CoalaViz: Supporting Traceability of Adaptation Decisions in Pervasive Communication Systems

Martin Pfannemüller*, Markus Weckesser†, Roland Kluge†, Janick Edinger*,
Manisha Luthra†, Robin Klose†, Christian Becker*, Andy Schürr†
*Universität Mannheim
Email: {martin.pfannemueller, janick.edinger, christian.becker}@uni-mannheim.de
†Technische Universität Darmstadt
Email: {markus.weckesser, roland.kluge, andy.schuerr}@es.tu-darmstadt.de,
manisha.luthra@kom.tu-darmstadt.de, robin.klose@seemoo.tu-darmstadt.de

***Abstract*—Today's pervasive communication systems are highly configurable to adapt themselves dynamically to continuously changing contexts of the system such as varying workloads and user preferences. For a particular context, usually numerous valid system configurations exist, and each configuration may perform differently in terms of nonfunctional properties like energy consumption or task throughput. For tackling these challenges, in previous work, we introduced COALA, a model-based adaptation approach to derive optimal system configurations considering multiple performance goals. In this paper, we present COALAVIZ, a novel tool for visualizing the self-adaptive behavior of pervasive communication systems. With COALAVIZ, we provide a tool for making adaptation decisions in self-adaptive pervasive communication systems traceable while being applicable for a wide range of use cases. COALAVIZ (i) visualizes the system performance over time, (ii) visualizes the system state as context feature model and graph-based network view, (iii) allows the user to change priorities of performance goals interactively, and (iv) provides a modular, extensible design. We demonstrate the applicability of COALAVIZ using three pervasive system use cases.**

I. INTRODUCTION

Today's pervasive communication systems consist of numerous networking capabilities and protocols, each tailored to certain contexts. In pervasive systems, the system context changes continuously (e.g., changing network loads or varying amount of available computing resources). Therefore, such systems need to adapt their (network) configuration to always achieve certain performance goals. Adapting these systems is very challenging because, often, (i) numerous suitable configurations are possible, but only certain system configurations are valid in the current context, (ii) the performance of a particular configuration strongly depends on the context (e.g., varying workloads and user preferences), and (iii) multiple potentially conflicting performance goals should be achieved at the same time with potentially changing and context-dependent priorities.

We previously presented COALA [1], a model-based self-adaptation approach to tackle these challenges. COALA is well-suited for *self-adaptive systems (SAS)* in the communication domain, which reconfigure themselves to match dynamically changing contexts [2]. COALA is based on Dynamic Software Product Line (DSPL) engineering [3] techniques. It employs (i) context feature models [4] to specify valid configurations and (ii) performance influence models [5] to estimate the expected influence of (re-)configurations on performance goals. Based on these models, COALA determines optimal system (re-)configurations considering multiple performance goals.

Still, the resulting reconfiguration decisions cannot be traced back to the current system configuration, the contextual parameters, or the system performance, which all contribute to the reconfiguration decision. In this paper, we present COALAVIZ, a novel tool for demonstrating the reconfiguration behavior of self-adaptive communication systems. COALAVIZ offers the following insights into the reconfiguration decisions of COALA: (i) The current system state can be investigated using a configurable graph-based network view (e.g., the overlay and underlay network of a distributed system). (ii) All the configuration space and the currently active configuration of the system can be inspected using a feature-model view. (iii) The current system performance in terms of nonfunctional properties is shown as one or more metric plots. (iv) The priorities of the performance goals can be inspected and adjusted at runtime. We designed COALAVIZ to be a standalone tool with clear technical interfaces for each of the described visualization components. These interfaces simplify the use of COALAVIZ in different evaluation scenarios (e.g., simulation and testbed) and use cases. We demonstrate the applicability of COALAVIZ using three use cases: the TASKLET computation offloading middleware, an adaptive wireless sensor network, and an adaptive complex event processing framework.

