Analysing the Lifecycle of Future Autonomous Cloud Applications

Geir Horn[1], Keith Jeffery[2], Jörg Domaschka[3] and Lutz Schubert[3]
[1]*University of Oslo, Olav Dahls hus, P.O. Box 1080, 0316 Oslo, Norway*
[2]*Keith G Jeffery Consultants, 10 Claypits Lane, Shrivenham, U.K.*
[3]*University of Ulm, 89069 Ulm, Germany*

Keywords: Cloud Computing, Software Design, Autonomy, Adaptability.

Abstract: Though Cloud Computing has found considerable uptake and usage, the amount of expertise, methodologies and tools for efficient development of in particular distributed Cloud applications is still comparatively little. This is mostly due to the fact that all our methodologies and approaches focus on single users, even single processors, let alone active sharing of information. Within this paper we elaborate which kind of information is missing from the current methodologies and how such information could principally be exploited to improve resource utilisation, quality of service and reduce development time and effort.

1 INTRODUCTION

Clouds are on the rise. A major problem is heterogeneity of offered platforms and their interfaces with consequent inability to port applications. Traditional methodologies and software engineering principles still dominate the commercial software industry. These principles were generated for single-user and typically single-processor use, i.e. not for cloud concepts.

This paper argues the necessity to involve additional skills beyond traditional software development in order to master successfully the Cloud deployment of an application. We describe the application aspects that need to be provided as a profile by the application owner; the characteristics of Cloud platforms that are supplied by the platform operator; the characteristics of the data used by the application and the characteristics of the user input must be provided by the user as a profile partly modifiable for each individual execution of the application. This information is metadata to be used by the systems development process.

The main contribution of this paper is to identify the data that must be considered for an effective deployment, and show that the data requires continuous monitoring to unlock the full benefits offered by the elasticity and scalability of the Cloud. This implies that the profiles may need continuous monitoring and essentially that the application owner needs to be aware of this application lifecycle. Section 2 highlights the core aspects of application deployment, and Section 3 identifies the data needed for the deployment and the data profiles.

The results presented here base on the initial work in the PaaSage project, and the architecture for a system to support model driven autonomic deployment and application adaptation is described in Section 4. Although the PaaSage project is working on implementing the necessary building blocks, it is too early to report on the implementation of the application lifecycle in this paper.

2 ASPECTS OF APPLICATION DEPLOYMENT

Current cloud applications are designed and developed in much the same way as traditional applications. The developer / provider only starts thinking about the cloud specific characteristics once he starts to deploy, respectively host the application.

In order to understand why traditional engineering principles are insufficient and how they need to change, it is necessary to analyse which kind of factors play a role at executing the main Cloud characteristics, including specifically:

1. sharing data (and computation)
2. replication and elasticity
3. (re)location and distribution

With traditional software engineering, the application would be designed to essentially host one complete instance for a single user. Data sharing has to be explicitly encoded and is not represented on the level of actual execution state, let alone that it automatically caters for consistency problems. To make best use of resources (and thus to optimise cost and quality of service), it is furthermore necessary to assess the scaling behaviour of the individual functionalities that compose the full application logic.

Figure 1 depicts such varying scaling behaviour of a simple online book store: following traditional principles would result in a single workflow per each user, including the services and data base. However, information such as the database needs to be available to all users, but the store front and the publisher access are not necessary per user. However, no current software engineering principle allows for description of a behaviour as depicted.

The implication of this scaling behaviour, however, has to be assessed on the overarching level, as for example two individual services in the application workflow may scale out beyond the total cost agreements, simply because the behaviour of the individual services did not cater for the total impact.

We will elaborate the impacting characteristics in more detail in the following sections:

2.1 Usage Aspects

There is a general tendency to treat a Cloud application as a self-sustained application in a full-blown virtual image, though an increasing number of modern cloud applications are actually complex business processes distributed over multiple resources and multiple virtual images and potentially integrating the capabilities of different Cloud infrastructures at the same time, cf. (Schubert and Jeffery, 2012). As has been shown, the intrinsic characteristics of such applications determine the scaling behaviour, and in particular its constraints, and thus the optimal resource utilisation, see e.g. (Becker, Koziolek and Reussner, 2009), (Rubio Bonilla, Schubert and Wesner, 2012) (cf. Figure 1).

