A novel development paradigm for deploying applications over programmable infrastructure

Panagiotis Gouvas
Ubitech Ltd.
Av. Mesogion 429,
Athens, Greece
pgouvas@ubitech.eu

Anastasios Zafeiropoulos
Ubitech Ltd.
Av. Mesogion 429,
Athens, Greece
azafeiropoulos@ubitech.eu

***Abstract* — Given the inability of application developers to foresee the changes as well as the heterogeneity on the underlying networking and computational infrastructure, it is considerable crucial the design and development of novel software paradigms that facilitate application developers to take advantage of the emerging programmability of the underlying infrastructure and therefore develop reconfigurable by design applications. In parallel, it is crucial to design solutions that are scalable, support high performance, are resilient-to-failure and take into account the conditions in their runtime environment. Towards this direction, this position paper proposes a novel reconfigurable by design applications development paradigm over programmable infrastructure.**

Keywords — programmable infrastructure; re-configuration; application development

I. INTRODUCTION

Taking into account the evolution towards the Future Internet and the challenges for the management of complex and heterogeneous infrastructure along with the efficient provision of a set of services with diverse requirements, it could be strongly argued that the "Future Internet" software will be the critical infrastructure on which all other critical infrastructures will depend [1]. The technological developments that are realized (including developments for realizing the Internet of Services, the Internet of Things, the Mobile Internet, Cloud Computing and other new software delivery platforms) are changing the dynamics of the software industry and posing new requirements and opportunities for improving Europe's competitive position in software.

As stated in [1], in order to provide their full power, software technologies have to be embedded in an object or a device, providing intelligence to the system. A transition from the 'intelligent design' approach, which currently rules software engineering to meta-design approaches as well as self-combining software systems has to be realized. The proposed approaches have to take into account the inability of software engineers to foresee all possible situations that systems, connected to the open physical world, have to face. Thus, focus should be given on the design of software components that have the ability to collaborate in an autonomous and decentralized fashion [1].

Furthermore, the deployment of systems and optimization platforms that take advantage of the emerging programmability of the underlying infrastructure should be fully exploited towards the design of infrastructural-agnostic applications. Key recommendations include the support to the effort that by 2020, software intensive real time systems should be executable on shared hardware and easily connectable to the outside world [1]. As stated in [2], it is envisaged that open programmable networks will enable eco-systems offering a plethora of innovation opportunities for software industry as well as SMEs, since new high-quality services and revenue streams can be introduced more quickly and at lower risk while keeping the network reliable and secure and improving its utilization and operational efficiency.

The above mentioned evolutions and recommendations are considered crucial for handling most of the challenges raised for the design and deployment of distributed applications [1]. These challenges include their deployment over heterogeneous infrastructures, the support of scalability, high performance and resilience to failure characteristics, the support of end-to-end security solutions as well as the need for programmability of the infrastructural resources. Under this perspective, the vision of the proposed approach in this paper is to provide a novel reconfigurable by design distributed applications' development paradigm over programmable Infrastructure (Figure 1).

The proposed approach relies on the development of an extensible context model which will be used by developers directly at the source-code level. Proper context model usage will be assisted and validated by an IDE plugin in order to re-assure that generated binary contains meaningful semantics. The generated executable should be on-boarded by a smart controller which will undertake the tasks of i) translating annotations to optimal infrastructural configuration ii) initializing the optimal configuration to the registered programmable resources and iii) reacting pro-actively to the configuration plan based on the infrastructural state. Such a smart controller will adhere to an open and extensible architecture; in the sense that future programmable infrastructural elements should be smoothly on-boarded. Finally, the context model and the aforementioned toolset will be complemented by a development methodology that will assure that developed distributed applications are reconfigurable by design.

The paper is organized as follows: section two presents the related work on the area along with the progress beyond the state of the art in the proposed approach, section three details

978-0-9836283-9-2

the overall conceptual architecture while section four concludes the paper and refers to future work.

