [26], XMPP [25]) each of which has its own data representation format. The adoption of different protocols by independent systems introduces further complexity on the integration of systems. This challenge has been tackled in the literature. For example, DeXMS [5] enables IoT devices to interact by synthesizing software mediators that translate between different communication protocols. However, issues such as runtime detection of protocol incompatibility and the on-fly creation of mediators are yet to be addressed.

The above challenges together highlight that composing systems that make use of a variety of technologies is difficult to design, maintain and adapt. Therefore, system engineering efforts to support post-deployment composition is needed [30, 12]. This calls for approaches that address these challenges and are able to (i) identify independent systems with their unique characteristics and contexts; (ii) recognize the need to compose with other systems; (iii) detect the need for mediation logistics; and (iv) set in action any support roles needed to actuate mediation.

In this paper we introduce the `Hetero-Genius` architecture to enable principled and automated construction and mediation in IoT and systems of systems contexts. `Hetero-Genius` makes novel use of an ontology for describing systems, facilitating comprehensive description of system properties, protocols and APIs. In turn, this supports system and service discoverability and automatic composability. This contribution aims to boost the productivity of composing IoT systems by shifting the responsibility of composing devices and sub-systems from system developers to the system itself. Based on an abstract workflow provided by a client, `Hetero-Genius` utilizes system descriptions and semantics to find the required systems and APIs to realize the abstract workflow as an executable workflow. Thereby, we make two contributions:

- `Hetero-Genius`, an architecture for automated system composition that favors description of independent systems, enabling systems to automatically: (i) discover other systems and their services; (ii) opportunistically integrate with them to satisfy a user request; and (iii) detect the need for mediation and to create mediators at runtime for direct interaction with other systems.

- A mixed-methods evaluation of `Hetero-Genius` using a real-world Internet of Vehicles (IoV) scenario. Specifically, we assess (i) effectiveness and added value through a user study with a group of developers, and (ii) scalability through extensive simulation of representative contexts.

Overall, `Hetero-Genius` cuts the amount of effort and, consequently, costs required from system developers to develop integrated systems almost by half compared to current development practices. Furthermore, it reduces the room for error resulting in code with 55% more correctness scores, on average.

2 Problem Statement

In this section, we outline the general problem space using an IoV-based motivating scenario to extract concrete technical challenges.

Motivating scenario: IoV. Highway systems are sophisticated networks of road infrastructure and devices. In France, the autoroute system has 11,882 km of highways, each of which is operated and maintained by a road company (*e.g.,* VINCI Autoroutes, the Bouygues Group). To improve the driving experience and public safety, these companies provide dig-

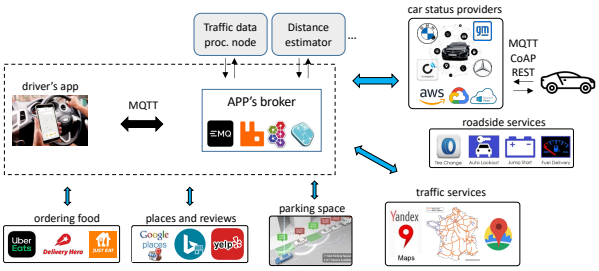

Figure 1: Examples of different constituent systems in the IoV context.

ital services for toll payment, real time traffic information, etc. In addition, roadside digital services related to parking capacity, charging/gas stations, are usually exposed as APIs. Many of the 30 million vehicles in France deploy a variety of sensors providing information related to vehicle characteristics (model, wheelbase, unladen weight) and status (speed, location, tire pressure) as well as trip and driver monitoring information (average fuel consumption, human/automated driver, alertness level). This has led to the emergence of the *Internet of Vehicles* (IoV) concept [8] which enables data exchange between vehicles and roadside services using diverse technologies. To develop IoV applications, it is essential to enable the exchange of data between heterogeneous vehicle sensors and highway/roadside digital services. For instance, charging an electrical vehicle requires 1-2 hours using fast-charging technology and upto 20 hours otherwise. Thus, proactive and deliberate planning is required especially on long trips. In particular, given the distance of a trip, the vehicle's battery level, speed, location and the driver's operating behavior, an IoV application, such the one shown in Figure 1, could anticipate and optimize the stop plan (charging stations, restaurants, etc.).

Existing cloud-based platforms, *e.g.,* AWS IoT Greengrass, are adopted by vehicle manufacturers to provide advanced car services such as parking assistant, concierge, live traffic information, etc. While such systems improve the driving experience, they are vertical and siloed solutions that depend on the technology stack leveraged by the vehicle manufacturer. Regardless of the platform used, such systems are data-centric where vehicle data must be collected on the cloud, processed and then provided to vehicle users. In contrast, distributed edge-based approaches favor several requirements of IoT systems such as privacy and security, timely message delivery, network latency and system functionality.

