Future Generation Computer Systems 157 (2024) 499–515

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

Principled and automated system of systems composition using an ontological architecture

Abdessalam Elhabbash [a,*], Yehia Elkhatib [b], Vatsala Nundloll [a], Vicent Sanz Marco [c], Gordon S. Blair [d]

[a] School of Computing and Communications, Lancaster University, UK
[b] School of Computing Science, University of Glasgow, UK
[c] National Institute of Advanced Industrial Science and Technology, Japan
[d] Centre for Ecology and Hydrology, Lancaster Environment Centre, Lancaster, United Kingdom

ARTICLE INFO

Keywords:
System of systems
Self-adaptation
Autonomous systems
Ontology
Runtime composition
Internet of things

ABSTRACT

A distributed system's functionality must continuously evolve, especially when environmental context changes. Such required evolution imposes unbearable complexity on system development. An alternative is to make systems able to self-adapt by opportunistically composing at runtime to generate *systems of systems* (SoSs) that offer value-added functionality. The success of such an approach calls for abstracting the heterogeneity of systems and enabling the programmatic construction of SoSs with minimal developer intervention. We propose a general ontology-based approach to describe distributed systems, seeking to achieve abstraction and enable runtime reasoning between systems. We also propose an architecture for systems that utilizes such ontologies to enable systems to discover and 'understand' each other, and potentially compose, all at runtime. We detail features of the ontology and the architecture through three contrasting case studies: one on controlling multiple systems in smart home environment, another on the management of dynamic computing clusters, and a third on autonomic connection of rescue teams. We also quantitatively evaluate the scalability and validity of our approach through experiments and simulations. Our approach enables system developers to focus on high-level SoS composition without being constrained by deployment-specific implementation details. We demonstrate the feasibility of our approach to raise the level of abstraction of SoS construction through reasoned composition at runtime. Our architecture presents a strong foundation for further work due to its generality and extensibility.

1. Introduction

Computing systems have advanced from a combination of connected devices to a more intricate arrangement of interconnected collections of heterogeneous systems, such as Internet of Things (IoT) [1,2], smart grids [3], ad-hoc networks (MANETS, VANETs, FANETs) [4], and others. Each of these types of systems comprises a number of heterogeneous components that collectively implement diverse functions and communicate using different protocols. Furthermore, these various systems often need to interact and collaborate to achieve their objectives. For example, isolated rescue teams need to access each other's services, such as data look-up; robots and drones need to coordinate to cover a deployment areas with minimal effort; environmental IoT systems process their data on a micro-cloud in their vicinity and offload certain processing to the cloud when needed; and so on. In this sense, larger systems are constructed through the interaction of smaller ones, a practice known as *System of Systems (SoS) composition* or *construction* [5].

An SoS thus involves a number of constituent systems (CSs). According to the level of authority the SoS has on the CSs, SoSs can be grouped into four types, *directed*, *acknowledged*, *collaborative* and *virtual*. A *directed* SoS is designed to achieve SoS-level goals. CSs do not have their own objectives and they all work to achieve the SoS goals. In an *acknowledged* SoS, CSs have their own independent goals but they contribute to specified SoS-level goals. In *collaborative* SoSs, CSs agree on a shared SoS-level goal and contribute to achieving it. A *virtual* SoS does not have either specified or agreed goals and its behavior is emergent at runtime [5,6].

However, the inherent characteristics of SoSs make their development challenging [7,8]. First, an SoS is by definition a complex system built from a number of sub-systems that in turn are made

* Corresponding author.
E-mail addresses: a.elhabbash@lancaster.ac.uk (A. Elhabbash), yehia.elhabbash@glasgow.ac.uk (Y. Elkhatib), vatsala@lancaster.ac.uk (V. Nundloll), v.sanzmarco@cmc.osaka-u.ac.jpk (V. Sanz Marco), g.blair@lancaster.ac.uk, gblair@ceh.ac.uk (G.S. Blair).

https://doi.org/10.1016/j.future.2024.03.034
Received 6 September 2023; Received in revised form 21 February 2024; Accepted 22 March 2024
Available online 10 April 2024

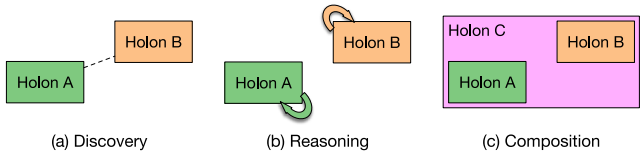

Fig. 1. An overview of how holons opportunistically compose to form more complex holons.

up of different components, and so on. Second, SoSs are typically deployed in environments where the context changes. This is where the need arises to veer from a strict operational plan that has been defined at design time, and form a more complex system in order to maintain the intended abstract behavior or to provide new behavior that is only possible by uniting with other systems under the new circumstances. Examples include adaptive IoT applications, volunteer and crowd computing, disaster recovery, military defense operations, and other forms of cyber–physical systems. Third, the development of SoSs is dynamic in nature. This is grounded in the knowledge that systems tend to be persistent and long-living [9] and, as such, their objectives and functionalities evolve over time as they are constantly added, modified, or removed at different time scales. This could also predicate changes in architectural and functional dependencies.

Current approaches to constructing SoSs assume that the developer has in-depth knowledge of the internal structure of each system and its components [10–13]. Considering the aforementioned challenges and characteristics of SoSs, it is evident that such approaches are deficient. We argue that SoS construction needs to be *autonomous* and *dynamic*, relying on systematic approaches to attain internal and context awareness. All systems should be able to accumulate knowledge about their own structure and behavior. Then, systems should exchange and be able to understand such knowledge at runtime so that they can opportunistically compose and form complex SoSs.

To achieve this form of behavior, there is a need for semantically and comprehensively expressing the system structure, capturing the information required for dynamically composing systems such as those relating to communication (*e.g.,* unique identifiers), service discovery, quality of service, physical properties (*e.g.,* power level and location), environmental properties (*e.g.,* temperature and pressure), etc.. We refer to this comprehensive structure description of a system as a *holon* [14]. A holon is constantly modified to reflect the system structure and contain any new or modified information, and then published to aid discovery and reasoning about composition. When holons compose, they form a new *super*-holon that represents the SoS. This newly constructed holon will have its own specification, which is published so it can further compose with other SoSs, and so on; see Fig. 1. In this manner, the holon concept forms the basis of an ecosystem where systems are able to opportunistically interact based on context, and compose to create more complex holons arising spontaneously in a bottom-up manner.

The holon ecosystem requires means of utilizing system description to reason about SoS construction at runtime in a programmatic and adaptive manner. Our solution is to adopt ontologies to capture holon specification at design time and as it evolves post-deployment. Ontologies are engineering tools for formal and explicit specification of a shared conceptualization [15]. They provide vocabularies to represent knowledge that can be utilized programmatically to understand the corresponding content, such as system parameters, offered services, and requirements. As these are expressed through standard concepts, they allow other holons to understand the described holon and make decisions about how to interact with it.

In [14] we proposed a vision towards the construction of distributed SoSs using holons. Then, in [16], we defined the basic conceptual structure of a holon with particular focus on self-describing IoT systems. In [17], we proposed an ontology-based approach for the dynamic and autonomous construction of distributed SoSs. In essence, once a holon is specified using our ontology, its description can be broadcasted to other systems that can in turn compile this description to understand the functionality of the sending system, reason about it, and determine how to communicate with it. In this article, we investigate how ontologies can be used to capture the holon specification, we elaborate on the architecture for constructing and composing SoSs, we offer an expanded set of use cases as well as a quantitative overhead evaluation, and we present a detailed discussion of the findings. The use cases demonstrate the feasibility of the using the holons for the construction of different types of SoSs. Our contributions in this article are the following:

1. A framework for enabling opportunistic SoS composition. This works by supporting holon specification, compilation, dissemination, and modification.
2. A qualitative evaluation of this holonic framework using three different case studies that emphasize the dynamic needs of SoS construction and configuration.
3. A quantitative evaluation of the scalability and validity of the proposed framework.

The remainder of this article is organized as follows. In Section 2, we introduce the preliminaries of SoS composition and describe the problem space. In Section 3, we outline our research methodology and research questions. Section 4 presents the architectural overview, which starts with the concept of the holon as a design primitive and its lifecycle. Section 5 expands on the specification of a holon using an ontological model that we devised by adopting and extending well known ontologies. Section 6 expounds the mechanics of using our holonic ontologies in order to reason about their composition to form an SoS at runtime. We then present three detailed case studies of using our framework in various contexts: a smart home of composite nature in Section 7, managing a computational cluster of unstable nature in Section 8, and establishing impromptu rescue teams under disaster conditions in Section 9. We provide an experimental feasibility study of our framework in Section 10. We provide reflective discussions in Section 11, provide a future outlook in Section 12, and conclude in Section 13.

2. Related work

The concept of SoSs has been featured in the literature for over two decades *cf.* [5]. Different approaches were proposed to construct SoSs. The significant body of work looming here is the Service Oriented Architecture (SOA) legacy. SOA allows a system to expose its functionalities as services, and then SOA composition technologies can be applied to form an SoS; *cf.* [18–20]. We argue that such SOA-based automation is not suitable for the following reasons. First, SOA does not readily provide concepts that capture physical system properties such as the context they are operating in. The service composition supported by SOA is mainly functionality-based composition. However, SoS composition is wider than this as it also requires knowledge about system properties and context to reason about the composition, especially in cyber–physical systems. Second, services in SOA are assumed to be published in a repository that an orchestrator can consult to select services from. This assumption is not valid in the general SoS context which is fully distributed. Third, SOA-based SoS developers are expected to know a lot about individual systems, or alternatively invest significant time learning them. Fig. 2 outlines the tasks that need to be performed by vendors, system developers, and the system for SOS composition. Developers are required to acquire knowledge about the APIs and properties of the elementary systems they need to compose. Then, they need to leverage that knowledge to design and implement the SoS, and finally deploy it. This requires the developer to focus on the elementary systems instead of the whole SoS. Efforts in this direction (*e.g.,* [21]) typically build a model (*e.g.,* a graph) to describe systems and their interactions. Then, contextual data is used to respond

to queries initiated by users to build an SoS at runtime by composing services that abstract the sought-after system functionalities.

Non-SOA efforts include facilitating discovery and composition using cellular infrastructure [22] and network middleboxes [23]. Such approaches, however, require specific infrastructure deployment for mediation and, more importantly, still assume too much on the part of the developer in terms of reasoning about the discovered systems. This final task is crucially difficult without identifying the modus operandi of the systems. Some studies have chosen to do this at design time (*e.g.,* [24]) by analyzing the qualitative mission objectives of systems, but these are difficult to express in a programmatic manner that enables automated reasoning at runtime.

Other studies that define a notion similar to that of a holon include [25–27]. However, these studies are focused on goal-driven service composition without means of allowing systems to reason and self-compose. Other works [28–30] extend the holonic approach to enable systems to reason about self-composition based on a captured context. The basic idea is to continuously resolve conflicts between low-level goals such that when these low-level goals are achieved, a high-level goal is also achieved. Nevertheless, these models still lack support for a comprehensive representation of systems (*e.g.,* including physical properties and reliability requirements), which limits system capabilities to reason about interaction with other systems to form an SoS.

Agent-based ontological approaches have been proposed in which each agent is defined by a set of static and dynamic aspects. [31] created a unified ontology that borrows concepts from a number of existing ontologies that are used to describe resources and data formats in IoT platforms. [32] proposed a framework to support development of IoT applications, leveraging basic ontology to describe sensor data and identify emerging events. In the context of adaptive IoT applications, [33] created an ontology-based knowledge base to represent behavior of system elements in addition to data generated from sensors. The purpose is to provide a shared source of information to enable adaptive IoT applications. Other examples include [34–36]. These studies are less systematic in their approach than our holon ontology. Moreover, they do not indicate how composition is reasoned about in a heterogeneous environment.

3. Problem space and research strategy

Composing SoS from existing systems at runtime is still an area that requires further research [14,37,38]. An open challenge is achieving the construction and adaptation of systems and composition functionalities with minimal developer effort. The current practice of SoS composition relies on manually composing elementary systems at design time. Unfortunately, this requires SoS developers to acquire knowledge about the internal properties of each system, such as their functionalities (*e.g.,* services) and vendor-specific APIs used for interactions. For example, consider the dimensionality and heterogeneity of IoT applications. Different vendors constantly produce devices using specific technology stacks [39]. This results in a large space of heterogeneous elementary systems that need to compose. Such challenges of heterogeneity and dimensionality make the design-time SoS composition approach time-consuming, error-prone, and beyond human capabilities [2]. Moreover, the dynamicity of current computing environments requires the runtime adaptation of SoSs by recomposing elementary systems, which adds to the complexity of the process.