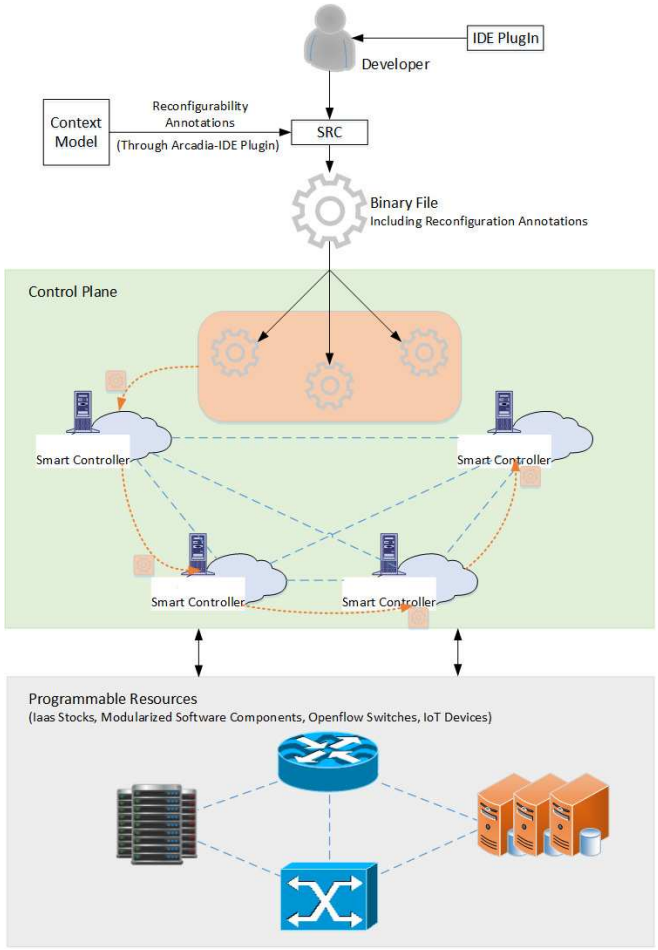

Figure 1: Approach for the deployment of distributed applications over programmable infrastructure

II. Challenges And Related Work

A. *Context Models that target Programmable Resources*

In the area of context models that target programmable resources there are a lot of mature modeling artifacts developed from many projects mainly related to Cloud Computing. Possibly the most notable one is CloudML [11] that is partially funded by the EU commission in the FP7 projects MODAClouds, PaaSage, and REMICS. The scope of CloudML is to manage the complexity of development and administration of multi-cloud systems. To do so, CloudML tames this complexity by providing a domain-specific modeling language along with a run-time environment that facilitates the specification of provisioning, deployment, and adaptation concerns of multi-cloud systems at design-time and their enactment at run-time [11]. Several extensions of CloudML have been proposed. Indicatively, inspired by the OMG Model-Driven Architecture (MDA), MODACLoudML has been proposed [12].

All these modeling artifacts constitute a major input for the proposed context model. However, the proposed context model is also taking under consideration configuration aspects and policies that are related to even programmable hardware resources such as Software Defined Networking (SDN) switching elements, smart routers, firewalls, IoT elements etc. Therefore, the execution engine that accompanies these languages cannot be used. The proposed approach in this paper is clearly not infrastructural-driven. It is driven by the requirements imposed by the distributed applications during run-time.

B. *Theoretical Approaches on Optimal Configuration of Programmable Resources*

One of the most notable research approaches regarding configuration of programmable resources is provided by [13]. The presented work is related to "Model-driven Configuration of Cloud Computing Auto-scaling Infrastructures", while the research outcomes are significant since they show i) how virtual machine configurations can be captured in feature models; ii) they describe how these models can be transformed into Constraint Satisfaction Problems (CSPs) for configuration and cost-specific optimization, iii) they demonstrate how optimal auto-scaling configurations can be derived from these CSPs with a constraint solver, and iv) they present a case-study showing the cost reduction produced.

