Complex Systems Informatics and Modeling Quarterly
CSIMQ, Issue 10, March/April 2017, Pages 21–37
Published online by RTU Press, https://csimq-journals.rtu.lv
https://doi.org/10.7250/csimq.2017-10.02
ISSN: 2255-9922 online

Value-Based and Context-Aware Selection of Software-Service Bundles: A Capability Based Method

Jānis Grabis[1*] and Kurt Sandkuhl[2*]

[1]Institute of Information Technology, Riga Technical University, Kalku 1, Riga, LV-1658 Latvia
[2]Faculty of Computer Science and Electrical Engineering, Business Information Systems, University of Rostock, Albert-Einstein-Straße 22, 18059 Rostock, Germany

grabis@rtu.lv (orcid.org/0000-0003-2196-0214),
kurt.sandkuhl@uni-rostock.de (orcid.org/0000-0002-7431-8412)

Abstract. In many application areas, vendors offer to their clients combinations of software products and services, which can be considered as software-service bundles. The clients select a combination of software product and associated service best suited to their individual requirements and circumstances. The bundle usually has to be configured by the vendor to fit the specific situation of the client. When circumstances change, the software-service bundle or its configuration need adaptation. The article proposes a method helping clients to select the most appropriate combination or configuration. The method is based on information sharing between the vendor and client and also supports the continuous improvement of the solution in response to changing circumstances. The method applies principles and techniques of the Capability Driven Development to specify performance objectives and contextual factors affecting delivery of a software-service bundle. Application of the method is shown using an illustrative case of a data processing service from utility industries.
Keywords: Software selection, capability, evolutionary development, software-service bundle.

1 Introduction

In many application areas, vendors offer combinations of software products and services that provide packaged offerings for specific applications. Such software-service bundles are provided by vendors as solutions to their clients. Depending on the client individual requirements and circumstances, the solutions have different configurations with regards to functionality provided. The configurations also differ by costs borne and benefits attained by clients. In order to nurture long-term vendor-client relationships, both parties are interested in rightsizing the service offering to maximize benefits [1].

[*] Corresponding author

Reference: J. Grabis and K. Sandkuhl, “Value-Based and Context-Aware Selection of Software-Service Bundles: A Capability Based Method,” Complex Systems Informatics and Modeling Quarterly, CSIMQ, no. 10, pp. 21–37, 2017. Available: https://doi.org/10.7250/csimq.2017-10.02

Balancing of client needs, functional capabilities, and associated development costs have long been acknowledged in the software engineering community. The Boehm's spiral model [2] for software development emphasized that in every iteration before adding new features a cost-benefit analysis should be conducted. Taudes et al. use option pricing models to evaluate information technology investment decisions and apply these models in software upgrading case study [3]. These early works focus on individual software applications rather than on software families and related services. Software product line engineering [4] investigates the problem of constructing the software solutions efficiently from the vendor perspective. Clients on the other hand are interested in the most feasible and efficient solution meeting their specific requirements. This problem is investigated in research on software selection [5] though this research mainly takes into account information available to a client, i.e., the vendor perspective is neglected. However, vendors have a wealth of information about their software's use by different clients [6]. This information could include contextual information about specific operating circumstances of individual clients as well as information dependencies between specific configurations of the solutions and the achieved performance. It is argued that by sharing historical context and performance data vendors and clients can collaborate for finding the right configuration for every client. This way every client would be provided a configuration appropriate for its operating context as well as estimates of expected performance of the solution [7]. From a service science perspective, the collaboration between client and vendor (i.e., co-creation of value) is considered as precondition for successfully implementing the service part of software-service bundles. Data and knowledge sharing has been shown to create substantial mutual benefits by suing techniques such as vendor managed inventory [8] and best practice based configuration of enterprise applications [9].

In order to enable the aforementioned approach, a framework for defining contextual information and performance objectives is required. Recently, capability driven development (CDD) has been proposed as an approach for ensuring that solutions can be delivered in different contexts at the desired level of performance [10]. The approach presumes that rather than providing a simple business solution the vendor possesses certain capabilities and is able to provide such a capability to its clients facing different operating circumstances. It is a model based approach and encompasses three development phases: 1) capability design explicitly defines performance goals, context factors affecting capability delivery, and context-dependent capability delivery solutions; 2) capability delivery phase concerns monitoring of context and performance data and adjusting the solution in response to changes in these data; and 3) feedback phase provides information for updating of the initial design.

The objective of this article is to elaborate a CDD based method allowing collaboration between vendor and client in selection of the right configuration of software-service bundles and continuous improvement of the selected configuration. It is assumed that a vendor has multiple clients. The clients share usage information about the software-service bundle. In the case of preparing a bundle for a new client, this information is used to create a decision-making matrix for selecting an appropriate configuration for this new client. The new client usually starts with a minimum satisfactory configuration; performance of this configuration is continuously monitored and if necessary the client upgrades its configuration which here is referred to as evolutionary development in analogy to evolutionary software development [11].