Enabling IoT composition. Creating an IoT deployment requires to consider numerous technical (Edge vs. Cloud technologies, protocols, APIs, QoS, etc.) and context (physical environment) related aspects. For instance, commercial street could host IoV deployments to: monitor vehicle traffic; track bus and tram locations; monitor harmful gases, smoke, and dust; to detect waste level for optimizing collection scheduling; and so on. Composing diverse IoT deployments may enable the design and fulfillment of higher-level operations that are not possible by any one IoT deployment alone. However, enabling the composition of an IoT deployment is a substantial undertaking as it requires each deployment to go beyond what it was originally designed for. This paper enables IoT composition via the *self-awareness* of the deployment's resources/capabilities and *meta-awareness* of the surrounding context.

3 `Hetero-Genius`

3.1 Architecture

The main components of `Hetero-Genius` are now described.

Constituent systems A constituent system represents a single system or device that provides one or more functionalities and has certain properties. Examples include parking services, restaurants, traffic control units, vehicles, etc. Each system exposes its APIs that enable other systems to interact with it and access its functionalities. In addition, each constituent system has a set of properties that specify constraints of service availability and provision. These include the messaging protocol, location, availability intervals, among others. Each constituent system is assumed to provide an ontological specification that describes its properties, functionalities, and binding information so that other systems can au-

tonomically find means to opportunistically interact with it.

Ontology manager In order to achieve automatic discovery and interaction of systems, there is a need for *constituent system ontology*; *i.e.,* semantics to capture the information of a given constituent system, such as APIs, properties, data types. There is also a need for a runtime to enable ontology creation and exploitation. In `Hetero-Genius`, the ontology manager provides the required semantics that are based on the standard CoDAMOS ontology [22]. The ontology manager provides interfaces for creating and parsing constituent system descriptions.

Holon registry (HR) An ontology registry provides storage of the constituent system specifications and enable their discovery. The registry stores an identifier, type, and location for each constituent system along with the detailed description. This enables various ways of looking up and aggregating systems and their services based on their properties.

Mediation synthesizer is designed to resolve interoperability issues that may arise from a mismatch between the messaging protocols of different systems. Upon the identification of such a mismatch, the Mediation Synthesizer creates a *mediator* that translates between an constituent system service and a consumer. Mismatch discovery takes place at runtime and is attainable by having the messaging protocols specified in the constituent system descriptions. Note that the placement of a mediator is an optimization problem in its own right, but it has been tackled by others, *e.g.,* [11].

Knowledge base is a data repository that stores real time information about the different constituent systems involved in the system such as vehicle fuel level and available parking slots. Each constituent system is identifiable by a unique identifier, and the knowledge bases provides interfaces to put and get data. The data are accessed by the constituent system services that are defined and published in the constituent system ontological description.

Workflow manager An abstract workflow represents the required application logic, using annotations to specify the required services. It also includes the types of systems that are required to execute the workflow. The workflow manager consults the ontology manager and ontology registry to select the required services using the constituent system types and service annotations. The selection of services results in an executable workflow that is passed back to the application broker for execution.

Application broker allows the user application to interact with the composite system (through the middleware). The broker receives user requests from the user application in the form of abstract workflows that are passed for concretization, which returns an executable workflow. The application broker then executes the workflow and returns the results to the user application.

3.2 System Composition

This subsection explains the ontology of constituent systems, the processes of discovering and selecting systems, and how we deal with interoperability issues.

3.2.1 Holon ontology.

Automatic IoT system composition requires devices to discover each other along with the functionalities they provide. This would in-turn require devices to advertise themselves. However, the enabler for device advertising and discovery is a definition using a uniform language. The definition needs to include all the concepts and properties of a device in order to provide its comprehensive description. For this, we build on our previous work in which we provided an ontological model in the literature that was designed to describe systems [4, 21, 12]. This *holonic* ontology is based on the CoDAMOS standard ontology [22], which is flexible and extensible for describing context-aware computing infrastructures. Concepts defined in the CoDAMOS ontology are based on the concept of *Thing*, which is an abstract concept to represent anything. Derived from this, concepts are then centered around four main concepts of *Service*, *Environment*, *Platform* and *User*.

The ***Service*** concept has a *service profile*, a *service model*, and a *service grounding*. The *service profile* contains information about the input and outputs of the service, the service provider and the quality of

the service. The *service model* contains information about data- and control-flow that occur when the service is executed. The *service grounding* contains information that are needed to interact with the service such as messaging protocols (e.g., an IoT protocol) and data formats. A system (*e.g.,* a smart device) can provide a certain service and another system can consume that service. The ***Environment*** concept defines context related information including location, time and environmental conditions. These information enables context-informed decision making. For example, a driver looking for a parking service would utilize the location information to select a nearby available parking spaces. The ***Platform*** concept contains information about the software that is installed on the device including the operating system, virtual machine and/or middleware. This also informs decision making *e.g.,* when systems need to interact using various communication technologies. Finally, the ***User*** concept contains information about the users of devices or systems.

