# A Middleware Mediated Application Layer Protocol to Decouple Provider-Consumer Relationship in Web Services Orchestration and Its Application in Novel IoT Integration

Devanathan Venkatesan(✉), Othiyappan Pandithurai, and Sundaramoorthy Sridhar

Anna University Chennai, Chennai, India
d.venkatesan@aubit.edu.in

**Abstract.** At present URI based web service orchestration poses a fundamental limitation on the nature of resulting web service applications restricting them to client-server paradigm and the partnership determination as a static design time activity. This naïve simplicity limits the scope and capability of orchestration technology that shows up sorely in our inability orchestrate IoT devices using application level web service protocols/technology. This work suggests and demonstrates a middleware (named here as Open Interaction Middleware Services-OIMS) based web service orchestration approach that unfetter the client-server nature of orchestration and introduces run time establishment of provider-consumer relationship. OIMS mediated orchestration permits speech-act based specification and orchestration of partners of the collaboration. OIMS functionality deserves to be a part of the WS infrastructure eventually. Decoupled service orchestration permit creation of novel application layer level web services based protocols and applications – that has the merits of both bus and broker based protocols combined - such as in the case of Internet of Things (IoT) monitoring & integration. This work illustrates the proposed approach using a case study in integrating IoT devices belonging to multi class IP network that provides several important quality attributes such as loose coupling and scalability to the resulting environment.

**Keywords:** Speech acts · Middleware based web service orchestration
Application integration · Loose coupling · Decoupled orchestration
Provider-consumer · Case study · WS infrastructure

## 1 Introduction

Explicit client-server nature of web service composition standards warrant determination and specification of WS provider address at the design time, resulting in applications that does not possess several important quality attributes such as loose coupling and separation of concern in a distributed computing environment (Venkatesan and Sridhar 2016, 2017). This deficiency shows up sorely in situations such as integration of IoT devices using web services standards where consumer cannot


D. Georgakopoulos and L.-J. Zhang (Eds.): ICIOT 2018, LNCS 10972, pp. 165–176, 2018.
https://doi.org/10.1007/978-3-319-94370-1_12

statically specify the provider URI addresses at the design time. This deficiency has forced IoT technology to groom kludges and low level ad-hoc protocol techniques to accomplish integration (Al-Fuqaha et al. 2015). This work presents a middleware based orchestration technology (that breaks off client-server nature of orchestration) and expand on the idea presented in Venkatesan and Sridhar (2016, 2017). A case based (Yin 2003) illustration is a good way to present and evaluate the outcome of the novel ideas and so this work adopts the case of IoT device integration. It should be noted the proposed protocol and middleware should be capable of handling integration with smart objects/small devices (that are power/resource constrained) that may have difficulties in dealing with protocols and SOA technologies designed for full-powered computers. Results obtained in this case may be extended subsequently to a more general situation and WS standards may be suitably amended.

This work is organized into 6 sections. Section 1 introduces the problem and outlines an approach and context of the solution. Section 2 describes over all architectural layout for message based orchestration (IEEE-FIPA 2018) using intelligent middleware namely OIMS. It describes the consumer application namely Intelligent Open Interaction Application Framework (IOIAF), messaging format used by IOIAF, its manner of interaction with OIMS and algorithm used by OIMS to determine and invoke WS providers. Section 3 describes the application layer level interaction protocol patterns, and implementation details of a working prototype (GitHub hosted) that uses Microsoft Windows Communication Foundation (WCF) technology (Sharp 2017). Section 4 analyses the results and makes a comparison with other popular IoT integration protocols. Section 5 provides a review of related literature. Section 6 provides a conclusion and proposal for further work. The suggested middleware technology is inspired by the works of Okouya et al. (2013).

## 2 The Open Interaction System Middleware

Venkatesan and Sridhar (2016) and Chap. 8 of Venkatesan (2018) propose an application layer protocol for integration of IoT devices using speech act messages and WS technology (as depicted in Fig. 1). This application involve a centralized web services consumer namely Intelligent Open Interaction Application Framework (IOIAF) based controller/dashboard, an agent oriented middleware service namely OIMS that provides facility to receive and transmit (IEEE-FIPA standard like speech act messages that carry information to carry out WS orchestration on providers using SOAP messages) across an ecosystem of IoT compute nodes connected to it. OIMS services act as intermediaries in relaying the messages to appropriate WS provider destination. The deployment level block diagram of this environment is depicted in Fig. 1.

