Middleware for Internet of Things: A Quantitative Evaluation in Small Scale

Andrei Palade, Christian Cabrera, Gary White, M.A. Razzaque, Siobhán Clarke
Distributed Systems Group, SCSS, Trinity College Dublin, Dublin, Ireland
Email: {paladea,cabrerac,whiteg5,razzaqum,Siobhan.Clarke}@scss.tcd.ie

***Abstract*—Recently, there have been a large number of proposals for IoT middleware solutions. In addition, a few recent studies have surveyed and qualitatively evaluated these IoT middleware proposals against functional and non-functional features. A quantitative evaluation is also needed to complement these existing qualitative studies and provide a more in-depth perspective of the state of the art. This paper presents a quantitative evaluation of 4 representative proposals: OpenIoT, CHOReOS, LinkSmart and UBIWARE. The evaluation results, based on a small real-life scenario, show that research is needed in the area of autonomous and scalable service registration, discovery and composition, heterogeneity, and interoperability of IoT middlewares.**

I. Introduction

Discovering, connecting and coordinating services in IoT in order to build new applications is a delicate, time-consuming, and error-prone process, which requires a considerable amount of low-level programming and system administration effort [1]. Middleware technologies have been developed to reduce the complexity of dealing with some of these technical challenges. A middleware eases application development by integrating heterogeneous computing and communication devices, and supporting interoperability within the diverse applications and services running on these devices. In IoT, a service provides a well-defined interface, which offers all the necessary functionalities for interacting with the resources exposed by devices or infrastructures [2], [3]. A middleware should offer, among other things, functional components necessary for service discovery, service composition, data management, event management and code management.

The IoT makes middleware tasks even more challenging, as services offered by *Things* are often dynamic, mobile, less reliable and device-dependent. An IoT middleware must address non-functional requirements, including scalability, timeliness, reliability, availability, security, privacy and ease of deployment. Moreover, an IoT middleware should include architectural features to provide programming abstraction, interoperability, adaptability, context-awareness, autonomy and distributiveness [4]. Recent research into IoT middleware has resulted in a number of middleware solutions. A few recent studies have surveyed and qualitatively evaluated these IoT middleware solutions against functional and non-functional features [4]–[6]. This paper complements previous qualitative studies and presents a quantitative evaluation of 4 representative IoT middlewares: OpenIoT [7], CHOReOS [8], UBIWARE [9] and LinkSmart [10]. These representative middlewares are the top 4 IoT middlewares in terms of supported features, selected from a list of 60 middlewares [4] using an analytic hierarchy process (AHP) [11].

The evaluation provides a deeper perspective of the state of the art with respect to a subset of functional and non-functional features of IoT middleware. The IoT middleware features captured in the considered evaluation scenario (multi-modal journey) particularly address the innovative capabilities (autonomy, context-awareness, interoperability) that the IoT brings to software service development. The rest of the paper is organised as follows: Section II presents the IoT middleware requirements addressed in this evaluation, the scenario that captures these requirements, and the methodology used to select the 4 representative middlewares. Section III introduces the metrics used in the evaluation and the experimental architecture. Section IV presents the results of the quantitative evaluation. Open research challenges including possible future research directions are presented in Section V. Section VI outlines related work. Section VII concludes the work and points to areas of potential future work.

II. Study Scope

The evaluation focuses on quantifying selected requirements of the IoT which have been identified by the existing studies. This is achieved through the scenario implementation which captures the innovative capabilities of the IoT through a number of use cases. The methodology used to select the representative IoT middlewares is also included.

1) Middleware Requirements. A middleware must offer components for registering, discovering and composing the available services. As the IoT's infrastructure is dynamic, heterogeneous and large-scale, human intervention for service registration and discovery can become infeasible (e.g., M2M communication systems). An IoT middleware potentially exposes services from a large number of heterogeneous resources, which can make manual registration and service discovery impractical. In this context reliability, precision and recall in search time, as well as the response time can degrade. A semi-automated or automated decentralised approach has the potential to provide scalable service registration and discovery in a sufficiently timely manner for real-time service provision, and this is considered a key requirement in this study.

