Demo: Visualizing Adaptation Decisions in Pervasive Communication Systems

Martin Pfannemüller*, Janick Edinger*, Markus Weckesser†, Roland Kluge†,
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*—Pervasive systems enhance our environment with various services under a wide range of contextual conditions. Hence, as a user's context may change over time, pervasive systems shall be able to reconfigure themselves. While such reconfigurations are typically difficult to conceive, a system designer might be interested in their triggers and execution. To this end, we demonstrate CoalaViz, a tool for making adaptation decisions in pervasive communication systems traceable. Our demonstration shows the capabilities of CoalaViz based on the Tasklet system, a distributed computing system for task offloading. The offloading mechanism can be reconfigured in terms of the scheduling strategy. We show the use of CoalaViz and how the Tasklet system behaves in different contextual situations and during reconfigurations based on changing goals.**

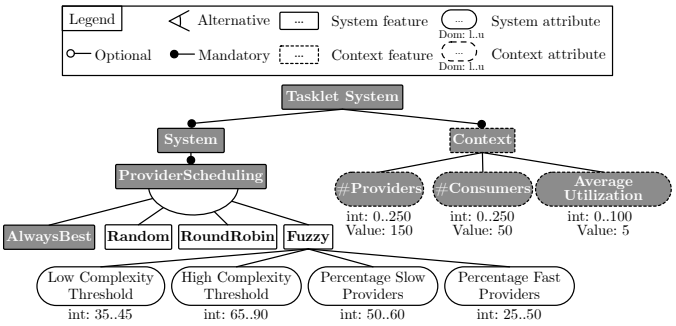

Fig. 1. Tasklet context feature model. Grey: current configuration.

I. Introduction

Pervasive (communication) systems need to operate in multiple continuously changing environmental contexts. Such systems provide a large number of configuration options to reflect these changes. However, tracing the current system status as well as the different system configurations is not yet provided in a simple manner. Due to the large configuration space, a possibility for tracing the current system state is required. Additionally, as of the context-dependency, nonfunctional properties change over time as well. In current work, we propose CoalaViz, a traceability platform for self-adaptive pervasive communication systems [1]. This paper accompanies this work by demonstrating the capabilities of CoalaViz in the Tasklet distributed computing system.

This paper continues with a brief introduction of the capabilities of CoalaViz in Section II. Then, Section III presents the Tasklet use case and a subset of its reconfiguration options. Section IV presents the timeline of the demo. Finally, Section V concludes this paper.

II. Achieving Traceability using CoalaViz

CoalaViz is a tool for making adaptation decisions in self-adaptive pervasive communication systems traceable [1]. Figure 2 shows a screenshot of the CoalaViz frontend, which allows to (i) visualize the network state as graph-based view (network view, **A**) (ii) visualize the system performance over time (metric view, **B**), (iii) visualize the system configuration (configuration view, **C**), and (iv) change weights of performance goals interactively (goal management, **D**).

In the network view, different colors of nodes and edges represent different device and connection types. The metric view on the top right is a plot of nonfunctional metric values over time. The context feature model (CFM) in the configuration view represents the configuration space and the current configuration of the target system (gray selection) [2]. Configuration options are shown in a tree structure, with cross-tree constraints restricting the configuration space. A configuration is a set of selected features that fulfills all constraints. The CFM used in this demo is shown in Figure 1 which specifies the configuration space of the Tasklet system. Besides system features in the left-hand side tree branch and context features in the right, features can have additional attributes such as the context attributes for the number of providers and consumers in the system. The context branch is selected according to changes in the system's environment which then can trigger a reconfiguration of the system branch. Finally, the goal management panel can be used to set the weights for achieving nonfunctional goals such as matching deadlines or achieving a high speed.

CoalaViz provides a socket-based interface for sending and receiving JSON messages to and from an actual system or a simulator and its adaptation logic. Thus, as a technical requirement, a system connected to CoalaViz must send conforming JSON messages. Each view supports different event messages. The backend of CoalaViz decouples the