The above concepts are considered upper classes of the ontology and are interconnected using ontological assertions that define relationships between the defined concepts. An example of such relationships is shown in Figure 2. In `Hetero-Genius`, we define each constituent system as a holon, implying that it is a stand-alone entity but can also be part of a whole system. Hence, a holon is a programmatic first-class entity that describes any unitary or composite system. Our system model includes an ontology for each device in the system. The holon description will then be utilized to automate and reason about system construction at runtime.

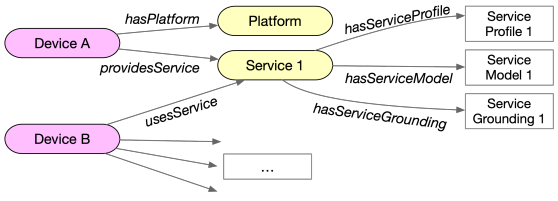

Figure 2: Examples of concept relationships in CoDAMOS.

3.2.2 Holon discovery

The holon ontology described above is the enabler for automatic discovery of systems and their functionalities. The different types of systems leverage the ontology semantics, which enables different system components to understand each other. This in turn enables all constituent systems to adopt the same mechanism to publish their services and properties, and to discover services and properties of other systems. For this purpose, the Holon Registry component provides a database where each system publishes their holon description in. This does not include real-time data of systems (*e.g.,* available spaces in a parking system), which are stored in the knowledge base component.

The description is required to be updated as the system properties and services evolve. For example, the introduction of new services of a certain type would require re-publishing the constituent system description to the registry in order to enable discovery of and access to those new services. Each constituent system in the holon has a unique ID and a tuple of `<type, location, holon>` in the registry. This enables service discovery and aggregation by ID, type and location. When a system/application requires access to the services of other systems, it queries the registry with the appropriate keys that are used by the registry to find an appropriate holon, which includes Holon ID, location, and type.

3.2.3 Workflow specification

A user application requires an abstract workflow that represents the sequence of functionalities that are required to satisfy the user goals. The workflow specification includes the system types that provide the required functionalities. The specified functionalities are independent of the physical constituent systems that exist in the composite system in the sense that users/developers do not need to know the specific APIs of the constituent systems. After providing an abstract workflow, it will then be the responsibility of the middleware (more specifically, the workflow manager) to concertize the workflow using the information contained in the system descriptions.

Figure 3 shows a simple example of an abstract

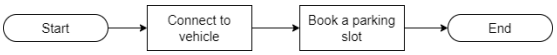

Figure 3: A simple abstract workflow from the IoV context.

workflow derived from the IoV scenario presented in Figure 1. The workflow that starts by connecting the Driver Application (DA) to the vehicle, which requires obtaining the holon of the vehicle by querying the HR using the car ID. Then the workflow involves booking a parking slot for the vehicle, which requires finding a parking service. This also requires the DA to read the vehicle data in order to complete the booking. It is worth emphasizing here that the APIs required to connect to the vehicle and to perform the booking are not known to the application developers. They are to be discovered at runtime using the ontological description of the vehicle and the parking system along with their service annotations.

3.2.4 Workflow concertizing

Upon receiving an abstract workflow, the ontology manager concretizes it, which includes selecting services to execute each task of the workflow and creating mediators (if needed) to facilitate the communication between the user application and the services.

The service selection process utilizes system types and annotations to concertize the abstract workflow. It executes the following:

1. The process starts by finding physical systems that provide the required services. For this purpose, the application broker queries the HR for descriptions of suitable systems, *e.g.,* location-based aggregation. The HR returns the holons of n systems that exist within a radius R of application broker and are of the required type.

2. The application broker parses the response to verify that the systems provide the required services by matching the annotations of the required services (specified in the workflow) and the annotations of the provided services (specified in the system's holon). It will then return the binding information of each service to the DA along with the messaging protocol of the corresponding system.

Upon receiving the service binding information and messaging protocols, the workflow manager detects if mediators are required for the interaction between the user application and the services. In cases where mediation is required, the workflow manager contacts the mediation synthesizer to synthesize mediators between the application and the services. The end result is a concrete (executable) workflow resulting from assigning the binding information to the abstract services of the workflow.

3.2.5 Addressing interoperability

IoT devices employ middleware-layer protocols such as MQTT, CoAP, ZeroMQ and more, to message each other. These protocols support different data-serialization formats (*e.g.,* JSON, XML, protobuf, etc.); different payloads suitable for resource-constrained or -abundant devices; and they follow different interaction paradigms such as client/server, publish/subscribe and data streaming.

IoT systems include heterogeneous IoT devices employing any combination of the above configurations. Moreover, sometimes new heterogeneous devices may be needed to be added to an IoT system in an on-demand fashion. For instance, in the IoV scenario, an end-user mobile application must interact with the services of, say, Mercedes-Benz that provide diagnostic and other information related to specific cars. Hence, automated solutions are required to enable data exchange in such IoT systems.