A schematic of the proposed integration environment is provided in the Fig. 2. This environment utilizes Microsoft Windows WCF ver. 4.5 technology at OIMS to realize discovery of partnerlink services, enumerate them, enumerate services interfaces/input and output parameters and bind with them to invoke. Interaction between IOIAF service and OIMSs take place using Microsoft Message Queue (MSMQ) technology where the messages are of FIPA-ACL kind as described in subsequent section. For light-weight cases where performance and simplicity is the criteria, the interaction between IOIAF and

OIMS is alternatively realized designed using WCF itself as it is the case between OIMS and IoT leaf nodes. A IOIAF server shall connect with one or more OIMS middleware servers using user supplied configuration information. OIMS in turn connect with other OIMS or IoT leaf nodes to execute the contract as indicated by IOIAF application.

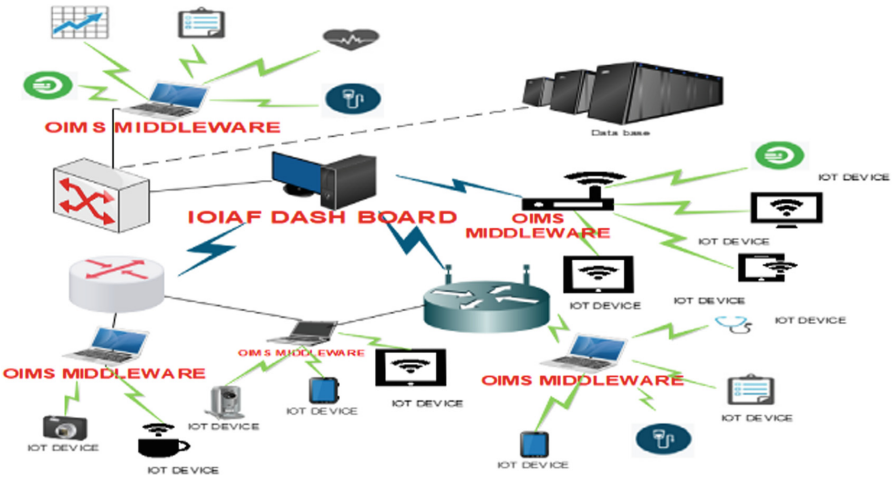


**Fig. 1.** Block diagram of IoT integration environment

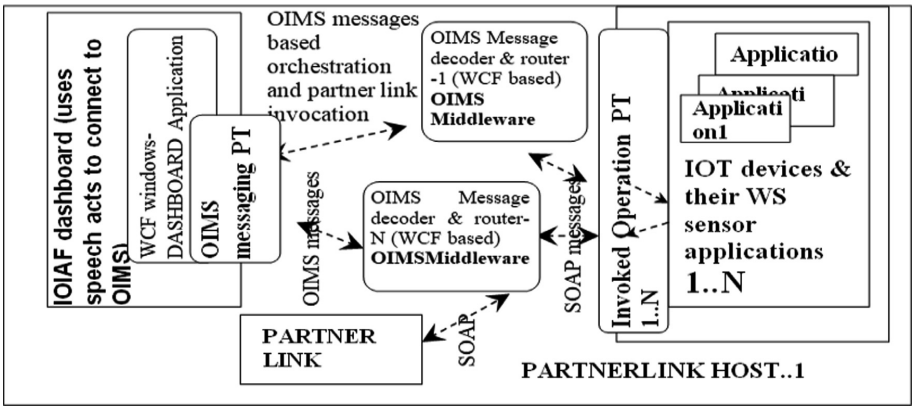


**Fig. 2.** Schematic of IOIAF orchestrator, OIMS middleware and IOT device (WS-Partner Link) leaf nodes

The operational algorithm of IOIAF service is described in the Fig. 3. The designer and programmer decide on the syntax and semantics of speech act messages based on nature of requirement demanded by the end user or business case. IOIAF service sends this message to various connected OIMS middleware service nodes. OIMS middleware service receives the IOIAF messages and decodes them to understand the nature of partner link invocation to be performed by it (for example asking for a specific set of

