Design of Capability Delivery Adjustments

Jānis Grabis and Jānis Kampars(✉)

Institute of Information Technology,
Riga Technical University, Kalku 1, Riga, Latvia
{grabis, janis.kampars}@rtu.lv

Abstract. Capabilities are designed for ensuring that business services can be delivered to satisfy business performance objectives in different circumstances. Run-time adjustments are used to adapt capability delivery to these specific circumstances. The paper elaborates the concept of the capability delivery adjustments on the basis of capability meta-model proposed as a part of the Capability Driven Development approach. The types of adjustments are identified as their specifications are provided. An example of adjustments modeling is developed.

Keywords: Capability · Adaptation · Run-time · Context

1 Introduction

Capabilities specify an ability and capacity to deliver business services to meet specific business performance objectives in different circumstances [1]. They are delivered in ever-changing contextual situations. The purpose of capability delivery adjustments is to alter capability delivery in response to the changing context and delivery performance without the need for redesigning the capability and underlying information systems. The run-time delivery adjustments specified in this paper support this objective by: (1) enabling specification of complex contextual data processing logics; (2) providing reconfigurable data bindings; and (3) separating contextual dependencies from business logic.

The adjustments provide a uniform way of defining computations associated with the concepts defined in the capability model and primarily of those associated with context elements (represents any information that can be used to characterize the situation of an entity) and context indicators (a property of the context relevant to the capability design and used for monitoring capability delivery). These computations can be specified by a capability designer, and they are decoupled from the rest of capability delivery logics. That allows to make changes in context processing without changing the rest of the capability delivery application. Algorithms for context aware capability delivery adjustment are defined as capability adjustments and provide decision-making logics for capability delivery variations.

The reconfigurable data bindings are important to incorporate new context element in the capability design. The new context elements need to be incorporated because all context elements affecting capability delivery are not known in advance during the capability delivery. Adjustments use constants that can be changed during run-time, allowing to alter the way capability reacts to context without stopping the underlying systems and redeploying solution.

The paper elaborates technical aspects of designing adjustments. The adjustments are executed using a technical platform for capability design delivery as described in [2]. The platform consists of the (1) Capability Delivery Application (CDA), which is responsive for the execution of the business logic with no regard to contextual dependencies; (2) Capability Navigation Application, which processes the context information and provides context awareness to CDA as-a-service; (3) Capability Context Platform, which integrates the context information and notifies CNA of context information changes.

The rest of the paper is organized as follows. Section 2 introduces types of adjustments. These adjustments are further elaborated in Sect. 3. The adjustment modeling is illustrated in Sects. 4 and 5 concludes.

2 Capability Adjustment

Adjustments are developed as a technical addition to the capability meta-model to represent processing of context and indicator data as well as to adapt capability delivery.

2.1 Background

The methodological foundation of capability design and delivery is provided by the core capability meta-model (CMM) in Fig. 1 (more details in [1]). In brief, the meta-model has three main sections: (a) *Enterprise model,* representing organizational designs with Goals, KPIs, Processes (with concretizations as Process Variants) and Resources; (b) *Context*, represented with Context Set for which a Capability is designed and Context Situation at runtime that is monitored and according to which the deployed solutions should be adjusted. Context Indicators are used for high level overview of the contextual situation; and (c) *Patterns,* for delivering Capability by reusable solutions for reaching Goals under different Context Situations.

2.2 Types of Adjustments

Adjustments are used for adjusting the capability based on important factors like context or KPIs. They are also used for monitoring the capability in run-time, by interpreting raw context data (Measurable Properties) in a more comprehensible way and to calculate the KPI current and target values. Two adjustment groups can be distinguished (Fig. 2):

1. `Calculation`, used for calculating `KPI` and `ContextElement` values. Results from `Calculation` instances can be used as input data for `CapabilityAdjusment` instances.