In addition, it is important to distinguish that the holonic approach is a **bottom-up** approach that aims to *describe*, as opposed to the numerous **top-down** architectures that pursue to *prescribe*, *e.g.,* ThingML [40], CDOnto [41], FloWare [42], and others. Some of these ontologies (*e.g.,* SAREF [43]) are very domain-focused and not suitable for general IoT systems.

Based on the above, our ultimate goal is to shift the developers' focus from learning the internals of elementary systems to thinking at the level of SoSs. The aftermath of describing high-level SoS logic (as is shown in Fig. 2) is that elementary systems would autonomously discover and compose with each other. The achievement of this goal requires (1) a comprehensive description of atomic systems (their services, properties, and context) that enable autonomous composition between systems; and (2) an architecture that exploits such descriptions and supports SoS composition and adaptation.

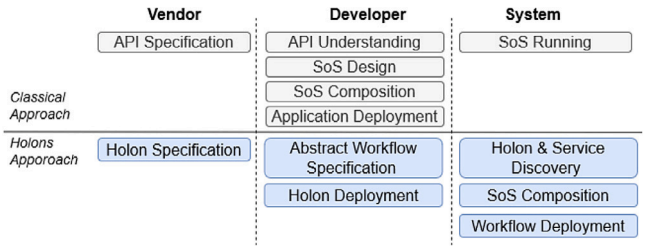

Fig. 2. Comparing the responsibilities in the current SoS composition approaches, and in the proposed approach.

We seek to answer the following general research questions.

RQ1: What are the right abstractions to represent different systems within an SoS framework?
RQ2: What systems principles and techniques are required to support SoS composition?
RQ3: Which extensions are needed for SoS composition and adaptation in heterogeneous, large-scale environments?

We propose an ontological approach towards real-time reasoning around the composition of systems of systems. This is made possible by an architecture we have built for comprehending what an SoS is made up of, if composition is required, and how such composition would take place. Moreover, the architecture automatically actions the low-level mechanics that enable the composition. As such, we are assessing our proposal's fit against the above research questions. Therefore, we pose additional research questions to investigate the feasibility and utility of our proposed solution.

RQ4: What is the cost of adopting the proposed holonic approach specifically in terms of reasoning about and actioning system composition at runtime?
RQ5: Are the outcomes of holonic composition correct in terms of meeting the developer's high-level SoS logic?

In our study, we adopt a hybrid experimental evaluation strategy: We conduct qualitative appraisals in controlled testbed experiments of various natures, specifically in the domains of IoT, emergency communication, and infrastructure management. We augment this by quantitatively assessing real open-source systems to investigate how our architecture can enable developers to easily build the next generation of adaptive applications.

4. The holonic lifecycle

A holon starts as an atomic one, which might then evolve to being a composite one. An *atomic holon* represents a single system that provides one or more functionalities. A *composite holon* includes functionalities provided by a number of systems interacting with each other either directly or through others. For holons to compose, atomic holons need to be comprehensively specified by the system developer. After that, the system will be dynamically constructed as the holons iterate in their lifecycle. The four stages of a holon's lifecycle (Fig. 3) are outlined below and described in more detail in the following two sections.

Specification. A system developer uses an ontology to create the holon from a number of elementary descriptions. This includes a holon

Fig. 3. A holon's lifecycle.

identifier, the physical properties of the system (*e.g.,* power level), quality of service (*e.g.,* availability and reliability), environmental properties (*e.g.,* location), policies (*e.g.,* routing protocol), services (*e.g.,* sensing temperature), among others. All these are specified using our ontological model, described in Section 5. For our purposes, we assume that device manufacturers undertake the task of defining the atomic holons that describe their devices. We will later revisit this assumption.

Dissemination. A created holon disseminates its specification so that it can be discovered by others. Different dissemination strategies can be adopted: push, pull, and lookup. In the *push* strategy, the holon periodically broadcasts its latest description. In contrast, a *pull* strategy uses heartbeat signals; *i.e.,* the holon periodically sends 'Hello' messages to establish interaction with other systems, which could then request the full holon description. This strategy reduces overhead, so it is useful in energy-constrained environments. In the *lookup* strategy, the holon registers its description with a registry that can be consulted by other systems to obtain it. This strategy is used when infrastructure assistance is guaranteed. We adopt the push strategy for the scope of this article. This achieves the goal of keeping holons proactively aware of changes in other holons with regard to their properties and the services provided. However, we leave the development and comparison of other strategies for future work. One other thing to discuss is the time frame of dissemination, which is a parameter that impacts both performance and consistency. A large time frame of dissemination would reduce the overhead of processing the holons but would slow conveying changes in the entire system. Intuitively, a small time frame would have the opposite impact. However, the problem of specifying the dissemination time frame is a classical problem that has been extensively discussed in the literature on distributed systems. The use of a protocol that adaptively adjusts the time frame is one attractive solution [44].

Compilation. This aims to understand other holons and their functionalities. This is achieved by parsing the received descriptions (in the form of XML representations), identifying the functionalities contained within, and how they interact with that of the received holon. Compiling these functionalities into one holon represents a new SoS comprising of the interacting holons. At this stage, the holon captures changes in the system, which may lead to the evolution of the holon. It is worth noting here that we assume that the atomic holons are specified at design time and do not evolve over the life cycle of the system. However, composite holons continuously evolve to reflect the changes in the system. In addition, a composite holon represents a composite system that may be involved in another system to form an SoS. In this case, a composite holon is used in another composite holon to describe the SoS.

Modification. Holons can change at runtime due to a change in the physical system (*e.g.,* update of a service) or due to a structural change (*e.g.,* composition to a new holon). In either case, the holon description will be modified to reflect the change. Upon obtaining a modified version of a holon, the receiving holon will recompile its own holon and disseminate it. This will convey the changes to the whole SoS.

The next section gives details about each of the above stages.

Table 1
Survey of existing sensor-based ontologies.

Ontology	Suitable?	Comments on suitability
A3ME	✓	
CoDAMOS	✓	
MMI	×	Not well tested in other contexts
O&M	×	No expressive representations of time and space
Ontonym	✓	
OntoSensor	×	Concepts and properties not transferable to different contexts
SDO	×	Not available for use
Sensei, SAREF	×	Some properties are not well defined
SensorML	×	Basic; weak concept documentation
Swamo	✓	

5. The ontological model

For a holon to be able to compose with another at runtime, it must advertise its own definition in a systematic way that can be easily understood by the receiving holon. For this, the definition needs to embody the different concepts surrounding the holon, triggering the need for an appropriate structure to represent it. In this respect, an ontology seems to be the best technique for capturing the definition and behavior of a holon [31].

5.1. Background and ontology selection

An *ontology* is a formal and explicit specification of a shared conceptualization. It models some aspect of the world (*i.e.,* a *domain*), and provides a simplified view of certain phenomena in this domain. The description of the domain is based on a vocabulary that explicitly defines its concepts, properties, relations, functions, and constraints.

For our purposes, the ontology will be used by a holon to advertise its services, the types of input parameters required for said services, and the types of outputs they produce, if any. Furthermore, the ontology is used to identify physical properties such as the power level (consumed during operation), location (the actual, tangible place where system components are situated), operating system, mobility (capability to relocate), etc..

Whilst an entirely new ontology can be developed from scratch, we deemed it more constructive to look at existing ontologies and extend suitable ones, *i.e.,* the ones that contain the semantics required to describe holons, if and where necessary. This is in line with the common practice within the semantic community to re-use existing ontologies wherever possible, which mitigates the heterogeneity caused by various ontologies serving similar purposes. In order to identify the most appropriate ontology to represent holons, we looked at different sensor and observation ontologies. We started with those surveyed by the W3C Semantic Sensor Network (SSN) Incubator Group [45],[1] and continued with our own study of others. Our main selection criteria are: (i) inclusion of well-defined concepts for device capabilities, including hardware and software (e.g. shared resources); (ii)inclusion of well-defined concepts for physical properties, including time and space; (iii) easily extensible for new devices and system contexts; (iv) available to use as open source; and (v) well documented.

Table 1 summarizes the main ontologies that we reviewed. Based on this investigation, the ontologies we found suitable for extension are (in chronological order): CoDAMOS, Swamo, A3ME, and Ontonym. We concluded to use CoDAMOS [46] due to its inherent predisposition for modification, making it easily extensible for defining context-aware

[1] We specifically did not opt for the SSN ontology, as our focus is not limited to sensor-based systems, but rather more on capturing the abstract nature of a holon and how to reason about whether a holon is atomic or composite.

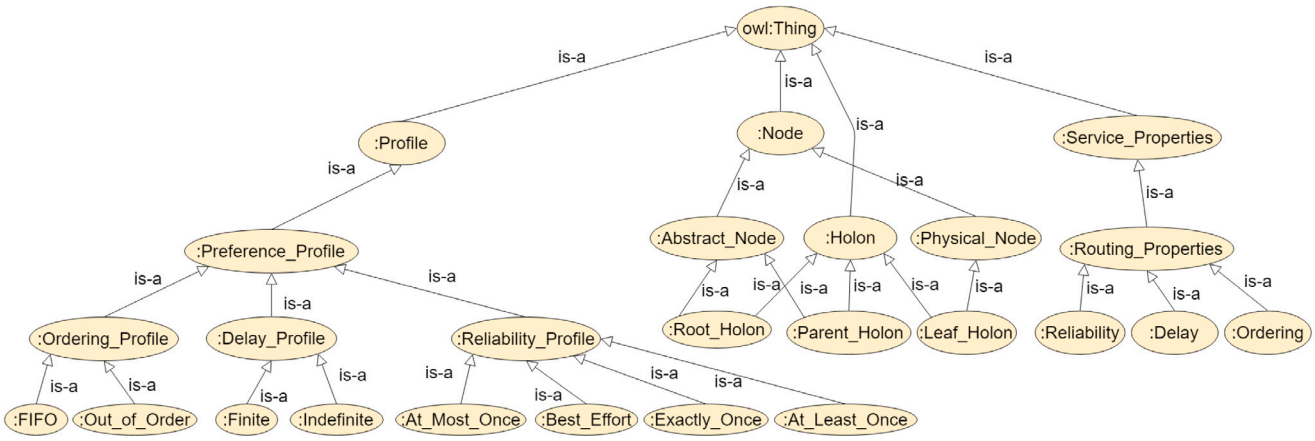

Fig. 4. Our extensions to the CoDAMOS ontology, enabling the representation of holons to form SoSs.

computing infrastructures varying from small embedded devices to high-end service platforms. Furthermore, CoDAMOS's concepts closely match the kind of definitions we want to create for a holon. For example, the *Service* concept – *i.e.,* having a service profile – can be used to define the types of services provided or required by a holon. Note, however, that this does not imply that the other shortlisted ontologies cannot be used. Indeed, A3ME [47] (to pick just one other ontology) could be merged with CoDAMOS to describe further concepts related to holonic design; for instance, the *APIPublic* CoDAMOS concept can be further described as *GlobalAddress*, *LocalAddress*, or *OtherAddress* using the *Address* concept of A3ME.

We use CoDAMOS as the basis ontology with the aim of answering the research questions posed in Section 3, particularly RQ1. Whilst we have not made use of all the CoDAMOS concepts, we have nonetheless extended this ontology to meet our requirements for a holon. However, unused concepts could be built upon for use cases beyond those presented in this work.

5.2. Extensions

Among its various concepts, there are four basic CoDAMOS concepts that stand out in terms of designing holons: *User*, *Environment*, *Platform*, and *Service*. The *User* concept represents individuals or entities interacting with the system. Users can be human users, other systems, or entities that engage with the defined system. It has a profile, a role, and a task. The task can have activities and/or uses a service. The *Environment* concept refers to the context or surroundings in which the system operates. It could include physical or virtual settings, conditions, or external factors that influence the behavior of the system. It defines a location (relative or absolute), a time, and an environmental condition (*e.g.,* temperature, pressure, humidity, lighting, noise). The *Platform* concept represents the underlying infrastructure or technology stack on which the system operates. This could include hardware, software frameworks, operating systems, or other components that support the system's functionality. It has an Environment and can provide a Service (used by User). The *Service* concept represents a distinct functionality or capability provided by the system. It has a profile, a model, and grounding. A *Software* concept links to Service through a property called *providesService*, and can be differentiated as middleware, VM, or OS. The *Task* concept makes use of a Service. Other resources such as memory, network, power or storage can also be modeled accordingly.

Fig. 4 depicts how we extended CoDAMOS to accommodate the requirements of a holon. The *Profile* is extended to show profile preferences for routing messages in a system such as *Ordering*, *Reliability* and *Delay*. A new concept *Node* is added to capture the types of nodes encountered in a system: physical and abstract. This allows systems to be organized in a hierarchical way with physical systems at the very bottom of the hierarchy as leaf holons, and abstract systems as their parents, and so on till a root holon at the very top. All of these are added as concepts under the *Holon* concept. Furthermore, Service Properties has been extended to accommodate routing properties such as Delay, Reliability and Ordering.

