Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 109C (2017) 1158–1163

Procedia Computer Science

www.elsevier.com/locate/procedia

International Workshop on Adaptive Technology (WAT 2017)

A middleware architecture for adaptive devices

Paulo Roberto Massa Cereda[a,*], João José Neto[a]

[a] *Escola Politécnica, Departamento de Engenharia de Computação e Sistemas Digitais, Universidade de São Paulo*
Av. Prof. Luciano Gualberto, s/n, Travessa 3, 158, CEP: 05508-900 – São Paulo, SP – Brasil

Abstract

The intuitive notion of an adaptive layer enclosing an underlying rule-driven device, thus making it adaptive, does not specify how coupling between the underlying rule-driven device and the adaptive mechanism should happen from an architectural point of view. In this paper, as a means to address component heterogeneity and exposure, we present a middleware architecture for adaptive devices, acting as a support layer between the underlying rule-driven device and the adaptive mechanism. The proposed middleware spontaneously provides aggregation and composition services to both components, making them interoperable and working as a single cohesive unit.

Peer-review under responsibility of the Conference Program Chairs.

Keywords: adaptive devices, middleware, adaptivity

1. Introduction

Adaptive rule-driven devices are formally defined as a tuple containing the underlying, potentially non-adaptive, rule-driven device, and the adaptive mechanism[1]. The former provides the current contextual behaviour, whilst the latter empowers modifications to such behaviour over time, in order to accommodate planned yet unexpected contexts[2]. This formulation creates the intuitive notion of an adaptive layer enclosing an underlying rule-driven device, thus making it adaptive, as seen in Figure 1.

This intuitive notion, however, does not specify how coupling between the underlying rule-driven device and the adaptive mechanism should happen from an architectural point of view. Rule-driven devices vary greatly in their own formulation, leading to a heterogeneous environment. Furthermore, a device should not expose its inner workings to the adaptive mechanism beyond the rule set. A common protocol must exist in order to make components visible and understandable to each other, yet without breaking encapsulation.

In this paper, as a means to address component heterogeneity and exposure, we present a middleware architecture for adaptive devices, acting as a support layer between the underlying rule-driven device and the adaptive mechanism. The proposed middleware spontaneously provides aggregation and composition services to both components, making

* Corresponding author, +55 11 3091-5583.
E-mail addresses: paulo.cereda@usp.br (Paulo Roberto Massa Cereda)., jjneto@usp.br (João José Neto).

Peer-review under responsibility of the Conference Program Chairs.
10.1016/j.procs.2017.05.388

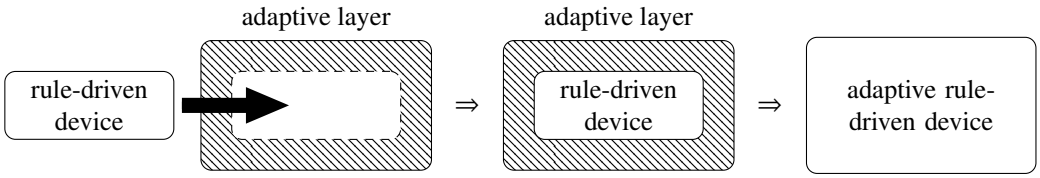

Fig. 1. Adaptive layer enclosing an underlying rule-driven device, making it adaptive.

them interoperable and working as a single cohesive unit. Once the protocol is correctly established, the middleware becomes transparent to the adaptive device.

The remainder of this paper is organized as follows: Section 2 formally introduces the concept of an adaptive rule-driven device, including the underlying rule-driven device and the adaptive mechanism. The middleware architecture is presented in Section 3, with a general overview and further details regarding the operation stages. At last, final remarks, as well as proposals for further studies and directions, are presented in Section 4.

2. Initial concepts

A general adaptive rule-driven device consists of an underlying, potentially non-adaptive, rule-driven device enhanced with an extension, namely the adaptive mechanism, that allows modifications on the current rule set over time, based on history and input stimuli [1]. This section formally introduces the concept of adaptive rule-driven devices.

