Run-Time Reconfigurable Middleware in Device Network Architecture

Marko Milovanović, Ivan T. Popović, and Aleksandar Ž. Rakić, *Member, IEEE*

***Abstract* — In this paper we introduce framework for implementation of automatically reconfigurable middleware layer for the system represented as a network of devices. The middleware enables communication between passive heterogeneous components of such distributed system by introducing dedicated service agents. Service agents establish run-time configurable data paths and they enable quality of service based reconfiguration and redundancy in data transport. The case study is presented to demonstrate utilization of quality of service measurements for automatic middleware data path reconfiguration.**

***Keywords* — automatically reconfigurable middleware, device network protocol, quality of service.**

I. Introduction

In this paper we introduce approach to reconfigurable middleware framework implementation for Device Network Architecture (DNA). DNA represents idea of physical network devices integration under unified hierarchy based on service oriented architecture (SOA). Device functionalities are implemented in form of group of services available on network nodes.

Advances in embedded hardware, embedded computing and diverse network communication implementations resulted in a considerable shift in economy of energy consumption, cost of system production, device size reduction. They also introduced high volume of contextually valuable information. Traditional approaches of data acquisition, storage and management of data transfer, information processing and availability do not meet high demand of throughput, resource and processing capability imposed by developments. This creates opportunity to envision, define and implement new strategies taking into account emerging needs.

In such environment, functional middleware has clearly defined role of abstracting out underlying complexity introduced by details of device implementation, low level resource representation, communication or computational pitfalls and concerns, enabling cooperation between participants. It should encompass provisioning of functional requirements and non-functional ones, from resource management to security, availability and service quality.

Type of infrastructure which needs to be supported is represented by spatially distributed, heterogeneous and dynamically rearranging network of nodes. They are required to exchange information in timely manner, taking their context into account. This setting makes parameters of QoS particularly relevant. Detecting change in communication performance, planning and execution of reconfiguration assures system reliability and performance. In this work, we analyze the integration of data path reconfiguration strategy governed by values of selected QoS parameters into the DNA framework.

Reconfiguration of the system, particularly in run-time, is thoroughly studied topic [1-4]. Multiple strategies have been developed to address reconfiguration issues such as potential downtime of the system components during reconfiguration, response time increase, data throughput capacity decrease and management of reliability and fault tolerance. These strategies offer tradeoffs in regard to specific parameter values such that one can select one most adequate to particular use case.

Approach of run-time reconfiguration has been considered in models from component view point taking into account interconnection both statically and dynamically such as Service Component Architecture (SCA) [5] and Fractal [6]. OSGi [7] platform uses application management resources to accomplish this. Supported functionalities include component and resource creation, installation, update, removal, dependency management and other. This is the point at which middleware resides, abstracting out all communication and management intrinsic details from functional development. It is technologically neutral and loosely coupled. It also offers ability of do composition at multiple levels of abstraction and promotes service discoverability, which might be challenging features in a large distributed network of devices.

Proposed framework aims to address systems which encompass components that may be scarce with resources, but aim for high availability and interoperability [8]. Data transport is passive with regard to service endpoints comprising of sensors, actuators or other functional units which lowers demand for resources on their end. Communication is facilitated through active components dedicated to the resources. This yields strict control over network traffic volume to be used at the application level, offering flexible modes of data, code and configurations transport and exchange between devices. Beneficial aspect of this approach is that architecture of the underlying middleware does not restrict possible style of the concrete

Marko Milovanović is a PhD student at the Electronics Department, University of Belgrade - School of Electrical Engineering, Serbia (e-mail: m.milovanovic@gmail.com).

Ivan T. Popović and Aleksandar Ž. Rakić are with the University of Belgrade - School of Electrical Engineering, Serbia (e-mail: popovici@el.etf.rs, rakic@etf.rs).

implementation or mode of operation, which may have arbitrary concrete implementation such as event driven or context-based. Active communication component operation may be triggered by external events, distributed network of nodes may be observed as a state database which can be aggregated and filtered, or accommodate any other application specific data structure, functionality or type of interaction.