The main contributions of the article are: 1) combination of vendor and client perspectives in an information sharing based method for selection of software-service bundles; and 2) selection of software-service bundle as an interplay among contextual data and performance objectives (i.e., selection is made in a context-aware performance driven fashion). The rest of the article is organized as follows. Section 2 provides overview of the method. Section 3 elaborates stages of the evolutionary development. The application example is provided in Section 4. Related work is reviewed in Section 5. Section 6 summarizes findings and future work.

2 Method Overview

The evolutionary development method for software-service bundles is based on the CDD approach and uses the capability model underlying the software-service bundle as a starting point for providing appropriate configurations to clients.

2.1 Problem Statement

The vendor offers its clients a software-service bundle *S*. The software-service bundle consists of a software product, know-how and supporting services ranging from helpdesk to business process outsourcing. *S* is designed in a way to deliver desired performance in different contextual situations, i.e., the vendor possesses the capability of providing the software-service bundle.

S is provided in one of *N* configurations $O_i,\ldots,O_N$ and the configurations differ by their price and other characteristics. Delivery of *S* depends on *M* context factors $C_1,\ldots,C_M$ and its performance is measured by *L* key performance indicators $K_1,\ldots,K_L$. Combinations of values of the context factors yield a context situation describing specific solution delivery circumstances. It is assumed that certain configurations provide better performance for specific context situations than other, i.e., they are better suited for these context situations. For instance, a configuration including an outsourcing service works better in the case of highly variable demand for troubleshooting services.

There are *P* clients using one of the configurations. It is assumed that existing clients have an incentive to share anonymized values of context factors and key performance indicators (KPIs) during operations of the software-service bundle.

Figure 1 illustrates the software-service bundle selection problem. The illustration shows that the vendor maintains a set of predefined configurations, which are used by the existing clients. The usage information from the clients is accumulated in the performance and context data database. A new client together with the vendor identify his context situation and use it to select an appropriate configuration.

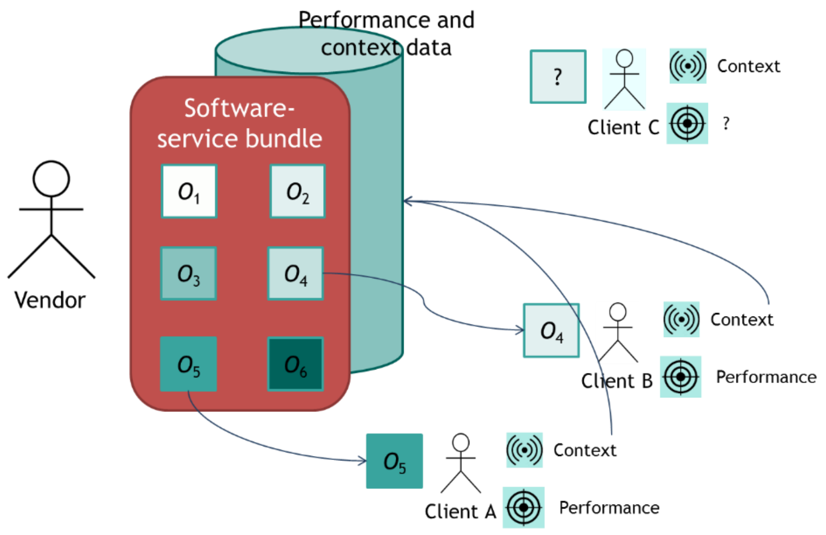

Figure 1. Contextualized and performance driven software-service bundle selection problem

Two decision-making challenges are: 1) to select appropriate configuration for a new client; and 2) to upgrade configurations used by existing clients in the case of changing circumstances or unsatisfactory performance. In the former case, selection is performed by matching a context situation of the new client with context situations supported by the vendor of the software-service bundle. In the latter case, an existing client switches from one configuration to another to adapt to the changing circumstances.

2.2 Evolutionary Development Process

The aforementioned challenges are addressed following an evolutionary development process (Figure 2). A vendor uses the CDD approach [10] to develop a capability model. The model specifies capability delivery goals, context, and solutions (i.e., software-service bundles and their appropriate configuration) for capability delivery offered to clients (see Section 2.3 for further discussion). The capability model covers all configurations supported by the vendor. Relationships among context situations and configurations are described in a capability support matrix (CSM). The matrix indicates configurations suitable for a particular context situation. It is used by the vendor and clients to find appropriate solution for client needs. CSM is developed on the basis of historical data analysis or according to judgment of the vendor. CSM defines whether a configuration meets client requirements while a service-software bundle (SSB) selection model addresses business value concerns. It evaluates expected returns from using a particular configuration. These expected returns are computed according to the goals specified in the capability model.