The proposed approach at this paper has much broader scope than cost-reduction extending the conceptualization of formulating CSPs based on context model annotations. An analogous approach has been performed in [14] that emphasizes on the selection of highly optimal architectural feature sets using Filtered Cartesian Flattening. This approximation technique is utilized for the selection of highly optimal feature sets while adhering to resource limits which is also of high importance for the optimal configuration component of the described Smart Controller.

C. *Theoretical Approaches to Autonomic Prediction of Application's State*

According to [15] there are five classes of quality prediction techniques proposed in the context of Service Based Systems (SBS): i) data mining techniques, which use machine learning algorithms to create prediction models by analyzing historical monitoring data; ii) run-time verification techniques, which check if a set of predefined properties hold during runtime iii) online testing techniques, which test the constituent services of a SBS in parallel to the live operation to collect evidence for failures iv) static analysis techniques, which systematically analyze models of SBS to infer approximations of their execution and v) simulation-based techniques, which execute dynamic models of SBS to simulate their future behavior. It could be argued that the first three fit-in to the proposed approach in this paper.

Concerning data mining techniques, the PREvent framework [3] [4] [5] is a system integrating event-based monitoring, prediction of SLA violations using machine learning and runtime prevention of such violations by triggering adaptation actions in service compositions realized by BPEL processes. The framework uses predictions of SLA violations, which are generated using regression models

generated by analyzing historical monitored runtime data, to trigger adaptations actions in the service composition. In relation to run-time verification techniques, the work presented in [6] uses Discrete Time Markov Chains to develop a mathematical framework for run-time probabilistic model checking that, given a reliability model and a set of requirements, statically generates a set of expressions (rules), which can be efficiently used at run-time to verify system requirements. In [7] there is presented SPADE, an automated technique that addresses failure prevention-based assumptions about the SBS constituent services, derived from service-level agreements. SPADE formally verifies the SBS against its requirements at run-time. PROSA [8] exploits online testing at run-time in order to proactively trigger adaptation, while in [9] there is proposed just-in-time testing of conversational services as a novel approach to detect potential problems and to proactively trigger adaptations, thereby preventing costly compensation activities.

All the techniques that have been discussed above are taken under consideration during the architectural design of the Autonomic Prediction & Reconfiguration Engine of the proposed Smart Controller, as detailed later on in section three.

D. *Network Programmability; from SDN to ADN*

The Software Defined Networking (SDN) paradigm changed radically the notion of network programmability. In a nutshell, there are three primary facets of programmability in software-defined infrastructure: control plane, management plane, and data plane. Each one has a specific purpose and they need not to work in concert. You can benefit from one without the others, depending on what your specific goals with respect to software-defined technologies may be (see Figure 2).

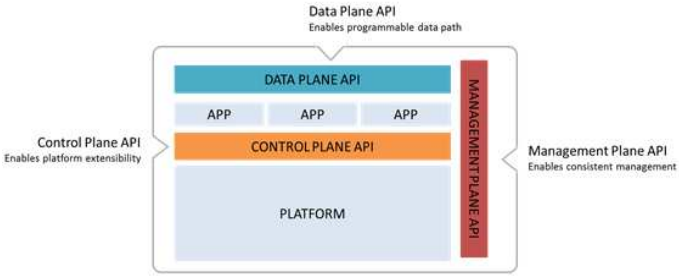

Figure 2: Facets of programmability in SDNs [17]

The management plane is enabled via an exposed API that allows programmatic control of the device/solution/software via external means. The control plane is what enables extensibility. By using this API, the platform (in SDN land this would be the SDN controller, in the Application Development Networking (ADN) world it would be the application delivery controller) can be extended to include new functions, protocols, and services. The data plane focuses on pushing rules that define behavior to a commoditized switch fabric.

The proposed approach at this paper is taking under consideration all the SDN planes during the formulation of the programmable resource abstraction layer. However, it could be argued that the most novel aspect of network programmability is related to the Application Defined Networking (ADN) since SDN is not designed for layers 4-7 and its focus on layer 2-3 – and specifically its packet-processing focus – has long been shown to be inadequate for managing application traffic at layer 4 and above [16]. ADN, on the other hand, is concerned with the routing and switching of applications and the forwarding of sessions. Its model is a mirror of that of SDN, but at the application layers (4-7 of the OSI model) [16]. ADNs will impose additional complexity which is taken under consideration during the formulation of the proposed conceptual architecture in section three.