The *Service* concept from CoDAMOS has been extended using *Service* from A3ME to describe service types such as *RealWorldService*, *HardwareService*, *SoftwareService*, and *OtherService*. The *StorageResource* concept from CoDAMOS has been extended using the *Storage* concept from A3ME, resulting in the sub-concepts: *Flash*, *HD*, *ROM*, *RAM*; the *PowerResource* concept from CoDAMOS has been extended through the *Energy* concept of A3ME which describes *Renewable*, *NotLimited*, *OtherEnergy*, *Passive* and *Battery*; the *APIPublic* concept has been further described as *GlobalAddress*, *LocalAddress*, *OtherAddress* using the Address concept of A3ME.

For implementing these extensions we used the Protégé ontology editor [48]. A high-level view of our full holonic ontology is shown in Fig. 5 and published as open source at https://github.com/Elhabbash-A/holons.git.

5.3. Application

As a demonstration, we show how two holons can be composed using their underlying description. As such, the starting point is where each holon defines itself using the ontology: its properties, the services it provides, and the parameters used. These descriptions are broadcast by each respective holon.

On receiving such broadcast, holons will compose with each other if they meet the criteria for composition, for instance: if a holon H_1 is requiring a service X, and it encounters another holon H_2 that is providing such service, then H_1 can initiate the composition procedure with H_2. Once H_1 gets composed with H_2, it needs to update its definition to reflect its new state as a composed holon, *i.e.,* an SoS, that is now providing service X. This update is carried out at runtime through creating an instance of the *holon* concept (called, say, H_3) to represent the new holon that has been encountered, and it also holds the definition of H_2 in this case.

Moreover, the ontology retains the ability to infer new knowledge based on the domain information provided. For example, there is a defined concept called *composedHolon* that is used to identify whether a holon is simple or composed. In this context, given that a holon has been composed to another holon, the ontology reasoner can determine whether this holon is a composed one. If H_2 gets out of reach, then the ontology of the composed H_1 will be updated to accommodate this change, simply by removing the H_3 instance and clearing *composedHolon*.

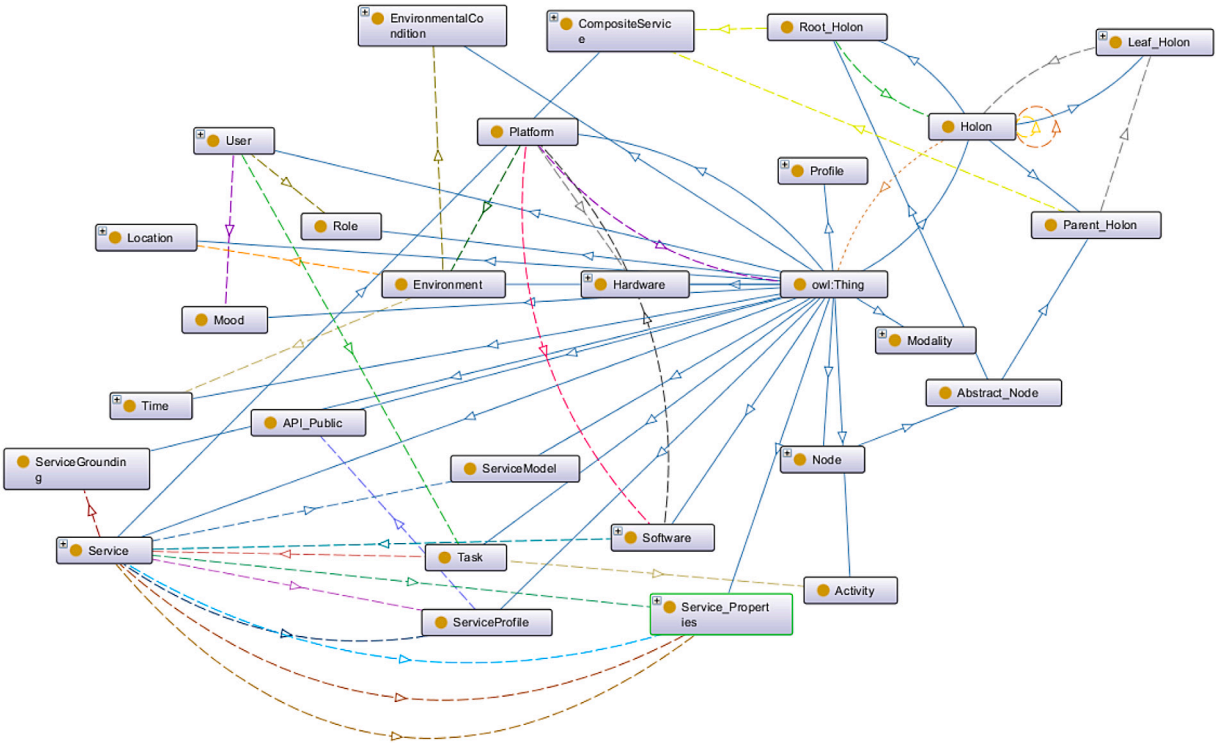

Fig. 5. The metal-model of the full holon ontology.

6. SoS construction model

At a high level, our approach first transforms a holon to be represented as a weighted tree that reflects its interaction with other holons. Upon receiving a request for a service, the tree is used to find the holon through which the service can be accessed. We present the architecture that realizes this approach.

6.1. Composition model

Each holon needs to build a model that represents its awareness about the existence of other holons and their services. This model (called the composition model) is used to interact (compose) with the other systems by accessing their services. The composition model of a holon is represented as a weighted tree T rooted by the holon and with a depth of three. The children of the root are the holons that are directly reachable by the root holon. The leaf nodes represent the detected functionalities that are provided by or accessed through the children holons. Each leaf node i is assigned a weight that represents the cost of accessing the corresponding functionality F_i. For simplicity, in this article we define the cost as the distance between the root and the holon providing F_i in terms of the number of intermediate holons. Other cost functions such as delay, reliability or aggregated Quality of Service (QoS) can also be used and the definition of cost SoS contexts can vary significantly based on mission logic, criteria, and other factors beyond simple levels of indirection. Nonetheless, it is worth mentioning that a holon is not aware where a functionality F_i is located. However, the holon is aware of the cost of accessing F_i and through which holon can F_i be accessed. Fig. 6 shows an example of a holon connected to three other holons in its neighborhood. Functionality F_2 can be reached through both H_1 and H_2, but it is less costly to access through H_1.

The composition model is frequently updated during the lifecycle of a holon, e.g. to add/remove new holon branches, and to update the services of current ones. Adding and updating are performed upon receiving holon ontologies, whereas removing is performed if the ontology is not received during a certain time period; this is set to be three times the dissemination period as sufficient length of time in most scenarios to receive a response, however this of course could be modified as necessary.

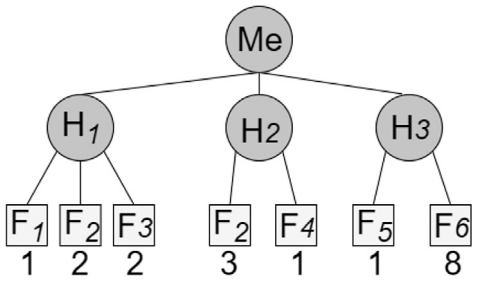

Fig. 6. Composition/interaction modeling example.

6.2. Behavioral model

We assume that a system developer will create the atomic holon using the ontology described in Section 5. Once deployed, the holon lives in the described lifecycle (Section 4). Fig. 7 offers a high level view of the behavioral model of the SoS composition process using the holonic approach. The figure shows the following main components.

OWL parser. This component receives and parses ontologies from surrounding holons to create objects that represent each holon and its functionalities. We adopt OWLAPI V5.1 [49] to parse ontologies and extract the knowledge therein.

Fig. 8 displays an overview of the mapping of ontology elements to a holon object. For each Class element, a Java class is created to represent it. For example, a Java class `Service` represents the ontology *Service* concept, `Profile` represents the *Profile* concept, and so on. The Instance elements of the ontology are instances of the Class elements. The instances are linked together using *DataType* or *Object* properties. Such structure is mapped as attribute objects of the `Holon` object.

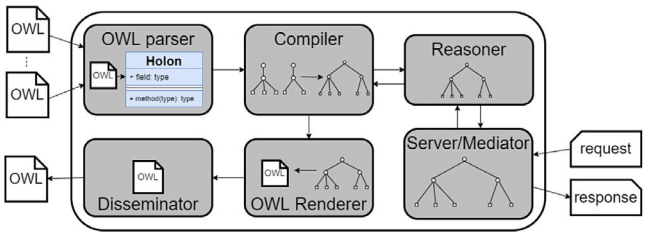

Fig. 7. The behavioral model of the composition process.

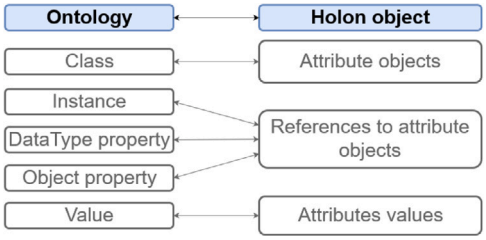

Fig. 8. The relationship between ontology elements and those of a holon.

For example, assume the ontology contains an element $service_i$ that is an element of class $Service$. This instance can be linked to the $Holon$ concept using an object property element called $hasService$. In the Java `Holon` class, this is mapped by having an attribute called `service_i` of type `Service` in the `Holon` class. Finally, in the ontology $Value$ elements represent some of the parameters values that are mapped as attribute values of the `Holon` Java objects. An example is the value 25 °C of the $Temperature$ element which is assigned to the `temperature` attribute in the `Holon` Java object. The OWL parser then passes the created `Holon` object to the Compiler.

Compiler. This component formulates a notional model of the developing SoS as seen from the perspective of the receiving holon. This model takes the form of a tree data structure where the receiving holon is at the root, with each branch representing a neighboring holon and the functionalities provided either natively by it or indirectly through it. To construct or modify the tree, the Compiler creates a branch that represents the received holon object and its provided functionalities as children. The cost of each functionality is also updated at this stage. The exact method for updating the cost would depend on the specific criteria and algorithms employed within the system for cost estimation. The goal is to ensure that the cost accurately reflects the resource requirements or other pertinent considerations associated with each functionality within the evolving System of Systems (SoS). For instance, if the cost method is based on the number of hops, it implies that the cost associated with each functionality is determined by the distance or level of connectivity between the receiving holon and the holons providing those functionalities. In this context, updating the cost would involve counting the number of intermediary holons (or "hops" between the receiving holon and each functionality-providing holon. After that, the newly formed branch is attached to the root holon as a child. The Compiler then passes the constructed tree to the Server/Mediator and OWLRenderer.

Server/Mediator. This component receives requests for the functionalities provided by the system and serves them. It uses the constructed holon tree to specify whether the request can be satisfied by the system or needs to be directed to another holon. If the requested functionality is provided by the holon, then the system processes the request and responds to the requester. Otherwise, the system acts as a mediator and redirects the request to the holon that provides the request, waits for the response, and passes back the response to the requester.

Reasoner. This component checks whether executable services are available to satisfy the received request. The Server/Mediator component uses it to check if concrete functionalities are available in the holon tree and whether all pre-conditions that are required to execute the services are satisfied before executing them. For this purpose, the reasoner component receives the holon tree from the Compiler to reason about existence of executable functionalities.

OWLRenderer Fundamentally, this component represents the knowledge of a developing SoS based on interaction with new holons. It reads the holon tree data structure from the Compiler and converts each node and the corresponding attributes into the appropriate XML tags that construct a valid ontology (*i.e.,* an inverse mapping of Fig. 8). To render the ontology into an OWL description, we also use OWLAPI.

Disseminator. Upon receiving an OWL description representing the holon and the compositions with other holons, the Disseminator publishes this description by broadcasting it — as is the policy in the *Push* strategy (see Section 4).

7. Case study I: Autonomic smart home

In our first case study, we examine the exchange of ontologies between holons to realize high-level SoS logic. This represents construction of an *acknowledged* SoS. The experimental context here is of a smart home application that emerges as an SoS constituting a number of independent IoT deployments.

7.1. Background and challenges

Smart Home is an IoT application that enables users to intelligently manage their homes with minimal intervention. Devices are utilized to monitor home conditions (*e.g.,* temperature, humidity) and the state of appliances (*e.g.,* battery levels) to enable smart management of energy consumption, security, and various housekeeping functions. However, it is no longer realistic to assume a defined and small scale of smart home deployments. As smart home systems are increasing in popularity, the dimensionality of IoT devices used is increasing as new devices with increased capabilities are constantly being offered in the market [50]. Thus, their deployment environments are continually evolving [51], making self-adaptation a necessity [1,10].