Upon engaging a new client, its typical context situation is assessed and the most beneficial configuration supporting this context situation is selected. The most beneficial configuration is evaluated using the SSB selection model, which is configured according to the needs of the new client by weighting his most important selection considerations. The selected configuration is provided to the client. It is used for capability delivery and delivery performance is monitored using the indicators defined in the capability model. If performance targets are not achieved or context values venture outside the defined context element range, the capability delivery solution is adjusted. Potential adjustments are: 1) selection of a more appropriate configuration; or 2) designing a new solution. The capability monitoring and adjustment are performed cyclically and the capability delivery solution evolves according to the business requirements of the client. The vendor accumulates capability delivery performance and context data from multiple clients and uses this information to update the capability delivery solution and validate CSM.

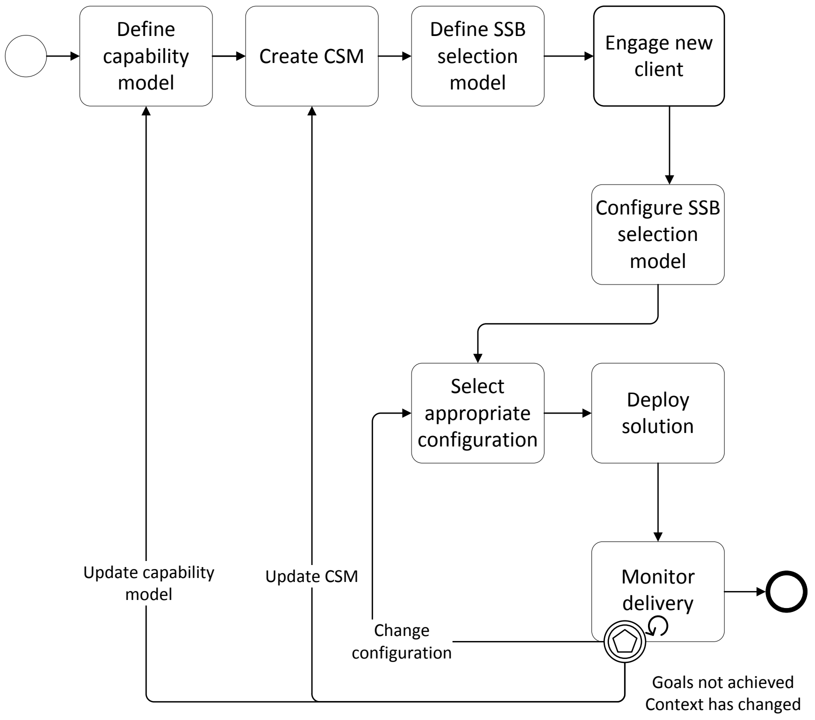

Figure 2. The evolutionary capability development process

2.3 Capability Modeling

The capability model defines vendor's ability and capacity to provide a solution to clients facing specific circumstances. Figure 3 provides a simplified overview of the key elements used in capability modeling as well as their relation to the configuration concept used in this article. Goals are business objectives the capability allows to achieve. They are measured by KPIs. The capability is designed for delivery in a specific context as defined using context elements. The context elements name factors affecting the capability delivery while context situations refer to combinations of context element values. The process element specifies a capability delivery solution. Process variants describe the capability delivery process for a specific context situation while the associated configuration of the solution encompasses all technical, human, and knowledge resources necessary to execute the process. Consequently, the configuration enables capability delivery. Multiple configurations can be used for capability delivery.

A configuration can include multiple process variants. The client can switch from one process variant to another or invoke them simultaneously during solution delivery depending on context situation.

In order to ensure that capability is delivered as expected in different contextual situations, adjustments are used to adapt capability delivery [12]. The adjustments take context data and KPIs as input and evaluate potential changes in capability delivery. Changing the configuration from one to another is perceived as one of the possible adjustments. Thus, the adjustments can be used to implement the SSB selection model.

