Governing Elastic IoT Cloud Systems under Uncertainty

Stefan Nastic, Georgiana Copil, Hong-Linh Truong, and Schahram Dustdar
Distributed Systems Group, TU Wien, Austria
Email: {*nastic,copil,truong,dustdar*}@*dsg.tuwien.ac.at*

***Abstract*—Emerging IoT cloud systems create unified IoT cloud infrastructures that offer large pools of elastic resources, which need to be governed through their entire lifecycle. However, numerous uncertainties are inherently present in such infrastructures, mainly due to the novel interactions of IoT elements, network elements, cloud resources and humans. They pose a plethora of challenges for the governance of such IoT cloud systems. In this paper we introduce U-GovOps – a novel framework for dynamic, on-demand governance of elastic IoT cloud systems under uncertainty. We introduce a declarative policy language to simplify the development of uncertainty- and elasticity-aware governance strategies. Based on that we develop runtime mechanisms, which enable mitigating the uncertainties by monitoring and governing the IoT cloud systems through specified strategies. We evaluate our approach using a real-life case study in the domain of predictive maintenance.**

I. Introduction

Emerging IoT systems extend contemporary cloud systems beyond the data centers to include a variety of edge IoT devices, such as sensors and sensory gateways. These systems are referred to as elastic IoT cloud systems [1]. On the one hand, such systems utilize the IoT infrastructure resources to deliver novel value-added services by leveraging data from different sensor devices and enabling timely propagation of the essential business decisions to the edge of the infrastructure. On the other hand, these systems utilize cloud resources, e.g., compute and storage services, to support functions of resource constrained IoT devices as well as to enable elastic delivery and consumption of the vast IoT resources through the cloud computing on-demand pay-per-use model. This has proliferated *unified IoT cloud infrastructures* [2]–[7], which comprise large pools of IoT and cloud resources resources from data centers and the edge of the network.

For the above-mentioned elastic IoT cloud systems, governance strategies are a useful mechanism to address issues related to risk mitigation, compliance and legal requirements [8]. However, supporting these strategies poses a plethora of challenges mostly due to numerous uncertainties inherently present in the systems infrastructure. For example, uncertainties related to state monitoring, data delivery and performance variability (e.g., due to probe failures, network issues or human error) often lead to imperfect data about the infrastructure. As a result, infrastructural information needed for governance operations might be incomplete or inaccurate, thus hindering the operational tasks of both automated management systems and the end users. Such uncertainties are mainly caused by the novel interactions of IoT elements, network elements, cloud resources and humans. This calls for rethinking the traditional infrastructure management approaches to include uncertainty considerations in operations and governance strategies.

In this paper we introduce the U-GovOps framework for dynamic, on-demand governance of elastic IoT cloud systems under uncertainty. In our previous work, we introduced GovOps methodology [8] and a supporting framework [9] to effectively manage runtime governance in IoT cloud systems. However, our previous work, as well as many other approaches (e.g., [2], [5]), implicitly assume perfect information (e.g., about IoT cloud resource states), and reliable and deterministic behavior of the IoT cloud infrastructure. Unfortunately, due to the infrastructure uncertainties, such assumptions are unrealistic and impossible to meet in practice, thus putting a lot of burden on the developers and operations managers (users) to deal with such uncertainties, in ad hoc fashion. The U-GovOps framework conceptually extends and technically refines our initial approach by introducing novel techniques to facilitate *developing and executing* the governance strategies under presence of the uncertainty. The main contributions of our framework include: i) A *declarative policy language* for developing uncertainty- and elasticity-aware governance strategies for IoT cloud systems. ii) *Runtime mechanisms and uncertainty mitigation techniques*, which support execution of such strategies under uncertainty.

The remainder of the paper is structured as follows: Section II presents a motivating scenario and research challenges; Section III outlines the design of the U-GovOps framework; In Section IV, we present U-GovOps declarative policy language and its most important runtime mechanisms; Section V describes the current prototype implementation and presents experiments; Section VI discusses related work; Finally, Section VII concludes the paper and gives an outlook of our future research.

II. Motivation

A. Scenario

Let us consider a scenario of Predictive Maintenance Application (PMA) in building management. Figure 1 shows the high-level architecture of the PMA in which data collected by sensors monitoring equipment in buildings is pre-processed locally and relayed via software-defined gateways [1], [7] to the cloud, where various services store and analyze the data to monitor and predict the status of equipment. The arrows show the typical propagation of the sensory data and the analytic sequences within PMA. The actuation sequences are omitted for readability. The PMA relies on a system blending IoT elements and cloud services that is implemented with different technologies[1].

In our scenario, PMA runs atop a complex IoT cloud infrastructure that includes (i) various edge devices, such as,

[1] We provided one implementation of such a high-level architecture at https://github.com/tuwiendsg/DaaSM2M/wiki/

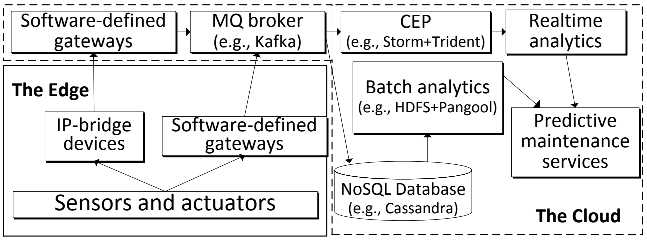
Software-defined gateways
MQ broker (e.g., Kafka)
CEP (e.g., Storm+Trident)
Realtime analytics
The Edge
IP-bridge devices
Software-defined gateways
Batch analytics (e.g., HDFS+Pangool)
Predictive maintenance services
NoSQL Database (e.g., Cassandra)
Sensors and actuators
The Cloud

Fig. 1. Architectural overview of Predictive Maintenance Application (PMA).

(software and hardware) sensors, actuators and gateways, (ii) network elements, and (iii) cloud services, e.g., for complex event processing, NoSQL data storage, and streaming data analysis. All of these infrastructural elements need to be governed throughout their entire lifecycle. As shown in [8], [9], this can be done by designing suitable governance strategies. However, numerous uncertainties interfere with the execution of such strategies, making the implementation of even rudimentary governance strategies a challenging task.

