A Repository for Pattern Governance Supporting Capability Driven Development

Janis Kampars[1], Janis Stirna[2]

[1]Information Technology Institute, Riga Technical University
Kalku iela 1, Riga, Latvia
Janis.Kampars@rtu.lv
[2]Department of Computer and Systems Sciences, Stockholm University,
Box 7003, Kista, 16407, Sweden
js@dsv.su.se

Abstract. Patterns have a great potential for improving various aspects of Information System (IS) designs by reuse. While they have been routinely used for conveying reusable design solutions in books and knowledge repositories, there is an ongoing debate about their impact in practice. This is due to the fact that insufficient efforts are devoted to elaborating effective solutions for eliciting and documenting patterns, evaluating them, tracing IS designs back to the applied patterns, and assisting the designer in choosing the right pattern in a given contextual situation. These tasks need to be supported by advanced pattern repositories that are able to manage patterns at runtime. The article presents the usage of a Capability Pattern Repository (CPR) in support of an approach for design and delivery of context dependent IS, namely, Capability Driven Development (CDD). The CPR together with CDD provides a tool and a method that support pattern governance for addressing both the design and run-time of IS. The described approach is not bound to CDD and can be adapted to support different types of patterns and development methodologies.

Key words: patterns, pattern evaluation, pattern performance, Capability Driven Development

1 Introduction

The increased complexity of modern information systems (IS) and the dynamically changing environment in which they operate motivate the need for improving efficiency of IS design and maintenance. Much of the solutions that need to be invoked once an organization needs to adjust its IS in order to react to certain external or internal situational changes exists e.g. in the form of best practices, configurable components, business services, business process variants. In this respect, patterns have proven to be of value because they offer means for reusing existing know-how (cf. for instance [1, 2, and 3]. The concept of pattern allows presenting reusable designs, models, components, code fragments, etc. They assist at various stages of IS development such as analysis, design, implementation, and maintenance.

A pattern is a proven reusable solution to a reoccurring problem. It has the potential to substantially increase the quality of IS design and architecture as well as lowering the costs by reusing tried-and-tested design and component implementations. Patterns typically offer a general solution that needs to be tailored and adapted to the particular application context. Applied patterns have a great potential for improving various aspects of IS design although there is an ongoing debate concerning whether patterns have a positive effect or should be used with caution, cf. for instance [4, 5]. Another underexplored area is pattern storage and discovery, which becomes especially important when dealing with a large numbers of patterns and when patterns are used over time, which is when they become the most useful. A key aspects of supporting the long-term patters use is the need to trace them to the IS that use them which is cumbersome without a methodological and technological solutions. Such solutions would significantly contribute to addressing the problem of pattern performance evaluation.

Most of the currently available pattern evaluation studies are performed by surveying the users, which can be subjective, hard to interpret, and involves hypothetical usage situations. Nevertheless, patterns need to be documented and applied under the correct contextual situation and their effectiveness in real application cases should be measured. In this regard a recently completed FP7 project "Capability as a Service in digital enterprises" (CaaS) has proposed a methodology for context depended development and delivery of IS services, namely, Capability Driven Development (CDD) [6]. More specifically, CDD uses capability notation to capture and analyse the changing business context in design of information systems and promotes the usage of patterns by providing the needed theoretical, technological and methodological support. In the specification and design of services using business planning as the baseline, capability is seen as an ability and capacity for an enterprise to deliver value, either to customers or shareholders. According to CDD, a capability is delivered based on existing organizational solutions that are captured and represented by patterns. Capability patterns are enterprise-size components (e.g. code fragments, web service definitions, business process models) carrying a software solution for an organizational problem in a given business context, in both the design and run-times [7]. A part of the CDD environment are patterns stored in a Capability Pattern Repository (CPR) [8], where Pattern Performance Indicators (PPI) are calculated based on the run-time capability Key Performance Indicator (KPI) values. PPIs show to what extent does a pattern contributes to reaching the KPI target value.