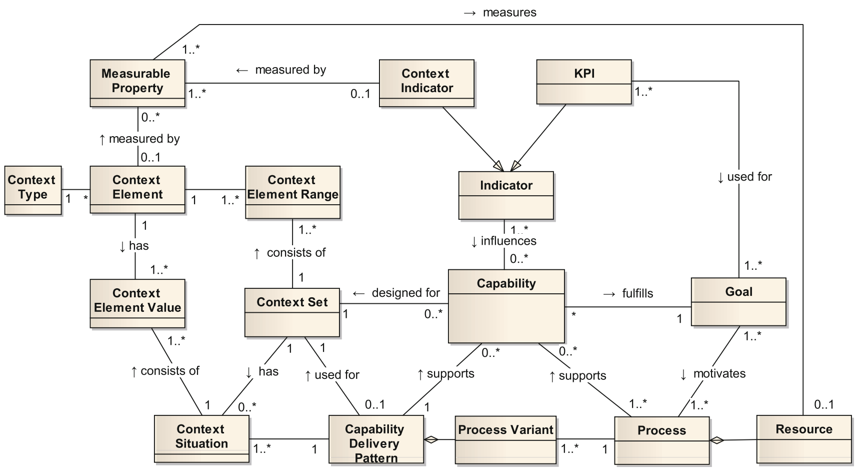

Fig. 1. A core meta-model for supporting capability driven development.

2. `CapabilityAdjustment`, that alter the capability based on context and other factors (e.g. if the average load time of a website is greater than 4 s, a new web server node is deployed in the cloud).

`Calculation` is further divided into `ContextCalculation` and `KPICalculation`. `ContextCalculation` is used for interpreting measurable properties and calculating the value of a `ContextElement`. `KPICalculation` is responsible for calculating target and current values of KPI based on various input data. The KPI values can be visualized using a set of predefined widgets in CNA during run-time or used as an input data for other adjustments. `EventBasedAdjustment` is exposed as a REST web service from CNA and is used for enabling process-based capabilities. In cases when CDA requires to choose an exit for a business process gateway based on the current contextual situation, it can query the CNA web service for making the right decision. `ScheduledAdjustment` allows to implement schedule based, in-code adjustments. `ScheduledAdjustment` is execute periodically and can trigger a change in the CDA based on the current contextual situation by calling a web service exposed by the CDA. In most cases `CapabilityAdjustment` instances rely on results received from `Calculation` instances.

Adjustment variables are used to implement the adjustment logic and they are accessible in the scope of the adjustment code. Optionally input parameters can be used for initializing the adjustment variable values. Adjustment variables are also initialized using Input Data Associations (IDA), which allow to bind adjustment constants or other global available data to local adjustment variables.

There are multiple alternatives for implementing adjustments among which are Java (in-code adjustments) and MathML (implemented using a visual tool with no need to write code manually). The Adjustment engine is responsible for execution of the adjustment code.

