System of System Composition Based on Decentralized Service-Oriented Architecture

Hasan Derhamy, Jens Eliasson, and Jerker Delsing, *Member, IEEE*

***Abstract*—As society has progressed through periods of evolution and revolution, technology has played a key role as an enabler. In the same manner, mechanical machines of the 1800s drove the industrial revolution, now digitalized machines are driving another industrial revolution. Manufacturers are increasing the digital footprint on the factory floor. It is challenging to harness the vast amounts of data generated, stored, analyzed, archived, and returned. Data centralization has several well-known challenges, such as collection bottlenecks, secure retrieval, single point of failure, and data scheme fragility as data heterogeneity increases. This paper proposes a method of information distribution based on the principle of *data at its source*. It proposes that contextual data be used at runtime through the creation of dynamic queries that build compositions of different systems. Such system of systems (SoS) compositions handle the flow of data across its life cycle and present it as information to the initiating system. The proposal starts by creating a graph model of the Arrowhead framework. Then, building on the graph model, the query-based approach for specifying, validating, and forming the SoS is proposed. The proposed graph model allows for unambiguous description of systems and their interrelations, including security relations. The proposed composer operates on the edge computing hardware and gives the production floor the ability to extract information without impacting the overall operation of the factory.**

***Index Terms*—Arrowhead framework (AF), edge computing, graph theory, Hypermedia as the engine of application state (HATEOAS), industrial Internet of Things (IIoT), information centric networking, RAMI4.0, RESTful, service-oriented architecture, system of systems (SoS), systems theory.**

I. Introduction

INITIATIVES in the manufacturing industry are looking to boost the productivity by leveraging advances in connectivity. The increase in connected devices, in large part, is due to increases in low cost networking and computing. This has meant that there is a significant increase in the volume and flow of information. All these developments have occurred within a landscape of ISA 95 [1] deployments. ISA 95 supports vertical integration and information flow. However, this also created bottlenecks from many perspectives, which are as follows,

1) *Storage:* Centralized storage exhibits database bottlenecks, meaning that storing and retrieving data is usually restricted to ensure that the database is not overloaded with requests. Local storage handles requests within its local scope, therefore, fewer requests would be present. Also, if a decentralized database does experience reduced performance due to overloading, the area of impact is local and does not reach to wider applications.
2) *Communication:* Centralized storage must have strong networks. The network becomes a bottleneck as traffic increases, meaning close monitoring and costly backbone infrastructure is required. Redundant networks become a necessity to reduce the down time in case of the network failure. Performance degradation on the network also has a wide impact. On the other hand with local storage, network performance degradation has a lesser impact on normal operations.
3) *Technology:* Centralized systems are sensitive to disruptions and so, change process must ensure normal operation. Also, technology selection is limited to those that have been tested for high performance operation. With a localized storage system, the sensitivity to disruption is reduced and the options to work around the disruption increases. Also, technology selection can evolve more readily, with a more heterogeneous set of technologies better suited for specific requirements.
4) *Engineering:* Centralized systems will have a specific set of competencies such as data warehousing, network administration, server maintenance, etc. These specialists are required to handle scaling up infrastructure and tools. While localized systems do not require specialized knowledge for scale, they simply do not reach such demanding levels and so competent general engineering skills are sufficient.

Building on from ISA 95, the Industry 4.0 initiative has proposed the Reference Architectural Model for Industry (RAMI) 4.0 [2] and I4.0 component model [3]. The I4.0 component model captures the notion of an administration shell that abstracts the digitalized equipment and products with high levels of connectivity. The RAMI 4.0 captures a three-dimensional cube for modeling architectural solutions. It presents I4.0 components at different "hierarchies" must be designed over a complete "life-cycle" and must participate in functional "layer." Or from a layer point of view, a single functional "layer" cannot be confined to a single level of the "hierarchy." Rather a "layer" is spread across many I4.0 components at different levels of the "hierarchy." This is shown in Fig. 1.

Manuscript received June 17, 2018; revised October 24, 2018; accepted December 24, 2018. Date of publication February 11, 2019; date of current version November 22, 2019. This work was supported in part by the Far-Edge project and in part by Productive 4.0 project. *(Corresponding author: Hasan Derhamy.)*

The authors are with the EISLAB, Department of Electrical and Computer Engineering, Luleå University of Technology, 971 87 Luleå, Sweden (e-mail: hasan.derhamy@ltu.se; Jens.Eliasson@ltu.se; jerker.delsing@ltu.se).

Digital Object Identifier 10.1109/JSYST.2019.2894649

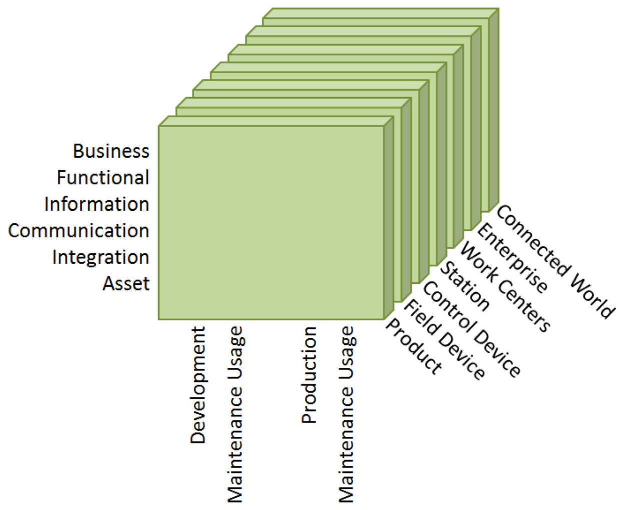

Fig. 1. RAMI4.0 three perspective cube. Figure adapted from [2].

A central philosophy of the RAMI 4.0 approach is that connectivity and integration is no longer purely vertical. I4.0 components can communicate with one and other vertically, horizontally, or diagonally. Many I4.0 components spread across the component hierarchy are participating in the information layer each contributing and consuming data. A smooth flow of data in the information layer enables decentralized decision making on the factory floor, reducing reliance on manufacturing execution system (MES) cloud so that work centers, cells, and stations can achieve high level of autonomy. The reduced reliance on the centralized decision support has the added benefit of reduced information system coupling between physical work cells so that a malfunction in one work cell does not lead to a performance degradation in adjacent work cells.

Development teams must rely on external resources to be able to handle the complexity and size of the automation solutions. Integrating commercial off the shelf (COTS) components from multiple vendors indicates that the systems may not all belong to a single owner. For example, using equipment as a service may be a viable manner to distribute life-cycle costs (design, development, and maintenance). Such integrations demands regular review of security assumptions regarding privacy, confidentiality, and trust. Heavy integration costs and information blockages must be avoided between multivendor systems. Requiring that systems are able to learn of or establish meaningful, trusted, and confidential connections for exchanging data.