Typically, the PMA polls diagnostic data from equipment, such as chiller plants (e.g., with CoAP), but for optimization purposes (mainly network consumption) not all available sensory data are polled from the cloud. However, in situations such as an emergency or multiple devices failure, the PMA needs to change its operation to be in accordance with company's legal regulatory compliance, e.g., to handle status updates in (near) real time. To satisfy such governance objective a maintenance manager could create a governance strategy which "activates" all available sensors, changes the communication protocol to push-based, e.g., MQTT, and sets the sensors update rate to maximum. Finally, after such situations have been dealt with, the PMA should return to its normal operation mode. In such situations, we need to rely on up-to-date and highly accurate infrastructural state information and stable performance of control actions of various resources to adjust the IoT cloud systems. However, in real world it is hard, if not impossible to achieve them. Therefore, to deal with such situations, on the one hand, we need to capture different types of uncertainties related to state information and performance variability of underlying resources to allow for strategies specified for different uncertainties. On the other hand, we need to develop runtime mechanisms to support these governance strategies under such uncertainties.

B. Research Challenges

1) Uncertainties in the infrastructure of IoT cloud systems: Inspired by the traditional fault, error, failure classification [10] and the general belief model [11], in our work we have identified different uncertainties for IoT cloud infrastructure. To systematically document uncertainties, we have developed a taxonomy and use this taxonomy to classify the uncertainties and analyze their effects on the typical governance strategies[2].

Our taxonomy mainly focuses on infrastructure uncertainties that originate at runtime. Other uncertainties such as design- or requirements-level uncertainties [12] are currently not considered. Figure 2 gives a high-level overview of the taxonomy and its main concepts (uncertainty classes) which are used to classify the infrastructure-level uncertainties: i) *Temporal manifestation* reflects the duration of the uncertain (or failure) state caused by an uncertainty. ii) *Nonfunctional dimensionality* captures affected nonfunctional properties of the infrastructure. For example, the uncertainties can affect well-known infrastructure's dependability [10] , quality of sensory data, or regulatory compliance [8]. iii) *Cause of* uncertainty can be a natural phenomenon, a human action or a technological phenomenon (anything caused by an infrastructure phenomenon, which is beyond user's control). iv) *Effect propagation* denotes whether an uncertainty affects the application or the physical environment. v) *Locality* describes where an uncertainty occurs. We differentiate between uncertainties present in hardware, platform (virtual infrastructure) or external to infrastructure.vi) *Functional dimensionality* denotes the category of infrastructure's functionality that is affected by an uncertainty, e.g., execution environment, actuation, data delivery and storage facilities.

Our taxonomy enables the users (e.g., maintenance manager) to capture their knowledge about potential uncertainties, in a systematic and structured way, based on the set of general, well-defined concepts. Besides the common elements used in the governance strategies, e.g., runtime monitoring information, enabling the users to embed such knowledge in the governance strategies is crucial for development of the strategies that can "cope with" the runtime uncertainties. For our following discussion, we focus on two most relevant uncertainty families[2], which affect the tasks performed in typical governance strategies namely *DataQualityUncertainties* and *ActuationDependabilityUncertainties*. We elaborate the main governance challenges caused by these uncertainties.

2) Main challenges of IoT cloud governance under uncertainty: One of the main tasks of governance strategies is to identify a *governance scope* [9]. Governance scopes represent a set of IoT cloud resources that should be governed by such strategies. The resources are selected and assigned to the governance scopes based on their properties, i.e, the governance scopes are specified as composite predicates referencing the resource attributes. Such attributes mainly reference resource's meta data (mainly specified by humans) and resource's profile data (mainly based on sensory readings) that are used to compute the governance scopes, as we have thoroughly discussed in [9]. However, in practice, the DataQualityUncertainties often lead to *incomplete or missing data* about resources and their states in IoT cloud systems, such as null attribute values (e.g., due to monitoring failures or human error). These quality of data problems make it very challenging to determine the governance scopes. Currently, the users deal with such imperfect information in ad hoc fashion, e.g., by writing complex queries or developing sophisticated probabilistic models. This pollutes the governance logic with uncertainty management, making the governance

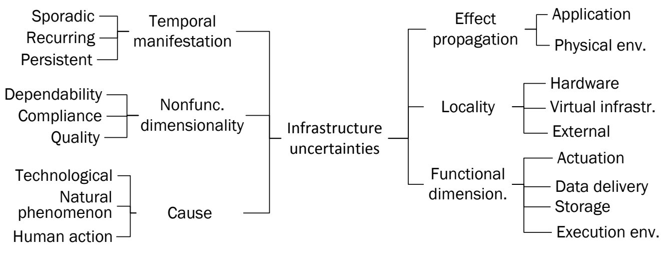

Fig. 2. Taxonomy for IoT cloud infrastructure uncertainties.

[2] The description of the taxonomy is out of the scope of this paper. A detailed description of the uncertainty taxonomy is provided as supplement material at: http://dsg.tuwien.ac.at/staff/snastic/public/u-taxonomy.pdf.

strategies difficult to maintain, less traceable and significantly increasing the development effort.

Another key task performed by governance strategies is (remote) invocation of the *governance actuations* [9], e.g., to increase sensors update rate, as well as the *elasticity actuations* [13], e.g., to keep the cloud services' response time within the specified limits, when the sensors update rate is increased. Such actuations are often subject to the ActuationDependabilityUncertainties, which degrade dependability of the actuation facilities (e.g., due to network latency, device failure or race conditions), manifesting itself often transparently to the users, as lost actuations, cascading failures or resource over-consumption. This usually causes an inconsistent realization of governance strategies or even renders them completely useless, thus causing breaches of regulations or compliance.

