An Agile Farm Management Information System Framework for Precision Agriculture

Pradeep Hewage
Edge Hill University
St Helens Road
Ormskirk
pradeep.hewage@edgehill.ac.uk

Mark Anderson
Edge Hill University
St Helens Road
Ormskirk
(+44)1695657634
mark.anderson@edgehill.ac.uk

Hui Fang
Edge Hill University
St Helens Road
Ormskirk
fangh@edgehill.ac.uk

ABSTRACT
Early Farm Management Information Systems (FMIS) implementations were focused on data entry and report generation within a farm. The growth in areas such as Precision Agriculture (PA) is changing the landscape and approach taken to develop such systems. The lack of standardised data formats to support the transfer of data between systems is one of the major concerns that have been raised relating to the development of FMIS. The modern FMIS based on the distributed systems which inherently supports the sharing of resources and data. Service-oriented architectures are more flexible in relation to the delivery of software packages; for example, cloud computing can be utilized to offer Software-as-a-Service (SaaS) to agriculturalists. In general, these systems are monolithic, which is built as a single unit of software. These monolithic systems have several challenges, especially scalability, rebuild, retesting and deployment issues. The modern microservice approach can be recognised as the solution for these issues. Microservice approach is organised a distributed application as a collection of services, where each service design, implement and run independently. The basic microservice concept is based on virtual machines and the Application Program Interface (API). The next level of container based microservice systems has less runtime overhead compared to virtual machine basic microsevices. This technology is also more efficient as many containers can be run in a single virtual machine. The container based system also comes with key features and tooling around to addressing microservice challenges. The researchers conclude the paper by recommending a container based microservice architecture for agile FMIS in precision agriculture.

CCS Concepts
• Enterprise computing→Service-oriented architectures • Computers in other domains→Agriculture

Keywords
Farm Management Information Systems; Microservices; Microservice Architecture;,Container-based microservices

ICIME 2017, October 9–11, 2017, Barcelona, Spain

ACM ISBN 978-1-4503-5337-3/17/10…$15.00
DOI: https://doi.org/10.1145/3149572.3149583

1. INTRODUCTION
The complexity and scale of data acquisition and processing within agriculture is growing at a significant rate. Whilst the notion of Farm Management Information Systems (FMIS) is not a new one, the earliest being cited in the mid-1980s, the growth in areas such as Precision Agriculture (PA) is changing the landscape and approach taken to develop such systems. Early FMIS implementations were focussed on data entry and report generation within a farm; mostly within the confines of a single farm or a highly defined farming sector. As such, these systems were limited in scope and did not offer transferable functionality that could be adopted by the wider farming community. They did, however, aid farmers and farm managers process elementary resource management data and adhere to some areas of legal compliance.

More recent developments have led to the consideration of extended capabilities to deliver efficiencies within agriculture and have led to an academic interest in the modelling and development of FMIS [1]. This has, in no small part, been driven by the rapid evolutions in Precision Agriculture (PA) [2][3]. Through the need to meet core requirements of PA, the academic focus has centred upon three key criteria [4]: Decision Support Systems (DSS), computational biological models and the usability of systems. The latter is a significant area of research; should a system not be usable for the target audience then it will not be adopted. In this scenario, the end-users can be categorised as being geographically disparate and non-IT specialists. This thereby raises a number of challenges in relation to the design of any user interface for such a system.

Perhaps of most interest has been researching into the development of DSS to support PA [5]. In many ways, it is this field of research that embodies the changing nature of FMIS. Within this context, a DSS will require access to timely and accurate data on which to base any outputs to the end user [4]. Complexity is introduced as the data that any DSS will use in this scenario will need to be drawn from multiple complex data sources. This may require access to geospatial data (for example, from GPS or GIS sources), real-time data (from farm machinery) and scientific modelling data (such as weather modelling and forecasting data). A significant difficulty in successfully achieving this lies in the lack of interoperability standards, although a number of standardised approaches have been considered and will be discussed below. It should be noted, however, that most of the research undertaken that relates to developing interoperability standards focusses on very defined areas within agriculture and, therefore, it is difficult to determine the scalability of these approaches. This would affect the operation of the system both at

the point of developing operation plans (such as those required for machinery) through to the development of strategic plans (such as the optimisation of resources required for treating crops).