E. *Industrial Approaches to Infrastructural Configuration*

The proliferation of virtualization coupled with the increasing power of industry-standard servers and the availability of cloud computing has led to a significant uptick in the number of servers that need to be managed. This is where infrastructural orchestration and configuration management tools come into play. The most notable industrial frameworks that meet these demands are Puppet, Chef, Ansible and Salt. Each one of these has its advantages and disadvantages. An excellent comparison report can be found in [10]. The industrial approaches that target automation in the management of vast infrastructures are of extreme importance since such frameworks highlight the industrial-driven aspect of resource programmability. As explained in section three, the proposed technical approach is supporting auto-generation of scripts that are compatible with one of the aforementioned frameworks.

III. Conceptual Architecture

As already explained, the vision of the proposed approach is to provide a novel reconfigurable by design distributed applications' development paradigm over programmable infrastructure. To do so, it relies on specific modelling artefacts, a concrete toolset and an applications development methodology. The overall proposed conceptual architecture is presented on Figure 3.

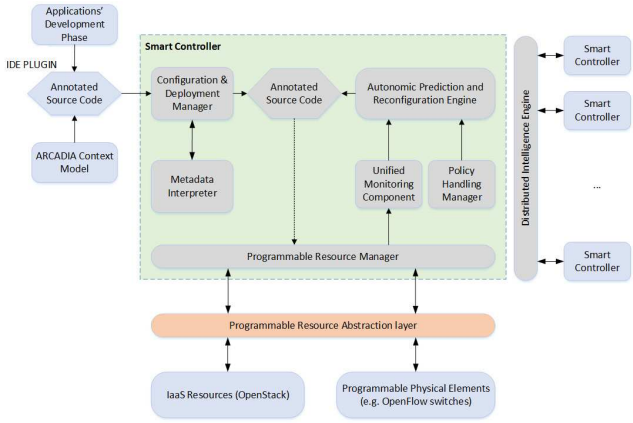

Figure 3: Conceptual architecture for the deployment of distributed applications over programmable infrastructure

According to the flow depicted in Figure 3, a developer is using an IDE in order to create a distributed application. We assume that he/she is using, at the source level, proper development architectural patterns in order for the application

to be potentially distributed under specific configuration. Failing to comply with the distributed applications development patterns will impose a theoretical barrier to the developed applications which cannot be raised or confronted by any infrastructural configuration.

Considering that a developer complies with distributed applications development patterns, he/she will use an IDE plugin in order to provide specific annotation at the source-level. These annotations will be instantiations of the proposed context model. As analyzed below, this model will conceptualize application, configuration and infrastructural requirements that should be taken under consideration in order for an optimal infrastructural configuration to be defined. According to the conceptual flow, this optimal configuration should be created and instantiated by the component which is called smart controller. This component is (among others) the distributed application's on-boarding utility which will undertake the tasks of i) translating annotations to optimal infrastructural configuration (through the configuration manager) ii) initializing the optimal configuration to the registered programmable resources (through the deployment manager) and iii) reacting pro-actively to the configuration plan based on the infrastructural state and the distributed application's state (through the autonomic prediction and reconfiguration engine). At this point we should point out that, special care has to be put in order for the smart controller to adhere to an open and extensible architecture; in the sense that future programmable infrastructural elements should be smoothly on-boarded.

In the following subsections, we provide a short analysis of the fundamental proposed artefacts; namely the context model, the smart controller and the methodology for the development of distributed applications.

A. *Context Model & Model Usage*

The proposed context model aims to conceptualize dynamic configuration and programmability aspects of the underlying resources. Therefore, the designed context model is going to be multi-faceted and extensible. Multi-faceting is necessary since requirements of dynamic configuration and programmability cannot be represented in a single-axis. Indicatively, modelling artefacts may span from application specific aspects such as clustering, fail-over to pure infrastructural-specific aspects such as security policies, QoS, even energy efficiency policies. An indicative high-level overview of the context model facets are presented on Figure 4.