The final aspect and consideration, is that software technology evolves at a rapid pace, much more quickly than the traditional mechanical or electronic technologies. Software development practices tend toward adoption of Agile [4] practices. This entails short development cycles with small releases and fast deployment. The solution, therefore, must support incremental change with minimized impact due to disruption of operations. In addition, the approach toward and treatment of software technology must be one of embracing change. In other words, a successful solution must have a clear path for change and ensure that it is not bound to a single technology that inhibits change.

Service-oriented architecture is one such architectural approach that tackles technology abstraction through abstract messaging interfaces. However, industrial Internet of Things (IIoT) systems go beyond pure software and include hardware. Physical, (real world) dependencies between services offered from a single hardware means, utilizing the system theory with services, provides a more meaningful modeling expression. Modular and independent systems, which can be pure software or embedded software, are modeled as black box with technology independent service interfaces between one and other for communication and collaboration.

A. *Requirements for Industrial Information Distribution*

Industrial engineers and designers need immediate access to data without dependence on high availability network infrastructure, rigid technology, and specialist competencies. Domain specialists on the shop floor must be able to access local data and change data routing/flows without direct involvement of information technology (IT) specialists. The solution must support independent evolution of both the tools for data access and the sources of data. The specialist support and restrictive technology choices required of both purely centralized and peer-to-peer solutions do not meet the aforementioned criteria. In both cases, system change is dependent on IT competencies or solution-specific technology.

Therefore, a successful solution for decentralized information distribution would require the following characteristics.

1) Ability to operate in an islanded mode without a centralized data store. Creating robustness to network performance degradation and supporting remote “no network” deployments.
2) Describe the data flow specification in a human readable and machine executable manner.
3) It must be technology agnostic not limiting solutions due to technological choices, for example, supporting both a push and pull communication paradigm.
4) Support for fine grained access control, enabling change by industrial designers and restricting untrained or adversarial change.
5) Loose couple among data sources, processors, and sinks. The solution must cope with rapid and unexpected changes to the factory floor as working areas are rebalanced or faulty machines and tools are replaced.

B. *Contribution*

The proposed solution is designed to satisfy the requirements from Section I-A. It proposes a software system that is able to process information queries and setup communication routing that produce and move information. A graph model is proposed to support reasoning on services, service types, systems, and security. The graph model developed in this research is used at runtime to build an online picture of the IIoT application space.

It is proposed to use a standard graph query language, Open Cypher, in a novel manner to describe the composition of systems and services for the information flows. It is in this way that the graph model is used to describe and form the system of system (SoS).

The proposed solution includes in-network data processing. The composed information flow is responsible for the data before it reaches the consuming system. The processing can be aggregation, filter, format manipulation, or signal processing.

The proposed SoS compositions are activated only where and when they are needed, thereby reducing unnecessary information flows.

In the next section, related work is introduced before presenting the proposed solution in Section III.

II. Related Works and Technologies

In this section, some work related to the challenges is presented and some technologies used within the solution are presented.

A. Mashups

The notion of mashups comes from mixing and combining the presentations of different data sources into a single view. For example, Twitter and Facebook feeds can be viewed on one page together with Instagram updates. This page would be a mashup of three different social media sources. Another example Trendsmap [5] is a web application that utilizes Twitter and Google Maps application program interfaces (APIs) to show the Twitter activity by location. Meaning that users are able to view what the trending Twitter topics are in particular cities. The same can be applied for sensory data from multiple sensors or equipment key performance indicators (KPI) along with the current production orders.

Mashup approaches such as the one offered by ThingWorx [6] are a great example of run-time-based creation of information flow. ThingWorx is a business-oriented platform for connecting IoT devices with different protocols and routing data between devices. Access to IoT data and so the visualization of related data from different sources is a primary driver. The Web of Things have developed WoTKit [7] for building mashups. The WoTKit targets web mashups and meaning that only web servers and clients are used. However, these solutions both relay on the centralization of either the data or the distribution of the data. Node-red [8] is Javascript-based graphical environment that can mash together different services into a composition. Its structure is similar to typed pipe and filter diagrams. The IFTTT ("if this then that") platform [9] is an IoT mashup cloud that supports applets and services that can connect heterogeneous things together. Using chained conditionals, it is a strong commercial mashup platform.

B. Arrowhead Framework (AF)

The AF [10] is a supporting technology for the proposed solution. The AF is a service-oriented architecture (SOA)-based SoS integration framework. Within Arrowhead, a system can provide and consume services while systems themselves are hosted upon devices. This allows hierarchical meta-data, split between devices, systems, and services. Reusable services abstract the physical and digital implementations. This means that applications can be specified using purely functional models.

Arrowhead systems are composed into SoS using orchestrations. A single system can provide and consume multiple services and can interact with multiple other systems. Interactions between systems can only occur through these services. For communication between systems, there must be matching service provider and consumer interfaces. The AF documentation structure [11] defines how to specify a service interface.

C. Next Generation Access Control (NGAC)

The NGAC is an attribute-based method of authorizing user operations on objects. By utilizing attributes, NGAC enables fine grained access control based on the context. A user that is allowed to operate on an object in one location may not have access from a different location, simply due to the user attribute association. Alternatively, if the object is moved to a new location (its attributes change), then it may reject operations from all users. Access control rules can then be expressed based on the context. Achieving such complex awareness becomes unmanageable with static rules. Therefore, NGAC uses a graph approach to describe the rules.

The NGAC defines objects, users, attributes, and operations as nodes within a graph. A policy must then define the required attributes by a user to reach an operation and an object to reach the same operation. When evaluating the policy, the evaluation engine must have reliable information regarding the current attributes of the object and the user. Therefore, a secure information extraction solution must work with context aware security such as NGAC.

D. In-Network Processing

In-network processing, a method treating the network as a database. As described by Tannenbaum and van Steen [12], the network will process raw data and aggregate or route the information to the calling application. Using IEEE 802.15.4 wireless networks, TinyDB [13] can perform in-network processing. TinyDB receives declarative queries that are passed down the network mesh. Nodes that are capable of handling the request will process it and pass the result back up the route. Each parent node of the mesh will aggregate the child node responses. Meaning that there is a natural formation for query delivery and results aggregation. In-network processing can also be applied to general computer networking. Any network can act as a database that can be queried for information that may not exist until the query is processed. One of the benefits of the in-network processing are to avoid network and memory overheads of centralizing data and information. Furthermore, by aggregating and preparing results, it can help reduce the complexity systems consuming data from multiple sources.

E. Information Centric Networking (ICN)

The ICN is a comprehensive approach toward in-network processing. The ICN proposes to move away from host addressing and to simply perform named information queries [14]. This is highly suitable for many Web examples such as video streaming and file sharing. Interest is expressed for information, and regardless of where the information exists, the network will seek out the information and return it to the caller. One of the issues with the native ICN is in deployment, ICN overlays are proposed so that the ICN and the transmission control protocol/Internet protocol can co-exist [15]. Service Centric Networking (SCN) [16] extends the ICN to include processing functions within the network infrastructure. This recursive operation, information $= f(\text{information})$, turns the existing information into new information.