2. INTEROPERABILITY STANDARDS

One of the major concerns that has been raised by researchers and practitioners relating to the development of FMIS has been the lack of standardised data formats to support the transfer of data between systems. Bespoke standards exist for elements that a FMIS would need to interact with; as an example, the ISOBUS standard for tractors and farm machinery. However, the development of a universally accepted standard for data interchange within the spectrum of FMIS has remained evasive.

Two attempts to develop such a standard are worthy of consideration within the context of this research. Both have attempted to address the implementation of a standard through the adoption of XML as a data description mechanism: agroXML [6] and Precision Agriculture Markup Language (PAML). Each was designed with a slightly different aim, yet both have the goal of offering a holistic solution to the implementation of a language for encapsulating agriculture-based information. The goal of agroXML has been as a means of passing data from within a FMIS to external sources. For example, this may be to suppliers or external partners. On the other hand, PAML was proposed by [7] as a solution to support interoperability between agriculture services within the context of a FMIS. In both cases, the languages extended Geography Markup Language (GML) [8]. Whilst GML, proposed by the Open Geospatial Consortium (OGC), is designed for expressing geographical features, its extension to allow description of field boundaries and operations would appear to be a natural development. Indeed, this only reflects the movement within the Precision Agriculture community to utilise geospatial data for planning and operational functions related to FMIS processing.

3. SOFTWARE ARCHITECTURE

In relation to architectural developments, several proposals have focussed on the need to develop systems based upon web service technology. Whilst the earliest systems were developed for individual farms, more recent developments have sought to make use of internet technology. This has primarily been driven with two goals in mind

1. Providing a FMIS that is based on a distributed system inherently supports the sharing of resources and data, thereby assisting collaboration between landowners and stakeholders

2. Service-oriented architectures are more flexible in relation to the delivery of software packages; for example, cloud computing can be utilised to offer Software-as-a-Service (SaaS) to agriculturalists.

However, this then leads to a further area of exploration in relation to the development of the software: how flexible is the architecture? Whilst the majority of the previous studies have been acknowledged to concentrate on a single area of farming, there is undoubtedly a generic set of functions or operations that the software supports which would be beneficial to a wider community of farmers. Equally, there have been studies that have considered the wider grouping of stakeholders who would share an interest in the processing of a FMIS.

4. FMIS AS MONOLITHIC APPLICATIONS

Table 1: Comparison of FMIS Systems

FMIS paper and crop	Description	Software architecture
A Conceptual Model of Farm Management Information System for Decision Support [9]	This is a framework that combines new agricultural technologies such as variable Rate Technology (VRT), Remote sensing, GIS, and GPS.	EA- Monolithic
A reference architecture for Farm Software Ecosystems [10]	This reference architecture should be used to map, assess design and implement Farm Software Ecosystems.	Open Software Enterprise - Monolithic
Mark Online, a Full Scale GIS-based Danish Farm Management Information System [11]	An internet based client-server concept with extensive use of web services based on open standards has proven to be a flexible way to integrate different applications and data from different platforms.	EA – Monolithic
The Study of SOA-based FMIS [12]	This is a framework to design and implement a SOA-based FMIS platform for a large-scale manufacturing enterprise.	SOA- Monolithic
Integrated Cloud Framework for Farm Management [13]	This is a multi-layer architecture in the context of IoT applied for a Smart Farming environment. This solution offers compose services executed on two different types of platforms: in the cloud, or on a mobile platform.	Cloud and IoT based monolithic
Cloud-Based Architecture for Farm Management [14]	This is a Cloud-Based Architecture for Farm Management which has two components: LFC (Local Farm Controller) and CFC (Cloud Farm Controller).	Cloud based- Monolithic

Monolithic system architecture commonly used in develop Enterprise Applications (EA) which is built as a single unit. Modern FMIS use web based EA which has three main sections; a user interface (client-side web page), a Database, and Server-side application. Table 1 lists examples of FMIS software developed through research projects, and also lists the architectures that were adopted. What is apparent it the monolithic approach that the projects have taken. This Server-side monolithic application able to handle requests from clients and manage transactions in the Database; meaning manage the entire system. Building and deploying a new version of the server-side application is always required to any changes to the system. These monolithic applications can be scaled horizontally by running many instances with a load balancer [15].