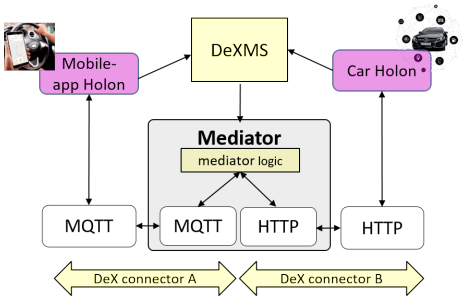

Figure 4: Data eXchange Mediator Synthesizer (DeXMS).

We represent each (e.g., a car, a mobile app, a restaurant, etc.) as a holon that provides *services* defined based on (possibly) heterogeneous profiles, models, and grounding concept. In this work, we rely on the Data eXchange Mediator Synthesizer (DeXMS) [5] to enable heterogeneous IoT devices to interact with each other. DeXMS addresses the heterogeneity of IoT devices and services by synthesizing software mediators. As depicted in Figure 4, the grounding of services is given as input to DeXMS. This includes information such as IoT protocol, data model, data format and operation (or topic) semantics. DeXMS accepts these descriptions, and provides back a concrete mediator for bridging purposes.

3.2.6 Workflow execution

Finally, the application broker orchestrates the selected services to execute the concrete workflow. The application broker executes the services and maintains the state of execution. It makes calls to the required services from the beginning to the end including the required mediation services based on control and data dependencies.

The sequence diagram depicted in Figure 5 illustrates the interactions between the driver application and the system components to execute the simple workflow shown in Figure 3.

4 IoV System Composition: a use case

This section exploits the motivating scenario introduced in Section 2 to demonstrate `Hetero-Genius`. We first demonstrate the ontological descriptions of the main systems of the scenario, then report on a case study to evaluate `Hetero-Genius`'s effect on developer productivity in constructing a relatively complex IoV system.

4.1 System description using holons

The main systems involved in our motivation scenario are: car, parking service, restaurant and traffic control unit (TCU).

A car is a holon. The *fuelStatus* and *location* are two services that are defined in the ontology. The Car concept is linked to the *fuelStatus* and *location* services via the *provideService* axiom. Each of these services exposes a *URL*, and takes input and returns values. The *URL* is a class in the ontology that is linked to the services using the *exposes* axiom. The ID input and the return values are of class *Parameter* and linked to the service via the *hasParameter* and *returns* axioms. The Car concept is also attached to the Environment concept via the *hasEnvironment* axiom. The Environment concepts is attached to the Location concept via the *hasLocation* axiom. The Location concept defines the location property of the car. Figure 6 illustrates a simplified version of the described ontology.

Similar ontology structure applies to the other constituent systems. A restaurant is a holon, which provides a *details* service and a *menu* service. The *details* service takes a restaurant ID and returns details of the restaurant including location, contact details, and average serving time, among others. These variables are needed to inform the decision making of selecting a convenient restaurant for the car driver to make orders from. The *menu* service returns the menu details including the menu items and prices. The TCU is a holon that provides information about the traffic status. It exposes *trafficStatus* service which takes a location and a direction and returns traffic status. Finally, the Mobile App is a holon that exposes an interface for a user to interact with the system. The Mobile App holon is attached to a Platform concept that defines the mobile operating system. The Mobile App exposes a *search* service which searches for a required service (*e.g.,* food or parking) and order service to submit orders to those service.

As depicted in Figure 4, DeXMS uses the Data eXchange (DeX) API, which implements `post` and `get` primitives for sending and receiving messages using existing IoT protocols such as CoAP, MQTT, XMPP, etc. In Figure 4, the mediator converts temperature data coming from a package (in JSON format through the HTTP protocol) to be received from a system's dashboard (in XML format through the MQTT protocol). Considering a set of heterogeneous IoT de-