F. Service Composition

The SOA-based applications use composition as a principle for their design. Software service building blocks are composed into interacting clients and servers with the sum of the interactions becoming the complete application. In [17], Eslamichalandar *et al.* present a survey on service composition adaptation methods. These methods look to enable adaptation of compositions on a service interface and behavior levels. They review the work by Tan *et al.* in [18] in which it is suggested to use the reachability graph to construct a data mapping between the two collaborative services. In [19], a continuous query is applied to input streams of a complex event processor. This query performs the required transformation on the data stream and passes the result to the connected service. Formal modeling languages such as finite-state machines, Petri-nets, and Business Process Execution Language are used.

In [20], Bouveret *et al.* present the concept of conditional importance networks (CI-net). This study utilizes the graph theory and combinatorial logic to express the usage preferences. A CI-net is built up based on the conditional important statements. This graph can be used with the presence of contextual information to find a suitable set of network nodes (or in the case of their work, goods, and materials). In the case of composition, it is needed to express preferences in service provision. Hence, this study could be directed toward the service composition problem of the SOA.

III. Proposed Solution

The solution proposes a software system, called the system composer, that is responsible for receiving the information query, processing the query against the system, service and access control graphs, formulating an information flow path, and issuing the orchestration rules for the SoS. The system that originated the information query receives instruction in the form of an address that it can message to retrieve the result of the SoS. Alternatively, an MQTT topic could be provided to publish/subscribe channel for the system to receive information from. This is the simple interface required of the system composer for the proposed solution. The system composer service interface is discoverable through a service registry and its access rights is secured through an NGAC-enabled authorization system. Internally, the system composer relies on a graph of the local scope systems, service instances, and types and access rights. This graph is based on the proposed graph model presented in Section III-A. It is built through interaction with Arrowhead; device, system, and service registries; and with service inventory and authorization system. The system composer is stateless in terms that none of its memory is persistent. The graph is rebuilt on startup and maintained by active communication with the mentioned AF core support systems. The information query is expressed in a graph query using Open Cypher. The complete solution is presented in the following sections, first, a graph model of an SoS-enabled SOA framework, the AF, is presented.

dev
Device
sd
Service Description
Sys
System
idd
Service Interface

Fig. 2. Building blocks of the AF.

A. Proposed Graph Model

The AF has three basic entities: devices, systems, and services. Devices are physical entities with processing and networking capability. Systems are software entities that are hosted on devices and that provide and consume services. Services are goal-oriented interfaces that form the basis for collaboration between systems. Services are then split into two entities.

1) The service description defines the goal of the service.
2) The interface design description defines the interaction pattern, communications protocol, data structure, and semantics of the service.

As shown in Fig. 2, these four entities form the base vertices of the graph.

The AF uses these primary blocks to builds SoS. To build an SoS, the primary entities must be related to one another. These relations are are as follows.

1) A System is Hosted_By a Device.
2) A Service Description is Provided_By a System (Provider).
3) A Service Interface is Offered_By a System (Provider).
4) A System (Consumer) Requires a Service Description.
5) A System (Consumer) Supports a Service Interface.
6) A Service Description is Implemented_By a Service Interface.

With this set of edges, it is possible to construct a simple path between systems. For example, the path can be functional using Required and Provided_By edges, or a communications path based on Supported and Provided edges. A sub-graph with a functional path is illustrated in Fig. 3. Here, it can be seen that the path makes up a bipartite graph. There are only edges between the two sets of System Nodes and Service Definitions. There are no edges within either set.

A functional specification of the SoS can be formed by looking only at the function graph. This can be formalized by

$G = \{V,E\}$
where
$V = \{A,B\}$
where $A =$ Set of systems
and $B =$ Set of service definition

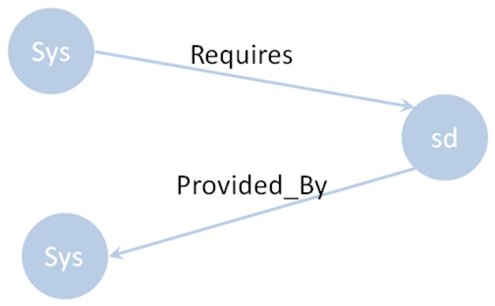

Fig. 3. First SoS: A functional path between two system.

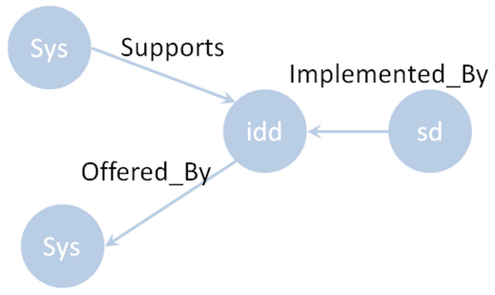

Fig. 4. First SoS: A bipartite communication graph with two participant systems.

G={V,E}
Where
V={A,B}
Where A = Set of Systems
and B = Set of Service Interfaces (with matching shared Service Definitions)
$| B| = | A| - 1$
$| E| = | B| * 2$
or
$| E| >= (| A| - 1) * 2$
therefore
$| V| = | A| * 2 - 1$

Fig. 5. Complete communications graph is formalized.

$|B| = |A| - 1$
E = B * 2.

The communications graph is built from a set of participant systems and service interfaces. To build this graph, the service definition is used as an anchor to the service interfaces. In this way, it is possible to move from the functional sub-graph to the communications sub-graph.

This sub-graph can be seen in Fig. 4.

The communications graph is a concrete interaction path between participant systems. A formal description of the communications path can be seen in Fig. 5. It can be used to validate the interaction path.

Based on the aforementioned graph definitions, it can be deduced that a graph is disconnected whenever the number of edges is less than $|B|$ then one or more required Service Definitions are missing and the SoS is incomplete. If the SoS is incomplete, then the SoS must be re-engineered. Either requirements can be modified for example remove unattainable requirements. Or adding new systems and enhancing existing systems with additional services.

Dynamic bridging is may be an alternative to full re-engineering. For example, where a missing edge is detected within the communications graph, this indicates that a service interface mismatch exists. In this case, a translator can be injected to the graph. Dynamic systems are discussed in detail in Section V-A. Once a communications graph has been established. The security authorization path must be checked.

usr
attr
obj
op

Fig. 6. NGAC graph entities/nodes. User as “usr,” Attribute as “attr,” Object as “obj,” and Operation as “op.”

