Complex Distributed Systems

The Need for Fresh Perspectives

Gordon S. Blair
School of Computing and Communications,
Lancaster University,
Bailrigg, Lancaster, UK
g.blair@lancaster.ac.uk

Abstract— **Distributed systems are at a watershed due to their increasing complexity. The heart of the problem is the extreme level of heterogeneity exhibited by contemporary distributed systems coupled with the need to be dynamic and responsive to change. In effect, we have moved from distributed systems to systems of systems. Following on from this, middleware is also at a watershed. The traditional view of middleware is no longer valid (i.e. as a layer of abstraction, masking the complexity of the underlying distributed system and providing a high-level programming model). In practice, middleware is often by-passed with complex systems constructed in a rather ad hoc manner as a mash-up of a variety of technologies. The end result is that middleware is no longer sure of its form or purpose and this lack of a viable approach is a huge barrier to the emergence of areas such as smart cities and emergency response systems. This paper argues that there is a need to fundamentally rethink the middleware landscape related to complex distributed systems. The core contribution of the paper is a set of fresh perspectives, which lead us in turn to novel principles and patterns for middleware and subsequently to new styles of platform. These perspectives include a move to emergent middleware, seeking flexible meta-structures for distributed systems, and a step away from generic to domain-specific technologies. A number of case studies are also presented to demonstrate what this might mean for future distributed systems.**

Keywords— distributed systems, systems of systems, middleware, complexity, systems integration.

I. INTRODUCTION

Distributed systems are at a watershed due to their increasing complexity. The heart of the problem is the extreme level of heterogeneity exhibited by contemporary distributed systems coupled with the need to be dynamic and responsive to change. In effect, we have moved from distributed systems to systems of systems. Following on from this, middleware is also at a watershed. The traditional view of middleware is no longer valid (i.e. as a layer of abstraction, masking the complexity of the underlying distributed system and providing a high-level programming model). In practice, middleware is often by-passed with complex systems constructed in a rather ad hoc manner as a mash-up of a variety of technologies. The end result is that middleware is no longer sure of its form or purpose and this lack of a viable approach is a huge barrier to the emergence of areas such as smart cities and emergency response systems.

The author argues that *there is a need to fundamentally rethink the landscape of distributed systems in what the paper refers to as complex distributed systems*. The *overall vision* of the paper is therefore *to examine the problem through fresh perspectives, that lead us in turn to novel principles and patterns for middleware and subsequently to new styles of platform*. The fresh perspectives include a move to emergent middleware, seeking flexible meta-structures for distributed systems, and a step away from generic to domain-specific technologies. These findings are strongly influenced by recent work by the author in applying distributed systems technology in understanding and managing the natural environment as it experiences pressures such as climate change.

The paper is structured as follows. The arguments in this paper are derived from experience in applying distributed systems principles and techniques in a real-work problem domain, that of supporting environmental scientists in understanding and managing the natural environment. Section II discusses this motivation, highlighting the intrinsic complexities in both the resultant underlying distributed systems and also associated with supporting environmental science. Section III then discusses the state of the art in complex distributed systems, examining in detail how existing middleware approaches support such complexities and also what can be learned from research in systems of systems. The section concludes that there are serious problems in the state of the art and a need for fresh perspectives and insights to drive a new approach to complex distributed systems. Section IV then moves on to the solution space by discussing five potential fresh perspectives on the problem that individually and/or collectively may point to new solutions and approaches to middleware design. Section V then presents four case studies that build selectively on the fresh perspectives in different combinations, thus further developing the potential solution space. Section VI presents some concluding remarks highlighting the need for greater focus on complexity in distributed ssytems and its application.

II. MOTIVATION: DISTRIBUTED SYSTEMS IN THE REAL WORLD

A. Background

This paper is motivated by recent research by the author in applying distributed systems technology in a real world setting, namely supporting environmental scientists in their quest to both understand the complexities of the natural environment

under periods of change (incl. climate change) and in developing well-founded mitigation and adaptation strategies.

Previous research includes the Environmental Virtual Observatories project, which examined the support offered by cloud computing for a more open and collaborative style of science, and also the Environmental Internet of Things project which addressed the potential role of Internet of Things (IoT) technology in supporting catchment management by providing real-time streaming data on a number of environmental facets including concerning soils, hydrology and animal movements.

Ongoing work is examining the natural synergy between IoT technology, cloud computing and data science in environmental science (generating rich data-sets, having the capacity to analyse such data sets, and also providing innovative techniques to make sense of this complex environmental data). Experience from this work has highlighted the complexities of working in this area. The rest of this section unpicks these complexities in more detail.

B. *Complex Distributed Systems*

The underlying distributed systems are intrinsically complex in terms of their design, programming, deployment and maintenance. They are classic examples of systems of systems architectures as recognised for example by the GEOSS project (Global Earth Observation Systems of Systems) [1]. GEOSS has the goal of bringing together a rich range of observational data and associated processing systems to provide a comprehensive and coordinated resource for a range of stakeholders, public and private, related to the state of the Earth (see Fig. 1).

Fig. 1. GEOSS System of Systems

Such systems typical involve rich architectures for data capture operating at different scales including ground-based sensors, e.g. exploiting emerging IoT technology, coupled with airborne sensors on aircraft and drones, and supplemented by remote sensing from satellites. Focusing on ground-based sensors, there will not just be one Internet of Things, but a cascade of deployments examining different environmental facets again at different scales. There is then a need to store this data and support subsequent discovery and manipulation, potentially using cloud technology to provide, e.g., the elastic capacity to manage such large volumes of data. Additional distributed systems elements may also be introduced. For example, there may be the need to support access to the data and services from mobile devices, implying support for context awareness. Similarly, in emergency response systems, it may be necessary to base elements of the design on ad-hoc networking principles to support self-organisation of a system. More generally, in such systems there is a need for dynamicity and to adapt to changes in the operating environment. There is also a need to enforce security and dependability across such systems. In other words, there is a need to embrace and build on many of the developments from the distributed systems community over the past 30-years. A key difference is that this has to be achieved in a single architecture embracing all of these aspects and this raises profound questions:

- what tools and techniques do we have to design such an architecture recognising its intrinsic systems of systems nature;
- what programming paradigms and languages are available to develop such complex distributed systems in a well-founded manner and that supports reasoning about end-to-end properties and evolution of such systems;
- how do we support a basic property of interoperability across such a complex architecture;
- how do we support more demanding properties of such systems such as meeting the end-to-end security and dependability requirements of such systems;
- how do you support adaptation and an ability to respond to changes in needs and/or context?

(Note that this complexity is not unique to earth observation and management; indeed, a similar case could be made for many other areas of real-world application e.g. in smart cities and digital health.)

C. *Complex Science*

There are also intrinsic complexities associated with the application domain that amplify the difficulties in building appropriate systems architectures. As raised above, there has been a significant shift in earth and environmental sciences towards approaches that are more integrated and collaborative and this often has gone hand in hand with more open approaches to science (hence the focus on cloud computing as a single, logical place to host the science). This stems from the need to answer 'big; scientific questions around for example, climate change and the impacts this might have on national and international policy and on more local issues such as catchment management. This is most evident in the move towards natural capital and ecosystem services [2, 3]. Natural capital is concerned with the world's stocks of natural assets, including its soil, water, air, energy sources and all living entities on the planet. The study of ecosystem services then investigates the sustainable and integrated management of complex ecosystems in the support of the services we need to live, hence reifying the complexity of this management in all its facets including environmental, social, health and economic considerations [4].