978-1-5386-2723-5/17/$31.00 2017 IEEE

Services provided by resource-constrained devices (e.g., the Telos motes in Figure 2) are likely to provide limited QoS, or to disappear abruptly. In many cases, another service may not be available that provides the required functionality. An IoT middleware should include a service composition component, where the available services can be composed or recomposed to provide the requested service. As the number of IoT resources increase a scalable, semi-automated or automated approach is needed to compose the optimal service. The middleware should proactively provide a timely response from the composed service even when IoT services become unavailable, or have an intermittent behaviour. An IoT middleware should dynamically adapt to contextual changes during the execution of the services, which may trigger further re-composition.

Installation, deployment and updating might make many middleware solutions unusable due to the significant time and skills required to set them up. A middleware must be easily deployable in a reasonable amount of time and should not require expert knowledge. In many cases, developer and community support are often critical in ensuring continuous improvement of the middleware and to address issues which are not included in the documentation.

2) Scenario Definition. The main goal of the scenario is to capture the uniqueness of the innovative capabilities that IoT brings to software service development. In particular, the scenario highlights autonomy, dynamism, adaptability, device heterogeneity, real time behaviour, interoperability and context awareness. The scenario presents a real-time and adaptive route planner application, where a user wants the optimal route from the Science Gallery to Dublin Spire.

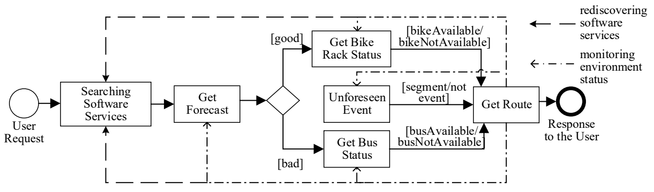
rediscovering software services
monitoring environment status
[good]
Get Bike Rack Status
[bikeAvailable/ bikeNotAvailable]
Searching Software Services
Get Forecast
Unforeseen Event
[segment/not event]
Get Route
User Request
Response to the User
[busAvailable/ busNotAvailable]
Get Bus Status
[bad]

Fig. 1. Basic workflow of the scenario in each middleware

There are two possible routes for the user in the scenario. In the first route the user walks towards the closest bike rack from The Science Gallery, gets a bike, cycles to the bike rack next to the spire, and walks to the Spire. If it rains, the system should proactively adapt to the changes in the physical environment, recalculate the route, and propose Route 2. In this route, the user walks to the closest bus stop and takes the bus to the stop next to the Spire and walks the rest of the trip.

Figure 1 represents the behaviour to be implemented in each middleware to support the defined scenario. The middleware searches for software services, which provide the necessary information to fulfill the user's request. If the weather conditions are permitting (defined as: temperature above 15°, humidity less than 80% and wind speed less than 4m/s), then, using *Bike Service*, check if there are any bikes available. Otherwise, using *Bus Service*, retrieve the bus status. Bike availability is checked at the bike stations next to The Science Gallery using *Bike Service*. If there are bikes available, the feature *Get Route* of *Routing Service* can be called with the parameter *bikeAvailable*, otherwise is called with parameter *bikeNotAvailable*. Bus availability at the stops next to The Science Gallery is checked using *Bus Service*. If there are buses available then *Get Route* is called with the parameter *busAvailable*, otherwise is called with the parameter *busNotAvailable*. For an unforeseen event, the route segment which triggered the event is passed as a parameter to *Get Route*. The output is the recommended route to the user (i.e., *Response to the User*). The route can be a multi modal route which combines two or more transport modes.

Five use cases capture the basic workflow and the dynamic requirements in the scenario: (i) *Simple Route Generation*: No relevant events in the environment or related to the services have occurred. The middleware should return a route to the user; (ii) *Event Happens*: An event happens in the environment (e.g., a bridge is closed). The middleware should return an alternative route to the user taking into account the event; (iii) *Service Not Available*: A software service is not available (e.g., bike service not available), and the information to make a decision in the basic workflow is not complete. The middleware should try to return the route with the available information. If this is not possible, the user should be informed; (iv) *Intermittent Service*: A software service is intermittent (e.g., the bike service is not available for a few seconds and then comes back on line). The middleware should manage the intermittent service and return the route to the user without any failure. (v) *Service Replacement*: A software service is not available but there is an alternative service that provides the same functionality (e.g., the bike service is not available but there is an alternative service). The middleware should manage these services and return the route to the user. Each use case simulates a possible event (e.g., environment event or software service event) with the goal of identifying and quantifying how each middleware responds to these events. Each use case can happen after the route request. The route planner should proactively adapt its response according to the events.