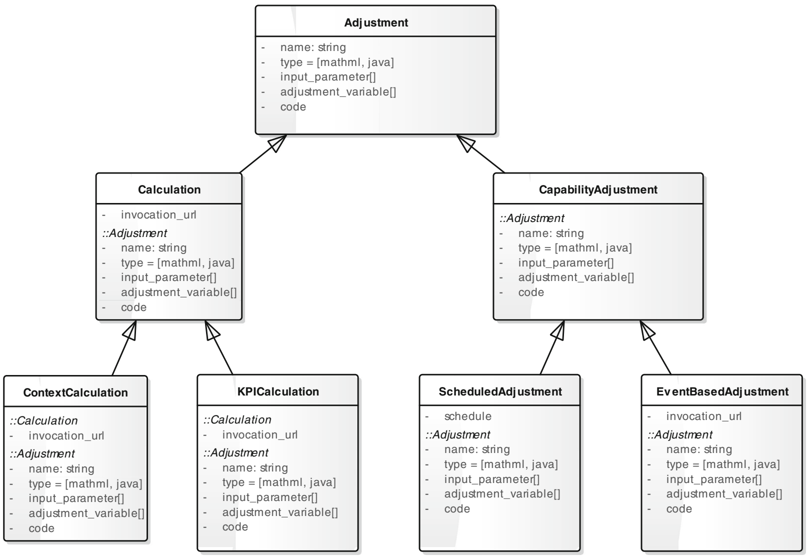

Fig. 2. Groups of adjustment

CapabilityAdjusment supports decision making for invoking an appropriate process variant for dealing with specific context situations It can use ContextElement value, KPI values and adjustment constants. Two subtypes of CapabilityAdjustment exist – EventBasedCalculation and ScheduledAdjustment. ScheduledAdjustment is executed based on a predefined schedule and it cannot be called through a web service. EventBasedAdjustment is deployed as a web service. It can be called whenever it is required to make a decision based on the current context.

3 Elaboration of Adjustments

This section briefly describes a procedure for adjustment modeling in the Capability Design Tool (an Eclipse based modeling tool which is part of the Capability Driven Development environment) consisting of three main activities (Fig. 3).

The Add ContextCalculation activity identifies all ContextElement class instances needed for adjusting the capability, creates a ContextCalculation instance for each of them and connects the ContextCalculation instance to the corresponding ContextElement instance (Fig. 4). Similarly, the Add KPICalculation specifies calculations for KPIs.

The Add EventBasedAdjustment activity specifies uses of adjustments for selecting the process execution variants. When using EventBasedAdjustment together with process gateways, their associations are used to specify what process instance specific values are passed to the EventBasedAdjustment and how the adjustment result is used to choose the desired process variant (Fig. 5).

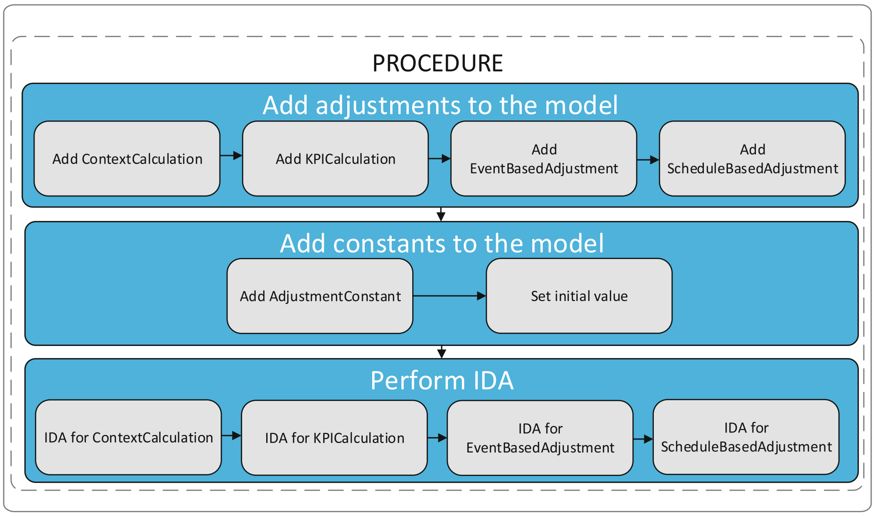
PROCEDURE
Add adjustments to the model
Add ContextCalculation
Add KPICalculation
Add EventBasedAdjustment
Add ScheduleBasedAdjustment
Add constants to the model
Add AdjustmentConstant
Set initial value
Perform IDA
IDA for ContextCalculation
IDA for KPICalculation
IDA for EventBasedAdjustment
IDA for ScheduleBasedAdjustment

Fig. 3. Adjustment modelling procedure

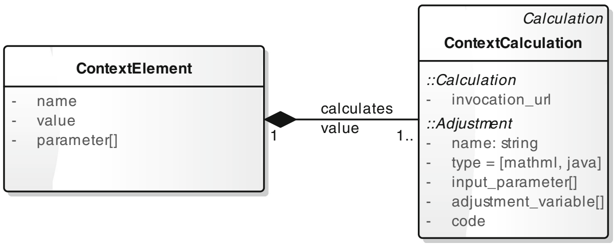

Fig. 4. Linking a `ContextElement` to a `ContextCalculation`

The IDA for ContextCalculation activity specifies data bindings for `ContextCalculation` instances. A `ContextCalculation` instance can be linked to instances of `AdjustmentConstant` and `MeasurableProperty`. The value of `AdjustmentConstant` can be changed during run-time by the designer, while the value of a `MeasurableProperty` is received from the CCP.