DNA defines facilities for concrete implementation inspired by SCA and implementations such as Spring framework [9]. It gives platform for management of distributed applications. It is agnostic to concrete implementation technology as well, representing software components as services. Hierarchical component composition is also inspired by model used in SCA. However, SCA lacks flexibility in terms of service management and configuration, especially in run-time. That model also fails to recognize service models which perform tasks other then functional domain focused ones, such as resource access control. Some of the issues were separately addressed [10, 11]. Proposed framework distinguishes between local and network resource access requests. It also supports access control on the end of service endpoints implementing write operation, following established good practices [12].

DNA addresses common middleware design challenges including resource, event, context and arbitrary data management, service and resource discovery, availability and reliability, fault tolerance and incident recovery, scalability, run-time reconfigurability, security, abstraction from base components and interoperability. Their relevancy is outlined in large body of related work. Concepts of cloud computing [13], internet of things [14], fog computing [15] and related ones, all deal with research of the issues at hand [16, 17]. Here we layout building blocks apt for employment in those use cases.

Following section, II, introduces relevant background. In section III DNA is presented, followed by reconfigurable middleware in the section IV. Section V introduces QoS within proposed framework and in VI describes case study and an experiment alongside the results. Finally, VII gives the conclusion and suggests expected direction of further research and development.

II. Related work

Interconnected networks of entities rely on currently available technical and technological solutions and adopt newly proposed and implemented ones. Ongoing research provides some prominent results which help shape future steps and focus on emerging ideas. These include low level framework implementations as well as high hierarchical and architectural approaches.

Relevant top level design approaches include agent based, event based, service oriented, virtualized environment based and others. Implementations focus on domain specifics, programming abstractions, context utilization and try to combine them in order exhibit desired performance.

Agent based middleware is described as self-adaptive system which maintains its state or adapts new one based on the input and current context of the its environment [18]. Applications are divided into modular units which can be distributed through the network. Major benefits from approach include fault and failure tolerance, resource management in terms of network load and latency control, ability to execute encapsulated processes and follow protocols, good overall availability and reliability and capacity to adapt to possibly resource restricted network entities [19, 20].

In practice, implementations often fail to support heterogeneous infrastructures. As agents consider functional requirements as primary goals, aspects such as security and privacy are not well supported. Also, real-time policies are not strongly guarantied. Usual weakens of the agent based systems is pure code management capacity due to unreliable messaging and unpredictable autonomous behavior in run-time.

Event based approach considers system components to be participants of interactions [21]. Events are predefined and structured units of communications protocol. They are generated by event producers, routed through the network of addressable entities and delivered to consumers. Producers are related to the consumers by means of subscriptions [22]. Subscribers are granted access to the streams of events they signed up for. Such approach has potential for implementation of security policies, gives foundation for high reliability and availability. It also has capacity to deliver real-time performance, without imposing scale restrictions. Decoupling of interaction participants makes it ideal for environments with expected high rate of failures and dynamic reconfiguration of the network topology and volatile node presence.

Although this approach solves an amount of the problems distributed systems are faced with, it usually offers limited support for interoperability and does not adapt to arbitrary underlying components. Implementations following this approach often do not operate as autonomous systems nor they support context awareness.

Service oriented middleware approach define system components as services [23]. This is most common industry approach as it offers many benefits. It is not related to particular technology which makes it highly interoperable and reusable on multiple levels [24]. Loose coupling between system nodes helps deal with failures and ensures reliability in adverse conditions. By means of composition it gives tools to implement arbitrary functionality, process or paradigm [25].

Some deficiencies of service oriented approach include pure code management, programming abstractions and lack support for heterogeneous systems. Other aspects such as arbitrary scaling and autonomous operation are active field of study. One important deficiency of this approach is available support for security and privacy, which is usually limited to basic authentication, without support for fine-grained access rules [26].

