Using Spreadsheet-defined Rules for Reasoning in Self-Adaptive Systems

Christian Krupitzer, Guido Drechsel, Deborah Mateja, Alina Pollkläsener, Florian Schrage, Timo Sturm, Aleksandar Tomasovic, and Christian Becker
Chair of Information Systems II, University of Mannheim, Germany
Email: christian.krupitzer@uni-mannheim.de, {gdrechse, dmateja, apollkla, fschrage, timsturm, atomasov}@mail.uni-mannheim.de, christian.becker@uni-mannheim.de

***Abstract*—Using rules to capture adaptation knowledge is a common approach for self-adaptive systems. Rule-based reasoning, i.e., using rules to analyze and plan adaptations, has several advantages: (i) it is easy to implement, (ii) it offers fast reasoning, and (iii) it works on resource-spare systems as historical knowledge is not required. Hence, the needed computational power is low and it perfectly suits systems in the pervasive IoT domain. However, the codification of rules poses a challenge to the system design. Existing approaches often require a specific syntax or programming language. Additionally, some approaches force the developer to customize the reasoning mechanism, hence, to re-implement parts of the reasoning. To address these shortcomings, we propose a reusable approach for rule-based reasoning in this paper. Rules can be defined in a spreadsheet without the need to neither learn a syntax nor implement a single line of code. We evaluate the benefits of our approach in two case studies conducted by Master students as well as a quantitative evaluation.**

I. Introduction

Self-adaptive systems enable autonomous adjustments of system parameters or structure as a reaction to changes in their resources or the environment [1]. For reasoning about necessary changes, these systems implement a feedback loop, the so called Monitor-Analyze-Plan-Execute-Knowledge (MAPE-K) loop [2]. Reasoning, i.e., analyzing the system and its environment and plan adaptations, is responsible to detect the need for adaptations and plan them accordingly. In literature, four approaches for reasoning – used in isolation or combined – can be found: (i) rules, (ii) goals, (iii) models, or (iv) utility functions [1].

Using rules for reasoning offers several advantages. First, the typically used event-action-condition (ECA) rules are easy to implement in if-else constructions. Second, these ECA rules omit historical data and do not need large computation. Hence, they can be evaluated fast and respectively lead to fast decisions. Third, as a consequence of the previously mentioned advantage, rule-based reasoning is resource-efficient. Especially the last advantage makes rule-based reasoning a suitable approach for the IoT domain.

However, existing approaches often require developers to learn or use a specific syntax or programming language. Additionally, some approaches force the developer to customize the reasoning mechanism, hence, to re-implement parts of the reasoning. In this paper, we propose a reusable approach for rule-based reasoning which offers a definition of rules in a spreadsheet. Therefore, developers neither have to learn a syntax nor implement a single line of code as the implementation is fully reusable. Our contributions are threefold. First, we present our system model for a reusable rule-based reasoner. Second, we present an easy to use and reusable implementation in Java, which can be integrated in adaptive systems, e.g., pervasive IoT systems. Third, to support our claims for reusability and usability, we evaluate our approach in three case studies as well as a quantitative evaluation.

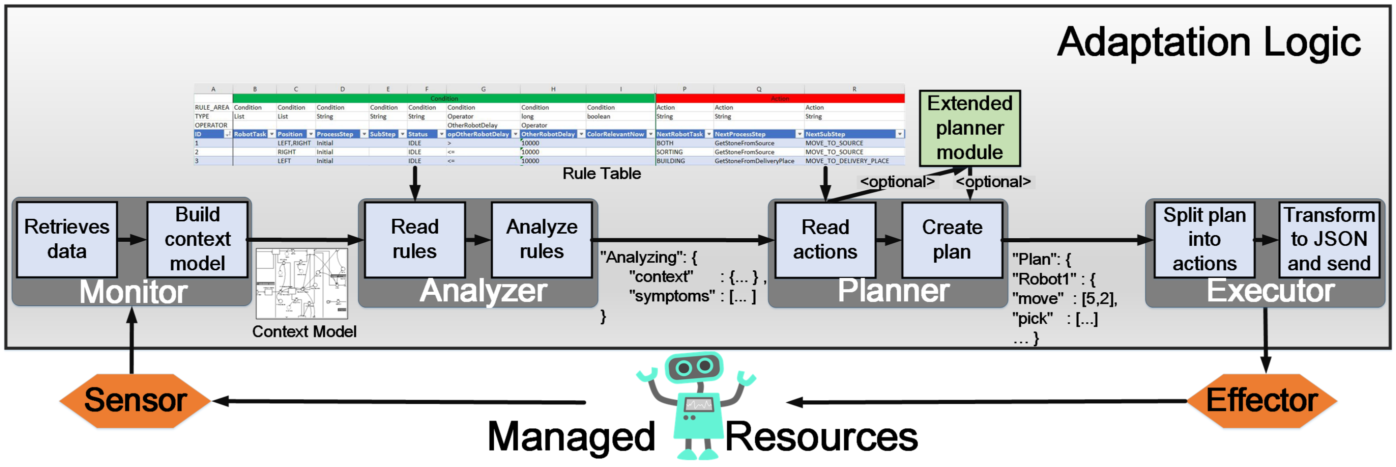
The remainder of the paper is structured as follows: Section II analyzes existing approaches for rule-based reasoning. Section III introduces our system model. Section IV explains the implementation of the reusable rule-based reasoning. Section V presents the case studies conducted by students and a quantitative analysis of the approach. Last, Section VI concludes this paper with a summary and future work.