4 Example

An example illustrating usage of scheduled adjustments considers on-demand scaling of computational resources in the cloud. Different methods and computational platforms have been proposed to address this issue (e.g., [3, 4]). The methods differ by scaling algorithms used, performance measures used and contextual factors considered. The proposed application of capability modeling and run-time adjustments allows for

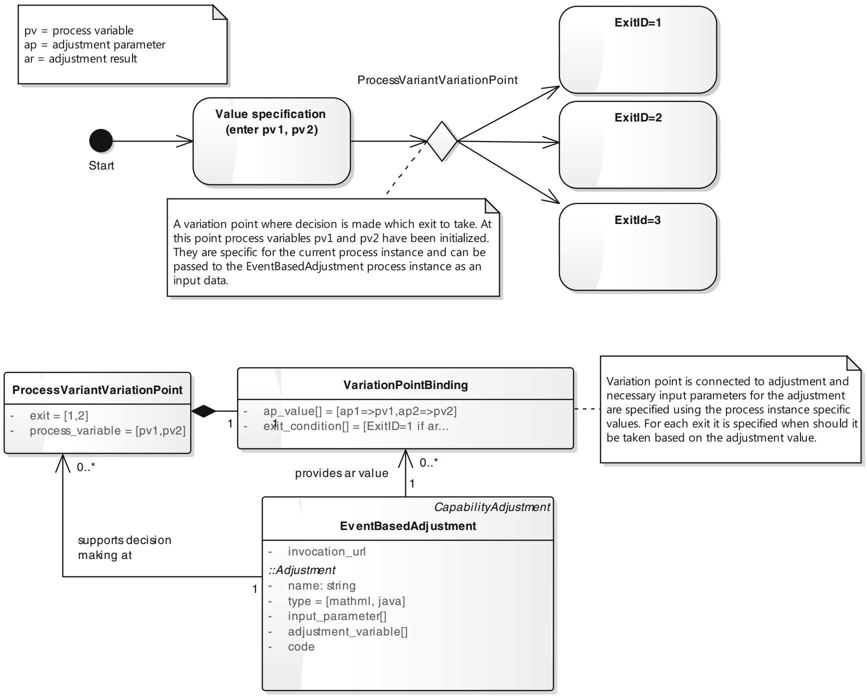
pv = process variable
ap = adjustment parameter
ar = adjustment result
Start
Value specification (enter pv1, pv2)
ProcessVariantVariationPoint
ExitID=1
ExitID=2
ExitId=3
A variation point where decision is made which exit to take. At this point process variables pv1 and pv2 have been initialized. They are specific for the current process instance and can be passed to the EventBasedAdjustment process instance as an input data.
ProcessVariantVariationPoint
- exit = [1,2]
- process_variable = [pv1,pv2]
VariationPointBinding
- ap_value[] = [ap1=>pv1,ap2=>pv2]
- exit_condition[] = [ExitID=1 if ar...
Variation point is connected to adjustment and necessary input parameters for the adjustment are specified using the process instance specific values. For each exit it is specified when should it be taken based on the adjustment value.
1
0..*
provides ar value
0..*
1
supports decision making at
1
CapabilityAdjustment
EventBasedAdjustment
- invocation_url
::Adjustment
- name: string
- type = [mathml, java]
- input_parameter[]
- adjustment_variable[]
- code

Fig. 5. Linking an `EventBasedAdjustment` to process variants

flexibility to use these different algorithms, performance measures and contextual factors within a single framework.