3) Methodology for Middleware Selection. Analytic hierarchy process (AHP) [11] framework was used to analyse and select the IoT middleware solutions for evaluation, using the IoT middleware requirements as selection criteria. We surveyed 60 middleware solutions [4], and made an initial selection of 10 of the most advanced approaches that consider the key IoT middleware requirements: OpenIoT, CHOReOS, UBIWARE, Linksmart, SOCRADES, Xively, Carriots, Prisma, Impala, MOSDEN [4]. Open-source middleware proposals had a higher preference. A score for each middleware was calculated by adding the scores of each requirements from their qualitative evaluation [4]. The features of interest were identified as follows: autonomy, heterogeneity, usability and response time for Service Registration, architecture (i.e., centralised or decentralised), autonomy, interoperability and response time for Service Discovery and Service Composition, online documentation, documented APIs, source code repository and ease-of deployment for the Developer Support component, and whether the proposal is non-proprietary, and continuously enhanced with a reasonably-sized community to offer technical support. CHOReOS, OpenIoT, UBIWARE and LinkSmart have scored higher than the other middlewares and also they have a low variation in their scores. These middlewares were selected for further evaluation.

III. Experimental Setup

This section describes the metrics used for the study, the infrastructure used during evaluation, and the detailed scenario implementation in each selected middleware.

1) Evaluation Metrics. We define metrics for service registration, discovery and composition separately, with some additional general perspectives as summarised in Table I. The scalability of service registration, service discovery and service composition is measure in a range from 5 to 1000 records. For interoperability, we count how many data formats can be managed by each middleware.

2) Common Experimental Architecture. A common experimental architecture for the 4 middlewares was implemented to support the services, workflow and use cases described in the scenario definition (see Figure 2). This architecture assumes a city-wide IoT environment, where there are sensors to get information about different variables in the city. In particular, for our mobility scenario we are interested in the sensors spread along the zone between the Science Gallery and the Dublin Spire and in the sensors inside the transport modes which move between those points.

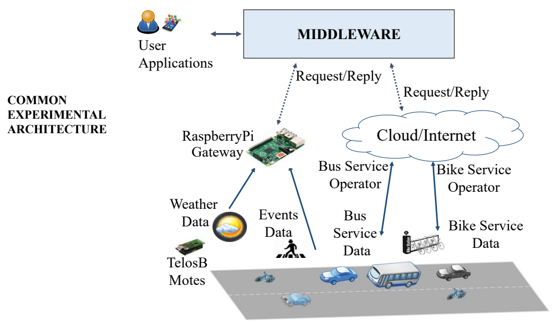

Fig. 2. Common Architecture Infrastructure

A set of sensors is responsible for collecting the weather data, which is exposed as *Weather Service* in the gateway. The gateway also hosts the *Unforeseen Events Service*, which simulates information about events in the streets of the city. Additionally, data is collected from the sensors in the transport modes: *Bus Service* and *Bike Service*, which are exposed as a service on the Internet by their operators (i.e., Bus Service Operator, Bike Service Operator). Each service exposes a REST interface. Finally, the middlewares are deployed on independent virtual machines and they make requests to the services on the Internet and the gateway in order to get the required information to provide the best route to the user.

IV. Results

This section presents the results of the evaluation, and are summarised in Table I.