Not only the number of users (which may vary significantly over time) impacts on application scale, but also the usage behaviour of each individual user. Already a comparatively simple application, such as book selling exposes multiple functionalities: searching for books, adding / removing from lists, ordering from publisher etc. – all triggering different processes and necessitating different data.

It may be safely assumed that the application structure, namely its processes and their potential relationship, is already known. This – structural – information serves also one further purpose: the relationship between processes (usually as services) and their potential impact on execution criteria such as performance, data consumption etc.:

Each service contributes differently to the overall

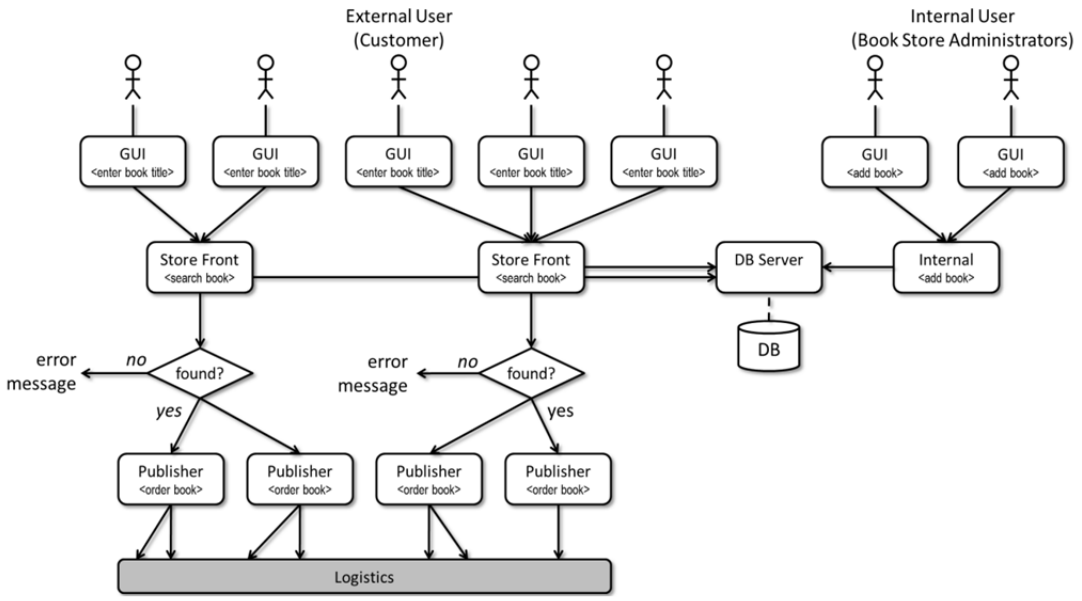

Figure 1: Potential distributed workflow with different overlapping processes and different scaling behaviour per service instance.

application workflow and hence exposes different (a) resource requirements and (b) connections during scaling. For example an individual instance of service A may serve up to 9(!) instances of service B provided that B only contributes 1/10 to the total load. The multiple instance of B may be the consequence of scaling out due to increased user demand.

2.2 Performance Aspects

The application workflow information can indicate problem areas, such as bottlenecks. In order to assess these, standard relationship information needs to be extended with data about the communication frequency and width (average size and range size of messages), as well as performance impact per delay.

It will be noted that the communication impact is not only defined by the data exchange, but logically by the data in use itself. As with any application, data size and structure can have strong impact on overall performance. Non- or semi-structured data has particular characteristics when used for searching, pattern detection etc. and streamed data has even more difficult characteristics requiring windowing of the stream and multiple complex parameters being maintained to assure effective and efficient management of the data. Restructuring for efficiency improvement may prove difficult to achieve in a Cloud environment especially if the data is shared.

Even if performance plays a secondary role in the context of cloud computing so far, *quality* of the provided services is a major concern. In this context, availability (uptime and response time) is crucial for customer satisfaction: if a specific service is not available at request time, it will stall the execution of the total application workflow. Scaling out to reach the desired quality of service would mean: (1) higher resource usage and thus higher cost, as well as (2) higher risk of bottlenecks and potential problems that cannot be compensated by scale out.

Also the technical characteristics of the hosting environment plays a major role in meeting the quality of service: for example, the database system may not be suited for the type of access requirements by the application, or storage may be insufficient etc.