The adjustment model in Fig. 6 shows context elements affecting the scaling decisions and KPIs used to monitor the cloud performance. The scheduled adjustment continuously monitors the cloud platform and if necessary increases the number of computational nodes available.

Context elements `webServerLoad` (the current average load of webservers), `webServerResponseTime` (the current average response time of webservers) and `nodeCount` (the current web server node count in the cloud) are used to describe the contextual situation. A KPI of `avgResponseTime` is defined to use it as an overall quality measure of the cloud-based solution. Two `KPICalculation` instances are used to calculate the target and current values of the KPI. The scaling process is limited by adjustment constants `maxNodes` and `minNodes` which represent the maximum and minimum number of web server in the cloud. These constants can be changed during run-time for adding more resources to the web application and reaching KPI target value. In order to use the context element values, adjustments constants and KPI values in the `cloudScales` adjustment IDAs need to be established. For each `ContextCalculator` an instance a `MeasurablePropertyIDA` is created and linked to the corresponding `MeasurableProperty`. To set the server type `Mea-surablePropertyIDA` relies on the `serverType` adjustment constant.

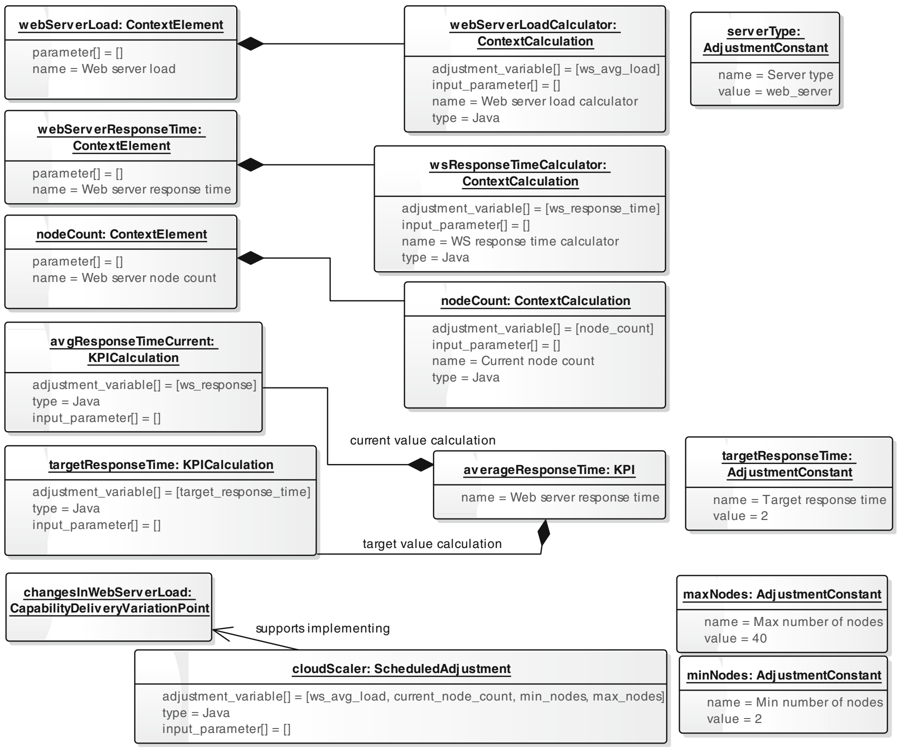
webServerLoad: ContextElement
parameter[] = []
name = Web server load
webServerLoadCalculator: ContextCalculation
adjustment_variable[] = [ws_avg_load]
input_parameter[] = []
name = Web server load calculator
type = Java
serverType: AdjustmentConstant
name = Server type
value = web_server
webServerResponseTime: ContextElement
parameter[] = []
name = Web server response time
wsResponseTimeCalculator: ContextCalculation
adjustment_variable[] = [ws_response_time]
input_parameter[] = []
name = WS response time calculator
type = Java
nodeCount: ContextElement
parameter[] = []
name = Web server node count
nodeCount: ContextCalculation
adjustment_variable[] = [node_count]
input_parameter[] = []
name = Current node count
type = Java
avgResponseTimeCurrent: KPICalculation
adjustment_variable[] = [ws_response]
type = Java
input_parameter[] = []
current value calculation
averageResponseTime: KPI
name = Web server response time
targetResponseTime: AdjustmentConstant
name = Target response time
value = 2
targetResponseTime: KPICalculation
adjustment_variable[] = [target_response_time]
type = Java
input_parameter[] = []
target value calculation
changesInWebServerLoad: CapabilityDeliveryVariationPoint
supports implementing
maxNodes: AdjustmentConstant
name = Max number of nodes
value = 40
cloudScaler: ScheduledAdjustment
adjustment_variable[] = [ws_avg_load, current_node_count, min_nodes, max_nodes]
type = Java
input_parameter[] = []
minNodes: AdjustmentConstant
name = Min number of nodes
value = 2