Virtualized environments provide encapsulated execution context for application within the system [27, 28]. It abstracts away details of the underlying infrastructure offering high level programming interfaces without sacrificing transparency in large heterogeneous systems. They can be implemented to operate as a low-level system software or as a layer on top of operating system already present at the system [29]. Implementation is selected such that provides enhancements of characteristics of executing environment by providing virtualized resources such as

available processing units count, number of task execution threads or processes, virtualized memory and other, which can be shared across instances of tasks or system entities.
Practical implementation of virtualized environments usually suffers the most in case of resource constrained devices and hardware components. Final implementations usually need to account for diversity and use cases which impose generality in approach of the design and development. Another related issue is undermining performance for portability. This and other issues are under intensive research.
There are many other aspects of approaches to implementations such as support for novel networking protocols, semantics and ontologies, machine learning techniques and more, but they are out of the scope of this paper.

III. Device Network Architecture

DNA defines hierarchical model of integration for functionalities in network architecture, according to SOA concepts, but it borrows some of the concepts from other approaches listed previously in the paper such as agent based one. Emphasis is placed on reconfigurability of the components and their composition during run-time of the system. Particular functionalities are defined in form of resource collections at point of physical nodes. Individual resource is represented by independently configurable group of services. Local Resource Management (LRM) layer exercises access to resources and their respective configurations. This layer governs resources addressing and dissociates it from issues of communication at the physical layer. It acts as an interface between managed resources and the systems that exchange information with them.
Fig. 1 illustrates integration of group of devices at independent network nodes according to DNA concepts.

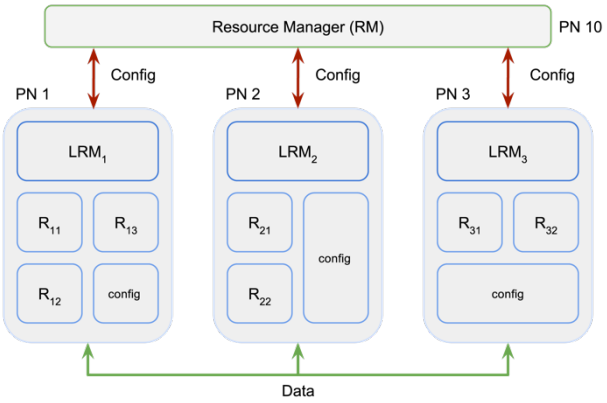

Fig. 1. Service architecture: Resource manager (RM) and Physical nodes (PN) interconnection using configuration and data paths.

Resource Manager performs system level integration of the resources and their interconnections via middleware configuration. Physical device consists of single local resource manager which acts as an interface to components, resources and their respective configurations. It manages all communication within the service, among components and resources, as well as outside service network requests. This gives support for composition of the entities comprising of physical nodes.

Three internal layers of architecture are defined by the role of components in system integration. LRM components support connectivity among devices and also define a model to address individual system resources. Particular functionalities of individual devices are encapsulated within passive layer of system resources. Service Agents (SA) collections define active layer. They are only active components of the system which can initiate data transfer. LRM components collection defines third connectivity layer. LRM communication is performed as defined by Device network protocol (DNP).

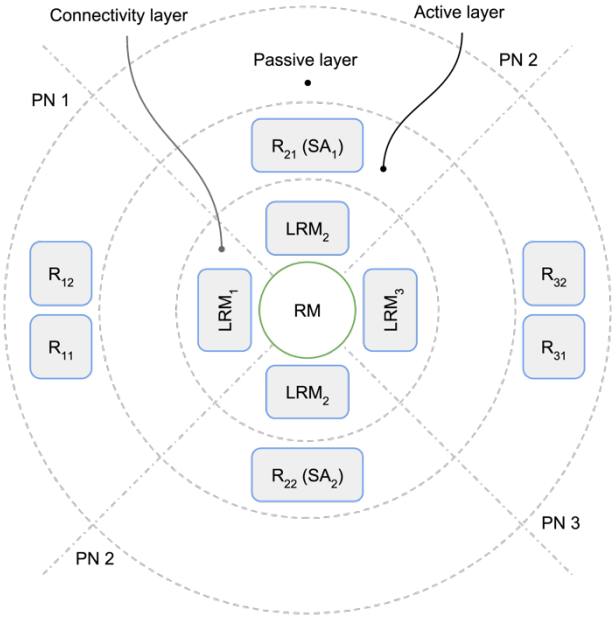

Fig. 2. Logical-layer view of DNA.