There are a number of intrinsic difficulties that stem from the move towards an assessment of ecosystems in all their complexities. Many of these are related to the need for, and difficulties associated with, integration. First of all, there is a

need to integrate data from a variety of sources and support the subsequent analysis of this data. Environmental data is however highly heterogeneous and this is much more difficult to deal with. In the data science domain, data is often described using the four 'V's of data: volume, velocity, variety and veracity [5]. In many areas of data science, volume and velocity dominate, and hence there is a need to design underlying distributed systems that scale in terms of space and time, and this is actually relatively well-understood. In the earth and environmental sciences, variety and veracity (cf. Accuracy or precision) often dominate. This is not to diminish the first two 'V's as there are areas where these can be very important, e.g. in climate science where models can generate massive data-sets but this only exacerbates the heterogeneity that you encounter. Some observers refer to the long tail of science whereby there are a number of very large data sets in the environmental sciences but equally there is a long tail of much smaller data-sets that can still be very important to the overall process of scientific discovery.

Veracity is equally important in this area of science as data is intrinsically linked to its source (cf. accuracy of instrumentation, the increasing importance of citizen science data, etc.).

The other key requirement is to support integrated modelling, that is the combination of multiple environmental models to solve complex real-world problems. This may include linking models that focus on different environmental facets to understand inter-dependencies, e.g. understanding the impact of climate change scenarios on soils and the resultant impacts of biodiversity [6]. In additional, individual models may be run very large numbers of times to understand potential uncertainties in model prediction and sensitivities to different parameters. Integrated modelling is complicated further by the existence of different styles of model including process models (capturing the underlying physical or chemical processes inherent in the system being modelled), statistical models (derived from the data) and agent-based models (increasingly used to capture complex behaviours in such systems).

Blair et al. [7] contains a more in depth analysis of the challenges associated with managing complex data and its analysis. The clear message that emerges though is that heterogeneity dominates and that designing for variety and veracity (effectively a dimension of heterogeneity) is much more difficult and less well understood than designing for volume and velocity. Equally importantly, this cannot be separated from the design of the underlying distributed system as they are intrinsically linked; the underlying distributed system platforms and services must support heterogeneity in the same way that they often successfully support scalability.

III. State-of-the-art in complex distributed systems

This section presents an analysis of the state-of-the-art in distributed systems, arguing that we are entering a world of increasing complexity, partly due to the extreme level of heterogeneity in contemporary distributed systems and magnified by the move towards systems of systems, and partly due to the complexity in the application domains we seek to support. The end result is a crisis in distributed systems, and an associated crisis in middleware as it struggles to deal with this complexity.

A. Middleware for Complex Distributed Systems

Middleware refers to a software layer that sits between applications/ services and the underlying physical distributed systems architecture, that provides appropriate programming abstractions and that masks out the heterogeneity and complexity of the underlying physical distributed systems infrastructure [8, 9]. As such, middleware has a crucial role to play in modern software architecture as we move to a world where distributed systems are ubiquitous. The earliest middleware focused on providing suitable communications primitives such as RPC [10, 11] or group communication [12] but these solutions lacked abstraction. In this respect, OMG CORBA [13] represented a major breakthrough offering a higher level programming abstraction based on distributed objects.

One of the key goals of middleware is to manage heterogeneity (in terms of underlying networks, machine architectures, operating systems and programming languages), but very quickly ***middleware heterogeneity itself became a major problem***. In other words, the intended solution is now part of the problem. This is partly due to competition in the marketplace given the importance of middleware in the software industry and partly due to technical innovation. The first step towards middleware heterogeneity was when Microsoft introduced DCOM [14] as a direct competitor to CORBA and this competition between major industry players has continued to this day. In terms of technical developments, there have been two key areas of innovation:

- *The search for appropriate programming abstractions*. It became clear, for example, that distributed object technology suffered from a number of significant shortcomings, particularly in larger-scale and more dynamic and evolving environments [15], and this led to consideration of component technologies as an alternative approach, including lightweight component technologies to support configurable distributed systems [16, 17], and more heavyweight Enterprise platforms combining component technology with the concepts of containers to introduce a more managed approach to distributed systems [18, 19]. Following on from this, web services were introduced as an approach that is more in line with web standards and hence better aligned with Internet-scale distributed systems [20].
- *The search for appropriate communication abstractions*. The earliest middleware technologies were based on client-server communication in the form of remote procedure calls or remote method invocation. These styles of communication are still important but they suffer from a number of drawbacks due to the tight coupling between communicating parties, leading to fragility in the underlying distributed systems. As such, researchers have investigated more indirect communication paradigms and these include message passing, group communication, publish-subscribe

systems, message queues, distributed shared memory, and tuple-space communication [8].

These two areas are not independent and the different solutions can be combined in different ways to provide a rather bewildering range of middleware options. The range of options (and associated level of heterogeneity) becomes even broader in systems of systems.

B. *Middleware for Distributed Systems of Systems*

As argued above, distributed systems are becoming even more complex. While some researchers emphasise the challenge of scale, as we move towards exascale distributed systems [21, 22], ***the author argues that the real complexity stems mainly from heterogeneity and the move from distributed systems to (distributed) systems of systems***. According to Jamshidi [23], systems of systems are "large-scale integrated systems which are heterogeneous and independently operable on their own, but are networked together for a common goal". This style of architecture is prevalent in many important areas of application including smart cities, intelligent energy management and emergency response systems.

The underlying systems that constitute a 'systems of systems' architecture can vary enormously. One common pattern is the bringing together of cloud computing [24] and Internet of Things (IoT) technology [25]. Indeed these two great pillars of innovation are highly synergistic, with the IoT providing fine-grained and real-time streaming data for a large number of facets, but without the capacity to store, process or generally make sense of this data. In contrast, cloud computing provides large and elastic storage and processing capabilities and also the potential for higher level services to analyse and visualise and generally make sense of the vast amounts of data that will be generated by the Internet of Things. It is also common to extend such systems of systems to support mobility [26], whereby users can access information on the move from smart phones or tablets. We now look at the state-of-the-art in middleware for each of these areas independently and then consider solutions for (systems of systems) integration.