Homogeneity also cannot be taken for granted [52]. As these devices are developed by various vendors, it is natural that heterogeneous technologies and APIs exist. Smart home application developers, thus, need to learn these various technologies and APIs in order to develop IoT applications that manage the different aspects of a smart home in a coherent manner. This can be even more challenging when taking into consideration the dynamicity of this environment as new devices can be constantly added to the smart home. Such challenges are making smart home application development a complex task.

Finally, IoT applications generally limit consumers to the predetermined deployment environments they were designed for [53]. Consequently, they are brittle and susceptible to suboptimal operation when their context changes, *e.g.,* due to a backhaul network fault or a server outage. Under such conditions, centralized (mainly cloud-based) approaches fall short especially in network-constrained conditions [54]. Instead, IoT applications often need a way to be able to adapt at the edge without preparation or central coordination.

7.2. Overview of our approach

We illustrate how the adoption of our ontology-based approach helps to significantly mitigate the mentioned complexity. Building on the approach of integrating web services with IoT technology to enable remote access of data gathered by IoT devices (*e.g.,* [55]), we argue that our approach enables the automatic discovery of services that are required for an adaptive smart home management application. Application developers need to specify the abstract workflow of the

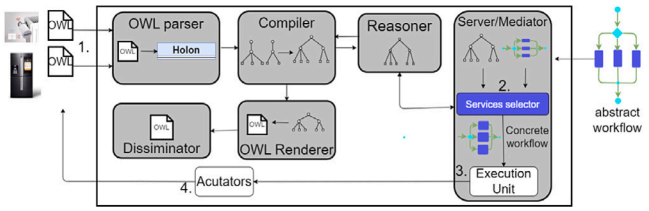

Fig. 9. Smart home controller.

Table 2
Smart home devices and services.

Device	Service	Description
Window	`openWindows`	Opens the windows (by actuators)
	`closeWindows`	Closes the windows (by actuators)
Thermo-meter	`getTemeprature`	Returns the home temperature
Heater	`switchHeaterOn`	Switches on the heating
	`switchHeaterOff`	Switches off the heating
	`getStatus`	Returns true if the heating is on
Fridge	`replaceFilter`	Tells if the filter should be replaced
	`foodLevelLow`	Returns true if food level is low

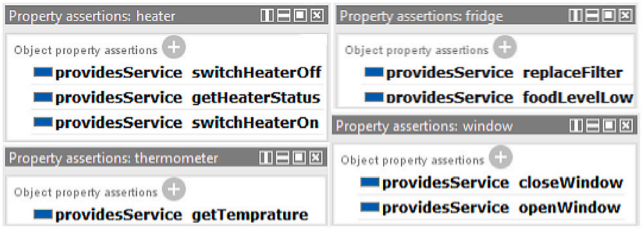

Fig. 10. Object properties of the device ontologies.

application execution where each task in the workflow represents an abstract service. Then, concrete implementations of the abstract services are selected for realizing the workflow.

We consider each device as a holon that is described ontologically (probably by the vendor). The management application is also considered a holon that receives other holon ontologies, parses and compiles them, and then uses their services. Application developers specify the abstract workflow of the smart home management application. Then, the Server/Mediator component selects the concrete services that implement the abstract services of the workflow. A concrete workflow (the application) is passed to an execution unit that executes the application and passes the results to the smart home actuators. Fig. 9 illustrates the above steps.

7.3. Experimental setup

We developed a proof of concept application focused on smart home temperature management. We assume it has the following smart devices: heaters, thermometers, windows, and a fridge, and they provide the services listed in Table 2.

We created a simple ontology (shown in Fig. 10) for each of the devices. We also created an abstract workflow to keep the smart home temperature at 22 °C, as depicted in Fig. 11. The application reads the temperature from the thermometers. If it is less than 22 °C, the application calls the heaters' `switchHeaterOn` service to switch on the heating and the windows' `closeWindows` to close the windows. If the temperature is higher than 22 °C but less than 25 °C, it turns off the heater (calling `switchHeaterOff`). If the temperature is higher than 25 °C it also opens the windows (by calling `openWindows`).

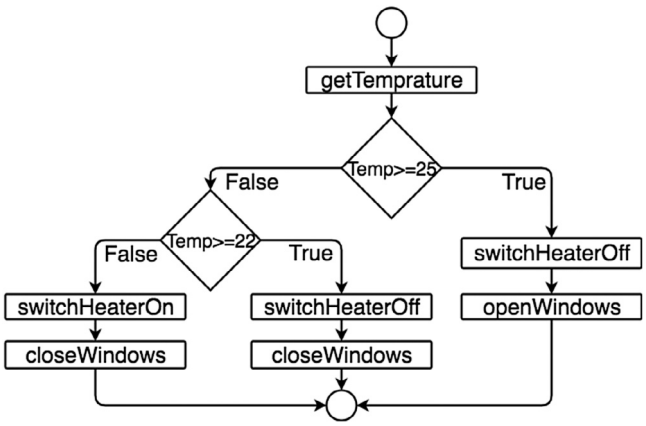

Fig. 11. Prototype workflow.

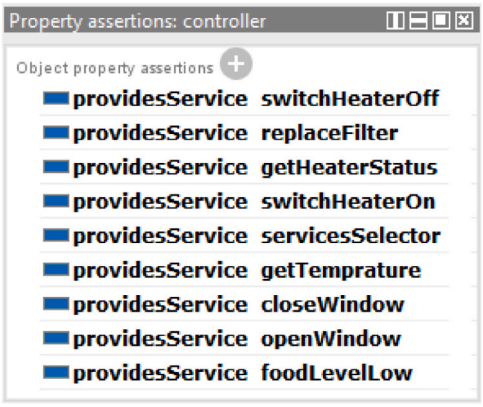

Fig. 12. Object properties of the controller ontology.

Table 3
Experimental cases of the workflow in Fig. 11.

Temp.	Invoked services
10 °C	`getTemperature` → `switchHeaterOn` → `closeWindows`
18 °C	`getTemperature` → `switchHeaterOn` → `closeWindows`
22 °C	`getTemperature` → `switchHeaterOff` → `closeWindows`
23 °C	`getTemperature` → `switchHeaterOff` → `closeWindows`
26 °C	`getTemperature` → `switchHeaterOff` → `openWindows`

7.4. Behavior

Fig. 12 shows a snapshot of the controller ontology after parsing the device ontologies and constructing the SoS tree. The ontology shows that the controller provides services that are provided by the devices themselves (see Fig. 10) in addition to `servicesSelector`, which selects concrete services for the abstract workflow.

The execution starts with reading the temperature from the thermometer by calling the `getTemeprature` service. Table 3 presents the experimental cases and the invoked cases for each case. As the table illustrates, the sequence of invoked services complies with the abstract workflow provided by the developer to the controller, despite the developer not hard-coding the required connections between devices. For example, when the temperature is 22 °C, the execution unit invokes the services `getTemperature`, `switchHeaterOff`, and `closeWindows` respectively, which complies with the abstract workflow (Fig. 11).

This case study demonstrates the potential for easing the development of IoT applications, where application developers do not need to know the details of smart device APIs. Definition of device ontologies

Table 4
Developer tasks required to implement the smart home case study.

Classical approach	Holonic approach
1. Understand the services, APIs, and properties of the devices listed in Table 2. 2. Design the composition of services based on required functionality and context. 3. Develop code to compose the services. 4. Deploy the system.	1. Define the abstract workflow of the required functionality on the controller. 2. Deploy the devices listed in Table 2.

(by vendors) and an abstract application workflow (by smart home developer) are sufficient for runtime application synthesis and execution by the proposed architecture. Table 4 presents a comparison of the tasks required from smart home developers to implement this case study in both the classical and holonic approaches.

8. Case study II: Dynamic cluster management

We now focus on dynamic holon interaction. In this case study, we demonstrate how holons of similar ontologies (*i.e.,* containing the same services) but in constant movement can interact with each other despite continuous creation and modification of new SoSs. Our context is cluster management using Apache Mesos. This represents construction of a *directed* SoS.

8.1. Background and challenges

Inspired by heterogeneous fog clusters [56,57], we set a scenario where device interaction is constant. We use Apache Mesos [58], an orchestration tool commonly used to manage resources that are shared between different applications and their sub-tasks. In effect, Mesos enables the viewing of data centers and other computing clusters as a single consolidated resource.

Although Mesos is a very useful utility, it was designed mainly for shared resources in relatively stable environments such as data centers. As such, the computing cluster can only change (*i.e.,* grow or shrink) through manual modifications to the configuration by the user. Mesos was not designed to work in an environment where node status is in constant flux due to movement, unreliable power, or communication outages. These are typical challenges in fog computing [59,60]. Mesos is also designed to work in a hierarchical fashion, whereby *Agents* (worker nodes) can only communicate directly to the *Master* but not through other Agents [58,61].

8.2. Overview of our approach

Both of the above restrictions can be overcome using the holonic ontology, which offers opportunistic composition (overcoming the first restriction) and horizontal composition between self-describing clusters in the form of holons (second restriction). We draw a scenario here to demonstrate this using containers running over an unreliable infrastructure such as edge PoPs [62]. In this scenario, each node can be a Master or an Agent. Following the basic design of Mesos, Masters are responsible for dispatching containers to the Agents, who in turn operate the containers.

8.3. Experimental setup

A simple example is given here to illustrate how holons could be used to facilitate the union of Mesos clusters with mobile nodes without the need for establishing direct communication. Owing to the high mobility of nodes during the tests, the composition of SoSs is dynamic, creating several additions and removals of the Mesos Services.

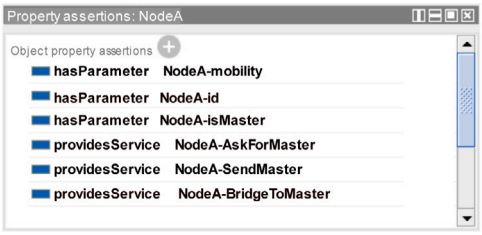

Fig. 13. The ontology model of Mesos NodeA.

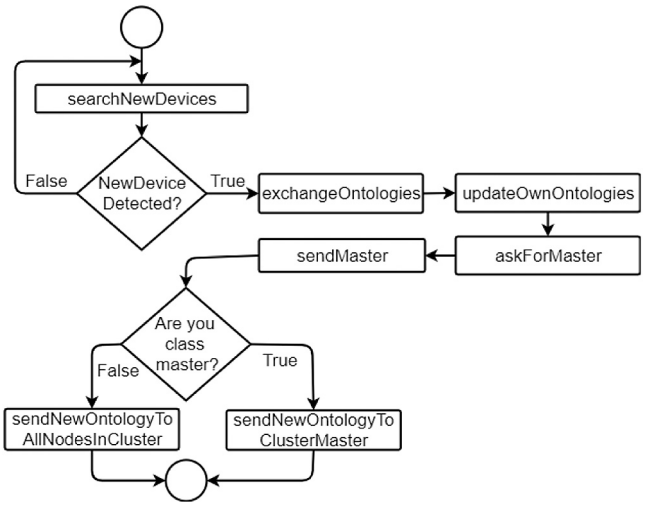

Fig. 14. Prototype workflow of Dynamic Cluster Management when a node reaches another node.

This would have been prohibitive to accomplish using Mesos' manual configuration.

We tested this use case study using 100 devices that can move freely. Every device is considered a node for Mesos and is individually defined as a holon at the beginning of the experiment. Shortly afterwards (a minute later), bigger holons begin to be created, containing one or more devices. After that, each holon starts an internal process of randomized leader election [63] to elect a master from amongst the constituent devices.

Each holon contains parameters to define its identifier, mobility, and whether or not it is a Mesos Master (*e.g.,* Fig. 13). Additionally, each holon contains three services: `AskForMaster` is performed every time two holons reach each other for the first time, where it would return the ID of the Master. After which, a holon will perform the `SendMaster` service to send such ID. `BridgeToMaster` is used by the nodes to communicate with the Master through its Agents. Fig. 14 shows an abstract workflow of the experiment when a node reaches a new node from another Mesos cluster. A second abstract workflow has been created when a node receives a new ontology, which is depicted in Fig. 15. Both abstract workflows are used by the nodes to allow dynamic union of Mesos clusters with mobile nodes. Device removal is left as future work.

8.4. Behavior

Fig. 16a presents the starting point with holons H_1 and H_2. H_1 is formed of four nodes: nodeA, working as a Master, while nodeB, nodeC and nodeD as Agents. On the other hand, H_2 is formed of two nodes: nodeF, a Master, and nodeE, an Agent. In this case, H_1 and H_2 cannot reach each other.

As nodeC and nodeE come in range of one another (Fig. 16b), they are triggered to exchange ontologies and call `AskForMaster` and

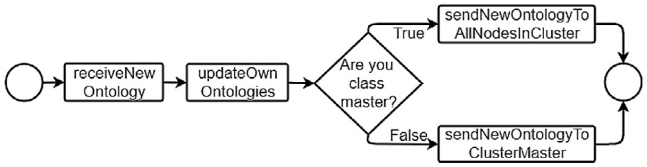

Fig. 15. Prototype workflow of Dynamic Cluster Management when a node receive a new ontology.