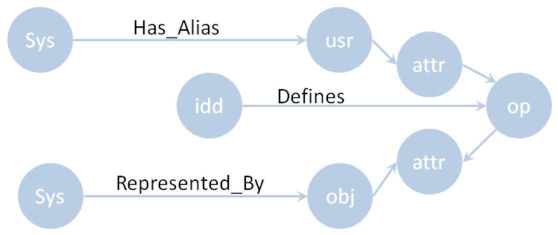

Fig. 7. NGAC: An authorization path between two system.

B. NGAC Graph Entities

The connection to the Arrowhead entities. Access control is a core element to secure SoS. Access control means restricted system interaction (service exchange) to only authentic and authorized peers. The NGAC [21] is a standard that utilizes the graph-based attribute-based access control. The NGAC entities can be seen in Fig. 6. To use the NGAC with Arrowhead, either an equivalence or a relation must be formed between the two models. Here, it is proposed that a relation mapping is formed as follows.

1) An user entity—aliases—a consuming system.
2) An object entity—represents—a providing system.
3) An operation entity is—defined—by a service interface.

Fig. 7 shows the graph relations between the AF nodes and the NGAC nodes. Utilizing the system nodes and the service interface node as anchors, the related user, object, and operation nodes can be found. Next, a path must be traversed between the two systems through only the NGAC graph nodes.

A complete path, in the NGAC graph, between the systems means that the required attribute permissions exist for the consuming system to invoke the operation upon the providing system. With these three primary graphs, it is possible to build

TABLE I
SYSTEM COMPOSER COLLABORATES WITH EXISTING SYSTEMS THAT STORE THE NEEDED INFORMATION TO BUILD THE GRAPH

Data Store	Description
Device Registry	The Device Registry contains records of each device and associated meta-data, such as physical location and operating conditions.
System Registry	The System Registry contains records of each system and associated meta-data, such as the physical device it is hosted on and what are the systems provide capabilities and consume requirements.
Service Registry	The Service Registry contains records of each service available (live) for consumption and associated meta-data, such as the responsible system, a reference to the service type and operating capabilities.
Service Inventory	The Service Inventory contains records of each service definition and possible service interface combinations.
Authorization System	The Authorization System contains the access control Policy Information at a role or attribute level.

validated and secured SoS. The next section will discuss the method of utilizing the graph models.

IV. PROPOSED SoS FORMATION

To build an SoS, there are multiple steps. The fundamental steps are as follows: 1) specification; 2) validation; and 3) run-time binding. The proposed solution can be used in each step.

1) *Specification*: In order to specify an SoS, an engineer must be able to express the expected systems and their expected interconnections. This can be expressed naturally using the already defined vertices and edges. Using the set theory, it is also possible to generate the mathematical representation of the SoS from the graph.
2) *Validation:* Utilizing the mathematical representation, it is possible to create a list of components that are expected to be present. This list can be used to provide accurate requirements to engineers and developers. It can also be used to check if a specification can be met by currently available systems.
3) *Run-Time Orchestration:* When the SoS specification is run against a live graph, the service exchanges between systems are identified. These identified service exchanges are what generate the orchestration rules.

A. Building the Graph

The data required to build the graph are distributed among different data sources. To perform the graph queries, the data must be gathered together and sorted into a graph database. The data are collected from data stores, which are described in Table I.

The data stored in each of these systems must be retrieved through service interfaces (see Fig. 8). In the case outlined in this paper, the interfaces are defined by the AF. These interfaces could be defined according to any preferred approach.

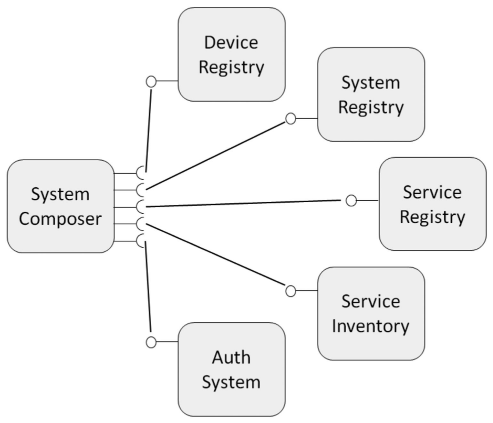

Fig. 8. System composer interaction with AF core systems.

A full graph of the local cloud can be maintained by synchronizing the different data stores. The graph data does not change rapidly as there is limitations on how quickly a Cyber Physical system can be relocated. Depending on the protocol choice, this could require polling the data stores for changes. However, to avoid this, the data stores can notify the graph that a change has occurred, allowing the pull from data sources to be event driven. At no stage does the graph make changes to the data stores. Therefore, write synchronizations are not required.

As described in the next section, the queries made to the graph are only reading based, changes such as injection of local bridges are either not reflected in the data stores, or are changed through the normal device, system, and service registration.

B. Querying the Graph

The whole value of the graph model is the ability to query the connected data. Before diving into the mechanisms, the objective of the query can be listed as follows:

1) to find a functional path connecting a set of systems;
2) to find a communications path connecting a set of systems;
3) to find an authority path connecting a set of systems;
4) to identify a system based on its relations.

1) Function Path: To find the function path a combination of known systems and service interfaces must be anchored (known). The information that makes up this graph is formed using System Registry and Service Inventory data. Given a start system, an ordered set of services, and an end system, a functional path can be built using the following query:

$$\text{MATCH p} = (a1 : System\{name : \text{“}a\text{”}\}) \\ - [\text{:Requires}||\text{Offered\_by*}] - > \\ (a2 : System\{name : \text{“}e\text{”}\}) \text{ RETURN p.} \quad (1)$$

The query will return all nodes and relations that create the path. The nodes will consist of a set of systems and a set of service definitions. These two sets are then used to find the communications path.

2) Communications Path: To find the communications path, each of the service definition and system pairs must form into a triple with the interface definition. A consuming system supports

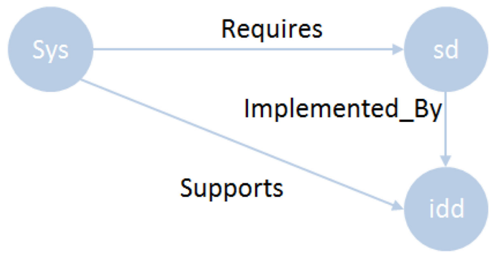

Fig. 9. Interface matching for a consumer system and service.

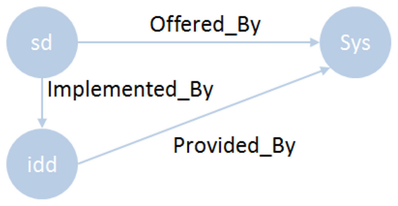

Fig. 10. Interface matching for a provider system and service.