- *Middleware for cloud computing*. Cloud computing is arguably the most transformative development in distributed systems with its move towards computing as a utility, making computational resources from (physical or virtualised) infrastructure through platforms to software applications available as services in the Internet [24]. There has been a corresponding burst of innovation in middleware for cloud computing. A number of commercial middleware solutions are available including Amazon Web Services, Microsoft Azure and the Google App Engine. Open source solutions are also available including Hadoop [27] and OpenStack [28]. There has also been a range of more specific innovations in areas such as consistency [29], software frameworks for computation [30], and data-storage services [31, 32]. Again, a prevalent trend is increasing heterogeneity with different vendors offering different middleware solutions, each with their own computational paradigm and set of APIs, leading to a real danger of vendor lock-in. While technologies such as jclouds [33] have emerged to tackle this problem, these are limited in scope, e.g. focusing on basic compute and data storage abstractions. Consequently, the area of cross-cloud management is now an important topic with researchers working on the concept of cross-cloud brokerage [34]. Note that the author has been heavily involved in such initiatives, organising a series of recent workshops on the topic of cross-cloud management (at IEEE Infocom'14 and Middleware'14) and carrying out research on the broker concept [35].
- *Middleware for the Internet of Things*. The Internet of Things is a second area of intense innovation in distributed systems with its emphasis on extending the scope of distributed systems to encompass not just computers but also physical objects with embedded sensors, actuators and computational/ communication capability [25]. Middleware for the Internet of Things is in its infancy, certainly when compared to the explosion of research in cloud computing. There have though been quite a number of projects looking at middleware for wireless sensors networks (a key enabling technology for the Internet of Things) [36, 37]. A significant number of projects have examined programming abstractions for wireless sensor networks, generally taking a more data-oriented approach based on SQL-like queries [38], tuple-spaces [39] or publish-subscribe [40]. Macroprogramming, i.e. programming for the network as a whole, is an associated interesting development [41]. There is also a plethora of research projects looking at underlying systems components that contribute to middleware architectures, such as operating system libraries [42, 43] or ad hoc routing protocols [44]. This work is feeding into middleware for the Internet of Things and a number of experimental platforms [45, 46, 47, 48] have been developed. There is generally a lack of maturity in this work, and again we see a high-level of heterogeneity at the middleware level (and also incidentally in the underlying operating systems, machine architectures and programming languages used).
- *Middleware for mobile computing*. Mobile computing is concerned with providing access to distributed services from mobile devices such as smart phones and tablets [26]. Once again, a large number of projects have looked at the design of suitable middleware abstractions and techniques to encompass mobile elements. Important developments include: i) communication paradigms to deal with disconnection, including the use of indirect communication technologies such as tuple-spaces or publish-subscribe [49, 50]; ii) support for spontaneous interaction through dynamic service discovery [51, 52]; iii) support for location- or context-awareness [53]; iv) associated techniques to support adaptation to varying context [54]. Compared to the Internet of Things, this area is more mature and many of the techniques developed are available in commercial products. Again though, there is a significant level of heterogeneity introduced, most notably in the areas of

service discovery and subsequent interaction where multiple approaches co-exist.

With the move to systems of systems [55], *systems integration* is crucial. In particular, there is a pressing requirement for middleware that spans the different systems and manages end-to-end properties of the system [56]. In particular, there is a need for techniques to support: i) end-to-end interoperability; ii) end-to-end quality of service (QoS), including key properties such as (real-time) performance, security and dependability; iii) overall programmability of the integrated system; iv) its subsequent management, including dealing with changing context. Despite the importance of this area, research is absolutely in its infancy. The tendency has been for researchers to focus on middleware for the given domains (cloud computing, the Internet of Things and mobile computing) and to ignore the crucial aspects of systems integration across these domains. It is interesting to note that there are two communities involved in this work: a community looking at systems of systems engineering and one looking at distributed systems/ middleware, and it is striking that there is little or no overlap between these communities. As an indication, if you carry out a Google search for middleware and "systems of systems", you get surprisingly few meaningful results despite its obvious importance (with a few notable exceptions incl. the work of my own group [57] and that of Doug Schmidt [58]). Similarly, looking at the major conference in systems of systems, the IEEE Conference on Systems of Systems Engineering [59], the technical programme emphasises topics such as areas of application, modelling and analysis methods, and control techniques, with little attention to distributed systems or middleware topics. A number of European projects have been launched in this area, including COMPASS [60], DANSE [61], Road2SoS [62] and T-AREA-SoS [63] but again distributed systems perspectives are missing. There is clearly a need for further research on middleware for systems of systems.

C. Overall Analysis

The overwhelming conclusion from this survey is that there is an ***explosion in heterogeneity*** and this is true in two dimensions: i) in the underlying network technologies, computer architectures, operating systems and programming languages used, and ii) in the middleware itself where there is an explosion in styles of middleware used in terms of programming abstractions, communication abstractions and underlying systems principles and techniques. The author refers to this as ***extreme heterogeneity*** and argues that this is a major problem for distributed systems. There is also a problem of ***dynamism*** with such systems operating in volatile environments, having to respond to changing context.

This leads the author to the conclusion that, despite its many achievements and breakthroughs, ***distributed systems is in crisis*** in that there is little understanding of how to achieve end-to-end interoperability, quality of service, programmability or management in such complex distributed systems. This is a ***crisis of complexity***. Following on from this, the author also argues that ***middleware is in crisis***, and that this is a ***crisis of identity***. The traditional approaches to middleware of offering generic, layered solutions, simply do not work in the complex distributed systems of today: i) such solutions only really work in the middle range in multi-scale systems, failing completely to address the needs smaller-scale IoT technology or indeed large-scale cloud-based environments; ii) they also fail to address a systems of systems perspective and the need to span multiple domains of usage; iii) they lack support for dynamism and the ability to respond to changing context. The end result is that developers operating in this space often by-pass middleware and develop applications in an ad hoc manner, creating mash-ups using a variety of different technologies, and being exposed to and having to manage the underlying complexity of distributed systems. This leads in turn to the crisis of identity, whereby middleware is no longer sure of its form or purpose, and this lack of effective solutions is a major barrier to the emergence of the areas such as smart cities, precision agriculture or emergency response systems.

It is clear that the solution is not just about making 'better' middleware. Rather, the author argues that there is a need to ***fundamentally rethink the role of middleware*** in what we refer to as complex distributed systems. This echoes the ***overall vision*** of this paper ***to examine the problem through fresh perspectives, that lead us in turn to novel principles and patterns for middleware and subsequently to new styles of platform.*** This paper is intended to provoke the distributed systems community to stand back from the body of work amassed over the last 30 years or so, and to go back to basics, to rethink the very foundations of distributed systems and middleware (and associated areas such as interoperability, quality of service, programmability and management) and to seek such fresh perspectives.

IV. Fresh Perspectives

As mentioned above, the goal of this paper is to stand back from the plethora of work in distributed systems (including the key area of middleware) and re-examine the area through fresh perspectives/ new lenses. In particular, ***four fresh perspectives*** are proposed, having been distilled following a period of deep reflection by the author, motivated by personal experiences in applying distributed systems in environmental science and also by the associated perception of a crisis of middleware. These are introduced briefly below and expanded on below:

- from design-time to run-time – the search for emergent middleware;
- from software platform to software frameworks – meta-structures for distributed systems;
- from generic to domain-specific – reasoning about domains and boundaries;
- from systems-oriented to application-oriented – raising the level of abstraction.

We look at each of these areas in more detail below.

A. Emergent Middleware

Middleware is generally considered as a technology that is designed in advance and then deployed as a fixed entity. With an ***emergent middleware*** approach, middleware is viewed as a run-time entity where middleware solutions are generated on-

demand for the current context. Such an approach inevitably involves a strong element of machine learning to determine the most appropriate structures for different contexts.

This generative approach offers a radically new perspective on middleware design. The advantage of an emergent middleware approach is that it is more naturally adaptive and able to react to change, including changes not previously seen or observed. Initial work was carried out in this area by the Connect project (discussed below).

B. Software Frameworks

Software frameworks emerged in the software engineering community, offering a given behaviour but where key parts of the implementation can be specialised or altered [64]. They offer a balance between reusability, in that large parts of the framework can be used as is, and customisability, in that key parts can be specialised, e.g. through over-riding or specialisation. They also offer an inversion of control whereby the software framework takes responsibility for what behaviours should be invoked for a given operation [65]. While they have been used extensively in some fields of application, their use in distributed systems or middleware is somewhat limited. Bertran *et al*. [66] experiment with the use of software frameworks in the design of sense/ compute/ control applications in distributed systems. The cloud computing platform, MapReduce, can also be seen as a limited form of software framework allowing significant re-use around the areas of distributed systems management, and also customisation of application behaviour through appropriate *map()* and *reduce()* functionality [30]. Other aspects though remain static. In this paper, we suggest going much further whereby the software framework concept acts as a scaffolding or meta-structure that can be instantiated in potentially radically different ways, achieving a sophisticated balance between reusability and customisation.