TABLE I
Evaluation Results

SERVICE REGISTRATION				
Metric	**UBIWARE**	**LinkSmart**	**OpenIoT**	**CHOReOS**
Human intervention: (Autonomy) - Fully automated. - Semi automated. - Not automated.	Semi automated	Semi automated	Semi automated	Not automated
Number of resources types supported by the middleware (i.e., SOAP, REST, etc.) (Heterogeneity)	2 resources types - Handlers - Services	1 resource type - SOAP services	Unlimited	1 resource type - SOAP services
Programming effort,time in hours. (Usability Developer Perspective)	2 hours	1 hours	3 hours	2 hours
Response time in service registry (ms). (Performance)	See Figure 3	See Figure 3	See Figure 3	See Figure 3
SERVICE DISCOVERY				
Metric	**UBIWARE**	**LinkSmart**	**OpenIoT**	**CHOReOS**
Human intervention: - Fully automated. - Semi automated. - Not automated. (Autonomy)	Semi automated	Semi automated	Semi automated	Semi automated
Programming effort, time in hours. (Usability Developer Perspective)	4 hours	2 hours	0.05 hours	0.05 hours
Reliability in search results: - Checked. - Does not check. (Look Up & Matching)	Does not check	Does not check	Does not check	Does not check
Precision in search results (P): $P = \frac{truePos}{truePos + falsePos}$ (Search Accuracy)	0.67	0.67	1 (by altering SPARQL query)	0.67
Recall in search results (R): $R = \frac{truePos}{truePos + falseNeg}$ (Search Accuracy)	0.5	0.5	1 (by altering SPARQL query)	0.5
Response time in service discovery (ms). (Performance)	See Figure 4	See Figure 4	See Figure 4	See Figure 4
SERVICE COMPOSITION				
Metric	**UBIWARE**	**LinkSmart**	**OpenIoT**	**CHOReOS**
Human intervention: - Fully automated. - Semi automated. - Not automated. (Autonomy)	Not automated	Not automated	Not automated	Not automated
Programming effort, time in hours (Usability Developer Perspective).	120 hours	80 hours	10 hours	5 hours
Response time in simple service composition, (ms). (Performance)	See Figure 5	See Figure 5	See Figure 5	-
Response time with a event in the scenario, (ms). (Performance)	See Figure 6	See Figure 6	See Figure 6	-
Response time when service is not available, (ms). (Performance)	See Figure 7	See Figure 7	See Figure 7	-
Response time when a service is intermittent, (ms) (Performance)	See Figure 8	See Figure 8	See Figure 8	-
Response time when a service is not available but there is an alternative service, (ms). (Performance)	See Figure 9	See Figure 9	See Figure 9	-
GENERAL				
Metric	**UBIWARE**	**LinkSmart**	**OpenIoT**	**CHOReOS**
Ease-of deployment, time in hours. (Usability)	4 hours	40 hours	10 hours	50 hours
Number of formats allowed in the middleware. (Interoperability)	6 data formats	Unlimited	Unlimited	1 data format

Service Registration. The results for service registration show that all the middlewares provide a semi-automated service registration component. However, in each case, a manual process is required to finalise the registration. None of the recorded times includes developer learning time. In our evaluation, we recorded the time to perform this process: 2 hours for UBIWARE, 1 hour for LinkSmart, 3 hours for OpenIoT and 2 hours for CHOReOS. In UBIWARE, the recorded time includes the development the handlers for different agents and the business logic of the communication between the scenario agents and the UDF agent. In LinkSmart, we counted the implementation time of the functionality necessary to use the registration capability provided by the network manager. In OpenIoT, the implementation time of the required connectors