III. The U-GovOps framework

A. *Managing uncertainties in governance strategies*

It is known that uncertainty is tightly related to the lack of knowledge [14]. Further, it strongly depends on the task-at-hand and on the system setup and environment [15]. Therefore, by categorizing the uncertainties, analyzing their *Effects* and measuring the degree of sensitivity to such uncertainties (in our taxonomy captured with *Nonfunctional dimensionality*), we can formulate more precise statements such as: "An uncertainty X affects the *application dependability* by causing *resource over-consumption* and potentially leads to a *complete functionality failure* of actuation facilities". This allows for streamlining the uncertainty management by enabling us to derive requirements, actions and configuration models needed for dealing with uncertainties. The main aim of our U-GovOps framework is to facilitate the runtime governance of elastic IoT cloud systems under presence of uncertainty by incorporating such requirements and configurations from the early stages of strategy design. To this end, U-GovOps supports the users to design elasticity- and uncertainty-aware governance strategies.

While governance strategies are mainly used to address issues related to risk mitigation, compliance and legal requirements [8], it is often useful to incorporate elasticity actuations in the governance strategies to enable the users to also anticipate changes in resource demand, costs and quality of the governed systems (encompassing both IoT and cloud infrastructures). For example, by considering elasticity relationships [1], while designing the elasticity-aware governance strategies, users can anticipate increases in resource demand, e.g., since they know that some other governance actions increase the sensors' update rate. They can utilize this knowledge, for instance to "warm up" VMs in order to mitigate the uncertainties related to the actuation delays (of spinning up the VMs) when scaling out related cloud services. However, due to an intrinsic "bootstrapping problem" this only facilitates uncertainty management to a certain extent, since the mechanisms underpinning the governance and elasticity actuations are also subject to uncertainty (Section II). For this reason the U-GovOps framework allows the users to incorporate uncertainty considerations in governance strategies, effectively raising the awareness level of such strategies. To this end, U-GovOps defines a *governance policy language* for developing uncertainty- and elasticity-aware governance policies (Section IV) and provides a language runtime (*Governance and Elasticity Controllers*) that does most of the "heavy lifting" to support executing governance policies, without explicitly worrying about the infrastructure uncertainties.

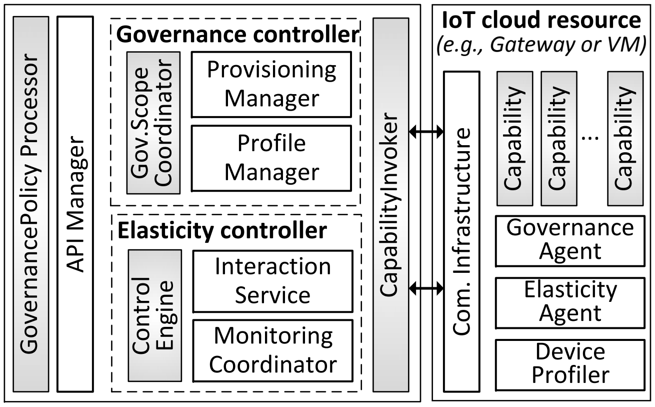

Fig. 3. Overview of U-GovOps architecture.

B. *U-GovOps architecture*

The U-GovOps framework is distributed across the clouds and IoT devices. It is designed based on the microservices architecture that enables evolvable and fault-tolerant system design, while allowing for flexible management and scaling of individual components. In U-GovOps, the *GovernancePolicyProcessor* (Figure 3) represents a central point of interaction with the *Governance Controller* and the *Elasticity Controller*, i.e., it is responsible to interpret the user-provided governance policies (strategies), described latter in Section IV and map them to the controllers API, exposed by the *API Manager*.

The *Governance Controller* comprises several microservices, the most important being the *GovernanceScopeCoordinator*. It provides mechanisms to define and manage the governance scopes, in order to determine which IoT cloud resources will be affected by a governance strategy. It relies on the *ProfileManager* to dynamically build and manage resource (e.g., device) profiles. This involves managing static device meta-information and performing profiling actions in order to obtain runtime snapshots of current device states. The *Elasticity Controller* [13] provides general mechanisms to handle elasticity actuations specified in governance policies. Its main microservices include: The *ControlEngine*, which implements the elasticity control algorithms, e.g., greedy planning; The *MonitoringCoordinator* that is used to integrate infrastructure monitoring frameworks such as Nagios or Ganglia and; The *InteractionService*, which encapsulates higher-level control mechanism, e.g., exposed by an IaaS provider.

These controllers rely on the *CapabilityInvoker* to perform the actual invocations of the underlying capabilities. i.e., perform actuations on the IoT cloud resources over the network (denoted as two-way arrows in Figure 3). For this purpose the framework uses the *ElasticityAgent*, the *GovernanceAgent* and the *DeviceProfiler*, which are responsible to manage local governance and elasticity capabilities and to expose them to the controllers. They are very light-weight components that run in all IoT cloud resources that are managed by U-GovOps.

IV. Developing uncertainty- and elasticity-aware governance strategies

A. *U-GovOps declarative policy language*

In order to facilitate governing IoT cloud systems under uncertainty, the U-GovOps framework provides a declarative policy language for developing uncertainty- and elasticity-aware governance policies. It is based on our previously developed SYBL [13] and its main aim is to support developers and operations managers (users) to design such policies on a higher-level of abstraction, without explicitly dealing with IoT cloud infrastructure uncertainties. Two main tasks that users perform are identifying the governance scopes and defining the governance and elasticity actuations, to be applied on such scopes (Section II). In our language, `STRATEGY` directive allows the specification of the governance or elasticity actuations to be undertaken (e.g., set sensor update rate) or desired behavior to be enforced (e.g., maximize throughput) when specific conditions are met. Further, to declare the governance scopes – determining which resources should be affected by such actuations – our language offers `GOVERNANCE_SCOPE` directive. Finally, to support the users to articulate their knowledge about the uncertainties, i.e., to raise the level of awareness in governance policies, the U-GovOps language provides the `CONSIDERING_UNCERTAINTY` construct. It is mainly used to specify configuration directives for determining the behavior of the governance scopes and governance or elasticity actuations.

Subsequently, we describe the most important language concepts and supporting runtime mechanisms in more detail[3], mainly focusing on: 1) Rough governance scopes and 2) Isolated (governance and elasticity) actuations. In the rest of the paper we mostly focus on describing our framework's support for managing the uncertainties related to the *actuation dependability* and *incomplete and missing data* about IoT cloud systems, which were identified as most relevant for our work.