II. Related Work

The literature in the domain of self-adaptive systems offers different approaches for rule-based reasoning. De Virgilio, Torlone, and Houben present a rule-based approach that supports the automatic adaptation of content delivery in Web Information Systems [3]. They make use of adaptation rules to compare profiles - in terms of sub elements of a given context (e.g., device, network, user) and their characteristics - with possible configurations of Web applications. Thereby, they differentiate between three levels of how to build a Web application: (i) content, (ii) navigation, and (iii) presentation. Based on rules, they determine required adaptations to the content presentation. The adaptation of content delivery was implemented in FAWIS (Flexible Adaptation of Web-based Information Systems). The tool includes an Adaptation Designer with a user interface that allows the specification of adaptation rules and configurations in a graphical way. The profiles can be expressed in different languages. The storing the adaptation rules is not explained.

Dynamo [4] is an assertion-based solution for self-healing BPEL processes. For defining monitoring and recovery activities, it provides specific languages. Dynamo makes use of the JBoss Rules Engine for logic and data separation, knowledge centralization, and tool integration [4].

Similar to the previous approach, Denaro, Pezzè, and Tosithis also follow the idea to enforce self-healing policies using a rules engine within their approach [5]. Their self-adaptive solution is able to detect requested and provided services autonomously and dynamically adapts the application of the client accordingly.



A	B	C	D	E	F	G	H	I	P	Q	R
	Condition								Action		
RULE_AREA	Condition	Condition	Condition	Condition	Condition	Condition	Condition	Condition	Action	Action	Action
TYPE	List	List	String	String	String	Operator	long	boolean	String	String	String
OPERATOR						OtherRobotDelay	Operator				
ID	RobotTask	Position	ProcessStep	SubStep	Status	opOtherRobotDelay	OtherRobotDelay	ColorRelevantNow	NextRobotTask	NextProcessStep	NextSubStep
1		LEFT,RIGHT	Initial		IDLE	>	10000		BOTH	GetStoneFromSource	MOVE_TO_SOURCE
2		RIGHT	Initial		IDLE	<=	10000		SORTING	GetStoneFromSource	MOVE_TO_SOURCE
3		LEFT	Initial		IDLE	<=	10000		BUILDING	GetStoneFromDeliveryPlace	MOVE_TO_DELIVERY_PLACE

```
"Analyzing": {
    "context"    : {... } ,
    "symptoms" : [... ]
}
```

```
"Plan": {
"Robot1" : {
"move"   : [5,2],
"pick"    : [...]
... }
```

Figure 1. Process for reasoning mapped to the elements of the adaptation logic. Note: The knowledge base is omitted.

Liu and Parashar present DIOS++, an infrastructure that enables rule-based control of distributed scientific applications [6]. Based on a distributed rules engine, DIOS++ is able to define, deploy, and execute rules in the area of autonomic application management. Similar to this paper, the rules within the DIOS++ framework are defined as a combination of conditions and actions in terms of the managed sensors and actuators. A difference is the alternative action part, that is executed when not every condition is fulfilled. The specific technology of the rules engine is not stated.

In [7], the authors present a rule-based framework for business activity management. The Rule-Based Business Activity Management combines policy definition and business activity management to a unified framework. This information is aggregated into business rules and integrated for evaluating conditions and determine future actions of the managed business applications. The business rules are represented in XML based on the rule type of reaction rules.