Fig. 6. `ScheduledAdjustment` example

Two `ContextElementValueIDA` instances are added to the model for using the contex element values. For using values of `AdjustmenConstant` instances, `AdjustmentConstantIDA` elements are created.

A simplified version of cloudScaler implementation is given in Fig. 7.

Based on the current load, current number of nodes and constants the `cloudScaler` can choose whether to scale up or down. `cloudApi` represents the cloud native API of the corresponding cloud computing platform.

5 Related Work

New business needs and requirements arise during service and software delivery and some of them might be introduced without interrupting the service and shutting down the software. These changes are variously described in literature as runtime adaptation, dynamic reconfiguration, autonomic computing, self-adaptation, dynamic evolution, runtime adjustment and others. The modern research on delivery and runtime modification originates from the vision of autonomic computing [5]. The vision was formulated in response to the software complexity, and it primarily focuses on technical aspects of running complex integrated software systems. The four main aspects of

```
   //Implementation of execute method for cloudScaling
   public String execute()
   {
   Integer minNodes =
Integer.parseInt(this.getVariableValue("min_nodes"));
   Integer maxNodes =
Integer.parseInt(this.getVariableValue("max_nodes"));
   Integer
currentNodes=Integer.parseInt(this.getVariableValue("current_node_co
unt"));
   String load= this.getVariableValue("ws_load");
   if ( load == "high")
   {
   // CloudAPI is an external JAVA library packed inside the
resulting JAR file.
         if (currentNodes< maxNodes){
         cloudAPI.scaleUp();
         return "Scaling up, current nodes  "+ currentNodes;
```

Fig. 7. Adjustment implementation

self-management in autonomic computing are self-configuration, self-optimization, self-healing and self-protection.

There are different types of adaptive systems. The common features of the adaptive systems are monitoring of changes, goal driven adjustment of the system and a feedback loop measuring the success of the adjustment [6]. Self-adaptive systems recently have attracted the most attention in computer science. The self-adaptive systems are able to modify their behavior and/or structure in response to their perception of the environment and the system itself, and their goals [7].

Weyns et al. [8] elaborate a formal reference model of the self-adaptive systems. The model is represented using both UML diagrams and Z language what allows reasoning about behaviour of adaptive systems. Multiple case studies are provided. The literature review on self-adaptive systems [9] shows that researchers focus on software design issues of self-adaptive systems and single MAPE feedback loops.

Their self-adaptive software life-cycle model includes the offline and online activities [10]. Dynamic reconfiguration is a mechanism that allows the modification of a software system during the execution time without shutting it down or restarting it [11]. Methods used for dynamic reconfiguration make changes at the code level or at the component level. In the era of cloud computing, many aspects of dynamic reconfiguration are addressed by dynamic provisioning of cloud computing resources, where intelligent algorithms are used to make provisioning decisions during the service delivery (e.g., [12]). Mori [13] talks of software evolution as a software engineering process to design context-aware adaptive applications resilient to context and user needs variations. Oreizy et al. [14] claim that runtime adaption consists of two interlinked cycles of evolution management and adaptation management. The evolution management deals with changing the application on the basis of interrelated models including code as a model.