1) Rough governance scopes: In order to support governance policy developers to deal with the uncertainties related to missing and incomplete data (Section II), the U-GovOps framework introduces a new concept called *rough governance scope*. Generally, a rough governance scope represents a formal approximation of a resource set, taking into account resources, which due to uncertainty, cannot be positively (i.e. with absolute certainty) characterized as members of the targeted governance scope. Rough governance scopes are modeled based on the rough set theory, which unlike fuzzy sets or probabilistic models, has an advantage of providing an objective formal approximation of membership relation [16]. Practically, this means that even with no user involvement, U-GovOps can make an objective approximation of resource assignments to governance scopes under data uncertainty.

Formally, a rough governance scope is defined as a tuple $\langle \underline{G}X, \overline{G}X \rangle$, where $\underline{G}X$ and $\overline{G}X$ are traditional (crisp) sets that represent lower and upper approximation in the rough governance scope, given the set of attributes G. The G-lower approximation is the union of all equivalence classes in $[x]_G$ that are a subset of the targeted governance scope X: $\underline{G}X = \bigcup\{x \mid [x]_G \subseteq X\}$. The G-upper approximation is the union of all equivalence classes in $[x]_G$ which have non-empty intersection with the targeted governance scope X: $\overline{G}X = \bigcup\{x \mid [x]_P \cap X \neq \emptyset\}$ [16]. Therefore, $\underline{G}X$ represents a positive (or pessimistic) approximation and $\overline{G}X$ represents an optimistic approximation of the targeted governance scope.

```
G:GOVERNANCE_SCOPE
 query:= location=buildingX & type=JACE-545
 CONSIDERING_UNCERTAINTY:
  missing_data = "location<='?',type<='*'" AND
  selection_strategy = optimistic AND
  use_cache = false
```

Listing 1. Example governance scope.

Listing 1 shows an example governance scope defined with U-GovOps policy language. In our language, a governance scope is specified as composite predicates referencing device meta information and profile attributes within the `query` parameter. To specify the behavior of governance scopes under data uncertainty users provide additional directives within the `CONSIDERING_UNCERTAINTY` construct. The `selection_strategy` parameter can take values: *optimistic, pessimistic* or *reduct*. It instructs the framework on how to treat the resources belonging to the boundary region ($\overline{G}X - \underline{G}X$), which due to uncertainty (e.g., incomplete attribute set) cannot be positively characterized as members of the governance scope. For example, selecting the optimistic strategy means that U-GovOps will compute the governance scope based on the upper ($\overline{G}X$) approximation. This behavior might be desirable when a governance policy can tolerate false positives, but it must not have any false negatives included in the governance scope. With this knowledge the framework can compute an objective approximation of the governance scope, even if the governed resources are indiscernible with the available attributes in G. More details about the underlying mechanisms are provided in Section IV-B1.

However, to be able to handle the missing data, the governance scope membership relation must be refined with a subjective extension. To this end U-GovOps utilizes the concepts of characteristic relations and characteristic sets [17]. Essentially, this enables the users to specify how the missing data should be interpreted. The `missing_data` directive enables the users to generally define interpretation of the missing attribute values as "do not know" [18] or "do not care" [19], depending on the task-at-hand, since there is no universally best interpretation of the missing attribute values [17]. The former concept (denoted with '?') is used to indicate the lost data, e.g., missing sensory readings for an attribute. The latter (denoted with '$*$' or '$-$') indicates the unavailable data, e.g., attributes initially deemed irrelevant by a human, thus potentially not included in all resource descriptions.

2) Isolated actuations: As mentioned earlier, governance and elasticity actuations are declared via the `STRATEGY` construct. It encapsulates actuations such as "change communication protocol" or "spin up a VM". However, the underlying capabilities (which implement the actuation logic) are mainly running at the edge of the infrastructure, e.g., inside IoT gateways, and are invoked remotely over the network. Therefore, this often leads to failures and functionality degradations (transparent to users) as we discussed in Section II.

```
S:STRATEGY CASE Fulfilled(CND1):
 setUpdateRate(5s) FOR G //see Listing 1
 CONSIDERING_UNCERTAINTY:
   run_in_isolation = true AND
   keep_alive = 5min AND
   degree_parallelism = 200 AND
   tolerate_fault_percentage  = 20% AND
   fallback_count = 2 AND
   time_to_next_fallback = 500ms
```

Listing 2. Example of an isolated actuation with uncertainty considerations.

In order to support the users in managing such uncertainties, U-GovOps offers two levels of actuation isolation – per

[3] The full syntax of the U-GovOps language is described in https://github.com/tuwiendsg/rSYBL/blob/uGovernance/UGovOpsSYBLLanguage.pdf

```
1 C:CONSTRAINT responseTime<150ms WHEN nrOfUsers<900
2  CONSIDERING_UNCERTAINTY:decision_confidence >=20%
3 S1:STRATEGY CASE Violated(C):scaleOut()
4 S2:STRATEGY CASE Fulfilled(C):maximize(throughput)
5  CONSIDERING_UNCERTAINTY:
6   considering_strategies = StrategyX
```

Listing 3. Example elasticity actuations with uncertainty considerations.

governance policy and per capability invocation. To instruct U-GovOps to isolate a governance policy users can specify `run_in_isolation = true` (Listing 2). This effectively tells the framework to create a separate resource pool (e.g., a thread pool) for the policy and perform all actuations within that resource pool. More details about the design of this mechanism are given in Section IV-B.

To provide finer-grained control for the isolated policies and actuations, the U-GovOps framework exposes additional configuration parameters. For example, the `keep_alive` parameter enables users to specify the maximal time slot that should be allocated to a governance policy to complete. The `tolerate_fault_percentage` is a similar concept, designed to temporarily stop the policy execution in case the percentage of failed actuations exceeds a pre-defined threshold. These two concepts are based on the circuit breaker pattern[4], which are especially useful for handling blocked or zombie policies and reducing the resources tied up in operations which are likely to fail due to uncertainties. Further, the `degree_parallelism` tells U-GovOps how many actuations should be performed in parallel. This is useful in capturing the user's knowledge about the infrastructure's scale and dynamicity in order to optimize resource consumption. For example, if a governance policy is meant to govern all active gateways in a building (e.g., $\approx$ 300 at the time) it makes little sense to set the degree of parallelism to 1000. Finally, the `fallback_count` and `time_to_next_fallback` parameters are used to handle uncertainty at the level of a single actuation. Its main purpose is to support graceful handling of network latencies and timeouts and to guaranty fail-fast behavior (with quick recoveries) and graceful functionality degradation (with fallback logic).