However, all of these approaches force developers to use a specific syntax or programming language. In this paper, we abstract from this knowledge by offering developers the possibility to specify the rules graphically using a spreadsheet which also avoids the need to learn a syntax.

III. System Model

This section presents our system model and the process for reasoning. Figure 1 shows this process. It is implemented within the adaptation logic of a self-adaptive system. Such systems are divided into the managed resources, which can be adapted, and the adaptation logic, which controls the adaptation of the managed resources [1] (see Figure 1). The managed resources can be any hardware or software that shall act autonomously in the system. The adaptation logic implements a MAPE-K loop. In the following, we describe how the adaptation logic works using our rule-based approach.

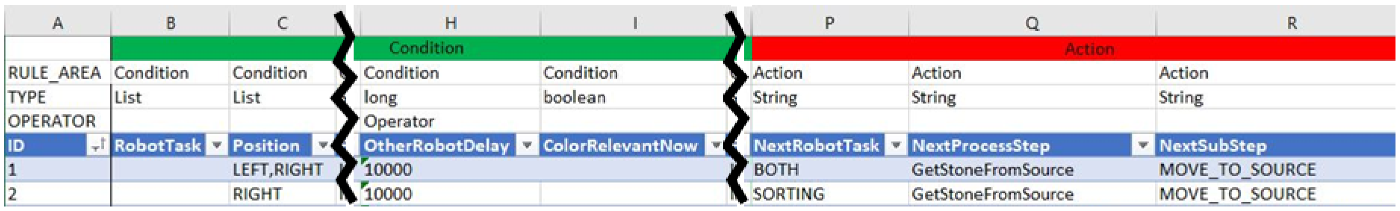
The *sensor* senses the managed resources. The *monitor* collects and aggregates this data. The result is a key-value based context model that contains all information about the current state of the managed resources and its environment. The *analyzer* retrieves and evaluates the context model to determine necessary adjustments of the managed resources. This is based on our rules engine that evaluates the provided context data against adaptation rules. The adaptation rules are defined in a customizable spreadsheet. This rules table is represented by an Excel file in which conditions (system state) and actions (adaptation commands) of rules are specified: If the conditions of a rule match the context data, the engine returns the commands defined in the rule's actions. Depending on the rule design, the action commands are either first interpreted by the planner or executed by the managed resources directly. The *planner* evaluates the analyzing results and decides which actions need to be performed and when. It depends on the rule design whether the results of the analyzer are already actions that can be forwarded to the managed resources or whether they need further interpretation. The planner forwards every statement that the managed resources should execute to the executor. The *executor* triggers the adaptation in the managed resources. In general, it forwards the results of the planner to the corresponding managed resources through the *effectors* which use actuators to adapt parameters or the structure of the resources. A *knowledge* component stores relevant information such as the generated context models, analyzing results, plans, or the rules table. All MAPE components can access the data and add new information.

In this paper, we focus on a centralized adaptation logic. It supervises all managed resources and their environment and adapts the parameters of the managed resources if necessary. However, our approach supports a decentralized setting, too. As this requires additional conflict handling if different parts of the system adapt contrarily, the application of our approach in a distributed adaptation logic is part of future work.

IV. Implementation

Based on the system model from Section III, this section describes the implementation of our rule-based reasoner in Java. The implementation is based on *Apache POI*[1] as well as the *EasyRules*[2] rules engine and offers implementations for the analyzer and the planner of the adaptation logic.

[1]https://poi.apache.org/
[2]https://github.com/j-easy/easy-rules



A	B	C	H	I	P	Q	R
			Condition			Action	
RULE_AREA	Condition	Condition	Condition	Condition	Action	Action	Action
TYPE	List	List	long	boolean	String	String	String
OPERATOR			Operator				
ID	RobotTask	Position	OtherRobotDelay	ColorRelevantNow	NextRobotTask	NextProcessStep	NextSubStep
1		LEFT,RIGHT	10000		BOTH	GetStoneFromSource	MOVE_TO_SOURCE
2		RIGHT	10000		SORTING	GetStoneFromSource	MOVE_TO_SOURCE

Figure 2. Excerpt of a rules table. Note: Some conditions are omitted (indicated by the jagged lines).