Definition 1 (rule-driven device)**.** A rule-driven device is defined as $ND = (C, NR, S, c_0, A, NA)$, such that ND is a rule-driven device, C is the set of all possible configurations, $c_0 \in C$ is the initial configuration, S is the set of all possible input stimuli, $\epsilon \in S$, $A \subseteq C$ is the subset of all accepting configurations (respectively, $F = C - A$ is the subset of all rejecting configurations), NA is the set of all possible output stimuli of ND as a side effect of rule applications, $\epsilon \in NA$, and NR is the set of rules defining ND as a relation $NR \subseteq C \times S \times C \times NA$. □

Definition 2 (rule)**.** A rule $r \in NR$ is defined as $r = (c_i, s, c_j, z)$, $c_i, c_j \in C$, $s \in S$ and $z \in NA$, indicating that, as response to a stimulus s, r changes the current configuration c_i to c_j, processes s and generates z as output [1]. A rule $r = (c_i, s, c_j, z)$ is said to be compatible with the current configuration c if and only if $c_i = c$ and s is either empty or equals the current input stimulus; in this case, the application of a rule r moves the device to a configuration c_j, denoted by $c_i \Rightarrow^s c_j$, and adds z to the output stream. □

Definition 3 (acceptance of an input stimuli stream by a rule-driven device)**.** An input stimuli stream $w = w_1 w_2 \dots w_n$, $w_k \in S - \{\epsilon\}$, $k = 1, \dots, n$, $n \geq 0$, is accepted by a device ND when $c_0 \Rightarrow^{w_1} c_1 \Rightarrow^{w_2} \dots \Rightarrow^{w_n} c$ (in short, $c_0 \Rightarrow^w c$), and $c \in A$. Respectively, w is rejected by ND when $c \in F$. The language described by a rule-driven device ND is represented by $L(ND) = \{w \in S^* \mid c_0 \Rightarrow^w c, c \in A\}$. □

Definition 4 (adaptive rule-driven device)**.** A rule-driven device $AD = (ND_0, AM)$, such that ND_0 is a device and AM is an adaptive mechanism, is said to be adaptive when, for all operation steps $k \geq 0$ (k is the value of an internal counter T starting in zero and incremented by one each time a non-null adaptive action is executed), AD follows the behaviour of an underlying device ND_k until the start of an operation step $k+1$ triggered by a non-null adaptive action, modifying the current rule set; in short, the execution of a non-null adaptive action in an operation step $k \geq 0$ makes the adaptive device AD evolve from an underlying device ND_k to ND_{k+1}. □

Definition 5 (operation of an adaptive device)**.** An adaptive device AD starts its operation in configuration c_0, with the initial format defined as $AD_0 = (C_0, AR_0, S, c_0, A, NA, BA, AA)$. In step k, an input stimulus move AD to the next configuration and starts the operation step $k + 1$ if and only if a non-adaptive rule is executed; thus, being the device AD in step k, with $AD_k = (C_k, AR_k, S, c_k, A, NA, BA, AA)$, the execution of a non-null adaptive action leads to $AD_{k+1} = (C_{k+1}, AR_{k+1}, S, c_{k+1}, A, NA, BA, AA)$, in which $AD = (ND_0, AM)$ is an adaptive device with a starting underlying device ND_0 and an adaptive mechanism AM, ND_k is an underlying device of AD in an operation step

k, NR_k is the set of non-adaptive rules of ND_k, C_k is the set of all possible configurations for ND in an operation step k, $c_k \in C_k$ is the starting configuration in an operation step k, S is the set of all possible input stimuli of AD, $A \subseteq C$ is the subset of accepting configurations (respectively, $F = C - A$ is the subset of rejecting configurations), BA and AA are sets of adaptive actions (both containing the null action, $\epsilon \in BA \cap AA$), NA, with $\epsilon \in NA$, is the set of all output stimuli of AD as side effect of rule applications, AR_k is the set of adaptive rules defined as a relation $AR_k \subseteq BA \times C \times S \times C \times NA \times AA$, with AR_0 defining the starting behaviour of AD, AR is the set of all possible adaptive rules for AD, NR is the set of all possible underlying non-adaptive rules of AD, and AM is an adaptive mechanism, $AM \subseteq BA \times NR \times AA$, to be applied in an operation step k for each rule in $NR_k \subseteq NR$. □