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

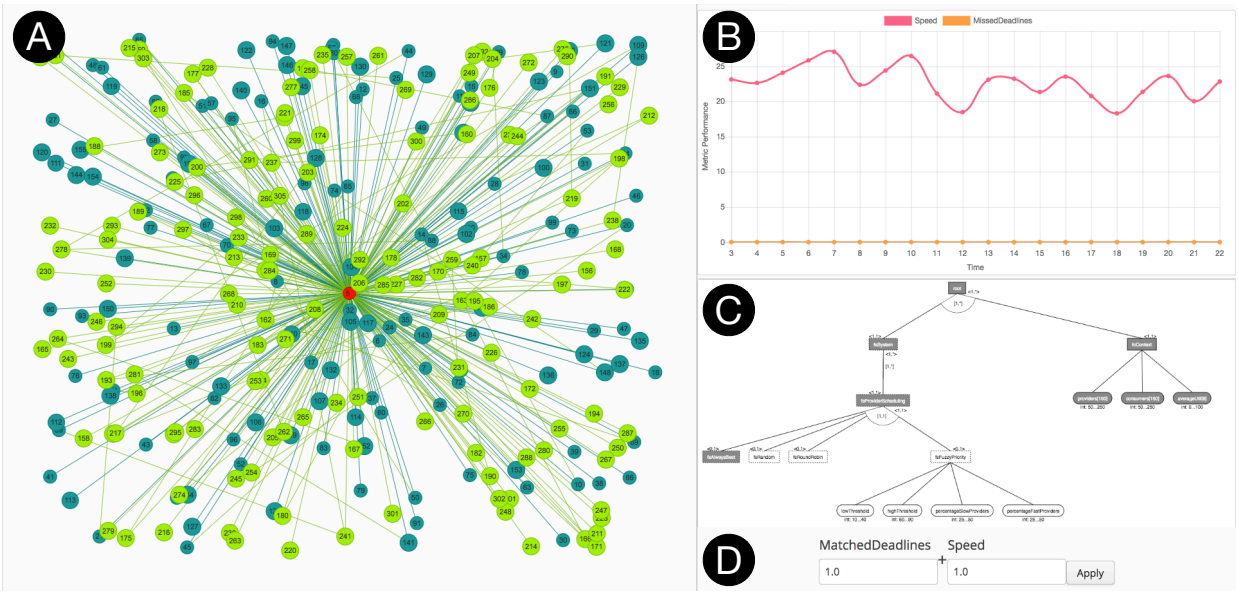

Fig. 2. Screenshot of CoalaViz running Tasklet use case. Red: Broker; Light Green: Consumer; Dark Green: Provider

messages from the actual view implementations. The network view supports adding and removing nodes and edges as well as modifying the visualization of the node and edge properties (e.g., color, line stroke). The metric view can receive events with a metric type, a metric value, and a timestamp. Further, the CFM view gets the current context of the system and its configuration as events. Finally, the goal management view gets the possible performance goals and sends back the weights for each goal when changed.

In our prototype, we use the approach from previous work for planning the adaptation decisions [3]. There, we learn the performance influence of configurations on nonfunctional metrics. Each nonfunctional metric can be weighted by the user. Thus, changing the weights of the performance goals changes the reconfiguration behavior. Taking all views together allows to trace the behavior of the connected system and to change performance goal weights at runtime. This facilitates traceability in self-adaptive pervasive communication systems.

CoalaViz is implemented as a Java-based web application using Vaadin and the JavaScript libraries vis.js (for the network view) and Chart.js (for the metric view)[1]. The CFM visualization is a custom implementation.

III. Use Case: Tasklets

We demonstrate CoalaViz using the Tasklet [4] system, a distributed system for computation offloading.

Concepts: The Tasklet system provides an abstraction for computation and allows to seamlessly exchange computational workload among multiple heterogeneous devices. A device may use different resources to offload *Tasklets*: (i) remote resources, (ii) closed units of computation, or (iii) any device with available computational capacity. The system consists of three types of entities: (i) a *(resource) provider* runs the Tasklet execution environment and offers its computation resources to other devices; (ii) a *(resource) consumer* runs applications that create and sends Tasklets to providers for execution; (iii) a *(resource) broker* coordinates required and provided resources of registered consumes and providers by assigning Tasklets to providers.

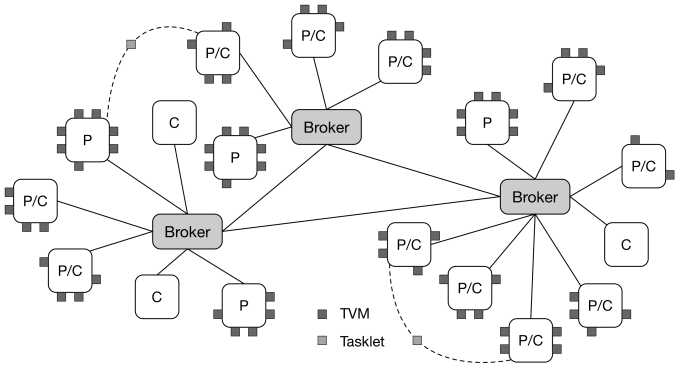

Fig. 3. Tasklet Overlay Network

A provider initially registers at one resource broker and publishes its capabilities (e.g., processing speed). Therefore, a broker has a global view of all registered providers. When a consumer sends a resource request (i.e., a Tasklet to be executed) to a broker, the broker selects a suitable provider for execution (*scheduling*). The Tasklet system currently does not support queuing. Thus, only idle providers get selected. Figure 3 shows the topology of a Tasklet overlay network.

Complexity-Aware Scheduling: In this demo, we consider a real-time offloading application that relies on Tasklet. The offloaded Tasklets have different complexities and need to be executed before a given deadline that is similar for all Tasklets (e.g., due to response time requirements). Thus, computationally intensive Tasklets are prone to missing the deadline, especially if it is scheduled to a slow provider. As a solution, the brokers can always select the fastest

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

available provider for execution. While this approach works well in underutilized environments with sufficiently many fast providers, it reaches its limits when the resource utilization increases. Eventually, slow provider inevitably get selected for executing computationally intensive Tasklets. As a countermeasure, brokers may reserve powerful providers for complex Tasklets and schedule less complex Tasklets to slow providers. This scheduling strategy leaves powerful resource providers unused if no pending complex Tasklets exist. To this end, the scheduling decision boils down to a trade-off between meeting deadlines and optimizing the performance of the overall system. To deal with the trade-off, the TASKLET system allows to switch strategies at runtime between the following four scheduling strategies.