Regarding the model-usage, one of the most crucial architectural decisions is that the incorporation of context model semantics will be exploited directly at the distributed applications' source code level. The proposed framework is envisaged to make use of SoTA Language-explicit mechanisms that accompany third-generation Object Oriented programming languages (such as Java and C#) which permit the creation of custom extensible metadata that are compiled along the application's source code. In other words, the context model may be maintained in two formalisms; i) one formalism will be development-language-agnostic and ii) another will be binded to a specific programming language through the creation of language specific annotations.

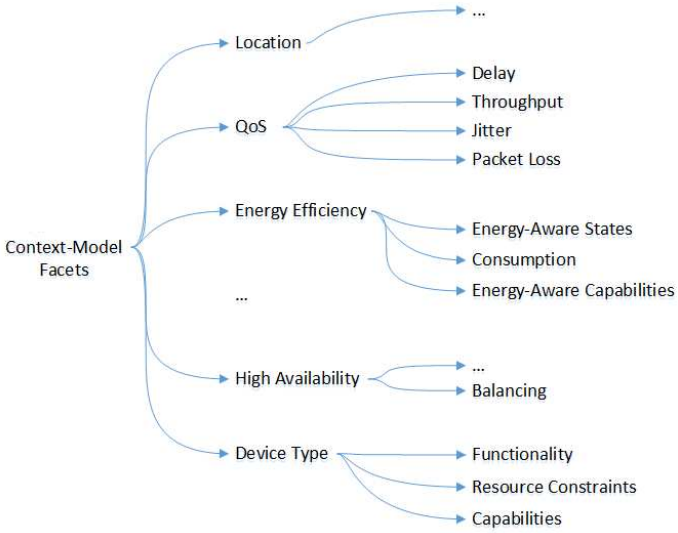

Figure 4: Indicative Context Model Facets

Exploiting context model semantics directly in the source code is an architectural decision with notable advantages. More specifically:

- the notion of distributed applications portability radically increases since there is no need of external deployment descriptors that should accompany the executable files that are hosted in the application server;
- a distributed application becomes extensible; yet run-time-agnostic since an application server can either take under consideration the extensibility semantics or totally ignore them. This strategy depends on the interpretation libraries that accompany the execution environment
- a-priori elimination of vendor-lock-in issues since competitive reference implementations of an interpretation library can be provided by many independent software vendors targeting the same context model.

B. *The Smart Controller*

The proposed approach aims to radically change the way that distributed applications are developed, on-boarded, deployed and managed. After the compilation of a distributed application component, the executable will be on-boarded to the smart controller which is the cornerstone of the proposed approach. Architecturally, its main sub-modules include:

a) a metadata interpreter which is able to interpret metadata that accompany compiled executables. The interpretation is crucial because it affects the deployment and management lifecycle. The interpretation will guide the smart controller's behavior during the entire lifecycle of the distributed application.

b) a programmable resource manager that exposes a specific interface where programmable resources are registered and managed. Programmable resources can span from

configured IaaS frameworks, programmable physical switching/routing equipment, programmable firewalls, application servers, modularized software entities (databases, HTTP Proxies etc.).

c) a configuration and deployment manager which undertakes the complex task of mapping source-code metadata to configuration and infrastructural requirements. These requirements have to be 'translated' in optimal configuration taking under consideration i) the registered resources and ii) their programmability capabilities. The goal of this module is to automate the selection and configuration of resources through the generation of industry-level configuration such as Puppet-scripts, Chef-scripts or Ansible-scripts. The challenges that have to be tackled by this module are many and diverse. Indicatively, the fact that i) registered resources are not static; iii) distributed applications run competitively to each-other (when hosted to same execution container) as far as their required resources is concerned and iii) required resources vary according to time and utilization, indicate that advanced theoretical techniques have to be used in order to end-up in an optimal and sustainable configuration.