sensor data from IoT node service). Based on the SA it communicates with various IoT nodes connected to it and consolidate the result and reply back to IOIAF server. The SA based interaction models (requirement scenarios) also provides hints that can be translated into system level functional test cases.

```
Algorithm IOIAF_Node_IoT_Data_Gatherer (input LIST oimsMWNodes)
loop ForEvery("HH:MM:SS")
{    iotData = NULL;
     LIST oimsNd = RefreshListOfConnectedOims (oimsMWNodes);
     SA qry= Generate IoTNodeQuery(input UsrIP);
     for each [oimsNd]
     {  MQ mq_channel;
        mq_channel  = Open Message Queue (Destination oimsNd)
        Send_SA_Msg_to_oimsNd (qry , oimsNd, mq_channel , qry);
        iotData = { iotData } U  receiveIoTDataFromOIMS_Middleware_Nd
        (oimsNd);
     } Report to consumer { iotData}
}
```

**Fig. 3.** IOIAF application (WS-Consumer) node "Processing" logic

The algorithmic description of OIMS capability is depicted in Fig. 4. Typically the IoT integration specific OIMS network shall have one or more controller or dashboard, each subscribing to an unspecified number of leaf IoT nodes and collect/control information and process (in accordance with the type of collaboration as listed in Table 1).

```
Algorithm OIMS_Middleware_Process (INPUT  IOIAF_iotSA)

    LIST NTWRK_Interfaces =Enumerate_NTWRKInterfaces_In_OIMSNode( );
    LIST OimsMsgQueue= RefreshOimsMsg_Queue();

    for each "NwIntrfc" item in [NW_Interfaces]
        LIST OimsSrvrs= Enumerate_attached_OIMS();

    Relay_Selectively_Msgs_to_Other_OIMS_And_FlushQueue (OimsMsgQueue );
    DATA_RECORDS iotData =NULL, ndData=NULL;
    for each "NwIntrfc" item in [NW_Interfaces]
    {
        LIST iotNd  = Enumerate ConnectedIoTNodes (NwIntrfc);
        //iotNode consists of IP addresses in the same Local Area Network
        for each [iotNd ]
        {
            soapMsg = Decode_SA_specific_to_node (IOIAF_iotSA);
          if (  needToInteractWithIoTNode (iotNd )==TRUE )
            {
            ndData = invoke_IoTNd_PL (iotNd, soapMsg);
            iotData = iotData U ndData;
            }
        }// all IoT nodes of a particular Network Interface contacted
    }// all IoT nodes of a particular Network Interface contacted
```

**Fig. 4.** Processing control loop in OIMS

While the centralized dashboard (such as closed loop video surveillance network) shall use "ask" and "tell" speech acts through OIMS and other network gateways on the IP network, a local dashboard application of an operation theater instruments shall use subscription to selective IoT devices without intervention of gateways or centralized servers but with the help of OIMS alone).

The OIMS makes use of WS-Addressing technology (for synchronous real-time data processing/Message queues (for asynchronous data processing) along with WS-Eventing technology to realize much of its infrastructure based loosely coupled integration capability. It may be noted OIMS and IOIAF support scalability like *per-instance, per-session or per-call semantics* based service provisioning to support state based, configuration based, service rendering in the case of multiple IOIAF dashboards on the same shared physical network infrastructure – say a server farm - as in the case of, for example a hospital network that has separate dashboards (IOIAF applications)