Fig. 2 shows logical organization of the layers of DNA. By abstracting out physical aspect of communication and system configuration, simplified model of resource integration and their respective interconnections according to DNA may be represented as in Fig. 3. There is a distinction between passive system resources (e.g. service groups), active system components (e.g. SAs) which embody middleware and data paths defined within active components configurations.

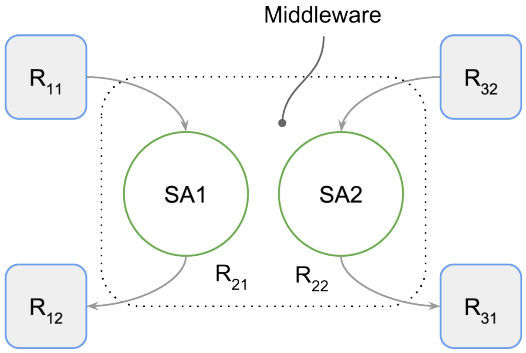

Fig. 3. Service instance hierarchy: Local manager governs access to available components and resources.

All communication is performed using DNP. It is stateless and does not employ handshake. It supports arbitrary packet sizes and has mandatory fields including designated status flags, which in turn resolve conditions for presence of optional fields.

All relevant and accessible aspects of a node, exposed through accessible endpoints, can be monitored and managed, assuring performance maintenance and QoS. Data and configuration exchange capability ensures management flexibility for resources and services.

IV. Reconfigurable middleware

Middleware is defined in form of a set of independent components which are configured to perform data transport over data paths using a pair of read and write service endpoints. Implementation of the concrete data exchange format and protocol is independent of underlying transport or application level protocol. Encapsulation is achieved using unified entities capable of communication, represented as SA. Service agents manage information data paths (IDP) and configuration data paths (CDP) used to exchange information and configure services, respectively. Messages are exchange and persisted in regard to respective service policies on participating nodes.

Middleware runs as an active part of the system architecture, among passive data providers and consumers. It consists of a service, which acts as a main architectural unit, which governs SAs, representing connections to accompanying resources. It may be a representation of a physical device or implementation of an arbitrary function identified by unique Service ID assigned to it on creation. Endpoints of an instance of a service must satisfy specification which consists of read and write endpoints for both data and configuration paths. Context of the individual node is preserved locally and can be managed using configuration path calls.

Once a service is instantiated it gives access to available components and resources configuration endpoints, if initialization is not required, otherwise it waits for initialization prior to enabling configuration access.

Any resource may be local, meaning that they have restricted access available only within the component, or they may be accessible through network requests. Access control is implemented by Local Resource Manager. It can allow or deny access to the resource if it comes from the network. At the service end, configuration of the IDP and CDP contains information about currently active path. Service allows single instance of data path pointing to a write endpoint, both information and configuration, to be active at any given moment. Granular access control based on arbitrary rules should be implemented at the application level and it is out of the scope of this paper.

Middleware handles IDPs and CDPs on all hierarchical levels of the system. Paths are defined via source and destination specifications. It may use local or network endpoints. Resource requests triggered by service calls activate SA engagement. Transport layer is encapsulated this way within configurable middleware entities. This yields resource implementation decoupling from data exchange protocol and enables context in which resources act as passive elements. Resource callbacks are reachable from LRM context.

Resources themselves define and implement message persistence policies, which can be accessed through resource managers. Policies may be implemented on an arbitrary level of the hierarchy and managed by service agents.