C. Domain-Specific Solutions

The dominant approach in the middleware community has been to seek generic platforms and interfaces that can be applied everywhere, hence offering portability and interoperability. It is clear though that this generic approach no longer works given the level of heterogeneity and the challenges of dealing with multi-scale environments. We are therefore looking at a move from generic to domain specific solutions that, in turn, will imply reasoning about domains and their boundaries. The field of domain specific languages is increasingly being used in distributed systems to capture specific behaviour [67, 68, 69]. This can also potentially help with the reasoning about boundaries between domains, particularly if languages share a common meta-model to support this reasoning. In general, though, there is less work on dealing with boundaries between domains. Lee et al. introduce the concept of a TerraSwarm, describe highly heterogeneous cyber-physical systems where systems must deal dynamically with resources they encounter at run-time [70]. This is though only at the vision stage. The Dionasys project, featured in Section V below, have introduced a generalised approach to the programming of systems of systems, based around the abstraction of holons (with holons meeting in real-time and having to dynamically reason about properties such as interoperability) [71]. Dionescu et al. [72] have also experimented with such a holonic abstraction for the goal-oriented self-management of complex systems. Finally, the ANA project introduces a similar concept called compartments but this only operates at the network level [73].

D. Raising the Level of Abstraction

There has been surprisingly little research on ***raising the level of abstraction*** in middleware. One notable exception is the vertical CORBA Facilities, which are functionalities that are useful to particular vertical application domains, including manufacturing, distributed simulation and accounting. Compared to the rest of the CORBA architecture, this though is rather under-developed and is now quite old.

It is though becoming increasingly important to offer higher levels of abstraction in middleware given the complexity of both the underlying distributed systems and also the applications and services being developed on these complex distributed environments. This is particularly true in the environmental science domain as discussed in Section II where it is essential to provide more abstract interfaces whereby scientists can focus on their science and not on the complexities of the underlying distributed systems.

The work on domain specific languages mentioned above is highly relevant in this context. Domain specific languages alongside associated model-driven engineering techniques have the capability to offer higher levels of abstraction that are more tailored towards given areas of application. An example of using such technologies is given in Section V.

E. Summary

In summary, while there have been some developments in each of these areas, work is generally at an infancy and no-one has looked at all these perspectives in tandem. Taken together, the author argues that ***this offers a manifesto for a new landscape for complex distributed systems and a new kind of middleware designed specifically to address the complexity in such systems***.

To take this further, also implies a new style of working. Historically, the distributed systems community has been quite siloed in its approach, working on principles and techniques in isolation from other disciplines. To address the challenges in this paper, this must change. This paper promotes a more ***holistic approach*** whereby software engineering methodologies and systems principles are developed in tandem. It is too often the case that methodologies, systems platforms, and indeed programming languages are developed in isolation from each other and this is untenable given the complexity of contemporary distributed systems. There are some notable positive examples in this area. The long-running SEAMS workshop (Software Engineering for Self-Adaptive Systems) has also helped to bring the two communities together [74]. Equally, there is a need for more ***cross-disciplinary*** initiatives where complex distributed systems are developed alongside their end users so that we offer solutions that meet their increasingly sophisticated needs.

V. Experiments in Complex Distributed Systems

The following projects have all been selected to illustrate the potential of the four proposed fresh perspective either individually or in combination. All projects have included the author and his research team, in collaboration with others.

A. Connect: An Experiment in Emergent Middleware

1) Motivation

The Connect project was a consortial project funded under the EU Framework 7 programme, under Future and Emerging Technologies (FET) [75, 76]. The project focused on interoperability recognizing the increasing level of heterogeneity in future systems and also the needs to adapt to change. Motivations include the emergence of ubiquitous computing and increasing use of mobile devices.

The project pioneered the concept of emergent middleware, seeking to develop an approach to synthesizing connectors at run-time instead of relying on static middleware to perform this role. Particular attention was paid to ensuring the resultant connectors also met specified security and reliability constraints. The consortium was cross-disciplinary in that it brought together experts in distributed systems, theory, software synthesis, machine learning and dependability.

2) Technical Approach

The overall Connect approach is captured in Fig. 2.

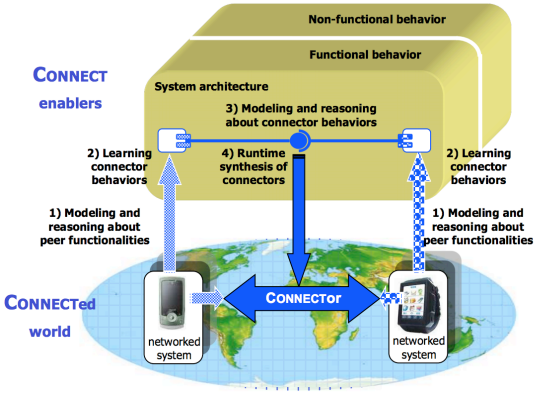

Fig. 2. The overall Connect process, including enablers

In this approach, connectors are developed through a comprehensive dynamic process, which is supported by dedicated middleware functions (referred to as enablers) that:

- "**Extract knowledge** from, **Learn** about, and **Reason** on the interaction behaviour of networked systems, so as to:
- **Synthesize** new interaction behaviours out of the ones exhibited by the systems, and further:
- **Generate** and **deploy** corresponding connector implementations to actually realize interoperability in the involved systems; and
- **Analyze dependability/security** of the realized system at predeployment time and runtime." [75]

Through this approach, Connect generates an appropriate connector dynamically that works for that context with this also being constantly re-evaluated over time. The intention is to have an approach that is better able to manage change, including the emergence of new protocols, standards and modes of interaction.

An underlying middleware technology called Starlink [77] was developed to support the dynamic deployment of connectors. This middleware is based on the concept of k-coloured automata as a concrete model representing the output of the synthesis project, which is the mediation solution designed to overcome the heterogeneity in the networked system. The underlying architecture of Starlink is as shown in Fig. 3.

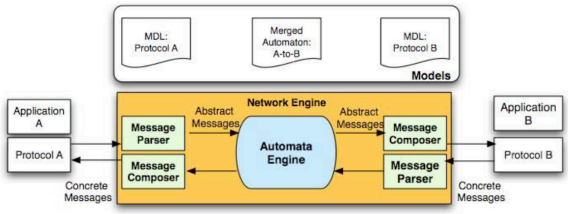

Fig. 3. The Starlink middleware technology

Connect was evaluated through two case studies, one on systems of systems for forest fire management, and another supporting mobile, collaborative working where heterogeneous mobile platforms interact with an increasingly diverse range of applications and services, including cloud services. Further details on Connect can be found on the project webpage: https://www.connect-forever.eu/.

3) Lessons Learned

Connect is important as it pioneered the concept of emergent middleware in the context of interoperability. The project also provided real insights into the underlying technology (enablers) that are required to support an emergent middleware approach, including crucial support for learning and synthesis, alongside a target environment for deployment. The case studies also demonstrated that the approach could be effective in dealing with important problem areas, notably related to extreme heterogeneity. Connect was a long-term research project and clearly significant research problems remain with further work needed in particular in the key enabling technologies including learning and synthesis. It would also be interesting going forward to look at broader areas of distributed systems and also application domains to provide more experience of using this style of emergent middleware in different settings.