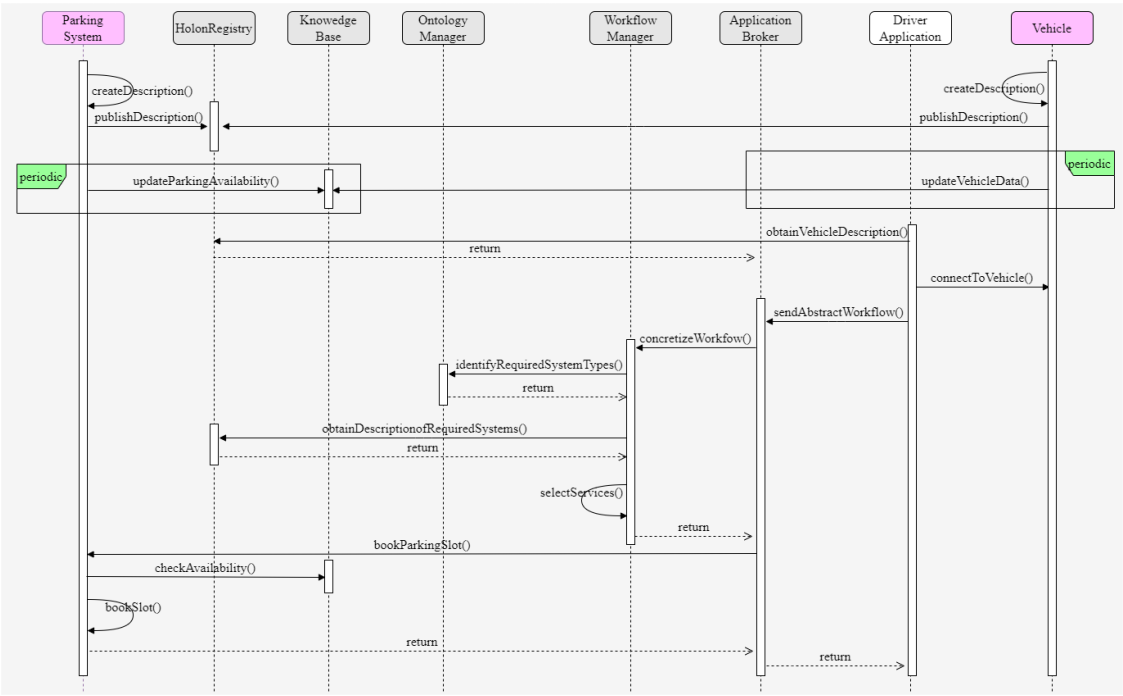

Figure 5: Sequence diagram of the workflow shown in Figure 3.

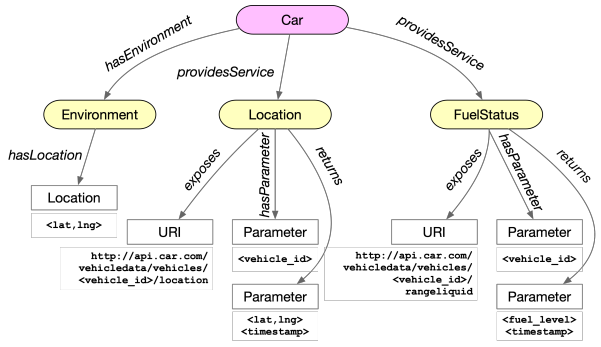

Figure 6: A simplified ontology of the car holon in the IoV scenario.

vices that have to interconnect with devices deployed in an IoT systems, DeXMS accepts as input their input/output data representation models and synthesizes the required mediators. More details on DeXMS can be found in [5].

4.2 Experimental design

Objectives. The experiment aims at evaluating developer productivity, in terms of the time required to develop application logic, and correctness, in terms of adherence to the usage of the correct APIs in the implemented logic.

Strategy. The experiment compares our approach against the manual construction of systems where developers need to manually and repeatedly inspect system APIs to find the appropriate definitions and protocols to integrate an application logic. We adopt a controlled experiment approach where participants are given a simple workflow to implement. The workflow is designed to be simple so that participants can implement it within a reasonable time frame.

Procedure. The experiment procedure lasts for up to an hour per participant. Participants are assigned the same workflow to implement but randomly assigned to implement it using holons or the provider APIs. Each participant fills a questionnaire about

their level of education, experience in systems programming, and API- and IoT-based application development.

Task. Participants were given a workflow along with the holon APIs and a list of provider APIs (*e.g.,* restaurant APIs, vehicle APIs and fuel station APIs). Those assigned the holon APIs were required to leverage holon annotations to develop an abstract workflow (shown in Figure 7) that is to be passed to the application broker for concretization. They need to find the holon annotations that correspond to the required functionality of the system type. On the other hand, participants assigned the provider APIs were required to find the specific URLs and methods of the required functionality of the system type.

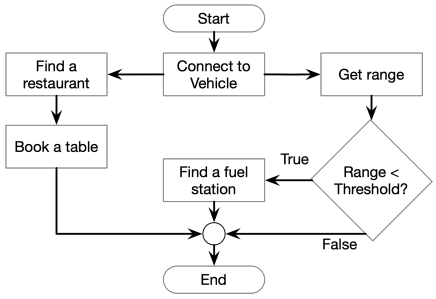

Figure 7: The IoV workflow used in the experiment.

Recruitment. Participants were recruited from computer science researchers and students at Lancaster University, University of Glasgow and Télécom SudParis / IP Paris as well as from developers at local startups and incubators. Each participant was given an incentive in the form of an Amazon shopping voucher. Overall, 26 participants were recruited with varying expertise levels as shown in Figure 8.