The *analyzer* evaluates the rules and determines the required adjustments. Therefore, it obtains the current context model from the monitor. It evaluates each object of the context to determine the necessary adjustments: For each single context object, the *AnalyzingRulesEngine* module evaluates the data and returns the required adjustments as symptoms. It uses the *ApachePOIExcelReader* module which handles the reading of rules (see Section IV-A). For analyzing the rules, the analyzer creates a rule representation of the class *AnalyzingRule* for every entry in the rule spreadsheet. Section IV-B describes this. These newly created rule objects are compared against the context data. If a rule fires - i.e., the context data matches the conditions - its actions are added to a list of required adjustments. The analyzer sends this list and the received context data to the *planner* for planning the execution of the required adjustments. In the following, we describe the structure of the rules table and the analysis of rules.

A. Definition of Rules

The *ApachePOIExcelReader* imports the rules table into the *analyzer*. Therefore, it reads the rules set and stores every rule including the meta data column. Figure 2 shows an extract of a rules table with the condition area (column B - I) and the action area (column P - R). The table includes two rules. The first five rows shown in Figure 2 form the meta data which is crucial for the interpretation of the specified rules. In the following, we describe its elements.

Condition/Action: The first table row serves orientation purposes only. It indicates whether a column contains a rule condition or an action which is performed if all related conditions are satisfied.

Rule Area: The rule area fields are evaluated to determine whether a column represents a condition or an action. Based on the information derived this way, the rules engine can identify the appropriate treatment for the distinct values.

Type: Indicates the data type of the rules' values. The specific evaluation process of the distinct values depends on this information. The implemented rules engine supports the standard Java data types `boolean`, `int`, `long`, `String`, and, additionally, Lists of Strings, ranges of integers, and ranges of times.

Operator: An additional operator is possible for conditions of data type `int` or `long`. It contains numerical comparison operators which define how to evaluate the related condition's value. If no operator is specified, it is compared whether the context data is equal to the value.

Name: Defines the name of the distinct conditions and actions. The data inserted here needs to conform the keys of the corresponding values in the context model. Otherwise, a correct matching cannot be ensured during rule evaluation.

Rules: Rules are represented as rows of the rules table. The first column of every rule contains a unique id. It is followed by the rule's conditions or empty fields, if the rule does not depend on all of the attributes. Adjacent to this, the rule's actions are specified. These are triggered if all conditions are fulfilled. As for the conditions, the fields of the action columns may remain empty if irrelevant for / not applicable to the specified rule.

B. Rule-based Reasoning

The *AnalyzingRule* class implements a rule representation with *EasyRules*, an open source rules engine. In the following, we describe the implementation of handling rules.

For each rule, an object of *AnalyzingRule* is initialized using the context data and the rule retrieved from the rules table. The *evaluate* method serves the purpose to compare the context data to the rule values as appropriate for the distinct data types of the conditions to determine whether the specified conditions are met in the current scenario. If all conditions of the rule are fulfilled, the *execute* method is performed. It creates a list of the specific rule's actions and sets the boolean class variable *'executed'* to *'true'* indicating that the rule has been triggered. The analyzer combines the list of actions from every successful rule and forwards them to the planner.

The analyzer does not return the list of actions directly. Rather, it sends the set of triggered rules to the planner. The planner offers a default implementation. This implementation takes instructions specified by the developer in the rules table, adds the receiving managed resources, and sends the plan to the executor. If further planning is necessary, the developer can easily customize the planner using predefined interfaces.

V. Evaluation

Developers can use our rule-based reasoning approach by just defining a rules table in Excel (see Section IV-A) and adjusting their monitor, so that it returns a context model that can be interpreted by the rule-based reasoner. If the managed resources already send only the context variables' information in the format specified in the rules table, the monitor just needs to forward this information, hence, the developer does not have to implement monitoring functionality. The module supports both, structure as well as parameter adaptation [1].

For showing the feasibility of our approach, we performed two evaluations. First, we asked 3 teams of 2-3 master students each to implement self-adaptive systems using the FESAS framework [8] and our rule-based reasoning approach. These students attended a lecture on adaptive systems, however, the did not have any implementation experience in implementing adaptive systems. We interviewed them to determine the usability of our approach and analyzed the resulting systems regarding the reusability of our rule-based reasoning approach. Second, we performed a quantitative analysis of the system to determine the performance of our rules engine. In the following, we present the results of both evaluations.

A. *Qualitative Evaluation*

This section describes the three applications from the IoT domain: (i) the iCasa Verde system, (ii) the smart warehouse control, and (iii) the collaborative industry robots. All have been developed in student projects. This included the implementation of the managed resources as well as the adaptation logic using FESAS and our rule-based reasoning approach. In the following, we describe the implementation of these systems.