2.3 Deployment Criteria Overview

We can thus isolate the following main “external” criteria that (should) influence the behaviour of a cloud application, but are not accessible to the execution environment in any form, so that it must be explicitly provided by the application owner (or the Cloud provider):

(1) *Number of service consumers (users)*: increased number of users will increase traffic and potential resource conflicts associated with access to threads of code, data etc.

(2) *Communication behaviour:* the time and size of data exchange is a critical performance factor and may lead to conflicts if it coincides with a large number of users or background processes

(3) *Usage behaviour*: process invocation frequency and sequence may lead to different resource usage and potentially conflicts

(4) *Workflow:* specifies the application’s structure and thus potential bottlenecks, scaling conditions etc.

(5) *Data structure*: classify the impact arising from typical data types and structures related to the execution of the application

(6) *Quality constraints* (provider, host, consumer) specify the actual business constraints and goals of the actors and must therefore be respected throughout the whole application lifecycle.

(7) *Technical characteristics*: define the performance of a given application with respect to its explicit and implicit resource need. This is especially true for specialised platforms.

Up until now, the developer and potentially the application provider had to cater for these aspects implicitly by generating the necessary configuration information manually, respectively by incorporating the according behaviour directly into the application code or workflow. Not only do most developers lack such expertise, but it also makes the code difficult to adjust to new environments and usage conditions.

3 CONCRETE DEPLOYMENT INFORMATION

The information identified in the previous section must not only be captured in a form that is intuitive and meaningful – more importantly, it must be specified in a form that can be used for deployment and execution control purposes. Therefore this information must be provided in a common format that allows combination, selection and reasoning over it, so as to select the right rules that apply under the given circumstances.

In general, this means that the specifications must consider the following aspects:

- *Conditions:* under which circumstances does the information apply. For example, given a generic rule to scale out when the number of users increase, when does this behaviour really apply? When would it create a bottleneck? The main problem with these conditions is that they are generally not well-defined and cannot be represented as simple scalar tests. Hence they must be able to cater for probabilistic values.
- *Actions* to perform when the conditions are met, e.g., to scale out by a certain number, to relocate the service etc. The composed actions of all services must thereby meet the overall quality constraints. This means that actions may vary between use cases.
- *Events:* conditions are only evaluated if certain events occur. In most cases these events will be triggered by environmental circumstances. Like conditions, they will have to deal with uncertainty in order to allow for delays and predict likely occurrences.

Even though these three aspects are clearly related to the Event-Condition-Action definition commonly used in logic reasoning (Helmer, Poulovassilis and Xhafa, 2011), there are some substantial differences, as will be elaborated in section 4.

3.1 User Profile

The *user profile* describes the way in which the end-user (application end user, application developer or administrator) interacts with the Cloud environment. The expected parameters are required for access control and security – however, other user characteristics are also of interest for better interaction, such as preferred interaction mode, preferred language, screen set-up etc. It should be noted that the user profile information can change dynamically, e.g. access control as the application accesses different (parts of) datasets or databases.

3.2 Application Profile

An application can be generalised to a set of modules exchanging data. The modules can be functions, objects, actors, processes, services etc. The *application profile* captures the characteristics of these modules and their relationships.

The application developer typically has limited knowledge about the temporal aspects of the application, e.g. how much data will be exchanged among the modules at what time, CPU and memory consumption of each module etc. These factors may depend on the data processed (see data profile) and the user behaviour (user profile). Such parameters must therefore be deduced from past executions and from the software structure and its deployment.
Due to the nature of these properties, they are mostly stochastic. In order to evaluate conditions on such parameters, one will have to resort to hypothesis testing, or observation of summarising quantities. For properties with high variability, such as CPU load, a high sampling frequency will be necessary to capture the variations and the large data volumes resulting might not be possible to store. Hence, one will have to resort to generalisations and fitting of standard parameterised density functions to capture the essential variability.

Finally, the application profile must encompass the operational goals, policies, and preferences of the application owner. These are not constraints by themselves, though they may be linked to constraints. Policies can, for example, put restrictions on where data is stored, or which Cloud providers can (not) be used for deployment – these could also indicate preferences of one Cloud provider over others.

The specification of constraints and goals is difficult – there is a clear need for a rich language to capture all aspects, that is tractable using knowledge engineering techniques but which in addition is able to capture the intent of the user.