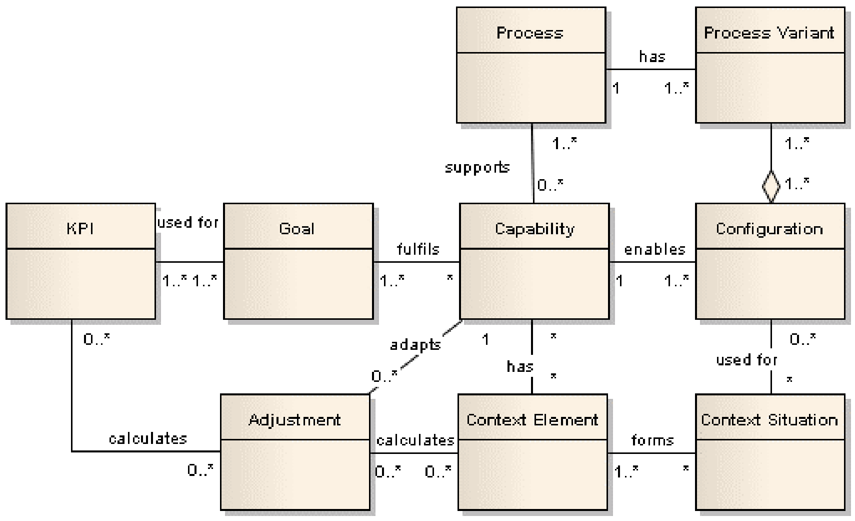

Figure 3. Key concepts of capability modeling

From the evolutionary development perspective, the key aspects are that: 1) capability can be delivered in different context situations while each individual client faces just some of these context situations; 2) process variants specify a solution for dealing with one or several context situations (for software-service bundle these variants cover both, different process variants in the software product if this product has a process-oriented architecture and different variants for the service bundled with the software product as a part of configurations); and 3) relationships among context situations, performance, and process variants are not necessarily known in advance and can be induced from the solution's usage data.

3 Evolutionary Development Stages

The evolutionary development process includes two distinctive phases: 1) design stage – when the initial configuration of the software-service bundle is selected and deployed for a new client;

and 2) delivery stage – when the software-service bundle is used by the client and it is adjusted according to changing circumstances.

3.1 Design Stage

During the design stage, SSB is selected by matching context situations and by evaluating expected business value of tentative configurations.

3.1.1 Matching Context Situations

At the beginning of the design stage the vendor develops a capability model corresponding to the software-service bundle to be provided to clients. Every context factor used in selection of the configuration has a finite set of values or context range $CR_i = (cr_{i1},...,cr_{iT_i})$, where T_i represents a number of values for the *i*th context element. These values are obtained by categorizing actual values of context observations also referred as to measurable properties [13]. The categories enable for relative comparison of clients.

Combination of context element values form the context range yields a set of context situations $(CS_1,...,CS_H) = CR_1 \times ... \times CR_N$ (*H* is the number of context situations). CSM is defined as a matrix with elements $a_{ij}, i = 1,...,H, j = 1,...,N$, where $a_{ij} \in \{0,1\}$ relates context situations to suitable configurations. The matrix element $a_{ij} = 1$ indicates that configuration O_j is suitable in the case of context situation CS_i. The same configuration could be suitable for multiple context situations. One configuration could encompass multiple process variants.

Upon engaging a new client, its most plausible context situation is identified as CS^{new}. Appropriate solutions are identified by

$$\Omega = (O_j \mid a_{ij} = 1 \wedge CS^{new} \in CS_i) \tag{1}$$

where Ω is a set of matching configurations for the new client.

Eq. 1 finds matching configurations appropriate for the context situation faced by the new client. If no appropriate configuration is available, the client has a choice to select a configuration having the highest level of overlapping with support context situations.

3.1.2 Value-Based Selection

The context matching step yields a set of tentative configurations without a regard to their potential business value balancing costs and benefits associated with using a particular configuration. It assumed that the value-based selection can be conducted on the basis of information provided in the goal model of the overall capability model. It selects the configuration providing the best expected business value what is formally expressed as

$$\max_{O_j \in \Omega}(V(O_j)) = F(K_{1t}^{new},...,K_{Lt}^{new}, w_1,...,w_L)\,. \tag{2}$$

V is an estimated SSB value for the client depending on the configuration selected, the superscript refers to the new client, $K_{1t}^{new},...,K_{Lt}^{new}$ are values of KPI at time period *t*, $w_1,...,w_L$ are weights indicating relative contribution of each KPI toward the overall business value and *F* is a functional relationship between *V* and KPI. It is assumed that KPIs are positively correlated with business value and large values are preferable (if small values are preferred then the negative value of KPIs is included in Eq. 2).

During the design stage (*t* = 0) some of the KPI values are known, some can be estimated and some are known or irrelevant. The weights set to zero for the latter KPI. An example of the known KPI is fixed licensing fee while fee per every service request can only be estimated during the design stage.

The value calculation function *F* is specified in the capability model as an adjustment. That enables specification of any desired functional form of the function and the same specification can be used during the run-time to evaluate a need for switching to a new configuration.