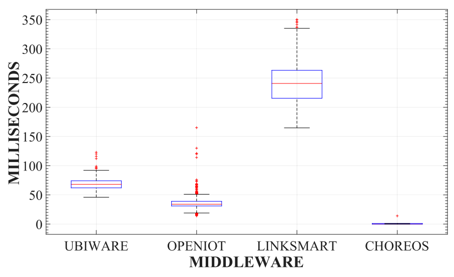

Fig. 3. Service Registration Response Time

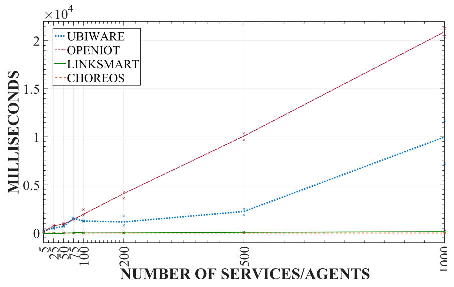

Fig. 4. Service Discovery Response Time

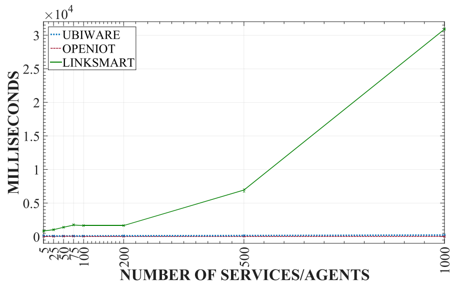

Fig. 5. Service Composition Response Time

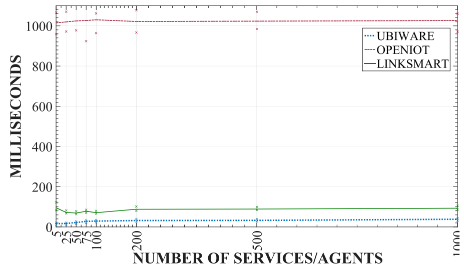

Fig. 6. Unforeseen Event Response Time

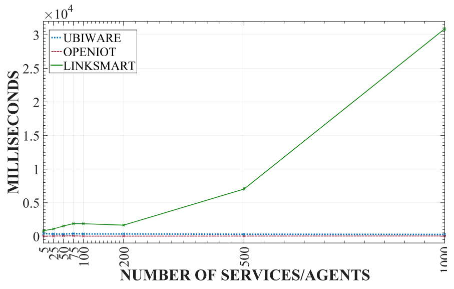

Fig. 7. No Alternative Service Response Time

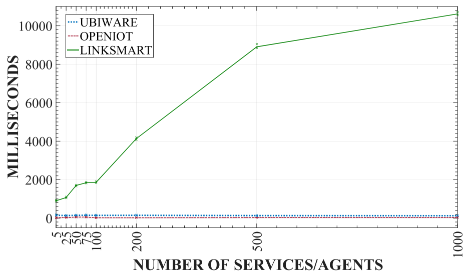

Fig. 8. Intermittent Service Response Time

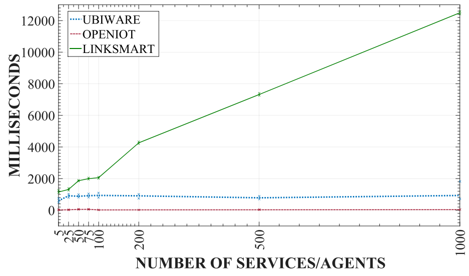

Fig. 9. Alternative Service Response Time

was counted. Also, in our case, it was necessary to implement a custom wrapper to handle the output of services. In CHOReOS, the implementation time of service wrappers for each service was counted. Figure 3 summarises the service registration performance. CHOReOS had the best performance with a median of 1 milliseconds in 1000 executions, followed by OpenIoT, UBIWARE and LinkSmart with medians of 34, 68 and 240 milliseconds respectively. CHOReOS performs well because it does not use persistent storage.

With reference to heterogeneity, CHOReOS and LinkSmart allow the registration of only 1 resource type (i.e., SOAP services), whereas UBIWARE supports 2 generic resources: services and handlers. In this case, each service is defined by a set of handlers (i.e., functions), and each handler is provided by one or more agents. In OpenIoT, an unlimited number of resources can be connected. This is possible because of the virtual sensor concept . A wrapper and a metadata file needs to be created for each resource connected to the middleware which increases the programming effort.

Service Discovery. All the middlewares provide a semi-autonomous discovery component and we programmed the functions to discover the services using the component provided by each middleware. None of the recorded times includes developer learning time. The usability (programming effort) of those components was 2 hours for LinkSmart and 4 hours for UBIWARE. In UBIWARE, this is the time needed to program the behaviours to send messages to the UDF agent, the subscriptions to different services and the rules that manage the output from the UDF agent for each discoverable service. In LinkSmart, this is the time needed to program a function to call the network manager and the instructions to manage the output from each service. OpenIoT and CHOReOS took approximately 3 minutes, which mainly involved accessing the necessary user interfaces.