Definition 6 (adaptive rules)**.** Adaptive rules $ar \in AR_k$ are defined as $ar = (ba, c_i, s, c_j, z, aa)$ indicating that, as response to an input stimulus $s \in S$, ar initially executes the prior adaptive action $ba \in BA$; the execution of ba is cancelled if this action removes ar from AR_k; otherwise, the underlying non-adaptive rule $nr = (c_i, s, c_j, z)$, $nr \in NR_k$ is applied and, finally, the post adaptive action $aa \in AA$ is applied[1]. □

Definition 7 (adaptive function)**.** Adaptive actions may be defined as abstractions named adaptive functions, similar to function calls in programming languages[1]. The specification of an adaptive function must include the following elements: *(a)* a symbolic name, *(b)* formal parameters which will refer to values supplied as arguments, *(c)* variables which will hold values of calls of elementary adaptive actions of inspection, *(d)* generators that refer to new value references on each usage, and *(e)* the body of the function itself. □

Definition 8 (elementary adaptive actions)**.** Three types of elementary adaptive actions are defined in order to perform tests on the rule set or modify existing rules, namely:

- inspection: the elementary action does not modify the current rule set, but allows inspection on such set and querying rules that match a certain pattern. It employs the form ?⟨pattern⟩.
- removal: the elementary action removes rules that match a certain pattern from the current rule set. It employs the form −⟨pattern⟩. If no rule matches the pattern, nothing is done.
- insertion: the elementary action adds a rule that match a certain pattern to the rule set. It employs the form +⟨pattern⟩. If the rule already exists in the rule set, nothing is done.

Such elementary adaptive actions may be used in the body of an adaptive function, including rule patterns that use formal parameters, variables and generators available in the function scope. □

Adaptivity poses as a convenient abstraction mechanism[2,3], as it offers a compact and better organized model representation[4]. Each underlying device covers a specific context at a time[5], whilst evolutions reflect device fitting towards incremental problem solving[6,2,7,8].

3. Middleware for adaptive devices

In general, a middleware is defined as an enabling technology for the development, execution and interaction of applications[9], represented as an abstraction layer standing between components[10,11], handling functionalities and providing support[12,9,11]. Our middleware aims at providing aggregation and composition services to the underlying rule-driven device and the adaptive mechanism, such that the resulting adaptive rule-driven device fully complies with its formal definition. Four key aspects are taken into consideration:

1. *Discoverability:* components need to be located and accessed before being composed[9]. The middleware must monitor the environment for newly deployed components and requests a special token containing the component's format. This format acts like a driver, specifying the component's interface as a means to restrict access to its inner workings.
2. *Context awareness:* the middleware must be aware in terms of attachment and detachment of components[9,10], such that an adaptive device can be transparently composed and decomposed, once both underlying rule-driven device and adaptive mechanism are functional.

3. *Spontaneous management:* this aspect concerns the ability of such middleware to compose components based on their formats independently of explicit requests[9,12]. Once a component is discovered, it automatically becomes eligible to compose an adaptive device.
4. *Autonomous management:* the middleware must control and manage its own services with little to no human intervention[11], making it transparent and pervasive to the environment[9]. Additionally, the support layer must be as much fault tolerant as possible[13,14,15].

The middleware architecture for adaptive devices is introduced in Figure 2. Observe that the middleware contains the minimum set of operations needed to establish a common protocol between the underlying rule-driven device and the adaptive mechanism. Once such protocol is correctly established, the middleware becomes transparent to the resulting adaptive device.