d) an unified monitoring manager which will rely on proper probes that will be configured during the deployment phase. Probing is related to active monitoring techniques. Such techniques can be used to monitor in a near-real-time fashion metrics in multiple levels e.g. OS-level (memory, cores etc.), application server level (connection pools, queues, etc.) or application level (heap, stack etc.). However, the unified monitoring manager will also employ passive techniques in order to aggregate measurements from resources that cannot be probed; yet they can be interfaced through their custom API. Indicative examples are switching and routing devices, firewalls etc. Metrics that are measured using both techniques will be aggregated and used from the "autonomic prediction and reconfiguration engine" (analyzed below).

e) an autonomic prediction and reconfiguration engine which is responsible for pro-active adjustment of the running configuration based on measurements that derive from the unified monitoring manager. The ultimate goals of this component are two: i) zero-service disruption ii) re-assure optimal configuration across time. In order to achieve both of these goals predictive algorithms have to be employed, which will eliminate, as much as possible, false-positives, regarding triggering of re-configuration.

f) a distributed intelligence engine that undertakes the task of communicating in a peer-to-peer manner with other smart controllers in order to cope with limitations, barriers or run-time problems that are caused by many reasons such as limited availability of physical resources, run-time performance issues etc. This engine radically augments the usability of the smart controller since it provides a programmable overlay scheme which spans to even geographically distributed locations. This scheme, must act as a virtual aggregation point of geographically dispersed programmable resources; hence providing distributed intelligence. However, distributed intelligence is accompanied by many challenges that have to be confronted. Indicative ones include security and trust among smart controllers, distributed inference and decision making, lack of centralized control etc.

g) a policy handling manager that is responsible for handling or generating triggering events. Such events can be generated through the interpretation of the proposed context model. These policies are by definition multidisciplinary; since they can be affected by many requirements. For example, a performance or QoS requirement may trigger an infrastructure reconfiguration event or a specific policy may prevent the transfer of the executable or its binded persistency units to a specific country.

C. *Distributed Application's Lifecycle*

As already explained, the vision of the proposed approach is to provide a holistic framework that will allow developers to create distributed applications that are reconfigurable by design. To do so, the generation of modelling artefacts, a development-assistance toolkit (IDE plugins) and a reference implementation of a smart controller engine are considered necessary. Without the adoption of the proposed approach, many manual and error-prone configurations have to be accomplished in order to do an optimal configuration. On the other hand, if the executable is "injected" with proper annotations (i.e. instances of the proposed context model), the smart controller will undertake optimal configuration of registered programmable resources. Furthermore, if the existing resources do not-fit (before initial deployment) or have been exhausted (or failed) the smart controller will rely on the distributed intelligence capabilities in order to cope with optimization management. The only penalty that is 'paid' in order to achieve this level of automation is the modelling overhead implying that proper annotations should be placed in proper parts of the code (e.g. before methods, classes, objects).

IV. Conclusions and Future Work

As already thoroughly discussed, the development, configuration and operation of distributed applications entail many and multi-faceted challenges. These challenges span across the entire applications' lifecycle; namely the software engineering, the infrastructural on-boarding, the execution and the optimization phase. Under this perspective, in this position paper, we have presented a novel reconfigurable by design distributed applications' development paradigm over programmable infrastructure. This approach, upon proper instantiation and application to diverse domains, is envisaged to leverage the re-configurability aspects of distributed applications and lead to the specification of a flexible and scalable framework for developing and deploying distributed applications over programmable and re-configurable infrastructure. In our future work, we plan to instantiate the proposed approach in diverse domains (e.g. energy efficient networking mechanisms, security mechanisms) and evaluate the overall performance achieved in comparison with existing approaches.