For the look up and matching feature, none of the middlewares checks the reliability of the returned services. In UBIWARE, LinkSmart and CHOReOS the metrics of precision and recall are subject to the correct naming of the services, as well as exact matching between the search inputs and the services name. For this reason, the values of precision and recall in the service discovery process were the same for both (i.e., 0.67 of precision and 0.5 of recall) with our defined set of services. In OpenIoT, the location is used as a parameter in the search process, and the result varies according to the number of services available at a particular location. A better precision and recall can be obtained by adding the capabilities of the service to the SPARQL query. The response time shows (Figure 4) that LinkSmart and CHOReOS had a high and stable performance with different scale of simulated services (i.e., 5, 25, 50, 75, 100, 200, 500 and 1000 simulated services), meanwhile, UBIWARE and OpenIoT did not scale well, and the number of services affects their performance considerably. The drawback in UBIWARE is that there is just one UDF agent, which is responsible for managing all the requests and the big number of messages that the process generates.

Service Composition. None of the selected middlewares provides support for automated service composition. The implementation of the proposed scenario took 5 hours using CHOReOS, 10 hours using OpenIoT, 80 hours using LinkSmart and 120 hours using UBIWARE. In CHOReOS, the scenario can be implemented using any BPMN2 diagram editor. The implementation time is bounded to the programming effort of the diagram editor. In OpenIoT, the mashup tool provided is limited, and the user needs to complete the implementation of the scenario in SPARQL. In LinkSmart and UBIWARE the implementation took considerably more time than in the other two middlewares because there are no user support tools. In these middlewares, the scenario was written as a script, and executed by the middleware at runtime. We were unable to execute the composed service in CHOReOS. We got the following results for UBIWARE, LinkSmart and OpenIoT:

1) Simple Route Generation: OpenIoT had the lowest response time (Figure 5) in a simple execution of the composition, followed by UBIWARE. Both response times were stable with different numbers of simulated services/agents. LinkSmart started with an acceptable performance but after 200 simulated services, its response time increased considerably because the middleware network manager returns an increasing number of services after the service discovery process. The middleware network manager has to manage a higher number of services.

2) Event Happens: UBIWARE had the lowest response time as the agent which identifies the event sends a notification message to the composition agent autonomously. This use case is managed similarly by LinkSmart, but it uses SOAP messages for the notification, which slightly increases its response time compared to UBIWARE (Figure 6). OpenIoT had the highest response time as it stores the data into a database. In all cases, the response time was stable for different numbers of simulated services/agents.

3) Service Not Available: OpenIoT had the lowest response time (Figure 7) when a necessary service is not available, followed by UBIWARE. In both cases, the response times scale well with the numbers of simulated services/agents. LinkSmart had the highest response time, and it does not scale well with the number of simulated services. In this case, the composition process takes a longer time because there are more services returned.

4) Intermittent Service: OpenIoT had the lowest response time (Figure 8) when a service is intermittent, followed by UBIWARE. In both cases, the response times scale well with the numbers of simulated services/agents. The agent communication-schema in UBIWARE keeps the response time low regardless of the number of agents, whereas in LinkSmart the number of returned services in the service discovery affects the composition process, so the response time was the highest.

5) Service Replacement: OpenIoT had the lowest response time followed by UBIWARE when a service is not available but there is a replacement. In both cases, the response times scale well with the numbers of simulated services/agents. UBIWARE keeps the response time low because of its agent communication-schema. In LinkSmart, the number of returned services by the network manager affects the response time of the composed service by increasing the response time (Figure 9). OpenIoT performed better than UBIWARE and LinkSmart. However, the virtualised sensors approach used in OpenIoT is a concern for real-time data based applications. While the virtualised sensor approach decouples the infrastructure from the middleware, the data stored in the middleware's database can be out-of-date. This means that the data services keep executing, while providing potentially out-of-date data.