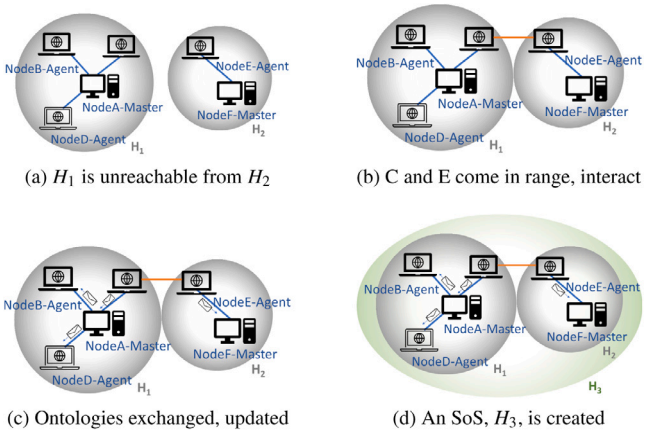

Fig. 16. How to use holons to construct an SoS of Mesos clusters running on constantly moving nodes.

Table 5
Tasks required from Mesos developers to implement the dynamic cluster management case study.

Classical approach	Holonic approach
Repeatedly: 1. Select a cluster Master. 2. Add nodes to cluster. 3. Add services to cluster. 4. Remove nodes from cluster. 5. Remove services from cluster.	1. Define the cluster node as a holon using the ontology. 2. Implement the cluster Master election algorithm. 3. Deploy the holons.

subsequently `SendMaster`. The orange arrow in the figure represents this interaction.

Then, nodeC and nodeE broadcast their new ontologies to neighbors (Fig. 16c). Now, H_1 and H_2 are not just in contact with each other, they are part of a new super-holon H_3, which is the union of H_1 and H_2 as all their nodes receive the updated ontology and adopt it as their new ontology.

All Agents are sharing the Masters nodeA (from H_1) and nodeF (from H_2). This is achieved using the service `BridgeToMaster`, performed continuously by nodeC and nodeE while they can reach each other. The resulting SoS H_3 has 2 Master nodes and 4 Agents. Therefore, each of the two Masters can communicate to all Agents in H_3 as if they were connected directly. Similarly, this approach is also able to deal with Master mobility as long as the basic structure of a Mesos cluster is not broken down.

As such, the ontological exchange and reasoning of holons allows Apache Mesos to transcend its innate design shortcomings and enables it to form a dynamic cluster structure. Achieving such structure through manual configuration (which is the only way possible using native Mesos) significantly restricts adaptation and reduces cluster efficiency by a factor of 5 compared to using holons. Table 5 compares the tasks required from Mesos developers to implement this case study in both the classical and proposed approaches.

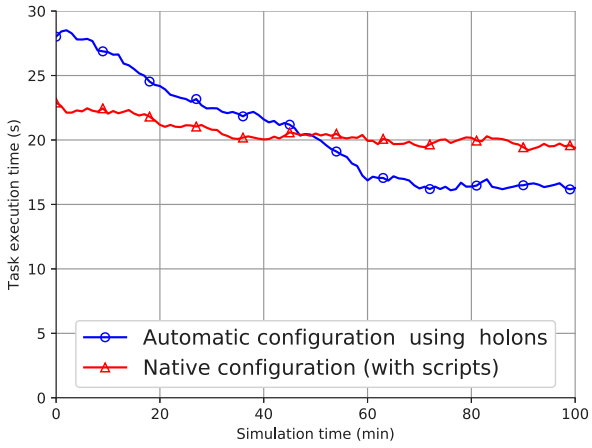

Fig. 17. Computational tasks are completed faster on dynamic Mesos clusters constructed using holons as opposed to native Mesos cluster configurations.

8.5. Simulations

For evaluating the efficacy of automated expansion of Mesos clusters through the use of holon ontologies, we used the Omnet++ discrete-event simulation framework [64] to simulate 100 nodes. We set the node transmission range to 20 m and speed to 1.43 m/s, an average walking speed [65]. The nodes used individual-level (random walks) [66] as a mobility model.

We used Mesosaurus [67] to create task loads to test the performance of the formed clusters. Specifically, we seek the length of time required by a Mesos Master to perform a specific task. The task created for this experiment is one that a Master with 5 Agents will normally perform in about 20 s. Execution time is expected to decrease with more Agents.

Fig. 17 plots the average task execution time across all Mesos clusters after 20 experiment runs. Using Mesos' native cluster management method, task execution time decreases slightly (from ≈ 23 s) as Masters expand their clusters through the use of scripts that add nodes they encounter in their environment. This improvement in performance, however, eventually plateaus (≈ 20 s) as churn overwhelms Masters through frequent configuration management, despite the use of automated scripts. On the other hand, using holons introduces some overhead in terms of ontology creation and reasoning. This results in a slightly inflated initial execution time (≈ 27 s). However, as nodes encounter others during the lifetime of the simulation, Mesos clusters identifying as holons expand dynamically according to changes in their environment. Compiling and reasoning overheads soon become relatively insignificant, enabling Mesos holons to achieve an average of 17 s execution time, a 15% improvement in performance.

9. Case study III: Disaster rescue team formation

In this final case study, we demonstrate how holons can interact opportunistically under unforeseen circumstances. This represents construction of a *collaborative* SoS. We study how holons discover each other and compose to gain mutual access to functionalities. Our context was an infrastructure-less environment caused by a natural/man-made disaster.

9.1. Background and challenges

This scenario involves a case of disaster settings where no infrastructure-based communication is available because of large-scale

power outage,[2] for instance. Rescue teams can be deployed over areas of interest. If we assume that such an area is wide, mountainous, or susceptible to high levels of interference (as in large metropoles) and the rescue teams employ MANET-based communication, then they could easily become isolated. To deal with this, the rescue administration can deploy some kind of vehicles – *e.g.,* robots or Micro Air Vehicles (MAVs) – to facilitate communications. The rescue team devices should be able to dynamically discover the MAVs, compose, and communicate with other teams through them. In effect, the MAVs act as bridges to enable communication between rescue teams. Furthermore, each rescue team system may provide services and/or data that can be needed by or useful to other rescue teams; such as specialist recovery procedures, computational capacity to carry out intensive real-time risk analytics, list of rescued individuals, list of confirmed casualties, disaster area maps and schematics, notification services for how the wider disaster unfolds, etc..

Similar scenarios have been discussed by other works. However, they tended to focus on software/hardware design (*cf.* [69]), coverage capabilities (*cf.* [70]), or geographical placement (*cf.* [71]). There currently is no work on how to enable impromptu deployed vehicles to compose at runtime with ground teams without any pre-configuration, and to adapt to changes in their execution environments, something that is likely.

9.2. Overview of our approach

In order to enable this self-organizing case, each rescue team system and MAV is considered a holon with its own self-describing ontology that it periodically broadcasts. The description, based on the standard ontology of Section 5, includes: the ID of the holon, the services that the holon provides, and the information that is required to communicate with the holon and access its services. Upon receiving a new ontology, the receiving holon parses the description and, thus, discovers the surrounding holons and how to interact with them. The ontology also includes any compositions the sending holon currently has, which enables the receiving holon to compose indirectly with other holons not in its one-hop range.

9.3. Experimental setup

Fig. 18a shows the deployment of the rescue teams. After deployment, `Team A` and `Team B` broadcast their respective holon descriptions (every 3 s). After mutually parsing the descriptions, the teams start to communicate; *e.g.,* by calling data look-up services. As rescue operations progress, teams move apart and eventually lose communication, as shown in Fig. 18b. In this case, the ontologies can no longer be received and `Team A` and `Team B` discover that the data look-up services are no longer available. To resolve this, the response coordination center prepares and deploys `Team C` in the form of a MAV. This system broadcasts its description and, similar to the above, can then communicate with `Team A` and `Team B`. Consequently, `Team A` and `Team B` discover (from the holon of `Team C`) that the data look-up services they previously used are available again through `Team C`. To test the above, we developed data look-up and messaging services as RESTful services. The data look-up service provides simulated data representing the number of victims found in the disaster area, whereas the messaging service provides reliable asynchronous communication between rescue teams. Each system broadcasts its holonic description, parses the received descriptions, and provides interfaces for communicating with and invoking the services of dynamically discovered holons. Fig. 18 also shows an abstract workflow of the experiment, whereby a team searches for another team's service to obtain data/access a service. The workflow is used to automatically discover data lookup services of other teams to facilitate knowledge sharing.

[2] Similar to what happened in Lancaster during Storm Desmond in December 2015 [68].

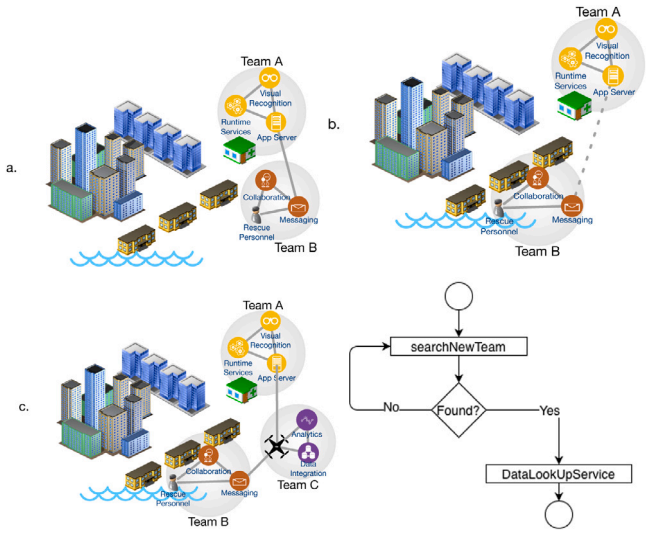

Fig. 18. A scenario of rescue teams during a flooding disaster where communication is disrupted due to power outages, and its abstract workflow.

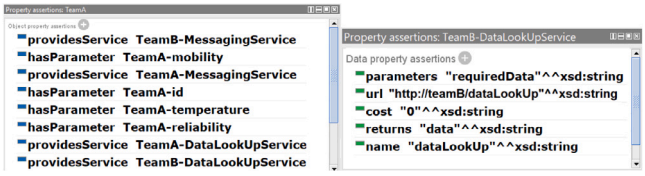

Fig. 19. The ontology model of the `Team A` system.

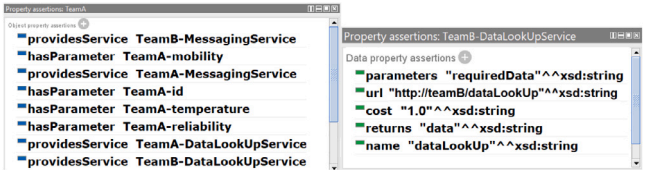

Fig. 20. The ontology model of the `Team A` system after the construction of `Team A-Team B` SoS.

9.4. Behavior

We demonstrate the opportunistic SoS construction and self-organization by examining the ontology model of `Team A` over the four states that constitute this case study, as follows.

State 1: The development of the ontology. Fig. 19a shows the object properties of the `Team A` holon, including the parameters it holds and the services it provides. In Fig. 19b we take the data look-up service as an example to show the data properties of the service. The data properties include the name (assumed to be unique), parameters (the required data), the cost (0 in this case as the service is hosted locally on the `Team A` system), the URL to access the service, and the return type.

State 2: Constructing the Team A-Team B SoS. Fig. 20 illustrates the updated ontology of the `Team A` system after receiving the ontology of the `Team B` system. Fig. 20a shows that two new services appear in the object properties of the ontology of `Team A`, while Fig. 20b lists the data properties of `Team B-DataLookUpService`. The cost of this service is 1 as it is hosted in a neighbor system a single hop away.

State 3: Deconstructing the Team A-Team B SoS. As rescue progresses, each team independently carries out its own operations that might

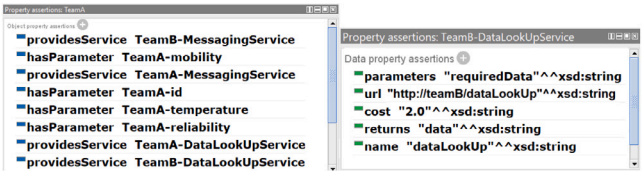

Fig. 21. The ontology model of the Team A system after the construction of Team A-Team C- Team B SoS.

Table 6
Tasks required from developers to realize the rescue case study.