3.3 Data Profile

The *data profile* characterises the data to be used by the application – this includes size (number and size of records), the kind of dataset (alphanumeric, file etc.), degree of dynamicity, privacy and security considerations. Even more important for Cloud performance are the access characteristics: do applications typically read the dataset sequentially or randomly based on some key attribute value? Is the data used concurrently? Does the data need to be kept consistent? Finally backup mechanisms need to be in place for system recovery.

3.4 Host Profile (Infrastructure Characteristics)

The technical characteristics of the infrastructure play a major role with two respects: (1) whether the host is suitable for the technical requirements and (2) what performance can be expected by using the respective host. The relationship between performance and executing platform is however generally undefinable, see e.g. (Skinner and Kramer, 2005). Similar to the technical requirements (see below), there are however indicators that help in performing an “educated guess” about the right matching, such as bandwidth between nodes, size of storage, database type etc.

Some of these factors can be influenced by the host, such as the number of virtual machines, other factors depend completely on the application and the data processed by it.

The developer should therefore be able to specify all the characteristics that are of essence to executing the application. Typically, this will be constrained to low-level aspects, such as the type of operating system, necessary libraries etc., as the average developer will not be able to specify all performance impacting factors. On the other hand, the developer is the only person able to assess what kind of communication density to expect etc.

Therefore the relevant information must be conveyed in a way that is intuitive and yet meaningful for deployment and configuration. This means that the developer will have to provide (a) the concrete technological deployment requirements (*deployment model*) and (b) any performance related information that compensate the *application profile* – this means in particular information regarding likelihood and category of occurrence. For example, the developer indicates how likely specific data sets occur, such as frequency of address changes versus person queries in a personnel database.

3.5 Expertise

As noted earlier (see Section 2.3), the translation of all the expectations towards the application is currently achieved through manual labour, i.e. by the expertise of the developers involved in the process. However, the scope of information to be considered may easily exceed any current developer’s expertise.

The problem is thereby not so much the complexity of the individual knowledge “items”, but the sheer scope of such expertise and the according fine-grained application context. For example

```
Scale out with numbers of users
```

is fairly easy to grasp and apply in general. However, correct application depends on a number of factors, starting with the networking capacity of the host, over its adaptation speed, up to the overall quality expectations. But there are also commercial aspects to be considered, such as maximum costs and cost to quality ratio.

What is more, however, is that the combination of rules for different services within an application can lead to undesired side effects, if not properly respected when applying these rules. As such, two connected components that communicate with each other may e.g. lead to mutual scale up beyond given boundaries.

As already noted in the beginning of this section, all rules must generally take an event-condition-action (ECA) like structure in order to be applicable under the right circumstances. As opposed to the other information types, which generally are of the form of conveying only parts of the ECA rules (such as the conditions), the “expertise rules” directly incorporate all information and should therefore be formalised as ECAs. In principle, “expertise rules” are the results of gathering all the historical information, generating the rules and improving them over time – with the difference, that such knowledge is already available for exploitation, though not properly formalised for automated usage and application across domains and use cases.

4 AN ARCHITECTURE FOR APPLICATION DEPLOYMENT

As noted, the rules and information to be applied to the actual execution of an application take an event-condition-action (ECA) like form – however:

(1) many of these rules have to be combined under *uncertain and partial unspecified* conditions, and

(2) the conditions will take the form of *mathematical equations* with domains of applicable values, rather than concrete Boolean formulas

4.1 Contextualising Rules

Normal ECA rules apply to strictly logical expressions that evaluate to true or false, similar to

```
if (10 < #users < 100) then
```

However, in the case here, the actual conditions (and also the events) are of a form that is closer to

`if (p(invocation`$_{processA}$`)>0.7) then`

where *p* means the probability for invocation. In other words, the rules apply when uncertain boundary conditions are met. This can be for example

Rule#1: scale out whenever the number of users (per instance) is bigger than 10

Rule#2: only scale out, when the service is not a bottleneck

Rule#3: service B has high communication needs when process A is invoked (bottleneck)

Rule#4: probability for invoking process A is 0.25

In other words, rule#1 applies successfully for service B in 75% of all cases.

To derive such rules and reason over them, the information described (section 3) must be formalised and provided to the executing environment.