The remainder of the paper is structured as follows: Section II presents the three pervasive communication use cases and discusses their respective challenges, which inspired the development of COALAVIZ. Section III presents a discussion of related work on visualizing different system metrics. Section IV provides background information and gathers requirements for COALAVIZ based on the discussed challenges for visualization of self-adaptivity and feature modeling. Section V presents the architecture and implementation of COALAVIZ. Section VI contains the results of an evaluation of COALAVIZ w.r.t. the challenges. Finally, Section VII summarizes our results and outlines directions for future work.

978-1-5386-9151-9/19/$31.00 ©2019 IEEE

II. Use Cases and Challenges

This section introduces three pervasive communication systems illustrating the key challenges of traceability, extensibility, and responsiveness for the development of CoalaViz.

Tasklet System: The Tasklet system is a context-aware computational offloading middleware [6]. It offers application developers a lightweight abstraction for computation to execute tasks on heterogeneous remote resources. In the system, resource consumers run computational intensive tasks, which they can offload in the form of so-called Tasklets to resource providers. A central broker performs the matchmaking between consumers and providers. For the scheduling decision, the broker takes context information into account, for example, to avoid failures or to meet deadlines of tasks. To avoid failures, it selects a scheduling algorithm that most accurately predicts the availability of the providers, which enter and leave the system dynamically [7]. To meet deadlines, the broker reserves powerful providers for complex, long-running Tasklets. However, in an underutilized system, access to these resources can be also granted to less complex Tasklets to increase the throughput of the system. For this decision, the broker has to monitor the utilization of the system as well as the availability and performance of resource providers.

Wireless Sensor Networks (WSNs): A wireless sensor network (WSN) consists of dozens to hundreds of cheap, battery-powered, resource-constrained sensor devices (called *motes*) that collectively serve a particular purpose (e.g., environmental monitoring) [8]. A modern mote provides numerous configuration options to adjust the WSN to the current system context (e.g., mobility pattern and robustness requirements). *Topology control* is a technique to address nonfunctional system goals (e.g., the energy consumption) of a WSN by thinning out the number of visible neighbors on the link layer. A topology control algorithm presents the resulting *virtual topology*, which is a subgraph view of the physical neighborhood, to the network layer. The sparsity of the virtual topology comes at the cost of decreased robustness and/or higher latency. Each of the numerous topology control algorithms that have been proposed in the literature provides a different trade-off between energy consumption, robustness, and latency. In the context of the Internet of Things (IoT) WSNs are being used in safety- and security-critical scenarios (e.g., e-health, intrusion detection). Therefore, reconfiguring the topology control algorithm of a mote periodically is required to meet the safety, security, and performance requirements [9].

Complex Event Processing (CEP): Complex Event Processing (CEP) deals with processing continuous streams of data from devices (*producers*) to derive meaningful events for the end-users (*consumers*). *Complex events* are highly relevant for applications in the context of IoT (e.g., weather monitoring using WSNs). A complex event can be expressed as continuous query that is registered with the CEP engine. The CEP query is composed of *logical* units called *operators*. The CEP system processes the query in a distributed manner by placing the operators on the devices in the network (e.g., *motes* in a WSN). *Operator placement* is a mechanism that places operators based on the nonfunctional requirements posed by the consumers. Therefore, a CEP system exposes an underlay view, consisting of connected consumers, producers, and brokers, and an overlay placement view, better known as an *operator graph*. The operator placement should be such that it fulfills the nonfunctional requirements of the *consumers*. However, the *consumers* may have distinct and conflicting nonfunctional requirements depending on the current environmental conditions (context), e.g., when the operators are placed on mobile devices or cloud resources. In particular, one nonfunctional requirement of IoT applications is to deliver complex events in minimum *response time*, but also at a low cost in terms of overhead for mobile devices (e.g., measured as number of messages exchanged). These distinct conflicting requirements are hard to be fulfilled using *one* operator placement mechanism, but requires a runtime reconfiguration of the CEP system with *multiple* operator placement mechanisms.

Three Challenges: All three use cases expose the following three key challenges that we tackle with CoalaViz.