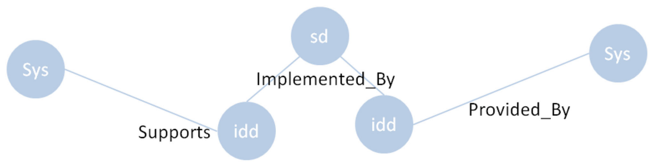

Fig. 11. Broken communications path is shown here. The two systems support and provide different interface designs for the same service definition.

an interface definition and the interface definition is offered by a providing system. Hence, Fig. 9 shows the result of the query in the following equation, that is, the consumering triple.

$$\text{MATCH}\\ (a: System\{name: \text{“}a\text{”}\}) - [\text{:Supports}] - > (inst)\\ (inst) - [\text{:Implements}] - > (\text{b:Type}\{name: \text{“}b\text{”}\})\\ \text{RETURN inst.} \quad (2)$$

Fig. 10 shows the results of the provider side query shown in the following equation:

$$\text{MATCH}\\ (a: System\{name: \text{“}c\text{”}\}) < -[\text{:Provided\_by}] - (inst),\\ (inst) - [\text{:Implements}] - > (\text{b:Type}\{\text{name:“b”}\})\\ \text{RETURN inst.} \quad (3)$$

A connected communication path will match to a single interface definition that means the two systems utilize interoperable services. This is shown in Fig. 4, which has a connected functional path between the two participant systems. Where a communication path is not found, the resulting graph will look like Fig. 11. In this situation, the SoS can be adapted to create a connected communication path.

This case is covered in the system composer implementation section that uses the translator to inject a local bridge and create a communications path.

$$\text{MERGE}\\ (\text{dev:Device}\{\text{name:’A8076’}\}),\\ (\text{loc:Location}\{\text{name:’A2301’}\}),\\ (dev) - - > (loc),$$

Fig. 12. Open Cypher query for storing meta-data into the graph database.

3) Authority Path: Using the Arrowhead and NGAC relationships, it is possible to shift from the functional domain to the security domain. An attribute path must be found between the providing system and consuming system, which passes through the operations that represent the service instances. The attribute designation is made through a secure management portal. The requirement for the attribute policy is also made through a secure management portal. Hence, this part of the graph is certainly read only. A break in the security path will mean that there is no authorization for the service exchange between the participant systems.

The graph query to find the security path must first identify the nodes that represent the users, objects, and operations. Using the definitions from Fig. 7, each interface definition is mapped to a single NGAC operation. A system can have at most one NGAC Object and one NGAC User.

A path must be found between the *user* that is the alias of the Consuming *System*, the *operation* defined by the *Interface Definition*, and the *object* represented by the Providing *System*. Fig. 7 shows a generic example of the authorization path.

V. Implementation of SoS Composer

It is now possible to use the graph model and SOA-based data stores to design a solution that in (soft) real time will compose the distributed functionality across the SoS.

The SoS Composer is a tool that can be utilized by technical and non-technical users to compose new systems combinations. It uses the processes described to build a graph of the device, system, service, and security landscape.

To navigate the graph, an anchor needs to be located. This is the starting system and is usually found through direct reference to a unique system name, or through a unique combination of meta-data connections.

The system composer is itself a participant in an SoS. Fig. 8 draws the SoS required to generate the graphs. The graphs are kept up to date through polling and notifications between the systems in Fig. 8.

Utilizing Open Cypher, the responses from the data stores are stored in a Neo4J instance using the following queries.

A merge operation is used to avoid creating duplicate nodes and edges. Device data can have many nodes connected to it. They represent connected meta-data. For example, a device is connected to a location. This can be physical location or logical location. No edge label is required for meta-data nodes (see Fig. 12).

System data will include an association to a device that is hosting the system, the services provided and consumed from the system, and any connected meta-data associated with the

```
MERGE
        (sys:System{name:'Nutrunner'}),
        (jobServiceOffering:ServiceType{name:'EquipmentJob'}),
        (notifier:ServiceType{name:'EquipmentNotification'}),
        (ins:ServiceInstance{name:'showworkorder-presenter'}),
        (jobServiceOffering) − [:OFFERED_BY]− > (sys),
        (notifier) − [:OFFERED_BY]− > (sys),
```

Fig. 13. Open Cypher query for storing data regarding a system into the graph database.

```
MERGE
        (ser:ServiceType{name:'showworkorder'}),
        (sys:System{name:'presenter'}),
CREATE
        (ins:ServiceInstance{name:'showwork'}),
        (sys) − [: OFFERS]− > (ins),
        (ins) − [: OF]− > (ser);
```

Fig. 14. Open Cypher query for storing security data into the graph database.

```
MERGE
        (sd:ServiceDefinition{name:'EquipmentJob'}),
        (idd:ServiceType{name:'EquipmentJobCoAP'}),
        (sd) − [:IMPLEMENTED_BY]− > (idd)
```

Fig. 15. Open Cypher query for storing service information from the service inventory.

system. A sample Cypher command to store the minimum system data is shown in Fig. 13.

To store the security attribute data and policy data, the Open Cypher query is shown in Fig. 14.

To store the service data (resulting from a query to the Service Inventory), the query shown in Fig. 15 must be executed.

Once the data have been collected and the graph has been built, the graph can be presented to a user to graphically draw the SoS composition.

The service interconnections are primarily and in some cases exclusively used to specify the SoS. This is achieved by querying the graph with a MATCH query like the one shown in Fig. 16.

When only validating the possibility of an SoS intermediate and end systems can be left anonymous. Otherwise, while generating the SoS, the path query should return the intermediate and end systems. It is possible that alternative paths exist, meaning that there are alternative service providers that can satisfy the specification. In the next stage, concrete SoS compositions can be generated by looking at the communications path. To do this, shared service interfaces that implement the service definitions must be found between the intermediate systems. The queries defined earlier are used to build the communications path, as shown in Fig. 17.

So long, as the consumer service interface “consumerInt” and the provider service interface “providerInt” are the same node, then this means that the communication path is valid. Building the communications path means performing this query on each pairing of systems found from the functional path query.

```
MATCH
        (sys:System{name:”}),
        (sd-1:ServiceDefinition{name:”}),
        (sd-2:ServiceDefinition{name:”}),
        (sd-3:ServiceDefinition{name:”}),
        (sys) − [:Requires]− >
        (sd-1) − [:Offered_By]− >
        (sys-1:System) − [:Requires]− >
        (sd-2) − [:Offered_By]− >
        (sys-2:System) − [:Requires]− >
        (sd-3) − [:Offered_By]− > (sysEnd:System)
RETURN
        sys-1, sys-2, sysEnd
```

Fig. 16. Open Cypher query for retrieving an SoS. This graph represents an SoS specification.

```
MATCH
        (sysCon)-[:Supports]->(conInterface),
        (conInterface)<-[:Implemented_By]-(sd),
MATCH
        (sysPro)<-[:Provided_By]-(proInterface),
        (proInterface)<-[:Implemented_By]-(sd)
WHERE
        (conInterface = proInterface)
RETURN
        conInterface, proInterface
```

Fig. 17. Query to identify the matching service interface, given a common service definition and two participant systems.