The objective of this paper is to contribute to the challenges of pattern cataloging, discovery, evaluation, and application by presenting the work on the Capability Pattern Repository. Design science [9] principles have been chosen as the research method for developing the main design artifacts of this paper, namely, (1) the CPR and (2) the needed methodology for its use.. The developed CPR provides means for more effective pattern storage, discovery, and evaluation. It can be customized to support different kinds of patterns as well as used in conjunction with other development methodologies. The functionality of the CPR will be evaluated by demonstrating a sample pattern that is published in the CPR.

The rest of the paper is organized as follows. Section 2 gives an overview of the CDD environment, the web services provided by the CPR, structure and lifecycle of a capability pattern, and pattern evaluation approach. Section 3 provides an example of using patterns for capability driven development. Section 4 concludes with a brief overview of the results and outlines directions for future work.

2 The Role of Capability Pattern Repository in CDD

This section presents the Capability Pattern Repository. It is an integral part of CDD from technological and methodological perspective. A more detailed description of the CDD is given in [6] while previous version of the CPR and its role in CDD is documented in [7].

2.1 Overview of the CDD Environment

An overview of the CDD environment from a perspective of pattern governance is given in Fig. 1.

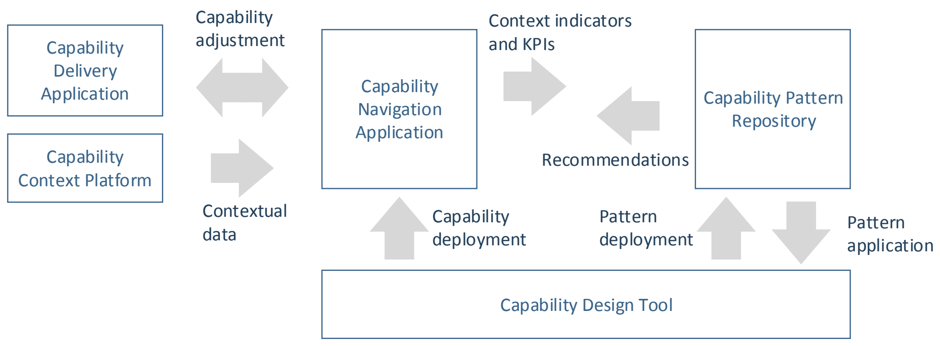

Capability Delivery Application
Capability Context Platform
Capability adjustment
Contextual data
Capability Navigation Application
Context indicators and KPIs
Recommendations
Capability Pattern Repository
Capability deployment
Pattern deployment
Pattern application
Capability Design Tool

Fig. 1. Overview of the CDD environment

Capability Design Tool (CDT) is an Eclipse based environment that is used to model and implement both capability patterns and capabilities applying them. It is integrated with the CPR using a group of REST web services. Both patterns and capabilities applying them are published from the CDT to the CPR, allowing tracing the capability back to applied patterns and vice versa. The CDT is integrated with the Capability Navigation Application (CNA), which is the main means for separating business logic from adaptation to contextual changes. Based on the contextual information received from the Capability Context Platform (CCP), the CNA can trigger changes in the Capability Delivery Application (CDA). During run-time the CNA sends back statistical information to the CPR that is used for quantitative evaluation of capability patterns.

2.2 Web Services Provided by the CPR

The main web services of CPR supporting effective governance of capability patterns are listed in Table 1.