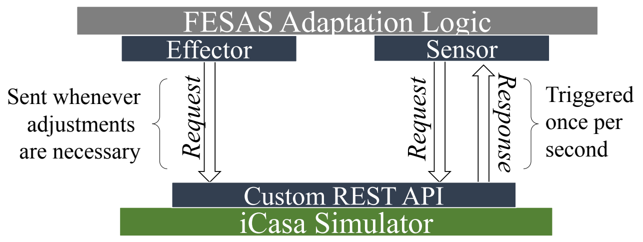


Figure 3. Abstract overview of the architecture of the iCasa Verde system.

1) iCasa Verde: The iCasa Verde implementation project incorporates the digital home execution platform and simulation tool iCasa[3]. It provides developers with the possibility to create and configure an arbitrary number of given smart home devices, e.g., lights, humidity or presence sensors. In the context of iCasa Verde, the framework is used to model a self-adaptive greenhouse: Various sensors (e.g., thermometer) and actuators (e.g., heater) pursue the goal of creating an optimal environment for distinct plants in various different zones. Consequently, the five distinct tasks are performed by iCasa Verde: temperature, illuminance and humidity adjustment, as well as gas level supervision and burglary prevention.

The following three components connect the adaptation logic and the simulation: A custom REST API for iCasa provides access to data measured by the sensors and the possibility to change the state of devices. The sensor requests the data measured in the iCasa simulation using the REST API. The effector receives a list of necessary adaptations and triggers changes in the simulation using the REST API. The communication is carried out in form of HTTP Requests transferring JSON Objects between the components. Figure 3 depicts the process and dependencies.

2) Smart Warehouse Control: A warehouse has many different requirements for storing items. The items might require different temperatures. Fast identification of items is the key for successful warehouse management. Further, in case of a fire, a fast reaction is necessary to limit the impact of the fire. Additionally, burglary detection is important for warehouses without 24x7 operation.

[3]http://adeleresearchgroup.github.io/iCasa/snapshot/index.html

As a second use case, the students implemented a smart warehouse control system. The implementation is based on the iCasa simulation, too, and uses the REST API presented in Section V-A1. The warehouse enables to control different zones of a warehouse, e.g., adjust the temperature differently. Additionally, systems for fire protection, burglary prevention, as well as intelligent light control are integrated. Figure 4 shows a screenshot of the iCasa simulation running the warehouse scenario. There, the warehouse is divided into two zones with different temperatures.

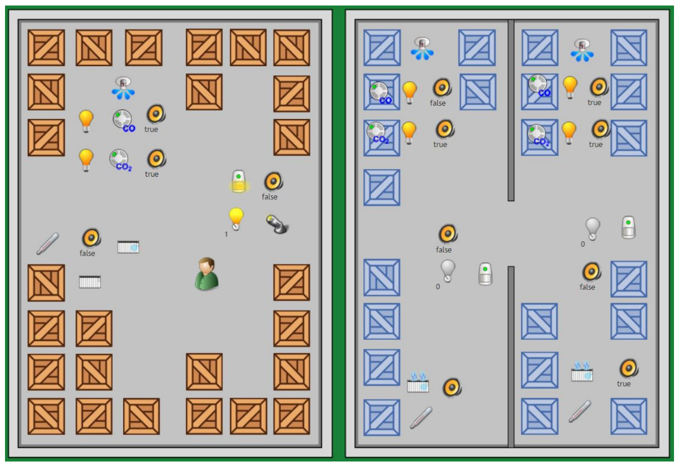


Figure 4. The iCasa simulation running the warehouse scenario.

3) Collaborative Industry Robots: As third system, the students implemented an Industry 4.0 scenario using LEGO Mindstorms robots and the accompanying leJOS[4] system. It aims at constructing an intermediate product, using two coordinated self-adaptive industry robots. Different colored LEGO bricks serve as raw material for the robots. The output of the robots' construction process is a two dimensional assembly of LEGO bricks. Figure 5 shows the system. A video can be found on YouTube[5].

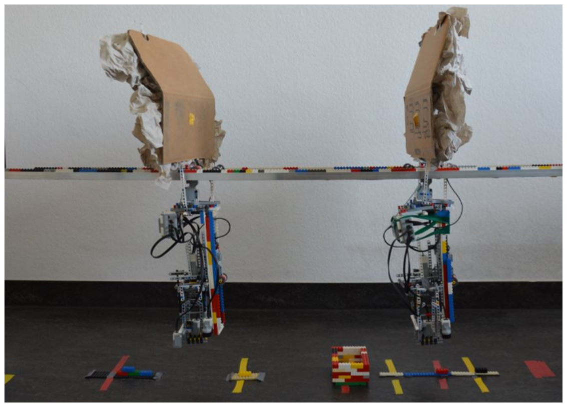


Figure 5. The collaborative industry robots.