C1 Traceability: How does the adaptation logic come up with an adaptation decision based on the current system state? With the assumption that adaptation decisions depend on the system context, the current system context needs to be presented in an understandable way. Additionally, for tracing an adaptation from one state to another, the current system state shall be visualized. As the state of a communication system can typically be represented by a graph, effects like entity churn or node movements can comprehensibly be visualized. In doing so, nodes can form both an overlay and an underlay network, resulting in different changing topologies, respectively. Besides the context and the system state, also nonfunctional requirements, such as execution speed, fairness or response time, may change during runtime. Hence, nonfunctional performance metrics shall be traceable as well. Additionally, as the goals of a system might change at runtime, our solution must show which goals are pursued at any point in time.

C2 Extensibility: While the three presented use cases address important and well-known challenges, they may still be considered just a problem subset in the field of self-adaptive systems. Hence, an important challenge is to develop a tool that exposes clear extension points for supporting new use cases. In doing so, we also take into account that each use cases may well exhibit its individual nonfunctional system goals. This would ensure that different scenarios and simulation tools could be combined with it.

C3 Responsiveness: The visualization tool shall address the pictured system's dynamics by providing a decent level of responsiveness. We note that communication systems have several degrees of dynamics at different levels of the communication protocol stack, e.g., movement or changing communication connections. Nevertheless, the performance metrics shall be monitored continuously and reliably and with low delay on all these levels. This shall assist system designers in maintaining a clear and comprehensible picture of the overall system and its adaptation decisions.

III. Related Work

For evaluating pervasive communication systems, network simulators are widely used. These simulators often provide some capabilities for tracing the system behavior. Thus, this sections briefly surveys related work on traceability in network simulators. We organize the presented related work along the discussed use cases. Some network simulators are general, which are not limited to the use case category it is part of.

Tasklets: For the TASKLET system, an OMNeT++ [10] implementation exists. The OMNeT++ network simulator can show network topologies including package transmissions between nodes. However, it does not allow to get live information on system metrics or system configuration without manual implementation inside the OMNeT++ ecosystem. Thus, even if these features are implemented inside OMNeT++, they cannot be reused with other simulators or even real systems.

WSNs: A state-of-the-art simulator for WSNs is SIMONSTRATOR [11]. The simulator provides basic building blocks for visualizing the topology of the network and plotting metrics as Java Swing components. In [9], a visualization for the virtual topology of a topology control algorithm is presented. A visualization of the configuration space and the current system configuration as well as a configuration panel for interacting with (adaptation logic) components are missing.

CEP: CEP use cases can be evaluated using network simulation, emulation, or on actual data stream processing (DSP) or CSP systems (e.g., Apache Storm, Flink, Akka, Esper). CEPSim [12] and DCEP-Sim [13] are the two simulators based on existing simulation platforms CloudSim and ns3 respectively, providing CEP abstractions on top of these platforms. Thus, both support visualization of CEP network and operator graph and can be extended to develop different use cases (**C1** and **C2**). However, these simulators only partially deal with **C1**. They do not support (i) visualizing adaptive decisions i.e., changes in nonfunctional metrics, (ii) interaction with the configuration space and visualization of these changes. CEP use cases can also be evaluated using emulation [14] or on DSP/CEP systems which are highly responsive in their design (**C3**). However, they lack proper network modeling and network graph representation as provided in this work (**C1**).

Further works: Existing additional visualization approaches are either tailored towards a single use case (e.g., [15]–[17]) or highly generic [18]. None of these approaches provide a visualization of the reconfiguration behavior in a reusable way.

IV. Visualization Techniques

To ensure traceability, our solution needs to provide views for the network topology, metrics, performance goals, configuration space, and the selected configuration. The *network topology* is a graph that represents the arrangement of the elements of a communication system, i.e., the devices and their communication links. Depending on the considered system, the topology view shows an overlay or an underlay view of the network, or a combination of both. A *metric* quantifies the current system performance, e.g., in terms of throughput or latency. Metrics are usually visualized with plots, while multiple metrics can be visualized in the same plot at each point in time. The performance goal of the system is often a weighted combination of individual performance goals. To evaluate whether a performance goal is pursued by the system, a corresponding metric plot can be used.