Described middleware supports flexible hierarchical organization of isolated system or arbitrary interconnected arrangements of systems. Ability to support arbitrary arrangements also helps enable distributed architectures. This allows for system scaling to accommodate application requirements, while components themselves keep small footprint in terms of individual network node resource requirements.

V. Quality of Service

QoS refers to wide variety of non-functional descriptors and requirements of the system [30]. This topic was researched extensively already and the research resulted in variety of implementations in related fields such as cloud computing [31]. It attempts to estimate utilization and manage available resources to the extent of quality parameters requirements fulfilment. Ultimately, reliability of the service is reflected in this point as this is the feedback loop which needs to unambiguously determine and communicate service status. Real-time applications are example use case where QoS is essential [32].

Initial step is to identify relevant parameters to be used in quality evaluation process. Those may include performance, reliability of the system, safety, fault tolerance, etc. When networking is considered under QoS, measures of request to response time delays, line noise, jitter, available bandwidth and packet loss are indicative of achievable quality of transmission and can be used to determine set of preferred or threshold values [33]. To achieve assumed values, system must adapt to the changes of operating and other conditions of the running environment. Proposed framework considers service response times and available data transport bandwidth as relevant QoS metrics.

Framework assumes time tracking support in form of timestamps embedded into DNP packets. It can be either universal or local system time, accessible through read and write service endpoints encapsulated into time service. Using this service, SA can calculate QoS related relevant time intervals. These include both read and write service endpoints calls, spanning from the moment request got dispatched until respective response is received and interval spanning from read request dispatch through response received from respective write request, for a configured data path.

QoS Service is implemented according to service specification, using read and write endpoints to form dedicated IDPs. Read and write events are managed by assigned SA nodes. Resource manager is configured to use single or multiple instances of QoS. Using assigned SA, it submits timing information from the data packet headers only. SA submits intervals data to QoS service using its service endpoints. For each pair of the service instances multiple SAs can be configured simultaneously in parallel, each defining single data path. All parallel paths perform reading operations on every read cycle and obtain data the for write part of the cycle, if any. Write part of the cycle is also performed simultaneously by all defined data paths and

target service commits values from active ones. Recorded timing and throughput values from all data paths, regardless of whether they are currently active one, are then sent to QoS for processing and ranking. This information is used to reconfigure active paths back at the monitored service.

QoS measurement is implemented at SA for each individual transport over particular IDP. Recorded QoS data encompasses read and write service calls request-to-response times, data transport time over IDP and volume of data transport. Data acquired by SA is sent to QoS. It then aggregates and processes this data and produces QoS parameter for particular IDP. Based on produced parameter, algorithm for IDP reconfiguration selects optimal path. This information may be used for IDP reconfiguration, both SA IDPs and service endpoint IDPs.

VI. MIDDLEWARE RECONFIGURATION CASE STUDY

Case study shows feasibility of reconfigurable middleware implementation within framework defined by DNA. Basic setup includes Resource provider, Resource consumer, Service Agents assigned to manage data exchange, respective configured data paths and Quality of Service endpoints used to manage defined IDPs.

Reconfiguration capability is tested using two SA nodes, with two IDPs each. IDPs are represented by pairs of *read* and *write* paths. Two functional services, Resource provider and Resource consumer, are deployed, as well as two SAs (SA1 and SA2). Two SAs define alternative paths which can be managed by QoS. Once the SA finishes data transport over particular data path, it sends measured QoS designated values to QoS service. Experiment components layout and connections setup diagram is shown on Fig. 4.