Table 1. An overview of CPR web services

Service Name	Description
Capability publishing	Publish capabilities from CDT user interface by providing a universally unique identifier (UUID) of a CPR user, access token, capability UUID, name, category and a base64 encoded archived CDT project containing the implementation of the capability.
Pattern publishing	Publish capability patterns from CDT user interface by providing a CPR user UUID, access token, pattern UUID, name, category and a base64 encoded archived CDT project containing the implementation of the pattern.
Context indicator value update	To gather statistical information about the effectiveness of a pattern it is important to be aware under which contextual conditions it has been used. The data is updated by CNA during run-time.
KPI value update	In CDD patterns can be connected to KPIs, therefore KPI data is needed for quantitative evaluation of capability patterns. KPI data is sent by CNA during run-time.
Pattern search	Patterns are typically searched from the CDT by providing either the name of the pattern, its UUID or UUIDs of the elements contained in the pattern.
Pattern recommendations	This web service is used by the CNA by providing the UUIDs of the capability and CNA (a company might choose to deploy the same capability to multiple CNAs). CPR is aware of the capability structure, currently applied patterns and contextual conditions. Based on that it can provide run-time recommendations of other patterns that might perform better in the current contextual situation.

2.3 Lifecycle and Structure of a Capability Pattern

The lifecycle of a capability pattern with relations to components of the CDD environment is summarized in Fig. 2. Patterns are modeled in the CDT and described by a set of problem, context, and solution diagrams. Diagrams are based on the elements of the Capability Meta Model [6]. Alternatively, the designer can enter a free text based version of problem, context, solution, and usage guidelines, as well as specify a category name for the pattern. It can be decided whether to document problem, context and solution in form of diagrams and/or free-form text. The structure of a capability pattern is summarized in Table 2.

Context diagrams are crucial for ensuring application of the pattern under correct contextual situation. In this case the most important elements of the context diagram are `Context Set` and `Context Element Ranges`. From the CDT interface the patterns are published to the CPR using a REST web service (see Table 1). In order to publish a pattern, the user must specify a CPR user UUID and access token in the CDT settings section.

After being published the pattern is indexed in CPR by parsing the model.xmi and contained diagrams. This is important for enabling the discovery of patterns based on the contained elements (using web service or web-based interface of CPR). When the pattern is published the designer can log in to the web-based interface of the CPR to finalize the textual description of problem, context, solution and usage guidelines in the WYSIWYG editor.

Table 2. Structure of a capability pattern

Name of the field	**Purpose of the field**
Name	Each pattern should have a `name` that reflects the problem/solution that it addresses. Names of patterns are also used for indexing purposes.
Problem description Problem diagrams	Describes the issues that the pattern wishes to address within the given context and forces. The goal model is typically used to represent the problem in form of a diagram.
Context description Context diagrams	Describes the preconditions under which the problem and the proposed solution seem to occur. This can be expressed in free text and/or represented in form of diagrams by creating a `Context Set` and `Context Element Ranges` of `Context Elements` that influence the applicability and variability of the solution proposed by the `Pattern`.
Solution description Solution diagrams and implementation	Describes how to solve the problem and to achieve the desired result. It consists of a textual solution description and a solution model fragment. `Process Variants` expressed in BPMN and adjustments programmed in JAVA are typical examples of a solution implementation.
Usage guidelines	Presents a set of usage tips to the potential user of the pattern about how the pattern can be tailored to fit into particular situations or to meet specific needs of an organization.
Pattern category	A list of category names (keywords) for each pattern in order to facilitate search and retrieval.

Similarly, to patterns, capabilities are also modeled in the CDT. During the modeling process the designer can choose to import a pattern from the CPR. The repository can be queried by providing a pattern name, UUID or UUIDs of elements contained in it. The designer can also rely on the web interface of CPR and provided faceted search functionality to find the desired pattern. Pattern import process starts by downloading an archived capability pattern project from CPR to CDT. It is extracted afterwards and all of pattern's model.xmi elements that were not contained in the original capability's model.xmi are appended to it. The problem, context and solution diagrams are added to the current capability project. The corresponding pattern element is added to the diagram of patterns to keep track of currently applied patterns (if no such diagram exists, it is created beforehand). The pattern element is linked to KPIs, to show which KPIs are expected to be influenced by the application of the pattern. A weight from 0 to 1 can be specified for the connection between a pattern and a KPI. If there is no direct effect on current KPIs this step can be omitted or new KPIs can be added to the project.

