Using Reference Attribute Grammar-Controlled Rewriting for Runtime Resource Management

Johannes Mey[1], René Schöne[1], Daniel Langner[1], and Christoff Bürger[2]

[1] Chair of Software Technology, TU Dresden, Germany
johannes.mey@tu-dresden.de, rene.schoene@tu-dresden.de,
daniel.langner@mailbox.tu-dresden.de
[2] Department of Computer Science, Faculty of Engineering, Lund University, Sweden
christoff.burger@cs.lth.se

Abstract. To make distributed systems resource aware and adaptive, they can be modeled as self-adaptive systems. Such systems have a view of their own state and context, which can be represented by a model that is continuously updated and analyzed at runtime. However, such analyses need to be concise and efficient to allow large models and high adaptation rates. To achieve this, we apply *reference attribute grammar controlled rewriting* to implement the runtime model of a distributed task-scheduling case study for energy optimization.

1 Modeling Self-Adaptive Systems

Self-adaptive systems [1] are used to cope with changing requirements and contextual information at runtime. Furthermore, they need to provide short response times while maintaining low resource consumption and a convenient way to specify their internal state and algorithms. Another challenge is the high update rate of their context information[2]. Self-adaptive systems usually employ a feedback loop, e.g. *MAPE-K* [3], and have representation of their context, e.g. a runtime model. The *models@run.time* approach [4] uses models not only during development but also as a data representation at runtime. It has been shown that auto tuning and resource awareness can save energy in Big Data scenarios [5]. Our use case is a small, yet scalable, distributed Big Data scenario on a network of embedded devices. We use a self-adaptive system built around a runtime model, which is easy to specify, and can run algorithms efficiently with regard to response time. It employs grammar-based modeling and analysis to deal with frequent model updates efficiently.

2 Attribute Grammars for Runtime Models

Our solution uses *reference attribute grammars* [6] (*RAGs*) as its underlying technology. RAGs originate from the area of compiler construction to describe abstract syntax trees of program code. However, their intrinsic advantage – incremental evaluation – fits well to the described problems. Using RAGs, we

describe the structure of runtime models with a context free grammar that is well-suited for hierarchical structures. Non-hierarchical parts of a model can be described as well using *reference attributes* forming arbitrary overlay graphs. The analysis of runtime data is done using *attributes*, which are defined declaratively for specific non-terminals, achieving a concise specification.

However, the aforementioned updates of the runtime model hinder incremental evaluation commonly used in RAG systems since they rewrite the AST and therefore invalidate all previously performed analyses. This work uses a novel approach called *RAG-controlled rewriting* (*RACR*) [7], which treats model changes as term rewrites [8]. This enables the tracking of dynamic dependencies between attributes and the model, and thus incremental evaluation across model changes. Therefore, constantly changing self-adaptive systems can be analyzed efficiently, thus allowing a more frequent analysis and larger model sizes.

3 Runtime Models with RACR

RACR works in a three-phase process. In the first phase, a context free grammar with inheritance describing the runtime model is specified, like the one depicted below for our case study presented in section 4. Terminals are in lowercase and non-terminals in title case optionally suffixed by an alternative name and a colon.

```
Root ::= scheduler switching CompositeWorker
AbstractWorker ::= id state timestamp
CompositeWorker:AbstractWorker ::= AbstractWorker*
Switch:CompositeWorker ::=
Worker:AbstractWorker ::= devicetype Queue:Request*
Request ::= id size deadline dispatchtime
```

The second phase involves the attribution, that is the specification of attributes for certain non-terminals. Below, the `schedule` attribute defined for `Root` is listed. It reads the terminal `scheduler` and invokes an attribute to find an insertion position. All attributes and rewrites are written Scheme, using the API functions `ast-child`, `create-ast` and `att-value` to get a certain child of a AST node, create a new AST node and call an attribute, respectively.

```
(ag-rule schedule
  (Root (lambda (n time work-id load-size deadline)
    (att-value (ast-child 'scheduler n) n
               time work-id load-size deadline))))
```

At runtime, the system is performing rewrites and attribute evaluations in turns. Rewrites, like the one shown below, change the model and invalidate cached attribute values. If those attributes are called, RACR ensures their re-evaluation.

```
(rewrite-insert
  (ast-child 'Queue worker) ;list node to insert into
  index                     ;position of insertion
  (create-ast 'Request (list id size deadline #f)))
                            ;subtree for the new request
```