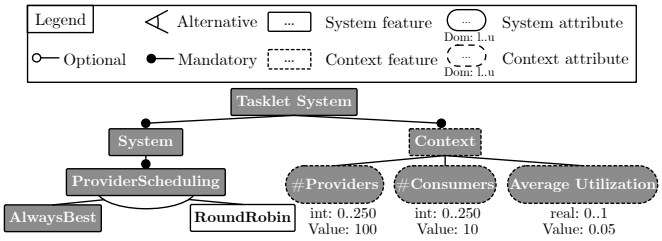

Fig. 1: TASKLET feature model. Gray: current configuration.

The context and the configuration space of a system can be represented jointly using feature models. A *feature model* is a set of features together with constraints among the features [19]. A *feature* represents a binary configuration option, which is either selected or deselected. A *configuration* represents the state and context of a system as a set of selected features. Figure 1 depicts a small excerpt of the configuration space of the TASKLET system as feature model. Features are organized in a parent-child feature relationship shown as tree structure. For example, the ProviderScheduling features has the child-features AlwaysBest and RoundRobin. A parent-child feature relationship is of type *or*, *alternative*, *mandatory*, or *optional*. For example, only one child-feature of ProviderScheduling can be selected in a configuration. An *attribute* represents a property of its parent feature [20] (e.g., integer-valued for #Providers and real-valued for Average Utilization). A *context feature model (CFM)* represents relevant properties of the environmental context as *context features* and their dependencies to *system features*. In Figure 1, system features and context features appear in separate branches of the feature model, i.e., context features below Context, and system features below System. A selection of context features reflects a given context. When the context changes, the context configuration (i.e., features in the context branch) is adjusted accordingly, and the system configuration (i.e., features of the system branch) may need to be adapted.

In a *self-adaptive system* (SAS), reconfiguration knowledge and decisions are managed by an *adaptation logic* (AL)The AL controls a *managed resource* (MR), which represents the actual system containing the business logic [2]. The MR provides sensor data of its current state and its context to the AL. Based on the monitored data, it decides if and how the MR should be reconfigured. If a reconfiguration is required, the decision is sent back to the MR, which performs the actual adaptation of the system. The COALA [1] adaptation approach employs CFMs to represent incoming context information and resulting system configurations.

V. CoalaViz Approach

Figure 2 illustrates the architecture of COALAVIZ with the COALA-based SAS [1] on the left-hand side and COALAVIZ

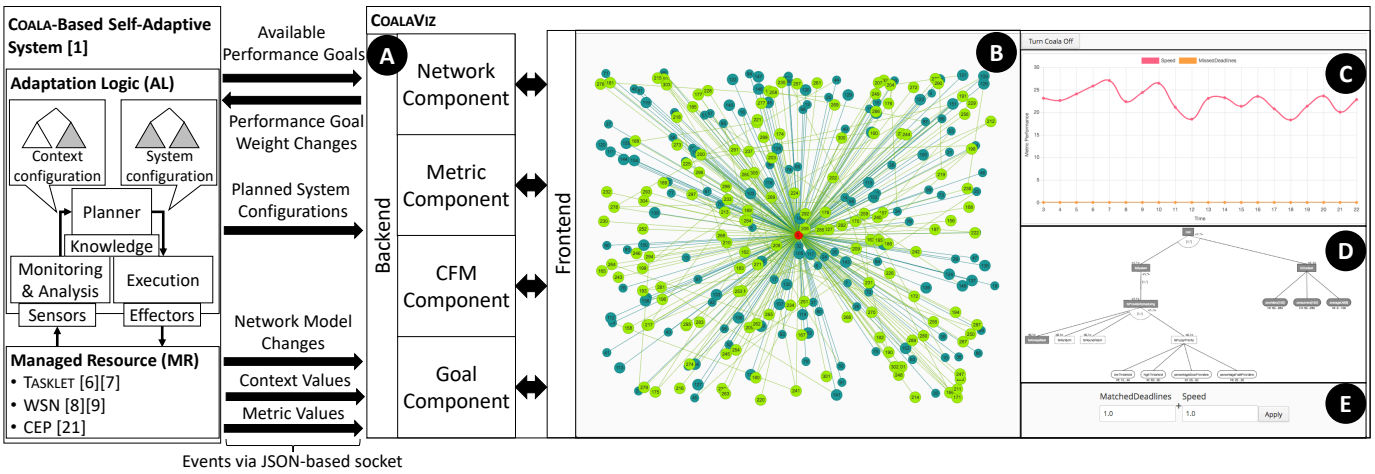

Fig. 2: COALAVIZ with self-adaptive system. Left: COALA adaptation logic [1] with a managed resource. Right: COALAVIZ with backend A, received events, and visualization of TASKLET with network B, metric C, CFM D, and goal E views.

consisting of its backend and frontend on the right-hand side.