Assuming that a client sets target values for KPI $\kappa_1^{new},....,\kappa_L^{new}$, where the subscript *l* refers to the KPI, an exact form of the value function can be specified as a weighted sum of ratios between KPI values and their targets (Eq. 3).

$$V = \sum_{i=1}^{L}\left(w_i \mathrm{K}_{it}\right), \tag{3}$$

where $\mathrm{K}_{ij} = K_{it}^{new}\left(\kappa_i^{new}\right)^{-1}$ if KPI and business value are positively correlated, otherwise $\mathrm{K}_{ij} = \kappa_i^{new}\left(K_{it}^{new}\right)^{-1}$.

The configuration selected based on the above considerations is set up for the client and made ready for operations. In this context, there might be a setup time required for deploying the configuration.

3.2 Delivery Stage

During the delivery stage, actual context situations and delivery performance are monitored. The performance monitoring is carried out by gathering real-time values of KPI K_{it}^{new}, where superscript identifies the client, the subscript *i* refers to KPI being measured and *t* refers to the measurement time. The current value is compared to the performance target. If $K_{it}^{new} < \kappa_i^{new}$ then the *i*th performance objective is not met and a recommendation to revise the solution is issued. Obviously, one should evaluate to what extent the software solution is responsible for underperformance.

The context monitoring is performed by comparing the observed context situation CS_t^{new} for the new client at the *t*th time moment to the context situations supported by the current configuration, i.e., relationship $CS_t^{new} \in \mathbf{CS}_O$, where $\mathbf{CS}_O$ is a set of context situations supported by configuration O_j used by the client. If the relationship does not hold then a warning is issued notifying that the observed context situation is not explicitly supported by the current configuration. The context monitoring serves as an advanced warning system to potential performance deterioration since it is not known whether the current configuration is suitable for the observed context situation. That might lead to an unexpected behavior.

3.3 Evolution

Evolution in software and systems engineering commonly is described as the process of gradually adapting a (software) system to requirements and changes which could not be anticipated at the time of systems development, but occurred during its time of use [14]. Three main causes for the evolution of a software-service bundle can be distinguished: (1) the client's requirements regarding the service remain unchanged but runtime shows that the features of the software do no longer meet these requirements, (2) the client demands regarding the service change and require changes in the software, and (3) the vendor's requirements to the software change and require changes in the software-service bundle.

In this article, the focus is on the first cause, i.e., when the software during runtime does not longer meet the requirements. Violations of performance objectives or observation of unsupported context situations triggers a warning suggesting an upgrade of the current configuration. In response to this warning a client might decide on upgrading the current configuration by selecting a more suitable configuration from CSM. This is a suitable approach if the actual context situation is different from the one identified during the design stage or it has changed. However, if underperformance is observed for the supported context situation and it is attributed to the software product then the vendor might need to reevaluate CSM or a special software-service bundle has to be developed for the particular client.

For the second evolution cause, changing client demands, new agreements between vendor and client regarding the software-service bundle and its configuration are required. The third cause, changing requirements to the software, might in some cases lead to a new software version which does not have any effects on the agreement with the client, i.e., the client would not notice that there is a new version. But a new software version could also have effects on the service quality. If this is the case, new agreements with the client might be needed.

4 Application Example

Business processes in many industries require collaboration among two or more companies. Often this collaboration involves business information exchange in a form of data exchange messages [15]. Such messages can contain errors, which need to be corrected before further utilization of the information. The correction of errors might require manual interventions which will be referred to as “clearing services”. The example discussed in this article is motivated by a real-life case in the energy industry, where a software development company provides software for exchanging energy consumption data as well as associated business process outsourcing services for handling data errors [16].

4.1 Case Description

Since the end of the 1990’s, energy industry in many European countries has undergone serious changes due to market liberalization and the separation of energy grid and energy production. Energy distribution companies are facing a changing business context caused by (a) new regulations and laws from regulating authorities and (b) innovative technical solutions, e.g., for implementing smart grids. Software solutions in this area have to support the business functions within energy distribution companies and data exchange between the different market roles (market communication). Examples for typical business functions are assets accounting, processing and examination of invoices, automatic billing initiated by meter readings, meter data evaluation, maintenance management (disposition, workforce management and mobility), and order management. Market communication related to these business functions includes exchange of information about consumption of energy, changes in customer data, or usage of energy grid – to name only some examples. Given the constantly rising complexity of the market communication, public utilities and other energy distributors increasingly implement outsourcing of their business processes to external service providers.

The application case considered in this article is a medium-sized company in Germany offering a software solution for energy distribution companies supporting the abovementioned business functions, and – based on the same software solution – business process outsourcing services for their clients. Thus, the company acts both as an independent software vendor (ISV) and as a business service provider (BSP). The software solution is widely used by public utilities in Germany, in particular for *electricity*, *natural gas*, *district heating*, and *water*. The focus of the BSP part are business processes that deal with market communication. Given the complexity of the market relationships, exchanged data may get into conflict or create inconsistencies with other data. If this is the case, manual intervention or clearing is required. Thus, the BSP acts as a “clearing center” for the occurring exceptions if they are covered by the contractual agreement between client (e.g. a public utility) and the BSP.