4.1.1 Exploiting Specialisation

It will be noted that the rules in this form are per se infrastructure agnostic – meaning in particular, that they generalise over the potential cloud behaviour. At this level, the rules can be executed on any platform that supports the necessary actions.

Some providers may however not offer full support for all of this information, e.g. not all platform providers allow fully detailed monitoring information and not all allow full control over scaling behaviour. The implication of this is, that not all of the rules are equally applicable to all environments.

By exploiting the *host profiles*, however, it can be easily identified which provider supports which rules. Implicitly, the combination of selected rules constraints the selection of providers.

Vice versa, and more interestingly, the same principle holds true to exploit specialised capabilities: these form nothing but action rules that only apply to a dedicated platform or host. Thus, these rules restrict the choice of providers, if the constraints demand for the according rule set. Hence, the probability of fulfilment may alter by selecting an alternative host.

This principle allows full exploitation of specialisation according to the abstract requirements of the application (see above).

4.2 Conceptual Architecture

Self-adaptive software systems have been studied extensively ever since the vision of autonomic computing arose (Jeffrey O. Kephart and David M. Chess 2003). This vision basically specified that the management of any adaptive system would have to carry out four essential processes: (1) the system must be monitored, (2) the gathered information must be analysed to decide what the current system state is and to decide if an adaptation is necessary, after which the adaptation is (3) planned before it is (4) executed. This has become known as the MAPE-K loop of autonomic systems, where the K stands for the knowledge available and exploited executing this loop. The MAPE-K loop must also be enacted for autonomous cloud applications.

One could build the adaptation plan reactively taking only the current application context into consideration when defining the next model and then

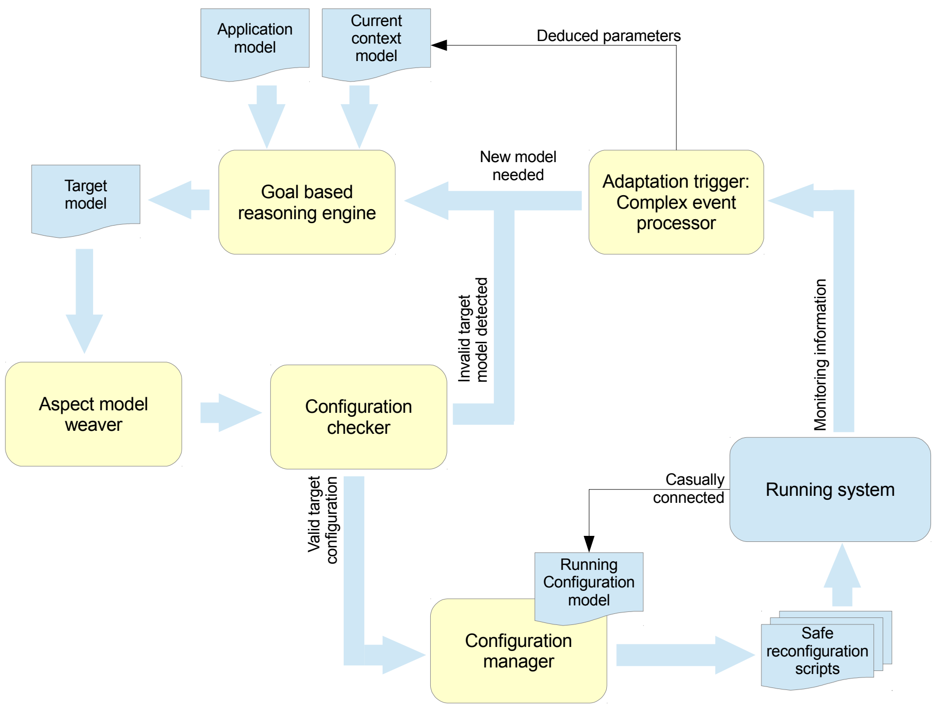

Figure 2: The fundamental modules for adapting a model-based application at runtime. Platform specific scripts ensure a safe transition from the current to the new application configuration. Taken from (Geir Horn 2013).

plan the adaptation as transformation from the running application model to the new one. This approach, known as models@run.time (Gordon Blair, Nelly Bencomo, and Robert B. France 2009) was used in (Brice Morin et al. 2009), which identified the following five modules necessary in order to carry out the MAPE-K loop:

(1) A *complex event processor* to identify when the running context of the application has changed to the extent that an adaptation is necessary.