The communications path is complete once the bipartite graph connecting systems and service interfaces has been formed. In case that the communications bipartite graph is disconnected, it can be assumed that there are at least two systems that do not share a common service interface for the same service definition.

Once a valid communications path has been found, the system composer checks for a valid authorization route between the participant systems using those service interfaces. Recalling the NGAC mapping utilized in Section IV, each service interface is mapped to an NGAC operation, and each system is mapped to a user and/or object depending on if it will consumer or provide the operations. The authorization path can be built using the query in Fig. 18. The providing system, consuming system, and operation nodes are inputs to this query.

This query must be executed for each system pairing within the SoS. If the path is successfully returned, then a valid authorization can exist between the participant systems.

The system composer now is able to create and validate the functional SoS, check for communications path, and valid authorization permissions.

```
MATCH
        (sysProv) − [: Represented_By]− > (obj : Object),
        (sysCons) − [: Has_Alias]− > (usr : User),
        p = (usr) − [: ∗]− > (opr) − [: ∗]− > (obj)
RETURNp
```

Fig. 18. Build and check for authorization path for a given set of communications paths.

A. Dynamic Systems and Their Composition

The system composer is able to compose systems to create new behaviors. So far, all systems have been static, meaning that they have fixed sets of service interfaces (functionality). Only the combination of the usage of these systems has been dynamic. Dynamic systems are defined as systems that are able to provision and consume new service interfaces at run time. On a continuum, a system on one side can be highly static or up to fully dynamic—likely using machine learning. Rigidity is both functional and operational. Proposed here are systems that are able to provision new service interface instances of a fixed type but with independent configurations and life cycles.

Dynamic systems as introduced in [22], provide a provisioning service interface. Requests to this service result in a new function being created within the host system. The new function has service interfaces as required to match the specific requirements. For example, a "consumer" service interface for initiating "pull" based interaction or a "provider" service interface to "push" based interaction.

These dynamic systems are able to create local bridges, connecting disconnected graphs. However, to allow interoperability between dynamic systems from different developers, the interface for provisioning must be standardized. In [22], the dynamic systems were provisioned using SenML. Hypermedia as the engine of application state (HATEOAS) is a method of creating dynamic APIs for web-based applications. IETF Standards RFC 5988 [23] and 6690 [24] introduce web linking as a way to achieve HATEOAS applications. In addition IETF draft "draft-ietf-core-interfaces-10" [25] (core-interfaces) builds on RFC 6690 to propose a set of standard interfaces for interacting with URI resources. Here, we refine the provisioning interface of [22] to utilize the core-interfaces specification. The specification handles link format and SenML for parametric values.

The translation system proposed in [26] is an example of a dynamic system. The translation system is reliant on protocol information and link information to build the protocol translator. The dynamic provisioning request is shown in Fig. 19.

Once the protocol translator and its interfaces have been provisioned, the response is sent. It contains link information, linking to the newly constructed service interfaces. The translator's response is provided in Fig. 20.

Dynamic systems are able to bridge two service providers that require information exchange. Because service providers cannot pro-actively seek out service consumers, then two providers or two consumers cannot communicate and so must have a bridging. In this case rather than a translator a dynamic "proxy" can mediate the two systems. The dynamic systems

```
[
  {
    "name":"dynamic",
    "value":"transaltor"
  },
  {
    "name":"service",
    "value":"<coap://{ip:port}/serviceinst>;
    rt=[servicetype];
    if=[interfacetype];
    ct=[contenttype];
    rel=provider"
  },
  {
    "name":"service",
    "value":"<mqtt://{ip:port}>;
    rt=[servicetype];
    if=[interfacetype];
    ct=[contenttype];
    rel=consumer"
  }
]
```

Fig. 19. Run-time request to the translation system to provision a new protocol translator. With this information, the translator is able to provision two interfaces that will interoperate with the specified interfaces.

```
[
  {
    "name":"response",
    "value":"translator"
  },
  {
    "name":"service",
    "value":"<mqtt://{ip:port}/topic>";
    rt=[servicetype];
    if=[interfacetype];
    ct=[contenttype];
    rel=provider
  }
]
```

Fig. 20. Response from the translation system once a new protocol translator instance has been provisioned. Only the new provider interface is shown, this should be compatible with the specified consumer interface to be satisfied.

being HATEOAS-based systems themselves will return links to the dynamic resources from the base URI of the service. The IETF core-interfaces draft defines sensor, actuator, and collection interfaces. However, in the case of configuring or provisioning dynamic systems, more complex configuration parameters are required. Here, we propose to add a new interface that will enable transport of provisioning information for dynamic systems. The interface, which is based on the aforementioned example, includes SenML with Link-Format embedded as an element. Link-Format items can point to external definitions or service endpoints. The "rel" element is used to understand the meaning of the link.

A general definition and mapping of Link-Format attributes to describe services (core and application) in the AF is shown in Fig. 21. The items within the inner square brackets represent variable values, whiles the rel values must be kept as "provider" or "consumer." All attributes (rt, if, ct, and rel) must be present

```
<coap://{ip:port}/serviceinstance>;
   rt=[servicetype];
   if=[interfacetype];
   ct=[contenttype];
   rel=provider,
```

Fig. 21. Core link format used to describe service interfaces.

in the interface Here, the servicetype refers to service type definition held in the service inventory.

VI. Case Study

The use case demonstrating the proposed solution is a typical industrial manufacturing environment. Production optimization relies on flexible access to metrics and information regarding the underlying processes. This information extraction at the edge means that centralized infrastructure is not loaded with messaging traffic and computational processes. During the manufacturing of large industrial vehicles, there are safety critical nuts that must be tightened according to a quality specification. The nuts must be tightened to a specific torque value, and then, rotated through a fixed angle after that. The target values along with actual values are archived for legal purposes. In case of an accident involving the said vehicle, these data are reviewed to invalidate manufacturing as a cause. Therefore, manufacturing data tied to a vehicle chassis must be collected, monitored for quality, and archived for possible retrieval. In this use case, operators working within the wheel alignment work station notice that the quality assured nut tightening is reducing efficiency and placing ergonomic strain on the staff. An industrial engineer or designer on the shop floor needs to compose a new SoS to extract process information regarding the quality assured nut tightening. In consultation with production engineers, it is agreed to collect operational information of the nutrunner at the station.

In RAMI 4.0, a working area in the automation hierarchy contains all the layers of the software stack including the business layer down to asset layer. Cross-layer applications will be composed of services executing across the automation hierarchy and at different software layers. This wheel alignment work station is an Arrowhead local cloud that provides SOA and SoS support. The electronic nutrunner is used, it publishes time stamped notifications with its specification and actual data. These data are normally kept as archive for the quality assured tightening process and utilized only for regulatory requirements. However, in order to measure the operational efficiency of the nut tightening, the notifications from the nut runner can be captured, stored, and processed (i.e., as KPI information for the operators and engineers).