**Table 1.** Patterns of interaction between IOIAF and OIMS

| **Messaging Pattern Type /(Abbreviation)** | **Sample SA Message format (as formatted by IOIAF app)** |
|---|---|
| Point-to-Point (PTP) | {<br>sa_type : "ask"; sender_id : "192.168.1.2"; receiver_id : "192.168.2.12"<br>command_query: {<br>NODE-ID=ROOM203-DEV1 &&<br>TYPE=SENSOR-DATA &&<br>SENSOR-ID =SENSID001 &&<br>REPEAT-TIMES=100 &&<br>REPEAT-FREQ=:01:00:00:00 //format DD:HH:MM:SS:MSEC }<br>}<br>//This message is interpreted by OIMS; This message means to gather data from a specific IoT node with Id. ROOM203-DEV1, for a specific sensor Id. SENSID001, for every I second for next 100 times. OIMS will keep the state info. And query the IoT node for data next 100 times every second and report the datum to IOIAF dash board. |
| Point-to-Everybody (PTE) | {sa_type : "ask"; sender_id : "192.168.1.2"; receiver_id : "*";<br>command_query: {SENSOR-DATA }<br>}<br>//This message is interpreted by OIMS; it means to gather data from all connected IoT node covering all connected sensors. |
| Broadcast (BBB) | {<br>sa_type : "tell";sender_id : "192.168.1.2"; receiver_id : "*"<br>command_query: {SLEEP }<br>}<br>//This message is interpreted by OIMS, meaning to inform intermediary OIMS to desist from contacting connected IoT nodes; they may cease to function until further instruction from Dashboard controller. This may be used to save IoT node battery and address lull in need to gather data. |
| Point-to-Multipoint (PTM) | {sa_type : "ask"<br>sender_id : "192.168.1.2"<br>receiver_id : "*"<br>command_query: {SENSOR-DATA }}<br>//This message is interpreted by OIMS, means to gather data from all connected IoT node covering all connected sensors. |

for video surveillance, visitor service, patient service, hospital housekeeping, patient monitoring etc.… that all still uses same OIMS middleware – say that are running on the network intermediary hardware – capable of supporting and sustaining multiple conceptual logical IoT networks using same WS executable using scalability (per-session invocation) logic.

The number of OIMS middleware attached to IOIAF server can be handled using different approaches. A straightforward approach is to provide a configuration file to IOIAF server that lists the IP addresses of the attached OIMS middleware servers. This work uses *ioiafconfig.ini* (Fig. 5a) to supply a list of OIMS nodes connected to IOIAF server. Alternatively the available list of OIMS nodes connected to an IOIAF server can be determined and discovered by a dynamic discovery protocol for OIMS middleware (like the case of DHCP or DNS server discovery). In turn, OIMS middleware server node is further configured using a *"oimsconfig.ini"* (Fig. 5b) file. This file provides a list of class of connected IoT networks its network-interface. OIMS search periodically over the entire subnet address space (wired or wireless) to discover available leaf nodes (i.e. IoTClient nodes), at the given moment, using node discovery protocols. This work uses a polling scheme for each IP address. Once the connected IoT leaf nodes are determined, OIMS will start interrogating the IoT leaf nodes for data as indicated by OIMS messages delivered to it by IOIAF server.

The IoT sensor nodes can also be configured to inform like the name of the sensors connected to it and the port details of the sensors (or hardware connection Id) (Fig. 5c). All these configuration information should use a shared ontology so that the received "commands" (from IOIAF application and OIMS messages) and reported "data values" are understood without ambiguity. This will help the entire IoT network work in a standardized manner and able to serve various custom queries. In cases where no standard compliance is available in IoI node, the IOIAF application need to specialize its messages taking into account specific terminology adopted by custom IoT device class. This research work uses a configuration file namely *Iotnodesvc.ini* (Fig. 5c) to supply configuration information of (*simulated)* IoT leaf node. This research assumes each IoT node runs a WS to permit data acquisition or else has at least provides a proprietary custom WS for data acquisition details of which can be supplied to the interfacing OIMS as customized node specific orchestration information for a specified unique node Id.

| | | |
|---|---|---|
| #File IOIAFconfig.ini…<br>#list of connected OIMS<br>[ICURoom208]<br>192.168.2.1=1<br>192.168.3.1=1<br>10.1.1.1=1<br>[SurveillanceCamera]<br>192.168.2.1=1<br>10.1.1.4=1<br>**(a)** | #File OIMSconfig.ini…<br>#1-use network; 0-dont<br>[CameraLAN]<br>10.1.60.*=1<br>10.1.70.*=1<br>[ICUR208]<br>192.168.2.*=1<br>[ICUR209]<br>192.168.3.*=1<br>**(b)** | # IoTNodeSvc.ini<br>#1-use sensor; 0-dont use<br>[AttachedSensors]<br>TempSnsr=1<br>HumiditySnsr=1<br>[IOTId]<br>iotNodeId=auIT01<br>**(c)** |