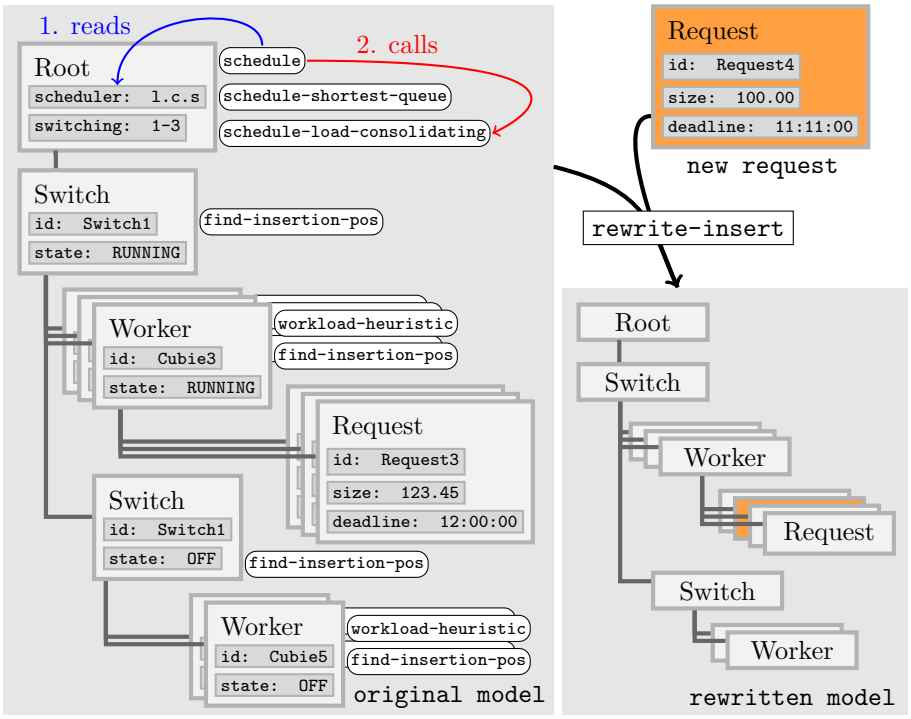

Fig. 1: Scheduler selection and scheduling of a request. Terminals are contained in non-terminal boxes, some selected attributes are attached. Terminals not relevant for the example are left out, `l.c.s` is the load consolidating scheduler.

4 A Case Study

To investigate the applicability of RACR to self-adaptive systems, we implemented the distributed indexing of Wikipedia pages using a network of system-on-a-chip workers [9]. These are Cubieboards having a 1 Ghz CPU, 1GB of RAM, running Linux, and are connected to a master via switches and Ethernet links. Every worker and switch is powered by a USB charging hub, which enables the switching and energy measurement of individual devices.

We developed an adaptation and two scheduling strategies, each written with RACR. The adaptation strategy controls the number of powered on workers. Our solution checks periodically for idle workers to be switched off while ensuring a minimum number of online workers to secure stable performance in case of load peaks. A round-robin scheduler always chooses the shortest queue, and a load-consolidating scheduler tries to use as few workers as possible.

The solution is evaluated in a small-scale case study, whose structure and the scheduling of a request on it are depicted in Figure 1. To analyze our approach's scalability, we developed a simulation environment that simulates the execution of