Xu et al. [16] state the monolithic architectural style has been dominant approach to structuring common EA since the early years of service-oriented architecture. These applications run few process where each process has a large number of functionalities. This may be a good method to start-up a company. The company is growing up means increasing the number of users, expand the database, and grows the domain logic complexity. One of the main challenges of this monolithic service-oriented application is the scalability issue. In scaling, the entire application must be scaled; and unable to scale a part of the application. Abbott and Fisher [17] added horizontal scaling of monolithic applications, especially in cloud computing, plays a crucial role by replicating multiple identical copies of the processes behind a load balancer. By way of explanation, the whole process has been duplicated if one of the component overloads, resulting in a high-cost involvement.

Another main challenge with the monolithic applications is to rebuild and deploy the entire application even changes made for a small part of the application. Most of the modern FMIS applications are deployed in a cloud system, resulting change cycles are more rigid and expensive. The server must be restarted to redeploying a new version of the application, resulting an increase time of system offline. This is especially problematic if the application is located on a cloud system with a large number of customers, where they cannot tolerate being offline frequently [16].

As a solution, the academics and researchers start to find an alternative architecture to develop EA which is capable of building applications as suites of services where each service can be individually developed (with different platforms and languages), deploy, and scalable. Each service should be able to manage by different teams and allow different services provide a firm module boundary. In another term, "split a SaaS application into a set of smaller, interconnected services" [18].

5. Microservices

Microservices can be described as organising a distributed application as a collection of services, where each service design, implement and run independently. Each independent service, or a microservice, can be managed, deployed and scaled its own. A microservice can be built based on its own architecture, platform, and technology. For example, an online ordering system application performs several functions, including address validation, access product catalogue, and credit check. In microservice architecture, these functions develop as small individual applications and group together to create the outcome. Therefore, highly decouple and focus tasks in a complex system can serve as microservices and, as a result, microservice act as basic building blocks in the application [19].

Xu et al., [16] stated: "microservices address the monolithic barriers by decomposing a system into a large number of fine-grained services that are connected together through a communication mechanism and supported by a deployment mechanism for replicating and relocating microservices in a cluster of servers". Microservice applications are based on the scope of the business context. These systems can be easily updated by replacing or updating relevant microservice as each microservice performs a singular task [20]. Collapsing or going offline a microsystem often not cause a devastating effect to the whole system. Only the particular microsystem is changed, rebuilt, tested, and redeployed within an extremely shorter time period compared to monolithic applications. These microservices are very simple and lightweight. Therefore, the update process might not recognise by the customers. The capacity of a microservice can be increased by increasing the number of instances or by moving its hosted target without any impact to consumers. Figure 1 shows the methodology of scaling monolithic and microservice applications. Fowler and Lewis [15] described microservices as the greatest way to assemble SaaS systems.

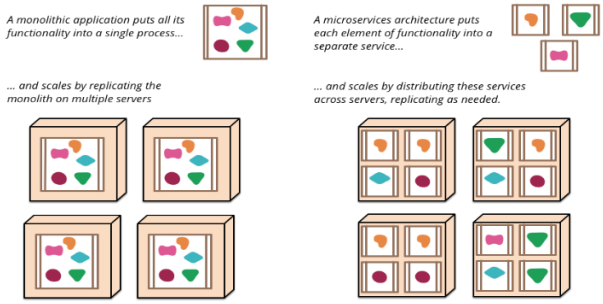

Figure 1. Monolithic and Microservices [15]

Wasson and Celarier [21] stated microservices applications suited and highly considered for "large applications that require a high release velocity, complex applications that need to be highly scalable, applications with rich domains or many subdomains, and an organisation that consists of small development teams". Most of these features included in an FMIS. Therefore, microservices architecture is highly considerable and highly suited to develop these applications.