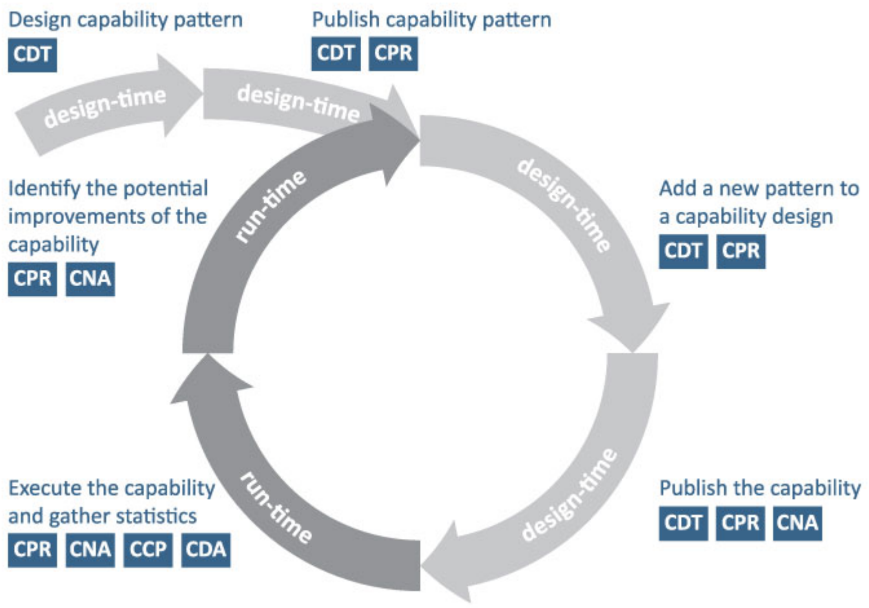

Design capability pattern
CDT
design-time
Publish capability pattern
CDT
CPR
design-time
design-time
Add a new pattern to a capability design
CDT
CPR
Identify the potential improvements of the capability
CPR
CNA
run-time
design-time
Publish the capability
CDT
CPR
CNA
run-time
Execute the capability and gather statistics
CPR
CNA
CCP
CDA

Fig. 2. Capability pattern lifecycle

After the pattern has been adapted for the specific requirements and the design of a capability is completed it is deployed to the CPR. The deployment of a capability to the CPR allows browsing patterns together with capabilities that are based on them. The connection between pattern and KPI enables quantitative evaluation of patterns in the CPR. After the deployment the capability write token and UUID are appended to the CDT project's model.xmi. These are used by the CNA in later stages to update the KPI and context indicator values in the CPR.

Next the capability is deployed to the CNA for execution. During the execution the CNA receives context data from the CCP and calculates `Context Element` values, `Context Indicators` and `KPIs`. Based on the current context the CNA adjusts capability in the CDA. One of the possible run-time adjustments is switching between multiple patterns based on the current context set (such adjustment has to be foreseen while designing the capability in the CDT). During run-time the CNA sends `KPI` and `Context Indicator` values to the CPR that are used for calculating Pattern Performance Indicators (PPIs).

During run-time designer can use the CNA web interface to check whether there are any patterns which are not implemented however could perform better under the current contextual situation. This is done via a REST web service invocation from the CNA to the CPR (see Table 1). The incorporation of new patterns is not supported during run-time, therefore to add a new pattern the designer needs to alter the capability project in the CDT and redeploy it to the CPR and the CNA.

2.4 Pattern Evaluation

The CPR supports two ways of evaluating a capability pattern. The CPR user can leave feedback in the comment section under the specific pattern and give pattern a user rating from 1 to 5 stars using the CPR web interface. This is especially important for patterns that have no KPIs associated with them.