**Fig. 5.** (a) IOIAFconfig.ini (b) OIMSconfig.ini (C) IoTNodeSvc.ini

End users will invoke IOIAF application with a process initialization argument (i.e. command-line argument) specifying its application identification. Corresponding application identification must be there in the *ioiafconfig.ini* so that this particular instance of IOIAF service goes on to initialize OIMS service instance belonging to it completing the bootstrap of network of monitoring application chain (to accomplish a given business functionality). Based on the logic of application architecture, a single IOIAF application instance can serve multiple business functionality.

OIMS and IOIAF server instances are controlled as to how these instances are created and destroyed by setting ServiceBehavior's InstanceContextMode along with configuration details to be used at the service deployment/invocation time. The implementation details of IOIAF, OIMS and simulated IoT services are documented and hosted in GitHub for easy exploration and experimentation (SIVAN 2018).

## 3 The Business Logic Based Patterns of Integration in IOIAF Environment

Various kind of business use case served by this arrangement involves, IOIAF application (WS-consumer) periodically sending messages encoding its intention of gathering IoT data from WS-providers (IoT nodes) as Agent Communication Language (IEEE-FIPA 2018) like-message payload (as illustrated in Table 1). This message can have different level of sophistication in describing the URI of the intended providers starting from syntactic encoding (as implemented in this work) to OWL-S based semantic encoding (Sect. 3 in Venkatesan and Sridhar 2017). OIMS in turn determine choice of WS providers to connect to and the nature of the contract to execute (SIVAN 2018; Venkatesan and Sridhar 2016, 2017). Some examples of the syntactic messages originated by IOIAF server is depicted in Table 1.

The mechanism of the orchestration of IoT services described here gives rise to the possibility of realizing application layer level protocol that permit creation of complex integration applications. This capability is called as Message Based Service Integration (MBSI) protocol in this work. Section 4 provides a survey and comparison of capabilities of various IoT (application level) integration protocols including that of MBSI. Table 2 reveal the nature and technical implication of using these protocols for developing data gathering or monitoring end user applications.

A snapshot of the sample experiment carried out to gather data from an IoT network of nodes involving multiple OIMS is illustrated in Fig. 6 below.

The manner of OIMS based WS orchestration envisaged here is quite a different kind of business workflow compared to regular Process Aware Information Systems (PAIS)/business applications (Aalst 2009). Presently PAIS typically invoke providers using flow based technology (i.e. events, activities, gateways) and is entirely based on design time decisions. The proposal made here is not disruptive and only adds a new layer of features above existing standards and infrastructure. Hence it is a kind of incremental innovation or suggestion.