General. UBIWARE was easier to deploy than the other middlewares. While we got a functional instance of UBIWARE in 4 hours, the deployment time for OpenIoT, LinkSmart and CHOReOS was 10, 40 and 50 hours respectively. The interoperability results show that LinkSmart and OpenIoT do not have limits in the supported data formats, whereas CHOReOS has a limited number of allowed data formats. In its current form, UBIWARE has a limited number of allowed data formats, but it can be extended to support new data formats.

V. Open Research Challenges

Autonomous Service Registration, Discovery and Composition: The selected middlewares offer semi-automated or non-automated components to address resource registration, discovery and composition. In IoT, human intervention should be minimized as it leads to errors and a non-scalable solution. In particular, the selected middlewares are unsuitable for systems with self-* properties (e.g., self-adaptive) including M2M communication systems, because of their semi-autonomous or non-autonomous service registration, discovery and composition. Along with the wider exploitation of context (e.g., QoS), integration and exploitation of intelligence (e.g., rich semantic service description), self-* properties of an IoT middleware system is a rich open research area for fully automated mechanisms for service registration, discovery and composition.

Scalable Service Registration, Discovery and Composition: The response times of service registration, discovery and composition of the middlewares are not ideal even in small scale scenarios. Scalability is a system-wide requirement, every component of a middleware, including resource registration, discovery and composition needs be scalable to achieve system-wide scalability. Probabilistic service registration and discovery is a potential solution for scalable service registration, discovery and composition in service oriented middlewares.

Heterogeneity: The selected middlewares support one or two types of resources, which is insufficient for IoT's heterogeneous resources. Support for application level heterogeneity in these middlewares is limited because of inflexible and non-autonomous service registration, discovery and composition. New approaches are required to manage resource and application level heterogeneity in ultra large-scale networks where those resources/applications can evolve or be created in a short period of time.

Interoperability: Syntactical interoperability is supported by the selected middlewares, but they lack support for semantic and network/resource interoperability. Heterogeneity of devices is a common feature of the IoT, and, since there are no ontology standards, semantic interoperability is challenging. In addition, a lack of support for mediation at data, network and process level introduces restrictions on network/resource level interoperability. An abstraction layer between the resources/network and the middleware is a potential solution to improve the interoperability at this level. Research on global, scalable, understanding of IoT services' resources and semantics is required.

VI. Related Work

A number of quantitative studies into IoT middleware addressed individual elements of an IoT middleware ecosystem such as specific components of the middleware [3], or different types of IoT middlewares (e.g., M2M [12], RFID [13]). Also, each of the presented middleware were quantitatively evaluated individually as follows: Nikitin et al. [14] evaluated the reusability of semantic components in the context of UBIWARE, and Kostelnik et al. [10] analysed the interoperability in the context of LinkSmart. Muhammad et al. [15] quantitatively evaluated the average processing time of with varying number of services and the scalability of X-GSN and LSM server of OpenIoT, but did not consider usability and performance of the specific business logic (i.e., scenario implementation). CHOReOS was quantitatively evaluated in 3 separate use cases. Scalability, interoperability, and heterogeneity of Service Registration, Service Discovery and Service Composition components were analysed [8]. The results for scalability in Registration and Discovery confirms our findings, whereas heterogeneity and interoperability analysis was limited to cross-cloud provider, and did not consider data level interoperability. These related studies also conclude that awareness and adaptability are still open challenges.

VII. Conclusion

This article presented a quantitative evaluation of 4 representative proposals: OpenIoT, CHOReOS, LinkSmart and UBIWARE to complement the existing qualitative studies of IoT middlewares. The evaluation is based on a small scale scenario that captures the innovative capabilities (e.g., autonomy, context-awareness, interoperability) that the IoT brings to software service development. The selected middlewares were evaluated at two different levels: (i) a component (service registration, discovery and composition) level against autonomous behaviour, heterogeneity (resource level), usability and response time metrics and (ii) a general level against deployment time and interoperability metrics. In summary, (i) none of the selected middlewares supports fully autonomous and scalable service registration, discovery and composition, (ii) only OpenIoT supports resource level heterogeneity, and (iii) none of them scales well in service discovery and service composition response time (e.g., LinkSmart and CHOReOS scale well in discovery not in composition, UBIWARE and OpenIoT scale well in composition not in discovery). UBIWARE is easier to deploy than the other middlewares. LinkSmart and OpenIoT offer full scale syntactical interoperability, whereas UBIWARE offers a limited scale syntactical interoperability.