Using *AlwaysBest* scheduling, the broker always selects the fastest available provider, independent of the Tasklet complexity. This ensures that powerful providers have the least idle times possible and the overall performance of the system is maximized. Slow providers are only selected if all fast providers are already in use.

Using *Fuzzy* scheduling, the broker takes the complexity of Tasklets into account and distinguishes between low, medium and high complexity. Similarly, providers are grouped into slow, medium and fast providers. During scheduling, low (medium/high) complexity Tasklets get allocated only to slow (medium/fast) providers. Within a group, the broker always selects the fastest available provider.

Using *Random* scheduling, the broker randomly selects an available provider. Using *Round Robin* scheduling, the broker selects providers in a round robin manner.

Strategy Selection: To ensure that the deadlines of all Tasklets are fulfilled, the broker needs to decide when to switch the strategy. The proportions of slow, medium, and fast providers, as well as the number and complexity distribution of the Tasklets influences this decision. In an underutilized system with many providers and few Tasklets, the *AlwaysBest* strategy might perform best and fulfill all deadlines. When using the *Fuzzy* strategy, the broker also needs to set the lower and upper bounds for assigning each Tasklet and provider to one of the three categories. Accurate settings can improve the overall system performance while avoiding missing deadline.

Figure 1 shows all configuration options (incl. the monitored context attributes) and the current configuration highlighted in gray. The attribute values on the right indicate a low utilization of the system because the ratio of the number of providers and consumers is high ($\frac{150}{50}$). Therefore, *AlwaysBest* scheduling is selected here.

IV. Demonstration Details

In this demonstration, we execute the TASKLET system in the OMNET++ simulator [5]. For determining reconfiguration decisions, we use COALA [3]. A timeline of the demo is shown in Figure 4. The figure shows how the TASKLET system adapts based on changed goal weights. As different configurations have different influence on the matched deadlines and the overall speed of the TASKLET execution, changing the weights

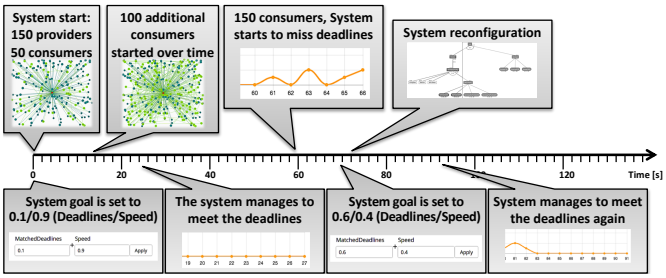

Fig. 4. Timeline of the demonstration

influences the reconfiguration behavior. Figure 1 shows the context and system configuration at the beginning of the demonstration. Additionally, the user weights the nonfunctional goals deadlines and speed by 10% and 90%. Thus, the broker uses the *AlwaysBest* strategy and always selects the fastest providers available. This results in a high average performance of the Tasklet executions and the providers can meet all deadlines. Over time, 100 additional consumers join the network using Poisson-based inter-arrival time. Therefore, after 40 s, 150 consumers causing a significant additional load for the 150 providers. At some point, all fast providers are busy and the broker has to select slow providers for execution. For complex Tasklets, the increased utilization results in deadline misses after ca. 60 s. The system operator changes the weights of the performance goals to prioritize meeting the deadlines over optimizing the execution speed by setting them to 60% and 40%. Based on the changed weights, COALA decides to change the scheduling strategy from *AlwaysBest* to *Fuzzy*. As a result, the system manages to meet the deadlines again with almost no deadline misses after around 90 s. A video of the demonstration is available at: https://vimeo.com/302803426.

V. Conclusion

In this demonstration, we show how COALAVIZ allows to inspect the behavior of a pervasive communication system using an implementation of the TASKLET distributed computing system in OMNET++ as example. Still, due to the JSON-based socket interface, COALAVIZ can be used in conjunction with various simulators and real systems, and its modular design allows to exchange view components easily.

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. Pfannemüller, M. Weckesser, R. Kluge, J. Edinger, M. Luthra, R. Klose, C. Becker, and A. Schürr, "CoalaViz: Supporting Traceability of Adaptation Decisions in Pervasive Communication Systems," in *PerCom Workshops '19*, 2019, accepted.

[2] K. Saller, M. Lochau, and I. Reimund, "Context-aware DSPLs: model-based runtime adaptation for resource-constrained systems," in *SPLC Workshops '13*, 2013.

[3] 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.

[4] D. Schafer, J. Edinger, J. M. Paluska, S. VanSyckel, and C. Becker, "Tasklets: better than best-effort computing," in *ICCCN '16*, 2016.

[5] A. Varga and R. Hornig, "An overview of the omnet++ simulation environment," in *SimuTools '08*, 2008.