Within this clearing center, different kinds of services are offered, some of them specializing on the kind of utility offered (energy, gas, water), others offering different levels of clearing support. These services usually are bundled with the case company’s software solution, i.e., the client buys or leases the software solution and orders clearing services. The case company offers several software-services bundles, two of which are “clearing support” and “enhanced exception handling”:

- *Clearing support.* If exceptions occur during data exchange, the erroneous data records are registered and forwarded to the BSP. Each record is analyzed and, depending on the contractual agreement with the client, the record is either corrected by the BSP or forwarded to the responsible unit at the client side. Examples for such clearing support software-service bundles are processing and consolidation of meter readings which are offered in different variations to energy, gas, and water suppliers. The software part provides automated processing of meter readings (mass, data; several thousands of records per week) exchanged between grid operator and metered service provider. The service part complements the automated processing with manual clearing of exceptions, which is done with the same software product, but requires manual entry of data.
- *Enhanced exception handling.* In some cases, erroneous data records cannot be corrected by the clearing support because of so far unknown errors or missing data. In these cases manual intervention by highly skilled experts is required which are called “knowledge workers”. This manual process basically follows a case handling strategy and sometimes requires interaction of BSP and client. Similar to the clearing service above, the enhanced exception handling is also offered to energy, gas and water suppliers. Evolution of these services often means to adapt the strategy of how to assign different qualifications of knowledge workers and software functionality in order to fulfill the contractually agreed service level.

4.2 Software-Service Bundle

From the perspective of software-services bundles, the situation described in Section 4.1 can be generalized as follows: A vendor offers business information processing services. These services typically consist of data processing software, data processing services and, – if required – clearing services by the back office staff of the vendor based on knowledge about the most common data exchange exceptions. The data processing services ensure business information processing on behalf of the client. Clients can choose between doing data processing and clearing in-house or outsourcing it to the vendor. Figure 4 shows an overall data exchange process from the client’s perspective.

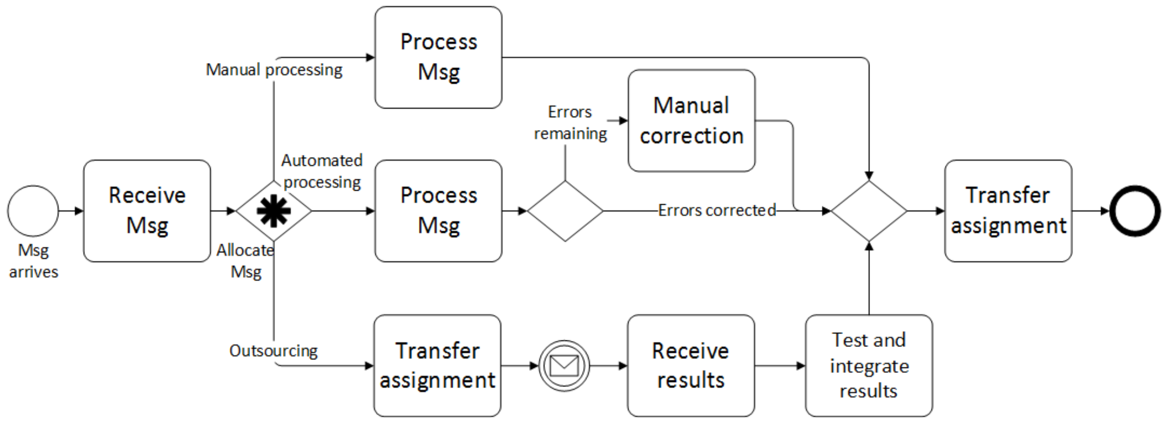

Figure 4. The overall business information exchange process (Msg – message)

The client receives a message. Depending on a decision-making logic the messages are processed along one or several process branches. The first branch represents a manual processing (client’s employees correct errors). The second branch represents an automated processing using the knowledge base on common exceptions provided by the vendor. However, some of the exceptions might require manual intervention (“clearing”). A client uses the outsourcing service provided by the vendor to deal with exceptions in the third branch. The client transfers the exceptions to the outsourcing service and receives back the remedied data. Multiple branches can be used simultaneously. For example, the client mainly uses in-house automated processing and invokes the outsourcing service only if internal resources are overloaded. This decision is made during the service delivery. Nevertheless, the solution should be configured in a way to support both automated in-house processing and usage of the outsourcing service.