The MR is decoupled from the COALA AL via sensor and effector interfaces. The AL uses the MAPE-K pattern [2]. The MR sends information about the current system to the AL, which analyzes the received data, resulting in a (partial) context configuration, and plans an appropriate reconfiguration of the system, resulting in a complete system configuration. The execution component transmits the decision to the effector interface of the MR. Details on COALA can be found in [1].

The architecture of COALAVIZ is separated into backend and frontend components. Backend components process the events that originate from the SAS and arrive via one or more event streams A, and notify the corresponding frontend components (B…E). The major event types are shown as broad black arrows in Figure 2. All information that is visualized in the frontend components originate from these received events. This allows COALAVIZ to replay events even in the absence of a running system. A JSON- and socket-based interface allows COALAVIZ to receive events independently of the programming language (e.g., C++ for OMNET++) and type of MR (e.g., simulators or actual devices). Each frontend component can be exchanged individually, e.g., to show a logical expression instead of the graphical feature modeling view. Figure 2 indicates that the backend consists of components for each view that appropriately translate events for the actual frontend implementation. The frontend consists of a graph-based network view B, a metric view showing the reported nonfunctional property values of the system C, a combined CFM and configuration view D, and the performance goals control panel, which shows the weighted performance goals and allows to update the weights interactively E. COALAVIZ interacts with the MR and the AL of an SAS. To establish compatibility with COALAVIZ, we extended the COALA AL to emit events about 1) available performance goals with default weights during initialization for the panel E, 2) the CFM of the AL for the view D, and 3) new system configurations whenever the planner produces a new reconfiguration decision, which are also visualized by D. Conversely, COALAVIZ informs the AL, when the user modifies the performance goal weights.

The MR (e.g., the TASKLET simulation in OMNET++) sends events about 1) modifications of the network state, such as node or edge additions, removals, or property modifications, which are visualized by B, and 2) new metric values as triple of metric name, timestamp, and metric value, as visualized by C. The network view interprets optional node and edge properties as rendering hints (e.g., node fill and border color, edge color and stroke, textual node and edge labels).

Network Component: The network component processes the network model change events from the managed resource. It supports events for adding and modifying nodes and edges that represent the network state. The network frontend view B shows a graph that represents the current state of the network. The position of each node is communicated by the MR. COALAVIZ maintains the graph structure based on the event stream and visualizes it in the network view. A user of COALAVIZ can set the colors of nodes and edges, or the thickness of edges for showing different weights. For example, in the TASKLET use case, the graph represents the overlay network consisting of the devices (as nodes), and their connection to a broker (as edges). The color of a node represents the device type (blue: providers, green: consumers, red: brokers). In the WSN scenario [1], the network view can also show multiple underlay topologies consisting of the devices and their physical communication links.

Metric Component: The metric component processes messages events with new metric values. Each such event provides the metric name, a numeric value, and a timestamp of the data point. The *metric view* C shows the evolution of one or more metric values in a combined x-y-plot. The x-axis shows the time (according to the timestamp values) and the y-axis shows the value per metric. To summarize the metric view allows to show multiple metric plots at once.

CFM Component: The CFM component receives events about the model and the context or system configurations and notifies the *CFM view* D accordingly. The model is typically

received only once when the system starts. At runtime, the CFM component gets the system context from the MR and the planned system configurations from the AL. The CFM view shows the configuration options of the system as attributed feature diagram together with the currently selected features and their attribute values. The view provides an aggregated, centralized perspective of the system compared to the detailed network view. This allows to monitor the reconfiguration decisions of the AL according to context changes.