If a pattern has KPIs connected to it and statistical data has been collected by the CPR, a PPI is computed as:

$$PPI = \sum_{i=1}^{n} \left(\frac{w_i}{\sum_{j=1}^{n} w_j} * \frac{KPI_i^{Current} * 100}{KPI_i^{Target}} \right), \qquad (1)$$

where w is weight reflecting the pattern's impact on a KPI, n is a total number of KPIs that pattern has been linked to, $KPI_i^{Current}$ is the current value of the i-th KPI, KPI_i^{Target} is the target value of the i-th KPI.

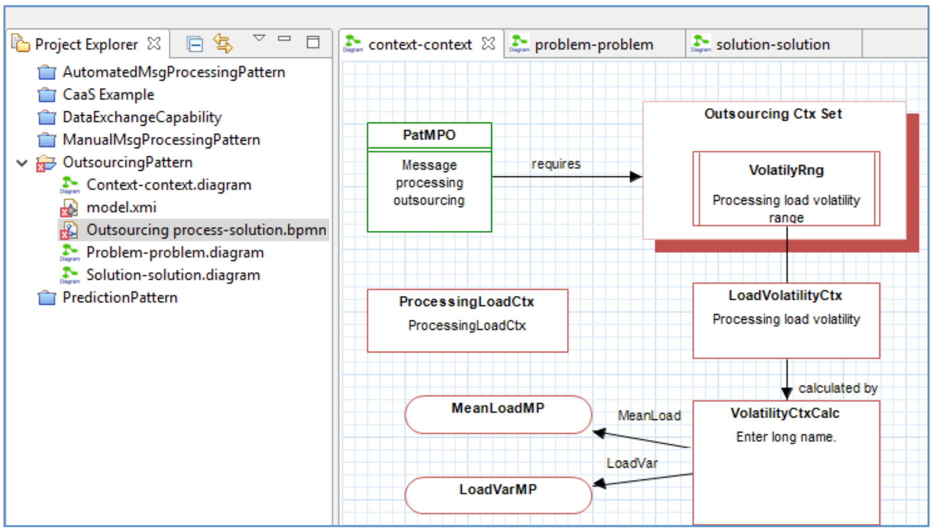


Fig. 3. Pattern design in CDT

3 Example

In order to demonstrate the operation of the CPR an example from the field of cloud application scalability is used. To provide the required Quality of Service cloud applications are usually scaled horizontally (the number of running instances is adjusted) based on the load and various performance metrics. Different strategies can be used for this purpose. Netto et al. [10] proposes to categorize them as follows:

- reactive – a scaling operation is performed immediately as soon as performance values have fallen out of a previously defined interval,
- conservative – a scaling operation is performed if during the last few time windows performance values have fallen out of a previously defined interval,
- predictive – performance values for the next time window are predicted and acted upon similarly as with the reactive strategy.

Reactive, conservative and predictive scaling strategies can be formulated as three interchangeable patterns. In this case we consider a general cloud based service where tasks are first sent to a queue and picked up from it by a varying number of worker nodes. The number of nodes has to be scaled according to the load. There exists a previously defined constant reflecting the maximum allowed task wait time in queue. The context in all three cases consists of resource utilization, queue size, and time in queue. Patterns are described using the structure given in Table 2. Solution diagrams consisting for reactive, conservative, and predictive scaling are given in Fig. 4, Fig. 5, Fig. 6 respectively.