Classical approach	Holonic approach
1. Develop the service discovery logic. Repeatedly: 2. Manually check for available teams in the range and select their services. 3. Understand the APIs of the selected services. 4. Manually wire the application logic with the selected services.	1. Define the team system as a holon using the ontology. 2. Implement the holon discovery and integration logic. 3. Deploy the holons.

involve moving to cover different parts of the disaster-struck terrain. As Team A and Team B drift away from each other, they lose direct communication as is depicted in Fig. 18b. Communications time out and the SoS formed in the previous step is dissolved. The ontology and the composition models of Team A revert to what is represented in Fig. 19.

State 4: Constructing Team A-Team C-Team B SoS. A new system, Team C, is deployed into the disaster terrain to act as a communication bridge. Fig. 21 outlines the updated ontology model of the Team A system after receiving the Team C ontology. Fig. 21a shows that the Team B services appear back in the object properties of the ontology of Team A. Moreover, Fig. 21b shows the data properties of Team B–DataLookUpService, the cost of which is 2 as it is now available indirectly through a third system (Team C). Table 6 compares the tasks required from developers to implement this case study in both the classical and proposed approaches.

9.5. Running example

To simulate the case study, we deployed Raspberry Pi microcomputers to represent members of the rescue teams. We specifically explore the efficacy of the mechanism to opportunistically compose a rescue super-system when faced with network-level disruption, as illustrated in Fig. 18.

The microcomputers were initially configured to communicate with each other through the same wireless network. Consequently, Teams A and B reason and compose directly (State 2). For this experiment, we created a stream of requests between the teams to measure quality of service. The requests were issued by Team A to obtain updates from Team B on the number of found victims. We then modified Team B microcomputers to use a different network and, hence, were no longer able to communicate with Team A. Next, we deployed Team C, which we configured to use both networks to act as a bridge between Teams A and B. Again we evaluate the ability of invoking services of other systems, now through indirect composition (State 4). Under both states of composition, requests were successfully received and responded to by Team B. For reference, the average response times under State 2 and State 4 were 91 ms and 281 ms, respectively. Naturally, the latter response time is higher as requests are inter-mediated through Team C.

10. Quantitative evaluation

The methods of our proposed framework include means of parsing holon ontologies to build a tree that represents SoS construction, and subsequently rendering the tree to disseminate modified ontologies that reflect SoS evolution. As the efficiency and efficacy of these methods are of paramount significance, we conducted experiments to investigate the time required to parse and render the ontologies at different scales. We also assess the validity of these different stages of processing holons. From this, we draw conclusions about the ability of using holons to compose SoS during runtime and at scale. The platform used in the following experiments was Intel Core i7 with 16 GB RAM, running Linux Ubuntu v16.04, and Java SE v1.8.0. The presented results are the mean values from 100 repetitions.

10.1. Parsing

This first experiment measures the time taken to convert a received ontology into a tree (Section 6). Recall that the tree contains the holon object as a root and the services provided by/via it as its children. We vary the number of children (*i.e.,* provided services) as the main dimension affecting the scalability of our parsing stage. In Fig. 22(a), we observe that parsing time increases linearly with the number of services, amounting to < 0.35 s for a very complex SoS that provides 1000 services either directly or indirectly. We find this level of complexity to be acceptable, as it indicates the feasibility of parsing an increasing number of holons in an SoS.

10.2. Ontology size

This experiment focuses on investigating the size of the ontology (in MB), which demonstrates the size of the control traffic exchanged between the constituent systems. Similar to the above, we vary the number of services provided by/via the holons. In Fig. 22(b), we observe that ontology size increases linearly with the number of services, amounting to around $2MB$ for a very complex SoS that provides 1000 services either directly or indirectly. We argue that this level of complexity to be practically acceptable for an extremely complex SoSs. However, compression techniques can be applied to reduce the size of the exchanged ontology between holons, when needed.

10.3. Rendering

We now examine rendering time defined as the time to convert a holon tree into an ontology that is ready to be disseminated. We varied both the number of neighbor holons (children) and their services (leaves), and observed that the rendering overhead increases with both (Fig. 22(c)). This is acceptable for SoSs with up to 100 services per holon, where rendering overhead is ≈ 5 s. However, this inflates for holons with 1000 or more sub-systems, where it could take up to a minute to create an ontology that could be used for composition.

10.4. Validation

In this subsection, we assess the efficacy of the parsing and rendering operations using two experiments. In the first experiment we adopt the process depicted in Fig. 23a. We create ontologies for 100 holons with random values for each of their parameter and service values. We then pass the ontology files to the OWLParser, which creates the corresponding holon trees. The trees are passed to the OWLRenderer to render them into ontologies. Next, we use OWLAPI to query both the created ontologies and the rendered ones to check if the results are equivalent. For all 100 ontologies, the returned values were indeed equivalent. In the second experiment, we adopt the process portrayed in Fig. 23b. Here, we synthesized 100 compiled trees, each of which represents an SoS. We then passed the trees to the OWLRenderer to render them into a separate SoS ontology for each tree. We queried both the rendered ontologies and the created trees to systematically compare the results. Again, the returned values were equal for all 100 cases.

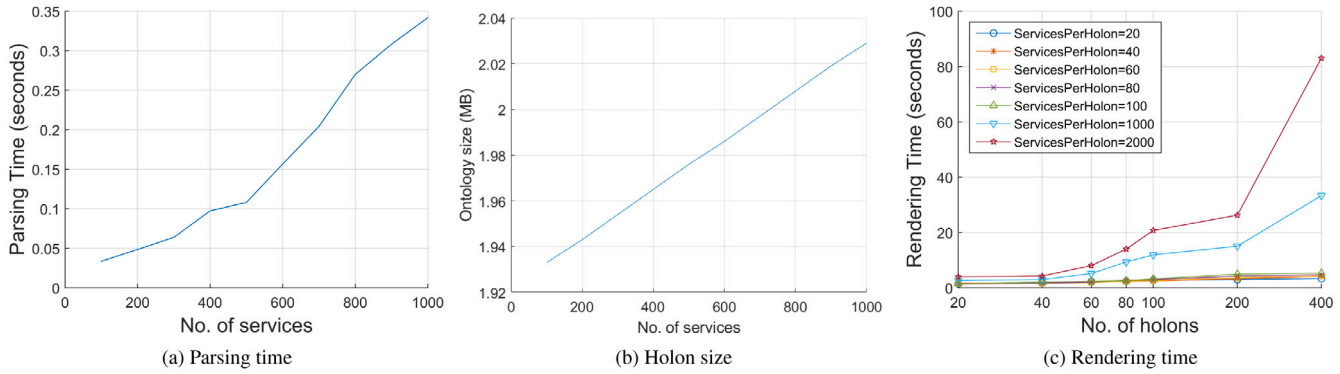

Fig. 22. Ontology parsing time (a), ontology size (b), and rendering time (c) as complexity increases through either the number of holons or services per holon.

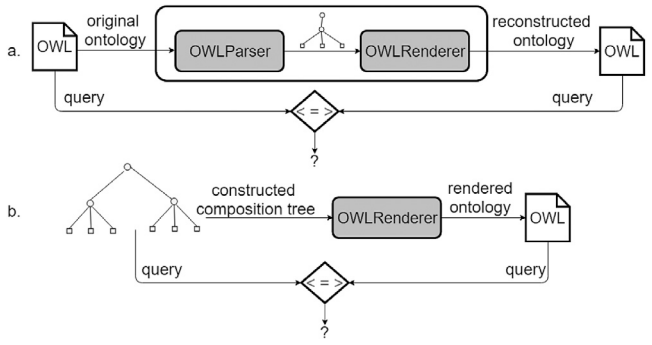

Fig. 23. Our methods to validate ontology parsing and rendering experiments, deemed valid when the information returned by the queries match.

11. Implications and limitations

In light of the above experiments, we now discuss the implications of our work by reflecting on the research questions we posed in Section 3. We also discuss the limitations of our work.

11.1. Abstractions for SoS representation (RQ1)

Holons provide a means for systems to richly describe themselves, and independently reason about the change in their environment and how it might affect their set up and operation. This enables systems to reflect on their existence and how they fit into their surroundings. This aligns with the ethos of both reflective middleware [72] (systems build representations of themselves that they can then adapt) and autonomic computing [73] (systems manage themselves according to high-level objectives). In addition, holonic representation allows systems to transfer knowledge about themselves to other systems that they get in touch with. In addition, each system can build a representation of the behavior of other systems in its vicinity and ergo form more complex systems without prior arrangement.

There is an assumption that each system needs to start with a representation of itself in its simplest form as an ontology. Therefore, we built our ontological architecture on the most generalizable and easy-to-use ontology available in the literature. Furthermore, using our framework makes it relatively easy to create ontologies, where the development overhead is of much smaller scale than defining a system's API, as illustrated by all three use cases. This modest additional development overhead enables the system to adapt after deployment, thereby unlocking a new world of complex system creation that facilitates new forms of context-aware applications.

Takeaway: Holons effectively represent systems in an SoS context. They are verbose enough to capture system characteristics that are used to reason about and action composition, and require only a modest development overhead.

11.2. Techniques for SoS composition (RQ2)

We demonstrated how a developer could define desired behavior at a high level (Tables 4–5), and a system is subsequently composed of appropriate sub-systems to align with this. The architecture that we presented in Section 6 allows systems to independently reason about their environment, specifically about updates in surrounding holons and whether they are reachable directly or indirectly. As a result, it also enables each holon to reason about how such changes would affect its set up and operation. This powerful concept maintains the separation of concerns, which is crucial for effective system development, whilst also reaping the rewards of complex system formation through autonomous composition. Furthermore, due to its recursive nature, the holonic ontology could be applied at different levels: at atomic service (*e.g.,* temperature sensor), system component (*e.g.,* smart sensors), or meta-system level (*e.g.,* smart home controller). This enables developers to write behavior at different levels of granularity with the same modeling effort, which is particularly beneficial for environments such as the IoT and cyber–physical systems where context-dependent behavior could be sought at different levels.

Takeaway: By parsing holons into weighted trees, we can support *reasoning* at the recursive holon level, and *composing* SoSs through direct and indirect contact. This is performed independently without centralized techniques.

11.3. SoS adaptation (RQ3)

The holonic ontology is designed for self-description to enable reasoning and composition, and the architecture that we built enables the continuous maintenance of the holonic lifecycle; *cf.* Fig. 3. This perpetual preservation and updating (or self-healing) of the different ontologies that are present in the system ecosystem allows the 'DNA' of the different systems to persist and to adapt to changes in the ecosystem in a timely fashion. This is fully in line with the vision of autonomous SoS composition and adaptation. Through our experiments with case studies, we have tested the ability to achieve such vision in different ecosystems of notable heterogeneity (IoT, rescue) and dynamism (cluster management). We have not, however, tested the ability to efficiently compose SoSs in ecosystems that exhibit *both* heterogeneity and dynamism, especially at a large scale. Nonetheless, we have no

reason to believe that the holonic approach would not be suitable in such environments.

Human intervention is not needed for composition or adaptation. Involvement is only called for when major requirements changes are required at the SoS level. For instance, a change in the smart home requirements that alters the SoS workflow would require the user to issue such changes as abstract logic. Nonetheless, such intervention is guaranteed to be minimal as there is no need to know the low-level details of post-deployment systems such as device-specific APIs.

In designing our framework, we valued the feasibility and correctness of composition and adaptation (we will discuss these issues further in the two following subsections, respectively). What our framework in its current form cannot assure, however, is tolerance due to failings at the holon computational task level. This issue could easily and rapidly compound in ecosystems of a large scale and dynamic nature. By definition, holons can have infinite task possibilities, such as temperature measurement, cluster management, traffic coordination, and content processing. Fault tolerance mechanisms need to be implemented by task developers as an inherent part of the task, notwithstanding holonic description. Such an addition does not modify the holonic ontology or the behavior we present.

Takeaway: Our architecture augments holonic description to provide a generic framework for supporting adaptation driven by high-level desired behavior adaptation. This is evident from our experiments in heterogeneous and dynamic environments.

11.4. Composition overhead (RQ4)

The architecture operates on the basis of continuously parsing and rendering holonic ontologies in order to reason about composition. After quantifying the cost of these operations over a wide range of system scales, we find that the cost to be quite minimal in most cases. It is only the ontology rendering cost that becomes somewhat non-trivial in cases where a holon would come in contact with a very large number (*i.e.,* 1000 s) of other holons. This overhead stems from the fundamental process of converting ontologies into weighted trees that can facilitate reasoning. We chose this technique as we designed our architecture to optimize for correctness and reliability. For such high scale environments, an alternative is to develop an alternative rendering process that is optimized instead for performance, perhaps using heuristic algorithms.

Takeaway: The cost of composition using our architecture is trivial and acceptable for most environments, except those with an extremely high number of interacting holons.

11.5. Composition correctness (RQ5)

Composition outcomes, *i.e.,* SoSs, were found to be valid and completely in line with the high-level workflows from which they were derived. This is unsurprising as our architecture was designed to prioritize correctness and reliability, which inadvertently can sacrifice performance, as already discussed. Despite our extensive efforts to test validity with a number of real-world and synthetic workflows, we cannot claim external validity.