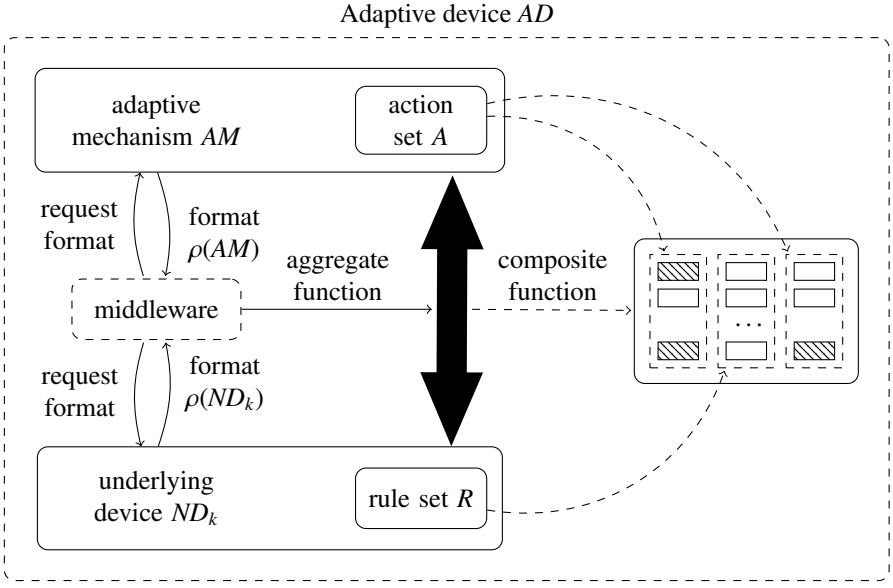

Fig. 2. Middleware architecture for adaptive devices.

According to Figure 2, the middleware detects the attachment of an underlying rule-driven device and the adaptive mechanism and requests both formats. Once the formats are retrieved, the middleware provides an aggregate function and establishes a common protocol between the components. Additionally, a composite function takes the rule and action sets from the underlying device and adaptive mechanism, respectively, based on their formats, and provides a merged set, representing the extended rule set for the newly created adaptive device. The middleware then becomes transparent and acts only as a background monitor for potential structural updates or detachments.

3.1. Requests and formats

The first stage covers component discovery and format requisitions. The middleware monitors the environment for newly deployed components; once a component is found, the middleware asks for a format, i.e, a description on the component's inner workings and its corresponding interface. Additionally, the format also indicates the component type, namely a device or adaptive mechanism. The concept of a format enforces encapsulation, as nothing beyond specification is exposed externally.

Definition 9 (format function)**.** Let there be a format function $\rho\colon ND^{all} \cup AM^{all} \cup AD^{all} \mapsto \Phi$, such that ND^{all}, AM^{all} and AD^{all} are enumerable sets of all possible non-adaptive rule-driven devices, adaptive mechanisms and adaptive

rule-driven devices, and Φ is the set of all formats, i.e, the format function takes any device or adaptive mechanism and returns its corresponding format. □

In order to become eligible for aggregation and composition, a component has to provide its own format function ρ, such that format requests from the middleware are correctly answered. It is important to observe that a resulting adaptive rule-driven device is also eligible for aggregation and composition in a higher abstraction level and thus must provide a format as well; a sensible approach would be $\rho(AD) = \rho(AM)$ for $AD = (ND, AM)$, i.e, an adaptive device would only expose its immediate adaptive mechanism, as a means to preserve a layered hierarchy for multilevel adaptivity [16,17]. However, other approaches are also valid, as the format dictates the device exposure.

3.2. Aggregation and bridging

The second stage covers format aggregation and component bridging. Once the first stage is completed, as format requests to newly attached components (namely, a rule-driven device and an adaptive mechanism) are correctly answered and gathered, the middleware applies an aggregate function on both formats in order to obtain a new format based on both interfaces. The aggregate format dictates how the inner workings of both components should be exposed, as a means to specify information exchange and usage. A bridging protocol is then established by the middleware (represented by a double-headed black arrow in Figure 2), such that both components are made interoperable and working as a single cohesive unit.