4.3 Productivity results

The productivity of participants is evaluated by measuring the time spent to implement the given task once implementation commences (*i.e.,* after reading the experiment consent information and guidelines). The productivity results obtained from applying both the classical and `Hetero-Genius` approaches are shown in Figure 9. We observe that participants

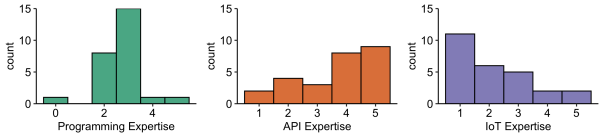

Figure 8: Self-reported expertise levels of the participants (5 is the highest).

spent an average of 43.82 minutes and a median of 45.3 minutes for completing the task using the classical approach and an average of 23.25 minutes and a median of 20.13 minutes for completing the task using the `Hetero-Genius` approach. The improvement in productivity varies, probably depending on the level of programming expertise of each participant. Overall, `Hetero-Genius` improves the productivity of system developers by 46.94%.

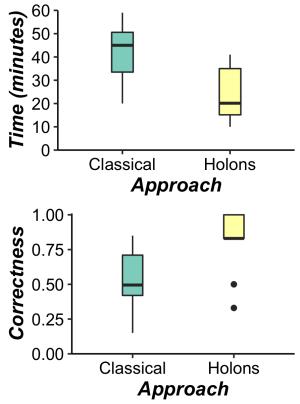

Figure 9: The time taken to accomplish the experiment task (l) and the correctness scores (r) following each approach.

4.4 Correctness results

We asses the correctness of the code developed by the participants by manually inspecting and testing the developed code in terms of (i) correct usage of the required APIs and (ii) adherence of the developed logic to realizing the given workflow. A correctness score has been given to each participant code. This has

been calculated by assigning one point to each correct API and one point to the milestone code statement then the total was divided by the maximum number of points resulting is a score out of 100%. The correctness results obtained from applying both the classical approach and the `Hetero-Genius` approach are shown in Figure 9. The correctness score of participants using native APIs has a mean of 54.25% and a median of 49.5%. In contrast, using `Hetero-Genius` results in correctness scores with a mean of 84.35% and a median of 83%. This outcome indicates that the `Hetero-Genius` approach improves the correctness of developing applications by 55% on average.

5 Related Work

This section presents existing solutions for system composition and the interoperability of IoT devices. In addition, a comparison of the `Hetero-Genius` approach against existing solutions is provided, and summarized in Table 1.

System composition. Composing Systems of Systems (SoS) was initially proposed by Maier [19]. SOA composition technologies can be applied to form a SoS [29, 3, 1]. We argue that such automation is not suitable for three reasons. First, SOA does not readily provide concepts that capture physical system properties such as the context they are operating in, which is required for reasoning about SoS composition. Second, services in SOA are assumed to be published in a registry; this is not valid in a fully distributed context with no prior knowledge of other systems. Third, SOA-based SoS developers are expected to know a lot about individual systems as demonstrated by our qualitative experiments.

Non-SOA efforts facilitate SoS discovery and composition, *e.g.,* using cellular infrastructure [15]. However, such approaches make reasoning assumptions about discovered systems. Some works have chosen to do this at design time through analyzing qualitative mission objectives of systems (*e.g.,* [20]), but these are hard to express in a programmatic way that enables automated reasoning at runtime. Other works have also defined the notion of a holonic system [16, 18, 24], but are focused on goal-driven service composition without means of allowing holons to reason and self-compose.

Table 1: A comparison of related works.

	Auto. Composition	Self Reasoning	Mediator Synthesis	Supported Protocols	IoT-enabled
SOA-based SoS [29, 3, 1]	✔	x	x	SOAP	x
Fodor et al. [15]	✔	x	x	Network level	x
Mokhtarpour et al. [20]	x	x	x	N/A	x
Holonic system [16, 18, 24]	✔	x	✔	HTTP	x
HTTP-CoAP Proxy [7, 6]	x	x	x	CoAP, REST	✔
Derhamy et al. [10]	x	x	x	CoAP, HTTP, MQTT	x
XWARE [23]	x	x	x	N/A	✔
SemIoTic [31]	x	x	✔	CoAP, MQTT	✔
Hetero-Genius [this]	✔	✔	✔	CoAP, HTTP, MQTT	✔

IoT interoperability. To compose IoT systems at runtime, it is essential to address the heterogeneity between involved IoT systems that employ diverse protocols, APIs, and data representations for exchanging data. Several approaches to bridge middleware-based protocols have been proposed, *e.g.,* the QEST broker for CoAP and RESTful APIs [7], HTTP-CoAP proxy [6], etc. These approaches implement one-to-one mappings between existing protocols, however, this is highly inefficient due to the vast development of IoT protocols. Derhamy et al. [10] and XWARE [23] use intermediate formats; the former by introduceing a protocol translator while the latter by implementing IoT mediators. Finally, [5] deals with IoT heterogeneity using software abstractions and code generation.