There are a number of ways that the nut runner KPI problem could be tackled.

1) A custom client application could be developed that subscribes to the nutrunner notifications and stores/processes the KPI results. The single purpose client application provides a set of fixed operations upon the data that would perform the required processing.
2) Alternatively, an SoS can be composed to capture the nutrunner data in a re-usable storage system. Then—on-demand—the data can be passed through re-usable filter systems to perform the required processing.

   Option one requires an IT engineering expertise to produce a customized application. During a requirements gathers phase, the IT engineer must consult with industrial engineers and designers to understand the needs. Then, implement the application according to the requirements. It is possible to utilize re-usable design methods to optimize the development time. However, if there are any change to processing requirements, this will likely require the IT engineers expertise to re-implement or make the change. This means increased lead times and costs for implementing the information extraction. A custom single purpose application need not rely on surrounding support infrastructure. It would subscribe to a configured nut runner and perform the required processing.

```
MATCH
      (consumer:System)-[:Requires]->
      (historian:Service(name:"HistorianService")),
      (historian)-[:Offered_By]->(a:System),
      (a)-[:Requires]->
      (filter:Service(name:"FilteringService")),
      (filter)-[:Offered_By]->(b:System),
      (b)-[:Requires]->
      (jobNotice:Service)name:"JobNotificationService")),
      (jobNotice)-[:Offered_By]->
      (nutRunner:System(name:"identity")),
WHERE
      (filter.fields =
      (jobNotice.dev.id, jobNotice.ts, jobNotice.type))
```

Fig. 22. Functional specification for the information processing including capture, storage, and processing requirement.

Option two, on the other hand, may not require IT competency. In this proposed approach, domain engineers are able to interact with the factory floor using a software tool, and express their information (KPI) requirement. The IT competency is only required to do initial setup of the system and its supporting functionality (the Arrowhead local cloud), but the generation of KPI information is completely in the hands of the industrial engineers.

This section describes the implementation of option two using the proposed method. With the Arrowhead local cloud setup, the nut runner will already be sending its tightening results to the manufacturing execution system for storage of the quality assured nut tightening. Hence, the industrial engineer will use the system composer to specify the (KPI) information extraction requirement. Fig. 22 shows this functional information specification in an Open Cypher query.

Here, the services are specified by their service names and the nut runner is specified by its system identity. The intermediate systems do not require specification and the composer is able to find suitable replacements. Once the composition engine has

```
MATCH
        (historian)-[:Implemented_By]->(c:Interface),
        (consumer)-[:Supports]->(c),
        (c)-[:Provided_By]->(a),
MATCH
        (filter)-[:Implemented_By]->(d:Interface),
        (a)-[:Supports]->(d),
        (d)-[:Provided_By]->(b),
MATCH
        (jobNotice)-[:Implemented_By]->(e:Interface),
        (b)-[:Supports]->(e),
        (e)-[:Provided_By]->(nutRunner),
WHERE
        (d.fields = e.dev.id, e.ts, e.type),
RETURN
        consumer.CompositionHandle
```

Fig. 23. Communication specification for the information processing including capture, storage, and processing requirement. Here, the e.dev.id is the device ID, e.ts is the job time stamp, and e.type is the job type. The composition handle is returned from the consumer data. This is a generic handle so that the same information specification can be used for multiple consumers.

parsed the functional query and created a valid path, a communications query will be generated according to the query shown in Fig. 23. The query shown contains references to variables from the earlier functional specification, such as historian, consumer, and filter.

The new system composition is routing new data from the nut runner system through a time filter and storing the results in a historian system. The consumer system, likely a graphical user interface system, is able to get the latest up-to-date KPI from the historian at any time.

The system composer takes this specification and builds functional, communications, and authorization graphs. Once the different paths are validated, the validated communications path is turned into composition rules that can be used by the orchestration engine. Existing SoS compositions are left unchanged and the new SoS has not yet impacted any operations on the station. Once executed, it will run alongside normal SoS compositions that achieve the normal station objectives. If degradation of the station performance is noticed due to introduction of the new composition, it can be reverted by removing the consuming system from the composer (the Arrowhead orchestrator). Collecting data in this manner can occur over any period of time.

To continue our use case; after a suitable length of time, the captured KPI information can be used to benchmark and report the inefficient nut tightening process. The industrial engineers suggest process to move the operation using the nutrunner to the next station. This suggestion is based on the qualitative assessment of the industrial engineer. Existing tasks at the next station involve orienting the product in such a manner that more ergonomic access is available for the nutrunner operation. Hence, human operators are able to complete the task more quickly and efficiently. When the equipment is introduced to the downstream station, the information specification is also moved and so a new SoS composition is created on the same specification, allowing an identical data collection and processing and comparison of the change. These local scope changes to SoS compositions are done at the edge and so do not impact the centralized infrastructure.

The final step of the use case is for the industrial engineer to run the same analysis and compare the results to the original KPI. Hence, a qualitative judgement of the process improvement can be quickly backup with quantitative results. This level of flexibility would normally require considerable assistance from tier 3 software support. It would also require significant change control procedures as underlying operations must be left unaffected.

VII. Conclusion

This paper has presented a method to dynamically extract information from its source within an IIoT network. Industrial designers are able to describe their information requirement in a simple query. Without assistance from specialized IT resources, the industrial designer creates new SoS compositions from local factory floor services. It is the graph query that unambiguously describes the SoS composition. Thus, during normal operation, the industrial engineer can manage KPI or new diagnostic information collection. Furthermore, centralized data stores are avoided because graph queries access the information at its source.

Utilizing the SOA and SoS theory, it is possible to visualize the IIoT application as sets of graph elements, create a mathematical expression of an information query in the form of data sources, transformations, and sinks, and compose systems, communications paths, and dynamic graph local bridges. The proposed graph model captures the services, systems and their associated abstract types, and a small set of edges representing the communication paths and functional dependencies. With separation between functional and communications specifications, the graph enables multilevel reasoning on requirements. Where a graph is disconnected, a local bridge can be injected between collaborating systems, thereby, creating a connected graph. The local bridge could be a translator in the communications path, or a data manipulator (i.e., filter or trigger) in the functional path.

The graph model goes further to map the SoS and SOA architecture to the NGAC model. Thus, enabling the proposed system composer to evaluate access control policies to ensure that SoS will be attainable under current conditions. The system composer does not hold the NGAC policy information, but is allowed to query the policy information and retrieve the rules. The system composer retrieves the system context information, and thus, has the knowledge required to reason upon the likelihood of access being granted between systems.