The robots arrange bricks according to a specified building plan, determining the colors of the product's components. This building process consists of two colluding sub-processes: (i)

[4]http://www.lejos.org/index.php

[5]The video of the robots can be found at: https://youtu.be/XH9tFyKVOU8

sorting the bricks according to their color as well as (ii) assembling the product.

As soon as the sorting robot delivered a brick to the construction area, the building robot picks up the brick and assembles it according to the given product plan. As long as both robots are working correctly, each of them is responsible for either sorting or building. If one robot fails – detected by missing heartbeats – the other robot performs both tasks. This building process as well as the failure handling is entirely supervised and coordinated by the adaptation logic.

4) Use Case Analysis: In both systems, the students were able to reuse our rule-based reasoning approach without any adjustment of the rule reasoning. They just needed to write their own rules tables and adjust monitoring. Additionally, the robot scenario needed an extension in the planning process as here the planning needs to interpret historical information, e.g., the last action performed by a robot. However, this information might be codified in the rules, too, with the disadvantage of having a more complex rule set.

In interviews, the students confirmed the ease of use of our approach w.r.t. reasoning for adaptation as the students only had to write rules and did not have to implement the analysis and planning functionality. This speeds up the development process as well as reduces complexity. Further, it offers a fast approach to rapid prototyping as rules can be tested easily without large configuration setup. Only the robots use case needed minor extensions of the planning functionality.

B. Quantitative Evaluation

Additionally to the qualitative evaluation, we performed a quantitative evaluation for measuring the performance of our rule-based reasoning approach for large data sets. The evaluation was executed on a common Windows laptop with an Intel Core i5 CPU and 8 GB RAM using Java 1.8.

According to an evaluation of Lightstone [9], typical middleware products provide between 384 and 1200 configuration/registry parameters. With these parameters, all the different aspects of the system are specified. To manage those systems within a self-adaptive system, the number of parameters of the system would be equal to the variables for the adaptation. Each of these variables would be represented by a condition in the rule set. This leads to an equivalent range of conditions in the rule set for the evaluation, e.g., 400 parameters leads to 400 conditions in the rule set. The largest evaluation was chosen with a rule set of 1000 conditions and 1000 rules. Additionally, we performed test runs with 400 – as the lower boundary of [9] – and 700 as well as smaller rule sets with 100, 50, and 15 conditions in alignment with the use cases presented in Section V-A. Next to the upper limit of 1000 rules per rule set, every number of conditions was evaluated with 500, 100, and 50 rules.

To ensure a reliable result, each evaluation setting was executed 20 times. For simulating numerous managed resources calling the adaptation logic, we ran 100 threads per execution of a simulation run simultaneously. To eliminate outliers, the upper and the lower 5% of the records were removed. We measured the time for reading the rule set and processing a request for reasoning (see Table I). In the following, we present and discuss the results of the measurements.

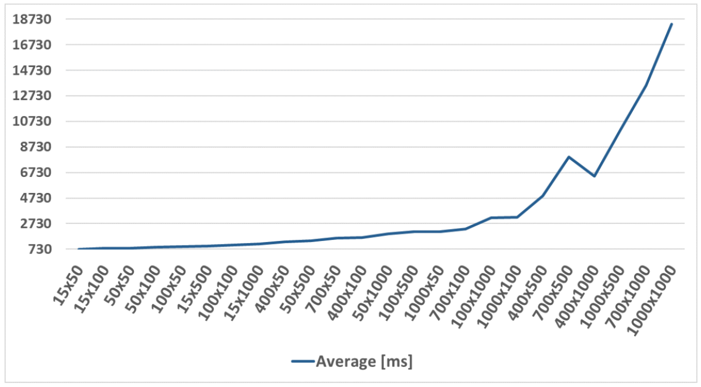


Figure 6. Average time for reading of the rule sets sorted by the number of cells (conditions x rules).

1) Performance for Reading the Rule set: Figure 6 shows an overview of the average time for reading the rule set. The average times are sorted by the number of cells (conditions x rules) of the rule set. Except for a decline at the data point 400x1000, the reading time of the rule sets is nearly constantly rising comparing the number of cells to be read. It is evident that the reading time increases sharply if the number of cells increases towards the largest setting.