**Table 2.** Comparison of popular IoT protocols

| Protocols vs attributes | MQTT | AMQP | COAP | JMS | DDS | REST | XMPP | MBSI |
|---|---|---|---|---|---|---|---|---|
| Applicable standard body | OASIS std.- Version 3.1.1 | OASIS std. – version 1.0 | IETF std. version-1.2 | Java Enterprise edition | Open International data centric connectivity version1.4 | IETF version 1.1 | IETF version 1.0 | Proposed |
| Protocol architecture type | Broker based | Broker based | Broker based | Broker based | Bus based | Bus based | Bus based | Bus & Broker – Hybrid |
| Message delivery type | Web based | Message Oriented & middleware | Web based and middleware based | Middleware based | Middleware based | Application based | Application based | Middleware & message oriented |
| Integration strategies | Publish subscribe, client broker | Publish subscribe, point | Request reply | Publish–subscribe | Peer to peer | Request–reply | Request–reply | Decoupled request-reply based |
| Connection type | TCP/IP | TCP/IP | UDP | TCP/IP | TCP or UDP | TCP/IP | TCP/IP | ACL + SOAP |
| Interoperability | Possible | Not possible | Possible | Possible | Possible | Possible | Possible | Possible |
| Security | TLS/SSL | TLS/SSL | DTLS | TLS/SSL | DTLS | TLS/SSL | DTLS | WS-Security |
| Message delivery type | Client-server | Peer–peer | Client-server | Client-server | Peer–peer | Client-server | Client-server | Point-to-MultiPoint |
| Address mode | Unicast | Multicast | Unicast | Unicast | Multicast | Unicast | Unicast | Multi-level Multicast |
| Implementation complexity | Simple | Complex | Simple | Simple | Simple | Simple | Complex | Simple |
| Website for more information | mqtt.org | https://www.amqp.org | http://coap.technology | Oracle Java developer site | | W3C | https://www.xmpp.org | http://www.openthesis.org/document/view/603477_1.pdf |
| Foot print | Tiny | Tiny | Tiny | Tiny | Tiny | Tiny | Tiny | Large |
| Message encoding/representation technology (generic/binary/XML) | Binary encoding | Binary encoding | Binary encoding | Distributed message encoding | Binary encoding | Plain Text, also content encoding e.g. ZIP, compress, deflate etc.… | Plain text | Plain text format; XML schema compliant |
| Message envelop details (header size and max length) | 2 and 5 chars. | 4 and 20 chars. typically | 8 chars. | JMS does not support the header field transmitted to non-JMS client | Variable length | 8 field | 5 field | Consumer to OIMS-MQ header; OIMS to IoT WS provider (as per SOAP) |
| Session handling support | Persistent session | Session can be supported as session modules | Sessions have a window-based flow control model | Session is single thread context | Session Initiation protocol & Session Description Protocol | Server session to maintain a persistent state for a period and allow authenticated | Server depends on the instance and presence session | Provided by OIMS middleware; based on message options; |
| Flow control of messages | Publisher => central broker => subscriber | Client => server | Publisher-central broker => subscriber | Client => server | Publisher => subscriber | Client => server | Publisher => central broker => subscriber | No need explicit for flow control; OIMS/underlying layers ensure it |

(*continued*)

**Table 2.** (*continued*)

| Protocols vs attributes | MQTT | AMQP | COAP | JMS | DDS | REST | XMPP | MBSI |
|---|---|---|---|---|---|---|---|---|
| Meta-data support in messaging (content type indicator) | Comprises of identifier (data type & value) | Comprises of representation of resource | Routing key, persistent | Comprises of element (info and pointer) | The meta data can be converted into Scratch Image | Instant and persistent | Instance and presence | Full support available |
| Performance when connected to 1000s of devices | Poor | Good | Poor | Poor | Poor | Poor | Good | Excellent |
| Security and QOS | Both | Both | Both | Security | Security | Security | Both | Both |
| Seamless correlation support | Publisher | Client | Publisher | Client | client | client | Client | App. level and infrastructure based correlation possible |
| Message ordering support | Re-sends any PUBLISH packets, in order the original PUBLISH packets were sent | All messages arrive in sequence and takes care of retransmission and assembly in correct order | AMQP divides message into header, properties, body and optional footer | JMS defines that messages sent by a session to a destination must be received in the order they were sent | Messaging is 'federated' in nature. Traffic in federation is arbitrated by a unique network-scheduler | Uses asynchronous messaging based systems fits well with the pipe line message ordering or event based message approach | Message can be addressed to another entity. Sender server uses routing for ordering and delivering messages to client | Any kind of message ordering is possible through conversation/SA identifier of the message between provider and consumer |
| Tool support | MQTT-spy, MQTT-inspector | Core DX DDS-spy /DDS multiprocessor | Popular Linux variants and PyPi | Any Java EE IDE | Eclipse visual modelling tool, Tuner tool | RESTful API based design tools | Kaiwa, ejabbered | All service technology compliant tools applicable |