The adjustment can be considered as a looser term compared to adaptation, which requires the adaptation goal and the feedback loop, and reconfiguration, which deals with structural properties of the system. Montani and Leonardi [15] use the term "run-time adjustment" in relation to agile workflow technologies. More importantly, the term is applied to modification of business processes rather than to modification of technical aspects of systems. User interface and underlying business logics also can be adapted using information provided in goal models [16]. Process adaption has been an active research area with initial emphasis on adaptive workflows, followed by QoS driven BPEL adaptation and lately on general business processes since BPMN has become executable. Change patterns are at the heart for the workflow adaptation [17]. The AGENTWORK [18] is one of the best know adaptive workflow management systems. The workflow can be automatically adapted either reactively or proactively. Alferez el al. [19] investigate dynamic adaption of service composition. They refer to dynamic adaption as opposite to static adaption, which requires shutting down the system for manual modification.

Business process variants currently is one of the most frequently used methods for supporting adjustment of business processes for specific conditions and requirements. These variants can be constructed either by configuration or adaptation [20]. That can be done in design time as well as in runtime. In the case of adaption, the variants are designed by applying business process change operations such as insertion, deletion of tasks or other process flow elements. Hallerbach et al. [21] elaborate a Provop approach to developing process variants. Their process variant life-cycle consists of four phases, namely, modelling, configuration, execution and optimization. The context-based configuration of the variants is supported. In the execution phase, switching between the process variants is possible to deal with dynamic context changes. A system is context-aware if it uses context to provide relevant information and/or services to the user, where relevancy depends on the user's task [22]. The context awareness does not necessarily involve adaption and reconfiguration. It is inherently dynamic since majority of context values are known only during the systems execution. Context awareness is also used in workflow adaptation. Smanchat et al. [23] show that majority of the surveyed context aware workflow solutions deal with workflow instance adaption.

6 Conclusion

The paper has described an approach for modeling capability delivery adjustments, which are used for context information processing and capability delivery adaptation in response to changing contextual situations during capability delivery. Capability delivery adjustment enable maintaining the desired level of delivery performance. The adjustments are specified as an extension of the capability meta-model.

The paper focused only on the adjustment modeling part. The adjustment deployment environment and adjustment execution activities are also essential for application of adjustments.