Definition 10 (aggregate function)**.** Let there be an aggregate function $\delta\colon \Phi \times \Phi \mapsto \Phi$, such that Φ is the set of all formats, i.e, the aggregate function takes any two valid formats (namely, applications of ρ on a rule-driven device and an adaptive mechanism) and returns an aggregate format based on both interfaces. □

The bridging protocol aims at solving the heterogeneity problem, as it allows direct, cohesive and coherent information exchange between the underlying rule-driven device and the adaptive mechanism, based on the aggregation of their corresponding formats. Observe that the established protocol covers all functionalities available in the surroundings while ensuring proper component encapsulation, since the formats (including resulting formats from applications of δ) explicitly specify the exposure level of their inner workings.

3.3. Set composition

The third and last stage provides set composition. Aside from establishing a direct information exchange between components, the bridging protocol applies a composite function on the rule and action sets from the underlying device and adaptive mechanism, respectively, in order to obtain an extended rule set for the newly created adaptive device. The resulting extended rule set aims at being adherent to the theory presented in Section 2.

Definition 11 (composite function)**.** Let there be a composite function $\Theta\colon R^{all} \times A^{all} \mapsto (A^{all} \cup \{\mu\} \times R^{all} \times A^{all} \cup \{\mu\})$, such that R^{all} and A^{all} are enumerable sets of all possible sets of rules and actions from the underlying rule-driven device and adaptive mechanism, respectively, and μ is a null adaptive action. i.e, the composite function takes both rule and action sets and returns an extended rule set decorated with prior and post adaptive actions (potentially none, as indicated by the null adaptive action μ). □

The extended rule set is decorated with prior and post adaptive actions, as implied by the theory [2,1], including the null adaptive action μ. The composite function provides a straightforward representation of both contextual and adaptive behaviours, simplifying queries and other operations. It is important to observe that the resulting set is a view [18,19,20,21], i.e, the composing elements are simply pointers to their original references; such references are accessible through the bridging protocol. Once the view is properly defined, the middleware becomes transparent and acts only as a background monitor for potential structural updates or detachments, repeating stages whenever needed.

4. Final remarks

This paper presented a middleware architecture for adaptive devices, acting as a support layer between the underlying rule-driven device and the adaptive mechanism, as a means to address component heterogeneity and exposure.

The proposed middleware aims at being adherent to the theory, as well as posing itself as a sensible approach to a cohesive, coherent and generic implementation model. Operations needed to establish a common protocol between the underlying rule-driven device and the adaptive mechanism were kept to a minimum. The middleware design presented in Figure 2 can be further improved in order to cover a broader number of domains and scenarios.

There is an ongoing research on providing a middleware implementation using the Java programming language. We are investigating existing frameworks as basis for format specification and component interoperability, such as a subset of projects from the Apache Foundation, namely Avro[1], Mina[2], Felix[3] and Karaf[4]. Preliminary results look promising, although there are challenges regarding fault tolerant interoperability. Further studies are needed.

A middleware approach presents as a viable solution to address how coupling between the underlying rule-driven device and the adaptive mechanism should happen from an architectural point of view, as well as a proper handling of component heterogeneity and exposure. Additionally, the architecture may be extended through additional modules in order to provide new features, such as persistence, security and quality of service, at almost no sensible cost.