Comparing the settings with a similar numbers of cells (e.g., 100x50 & 50x100 or 700x500 & 400x1000) it is evident that the reading of rule sets with more conditions is more time consuming than the one with less conditions and more rules. Comparing the larger settings, the decline of the runtime at 400x1000 in Figure 6 can therefore be traced back to the smaller number of conditions. An aspect that strengthened this behavior is that for the data types `int` and `long` an additional operator column is added. Hence, for two out of seven possible conditions the number of columns that need to be read is two, increasing the number of columns by 28.6% on average.

Table I. EVALUATION RESULTS. RULE SET SETTINGS ARE DEFINED IN CONDITIONS X RULES. ALL VALUES ARE MEASURED IN MILLISECONDS. AV = AVERAGE (ROUNDED), SD = STANDARD DEVIATION.

Rule Set	Reading a Rule Set				Evaluating a Rule Set			
	Min	**Max**	**Av.**	**SD**	**Min**	**Max**	**Av.**	**SD**
15x50	717	766	736	11.5	16	67	38	10.5
15x100	764	806	788	12.4	47	113	74	14.5
50x50	802	851	820	15.3	15	45	25	8.0
50x100	872	927	898	14.8	15	50	25	9.2
100x50	902	1222	944	32.3	44	111	59	13.1
15x500	937	987	960	13.3	31	79	41	10.7
100x100	1018	1145	1066	30.4	46	104	66	14.6
15x1000	1112	1222	1151	28.7	37	94	59	12.8
400x50	1244	1338	1290	27.8	43	104	63	14.8
50x500	1328	1436	1373	28.3	84	170	106	18.6
700x50	1544	1646	1585	30.9	37	101	64	16.3
400x100	1607	1721	1652	35.5	31	90	48	13.2
50x1000	1810	2044	1941	62.6	413	600	459	41.8
100x500	2015	2254	2094	52.0	348	526	385	32.7
1000x50	1953	2228	2095	69.1	132	227	153	16.1
700x100	2207	2418	2292	69.8	351	589	397	54.0
100x1000	2958	3472	3189	156.5	201	432	260	58.6
1000x100	3041	3427	3225	116.7	216	493	284	70.8
400x500	4703	5085	4900	132.1	384	569	428	38.2
700x500	7112	8297	7932	317.3	352	522	385	28.7
400x1000	6068	6948	6457	262.1	335	481	360	21.5
1000x500	8631	11050	10069	804.1	357	613	430	82.0
700x1000	12428	14858	13536	579.0	385	666	479	93.0
1000x1000	17175	20164	18355	919.0	833	1643	1275	197.7

2) Performance for Processing Requests for Reasoning: Figure 7 shows the average runtime of each request of the rules engine. Compared to the runtime of the reading, the trend of the needed time related to the number of rules and conditions appears less clear. There is a positive trend between the runtime and the number of rules, but the number of conditions within an equal number of rules does not seem to increase the execution time, e.g., the evaluation of 15 conditions and 500 rules compared to 100 conditions and 500 rules. The largest evaluation, with 1000 conditions and 1000 rules, was running significantly longer than the other evaluations.

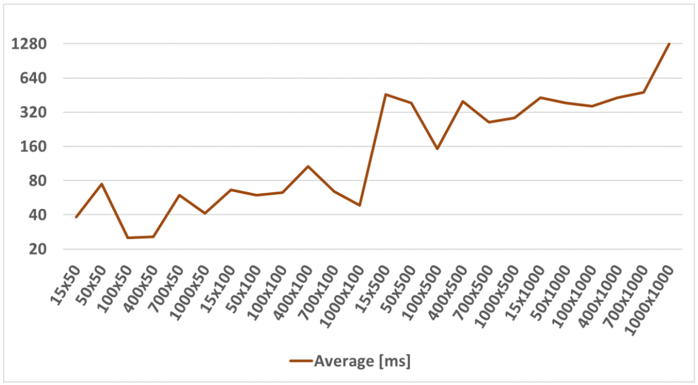


Figure 7. Average runtime of an request of the rules engine. The y-axis uses a logarithmic scale of base two to facilitate visualization of the variations of smaller settings.