Goal Component: The goal component receives the available performance goals from the AL and sends events about changed weights back to the AL. The corresponding view component is the *performance goal control panel* Ⓔ. It shows the available performance goals with weights for each goal. In combination with the network, metric, and CFM views, the goal view allows to assess how well and how quick a defined system goal is met by the adaptation of the SAS. The user may also adjust the weights of performance goals at runtime to explore how the SAS reacts. To sum up, this component allows to explore the reconfiguration behavior of an AL and the resulting system states under changing performance goals.

VI. Evaluation

This section evaluates CoalaViz concerning the challenges Traceability (C1), Extensibility (C2), and Responsiveness (C3), stated in Section II. Traceability and responsiveness is evaluated qualitatively given the capabilities and architecture of CoalaViz. Responsiveness is evaluated quantitatively by running CoalaViz with an JSON event stream on a laptop with an i5-5257U CPU and 8GB of memory.

C1 Traceability: C1 is concerned with the traceability of different aspects of the inspected self-adaptive pervasive communication system. This includes the system state of the network, metrics, the reconfiguration space including the current configuration, and the performance goals. The system state of the network can be viewed using the network component and the connected network view Ⓑ. It shows nodes and edges and can be styled to show different node or edge types. Thus, a changing network topology can be tracked visually. The metric view shows the current value of multiple nonfunctional metrics as well as a history of each value Ⓒ. Concerning the context and the system configuration, CoalaViz is able to show CFMs including the current system configuration in its CFM view Ⓓ. The performance goal panel shows the nonfunctional system goals and their weights and allows to change these weights at runtime Ⓔ. This allows to assess the influence of the adjusted goal on the adaptation decisions.

In summary, we qualitatively evaluated the traceability of CoalaViz for the three use cases presented in Section II and found that Ⓐ, Ⓑ, Ⓒ, Ⓓ and Ⓔ contributes in analyzing the system and context configuration esp., since adaptations affect the performance of the system tremendously. For instance, using CoalaViz, we can monitor how adaptations influence throughput and deadline misses in Tasklet, energy consumption in WSN and response time in CEP. Summarizing, the implemented views of CoalaViz provide continuous insights into different aspects of a self-adaptive pervasive communication system and make adaptation decisions traceable.

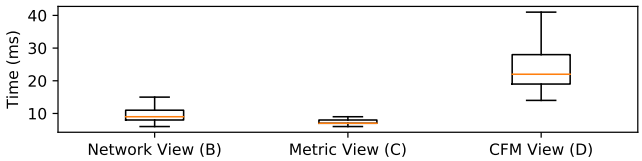

Fig. 3: Responsiveness per view with artificial data.

C2 Extensibility: The goal of the second challenge was to make sure, that our solution is easily extensible and changeable. CoalaViz is a web application using the Vaadin framework[1]. This allowed us to implement the backend code in Java while for the frontend standard JavaScript libraries could be used. The socket-bases JSON interface makes sure that CoalaViz can easily be made compatible with different ALs and MRs. This allows us to use it with all three use cases although the Tasklet system is simulated using OMNeT++ [10], the WSN scenario uses Simonstrator [11], and the CEP case uses a custom implementation [21]. The view components, in particular, are abstractions in the backend for using different views in the frontend. These components translate incoming events by calling according JavaScript methods of the frontend. For the network view, we use VisJS, while the metric view is implemented using ChartJS[1]. The CFM view is a customized implementation while the goal view consists of standard UI elements of Vaadin. Due to the modular design, views can easily be exchanged.