(2) A *goal based reasoning engine* to select the model configuration that would be "best" for the current context, i.e. the target model.

(3) An *aspect model weaver* to realise the target model and its bindings.

(4) A *configuration checker* to verify that the target model is consistent and deployable.

(5) A *configuration manager* to derive a safe sequence of application reconfiguration commands.

The relations of these modules are depicted in Figure 2, which also advocates the need for a stochastic goal based reasoning engine.

4.2.1 Monitoring

Monitoring is done by the execution platform (inside the virtual machine or on system level). The monitor provides information about the current execution status, insofar as supported by the host (cf. section 4.1.1). Therefore the monitor provides relevant information whether the expected constraints are met.

Each monitoring instance fulfils a two-fold role (cf. next section): (1) checking the performance of the individual service (instance), and (2) contributing to the analysis of the overall behaviour.

A definition of a monitor incorporates (1) what type of information needs to be gathered (2) at what frequency and (3) potentially how it needs to be generated (metric), as well as (4) where it needs to be sent to (such as the event filter).

Obviously, the information is constrained by the hosting platform's capabilities - in other words, no metrics can be applied to data that is not provided by the host. This is hence a selection criteria for the right host (see section 4.1.1).

4.2.2 Analysis

The analysis of the monitored information can be done locally or centrally: for instance local monitoring can decide to scale out a local service, but application level adaptation must be analysed centrally to weigh the effect of all individual actions.

The simplest form of analysis is to compare the monitored data against a threshold. However, this will be vulnerable to variations. For instance, the response time of a database query depends on its complexity - hence, a complex query may exceed the response threshold. Adapting on a single event may therefore lead to erratic behaviour.

Statistical hypothesis testing is one alternative. However, one should bear in mind that most methods are developed for the linear model (Franklin A. Graybill, 1976) and therefore sensitive to deviations from the normal distribution of large samples. The alternative is to use non-parametric tests (Conover, 1999) or, if one is just interested in detecting out of bounds events, one could use the sample version of the Chebyshev inequality (John G. Saw, 1984).

Finally, there are numerous techniques from the field of information flow processing that can be applied to correlate and understand the true system state based on monitored events (Gianpaolo Cugola and Alessandro Margara, 2012).

4.2.3 Planning

Adaptation planning takes place at three levels:

(1) The developer plans adaptation steps that make sense for the application logic – for example related to the workload assignment in distributed applications.

(2) Plans for each Cloud provider adaptation using the elasticity mechanisms offered by that infrastructure. Such plans are provided as part of the deployment of the application, and they are therefore a part of the currently deployed configuration.

(3) When the system requires adaptation that is not catered for, a new configuration is needed. Such a global adaptation must be planned in response to the detected changes in the application's running context using models@run.time as outlined above, trying to meet all operational constraints (Geir Horn 2013).

4.2.4 Execution

The adaptation plan should be executed hierarchically with the central authority maintaining overall application consistency, whereas platform level mechanisms are used to the largest possible extent to allow a smooth transition.

The configuration manager will need to identify the difference between the running and the new configuration, to generate reconfiguration scripts for each Cloud platform. This is also highly use case dependent, as some applications allow checkpointing and state recovery, whereas other applications must simply be stopped with the new configuration started from scratch.

The main point thereby is that not the services themselves are adapted (in the sense of re-programmed or re-configured with new parameters), but the execution context instead. Using above building blocks, the profiles and service descriptions, the tasks (i.e. services) can be treated individually as black boxes that are scaled out, relocated, reconfigured much the same way a virtual image would. In other words, the services contributing to the overarching application logic are basically subject to common cloud strategies that together meet the overarching goals and constraints.

To realise such behaviour, the configuration manager is closely linked to local execution engines that perform the actual adaptation on a service level, ensuring that the individual steps in the adaptation script are executed correctly.

5 CONCLUSIONS

Our proposed model aims at integration across multiple Cloud platforms of any kind. It will also allow the application to be deployed optimally taking account of the specialised characteristics of different platforms matched to the requirements of the application and its usage constraints.

Not all additional information necessary can always be provided, or properly matched: so far, no proper programming and modelling mechanism exists that allows easy and intuitive definition of the right type of information. Furthermore, reactive adaptation planning using models@run.time is an active research topic (Svein Hallsteinsen et al. 2012), and using these concepts for Cloud deployment is currently under investigation.