Listing 3 gives an example of using elasticity actuations in governance policies. It first defines a `CONSTRAINT` directive, which describes desired conditions of keeping the response time below 150 ms if the number of current users is below 900. Lines 3 and 4 in Listing 3 tell U-GovOps to fire appropriate elasticity actuations based on the status of the constraint. However, the elasticity actuations are also subject to uncertainty, for example, due to platform glitches (e.g., unsuccessful network interface attachment) or infrastructure overload (e.g., collocation issues on physical servers) leading to unexpected behavior such as actuation delays. To account for such issues, U-GovOps allows users to specify their knowledge about the elasticity relationships, such as that increasing sensors update rate will most probably require scaling out the cloud services. The elasticity relationships can be specified via `considering_strategies` parameter, effectively enabling the framework to anticipate the aforementioned situations and for instance preemptively spin up required VMs. Naturally, all the uncertainty directives shown in Listing 2 are also valid in this context.

B. U-GovOps runtime mechanisms

1) Resolving rough governance scopes at runtime: When a request to compute a rough governance scope (Listing 1) arrives in U-GovOps runtime, the framework performs the following general steps: i) It first evaluates the user-provided `query` and performs the resource selection with the currently available data. If no uncertainty parameter is specified or `use_cache=true` and there is a precomputed governance scope for the query, the U-GovOps framework immediately returns the obtained resource set. Otherwise, it proceeds with the next steps. ii) Parametrize the missing data. iii) Calculate Similarity Classes [18]. iv) Calculate characteristic sets. v) Return a governance scope approximation.

In order to parametrize the missing data the U-GovOps framework first tries the assignments from $missing\_data$ directive. The permissible values to assign to the missing attributes include ‘?’, ‘$*$’ and ‘$-$’. The ‘?’ is used to denote that the attribute value might be lost and ‘$*$’ or ‘$-$’ mean that the user suspects that the attribute values were unavailable in the first place. If no user-provided parameter exists for an attribute, U-GovOps will associate it with the ‘?’ by default. Although straightforward, this process has a significant impact on the framework's decisions how to compute the the governance scope. For example, assigning the ‘?’ to a device's attribute instructs U-GovOps not to include that device in any Similarity Classes for such attribute. Further, the ‘$*$’ tells U-GovOps that the original values were irrelevant, thus can be considered as any value consistent with that attribute. Finally, the ‘$-$’ tells the framework that these missing values can be considered as any value consistent with that concept, as discussed in [17]–[19].

Algorithm 1: Computing characteristic sets.

```
input: $res$ : Governed resource, $GS$ : Global scope, $G$ : Attribute list
result: $CS$ : Characteristic set for the $res$.
$CS \leftarrow GS$
forall the $attr$ in $G$ do
    switch $attr$ do
        case $'?' = res.attr$
            $CS \leftarrow GS$
        case $'*' = res.attr$
            foreach $val \in AttrDomain(attr)$ do
            $CS \leftarrow CS \cup SimilarityClass(attr, val)$
        case $'-' = res.attr$
            $V \leftarrow \{r | r \in GS, isDefined(r, attr), r.d = res.d\}$
            if $V \neq \emptyset$ then
                foreach $r \in V$ do
                    $CS \leftarrow CS \cup SimilarityClass(attr, r.attr)$
                end
            else $CS \leftarrow GS$
        otherwise /*attr is defined (not missing)*/
            $CS \leftarrow CS \cap SimilarityClass(attr, res.attr)$
        end
    endsw
end
```

To calculate the characteristic set for a resource, e.g., a device, the U-GovOps framework performs the calculation as shown in Algorithm 1. The intuition behind the algorithm is to enable determining similar resources, under attributes G with missing information, by considering problem-dependent uncertainty parametrization. Finally, based on the specified `selection_strategy` the U-GovOps returns a governance scope. For example, for optimistic selection strategy the governance scope, to be returned, is calculated as upper approximation of the targeted scope X with: $\overline{G}X = \bigcup\{CS_G(r) \mid r \in X\}$, where CS is a characteristic set for a resource r and G is the specified attribute set.

2) Actuating under uncertainty: Figure 4 outlines the most important steps performed by U-GovOps to support the isolated actuations (we omit loops, caching, error handling, etc., for

[4] http://martinfowler.com/bliki/CircuitBreaker.html

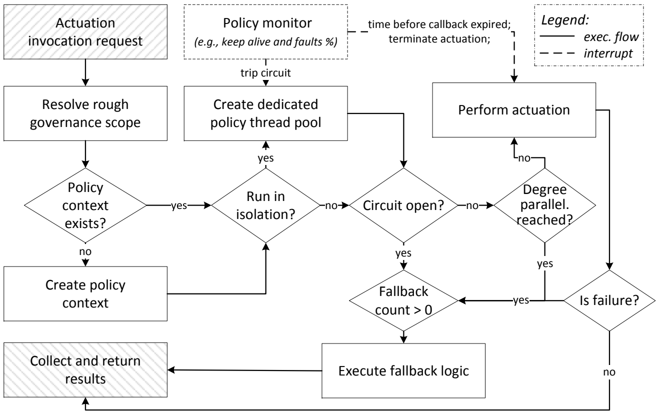

Fig. 4. Execution flow for isolated actuations.

readability purposes.). This mechanism is triggered when a user submits a policy (e.g., as shown in Listing 2) to U-GovOps for execution. A user only observes the invocation calls and the returned results (shown hatched in Figure 4). The other steps are performed by the framework, transparent to the users.