B. Emergent Software Systems: The Role of Self-adaptation

1) Motivation

Emergent software systems is a recently completed PhD project by Roberto Rodrigues at Lancaster University. In this work, Emergent Software Systems are defined as "systems built from small and reusable units of software behaviour, and are capable of self-compose and self-optimise as a result of the characteristics of its operating environment" [78]. The

realisation of the concept relies on the brining together of component technology with machine learning techniques. The key motivation for the work is the sheer complexity of contemporary software systems that typically consist of millions of lines of code and operate over complex and potentially highly distributed infrastructures that are also prone to change. The work is therefore closely related to the Connect project but looking more generally at discovery of optimal software architecture at run-time without human intervention.

2) Technical Approach

The implementation work is carried out using Dana, a multi-purpose programming language developed at Lancaster featuring a fine-grained component model and offering run-time support for component configuration and re-configuration [79].

The framework to support Emergent Software Systems then consists of two parts: a local framework supporting the concept on a single machine instance, followed by an extension to allow this to extend to a distributed setting.

The local framework consists of three key modules, namely Perception, Assembly and Learning (PAL), with the overall approach summarized in Fig. 4 below.

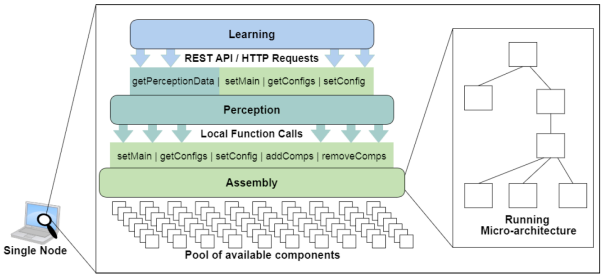

Fig. 4. The PAL Architecture (Perception, Assembly and Learning)

The Assembly module searches for components in an underlying repository of possible component implementations of different functions, creates an in-memory representation of all available architectural compositions the system can be assembled into, and supports composition changes at runtime. The Perception module generates and adds proxy components to the system's architectures to monitor the system health status and the operating environment. Finally, the Learning module leads the overall autonomous design process, based on a reinforcement learning algorithm. This overall approach is referred to in the thesis as design by composition. The approach is then extended to operate in a distributed setting by the introduction of a hierarchical coordination strategy. Further details of this can be found in [78, 80].

The approach has been evaluated through a substantive case study, the development of an emergent web server, that is a web server that can autonomously adapt its own software architecture acceding to current operational conditions. The results from this case study are strongly encouraging, demonstrating that the PAL architecture is capable of successfully configuring and reconfiguring the web server under changing operational environments.

3) Lessons Learned

This project takes the concept of emergent middleware further. Whereas Connect focused on the specific function of connectors to provide interoperability, the emergent software systems project takes this a stage further by seeing to assembly arbitrary software architectures for arbitrary purposes. The successful implementation of an emergent web server has shown that the approach is feasible and that it can work in both local and distributed settings. This is promising research and further work is now required to prove and refine the concept in other areas of application.

C. Dionasys: Programming for Systems of Systems

1) Motivation

The DIONASYS project is a joint initiative of four research institutions (Universities of Neuchâtel, Bordeaux, Lancaster and Technical University of Cluj-Napoca) in four countries, funded by the CHIST-ERA ERA-NET. The goal of DIONASYS is to make the programming of complex and heterogeneous Systems-of-Systems simpler, more straightforward by allowing a higher level of abstraction and allowing advanced features such as automatic adaptation, automatic interoperation, and support of programmable networks for these tasks. The project started in January 2015.

2) Technical Approach

The research carried out in Dionasys is wide-ranging, covering a number of different aspects related to systems-of-systems and full details can be found on the website: http://www.dionasys.eu/. Here, we focus on the core approach to programming systems of systems, namely the use of a holons abstraction to offer a systematic way of composing systems of systems.

Our first principle is to model a given distributed system as a unitary first-class programmatic entity that we call a *holon*, that can be specified, manipulated, and reasoned about in a program; and then to provide programmatic concepts that enable a developer to construct systems of systems through programmatic holon composition. This composition process is intended to be very simple and straightforward, requiring only a few program lines or simple graphical tools. An important aspect of holons is that it allows us to abstract away from the node-level detail and focus on the behaviour of a given system/subsystem and use this as the basis of reasoning when it interacts with other holons. This is therefore a direct realisation of what was discussed in Section IV, in terms of reasoning about given domains and their boundaries, and how they then interact with other domains. As concrete examples, holons can be used to specify how different mobile ad-hoc networks (MANETs) should interact if they meet dynamically. Other examples can be found in Blair et al. [81].

In more detail, a holon is recursive, hierarchical composition of other systems or holons, with holons at the level below referred to as sub-holons The hierarchy bottoms out with leaf holons representing the smallest possible systems in our model. A given holon also has an associated service, that is a specification of the value-added functionality that a holon offers over and above its sub-holons. Holons are subject to both vertical composition and horizontal composition allowing

the construction of arbitrarily complex distributed systems of systems.

Blair et al takes this concept further by presenting a systems architecture for the specification, implementation, management and deployment of systems of systems using holons, with this architecture showing in Fig. 5.

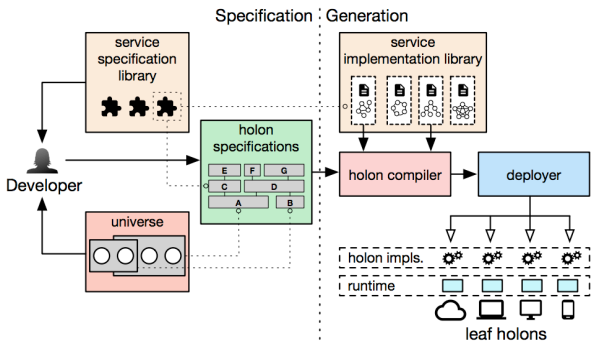

Fig. 5. Supporting holons

3) Lessons Learned

The research in the Dionasys project has shown that is increasingly important to deal with level of complexity of contemporary distributed systems structures and that it is becoming increasingly imperative to provide architectures and approaches that allows us to reason about such systems as systems of systems, abstracting away from detail and also allowing such systems to reason about how they may interoperate if they encounter each other dynamically. Holons represent one potential abstraction that serves this purpose. In comparison to other perspectives introduced in this paper, this is perhaps the most demanding and immature and further research is required to gain experiences of reasoning over complex systems of systems.

D. Models in the Cloud: Raising the Level of Abstraction

1) Motivation

The 'models in the cloud' project is a 3-year EPSRC-funded initiative involving computer scientists and environmental scientists at Lancaster University. The central hypothesis underlying the research is that a combination of model-driven engineering and software frameworks will enable a paradigm shift in terms of the flexible and tailored support offered by cloud computing for given application domains, including the key area of environmental modelling.

Environmental modelling is a large and diverse research field, spanning many areas of environmental management (e.g. weather or climate prediction, flood prediction) and at different scales (e.g. global, national and local). For a given area, there are many models with different assumptions, level of parameterisation, modelling approach and complexity. Some models run on individual workstations while others execute on dedicated supercomputers. In addition, model simulations are often combined as *ensembles*: i) through running the same model multiple times while varying the starting point or assumptions; or ii) through running multiple (distinct) models with different parameters and assumptions but which predict the same output variables. Additionally, different models can be combined in predictive cascades (e.g. cascading climate and hydrology models to project future flooding).