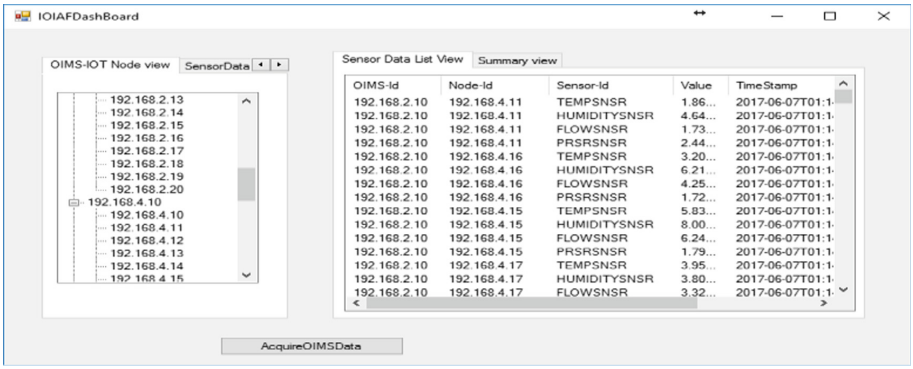


**Fig. 6.** A snapshot of IOIAF dashboard gathering sensor data from IoT network

## 4 Related Work

Higher-level business processes can be created – that are solution to business problems - by composing web services together. Service composition standards (Dustdar and Wolfgang 2005) provide standards-based, an open approach to connect web services with reduced complexity. WS standards (Weerawarana et al. 2005) reduces time and costs, and increase overall efficiency in formulating software for businesses problems. A well-established method of composing web services to create business applications is to use WS-BPEL (WS-BPEL 2007). BPEL is a model based, flow based, XML-based, imperative WS composition language which supports WS technology stack fully. However due to static nature of the orchestration provided in this language, it is not suited for IoT integration unless extensions are implemented to the BPEL engine/language (Venkatesan and Sridhar 2017). Okouya et al. (2013) propose middleware technology for integrating applications using speech-act like messages. This work draws inspiration and hints from that work to formulate the novelty presented here. Blake and Gomaa (2005) provides an early account of agent based cross organizational workflow composition. However it lacks a rigorous specification and description of capabilities of agent oriented middleware. It also did not provide an operational description how to realize the system making it a theoretical treatise.

The hardware limitations of IoT devices make design of software application unique and challenging. Jawad et al. (2017) provides a survey of various issues involved in integrating constrained devices in agricultural field monitoring including IoT integration issues. For example regular application level protocols like HTTP, SOAP may not work if one ignores network topology, packet size limitation and data rates and frequency of packet transaction. Hence software applications should explicitly address these constraints in designing IoT application such as minimize unwanted processing, reduce network data transaction.

Most of the existing application level protocols (given in Table 2) do not anticipate or take into account future changes in network topology, application architectures and IoT software stack evolution. Protocol innovations for IoT integration related can be

applied to any layers of the network such as physical layer, data link layer, network layer and application layer. *This work specifically addresses problems of integration at the application level protocol only*. New IoT specific protocols (Al-Fuqaha et al. 2015) tends to reduce data errors, avoid unwanted re-transmission, keep simple flow of data, avoid complex buffering/computation algorithms (saving on RAM/CPU cycles) to reorganize packets, increase reliability and wireless range. But one of the main issues here is difficulty of writing device integration, control and monitoring applications using these protocols. They do not offer satisfying experience for varying technical and practical reasons and offer little room for evolution and capability to coexist with newer developments. Presently bus based and broker based approaches are popular to network these devices. Some popular application level protocols include Message Queue Telemetry Transport (MQTT), Advanced Message Queuing Protocol (AMQP), Constrained Application Protocol (COAP), Java Message Service (JMS), Data distribution Service (DDS), Representational State Transfer (REST) and Extensible Message and Presence Protocol (XMPP). Table 2 provides a survey and comparison of capabilities of various IoT application level integration protocols including the Message Based Service Integration (MBSI) protocol proposed here. Venkatesan and Sridhar (2019) argues agent metaphor based system modeling and software development result in manageable and intuitive information systems. Venkatesan and Sridhar (2018a) argues agent metaphor based information system modeling anddevelopment result in superior model driven development software environment. Hence this work embraces agent oriented approach to realize the middleware solution.