Initially, the U-GovOps framework resolves the rough governance scope and creates a policy context, which stores the uncertainty parameters (supplied by the user), the computed governance scope and the policy invocations cache. The subsequent steps are mainly determined, by the user-provided uncertainty configuration directives (Listing 2). If the `run_in_isolation` is set to true, U-GovOps isolates the policy by allocating a dedicated resource pool for it. In the current prototype this is realized by instantiating a dedicated thread pool (per policy) and performing all policy actuations (on separate threads) within that thread pool. In case a policy should not be executed in isolation, individual actuations will still remain isolated, but they will share the same global resource pool.

The Policy Monitor (Figure 4) implements the circuit breaker[4] and continuously monitors the threads and the thread pools (policies) for the aforementioned conditions such as the permissible fault percentage and keep alive timeouts. Currently this is implemented based on the Netflix OSS Hystrix, since U-GovOps uses HTTP as the underlying protocol for Remote Procedure Calls. If the constraints are violated, the Policy Monitor trips the circuit, denying the further resources to that policy and temporally putting its execution on hold or interrupting its execution if keep alive expired. Generally, this or an actuation failure will trigger the execution of the fallback logic (if `fallback_count>0`). Currently, the U-GovOps framework only provides a rudimentary support for specifying the fallback logic, by retrying the normal flow or returning a generic error if everything else fails. In the future we plan to address this and allow injecting custom fallbacks. Finally, U-GovOps collects the actuation results (if any) and returns them to the calling policy. through the utilization of Futures and Promise pipelining, which enable asynchronous result processing with minimal latency.

V. Evaluation

In this section, we present the preliminary results of our experiments. Our experiments comprise two general parts. First, we perform a functional evaluation of U-GovOps's language support for implementing *uncertainty- and elasticity-aware governance policies*, based on our real-life use case (Section II). Second, we evaluate U-GovOps's main runtime mechanisms for mitigating runtime infrastructure uncertainties.

A. Experiments setup

In order to evaluate how our framework behaves under uncertainty, we created a testbed for virtualized IoT cloud systems using CoreOS. We used Docker containers to virtualize and mimic physical gateways in the cloud. These containers are based on a snapshot of a real-world, proprietary IoT gateway. The Docker base image is publicly available in Docker Hub under dsgtuwien/govops-box.

For the subsequent experiments we deployed the testbed on our local OpenStack cloud, running approximately 1000 Docker containers (simulating the gateways/nodes). Each of the containers "hosts" different virtual sensors (e.g., location) and are associated with different meta data (e.g., owner). These sensors replay the prerecorded real-life data, obtained in our case study. Since the main aim is to govern the infrastructure services and resources, we only consider the infrastructure state data relevant for the governance policies and not the data used by the business logic cloud services (although these might overlap). The U-GovOps controllers and the demo application (Section II) are deployed separately, in the same cloud on 4 Ubuntu 14.04 VMs (with 2VCPUs and 3GB of RAM) and used to execute our governance policies. Finally, to simulate the uncertainties, i.e., the *missing or incomplete data* (about the infrastructure states) and *actuation uncertainties*, we developed three mechanisms (based on Dell Blockade[5]), which perform random fault injections: (i) killing of the containers, (ii) dropping of data packets and, (iii) slowing down the network.

B. Example governance policy implementation

We first show how U-GovOps language is used to develop the real-life governance policy for the PMA application, presented in our case study (Section II). Listing V shows the complete source code of the governance policy. Since it mostly uses the familiar language concepts, presented earlier in the paper, we focus on the most important features of our language.

We notice that a user utilizes intuitive, high-level abstractions and configuration directives to declare what needs to be done instead of specifying how to do it (e.g., Listing V, lines 4-7). For example, a user does not directly invoke the individual actuations nor has to explicitly handle actuation failures or recovery logic, since the actual invocations are pushed down to U-GovOps, who transparently handles lost actuations and prevents cascading failures, based on the user-provided configurations. Further although our framework limits the expressiveness to a certain extent, the users can still express many common behaviors of governance strategies. For example, the user can easily specify the desired elasticity behavior, taking into account possible uncertainties caused by related actions (lines 8-9). Finally, our framework simplifies the user effort in dealing with the data uncertainties (lines 1-2), since the users do not have to write complex queries or explicitly deal with False Positive (FP) and False Negative (FN) results.

C. Experiments results

Next, we evaluate main U-GovOps runtime mechanisms: *resolving rough governance scopes* and for *isolating the actuations* under presence of two main uncertainties: *missing or incomplete data* and *actuation uncertainties* (simulated as described above).

[5] https://github.com/dcm-oss/blockade

```
G1: GOVERNANCE_SCOPE query: location=building3&type=JACE-545||owner=TUW
      CONSIDERING_UNCERTAINTY: missing_data=location<=’?’, owner<=’*’ AND selection_strategy=optimistic;
M1: MONITORING abnormal_behavior := sensorAlert(G1)==true OR heartBeatAVG(G1)>5min;
S1: STRATEGY CASE abnormal_behavior: setProtocol(’mqtt’), changeUpdateRate(’5s’) FOR G1
      CONSIDERING_UNCERTAINTY: run_in_isolation=true AND keep_alive=1min AND fallback_count=2 AND
                               tolerate_fault_percentage = 20% AND invocation_caching=true;
C1: CONSTRAINT cost<200 CONSIDERING_UNCERTAINTY: decision_confidence >=20%;
S2: STRATEGY CASE responseTime>250ms: scaleOut() CONSIDERING_UNCERTAINTY:considering_strategies = S1;
```

Listing 4. Example PMA governance policy.

The experiment results are averaged on 50 repetitions and we have experimented with 7 different governance policies, which have different properties regarding query complexity and actuation types (e.g., execution time and computational complexity).

TABLE I. AVERAGED F1 SCORES FOR THE GOVERNANCE SCOPES.

Percentage of the missing data	10%	20%	30%	40%	50%
F1 scores - optimistic strategy	0.95	0.86	0.86	0.80	0.74
F1 scores - pessimistic strategy	0.90	0.90	0.80	0.79	0.72
F1 scores - no uncertainty consideration	0.91	0.80	0.66	0.50	0.29

To evaluate the coverage of our governance policies, i.e., the “goodness” of approximation of our governance scopes under uncertainty (missing data) we show two relevant metrics: the F1 scores and the error rates, as cumulative metric for the FPs and FNs. The baseline is calculated with “perfect information” (no missing data) and then we repeated the policies execution, while simulating the data losses. Table I, shows the resulting averaged F1 scores. The missing data represents the percentage of missing data instances randomly distributed across the resources and the resource attributes. We run 3 different setups: not considering the data uncertainty (i.e., ignoring the missing data), using optimistic selection strategy and using the pessimistic selection strategy. The corresponding error rates are shown in Figure 5.