5.1 Microservice Architecture

Wasson and Celarier [21] presented a typical microservice architecture based on services communicate with each other by using well-defined Application Program Interface (API). In addition to services, there are three main sections included in this basic architecture, namely; management (responsible for identifying failures, placing services on nodes, rebalancing services across nodes), service discovery (responsible to maintain a list of services and list of nodes with locations), API gateway (the entrance point for the clients which forwards the call to the appropriate inspection and repairs on the back end). The API gateway is capable of communicating with several services, aggregate responses, and return the aggregated response. The clients access to the API gateway and API decouple clients from services. APIs facilitate to communicate with services which are not web friendly, such as AMQP, and perform cross-cutting functions such as logging, load balancing, authentication, and SSL termination. Figure 2 shows the typical microservice architecture based on the API.

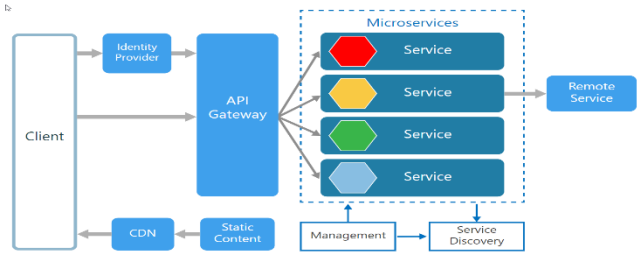

Figure 2. Typical microservice architecture [21]

Slater [22] stated the container based microservice architecture becomes the evolution of developing EAs, and it facilitates better use of computer resources and ability to maintain complex applications. For an example, Wasson and Celarier [21] added, "Azure Container Services supports several popular container orchestrators, including Kubernetes, DC/OS, and Docker Swarm". The container based microservice architecture comes with several components; a) Public nodes (nodes with the user interface reachable through a public-facing load balancer) b) Backend nodes (nodes run services but not receive user traffic directly. VMs utilise with different hardware profile) c) Management VMs (runs master nodes) d) Networking (managing VMs, backend nodes, and public nodes are placed in separate subnet within the same virtual network / VNet) e) Public node load balancers (internet requests are distributed to public nodes) f) Management VM load balancer (secure shell traffic to manage VMs). Figure 3 shows the typical container based microservice architecture.

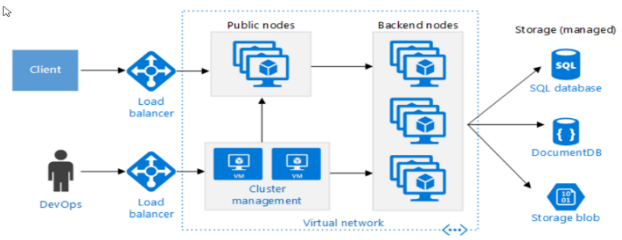

Figure 3. Container-based microservice architecture [21]

Xu et al. [16] presented a different microservice architecture based on the maturity model of SaaS applications, which has 4 layers, namely; a) Level 1 (application runs in a dedicated server for each customer with their own version of customised and tailored interface) b) Level 2 (a configurable copy of the application is provided to each customer) c) Level 3 (Meta data is used to configuration and customisation of the software. The vendors provide virtual separation between the tenets and the tenants share an instance of the software) d) Level 4 (A load balancer is used to serve multiple instances of the software). The improvements in efficiency, scalability and reconfigurability evolve from lower level to the higher level. Level 4 of this architecture can be considered as the microservice architecture since only the components of a SaaS application / microservices are duplicated and distributed to different servers. This system answer for scalability and efficiency issues, but not for flexibility for configuration and customisation.

As a solution, virtual machine (VM) concept is introduced to the system as a heavyweight solution as VMs visualise hardware and gives the physical customer separation. The VMs execute services regardless operating system and the hardware platform. The only challenge is unable to deploy many VMs to a single server as these VMs consumes system resources. As a solution to the shortage of VM issues, Pahl [23] suggested a container solution. These containers share system processes, system memory and the file systems and facilities to provide separation to customers. The container technology is more efficient compared to VMs as many containers can be run in a single virtual machine. This has less runtime overhead compared to running several VMs, as shown in figure 4.