References

[1] "Toward a Strategic Agenda for Software Technologies in Europe", Information Society Technologies Advisory Group (ISTAG), July 2012,

Available Online: http://cordis.europa.eu/fp7/ict/docs/istag-soft-tech-wgreport2012.pdf, Access: April 2014

[2] NESSI Strategic Research and Innovation Agenda (SRIA) - April 2013, Available Online: http://www.nessi-europe.com/Files/Private/NESSI_SRIA_Final.pdf, Access: April 2014

[3] Leitner, P., Wetzstein, B., Rosenberg, F., Michlmayr, A., Dustdar, S., & Leymann, F. (2009). Runtime prediction of service level agreement violations for composite services. In Proceedings of the 2009 International Conference on Service-oriented Computing (pp. 176–186). Berlin, Heidelberg: Springer-Verlag.

[4] Leitner, P., Michlmayr, A., Rosenberg, F., Dustdar, S. (2010). Monitoring, Prediction and Prevention of SLA Violations in Composite Services. In Proceedings of the 2010 IEEE International Conference on Web Services (ICWS '10). pp. 369–376. IEEE, 2010.

[5] Leitner, P. (2011). Preventing SLA Violations in Composed Services using Self-Adaptation (PhD Thesis). University of Stuttgart.

[6] Filieri, A., Ghezzi, C., Tamburrelli, G. (2011). Run-time efficient probabilistic model checking. In Proceedings of the 33rd International Conference on Software Engineering. pp. 341–350. ACM, New York, NY, USA.

[7] Schmieders, E., Metzger, A. (2011). Preventing performance violations of service compositions using assumption-based run-time verification. Towards a Service-Based Internet. pp. 194–205. Springer.

[8] Hielscher, J., Kazhamiakin, R., Metzger, A., & Pistore, M. (2008). A Framework for Proactive Self-adaptation of Service-Based Applications Based on Online Testing. In Proceedings of the 1st European Conference on Towards a Service-Based Internet (ServiceWave '08) (Vol. 5377, pp. 122–133). Berlin, Springer Berlin Heidelberg.

[9] Dranidis, D., Metzger, A., & Kourtesis, D. (2010). Enabling Proactive Adaptation through Just-in-Time Testing of Conversational Services. In E. Nitto & R. Yahyapour (Eds.), Towards a Service-Based Internet (Vol. 6481, pp. 63–75). Berlin, Springer Berlin Heidelberg.

[10] Review: Puppet vs. Chef vs. Ansible vs. Salt, Available Online: http://www.infoworld.com/d/data-center/review-puppet-vs-chef-vs-ansible-vs-salt-231308

[11] Brandtzæg, E., Parastoo, M. Mosser, S. (2012). Towards a Domain-Specific Language to Deploy Applications in the Clouds. In Cloud computing 2012: 3rd international conference on cloud computing, grids, and virtualization, pages 213-218. IARIA.

[12] Ferry, N., Rossini, A., Chauvel, F., Morin, B., Solberg, A. (2013). Towards model-driven provisioning, deployment, monitoring, and adaptation of multi-cloud systems. In Lisa O'Conner, editor, Cloud 2013: ieee 6th international conference on cloud computing, pages 887-894. IEEE Computer Society.

[13] Dougherty, B., White, J., Schmidt, D.C. Model-driven Configuration of Cloud Computing Auto-scaling Infrastructure, Available Online: http://www.cse.wustl.edu/~schmidt/PDF/Models-Dougherty.pdf

[14] White, J., Dougherty, B., Schmidt, D.C. (2009). Selecting highly optimal architectural feature sets with Filtered Cartesian Flattening. J. Syst. Softw. 82, 8 (August 2009), 1268-1284.

[15] Metzger, A., Pohl, K. (2009). Towards the Next Generation of Service-Based Systems: The S-Cube Research Framework. CAiSE 2009: 11-16

[16] SDN is Network Control. ADN is Application Control. Available Online: https://devcentral.f5.com/articles/sdn-is-network-control-adn-is-application-control#.U5aeZfmSy7I

[17] The Three Faces of Programmability in Software-Defined Infrastructure. Available Online: https://devcentral.f5.com/articles/the-three-faces-of-programmability-in-software-defined-infrastructure#.U5afPvmSy7J