C3 Responsiveness: For measuring the responsiveness, we log the timestamps right after the socket on the sending side is flushed and when the JavaScript code for changing something in the view was executed. As a first step, the responsiveness of the network, metric, and CFM views are evaluated separately with artificial data. In all three cases, we simulate JSON requests with events for 5 minutes real time. For evaluating the network view, we use a poisson distribution with an average arrival time of 1 event per second. The poisson distribution is a commonly used model to describe inter arrival times of incoming or departing data entities. By that, we add randomly nodes and edges to the system. In case of the metric and CFM view, we send one event per second, as metrics and context changes typically happen on a more regular basis. Finally, we send two metric values at once each second, and a random configuration / context respectively. The results of the three evaluation runs can be seen in Figure 3. The figure shows that the network and metric views, which are implemented using standard open source components, perform better than the custom-built CFM view. In fact, the CFM view is rendered as image, which makes it slower compared to the JavaScript views. Additionally, the whole image is (re-) rendered even for small changes. As this is a low load scenario, this basically shows the best performance we can get for each view.

More realistic, Figure 4 shows the playback of one hour simulated time in the Simonstrator in the WSN case. Here,

[1]https://vaadin.com/, http://visjs.org/, https://www.chartjs.org/

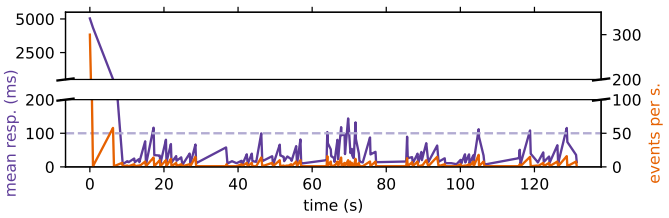

Fig. 4: Mean of the responsiveness over time in WSN case.

all different events in the three views are measured. The stream resulted in 2.5 minutes of runtime in CoalaViz. As we can see, the first 302 events have a high latency as in this period initial nodes and edges are added to the system. After this warmup phase, we end up with a median of 55 ms for the responsiveness. In UI research, 100 ms are considered as a limit for instant reaction [22]. The dashed line in the figure indicates this value. Most reactions in the UI can be considered responsive. Thus, we consider CoalaViz to be sufficiently responsive to provide traceability given this condensed event stream and the hardware the evaluation was executed on.

Limitations: CoalaViz only shows an aggregated or global view of a system rather than a local entity-based view. Also, the CFM view requires a CFM-based AL. If this is not the case, however, the other views could still be used. Filtering or clustering capabilities for the network and metric view could further improve the traceability. Finally, it has to be investigated how scalable the approach is considering larger systems.

VII. Conclusion

In this paper, we present CoalaViz, a modular, web-based, and reusable visualization platform that allows tracing the reconfiguration behavior of self-adaptive pervasive communication systems while interactively adjusting the system's optimization goals. While we employ Coala [1] as a sample adaptation logic, we highlight that CoalaViz provides a flexible interface that builds on one or more JSON-based event streams over network sockets. Hence, CoalaViz can flexibly be utilized in conjunction with various kinds of self-adaptive systems. Further, we show the applicability of CoalaViz using three use cases: Tasklet [6], adaptive WSNs [1], CEP [21]. To this end, we have devised reusable components for emitting CoalaViz events from the network simulators OMNeT++ [10] and Simonstrator [11].

In the future, we will utilize CoalaViz to investigate further use cases and mechanisms. In particular, we envision the development of specialized mechanisms that shall be tailored towards specific optimization goals for certain scenarios. Recent research has demonstrated that such network mechanisms can be automatically generated in a data-driven fashion, such as with Remy [23] for TCP congestion control. We note that in order to operate a network with changing context and variable optimization goals, multiple instances of these proactively generated algorithms would be required. The SAS would then have to dynamically switch between these alternatives, depending on the context features and the optimization goal. We foresee that such a system design demands tools to analyze the overall system's behavior, and we consider CoalaViz to be an important step in this direction. Further, we plan to extend the frontend of CoalaViz, e.g., with capabilities to capture and replay events, which shall allow to save, reproduce, and demonstrate SAS experiments.

Acknowledgment

This work has been funded by the German Research Foundation (DFG) as part of projects A1, A4, C1, and C2 of the Collaborative Research Center (CRC) 1053–MAKI.