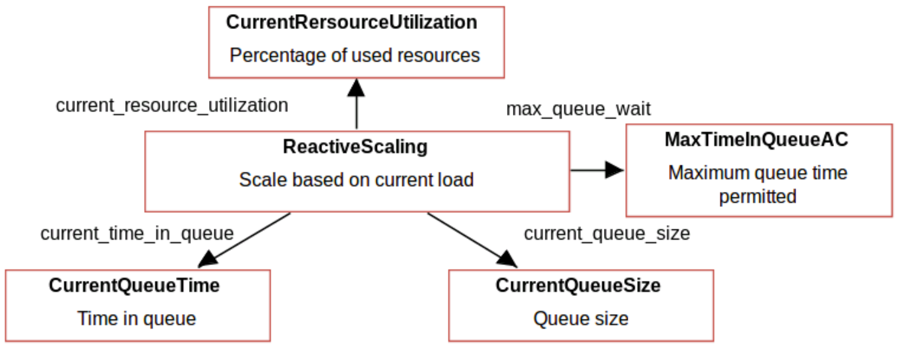


Fig. 4. Reactive scaling pattern solution

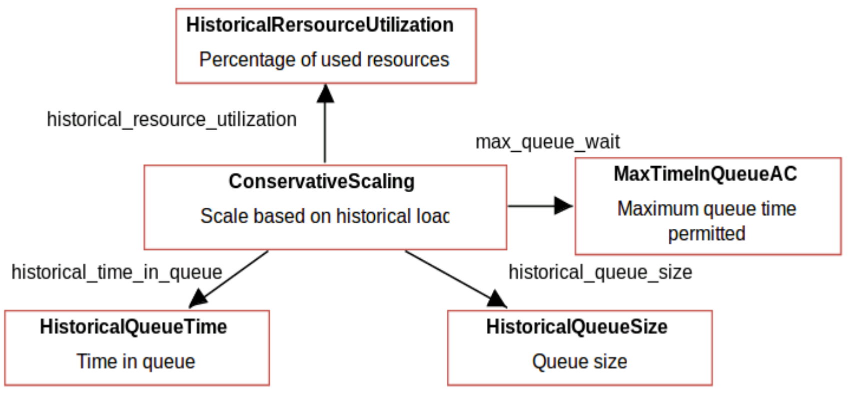


Fig. 5. Conservative scaling pattern solution

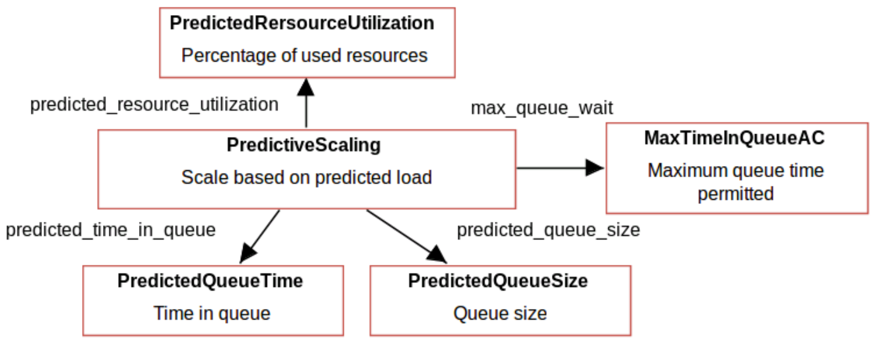


Fig. 6. Predictive scaling pattern solution

All patterns use a constant `MaxTimeInQueueAC` however the three context elements are interpreting the contextual situation according to the scaling strategy (historical, current, and predicted load). All three patterns present the best practices together with the Context Elements that can be used for solving an application scaling problem.

Once patterns have been published in the CPR they can be used for capability design. For example, a scaling capability can be based on one of the mentioned strategies through reusing a pattern. Let's consider a cloud service providing a video transcoding service, where all videos are first queued and then processed by worker nodes. As it can be seen the problem resembles one described by all three patterns and designer can choose between them based on the PPI accumulated in CPR and characteristics of the specific case. The final solution differs from the original pattern since case specific details have been added (see Fig. 7).