## 5 Conclusion

This work presented an agent oriented approach to WS application composition and business logic enactment that fuses technological sophistication of WS technologies/standards and theoretical depth of agent oriented system modelling and engineering. Due to space limitation a comparative and experimental evaluation MBSI protocol with other IoT application level protocol is not carried out here. This work highlighted only a few elementary cases of patterns of interaction between IOIAF, OIMS and IoTNode services. This work did not carry out a comparative experimental evaluation of (possibility of) realization of patterns of interaction in other IoT protocols discussed here. These deficiencies need to be addressed in an elaborate study in the near future. From the conceptual viewpoint, the pattern of interaction described here in purely syntactical. This environment can be extended to permit varieties of application integration such as semantic web technology standard based partner/provider identification (using SPARQL, for example). The results arrived here can be utilized to formulate and extend orchestration standards by incorporating WS infrastructure support for the kind of orchestration approach envisaged here. These extensions remain to be carried out in the future.

## References

Aalst, W.M.P.: Process-aware information systems : design, enactment and analysis. In: Wah, B. W. (ed.) Wiley Encyclopedia of Computer Science and Engineering, pp. 2221–2233. Wiley, Chicester (2009). ISBN 978-0-471-38393-2

Al-Fuqaha, A., Guizani, M., Mohammadi, M., Aledhari, M., Ayyash, M.: Internet of Things: a survey on enabling technologies, protocols and applications. IEEE Commun. Surv. Tutor. (2015). https://doi.org/10.1109/COMST.2015.2444095

Blake, B.M., Gomaa, H.: Agent-oriented compositional approaches to services-based cross-organizational workflow. Decis. Support Syst. **40**(1), 31–50 (2005). https://doi.org/10.1016/j.dss.2004.04.003

Dustdar, S., Wolfgang, S.: A survey on web services composition. Int. J. Web Grid Serv. **1**(1), 1–30 (2005)

IEEE-FIPA: IEEE CS Approved Committee on Foundation for Intelligent Physical Agents (2018). https://www.computer.org/web/standards/fipa

Jawad, H.M., et al.: Energy-efficient wireless sensor networks for precision agriculture: a review. Sensors **17**, 1781 (2017). https://doi.org/10.3390/s17081781

Okouya, D., Fornara, N., Colombetti, M.: An infrastructure for the design and development of open interaction systems. In: Cossentino, M., El Fallah Seghrouchni, A., Winikoff, M. (eds.) EMAS 2013. LNCS (LNAI), vol. 8245, pp. 215–234. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-45343-4_12

Sharp, J.: Microsoft Windows Communication Foundation Step by Step. Microsoft Press (2017). ISBN 978-0735623361

SIVAN: IOIAF, OIMS, IoTSvc application code base at GitHub (2018). https://github.com/sivanagent/{IOIAFDashBoard, OIMS, IOTNodesvc}

Venkatesan, D.: A novel agent-based enterprise level system development technology, Ph.D. thesis, Anna University (2018). http://www.openthesis.org/document/view/603477_1.pdf

Venkatesan, D., Sridhar, S.: Promoting Business -IT Alignment through Agent Metaphor Based Software Technology, Int. J. of Inf. Technol. Manag. (2018a). https://doi.org/10.1504/IJITM.2018.10013469

Venkatesan, D., Sridhar, S.: A rationale for the choice of enterprise architecture method and software technology in a software driven enterprise, Int. J. of Bus. Inf. Syst. (2019). https://doi.org/10.1504/IJBIS.2019.10013326

Venkatesan, D., Sridhar, S.: A novel programming framework for architecting next generation enterprise scale information systems. Inf. Syst. E-Bus. Manag. **15**(2), 489–534 (2017). https://doi.org/10.1007/s10257-016-0330-y

Venkatesan, D., Sridhar, S.: A novel method and environment for scalable web service orchestration. In: Proceedings of IEEE 12th 2016 World Congress on Services Computing (SERVICES 2016), San Francisco, USA, pp. 128–129 (2016). https://doi.org/10.1109/SERVICES.2016.27

Weerawarana, S., Curbera, F., Leymann, F., Storey, T., Ferguson, D.F.: Web Services Platform Architecture: SOAP, WSDL, WS-Policy, WS-Addressing, WSBPEL, WS-Reliable Messaging and More. Prentice Hall PTR, Upper Saddle River (2005)

WS-BPEL: OASIS Web Services Business Process Execution Language v 2.0 (2007). http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html

Yin, R.K.: Case Study Research: Design and Methods, 3rd edn. Sage Publication, Thousand Oaks (2003). ISBN 0-7619-2553-8