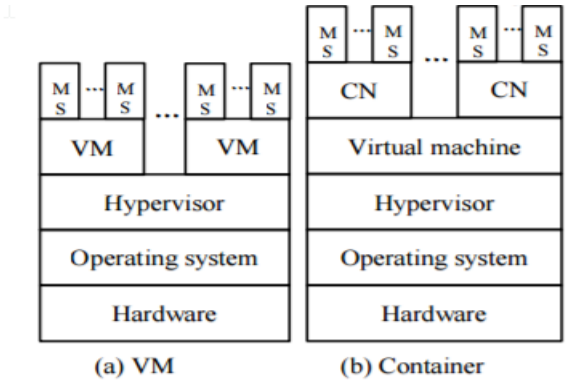

Figure 4. Architectures for Microservices Technology [16]

5.2 Framework for Microservice FMIS

As described in section 5.2, the VM systems execute microservices regardless operating system and hardware platform. The main challenge with this system is to enable deploy many VMs to a single server. The VMs consumes system resources and increase runtime overhead [16]. The popular programming languages such as Java, C#, C, Ruby, Python and C++ also use the VMs concept to execute a microservice applications [24]. In the agricultural industry, there is no universally accepted standard for data interchange in the; bespoke standards exist for elements that a FMIS would need to interact with. Therefore, it is a challenging issue to convert data into acceptable standard and use an above popular language to develop a microservice application for FMIS.

Jaramillo, Nguyen and Smart [25] describe the container based system comes with the key features and tooling around to addressing microservice challenges such as failure isolation, higher level complexity of development and deployment, testing challenges, critical requirements of automation and so on. They further added the container systems enhance the value of microservice systems by; a) Accelerate automation (containers can be used as a deployment unit to granularly contain service, and there are plenty of tools for creating and launching) b) Accelerate the independence (Containers are isolated boxes which can be run on any platform. The development team can select best technology, language, tools, or process for the service) c) Accelerate portability (Containers are portable, platform independent, and especially facilitate to independent isolated testing) d) Accelerate resource utilization (Containers help to create more instances easily resulting resource utilisation due to their light weight nature) e) Secured (Most container providers offering flexibility to maximize the security at different levels).

There are many kinds of container services emerge in the last few years. Many companies have adopted to Docker open source

which automates the deployment of applications inside containers. This is done by providing an additional layer of abstraction and automation of operating system-level virtualization. Google has published Kubernetes open source container system which is highly scalable and automate the deployment. Amazon published a highly scalable container service called Amazon ECS. This makes easy to run and manage containers on a cluster of Amazon EC2 instances. Oracle has published Solaris Zones container system, and Microsoft has published the Windows container system [16].

5.3 Benefits of Microservices

There are many benefits of Microservice application compared to monolithic. Most of these benefits already discussed in the above chapters. Wasson and Celarier [21] discuss the main benefits of microservices are; a) Independently deployments (able to update a service without redeploying the entire system, resulting less risky and more manageable for bug fixing and feature releases) b) Independent development (build, test and deploy a service by a single development team focusing continues innovation and faster release cadence) c) Small, focus team (Each service is belonging to a single team. The service is a smaller scope means easier to understand code base and easy to train a new team member) d) Fault isolation (the entire application will not take out for a service down) e) Mixed technology stacks (best technology and platform can be selected based on the service) f) Granular scaling (independently scale the services)

5.4 Challenges in Microservices

In contempt of many advantages, there are several disadvantages in microservices, especially in designing stage of the process. Wasson and Celarier [21] discuss the main challenges in microservices; a) Complexity (there are several moving parts compared to monolithic applications. The entire system is more complex, even though each service is simpler) b) Development and test (different approach requires developing against service dependencies. It is also difficult to refactor across service boundaries. The application is evolving quickly, therefore, challenging to test service dependencies) c) Lack of governance (Hard to maintain the application as there are so many different approaches, languages and frameworks are used to develop services. The introduction of project-wide standards is a solution to overcome this situation by restricting team flexibility) d) Network congestion and latency (More inter-service communication is required to communicate in-between many small and granular services. Additional latency problems will generate if the chain of service dependencies gets too long. These problems can overcome by carefully designing, use asynchronous communication for suitable services, and use serialisation formats) e) Data integrity (The data consistency can be a challenge as each microservice responsible for its own data persistence) f) Management (it is challenging the correlated logging across services. A mature DevOps culture utilise to overcome this situation) g) Versioning (Updating several services sometimes challenging as this might break services that depend upon) h) Skillset (Each team should have skills, capabilities, and experiences to success with highly distributed systems)