While the above approaches reduce the develop-

ment effort, they do not consider semantic layer incompatibilities prevalent in the IoT, *e.g.,* operation/resource names, data semantics, etc. Ontologies [17] provide a common model for annotating content and thus help systems to interoperate. Depending on the application domain, developers can use the appropriate ontology; *e.g.,* SAREF [9] models appliance information in smart homes. IoT platforms such as SemIoTic [31] provide end-to-end IoT interoperability in smart buildings by leveraging the SSN/SOSA ontologies and mediating adapters. IoT-Stream [13] presents a lightweight model and associated system architecture to semantically annotate IoT data streams.

6 Discussion and Concluding Remarks

Reflecting on the research direction

The difficulty in building integrated systems lies largely in the heterogeneity of modern IoT ecosystems. In this work, we initially set out trying to answer the question: *Could building integrated systems be achieved using existing software engineering practices?* The answer to this is naturally 'yes, but at a relatively high cost' as evidenced by our own experience along with that of our experiment participants, in addition to related findings reported in the literature [27, 28]. Developing using a myriad of divergent APIs and protocols is time-consuming and prone to errors. This is particularly true in IoT and CPS contexts where heterogeneity and fast pace of deployment are common.

Subsequently, we explored the question: *How can we improve software engineering practices in order to make development of integrated systems easier and more reliable?* In order to achieve this, we sought means of reasoning about building integrated systems and putting in place mechanisms in order to facilitate such automatic composition. As a result, we designed and implemented the `Hetero-Genius` architecture to leverage ontological descriptions of independent IoT systems to discover their APIs and protocols; as well as to automatically integrate independent systems into an integrated one using an abstract workflow. Interoperability between independent systems is achieved using mediators that translate different data exchange protocols.

Experimental outcomes

We implemented the `Hetero-Genius` architecture in Java and evaluated it using subjective user experiments that involved 26 developers of varying levels of experience. The evaluation results show that our approach reduces development overhead by 47%, on average. Along with improved productivity, correctness of the developed code improved by 55% on average. In addition, we also assessed the feasibility of `Hetero-Genius` at scale using controlled simulations. We found the real-time overheads to be realistically feasible with performance exhibiting a linear trend. However, these results are not included for space limitations.

Limitations

Defining an ontology to describe any IoT system is quite a challenge, with a tradeoff between generality and usefulness. Our intention is *not* to define "one ontology to rule them all". Moreover, we are aware that any ontology will have shortcomings.

Furthermore, we do not expect that all IoT developers and operators to unite on using our holonic ontology. There is an overhead in adopting such a semantic tool, which adds to the resistance to wide adoption. Nevertheless, the overhead is not insurmountable as evidenced by our experiments where developers with no experience in holons were able to use it with relative ease and effectiveness.

In addition, we do not assume that constituent systems would readily provide ontological specifications; nor do we envision this. Instead, we see that deriving such specification using programmatic means is an achievable objective and are currently working on doing so to relax such fundamental constraint that underlies our work.

Future work

We are currently working on using NLP and ML tools to automatically generate the holon description of any given IoT device. We will next target the providing of tools for defining domain-specific semantics, high-level description of systems and abstract application logic that will integrated in a complete solution for automatic construction of integrated systems.

Acknowledgments

This work was partly supported by the UK EPSRC under grant number EP/R010889/2.

References

[1] Aschoff, R. R., and Zisman, A. Proactive adaptation of service composition. In *Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)* (2012), ACM.

[2] Banks, A., and Gupta, R. MQTT version 3.1.1. Standard, OASIS, 2014.

[3] Baresi, L., and Pasquale, L. Live goals for adaptive service compositions. In *Workshop on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)* (2010), ACM, pp. 114–123.

[4] Blair, G., Bromberg, Y.-D., Coulson, G., Elkhatib, Y., Réveillère, L., Ribeiro, H. B., Rivière, E., and Taïani, F. Holons: Towards a systematic approach to composing systems of systems. In *Workshop on Adaptive and Reflective Middleware (ARM)* (2015), ACM.

[5] Bouloukakis, G., Georgantas, N., Ntumba, P., and Issarny, V. Automated synthesis of mediators for middleware-layer protocol interoperability in the IoT. *Future Generation Computer Systems 101* (2019), 1271 – 1294.

[6] Castellani, A. P., Fossati, T., and Loreto, S. HTTP-CoAP cross protocol proxy: an implementation viewpoint. In *Int. Conf. on Mobile Ad-Hoc and Sensor Systems (MASS)* (2012).

[7] Collina, M., Corazza, G. E., and Vanelli-Coralli, A. Introducing the QEST broker: Scaling the iot by bridging MQTT and REST. In *Int. Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC)* (2012).

[8] Contreras-Castillo, J., Zeadally, S., and Guerrero-Ibañez, J. A. Internet of vehicles: Architecture, protocols, and security. *IEEE Internet of Things Journal 5*, 5 (2017), 3701–3709.