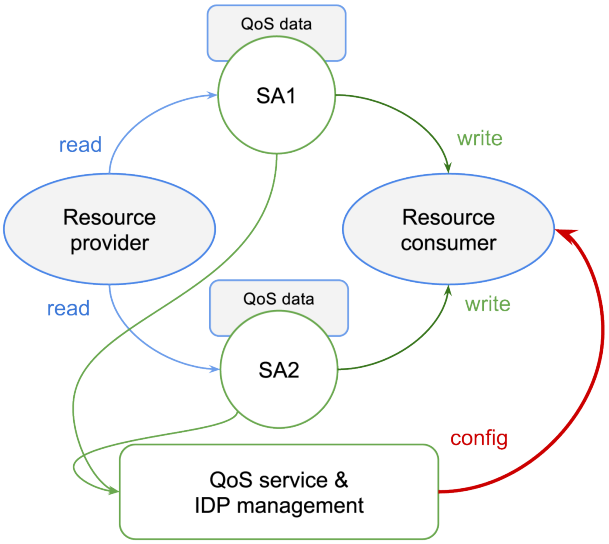

Fig. 4. Experiment setup.

QoS service contains IDP management which implements algorithm for data path selection. In this case, algorithm output selects active data path of the Resource consumer using its CDP over appropriate configuration interface. Active IDP is selected by means of configuration of access permissions at particular physical network node (PN). Measurements are submitted to QoS write endpoint, updating on each data transport.

QoS parameter at Fig. 5. is represented by normalized message transfer time (MTT) over particular IDP. Normalization iz performed by predefined maximum MTT. In effect, algorithm calcualtes mean value of this parameter in a given time frame defined by measurement intervals, for multiple parallel IDPs to determine which one is optimal. Once the active path degrades Resource consumer CDP endpoint is invoked in order to alter resource configuration. Initial data path setup asserts active path operated by SA2. This data path response time is degraded gradually over time. Alternative data path operated by SA1 is maintained at constant response time. Data throughput is maintained at constant value. Vertical line on graph designates time moment of the switch of active data path, which occures at 31 seconds from the start of measurement session. Prior to the switch, SA2 has active data path and after the switch SA1 path is selected.

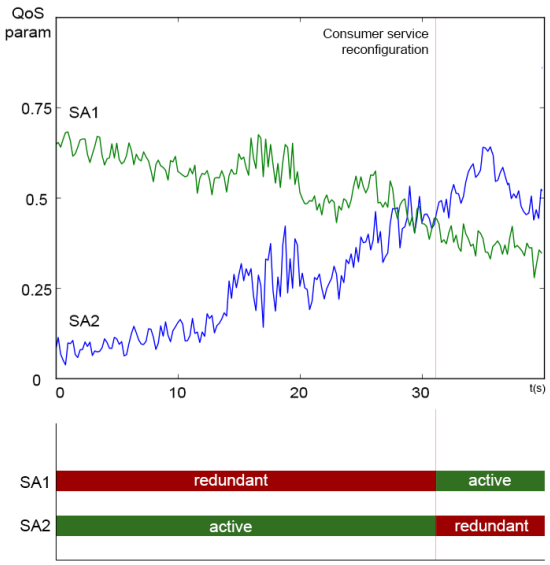

Fig. 5. QoS parameter based active IDP selection.

Implemented algoritam allows for memory of the previous states to be used in the decision making process. This results in a degree of tolerance to incidental unusually long response times for individual or small number of sequential requests, without performing the switch of active data path. If the response times of the active data path degrade over interval of time enough, better performing data path will be selected as active.

This test case demonstrates DNA support for definig redundant data paths. They are important for assuring QoS requirements. QoS parameter measurements are localized within middleware components as well. Subsequent processing of measured data is integrated into architecture as an independent resource and finally, resource and IDP reconfigurability is excercised using CDPs.

Provider and consumer components are accessed using unified protocol.

System shows flexibility of data path reconfiguration. This elevates both availability and reliability of the implemented solution. Locally, nodes store measurement values and messages can be recreated in case of transmission failure. Respective nodes which act as producers or consumers can be deployed in arbitrary large quantities. Same holds true for IDPs and CDPs which serve them.

Nodes can be distributed across arbitrary locations, as long as underlying networking protocol can uniquely identify entities participating in the interaction. Context is preserved by managing SAs which hold information about nodes configuration and interaction.