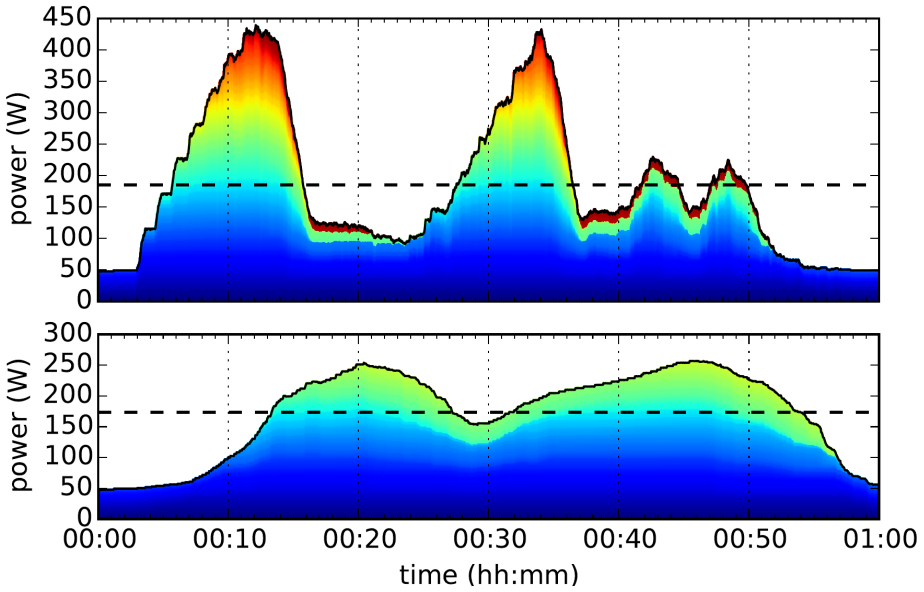

Fig. 2: Power consumption when scheduling a workload with a round-robin (top) and a load-consolidating (bottom) scheduler. The dashed line shows the average power consumption.

tasks and their associated energy consumption based on models we acquired from our physical use case. The simulated use case comprises 315 workers connected via 63 switches fulfilling 4,600 requests within an hour. Figure 2 shows the power consumption of the two scheduling strategies, using a different color for each worker. The load-consolidating scheduler (shown at the bottom) uses 6.4% less energy than the round-robin scheduler while using less workers.

As the model can be modified with rewrites, it permits addition and removal of workers and the exchange of scheduling and adaptation strategies during runtime.

5 Conclusion and Outlook

In this work, we showed the applicability of *RAG-controlled rewriting* for self-adaptive systems in a distributed data processing use case. In addition, we plan to conduct more case studies exploring the scalability and adding heterogeneity. Another case study involves an extended runtime model with included software structure, in which the presented concepts are applied to iteratively transform the model to code describing a constraint problem. First measurements show very short response times for every transformation after the initial one.

In conclusion, *RACR* enables incremental evaluation for large runtime models with high update rates, hence offering opportunities for the usage in adaptive, resource aware systems, such as Big Data systems.

Acknowledgments

This work is partly supported by the German Research Foundation (DFG) in the SFB 912 "Highly Adaptive Energy-Efficient Computing", the cluster of excellence cfaed, and within the Research Training Group "Role-based Software Infrastructures for continuous-context-sensitive Systems" (GRK 1907).

References

1. R. Lemos *et al.*, "Software Engineering for Self-Adaptive Systems: A Second Research Roadmap," in *Software Engineering for Self-Adaptive Systems II*, ser. LNCS, R. Lemos, H. Giese, H. A. Müller, and M. Shaw, Eds. Springer, 2013, vol. 7475.
2. Y. Chen, J. Dunfield, M. A. Hammer, and U. A. Acar, "Implicit Self-adjusting Computation for Purely Functional Programs," in *ICFP*. New York, NY, USA: ACM, 2011.
3. J. Kephart and D. Chess, "The vision of autonomic computing," *Computer*, vol. 36, no. 1, Jan. 2003.
4. G. Blair, N. Bencomo, and R. B. France, "Models@run.time," *Computer*, vol. 42, no. 10, Oct. 2009.
5. S. Götz, T. Ilsche, J. Cardoso, J. Spillner, T. Kissinger, U. Aßmann, W. Lehner, W. E. Nagel, and A. Schill, "Energy-Efficient Databases Using Sweet Spot Frequencies," in *Proceedings of the 2014 IEEE/ACM 7th International Conference on Utility and Cloud Computing*. IEEE Computer Society, 2014.
6. G. Hedin, "Reference attributed grammars," *Informatica (Slovenia)*, vol. 24, no. 3, 2000.
7. C. Bürger, "Reference Attribute Grammar Controlled Graph Rewriting: Motivation & Overview," in *SLE*. ACM, 2015.
8. F. Baader and T. Nipkow, *Term rewriting and all that*. Cambridge university press, 1999.
9. C. Bürger, J. Mey, R. Schöne, S. Karol, and D. Langner, "Using Reference Attribute Grammar-Controlled Rewriting for Energy Auto-Tuning," in *10th International Workshop on Models@run.time*, Ottawa, Canada, Sep. 2015.