References

1. J. José Neto, Adaptive rule-driven devices: general formulation and case study, in: International Conference on Implementation and Application of Automata, 2001.
2. J. José Neto, Adaptive automata for context -sensitive languages, SIGPLAN Notices 29 (9) (1994) 115–124.
3. P. R. M. Cereda, J. José Neto, Utilizando linguagens de programação orientadas a objetos para codificar programas adaptativos, in: Memórias do IX Workshop de Tecnologia Adaptativa – WTA 2015, São Paulo, 2015, pp. 2–9.
4. P. R. M. Cereda, J. José Neto, A recommendation engine based on adaptive automata, in: Proceedings of the 17th International Conference on Enterprise Information Systems, Vol. 2, Barcelona, Spain, 2015, pp. 594–599.
5. P. R. M. Cereda, J. José Neto, Persistência em dispositivos adaptativos, in: Memórias do VIII Workshop de Tecnologia Adaptativa – WTA 2014, 2014, pp. 120–125.
6. J. José Neto, Um levantamento da evolução da adaptatividade e da tecnologia adaptativa, IEEE Latin America Transactions 5 (2007) 496–505.
7. J. Burg, S. Thomas, Computer science: From abstraction to invention, Tech. rep., Department of Computer Science, Wake Forest University (2003).
8. C. A. Knoblock, Learning hierarchies of abstraction spaces, in: Proceedings of the Sixth International Workshop on Machine Learning, Ithaca, NY, USA, 1989, pp. 241–245.
9. N. Ibrahim, F. Le Mouel, A survey on service composition middleware in pervasive environments, International Journal of Computer Science Issues 1 (2009) 1–12.
10. E. Cecchet, G. Candea, A. Ailamaki, Middleware-based database replication: The gaps between theory and practice, in: Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data, ACM, Vancouver, Canada, 2008, pp. 739–752.
11. K.-D. Kim, P. R. Kumar, Networked Control Systems, Springer London, London, 2010, Ch. The Importance, Design and Implementation of a Middleware for Networked Control Systems, pp. 1–29.
12. Q. T. D. Nguyen, Design and implementation of a distributed middleware for parallel execution of legacy enterprise applications, Master thesis, Department of Computer Science and Software Engineering, Concordia University, Montreal, Quebec, Canada (2008).
13. Y. Yang, Y. Huang, X. Ma, Enabling context-awareness by predicate detection in asynchronous environments, IEEE Transactions on Computers 65 (2) (2016) 522–534.
14. C. Xu, S. C. Cheung, W. K. Chan, C. Ye, Partial constraint checking for context consistency in pervasive computing, ACM Transactions on Software Engineering and Methodology 19 (3) (2010) 9:1–9:61.
15. Y. Huang, Y. Yang, J. Cao, X. Ma, X. Tao, J. Lu, Runtime detection of the concurrency property in asynchronous pervasive computing environments, IEEE Transactions on Parallel and Distributed Systems 23 (4) (2012) 744–750.
16. R. I. Silva Filho, Uma nova formulação algébrica para o autômato finito adaptativo de segunda ordem aplicada a um modelo de inferência indutiva, Phd thesis, Escola Politécnica, Universidade de São Paulo, São Paulo (2011).
17. R. I. Silva Filho, R. L. d. A. Rocha, R. H. G. Guiraldelli, Algorithmic Probability and Friends, Bayesian Prediction and Artificial Intelligence: Papers from the Ray Solomonoff 85th Memorial Conference, Springer Berlin Heidelberg, 2013, Ch. Learning in the Limit: A Mutational and Adaptive Approach, pp. 106–118.
18. R. Hull, J. Su, On the expressive power of database queries with intermediate types, Journal of Computer and System Sciences 43 (1) (1991) 219–267.
19. A. Chandra, D. Harel, Structure and complexity of relational queries, Journal of Computer and System Sciences 25 (1) (1982) 99–128.
20. M. Barr, Relational algebras, in: Reports of the Midwest Category Seminar IV, Vol. 137 of Lecture Notes in Mathematics, Springer, Berlin, 1970, pp. 39–55.
21. M. C. Bunge, Relative functor categories and categories of algebras, Journal of Algebra 11 (1) (1969) 64–101.

[1] Apache Avro is a data serialization system. Official page: `https://avro.apache.org/`

[2] Apache Mina is a network application framework. Official page: `https://mina.apache.org/`

[3] Apache Felix is a OSGi framework and service platform implementation. Official page: `http://felix.apache.org/`

[4] Apache Karaf is a modern and polymorphic container. Official page: `http://karaf.apache.org/`