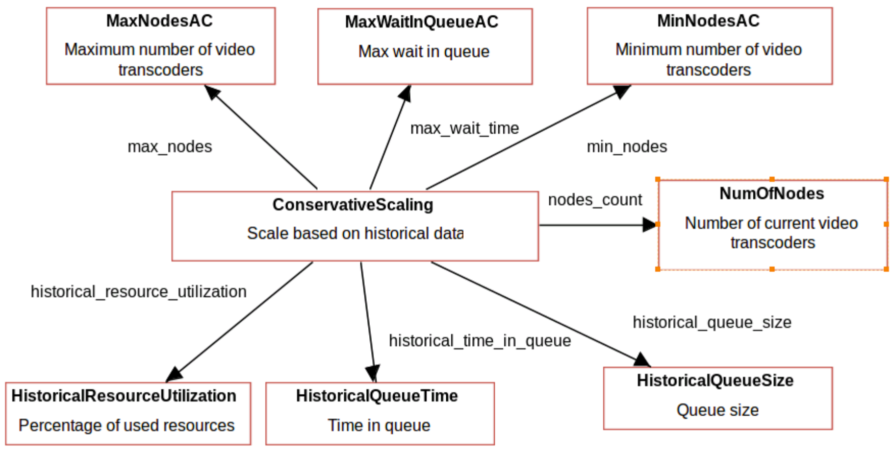
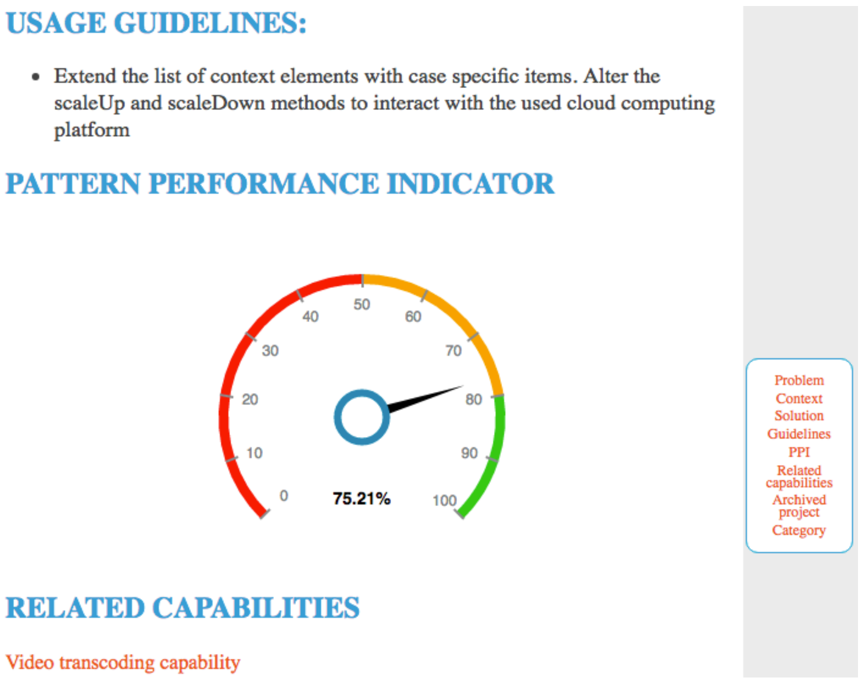


Fig. 7. Capability based on ConservativeScaling

`MaxNodesAC` and `MinNodesAC` are constants used to limit the minimum and maximum number of online video transcoder nodes. `NumOfNodes` is a Context Element used to reflect current number of online video transcoders. If `NumOfNodes` is equal to `MaxNodesAC` scaling up is not allowed, similarly, if `NumOfNodes` is equal to `MinNodesAC` scaling down is prohibited. The implementation code of the algorithm should be adjusted accordingly to the cloud computing platform so that originally available methods `scaleUp` and `scaleDown` would be able to interact with the infrastructure services.

In the given example user satisfaction survey results received after transcoding of the video would serve as the KPI. This is used to calculate the PPI based on Formula 1. The result is shown using a dashboard in the user interface of CPR (see Fig. 8).

An example of a Context Indicator correlating with the PPI of the pattern is the average server startup time. If it takes seconds to startup a new video transcoder, Reactive strategy could be the most effective while Proactive would work better in scenarios where it takes a relatively long time to start a new worker node. This knowledge would be typically discovered through analyzing correlation between Context Indicator values and PPIs stored in CPR. Upon discovering a more suitable pattern the designer can perform according changes to the capability design.