References

[1] M. Weckesser, R. Kluge, M. Pfannemüller, M. Matthé, A. Schürr, and C. Becker, "Optimal Reconfiguration of Dynamic Software Product Lines Based on Performance-Influence Models," in *SPLC '18*, 2018.
[2] J. O. Kephart and D. M. Chess, "The vision of autonomic computing," *Computer*, vol. 36, no. 1, 2003.
[3] S. O. Hallsteinsen, M. Hinchey, S. Park, and K. Schmid, "Dynamic software product lines," *IEEE Computer*, vol. 41, no. 4, 2008.
[4] K. Saller, M. Lochau, and I. Reimund, "Context-aware DSPLs: model-based runtime adaptation for resource-constrained systems," in *SPLC Workshops '13*, 2013.
[5] N. Siegmund, A. Grebhahn, S. Apel, and C. Kästner, "Performance-influence models for highly configurable systems," in *ESEC/FSE'15*, 2015.
[6] D. Schafer, J. Edinger, J. M. Paluska, S. VanSyckel, and C. Becker, "Tasklets: better than best-effort computing," in *ICCCN '16*, 2016.
[7] J. Edinger, D. Schäfer, C. Krupitzer, V. Raychoudhury, and C. Becker, "Fault-avoidance strategies for context-aware schedulers in pervasive computing systems," in *PerCom '17*, 2017.
[8] Y. Wang, "Topology control for wireless sensor networks," *Sig Comm Tech*, 2008.
[9] R. Kluge, M. Stein, G. Varró, A. Schürr, M. Hollick, and M. Mühlhäuser, "A Systematic Approach to Constructing Families of Incremental Topology Control Algorithms Using Graph Transformation," *SoSyM*, 2017.
[10] A. Varga and R. Hornig, "An overview of the OMNeT++ simulation environment," in *SimuTools '08*, 2008.
[11] B. Richerzhagen, D. Stingl, J. Rückert, and R. Steinmetz, "Simonstrator: simulation and prototyping platform for distributed mobile applications," in *SimuTools '15*, 2015.
[12] W. A. Higashino, M. A. M. Capretz, and L. F. Bittencourt, "CEPSim: A Simulator for Cloud-Based Complex Event Processing," in *Big Data '15*, 2015.
[13] F. Starks, T. P. Plagemann, and S. Kristiansen, "DCEP-Sim: An Open Simulation Framework for Distributed CEP," in *DEBS '17*, 2017.
[14] D. O'Keeffe, T. Salonidis, and P. Pietzuch, "Frontier: Resilient edge processing for the internet of things," *VLDB '18*, vol. 11, 2018.
[15] C. Buschmann, D. Pfisterer, S. Fischer, S. P. Fekete, and A. Kröller, "SpyGlass: a wireless sensor network visualizer," *SIGBED Review*, 2005.
[16] Y. Yang, P. Xia, L. Huang, Q. Zhou, Y. Xu, and X. Li, "SNAMP: A multi-sniffer and multi-view visualization platform for wireless sensor networks," in *ICIEA '06*, 2006.
[17] Y. Hu, D. Li, X. He, T. Sun, and Y. Han, "The implementation of wireless sensor network visualization platform based on wetland monitoring," in *ICINIS '09*, 2009.
[18] D. Estrin, M. Handley, J. Heidemann, S. McCanne, Y. Xu, and H. Yu, "Network visualization with nam, the vint network animator," *Computer*, vol. 33, no. 11, 2000.
[19] H. Hartmann and T. Trew, "Using Feature Diagrams with Context Variability to Model Multiple Product Lines for Software Supply Chains," in *SPLC '08*, 2008.
[20] K. Czarnecki, T. Bednasch, P. Unger, and U. W. Eisenecker, "Generative Programming for Embedded Software: An Industrial Experience Report," in *GPCE '02*, 2002.
[21] M. Luthra, B. Koldehofe, P. Weisenburger, G. Salvaneschi, and R. Arif, "TCEP: Adapting to Dynamic User Environments by Enabling Transitions Between Operator Placement Mechanisms," in *DEBS '18*, 2018.
[22] J. Nielsen, *Usability engineering*. AP Professional, 1993.
[23] K. Winstein and H. Balakrishnan, "TCP Ex Machina: Computer-generated Congestion Control," *ACM SIGCOMM*, vol. 43, no. 4, 2013.