References

1. Bērziša, S., Bravos, G., González, T., Czubayko, U., España, S., Grabis, J., Henkel, M., Jokste, L., Kampars, J., Koç, H., Kuhr, J., Llorca, C., Loucopoulos, P., Pascual, R.J., Pastor, O., Sandkuhl, K., Simic, H., Stirna, J., Giromé, F.V., Zdravkovic, J.: Capability driven development: an approach to designing digital enterprises. Bus. Inf. Syst. Eng. **57**, 15–25 (2015)
2. Zdravkovic, J., Stirna, J., Henkel, M., Grabis, J.: Modeling business capabilities and context dependent delivery by cloud services. In: Salinesi, C., Norrie, M.C., Pastor, Ó. (eds.) CAiSE 2013. LNCS, vol. 7908, pp. 369–383. Springer, Heidelberg (2013)
3. Han, R., Guo, L., Ghanem, M.M., Guo, Y.: Lightweight resource scaling for cloud applications. In: 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid), pp. 644–651 (2012)
4. Lorido-Botran, T., Miguel-Alonso, J., Lozano, J.A.: A review of auto-scaling techniques for elastic applications in cloud environments. J. Grid Comput. **12**(4), 559–592 (2014)
5. Kephart, J.O., Chess, D.M.: The vision of autonomic computing. Computer **36**, 41–50 (2003)
6. Heylighen, F.: Web dictionary of cybernetics and systems; principia cybernetica web (2004). http://pespmc1.vub.ac.be. Accessed 27 Dec 2013
7. de Lemos, R., et al.: Software engineering for self-adaptive systems: a second research roadmap. In: de Lemos, R., Giese, H., Müller, H.A., Shaw, M. (eds.) Software Engineering for Self-Adaptive Systems. LNCS, vol. 7475, pp. 1–32. Springer, Heidelberg (2013)
8. Weyns, D., Malek, S., Andersson, J.: FORMS: unifying reference model for formal specification of distributed self-adaptive systems. ACM Trans. Auton. Adapt. Syst. **7**(1), 8–61 (2012)
9. Weyns, D., Ahmad, T.: Claims and evidence for architecture-based self-adaptation: a systematic literature review. In: Drira, K. (ed.) ECSA 2013. LNCS, vol. 7957, pp. 249–265. Springer, Heidelberg (2013)
10. Andersson, J., Baresi, L., Bencomo, N., de Lemos, R., Gorla, A., Inverardi, P., Vogel, T.: Software engineering processes for self-adaptive systems. In: de Lemos, R., Giese, H., Müller, H.A., Shaw, M. (eds.) Software Engineering for Self-Adaptive Systems. LNCS, vol. 7475, pp. 51–75. Springer, Heidelberg (2013)
11. Eddin, M.C.: Towards a taxonomy of dynamic reconfiguration approaches. J. Softw. **8**(9), 2202–2207 (2013)
12. Islam, S., Keung, J., Lee, K., Liu, A.: Empirical prediction models for adaptive resource provisioning in the cloud. Future Gener. Comput. Syst. **28**(1), 155–162 (2012)
13. Mori, M.: A software lifecycle process for context-aware adaptive systems. In: Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European conference on Foundations of Software Engineering, pp. 412–415 (2011)
14. Oreizy, P., Medvidovic, N., Taylor, R.N.: Runtime software adaptation: framework, approaches, and styles. In: Proceedings of ICSE Companion 2008, Companion of the 30th International Conference on Software Engineering, pp. 899–910 (2008)
15. Montani, S., Leonardi, G.: Retrieval and clustering for supporting business process adjustment and analysis. Inf. Syst. **40**, 128–141 (2014)
16. Liaskos, S., Khan, S.M., Litoiu, M., Jungblut, M.D., Rogozhkin, V., Mylopoulos, J.: Behavioral adaptation of information systems through goal models. Inf. Syst. **37**, 767–783 (2012)

17. Weber, B., Reichert, M., Rinderle-Ma, S.: Change patterns and change support features - enhancing flexibility in process-aware information systems. Data Knowl. Eng. **66**, 438–466 (2008)
18. Muller, R., Greiner, U., Rahm, E.: AGENT WORK: a workflow system supporting rule-based workflow adaptation. Data Knowl. Eng. **51**, 223–256 (2004)
19. Alférez, G.H., Pelechano, V., Mazo, R., Salinesi, C., Diaz, D.: Dynamic adaptation of service compositions with variability models. J. Syst. Softw. **91**, 24–47 (2014)
20. Döhring, M., Reijers, H.A., Smirnov, S.: Configuration vs. adaptation for business process variant maintenance: an empirical study. Inf. Syst. **39**, 108–133 (2014)
21. Hallerbach, A., Bauer, T., Reichert, M.: Capturing variability in business process models: The Provop approach. J. Softw. Maint. Evol. **22**(6–7), 519–546 (2010)
22. Abowd, G.D., Dey, A.K.: Towards a better understanding of context and context-awareness. In: Gellersen, H.-W. (ed.) HUC 1999. LNCS, vol. 1707, pp. 304–307. Springer, Heidelberg (1999)
23. Smanchat, S., Ling, S., Indrawan, M.: A survey on context-aware workflow adaptations. In: Proceedings of MoMM 2008, pp. 414–417 (2008)