USAGE GUIDELINES:

- Extend the list of context elements with case specific items. Alter the scaleUp and scaleDown methods to interact with the used cloud computing platform

PATTERN PERFORMANCE INDICATOR

RELATED CAPABILITIES

Video transcoding capability

Fig. 8. Visualization of a PPI in CPR

4 Conclusions and Future Work

The CPR has been developed and integrated with other components of the CDD environment. It facilitates effective pattern search, application, and quantitative evaluation. CPR enables a bidirectional trace between the capability delivery solution and the patterns that were used in the construction of it. This together with the availability of run-time statistics and PPIs allows improving the overall quality of capabilities, determining under which contextual conditions a specific pattern operates better, and simplifying the usage of capability patterns in general. Furthermore, with respect the challenges outlined in section 2, in broad terms the CPR facilitates collecting evidence about the pattern utility based real application cases and run-time data.

Among the directions for future work are experiments determining to what extent recommendations from the CPR lead to improvements of the performance of a capability. Another direction is pattern mining, which is made possible by hosting all capabilities and their meta-data in a single repository. Furthermore, investigations into CPRs support of other pattern types and development methodologies and tools (as outlined in [11]) will also be carried out.

5 References

1. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design patterns: elements of reusable object-oriented software. Addison-Wesley Longman Publishing Co., Inc. (1995).
2. Falk, T., Griesberger, P., Leist, S.: Patterns as an artifact for business process improvement - Insights from a case study, (2013).
3. Prechelt, L., Unger-Lamprecht, B., Philippsen, M., Tichy, W.F.: Two controlled experiments assessing the usefulness of design pattern documentation in program maintenance. IEEE Trans. Softw. Eng. 28, 595–606 (2002).
4. Khomh, F., Guéhéneuc, Y.G., Reengineering, F.: Do design patterns impact software quality positively? In: CSMR 2008 - 12th European Conference on Software Maintenance and Reengineering. pp. 274–278. , Athens (2008).
5. Wendorff, P.: Assessment of design patterns during software reengineering: lessons learned from a large commercial project. In: Software Maintenance and Reengineering, 2001. Fifth European Conference on. pp. 77–84 (2001).
6. J. Grabis, M. Henkel, J. Kampars, H. Koç, K. Sandkuhl, D. Stamer, J. Stirna, F. Valverde, J. Zdravkovic "Deliverable 5.3: The Final Version of Capability Driven Development Methodology," FP7 Proj. 611351 CaaS – Capability as a Service in digital enterprises, Stockholm University, pp. 266, 2016. Available: https://doi.org/10.13140/RG.2.2.35862.34889
7. Stirna, J., Zdravkovic, J., Henkel, M., Kampars, J.: Capability Patterns as the Enablers for Model-based Development of Business Context-aware Applications. In: CBI, COBI. pp. 1–12. CEUR Workshop Proceedings, Lisbon (2015). http://ceur-ws.org/Vol-1408/paper2-cobi.pdf
8. Capability Pattern Repository, http://patterns.caas-project.eu/
9. Hevner, A.R., March, S.T., Park, J., Ram, S.: Design Science in Information Systems Research. MIS Q. 28, 75–105 (2004).
10. Netto, M.A.S., Cardonha, C., Cunha, R.L.F., Assuncao, M.D.: Evaluating auto-scaling strategies for cloud computing environments. In: Proceedings - IEEE Computer Society's Annual International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems, MASCOTS. pp. 187–196 (2015).
11. M. Henkel, C. Stratigaki, J. Stirna, P. Loucopoulos, Y. Zorgios and A. Migiakis, (2017) Combining Tools to Design and Develop Software Support for Capabilities, Complex Systems Informatics and Modeling Quarterly, CSIMQ, no. 10, pp. 38–52, Available: https://doi.org/10.7250/csimq.2017-10.03