Focusing on the rule set with 15 conditions and 50 rules – this represents the use cases from Section V-A – the average runtime of the rule evaluation was 38.41ms. This is below the adaptation logic's frequency of requesting data at the managed resources, which was 1000ms for iCasa Verde and the smart warehouse and 500 ms for the robots. Consequently, the rules engine is appropriate to run these scenarios as the reasoning for adaptation is much faster than the monitoring frequency. Hence, the rules engine fits pervasive IoT systems as they have similar adaptation requirements. Analyzing the runtime of the rule evaluation for settings in large systems with at least 400 conditions and at least 500 rules (cf. [9]), the setting with 700x500 reached the fastest average results with 260.36ms. The largest setting of 1000x1000 achieved results of 1275.93ms. In general, the processing time ranges from 16ms (for the 15x50 scenario) up to 1643ms (for the 1000x1000 scenario). This might be not enough in large real time systems, however, depending on the application, the processing time is sufficient. Especially, the system provides a good trade-off between usability and performance.

VI. Conclusion

This paper provides the first results of our rule-based reasoning approach for self-adaptive systems. The approach offers the specification of adaptation rules in a spreadsheet. This enables to split the responsibilities for the adaptation implementation: As no code has to be written, designers with domain knowledge can write the rule sets even if they lack development knowledge. The code for reasoning can be reused without any changes. Developers just have to adjust the monitoring and in some cases extend the planning procedure.

The evaluation in three projects conducted by Master students who implemented IoT systems showed the ease of use. Further, the students used the reasoning approach without any modifications of the code just by defining rules in a spreadsheet. Additionally, we evaluated the performance of our approach. In general, the processing time ranges from 16ms (15x50 setting) up to 1643ms (1000x1000 setting). Hence, for smaller systems as the use case systems, the performance is fast enough. However, for larger real time systems, this might be an issue. This might be improved using associative rules (cf. [10]). Nevertheless, the system provides a good trade-off between usability and performance. The performance for reading the rule set can be improved. So far, the system forces a trade-off between reading the rule set once – requiring much memory – versus reading the rules on demand which requires time especially for large rule sets.

This paper presented the first prototype of our rule-based reasoning approach. As future work, we plan to compare the results of our evaluation to related work. Additionally, in this paper, we presented an analysis of systems with a centralized adaptation logic. As control in systems of the pervasive IoT domain is often distributed, we need to analyze the performance of our rule-based reasoning approach in decentralized settings. Therefore, the application of our approach in a distributed adaptation logic is part of future work. So far, the definition of rules is limited to using Excel. Besides the common usage of Excel in offices around the world, it uses a proprietary format. Hence, for future work, we plan to offer alternative spreadsheet formats for the definition of rules.

Acknowledgment

We would like to thank Germán Vega for his support w.r.t. iCasa and all students, especially Steffen Jacobs and Christoph Löffel, for their implementation effort.

References

[1] C. Krupitzer, F. M. Roth, S. VanSyckel, G. Schiele, and C. Becker, "A survey on engineering approaches for self-adaptive systems," *Pervasive and Mobile Computing*, vol. 17, no. Part B, pp. 184–206, 2015.

[2] J. O. Kephart and D. M. Chess, "The Vision of Autonomic Computing," *IEEE Computer*, vol. 36, no. 1, pp. 41–50, jan 2003.

[3] R. De Virgilio, R. Torlone, and G.-J. Houben, "Rule-based Adaptation of Web Information Systems," *World Wide Web*, vol. 10, no. 4, pp. 443–470, 2007.

[4] L. Baresi, S. Guinea, and L. Pasquale, "Self-healing BPEL processes with Dynamo and the JBoss rule engine," in *Proc. ESSPE*, 2007, pp. 11–20.

[5] G. Denaro, M. Pezze, and D. Tosi, "SHIWS: A Self-Healing Integrator for Web Services," in *Proc. ICSE*, 2007, pp. 55–56.

[6] H. Liu and M. Parashar, "DIOS++: A Framework for Rule-Based Autonomic Management of Distributed Scientific Applications," in *LNCS 2790*. Springer, 2003, pp. 66–73.

[7] J. J. Jeng, D. Flaxer, and S. Kapoor., "RuleBAM: a rule-based framework for business activity management," in *Proc. SCC*, 2004, pp. 262–270.

[8] C. Krupitzer, F. M. Roth, C. Becker, M. Weckesser, M. Lochau, and A. Schurr, "FESAS IDE: An Integrated Development Environment for Autonomic Computing," in *Proc. ICAC*, 2016, pp. 15–24.

[9] S. Lightstone, "Foundations of Autonomic Computing Development," in *Proc. EASe*, 2007, pp. 163–171.

[10] J. Hall and R. Iqbal, "CoMPES: A Command Messaging Service for IoT Policy Enforcement in a Heterogeneous Network," in *Proc. IoTDI*, 2017, pp. 37–44.