Takeaway: SoS composition was reliably valid both in qualitative case studies and in synthesized experiments.

11.6. Limitations

Semantic authority. Defining an ontology to describe any IoT system is an arduous challenge that presents a difficult tradeoff between generality and usefulness. Moreover, any ontology will have its own shortcomings. However, our intention is *not* to designate a *definitive* ontology that should be adhered to by all device manufacturers or system developers. Instead, our aim is to provide means by which post-deployment composition is possible in environments that host various systems. In this spirit, although creating holonic descriptions by manufacturers is not an unreasonable expectation (*cf.* recent devices from Bosch, Honeywell, Siemens), automated generation of holon descriptions is possible and highly accurate [74].

Coverage. Our architecture is applicable to various types of SoS, as is evident from our experiments that cover IoT, computational cluster, and ad hoc networking. However, our ontology was founded on another that was developed for IoT devices. We endeavored to generalize the ontology to support SoS composition outside the IoT context, bearing in mind the aforementioned tradeoff. Nevertheless, further experimentation is warranted on other SoS contexts.

Developer overhead. Obviously, there is an overhead associated with adopting the semantic tools we present here, which potentially adds to the resistance to wide adoption. Nevertheless, the overhead is not insurmountable and is acceptable considering the added value it brings. Besides, we see that deriving semantic specification using programmatic means is an achievable objective. Indeed, we are currently developing an approach using graphical programming techniques [75] to relax such fundamental constraint that underlies our work.

12. Future directions

We hereby present possible future directions of interest and potential reward to the community.

12.1. Composition validity and optimization

Although all the composition outcomes were correct in our experiments, there is still work to be done in two primary directions. First, more tests could be performed to ensure valid composition (*i.e.,* in line with the high-level objectives set out by the abstract SoS workflow) with corner cases such as legacy devices with varying firmware versions. Second, a given SoS workflow can have multiple valid composition outcomes. Our architecture can be extended to compare such possibilities using certain criteria such as energy efficiency or operational cost.

12.2. Alternative ontology rendering

Further research is needed to study SoS adaptation in more extreme environments: *e.g.,* highly heterogeneous *and* dynamic, or with high sensitivity to faults. This could be done by using the architecture we propose here, replacing the rendering module by another that is more performance-savvy, *e.g.,* using heuristic algorithms or constraints programming.

12.3. Workflow composition

The presented architecture assumes that developers are responsible for providing a workflow of abstract services that will be used to identify concrete services to fulfill the system requirements. Two issues need to be addressed in future research.

The first challenge is the potential presence of conflicts in the holon ontologies. In the case of multiple services that can satisfy an abstract service, the suitability of such concrete services needs to be quantified at runtime in order to reason about their selection. Therefore, an intelligent and scalable selection of optimal services is required.

The second issue is to further raise the level of abstraction, requiring developers to provide only high-level goals instead of a detailed abstract workflow. The system is then responsible for constructing and maintaining a workflow that is able to achieve the given goals. This also calls for intelligent approaches that accumulate and utilize knowledge of the ability of the constructed workflow to satisfy high-level goals.

13. Conclusion

This article addressed the challenge of constructing systems of systems (SoSs) in a dynamic and unplanned manner. It presented a general approach of how to do this based on defining the concept of a *holon* as a self-describing system that can recurse to encapsulate other holons. Beside providing self-awareness, an ontological description of a holon helps attain context-awareness through the discovery of other holons. For this, we provided an architecture for SoS construction that makes use of holon descriptions to discover them, reason about their functionalities, and integrate them to form complex SoSs.

We demonstrate the feasibility of our framework by examining the details of three case studies that implement contrasting SoS construction scenarios. The case studies show that our framework reduces the complexity of SoS development by abstracting system heterogeneity using holon descriptions and their autonomic manipulation at runtime. They also illustrated forms of system composition that are incredibly difficult, if not impossible, to achieve without prior planning. Furthermore, we assess scalability and validity through a testbed and simulation experimentation, concluding that our framework is realistically feasible with performance exhibiting a linear trend for manipulating and reasoning about holonic system descriptions.

This work represents a generic, extensible, and sustainable way of raising the level of abstraction of SoS construction. In other words, defining ontologies for the holons and using our simple reasoning architecture liberates developers from specific system composition and allows them to be more concerned with how whole systems can interact with each other at runtime. This novel contribution has strong potential for applications in various fields as is evident from our contrasting case studies. In effect, this study generates a new breed of forming systems at runtime. Moreover, our architecture can be modified to cater to domain-specific interactions, *e.g.,* if particular transactional or situational types of awareness are needed.

CRediT authorship contribution statement

Abdessalam Elhabbash: Conceptualization, Investigation, Methodology, Software, Validation, Visualization, Writing – original draft, Writing – review & editing. **Yehia Elkhatib:** Conceptualization, Methodology, Project administration, Supervision, Visualization, Writing – review & editing. **Vatsala Nundloll:** Visualization, Writing – original draft. **Vicent Sanz Marco:** Validation, Writing – original draft. **Gordon S. Blair:** Conceptualization, Funding acquisition.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Abdessalam Elhabbash reports financial support was provided by Engineering and Physical Sciences Research Council.

Data availability

No data was used for the research described in the article.

Acknowledgment

This work was partly supported by CHIST-ERA, Europe under the UK EPSRC grant EP/M015734/1 (DIONASYS).

References

[1] D. Hughes, Self adaptive software systems are essential for the internet of things, in: Symposium on Software Engineering for Adaptive and Self-Managing Systems, SEAMS, ACM, 2018, p. 21.

[2] G. Fortino, C. Savaglio, G. Spezzano, M. Zhou, Internet of things as system of systems: A review of methodologies, frameworks, platforms, and tools, IEEE Trans. Syst. Man Cybern.: Syst. 51 (1) (2021) 223–236, http://dx.doi.org/10.1109/TSMC.2020.3042898.

[3] X. Fang, S. Misra, G. Xue, D. Yang, Smart Grid – the new and improved power grid: A survey, IEEE Commun. Surv. Tutor. 14 (4) (2012) 944–980.

[4] İlker Bekmezci, O.K. Sahingoz, Şamil Temel, Flying ad-hoc networks (FANETs): A survey, Ad Hoc Netw. 11 (3) (2013) 1254–1270, http://dx.doi.org/10.1016/j.adhoc.2012.12.004.

[5] M.W. Maier, Architecting principles for systems-of-systems, Syst. Eng. 1 (4) (1998) 267–284.

[6] J.S. Dahmann, K.J. Baldwin, Understanding the current state of us defense systems of systems and the implications for systems engineering, in: Annual IEEE Systems Conference, IEEE, 2008, pp. 1–7.

[7] J. Boardman, B. Sauser, System of systems - the meaning of of, in: IEEE/SMC International Conference on System of Systems Engineering, 2006, http://dx.doi.org/10.1109/SYSOSE.2006.1652284.

[8] C.B. Nielsen, P.G. Larsen, J. Fitzgerald, J. Woodcock, J. Peleska, Systems of systems engineering: Basic concepts, model-based techniques, and research directions, ACM Comput. Surv. 48 (2) (2015) 18:1–18:41, http://dx.doi.org/10.1145/2794381.

[9] K. Petersen, M. Khurum, L. Angelis, Reasons for bottlenecks in very large-scale system of systems development, Inf. Softw. Technol. 56 (10) (2014) 1403–1420, http://dx.doi.org/10.1016/j.infsof.2014.05.004.

[10] H. Muccini, M. Sharaf, D. Weyns, Self-adaptation for cyber–physical systems: A systematic literature review, in: Symposium on Software Engineering for Adaptive and Self-Managing Systems, SEAMS, ACM, 2016, pp. 75–81, http://dx.doi.org/10.1145/2897053.2897069.

[11] P. Varga, F. Blomstedt, L.L. Ferreira, J. Eliasson, M. Johansson, J. Delsing, I.M. de Soria, Making system of systems interoperable – the core components of the arrowhead framework, J. Netw. Comput. Appl. 81 (2017) 85–95, http://dx.doi.org/10.1016/j.jnca.2016.08.028.

[12] P. Kumar, R. Merzouki, B. Ould Bouamama, Multilevel modeling of system of systems, IEEE Trans. Syst. Man Cybern.: Syst. 48 (8) (2018) 1309–1320, http://dx.doi.org/10.1109/TSMC.2017.2668065.

[13] K.W. Hipel, L. Fang, The graph model for conflict resolution and decision support, IEEE Trans. Syst. Man Cybern.: Syst. 51 (1) (2021) 131–141, http://dx.doi.org/10.1109/TSMC.2020.3041462.

[14] G.S. Blair, Y.-D. Bromberg, G. Coulson, Y. Elkhatib, L. Réveillère, H.B. Ribeiro, E. Rivière, F. Taïani, Holons: Towards a systematic approach to composing systems of systems, in: Workshop on Adaptive and Reflective Middleware, ARM, 2015, pp. 5:1–5:6, http://dx.doi.org/10.1145/2834965.2834970.

[15] D. Fensel, Ontologies: A Silver Bullet for Knowledge Management and Electronic Commerce, Springer, 2001.

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

[17] A. Elhabbash, V. Nundloll, Y. Elkhatib, G.S. Blair, V. Sanz Marco, An ontological architecture for principled and automated system of systems composition, in: Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS), 2020, http://dx.doi.org/10.1145/3387939.3391602.

[18] X. Ye, Towards a reliable distributed web service execution engine, in: International Conference on Web Services, ICWS, IEEE Computer Society, 2006, pp. 595–602, http://dx.doi.org/10.1109/ICWS.2006.131.

[19] L. Baresi, L. Pasquale, Live goals for adaptive service compositions, in: Workshop on Software Engineering for Adaptive and Self-Managing Systems, SEAMS, ACM, 2010, pp. 114–123, http://dx.doi.org/10.1145/1808984.1808997.

[20] R.R. Aschoff, A. Zisman, Proactive adaptation of service composition, in: Symposium on Software Engineering for Adaptive and Self-Managing Systems, SEAMS, 2012, pp. 1–10, http://dx.doi.org/10.1109/SEAMS.2012.6224385.

[21] H. Derhamy, J. Eliasson, J. Delsing, System of system composition based on decentralized service-oriented architecture, IEEE Syst. J. 13 (4) (2019) 3675–3686, http://dx.doi.org/10.1109/JSYST.2019.2894649.

[22] G. Fodor, E. Dahlman, G. Mildh, S. Parkvall, N. Reider, G. Miklós, Z. Turányi, Design aspects of network assisted device-to-device communications, IEEE Commun. Mag. 50 (3) (2012) 170–177.

[23] A. Elhabbash, G.S. Blair, G. Tyson, Y. Elkhatib, Adaptive service deployment using in-network mediation, in: International Conference on Network and Service Management, CNSM, 2018, pp. 170–176.

[24] B. Mokhtarpour, J. Stracener, A conceptual methodology for selecting the preferred system of systems, IEEE Syst. J. 11 (4) (2017) 1928–1934.

[25] L. Sabatucci, C. Lodato, S. Lopes, M. Cossentino, Highly customizable service composition and orchestration, in: M. Villari S. Dustdar (Ed.), Service Oriented and Cloud Computing, Springer International Publishing, 2015, pp. 156–170.

[26] M. Kit, I. Gerostathopoulos, T. Bures, P. Hnetynka, F. Plasil, An architecture framework for experimentations with self-adaptive cyber–physical systems, in: Symposium on Software Engineering for Adaptive and Self-Managing Systems, SEAMS, 2015, pp. 93–96, http://dx.doi.org/10.1109/SEAMS.2015.28.

[27] A.R. Sadik, B. Bolder, P. Subasic, A self-adaptive system of systems architecture to enable its ad-hoc scalability: Unmanned vehicle fleet-mission control center case study, in: Proceedings of the 2023 7th International Conference on Intelligent Systems, Metaheuristics & Swarm Intelligence, 2023, pp. 111–118.

[28] S. Frey, A. Diaconescu, D. Menga, I. Demeure, A generic holonic control architecture for heterogeneous multiscale and multiobjective smart microgrids, ACM Trans. Auton. Adapt. Syst. 10 (2) (2015) http://dx.doi.org/10.1145/2700326.

[29] A. Diaconescu, S. Frey, C. Müller-Schloer, J. Pitt, S. Tomforde, Goal-oriented holonics for complex system (self-)integration: Concepts and case studies, in: IEEE International Conference on Self-Adaptive and Self-Organizing Systems, SASO, 2016, pp. 100–109, http://dx.doi.org/10.1109/SASO.2016.16.

[30] P. Hnetynka, T. Bures, I. Gerostathopoulos, J. Pacovsky, Using component ensembles for modeling autonomic component collaboration in smart farming, in: Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS), 2020, pp. 156–162, http://dx.doi.org/10.1145/3387939.3391599.