There is a significant scope for future work in the area of autonomous and scalable service registration, discovery and composition, heterogeneity, and interoperability of IoT middlewares, and our future endeavours will focus on autonomous and scalable service registration, discovery and composition for service oriented middlewares.

Acknowledgment

This work was funded by the Science Foundation Ireland (SFI) under grant 13/IA/1885.

References

[1] B. Benatallah, R. M. Dijkman, M. Dumas, and Z. Maamar, "Service Composition: Concepts, Techniques, Tools and Trends," 2005.

[2] W. Wang, S. De, G. Cassar, and K. Moessner, "Knowledge representation in the Internet of Things: Semantic modelling and its applications," *Automatika–Journal for Control, Measurement, Electronics, Computing and Communications*, vol. 54, no. 4, 2013.

[3] S. De, B. Christophe, and K. Moessner, "Semantic enablers for dynamic digital-physical object associations in a federated node architecture for the internet of things," *Ad Hoc Networks*, 2014.

[4] M. Razzaque, M. Milojevic-Jevric, A. Palade, and S. Clarke, "Middleware for internet of things: A survey," *Internet of Things Journal, IEEE*, vol. 3, no. 1, Feb 2016.

[5] S. Li, L. Xu, and S. Zhao, "The internet of things: a survey," *Information Systems Frontiers*, vol. 17, no. 2, pp. 243–259, 2015.

[6] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash, "Internet of things: A survey on enabling technologies, protocols, and applications," *Communications Surveys Tutorials, IEEE*, vol. 17, no. 4, 2015.

[7] J. Kim and J.-W. Lee, "Openiot: An open service framework for the internet of things," in *Internet of Things (WF-IoT), 2014 IEEE World Forum on*. IEEE, 2014, pp. 89–93.

[8] A. Hamida, F. Kon, N. Lago, A. Zarras, D. Athanasopoulos, D. Pilios, P. Vassiliadis, N. Georgantas, V. Issarny, G. Mathioudakis, G. Bouloukakis, Y. Jarma, S. Hachem, and A. Pathak, "Integrated CHOReOS middleware - Enabling large-scale, QoS-aware adaptive choreographies (D3.3)."

[9] A. Katasonov and M. Cochez, "UBIWARE Platform - RAB overview," p. 42, 2012. [Online]. Available: http://www.cs.jyu.fi/ai/OntoGroup/ubidoc/RAB_overview.pdf

[10] P. Kostelnik, M. Sarnovsk, and K. Furdik, "The semantic middleware for networked embedded systems applied in the Internet of Things and Services domain," *Scalable Computing: Practice and Experience*, vol. 12, no. 3, 2011.

[11] T. L. Saaty, "Decision making with the analytic hierarchy process," *International journal of services sciences*, 2008.

[12] F. Campos and J. Pereira, "An experimental evaluation of machine-to-machine coordination middleware," in *Proceedings of the 30th Annual ACM Symposium on Applied Computing*, 2015.

[13] G. O. Oh, D. Y. Kim, S. I. Kim, and S. Y. Rhew, "A quality evaluation technique of rfid middleware in ubiquitous computing," in *Hybrid Information Technology, 2006. ICHIT '06. International Conference on*, vol. 2, Nov 2006, pp. 730–735.

[14] S. Nikitin, A. Katasonov, and V. Terziyan, "Ontonuts: Reusable semantic components for multi-agent systems," in *Autonomic and Autonomous Systems, 2009. ICAS '09. Fifth International Conference on*, April 2009, pp. 200–207.

[15] M. I. Ali, N. Ono, M. Kaysar, K. Griffin, and A. Mileo, "A semantic processing framework for iot-enabled communication systems," in *The Semantic Web - ISWC 2015*, ser. Lecture Notes in Computer Science. Springer International Publishing, 2015, vol. 9367, pp. 241–258.