VII. Conclusion

Middleware and framework proposed in this paper successfully demonstrate capability of run time data path reconfiguration in DNA based network of devices. Service agents as unified middleware components encapsulate data transport functionality across defined DNA connectivity layer. This successfully separates mechanism and format of communication from concrete implementation of the device itself. Idea of middleware components reconfigurability offers possibility for redundant data paths, data transport optimization and a way to intervene with already established and setup systems in run-time by integrating devices and possibly new functionalities.

Focus of the future work will be placed to areas which include issues of reliability and robustness of the middleware implementation while using underlying network protocols for particular application on top of Internet Protocols, analysis of fault tolerance capability of the middleware as well as capacity to deliver code and configuration under the assumptions of QoS.

Appendix

Implementation of the framework used for experiment in this paper, with accompanying utilities, is available at: https://github.com/DNAIoT/prototype.

DNA components implementation targeted at embedded devices under active development is available at: https://github.com/DNAIoT/components.

Acknowledgments

This work was supported by the Serbian Ministry of Education, Science and Technological Development under contract no. TR-32043.

References

[1] W. Li, "Evaluating the impacts of dynamic reconfiguration on the QoS of running systems", *Journal of Systems and Software*, vol. 84, no. 12, 2011, pp. 2123-2138.

[2] M. G. Valls, P. B., Val, "Comparative analysis of two different middleware approaches for reconfiguration of distributed real-time systems", *Journal of Systems Architecture*, vol. 60, no. 2, 2014, pp. 221-233.

[3] J. Almeida, M. van Sinderen, et al., "Designing Interaction Systems for Distributed Applications", *IEEE Distributed Systems Online*, vol. 6, no. 3, 2005, pp. 1-1.

[4] A. Agirre, J. Parra, et al., "QoS management for dependable sensory environments", Multimedia Tools and Applications, 2015, pp. 1-23.

[5] M. Beisiegel, H. Blohm, et al., "Service component architecture: Building systems using a Service Oriented Architecture", [online], Whitepaper, 2005, pp. 1-31.

[6] E. Bruneton, T. Coupaye, et al., "The FRACTAL component model and its support in Java", *Softw: Pract. Exper.*, vol. 36, no. 11-12, 2006, pp. 1257-1284.

[7] "OSGi Service Platform Release 6.0", (2014), *OSGi Alliance*, [Online], https://osgi.org/download/osgi.enterprise-6.0.0-earlydraft2.pdf.

[8] D. Ingram, "Reconfigurable middleware for high availability sensor systems", In Proceedings of the Third ACM International Conference on Distributed Event-Based Systems, 2009, p. 20.

[9] R. Johnson et. al., (2016), "Spring Framework Reference Documentation", Whitepaper, [online], , http://docs.spring.io/spring/docs/current/spring-framework-reference/pdf/spring-framework-reference.pdf,

[10] J. Chen, R. Childress, et al., "A service management architecture component model", In Proceedings of the 7th International Conference on Network and Services Management, International Federation for Information Processing, 2011, pp. 316-319.

[11] Buyya, R., Abramson, D., at al., "Nimrod/G: An architecture for a resource management and scheduling system in a global computational grid", In High Performance Computing in the Asia-Pacific Region, Proceedings, The Fourth International Conference/Exhibitiom, Vol. 1, 2000, pp. 283-289, IEEE.

[12] H. A. Duran-Limon, G. S. Blair, et al., "Adaptive resource management in middleware: A survey", IEEE Distributed Systems [Online], 2004, http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=1323036

[13] P. Mell, T. Grance, "The NIST definition of cloud computing", National Institute of Standards and Technology, 2009, 53(6), p.50.

[14] K. Ashton, "That 'internet of things' thing". RFiD Journal, 22(7), 2009., pp.97-114.

[15] F. Bonomi, R. Milito, J. Zhu, S. Addepalli. "Fog computing and its role in the internet of things." In Proceedings of the first edition of the MCC workshop on Mobile cloud computing, ACM, 2012, pp. 13-16.