[31] R. Agarwal, D.G. Fernandez, T. Elsaleh, A. Gyrard, J. Lanza, L. Sanchez, N. Georgantas, V. Issarny, Unified iot ontology to enable interoperability and federation of testbeds, in: IEEE World Forum on Internet of Things (WF-IoT), 2016, pp. 70–75, http://dx.doi.org/10.1109/WF-IoT.2016.7845470.

[32] M.I. Ali, P. Patel, S.K. Datta, A. Gyrard, Multi-layer cross domain reasoning over distributed autonomous IoT applications, Open J. Internet Things (OJIOT) 3 (1) (2017) 75–90.

[33] E. Giallonardo, F. Poggi, D. Rossi, E. Zimeo, Making smart buildings and personal systems cooperate via knowledge base overlays, in: GoodTechs, ACM, 2020, pp. 181–186, http://dx.doi.org/10.1145/3411170.3411261.

[34] M.G. Gillespie, H. Hlomani, D. Kotowski, D.A. Stacey, A knowledge identification framework for the engineering of ontologies in system composition processes, in: International Conference on Information Reuse & Integration, 2011, pp. 77–82, http://dx.doi.org/10.1109/IRI.2011.6009524.

[35] J.-B. Soyez, G. Morvan, R. Merzouki, D. Dupont, Multilevel agent-based modeling of system of systems, IEEE Syst. J. 11 (4) (2017) 2084–2095.

[36] F. Cervantes, F. Ramos, L.F. Gutiérrez, M. Occello, J.-P. Jamont, A new approach for the composition of adaptive pervasive systems, IEEE Syst. J. 12 (2) (2018) 1709–1721, http://dx.doi.org/10.1109/JSYST.2017.2655031.

[37] G. Coulson, G.S. Blair, Y. Elkhatib, A. Mauthe, The design of a generalised approach to the programming of systems of systems, in: Workshop on Autonomic and Opportunistic Computing, 2015.

[38] Z. Fang, System-of-Systems Architecture Selection: A survey of issues, methods, and opportunities, IEEE Syst. J. 16 (3) (2022) 4768–4779, http://dx.doi.org/10.1109/JSYST.2021.3119294.

[39] T. Yamakami, A social dimension view model of divergence of iot standardization, in: L. Barolli, F. Xhafa, N. Javaid, T. Enokido (Eds.), Innovative Mobile and Internet Services in Ubiquitous Computing, Springer International Publishing, Cham, 2019, pp. 738–747.

[40] N. Harrand, F. Fleurey, B. Morin, K.E. Husa, ThingML: A language and code generation framework for heterogeneous targets, in: ACM/IEEE International Conference on Model Driven Engineering Languages and Systems, MODELS, ACM, 2016, pp. 125–135, http://dx.doi.org/10.1145/2976767.2976812.

[41] S. Benkhaled, M. Hemam, M. Djezzar, M. Maimour, An ontology-based contextual approach for cross-domain applications in internet of things, Informatica 46 (5) (2022) http://dx.doi.org/10.31449/inf.v46i5.3627.

[42] F. Corradini, A. Fedeli, F. Fornari, A. Polini, B. Re, FloWare: a model-driven approach fostering reuse and customisation in IoT applications modelling and development, Softw. Syst. Model. 22 (1) (2023) 131–158, http://dx.doi.org/10.1007/s10270-022-01026-9.

[43] L. Daniele, F. den Hartog, J. Roes, The Smart Appliances REFerence (SAREF) Ontology, in: Workshop on Formal Ontologies Meet Industries, 2015, http://dx.doi.org/10.1007/978-3-319-21545-7_9.

[44] P. Levis, E. Brewer, D. Culler, D. Gay, S. Madden, N. Patel, J. Polastre, S. Shenker, R. Szewczyk, A. Woo, The emergence of a networking primitive in wireless sensor networks, Commun. ACM 51 (7) (2008) 99–106, http://dx.doi.org/10.1145/1364782.1364804.

[45] W.S. XG, Review of sensor and observations ontologies, 2011, https://www.w3.org/2005/Incubator/ssn/wiki/Review_of_Sensor_and_Observations_Ontologies.

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

[47] A. Herzog, D. Jacobi, A. Buchmann, A3ME - an agent-based middleware approach for mixed mode environments, in: International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies, UBICOMM, 2008, pp. 191–196, http://dx.doi.org/10.1109/UBICOMM.2008.78.

[48] M.A. Musen, The Protégé Project: A look back and a look forward, AI Matters 1 (4) (2015) 4–12, http://dx.doi.org/10.1145/2757001.2757003.

[49] M. Horridge, S. Bechhofer, The OWL API: A Java API for OWL ontologies, Semant. Web 2 (1) (2011) 11–21.

[50] K. Xu, X. Wang, W. Wei, H. Song, B. Mao, Toward software defined smart home, IEEE Commun. Mag. 54 (5) (2016) 116–122, http://dx.doi.org/10.1109/MCOM.2016.7470945.

[51] B. Vogel, D. Gkouskos, An open architecture approach: Towards common design principles for an IoT architecture, in: European Conference on Software Architecture, ECSA, 2017, pp. 85–88, http://dx.doi.org/10.1145/3129790.3129793.

[52] I. Yaqoob, E. Ahmed, I.A.T. Hashem, A.I.A. Ahmed, A. Gani, M. Imran, M. Guizani, Internet of things architecture: Recent advances, taxonomy, requirements, and open challenges, IEEE Wirel. Commun. 24 (3) (2017) 10–16, http://dx.doi.org/10.1109/MWC.2017.1600421.

[53] A.H. Ngu, M. Gutierrez, V. Metsis, S. Nepal, Q.Z. Sheng, IoT middleware: A survey on issues and enabling technologies, Internet Things J. 4 (1) (2017) 1–20, http://dx.doi.org/10.1109/JIOT.2016.2615180.

[54] Y. Elkhatib, Building cloud applications for challenged networks, in: R. Horne (Ed.), Embracing Global Computing in Emerging Economies, in: Communications in Computer and Information Science, vol. 514, Springer International Publishing, 2015, pp. 1–10, http://dx.doi.org/10.1007/978-3-319-25043-4_1.

[55] M. Soliman, T. Abiodun, T. Hamouda, J. Zhou, C.-H. Lung, Smart home: Integrating internet of things with web services and cloud computing, in: IEEE International Conference on Cloud Computing Technology and Science, vol. 2, 2013, pp. 317–320, http://dx.doi.org/10.1109/CloudCom.2013.155.

[56] M. Chiang, T. Zhang, Fog and IoT: An overview of research opportunities, IEEE Internet Things J. 3 (6) (2016) 854–864, http://dx.doi.org/10.1109/JIOT.2016.2584538.

[57] M. Noura, M. Atiquzzaman, M. Gaedke, Interoperability in internet of things: Taxonomies and open challenges, Mob. Netw. Appl. 24 (3) (2019) 796–809, http://dx.doi.org/10.1007/s11036-018-1089-9.

[58] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A.D. Joseph, R.H. Katz, S. Shenker, I. Stoica, Mesos: A platform for fine-grained resource sharing in the data center, in: Symposium on Networked Systems Design and Implementation, NSDI, USENIX, 2011.

[59] L.M. Vaquero, F. Cuadrado, Y. Elkhatib, J. Bernal-Bernabe, S.N. Srirama, M.F. Zhani, Research challenges in nextgen service orchestration, Future Gener. Comput. Syst. 90 (2019) 20–38, http://dx.doi.org/10.1016/j.future.2018.07.039.

[60] B. Varghese, P. Leitner, S. Ray, K. Chard, A. Barker, Y. Elkhatib, H. Herry, C.-H. Hong, J. Singer, F.P. Tso, E. Yoneki, M.F. Zhani, Cloud futurology, IEEE Comput. 52 (9) (2019) 68–77, http://dx.doi.org/10.1109/MC.2019.2895307.

[61] D. Kakadia, Apache Mesos Essentials, Packt Publishing Ltd, 2015.

[62] Y. Elkhatib, B.F. Porter, H.B. Ribeiro, M.F. Zhani, J. Qadir, E. Rivière, On using micro-clouds to deliver the fog, Internet Comput. 21 (2) (2017) 8–15, http://dx.doi.org/10.1109/MIC.2017.35.

[63] K. Nakano, S. Olariu, Randomized leader election protocols in radio networks with no collision detection, in: Algorithms and Computation, Springer Berlin Heidelberg, Berlin, Heidelberg, 2000, pp. 362–373.

[64] A. Varga, R. Hornig, An overview of the OMNeT++ simulation environment, in: Conference on Simulation Tools and Techniques for Communications, Networks and Systems, ICST, 2008, p. 60.

[65] R.V. Levine, A. Norenzayan, The pace of life in 31 countries, J. Cross-Cult. Psychol. 30 (2) (1999) 178–205.

[66] H. Barbosa, M. Barthelemy, G. Ghoshal, C.R. James, M. Lenormand, T. Louail, R. Menezes, J.J. Ramasco, F. Simini, M. Tomasini, Human mobility: Models and applications, Phys. Rep. (2018).

[67] Mesosphere, Mesosaurus, 2016, https://github.com/mesosphere/mesosaurus.

[68] Royal Academy of Engineering, Living without electricity - one city's experience of coping with loss of power, 2016.

[69] J.Q. Cui, S.K. Phang, K.Z.Y. Ang, F. Wang, X. Dong, Y. Ke, S. Lai, K. Li, X. Li, F. Lin, J. Lin, P. Liu, T. Pang, B. Wang, K. Wang, Z. Yang, B.M. Chen, Drones for cooperative search and rescue in post-disaster situation, in: IEEE International Conference on Cybernetics and Intelligent Systems (CIS) and IEEE Conference on Robotics, Automation and Mechatronics, RAM, 2015, pp. 167–174, http://dx.doi.org/10.1109/ICCIS.2015.7274615.

[70] M. Narang, S. Xiang, W. Liu, J. Gutierrez, L. Chiaraviglio, A. Sathiaseelan, A. Merwaday, Uav-assisted edge infrastructure for challenged networks, in: IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), 2017, pp. 60–65, http://dx.doi.org/10.1109/INFCOMW.2017.8116353.

[71] A. Kumbhar, İsmail Güvenç, S. Singh, A. Tuncer, Exploiting LTE-advanced HetNets and FeICIC for UAV-assisted public safety communications, IEEE Access 6 (2018) 783–796, http://dx.doi.org/10.1109/ACCESS.2017.2776120.

[72] F. Kon, F. Costa, G. Blair, R.H. Campbell, The case for reflective middleware, Commun. ACM 45 (6) (2002) 33–38, http://dx.doi.org/10.1145/508448.508470.

[73] J.O. Kephart, D.M. Chess, The vision of autonomic computing, Computer 36 (1) (2003) 41–50, http://dx.doi.org/10.1109/MC.2003.1160055.

[74] Z. Zhang, Y. Elkhatib, A. Elhabbash, NLP-based generation of ontological system descriptions for composition of smart home devices, in: International Conference on Web Services, ICWS, IEEE, 2023, http://dx.doi.org/10.1109/ICWS60048.2023.00055.

[75] Z. Wang, Y. Elkhatib, A. Elhabbash, HolonCraft – an architecture for dynamic construction of smart home workflows, in: Conference on Future Internet of Things and Cloud (FiCloud), IEEE, 2022, pp. 213–220, http://dx.doi.org/10.1109/FiCloud57274.2022.00036.

Dr. Abdessalam Elhabbash is a Lecturer in Self-adaptive Distributed Systems at Lancaster University. He researches how to develop systems that can learn and adapt to changes in their environment. His current interests include adaptive system composition, cloud adaptive decision support, adaptive distributed computing, and trustworthy self-adaptive systems.

Dr. Yehia Elkhatib is a Reader (Associate Professor) at the School of Computing Science, University of Glasgow, UK. He works on using data-driven approaches to improve the performance and programmability of distributed systems. This includes work on optimized deployment on cloud and edge resources, post deployment system composition, and intent-driven network architectures.

Dr. Vatsala Nundloll has a PhD in Computer Science. She has worked on diverse research projects, investigating into data mining, sensor networks, data integration and prediction.

Dr. Vicent Sanz Marco is a researcher at the National Institute of Advanced Industrial Science and Technology (AIST) in Japan. He received his PhD in Computer Science from the Universitat Politècnica de València in 2018. His research interests include self-adaptive systems, embedded systems, and machine learning.

Prof. Gordon Blair is Head of Environmental Digital Strategy at UKCEH. He is also a Distinguished Professor of Distributed Systems at the School of Computing and Communications - Lancaster University. He is also Co-Director of the Center of Excellence in Environmental Data Science (CEEDS), a joint initiative between UKCEH and Lancaster University.