It is important to notice (Figure 5) that not considering uncertainties and pessimistic strategy only contain FNs (i.e., resources that should be included in the governance scope, but were not due to the lack of information), while optimistic strategy only returns FPs (i.e., includes the desired resources with certainty). This shows an important property of our approach, i.e., it enables users to make trade-offs depending on the task-at-hand. For example, governance policies that do not care about FPs (formulated as: “ALL resources with specific properties MUST be included”) can be easily specified with optimistic selection strategy. Additionally, compared to traditional approaches (no uncertainty consideration) our pessimistic selection strategy generally behaves better, i.e., displays on average about 20% less errors. Finally, it is worth noting that parametrization of missing data (different combinations of ‘?’, ‘$*$’ and ‘$-$’) had a significant impact on the quality of the results. This can be considered a drawback, since it steepens the learning curve of U-GovOps language. In the future we plan to explore this phenomenon in order to derive suitable heuristics for parameterizing the missing data in governance scopes.

Figure 6 shows the percentages of lost actuations, with and without U-GovOps mechanism for isolated actuations, for different fault rates. For example, for fault rate of 10% we know that 10% of all actuations will be affected by at least one of the 3 fault injection actions. For all the evaluated policies we use the same base-line, e.g., number of containers (≈ 1000), configurations, etc. The isolated actuations are configured in a greedy fashion, with main objective to mitigate as many uncertainties as possible. Generally, by isolating the actuations we managed to reduce the rate of lost actuation by more then 50% on average, compared with the traditional approaches, which do not consider uncertainties. The majority of unaccounted uncertainties were due to the killed containers, since it is currently not possible to compensate this with U-GovOps.

On the secondary axes (Figure 6), we show the average execution time of the governance policies. We notice that average execution time of the policies without uncertainty consideration was only slightly affected by the faults, mainly due to network slowdowns. On the other hand, our approach had an exponential increase in execution time with high fault rates. This is mainly due to the exponential back-off policy implemented by the framework in order to be “fair” to the underlying actuators, i.e., not overload them with requests, e.g., in case of major network problems. This shows an important property of the uncertainty management that it does not come “for free”, in the sense that users need to accept some overhead, e.g., of performance or additional costs, in order to account for the uncertainties. Naturally, the users can control these aspects by relaxing the uncertainty constraints.

We are also aware of certain limitations of our framework. For example, maintaining the threads and the thread pools per actuation/policies causes an additional computational overhead, due to thread queueing, scheduling, and context switching.

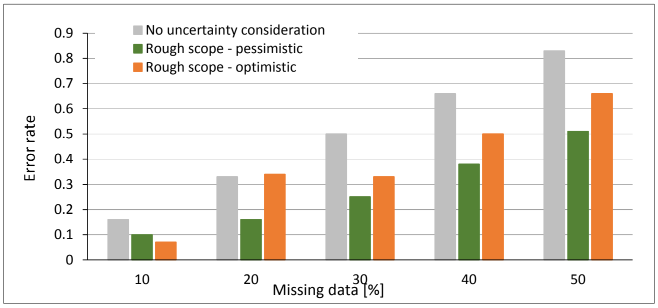

Fig. 5. Error rates for governance scopes due to missing data.

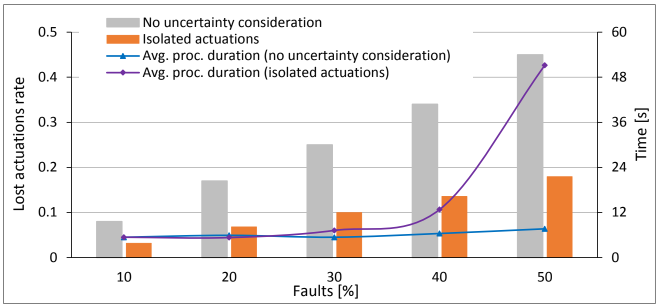

Fig. 6. Lost actuations rates for isolated actuations.

We deliberately decided to make a tradeoff here, since we believe that the overall advantages of having more resilient and fault tolerant governance overweight the additional costs in the long run. Finally, currently U-GovOps mainly focuses on runtime infrastructure uncertainties and does not explicitly consider cumulative effects and uncertainty propagation. This is, however, subject of our future work.

VI. Related Work

Many approaches dealing with IoT cloud operations, resources management and governance have recently emerged. For example, in [2] and [4] the authors mostly deal with IoT infrastructure virtualization and its management on cloud platforms. A number of different approaches (e.g., [5]) employ semantics aspects to enable discovering, linking and orchestrating heterogeneous IoT devices. In [6] the authors propose utilizing cloud for additional computation resources and approaches presented in [3] focus on utilizing cloud's storage resources for sensory data. Also in [20] the author evaluates various governance aspects, such as privacy, security, ethics, etc., and defines main principles of IoT governance. However, contrary to the U-GovOps framework, most of these approaches provide little or no support for considering IoT cloud infrastructure uncertainties in governance strategies.

In the field of self-adaptive systems (SAS) there are many approaches dealing with uncertainties and faults. For example, in [12], the authors present a taxonomy of uncertainty for dynamically SAS. Whittle et al. [21] developed RELAX, a textual requirements language that provides fuzzy logic-based operators to facilitate the specification of uncertainties in SAS at requirements level. Weyns et al. [31] introduced FORMS, a formal reference model for self-adaptation that builds upon feedback loops to enable addressing uncertainties at design level. The runtime uncertainties are addressed in [22], mainly using the concept of reactive feedback loops. Such approaches conceptually complement our own, by providing valuable insights and techniques to understand and analyze uncertainties. However, the distinct feature of U-GovOps is that it considers both elasticity and uncertainty at the level of governance policies.