[16] M.A. Razzaque, M. Milojevic-Jevric, A. Palade, S. Clarke, "Middleware for internet of things: a survey", IEEE Internet of Things Journal, 3(1), 2016., pp.70-95.

[17] A. Munir, P. Kansakar, S.U. Khan, "IFCIoT: Integrated Fog Cloud IoT Architectural Paradigm for Future Internet of Things", arXiv preprint arXiv:1701.08474, 2017.

[18] J. Ceclio and P. Furtado, "Existing middleware solutions for wireless sensor networks," in Proc. Wireless Sens. Heterogen. Netw. Syst., 2014, pp. 39–59.

[19] F. Aiello, G. Fortino, and A. Guerrieri, "Using mobile agents as enabling technology for wireless sensor networks," in Proc. SENSORCOMM, Aug. 2008.

[20] T. Liu and M. Martonosi, "Impala: A middleware system for managing autonomic, parallel sensor systems," ACM SIGPLAN Notices, vol. 38, no. 10, pp. 107–118, 2003.

[21] R. Meier and V. Cahill, "Steam: Event-based middleware for wireless ad hoc networks," in Proc. 22nd Int. Conf. Distrib. Comput. Syst.Workshops, 2002, pp. 639–644.

[22] P. T. Eugster, P. A. Felber, R. Guerraoui, and A.-M. Kermarrec, "The many faces of publish/subscribe," ACM Comput. Surveys, vol. 35, no. 2, pp. 114–131, 2003.

[23] M. Papazoglou, "Service-oriented computing: Concepts, characteristics and directions," in Proc. 4th Int. Conf. Web Inf. Syst. Eng. (WISE'03), Dec. 2003 pp. 3–12.

[24] C. L. Fok, G. C. Roman, and C. Lu, "Servilla: A flexible service provisioning middleware for heterogeneous sensor networks," Sci. Comput. Programm., vol. 77, no. 6, pp. 663–684, 2012.

[25] N. Reijers, K.-J. Lin, Y.-C.Wang, C.-S. Shih, and J. Y. Hsu, "Design of an intelligent middleware for flexible sensor configuration in M2M systems," in Proc. SENSORNETS, 2013, pp. 41–46.

[26] S. Satyadevan, B. Kalarickal, and M. Jinesh, "Security, trust and implementation limitations of prominent IoT platforms," in Proc. Front. Intell. Comput. Theory Appl. (FICTA'14), 2015, vol. 328, pp. 85–95.

[27] N. Mohamed and J. Al-Jaroodi, "A survey on service-oriented middleware for wireless sensor networks," Serv. Oriented Comput. Appl., vol. 5, no. 2, pp. 71–85, 2011.

[28] J. Koshy and R. Pandey, "Vm*: Synthesizing scalable runtime environments for sensor networks," in Proc. 3rd Int. Conf. Embedded Netw. Sensor Syst. (Sensys), 2005, pp. 243–254.

[29] P. A. Levis, "Application specific virtual machines: Operating system support for user-level sensornet programming," Ph.D. dissertation, Berkeley, CA, USA, 2005.

[30] C. Gronroos, "Service quality: The six criteria of good perceived service", Review of business, 9(3), 1988, p. 10.

[31] G. F. Anastasi, E. Bini, A. Romano, and G. Lipari, "A service-oriented architecture for QoS configuration and management of wireless sensor networks," in Proc. IEEE Conf. Emerging Technol. Factory Autom. (ETFA), 2010, pp. 1–8.

[32] J. M. Hyman, A. A. Lazar, G. Pacifici, "Real-time scheduling with quality of service constraints", IEEE Journal on Selected Areas in Communications, 9(7), 1991., pp.1052-1063.

[33] T. Chauhan, S. Chaudhary, V. Kumar, and M. Bhise, "Service Level Agreement Parameter Matching in Cloud Computing", World Cong. ICT, 2011., pp. 564-570.