[9] Daniele, L., den Hartog, F., and Roes, J. The Smart Appliances REFerence (SAREF) Ontology. In *Workshop on Formal Ontologies Meet Industries* (2015).

[10] Derhamy, H., Eliasson, J., and Delsing, J. IoT interoperability – on-demand and low latency transparent multiprotocol translator. *IEEE Internet of Things Journal 4*, 5 (2017).

[11] Elhabbash, A., and Elkhatib, Y. Energy-aware placement of mediation services in IoT systems. In *International Conference on Service-Oriented Computing (ICSOC)* (Nov. 2021).

[12] Elhabbash, A., Nundloll, V., Elkhatib, Y., Blair, G. S., and Sanz Marco, V. An ontological architecture for principled and automated system of systems composition. In *Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)* (2020).

[13] Elsaleh, T., Enshaeifar, S., Rezvani, R., Acton, S. T., Janeiko, V., and Bermudez-Edo, M. IoT-Stream: A lightweight ontology for Internet of Things data streams and its use with data analytics and event detection services. *Sensors 20*, 4 (2020).

[14] Erl, T. *SOA Principles of Service Design*. Prentice Hall PTR, 2007.

[15] Fodor, G., Dahlman, E., Mildh, G., Parkvall, S., Reider, N., Miklós, G., and Turányi, Z. Design aspects of network assisted device-to-device communications. *IEEE Communications Magazine 50*, 3 (March 2012), 170–177.

[16] Frey, S., Diaconescu, A., Menga, D., and Demeure, I. A holonic control architecture for a heterogeneous multi-objective smart microgrid. In *Conference on Self-Adaptive and Self-Organizing Systems (SASO)* (2013), pp. 21–30.

[17] Gruber, T. R. A translation approach to portable ontology specifications. *Knowledge Acquisition 5*, 2 (1993).

[18] Kit, M., Gerostathopoulos, I., Bures, T., Hnetynka, P., and Plasil, F. An architecture framework for experimentations with self-adaptive cyber-physical systems. In *Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)* (2015), ACM, pp. 93–96.

[19] Maier, M. W. Architecting principles for systems-of-systems. *Systems Engineering 1*, 4 (1998), 267–284.

[20] Mokhtarpour, B., and Stracener, J. A conceptual methodology for selecting the preferred system of systems. *IEEE Systems Journal 11*, 4 (Dec 2017), 1928–1934.

[21] Nundloll, V., Elkhatib, Y., Elhabbash, A., and Blair, G. S. An ontological framework for opportunistic composition of iot systems. In *Conference on Informatics, IoT, and Enabling Technologies (ICIoT)* (Feb. 2020), IEEE.

[22] Preuveneers, D., Van den Bergh, J., Wagelaar, D., Georges, A., Rigole, P., Clerckx, T., Berbers, Y., Coninx, K., Jonckers, V., and De Bosschere, K. Towards an extensible context ontology for ambient intelligence. In *European Symposium on Ambient Intelligence* (2004), Springer, pp. 148–159.

[23] Roth, F. M., Becker, C., Vega, G., and Lalanda, P. XWARE - A customizable interoperability framework for pervasive computing systems. *Pervasive Mob. Comput. 47* (2018).

[24] Sabatucci, L., Lodato, C., Lopes, S., and Cossentino, M. Highly customizable service composition and orchestration. In *Service Oriented and Cloud Computing* (2015), S. Dustdar, F. Leymann, and M. Villari, Eds., Springer International Publishing, pp. 156–170.

[25] Saint-Andre, P. Extensible messaging and presence protocol (XMPP): Core. RFC 3920, IETF, 2004.

[26] Shelby, Z., Hartke, K., and Bormann, C. The constrained application protocol (CoAP). RFC 7252, IETF, 2014.

[27] Sinche, S., Raposo, D., Armando, N., Rodrigues, A., Boavida, F., Pereira, V., and Silva, J. S. A survey of IoT management protocols and frameworks. *IEEE Communications Surveys & Tutorials 22*, 2 (2020), 1168–1190.

[28] Sobin, C. C. A survey on architecture, protocols and challenges in IoT. *Wireless Personal Communications 112*, 3 (2020), 1383–1429.

[29] Ye, X. Towards a reliable distributed web service execution engine. In *International Conference on Web Services (ICWS)* (2006), IEEE Computer Society, pp. 595–602.

[30] Younes, W., Trouilhet, S., Adreit, F., and Arcangeli, J.-P. Towards an intelligent user-oriented middleware for opportunistic composition of services in ambient spaces. In *Workshop on Middleware and Applications for the Internet of Things (M4IoT)* (2018), ACM, p. 25–30.

[31] Yus, R., Bouloukakis, G., Mehrotra, S., and Venkatasubramanian, N. Abstracting interactions with IoT devices towards a semantic vision of smart spaces. In *Conf. on Systems for Energy-Efficient Buildings, Cities, and Transportation* (2019).