Approaches form Wireless Sensor Networks (WSN) also deal with uncertainties, e.g., [23], [24]. However, they mostly deal with sensor network deployments and detecting redundant sensors in the WSN. Contrary to such approaches, to our best knowledge, our U-GovOps is the first attempt to enable developing uncertainty- and elasticity-aware governance strategies encompassing both IoT and cloud infrastructures.

VII. Conclusions and Future Work

In this paper we introduced the U-GovOps framework for governing elastic IoT cloud systems under uncertainty. We presented the U-GovOps *declarative policy language* for developing uncertainty- and elasticity-aware governance policies. The main U-GovOps runtime mechanisms for *managing rough governance scopes* and *enabling isolated actuations* were introduced to facilitate enforcing such polices, by effectively mitigating the infrastructure uncertainties, as demonstrated on a real-life case study. The initial results are promising in the sense that with the U-GovOps framework users can develop custom governance strategies efficiently, by using intuitive, high-level abstractions and configuration parameters without explicitly dealing with the uncertainties of complex interactions in IoT cloud infrastructure.

In the future we plan to improve the U-GovOps runtime mechanisms, especially to include heuristics for parameterizing the missing data and support for custom fallback logic. We also plan to extend U-GovOps to include support for managing the remaining uncertainty families identified in our taxonomy, as well as cumulative uncertainty effects.

Acknowledgment

Supported by the EU H2020 U-Test project, grant No. 645463.

References

[1] H.-L. Truong and S. Dustdar, "Principles for engineering IoT Cloud systems," *Cloud Computing, IEEE*, vol. 2, pp. 68–76, 2015.

[2] M. Yuriyama and T. Kushida, "Sensor-cloud infrastructure-physical sensor management with virtualized sensors on cloud computing," in *NBiS*, 2010.

[3] P. Stuedi, I. Mohomed, and D. Terry, "Wherestore: Location-based data storage for mobile devices interacting with the cloud," in *MCS*, 2010.

[4] M. M. Hassan, B. Song, and E.-N. Huh, "A framework of sensor-cloud integration opportunities and challenges," in *ICUIMC*, 2009.

[5] J. Soldatos, M. Serrano, and M. Hauswirth, "Convergence of utility computing with the internet-of-things," in *IMIS*, pp. 874–879, 2012.

[6] B.-G. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti, "Clonecloud: elastic execution between mobile device and cloud," in *Conference on Computer systems*, ACM, 2011.

[7] S. Nastic, S. Sehic, D.-H. Le, H.-L. Truong, and S. Dustdar, "Provisioning Software-defined IoT Cloud Systems," in *FiCloud'14*, 2014.

[8] S. Nastic, C. Inzinger, H.-L. Truong, and S. Dustdar, "GovOps: The Missing Link for Governance in Software-defined IoT Cloud Systems," in *WESOA14*, 2014.

[9] S. Nastic, M. Voegler, C. Inziger, H.-L. Truong, and S. Dustdar, "rtGovOps: A Runtime Framework for Governance in Large-scale Software-defined IoT Cloud Systems," in *Mobile Cloud 2015*, 2015.

[10] A. Avizienis, J.-C. Laprie, B. Randell, *et al.*, *Fundamental concepts of dependability*. University of Newcastle, Computing Science, 2001.

[11] K. Potter, P. Rosen, and C. R. Johnson, "From quantification to visualization: A taxonomy of uncertainty visualization approaches," in *Uncertainty Quantification in Scientific Computing*, pp. 226–249, Springer, 2012.

[12] A. J. Ramirez, A. C. Jensen, and B. H. Cheng, "A taxonomy of uncertainty for dynamically adaptive systems," in *SEAMS'12*, 2012.

[13] G. Copil, D. Moldovan, H.-L. Truong, and S. Dustdar, "Sybl: an extensible language for controlling elasticity in cloud applications," in *CCGRID'13*.

[14] W. E. Walker, P. Harremoës, J. Rotmans, J. P. van der Sluijs, M. B. van Asselt, P. Janssen, and M. P. Krayer von Krauss, "Defining uncertainty: a conceptual basis for uncertainty management in model-based decision support," *Integrated assessment*, vol. 4, no. 1, pp. 5–17, 2003.

[15] M. Zhang, S. Ali, T. Yue, D. Pradhan, B. Selic, O. Okariz, and R. Norgren, "An Uncertainty Taxonomy to Support Model-Based Uncertainty Testing of Cyber-Physical Systems," tech. rep., Simula, 2015.

[16] Z. Pawlak, "Rough sets," *International Journal of Computer & Information Sciences*, vol. 11, no. 5, pp. 341–356, 1982.

[17] J. W. Grzymala-Busse, "Three approaches to missing attribute values: A rough set perspective," in *Data Mining: Foundations and Practice*, pp. 139–152, Springer, 2008.

[18] J. Stefanowski and A. Tsoukias, "Incomplete information tables and rough classification," *Computational Intelligence*, vol. 17, no. 3, 2001.

[19] M. Kryszkiewicz, "Rules in incomplete information systems," *Information sciences*, vol. 113, no. 3, pp. 271–292, 1999.

[20] R. H. Weber, "Internet of things–governance quo vadis?," *Computer Law & Security Review*, vol. 29, no. 4, pp. 341–347, 2013.

[21] J. Whittle, P. Sawyer, N. Bencomo, B. H. Cheng, and J.-M. Bruel, "Relax: Incorporating uncertainty into the specification of self-adaptive systems," in *RE'09*, 2009.

[22] D. Garlan, S.-W. Cheng, A.-C. Huang, B. Schmerl, and P. Steenkiste, "Rainbow: Architecture-based self-adaptation with reusable infrastructure," *Computer*, vol. 37, no. 10, pp. 46–54, 2004.

[23] M. R. Senouci, A. Mellouk, L. Oukhellou, and A. Aissani, "Uncertainty-aware sensor network deployment," in *GLOBECOM'11*, IEEE, 2011.

[24] S. Mal-Sarkar, I. U. Sikder, C. Yu, and V. K. Konangi, "Uncertainty-aware wireless sensor networks," *International Journal of Mobile Communications*, vol. 7, no. 3, pp. 330–345, 2009.