Moving environmental models to the cloud has the potential to revolutionise Environmental Science through supporting a more open, collaborative and integrative approach. This area, however, is in its infancy. In order to achieve out vision of 'models in the cloud', it is necessary to significantly raise the abstraction of the underlying cloud services, to manage the distributed computation and to allow scientists to operate in their domain and express their domain specific knowledge, effectively allowing scientists to do their science rather than spending too much time dealing with low level details of the distributed infrastructure.

2) Technical Approach

The overall approach is as shown in Fig. 6.

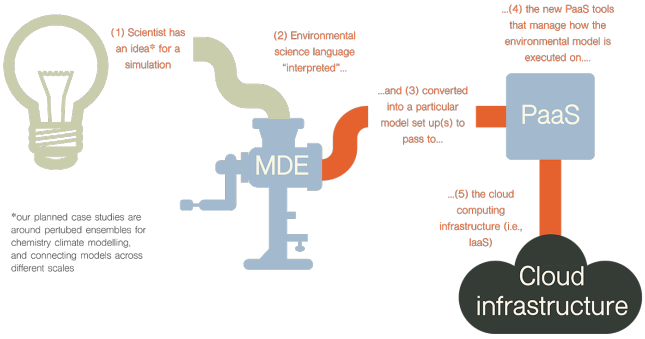

Fig. 6. Models in the cloud – overall approach

This highlights the central role of model driven engineering in supporting the execution of environmental model runs. The initial scientific objectives and experiment are described through a domain specific language. From this, an appropriate model execution setup is generated and passed on to the cloud for execution making use of appropriate software frameworks in the cloud (cf. Platform as a Service).

More specifically, two domain specific languages are used:

- **DSL for scientific experiment.** This DSL captures the scientific intend in terms of the environmental models to be used and associated assumptions. This may be a single model run or could involve arbitrarily complex configurations of models including ensemble models and/or predictive cascades. The experiment may also involve additional elements to reason about uncertainty. The design of this DSL is informed by a series of in-depth interviews with environmental modellers [82].
- **DSL for cloud deployment.** This DSL captures details of how the environmental models should be deployed into the cloud including consideration of mapping to containers and possibly micro-services and also the style and number of virtual machines required. This may also include additional elements to manage the execution, including exploiting the natural elasticity in the cloud.

The first case study, which is nearing completion, provides support for the cloud deployment of the Weather Research and Forecasting model (WRF), a complex atmospheric numerical weather prediction system. This was initially deployed on Microsoft Azure but we are now utilising the container technology, Docker, to provide platform independence.

Current work is looking at the role of machine learning to further raise the level of abstraction of the DSLs. For cloud deployment, this would allow the scientific team to express their desired performance and indeed cost vs. performance trade-offs in terms of goals allowing underlying machine learning modules to determine how to achieve these multi-criteria goals (with this building on previous work by Samreen et al. [83]). For scientific experimentation, machine learning also potentially has a role in determining how a model should be set up in terms of parameterisation, configuration and assumptions to best match the characteristics of a given place of study, cf. Beven's Models of Everywhere philosophy whereby models are trained automatically to best fit a given place or situation of use. Future research will involve additional case studies featuring different environmental models operating at different scales, and also examining in more depth ensemble modelling and model cascades.

3) Lessons Learned

This work is a relatively early stage but the results so far have been strongly encouraging. The research has strongly reinforced our view that we need to raise the level of abstraction of distributed systems technologies and it is particularly striking how long a scientific team will spend on getting environmental models to execute correctly and efficiently on distributed infrastructure. Automating this deployment and execution is therefore a huge step forward for the environmental modelling community, and DSLs have the potential to go significantly further in capturing scientific processes in a more sophisticated manner with this also supporting a step towards reproducibility in science.

E. Overall Reflections

The intention of the four perspectives in this paper is to stimulate thinking about new principles and patterns for distributed systems and subsequently to new styles of middleware platform. The four projects described above provide more concrete glimpses about what this may entail. They all apply one or two of the perspectives (see Fig. 7) but interestingly no one project embraces all four perspectives.

Perspective	Connect	Emergent Software Systems	Dionasys	Models in the cloud
Emergent middleware	✔	✔		
Software frameworks				✔
Domain-specific solutions			✔	
Raising the level of abstraction				✔

Fig. 7. Projects vs. perspectives

This opens the door for researchers to experiment further with the four perspectives and to see what might emerge. Interestingly, all four projects have embraced cross-disciplinary thinking as a fundamental part of their research design and this has contributed to the level of innovation in the projects.

VI. Conclusions

This paper has examined the increasing complexity in distributed systems both in terms of the underlying distributed systems architectures and their areas of application. It has been argued that, while distributed systems have responded successfully to demands over scalability, the extreme level of heterogeneity is proving to be much more challenging. There is also a pressing need to address the fact that we are dealing with distributed systems of systems. This paper argues that the field of distributed systems is at a watershed and that new approaches are urgently needed. The main contribution of the paper is four fresh perspectives on distributed systems, emphasizing: i) the importance of run-time techniques and emergent middleware; ii) the role of software frameworks as meta-strictures; iii) modeling systems as domains and reasoning about boundaries; iv) raising the level of abstraction in distributed systems. The paper concluded with a set of four projects that pick up on one or more of the perspectives, offering insights into what this might mean for platform design. Clearly, this work is at an early stage and the author concludes the paper by calling out to others in the community to collaborate and seek new insights and principles of distributed systems going forward that in turn will lead to a new generation of middleware technologies.

Acknowledgments

The work presented in this paper is partially supported by the following three grants: DT/LWEC Senior Fellowship (awarded to Blair) in the Role of Digital Technology in Understanding, Mitigating and Adapting to Environmental Change, EPSRC: EP/P002285/1; Models in the Cloud: Generative Software Frameworks to Support the Execution of Environmental Models in the Cloud, EPSRC: EP/N027736/1; Declarative and Interoperable Overlay Networks, Applications to Systems of Systems (the Dionasys Project), EPSRC/Chistera: EP/M015734/1; Emergent Connectors for Eternal Software intensive Networked Systems (the Connect Project), EU FP7/FET Proactive grant 231167.

The author would like to thank colleagues in the Ensemble and MetaLab research initiatives at Lancaster for many stimulating discussions and insights on the topic of this paper. He would also like to thank the many people associated with the projects featured in Section V (Connect, Dionasys, Models in the Cloud, and the PhD work of Roberto Rodrigues that have all contributed hugely to the argumentation in this paper. And finally, I would like to thank Maarten van Steen for encouraging me to put together this "vision" paper. It has been a very stimulating and enjoyable experience, drawing on various strands of my recent work. We also thank Microsoft for the Azure for Research award that supported the experimentation with environmental models in the cloud.

REFERENCES

[1] The GEOSS Project, http://www.earthobservations.org/geoss.php.

[2] Helm, D. (2015). Natural Capital - Valuing Our Planet. Yale University Press.

[3] Potschin, M., Haines-Young, R., Fish, R., Kerry Turner, R., Routledge Handbook of Ecosystem Services. Routledge (2016).