The process execution goals are timely processing of all messages and handling of all exceptions. The main context factors affecting the process execution are the number of data exchange messages received or processing load and load volatility. The load volatility characterizes variations in the processing load what might have adverse consequences on scheduling of resources assigned to the manual processing.

4.3 Model

The vendor possesses the data exchange capability provided to its clients by means of the software-service bundle. The data exchange capability model (Figure 5) is created using the concepts defined in Section 2.3. Goals, context, and process variants are the main elements of the capability model important for the software solution selection method.

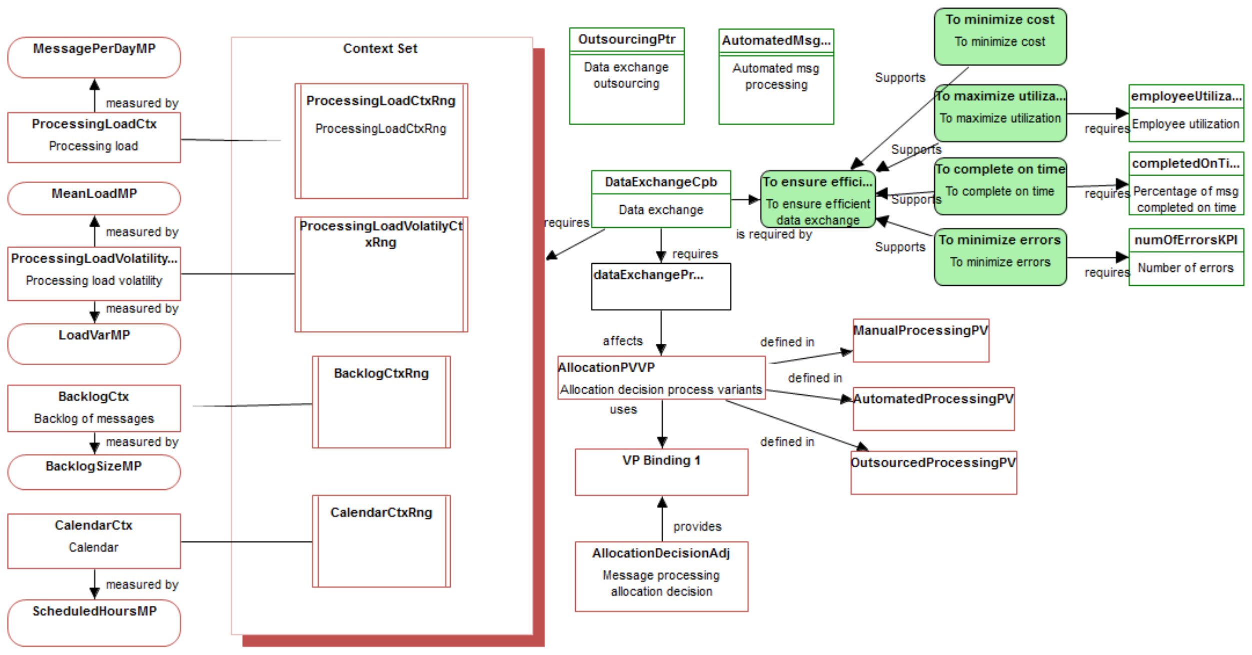

Figure 5. Data exchange capability model

The capability goals are timely data processing, correction of data exchange errors as well as cost minimization and efficient utilization of resources involved in the data exchange process. These goals are measured by the corresponding KPIs. The timely data processing goal is measured as the processing time KPI K_{PT}. The cost efficiency goal is measured by return on investment KPI, which is calculated using KPIs measuring revenues from message processing, fixed configuration setup cost and variable message processing cost.

The context elements affecting the capability are defined in Table 1. The processing load context element is measured by the number of messages received per day and it assumes values from the range of values. Not all context elements are used in configuration selection for the software-service bundle. Current backlog and schedule context elements are used for run-time decision-making.

The capability model also defines the overall data exchange process including three processing variants (Figure 4). These process variants serve as the basis for defining configurations of the software solution. The Allocate messages gateway represents decision logics for run-time allocation of messages among process variants if several of them are included in the configuration.

Table 1. Context values and their context ranges

Context element	Context element value range
Processing load (C_{PL})	Low, medium, high
Processing load volatility (C_{LV})	Low, medium, high
Backlog (number of messages waiting for processing)	0,…,1000
Calendar (scheduled hours for human resources)	0,…,100

Table 2. Capability support matrix for data exchange software-service bundle

Processing load level	Load volatility	O_1	O_2	O_3
Low	Low	1		
Low	Medium	1	1	
Low	High		1	
Medium	Low		1	
Medium	Medium		1	1
Medium	High			1
High	Low		1	
High	Medium			1
High	High			1

Three configurations are offered to clients: O_1 – manual processing of data exchange exceptions; O_2 – automated processing of data exchange exceptions; and O_3 – combination of automated processing of data exchange exceptions with availability of exceptions handling outsourcing services. The configurations also vary by the fixed setup prices. The setup price of O_1 is assumed as a baseline or 100%. The setup price for O_2 in the terms of the baseline price is 250%, and it is 300% for O_3.