The proposed solution is able to operate within an Arrowhead local cloud. The local cloud supports islanded mode with all required support infrastructure co-located within the working area. The information extraction is described in a human readable query language. It uses the Open Cypher query language with reuse of symbols to enable reasoning on SoS creation. The proposed method relies on the graph theory and is implementation technology independent. It can be used with message-based protocols such as MQTT or request/response protocols such as HTTP. The NGAC standard is supported by the solution for secure access control. Enabling a third level of reasoning on the graph, based on attribute distribution between systems. Finally, the solution uses run-time-based lookup and late binding. Defining types and attributes that can be resolved to instances of systems. This enables fault tolerance and sporadic changes to the working area without redefinition of the information queries. The SoS specification is updated based on the information extraction specification.

VIII. Future Work

The research presented has laid the foundation for a formal description of SOA-based SoS composition and information extraction within an industrial setting. There exist many ways that this study can be further refined and extended. It would be interesting to define a method and its associated software components that would enable validation of correct operation of the specified SoS composition. Continuous quality monitoring of this validity would ensure that the objectives of the SoS are achieved. Extending the Arrowhead quality of service manager with composition rules could be a possible solution.

The basis for secure access control within the graph composition is proposed but a full security analysis of the solution is not performed. It would be interesting to understand what the vulnerabilities of the solution are to accidental and purposeful misuse by humans and computers systems operating in the area of the Arrowhead local cloud.

References

[1] B. Scholten, *The Road to Integration : A Guide to Applying the ISA-95 Standard in Manufacturing*. Research Triangle Park, NC, USA: International Society of Automation, 2007.

[2] *Reference Architectural Model Industrie 4.0 (RAMI4.0)—An Introduction*, Apr. 2018. [Online]. Available: https://www.plattform-i40.de/I40/Redaktion/EN/Downloads/Publikation/ram i40-an-introduction.pdf

[3] *Structure of the Administration Shell*, Apr. 2018. [Online]. Available: https://www.plattform-i40.de/I40/Redaktion/EN/Downloads/Publikation/structure-of-the-administration-shell.pdf

[4] A. Cockburn and J. Highsmith, "Agile software development, the people factor," *Computer*, vol. 34, no. 11, pp. 131–133, Nov. 2001.

[5] *Real-Time Twitter Trending Hashtags and Topics*, 2018. [Online]. Available: https://www.trendsmap.com/

[6] *ThingWorx Industrial Innovation Platform—PTC*. PTC. [Online]. Available: https://www.ptc.com/en/products/iot/thingworx-platform/

[7] M. Blackstock and R. Lea, "IoT mashups with the WoTKit," in *Proc. IEEE 3rd Int. Conf. Internet Things*, Oct. 2012, pp. 159–166.

[8] *Node-Red*, 2018. [Online]. Available: https://nodered.org/

[9] *IFTTT Helps Your Apps and Devices Work Together*, 2018. [Online]. Available: https://ifttt.com/

[10] J. Delsing, Ed., *Arrowhead Framework: IoT Automation, Devices, and Maintenance*. Boca Raton, FL, USA: CRC Press, Dec. 2016. [Online]. Available: http://amazon.com/o/ASIN/1498756751/

[11] F. Blomstedt *et al.*, "The arrowhead approach for SOA application development and documentation," in *Proc. 40th Annu. Conf. IEEE Ind. Electron. Soc.*, Oct. 2014, pp. 2631–2637.

[12] A. S. Tanenbaum and M. van Steen, *Distributed Systems: Principles and Paradigms*, 2nd ed. Upper Saddle River, NJ, USA: Prentice-Hall, 2006.

[13] S. R. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong, "TinyDB: An acquisitional query processing system for sensor networks," *ACM Trans. Database Syst.*, vol. 30, no. 1, pp. 122–173, Mar. 2005. [Online]. Available: http://doi.acm.org/10.1145/1061318.1061322

[14] M. A. Yaqub, S. H. Ahmed, S. H. Bouk, and D. Kim, *Information-Centric Networks (ICN)*. Singapore: Springer, 2016, pp. 19–33.

[15] G. Xylomenos *et al.*, "A survey of information-centric networking research," *IEEE Commun. Surveys Tuts.*, vol. 16, no. 2, pp. 1024–1049, Second Quarter 2014.

[16] T. Braun, V. Hilt, M. Hofmann, I. Rimac, M. Steiner, and M. Varvello, "Service-centric networking," in *Proc. IEEE Int. Conf. Commun. Workshops*, Jun. 2011, pp. 1–6.

[17] M. Eslamichalandar, K. Barkaoui, and H. R. Motahari-Nezhad, "Service composition adaptation: An overview," in *Proc. 2nd Int. Workshop Adv. Inf. Syst. for Enterprises*, Constantine, Algeria, Nov. 2012, pp. 1–8.

[18] W. Tan, Y. Fan, M. Zhou, and M. Zhou, "A petri net-based method for compatibility analysis and composition of web services in business process execution language," *IEEE Trans. Autom. Sci. Eng.*, vol. 6, no. 1, pp. 94–106, Jan. 2009.

[19] Y. Taher, M. Parkin, M. P. Papazoglou, and W.-J. van den Heuvel, "Adaptation of web service interactions using complex event processing patterns," in *Service-Oriented Computing*, G. Kappel, Z. Maamar, and H. R. Motahari-Nezhad, Eds. Berlin, Germany: Springer, 2011, pp. 601–609.

[20] S. Bouveret, U. Endriss, and J. Lang, "Conditional importance networks: A graphical language for representing ordinal, monotonic preferences over sets of goods," in *Proc. Int. Joint Conf. Artif. Intell.*, 2009, pp. 67–72.

[21] *Information Technology—Next Generation Access Control—Functional Architecture (NGAC-FA)*, American National Standards Institute, Washington, DC, USA, supersedes INCITS 499-2013, INCITS 499-2018, Jan. 2018.

[22] H. Derhamy, J. Eliasson, J. Delsing, and J. van Deventer, "In-network processing for context-aware SOA-based manufacturing systems," in *Proc. 43rd Annu. Conf. IEEE Ind. Electron. Soc.*, Oct. 2017, pp. 3460–3465.

[23] M. Nottingham, "Web linking," RFC Editor, RFC 5988, Oct. 2010. [Online]. Available: http://www.rfc-editor.org/rfc/rfc5988.txt

[24] Z. Shelby, "Constrained restful environments (core) link format," RFC Editor, RFC 6690, Aug. 2012. [Online]. Available: http://www.rfc-editor.org/rfc/rfc6690.txt

[25] Z. Shelby, M. Vial, M. Koster, C. Groves, J. Zhu, and B. Silverajan, "Reusable interface definitions for constrained restful environments," Working Draft, IETF Secretariat, Internet-Draft draft-ietf-core-interfaces-11, Mar. 2018. [Online]. Available: http://www.ietf.org/internet-drafts/draft-ietf-core-interfaces-11.txt

[26] H. Derhamy, J. Eliasson, and J. Delsing, "IoT interoperability—On-demand and low latency transparent multi-protocol translator," *IEEE Int. Things J.*, vol. 4, no. 5, pp. 1754–1763, Oct. 2017.