6. Conclusion

The lack of standardised data formats to support the transfer of data between systems is one of the major concerns that have been raised relating to the development of FMIS. Bespoke standards exist for elements that a FMIS would need to interact with; as an example, the ISOBUS standard for tractors and farm machinery. Whilst the majority of the previous studies have been acknowledged to concentrate on a single area of farming, there is undoubtedly a generic set of functions or operations that the software supports which would be beneficial to a wider community of farmers. Modern FMIS based on a distributed system which inherently supports the sharing of resources and data. Service-oriented architectures are more flexible in relation to the delivery of software packages; for example, cloud computing can be utilised to offer Software-as-a-Service (SaaS) to agriculturalists. In general, these systems are monolithic, which is built as a single unit of software.

There are several challenges in monolithic applications especially scalability, and rebuilt and deployment issues. As a solution, the microservices architecture has been introduced. The basic architecture is based on virtual machines and the APIs. The microservices can be described as organising a distributed application as a collection of services, where each service design, implement and run independently. Each independent service, or a microservice, can be managed, deployed and scaled its own. A microservice can be built based on its own architecture, platform, and technology. There are several challenges in microservice systems, but this is a highly productive and efficient system for Enterprise Applications with added benefits. This also a better solution for interoperability issues in FMIS. Therefore, microservice architecture is recommended for FMIS.

Container based microservice systems have less runtime overhead compared to virtual machines. This technology is also more efficient as many containers can be run in a single virtual machine. The container based system also comes with key features and tooling around to addressing microservice challenges such as failure isolation, higher level complexity of development and deployment, testing challenges, critical requirements of automation and so on. Hence the container system is an effective solution to microservice architecture; therefore, the container based microservice architecture is recommended for agile FMIS in precision agriculture.

7. REFERENCES

[1] Nash, E., Dreger, F., Schwarz, J., Bill, R., and Werner, A. (2009) “Development of a model of data-flows for precision agriculture based on a collaborative research project”, Computers and Electronics in Agriculture, no 66, pp 25-37, Elsevier

[2] Camilli, A., Cugnasca, C. E., Saraiva, A. M., Hirakawa, A. R. and Correa, P. L. P. (2007) “From wireless sensors to field mapping: Anatomy of an application for precision agriculture”, Computers and Electronics in Agriculture, no 58, pp. 25-36, Elsevier.

[3] Nikkilä, R., Seilonen, I., and Koskinen, K. (2010) “Software architecture for farm management information systems in precision agriculture”, Computers and Electronics in Agriculture, no 70, pp. 328-336, Elsevier.

[4] Fountas, S., Carli, G., Sørensen, C. G., Tsiropoulos, Z., Cavalaris, C., Vatsanidou, A., Liakos, B., Canavari, M., Wiebensohn, J., and Tisserye, B. (2015) “Farm management information systems: Current situation and future perspectives”, Computers and Electronics in Agriculture, no 115, pp 40-50, Elsevier.

[5] Tan, L., Hou, H., and Zhang, Q. (2016) “An Extensible Software Platform for Cloud-based Decision Support and Automation in Precision Agriculture”, In Proceedings of 17th International Conference on Information Reuse and Integration, 28–30 July 2016, Pittsburgh, Pennsylvania, IEEE

[6] Santos Junior, C. and Hirakawa, A. R. (2012) “The Use of Agroxml Standard for Data Exchange Processes in the Cotton Culture”, IEEE Latin American Transactions, 10(1), pp 1425-1427.

[7] Murakami, E., Saraiva, A. M., Ribeiro Junior, L. C. M., Cugnasca, C. E., Hirakawa, A. R. and Correa, P. L. P. (2007) “An infrastructure for the development of distributed service-oriented information systems for precision agriculture”, Computers and Electronics in Agriculture, no 58, pp 37-48, Elsevier.