[4] Muller, F, de Groot, R, Willemen, L. (2010). Ecosystem services at the landscape scale: the need for integrative approaches. *Landscape Online*. 23, 1-11. DOI:10.3097/LO.201023

[5] Jagadish, H.V., Gehrke, J., Labrinidis, A., Papakonstantinou, Y., Patel, J.M., Ramakrishnan, R., Shahabi, C. (2014). Big data and its technical challenges. *Commun. ACM.* 57:7, 86-94. DOI: https://doi.org/10.1145/2611567.

[6] Laniak, G.F., Olchin, G., Goodall, J., Voinov, A., Hill, M., Glynn, P., Whelan, G., Geller, G., Quinn, N., Blind, M., Peckham, S., Reaney, S., Gaber, N., Kennedy, R., Hughes, A. (2013). Integrated environmental modeling: a vision and roadmap for the future, *Environmental Modelling & Software. 39, 3-23. DOI:* 10.1016/j.envsoft.2012.09.006

[7] Blair, G.S., Henrys, P., Leeson, A., Watkins, J., Eastoe, E., Jarvis, S., Young, P. (2017). Data science of the natural environment: a research roadmap. Unpublished.

[8] Coulouris, G., Dollimore, J., Kindberg, T., Blair, G. (2011). Distributed Systems: Concepts and Design, 5th Edition, Addison-Wesley.

[9] Tanenbaun, A., Van Steen, M. (2007). Distributed Systems: Principles and Paradigms, 2nd Edition, Prentice-Hall.

[10] Birrell, A.D, Nelson, B.J. (1984). Implementing remote procedure calls. ACM Transactions on Computer Systems (ACM TOCS), 2(1), pp. 39-59.

[11] The Open Group, Distributed Computing Environment (DCE): http://www.opengroup.org/dce/

[12] Birman, K.P. (1993). The process group approach to reliable distributed computing. Comm. ACM, 36(12), pp. 36–53.

[13] Vinoski, S. (1997). CORBA: integrating diverse applications within distributed heterogeneous environments. Communications Magazine, IEEE, 35(2), 46-55.

[14] Sessions, R. (1997). COM and DCOM: Microsoft's vision for distributed objects. Wiley.

[15] Szyperski, C. (2002). Component software: beyond object-oriented programming. Pearson.

[16] Coulson, G., Blair, G., Grace, P., Taiani, F., Joolia, A., Lee, K., Ueyama, J., Sivaharan, T. (2008). A generic component model for building systems software. ACM Transactions on Computer Systems (ACM TOCS), 26(1), pp. 1-42.

[17] Blair, G., Coupaye, T., Stefani, J. B. (2009). Component-based architecture: the Fractal initiative. Annals of Telecommunications, 64(1), 1-4.

[18] Emmerich, W., Kaveh, N. (2002). Component technologies: Java beans, COM, CORBA, RMI, EJB and the CORBA component model. In Proceedings of the 24th International Conference on Software Engineering (ICSE'02), pp. 691-692).

[19] Fleury, M., Reverbel, F. (2003). The JBoss extensible server. In Proceedings of the ACM/IFIP/USENIX International Conference on Middleware, Springer-Verlag, pp. 344-373).

[20] Alonso, G., Casati, F., Kuno, H., Machiraju, V. (2004). Web services, Springer.

[21] Dongara, J., et al. (2011).The International Exascale Software Project Roadmap, http://www.exascale.org/mediawiki/images/2/20/IESP-roadmap.pdf

[22] Northrup, L. et al. (2009). Ultra-Large-Scale Systems The Software Challenge of the Future. Software Engineering Institute, Carnegie Mellon. 134 pages.

[23] Jamshidi, M. (Ed.). (2008). Systems of systems engineering: principles and applications. CRC.

[24] Zhang, Q., Cheng, L., Boutaba, R. (2010). Cloud computing: state-of-the-art and research challenges. Journal of internet services and applications, 1(1), 7-18.

[25] Atzori, L., Iera, A., Morabito, G. (2010). The internet of things: A survey. Computer networks, 54(15), pp. 2787-2805.

[26] Forman, G. H., Zahorjan, J. (1994). The challenges of mobile computing. Computer, 27(4), pp. 38-47.

[27] Apache Hadoop: https://hadoop.apache.org/

[28] OpenStack: https://www.openstack.org/

[29] Chandra, T. D., Griesemer, R., Redstone, J. (2007). Paxos made live: an engineering perspective. In Proceedings of the twenty-sixth annual ACM symposium on principles of distributed computing (pp. 398-407).

[30] Dean, J. Ghemawat, S. (2010). MapReduce: a flexible data processing tool. Comm. ACM, 53(1), pp. 72-77.

[31] DeCandia, G., et al. (2007). Dynamo: Amazon's highly available key-value store. In ACM SIGOPS Operating Systems Review, 41(6), pp. 205-220).

[32] Cassandra: http://cassandra.apache.org.

[33] Petcu, D. (2011). Portability and interoperability between clouds: challenges and case study. In Towards a Service-Based Internet, pp. 62-74, Springer.

[34] Petcu, D., Di Nitto, E., Ardagna, D., Solberg, A., Casale, G. (2014). Towards multi-clouds engineering. In IEEE Conf. on Computer Communications (INFOCOM) Workshop), pp. 1-6.

[35] Samreen, F., Blair, G. S., Rowe, M. (2014). Adaptive decision making in multi-cloud management. In Proceedings of the 2nd International Workshop on CrossCloud Systems.

[36] Mottola, L., Picco, G. P. (2011). Programming wireless sensor networks: fundamental concepts and state of the art. ACM Computing Surveys, 43(3), 19pp.

[37] Mottola, L., Picco, G.P. (2012). Middleware for wireless sensor networks: an outlook. Journal of Internet Services and Applications, 3(1), pp. 31-39.

[38] Madden, S. R., Franklin, M. J., Hellerstein, J. M., Hong, W. (2005). TinyDB: an acquisitional query processing system for sensor networks. ACM Trans. on database systems, 30(1), pp. 122-173.

[39] Costa, P., Mottola, L., Murphy, A. L., Picco, G. P. (2006). TeenyLIME: transiently shared tuple space middleware for wireless sensor networks. In Proceedings of the international workshop on Middleware for sensor networks, pp. 43-48. ACM.

[40] Souto, E., Guimarães, G., Vasconcelos, G., Vieira, M., Rosa, N., Ferraz, C., Kelner, J. (2006). Mires: a publish/subscribe middleware for sensor networks. Personal and Ubiquitous Computing, 10(1), pp. 37-44.

[41] Gummadi, R., Gnawali, O., Govindan, R. (2005). Macro-programming wireless sensor networks using Kairos. In Distributed Computing in Sensor Systems, pp. 126-140, Springer.

[42] Hill, J., Szewczyk, R., Woo, A., Hollar, S., Culler, D., Pister, K. (2000). System architecture directions for networked sensors. Proc. of the 9th Int. Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS-IX), pp. 94-104.

[43] Dunkels, A., Gronvall, B., Voigt, T. (2004). Contiki-a lightweight and flexible operating system for tiny networked sensors. In Proc. 29th Annual IEEE International Conference on Local Computer Networks, pp. 455-462.

[44] Akkaya, K., Younis, M. (2005). A survey on routing protocols for wireless sensor networks. Ad hoc networks, 3(3), pp. 325-349.

[45] Teixeira, T., Hachem, S., Issarny, V., Georgantas, N. (2011). Service oriented middleware for the internet of things: a perspective. Towards a Service-Based Internet, pp. 220-229, Springer.