A major open challenge thereby remains in maintaining the compositional correctness of the decomposed rules and actions: during deployment and adaptation, the overarching constraints have to be broken down to low-level rules that can be enacted individually. To this end, the information does have to be provided in a fashion that incorporates both high- and low-level descriptions.

This paper described the approach pursued in the PaaSage project, which develops the necessary language, modelling and reasoning tools to allow provisioning and exploitation of the type of information described here. The tools will allow the individual stakeholders to provider their respective view on the goals and constraints by building up from proven patterns that can be decomposed through according reasoning mechanisms. The goal is to make it easier to create and host applications that can run effectively and efficiently on various Cloud, thereby addressing a major barrier to take-up of Clouds.

ACKNOWLEDGEMENTS

The research leading to these results has received funding from the European Union Seventh Framework Programme (FP7/2007-2013) under grant agreement n° 317715. The views expressed in this paper are those of the authors and do not necessarily represent those of the consortium.

REFERENCES

Becker, S., Koziolek, H. & Reussner, R., 2009. The Palladio component model for model-driven performance prediction. *Journal of Systems and Software, 82:3-22.*

Brice Morin, Olivier Barais, Jean-Marc Jézéquel, Franck Fleurey, and Arnor Solberg. 2009. 'Models@run.time to Support Dynamic Adaptation'. *Computer* 42 (10): 44–51. doi:10.1109/MC.2009.327.

Conover, W. J.., 1999. *Practical Nonparametric Statistics*. 3rd ed. Wiley Series in Probability and Statistics: Applied Probability and Statistics Section. John Wiley & Sons.

Franklin A. Graybill. 1976. *Theory and Application of the Linear Model*. North Scituate, MA, USA: Duxbury Press.

Geir Horn. 2013. 'A Vision for a Stochastic Reasoner for Autonomic Cloud Deployment'. In *Proceedings of the Second Nordic Symposium on Cloud Computing & Internet Technologies*, edited by Muhammad Ali Babar, Marlon Dumas, and Arnor Solberg, 46–53. NordiCloud '13. New York, NY, USA: ACM. doi:10.1145/2513534.2513543. http://doi.acm.org/10.1145/2513534.2513543.

Gianpaolo Cugola, and Alessandro Margara. 2012. 'Processing Flows of Information: From Data Stream to Complex Event Processing'. *ACM Comput. Surv.* 44 (3) (June): 15:1–15:62. doi:10.1145/2187671.2187677.

Gordon Blair, Nelly Bencomo, and Robert B. France. 2009. 'Models@run.time'. *Computer* 42 (10): 22–27. doi:10.1109/MC.2009.326.

Hallsteinsen, S., Geihs, K., Paspallis, N., Eliassen, F., Horn, G., Lorenzo, J., Mamelli, A., and Papadopoulos, G. A., 2012. A Development Framework and Methodology for Self-adapting Applications in Ubiquitous Computing Environments. *Journal of Systems and Software* 85 (12) (December): 2840–

2859. doi:10.1016/j.jss.2012.07.052.

Helmer, S., Poulovassilis, A. & Xhafa, F., 2011. Reasoning in Event-Based Distributed Systems. *Studies in Computational Intelligence 347*, Springer-Verlag Berlin Heidelburg.

Jeffrey O. Kephart, and David M. Chess. 2003. 'The Vision of Autonomic Computing'. *Computer* 36 (1): 41–50. doi:10.1109/MC.2003.1160055.

John G. Saw, Mark C. K. Yang, and Tse Chin Mo. 1984. 'Chebyshev Inequality with Estimated Mean and Variance'. *The American Statistician* 38 (2) (May): 130. doi:10.2307/2683249.

Rubio Bonilla, D., Schubert, L. and Wesner, S., 2012, The Need to Comprehend Clouds: Why We Still Can't Use Clouds Properly. *CloudComp Conference Proceedings*.

Schubert, L., & Jeffery, K., 2012. Advances in Clouds - Research in Future Cloud Computing. Cordis (Online). Brussels, BE: European Commission. Retrieved from http://cordis.europa.eu/fp7/ict/ssai/docs/future-cc-2may-finalreport-experts.pdf.

Skinner, D. & Kramer, W., 2005. Understanding the causes of performance variability in HPC workloads. *International Symposium on Workload Characterization*.