[8] Cox, S., Cuthbert, A., Lake, R., Martell, R., 2002. GML Implementation Specification, version 2.1.2. OpenGIS Implementation Specification. Available from https://portal.opengeospatial.org/modules/admin/license_agreement.php?suppressHeaders=0&access_license_id=3&target=http://portal.opengeospatial.org/files/%3fartifact_id=26765 (28-03-2017).

[9] Burlacu, G., Costa, R., Sarraipa, J., Jardim-Gonçalves, R. and Popescu, D., 2014, April. A Conceptual Model of Farm Management Information System for Decision Support. In DoCEIS (pp. 47-54)

[10] Kruize, J.W., Wolfert, J., Scholten, H., Verdouw, C.N., Kassahun, A. and Beulens, A.J., 2016. A reference architecture for Farm Software Ecosystems. Computers and Electronics in Agriculture, 125, pp.12-28.

[11] Bligaard, J., 2014. Mark Online, a Full Scale GIS-based Danish Farm Management Information System. International Journal on Food System Dynamics, 5(4), pp.190-195.

[12] Zhou, Y.J. and Li, X.Y., 2011, September. The study of SOA-based FMIS. In Industrial Engineering and Engineering Management (IE&EM), 2011 IEEE 18Th International Conference on (pp. 485-488). IEEE.

[13] Radu, C., Apostol, E., Leordeanu, C. and Mocanu, M., 2016, July. Integrated Cloud Framework for Farm Management. In Complex, Intelligent, and Software Intensive Systems (CISIS), 2016 10th International Conference on(pp. 302-307). IEEE.

[14] Mocanu, M., Cristea, V., Negru, C., Pop, F., Ciobanu, V. and Dobre, C., 2015, May. Cloud-based architecture for farm management. In Control Systems and Computer Science (CSCS), 2015 20th International Conference on (pp. 814-819). IEEE.

[15] Fowler, M. and Lewis, J. (2014) Microservices- a definition of this new architectural term, 25 Mar, [Online], Available: https://martinfowler.com/articles/microservices.html [22 Jun 2017].

[16] Xu, C., Zhu, H., Bayley, I., Lightfoot, D., Green, M. and Marshall, P. (2016) 'CAOPLE: A Programming Language for Microservices SaaS', IEEE Symposium on Service-Oriented System Enginering, Mar, pp. 42-52.

[17] Abbott, M.L. and Fisher, M.T. (2009) The Art of Scalability: Scalable Web Architecture, Processes, And Organizations for The Modern Enterprise, Pearson Education.

[18] Richardson, C. (2015) Introduction to Microservices, 19 May, [Online], Available: https://www.nginx.com/blog/introduction-to-microservices/ [27 Jun 2017].

[19] Xiao, Z., Wijegunarathne, I. and Qiang, X. (2016) 'Reflections on SOA and Microservices', 4th International Conference on Enterprise Systems, 60-66.

[20] Vandikas, K. and Tsiatsis, V. (2016) 'Microservices in IOT clouds', 1-6.

[21] Wasson, M. and Celarier, S. (2017) Microservices architecture style, 26 Jun, [Online], Available: https://docs.microsoft.com/en-us/azure/architecture/guide/architecture-styles/microservices [27 Jun 2017].

[22] Slater, N. (2015) Using Containers to Build a Microservices Architecture, 20 Jan, [Online], Available: https://medium.com/aws-activate-startup-blog/using-containers-to-build-a-microservices-architecture-6e1b8bacb7d1 [27 Jun 2017].

[23] Pahl, C. (2015) 'Containerization and the PaaS Cloud', IEEE Cloud Computing, vol. 2, no. 3, Jun, pp. 24-31.

[24] Nolle, T. (2016) EVALUATE: The right language for microservice orchestration and development, Jul, [Online], Available: http://www.theserverside.com/feature/The-right-language-for-microservice-orchestration-and-development [28 Jun 2017].

[25] Jaramillo, D., Nguyen, D.V. and Smart, R. (2016) 'Leveraging microservices architecture by using Docker technology', pp. 01-05..