[46] Hughes, D., Thoelen, K., Horré, W., Matthys, N., Cid, J. D., Michiels, S., Joosen, W. (2009). LooCI: a loosely-coupled component infrastructure for networked embedded systems. In Proceedings of the 7th ACM International Conference on Advances in Mobile Computing and Multimedia, pp. 195-203.

[47] Eisenhauer, M., Rosengren, P., Antolin, P. (2010). Hydra: A development platform for integrating wireless devices and sensors into ambient intelligence systems. In The Internet of Things, pp. 367-373. Springer.

[48] Song, Z., Cárdenas, A.A., Masuoka, R. (2010). Semantic middleware for the Internet of Things. In IoT.

[49] Kistler, J. J., Satyanarayanan, M. (1992). Disconnected operation in the Coda file system. ACM Transactions on Computer Systems (TOCS), 10(1), pp. 3-25.

[50] Fiege, L., Gärtner, F. C., Kasten, O., Zeidler, A. (2003). Supporting mobility in content-based publish/subscribe middleware. In Proceedings of the ACM/IFIP/USENIX International Conference on Middleware, pp. 103-122. Springer-Verlag.

[51] Bromberg, Y. D., Issarny, V. (2005). INDISS: Interoperable discovery system for networked services. In Proceedings of the ACM/IFIP/USENIX 2005 international Conference on Middleware, pp. 164-183. Springer-Verlag.

[52] Flores, C., Grace, P., Blair, G. S. (2011). Sedim: A middleware framework for interoperable service discovery in heterogeneous networks. ACM Trans. Autonomous and Adaptive Systems, 6(1).

[53] Capra, L., Emmerich, W., Mascolo, C. (2003). Carisma: Context-aware reflective middleware system for mobile applications. IEEE Trans. Software Engineering, 29(10), pp. 929-945.

[54] Grace, P., Blair, G. S., Samuel, S. (2005). A reflective framework for discovery and interaction in heterogeneous mobile environments. ACM Mobile Computing and Communications Review, 9(1), pp. 2-14.

[55] Luzeaux, D. and Ruault, J.-R. (eds) (2013). Systems of Systems, John Wiley & Sons.

[56] Saltzer, J. H., Reed, D. P., Clark, D. D. (1984). End-to-end arguments in system design. ACM Transactions on Computer Systems (TOCS), 2(4), pp. 277-288.

[57] Grace, P., Bromberg, Y. D., Réveillere, L., Blair, G. (2012). Overstar: An open approach to end-to-end middleware services in systems of systems. In Proceedings of the ACM/IFIP/USENIX Middleware Conference, pp. 229-248, Springer.

[58] Schmidt, D. C., Gokhale, A. S., Schantz, R. E., Loyall, J. P. (2004). Middleware R&D challenges for distributed real-time and embedded systems. SIGBED Review, 1(1), pp. 6-12.

[59] IEEE Systems of Systems Engineering Conference: http://sosengineering.org/

[60] The COMPASS Project: http://www.compass-research.eu/

[61] The DANSE Project: http://www.danse-ip.eu/home/

[62] The Road2Sos Project: http://road2sos-project.eu/

[63] The T-Area-SoS Project: https://www.tareasos.eu/

[64] Johnson, R.E. (1997). Frameworks = (components + patterns). CACM 40(10), pp. 39-42.

[65] Fayad, M., Schmidt, D.C. (1997) Object-oriented application frameworks. Commun. ACM, 40(10), pp. 32-38.

[66] Bertran, B., Bruneau, J., Cassou, D., Loriant, N., Balland, E., Consel, C. (2014). DiaSuite: A tool suite to develop sense/compute/control applications. Science of Computer Programming, 79, pp. 39-51.

[67] Brandtzæg, E., Mohagheghi, P., Mosser, S. (2012). Towards a domain-specific language to deploy applications in the clouds. In Proc. Third International Conference on Cloud Computing, GRIDs, and Virtualization, pp. 213-218.

[68] Bissyandé, T. F., Réveillère, L., Lawall, J. L., Bromberg, Y. D., Muller, G. (2015). Implementing an embedded compiler using program transformation rules. *Software: Practice and Experience*, *45*(2), 177-196.

[69] Burgy, L., Réveillère, L., Lawall, J., Muller, G. (2011). Zebu: A language-based approach for network protocol message processing. IEEE Trans. Software Engineering, 37(4), 575-591.

[70] Lee, E.A., *et al*. (2014). The swarm at the edge of the cloud. Design & Test, IEEE, 31(3), pp. 1-13.

[71] Coulson, G., Blair, G.S., Elkhatib, Y., Mauthe,. A. (2015). The design of a generalised approach to programming systems of systems. To appear: Proc. IEEE WoWMoM.

[72] Diaconescu, A., Frey, S., Müller-Schloer, C., Pitt, J., Tomforde, S. (2016). Goal-oriented holonics for complex system (self-)Integration: concepts and case studies. In: Proc. IEEE 10th Int. Conference on Self-Adaptive and Self-Organizing Systems (SASO'16), 100-109. 10.1109/SASO.2016.16.

[73] Schmid, S., Schuetz, S., Zimmermann, K., Nunzi, G., Brunner, M. (2007) Autonomic and decentralized management of wireless access networks. IEEE Transactions on Network and Service Management, 4(2), pp. 96-106.

[74] SEAMS Workshop Series: https://www.hpi.uni-potsdam.de/giese/public/selfadapt/seams/

[75] Grousset, E., Issarny, V., Bennaceur, A., Bertolino, A., Mulas, D., et al. (2012) Project Final Report Final Publishable Summary Report. Available from: https://hal.inria.fr/hal-00805639.

[76] Blair, G.S., Bennaceur, A., Georgantas, N., Grace, P., Issarny, V., Nundloll, V., Paolucci. M., (2011). The role of ontologies in emergent middleware: supporting interoperability in complex distributed systems. In Proceedings of the 12th International Middleware Conference (Middleware '11). International Federation for Information Processing, Laxenburg, Austria, Austria, 400-419.

[77] Bromberg, Y-D, Grace, P., Réveillère, L. (2011.) Starlink: runtime interoperability between heterogeneous middleware protocols. The 31st International Conference on Distributed Computing Systems (ICDCS 2011), Jun 2011, Minneapolis, United States. 2011.

[78] Rodrigues, R. (2018). Emergent software ssytems. PhD theses, Lancaster University, UK, unpublished.

[79] The Dana Programming Language, http://www.projectdana.com/.

[80] Rodrigues, R., Porter, B. (2017). Defining emergent software using continuous self-assembly, perception, and Learning. ACM Trans. Auton. Adapt. Syst. 12, 3, Article 16 (September 2017), 25 pages. DOI: https://doi.org/10.1145/3092691.

[81] Blair, G., Bromberg, Y-D., Coulson, G., Elkhatib, Y., Réveillère, L., Ribeiro, H.BEtienne Rivière,E., Taïani, F. (2015). Holons: towards a systematic approach to composing systems of systems, In Proceedings of the 14th International Workshop on Adaptive and Reflective Middleware (ARM 2015).

[82] Simm, W., Samreen, F., Ferrario, M.A., Bassett, R., Young, P., Whittle, J., Blair, G.S. (2018). SE in ES: Opportunities for Software Engineering and Cloud Computing in Environmental Science, Unpublished.

[83] Samreen, F., Elkhatib, Y., Rowe, M., Blair, G.S. (2016). Daleel: Simplifying Cloud Instance Selection Using Machine Learning. In Proc. IEEE/IFIP Symposium on Network Operations and Magamenet (NOMS.16).