CMS is prepared according to the vendor's expertise and historical data (Table 2). The matrix lists context situations as combinations of values of C_{PL} and C_{LV} context elements. It shows that, for instance, configuration O_1 is suited for $CS_1 = \{\text{low,low}\}$. If multiple configurations are suitable for a context situation, then the value-based selection is used to select one of the configurations.

4.4 Results

The aforementioned capability model provides a foundation for delivering data exchange solutions to clients. A simulated experiment is conducted to illustrate the software-service bundle selection method. It simulates a flow of data exchange messages for a single client and the client attempts to process these messages using one of the solutions provided by the vendor. Execution of manual data processing activities in all configurations requires human resources drawn from a limited pool of resources and has a variable duration depending on complexity of exceptions.

The message flow D_t varies over time and is described as an autoregressive process $D_t = \rho + \alpha D_{t-1} + \varepsilon$, where ρ and α are coefficients defining process shape and $\varepsilon = N(0,\sigma)$ is normally distributed with the standard deviation σ. The average flow of messages $\mu = \rho / (1-\alpha)$ and affects the processing load context element. The relationship between D_t and value of the processing load context element C_{PL} is expressed as depicted in Eq. 4.

$$C_{PL} = \begin{cases} \text{low,if } \mu < 100 \\ \text{medium, if } 100 \le \mu < 1000 \\ \text{high,if } \mu \ge 1000 \end{cases} \qquad (4)$$

The coefficients α and σ affect processing load volatility, i.e., larger values of these coefficients result in a more volatile message flow. In this experiment $\alpha = 0.8$ and $\sigma = \mu / 5$. The value of ρ is varied to evaluate different context situations: 1) in the first experiment (EXP1) ρ is set to 10 to evaluate a low processing load situation; and 2) in the second experiment (EXP2) ρ is increased from 10 to 100 during the course of message processing simulation to evaluate the impact of changes in context. Numerical values used in the experiments are practically grounded though do not represent actual observations.

A new client defines that its typical context situation $CS^{new} = \{\text{low,low}\}$. That is supported by either O_1 or O_2. The most suitable configuration is selected according to the estimated value. In this case, the value is determined by the return on investment KPI (i.e, all other KPIs have zero weights). The return on investment KPI (Table 3) is calculated as a ratio between profit and total costs and assuming that in the case of O_1 there is 20% difference between single message processing fee and costs, while this difference is 180% for O_2. The calculation is performed for three years period taking into account the expected load. At the design stage, KPI values can only be estimated and actual values are evaluated once the configuration is up and running.

Since O_1 has a better expected value of the return on investment KPI, this configuration is setup for the client. That implies client receiving data exchange software and using manual exceptions handling.

Table 3. Estimated KPI values at the design time

KPI	***O*₁**	***O*₂**
Return on investment	14%	10%
Revenues	273 750	273 750
Fixed setup cost	10 000	150 000
Variable message processing cost	229 950	98 550

Figure 6.a shows monitoring results for EXP1. It includes values of $K_{PT,t}^{new}$ and the threshold value $\kappa_{PT}^{new} = 120$ minutes. The processing load context element C_{PL} is constant while load volatility C_{LV} exhibits slight variations and occasionally assumes medium value not explicitly supported by the current configuration O_1. More importantly, $K_{PT,t}^{new}$ frequently exceeds the threshold value what triggers a recommendation to reconsider the configuration. CSM suggests that O_2 is appropriate for dealing with $CS_2 = \{\text{low,medium}\}$. The switch to O_2 takes place at time period 250 minutes. One can observe that performance is significantly improved.

EXP2 simulates a permanent change of the context situation and C_{PL} assumes medium value. O_2 is suitable for both $CS_4 = \{\text{medium,low}\}$ and $CS_5 = \{\text{medium,medium}\}$, and experimental results (Figure 6.b) show that O_2 delivers satisfactory performance after the change in the context. However, $K_{PT,t}^{new}$ is close to its threshold. Assuming, that a similar behavior is observed also for other clients using O_2 in similar conditions, the vendor might decide on updating CSM and recommending to use exclusively O_3 in context situation $CS_5 = \{\text{medium,medium}\}$. The simulated switching takes place at time period 750 minutes, when O_3 is deployed. This change results into reduction of the processing time.

The experimental results demonstrate that context observations can be used to drive selection of appropriate configurations on the basis of the common capability model.

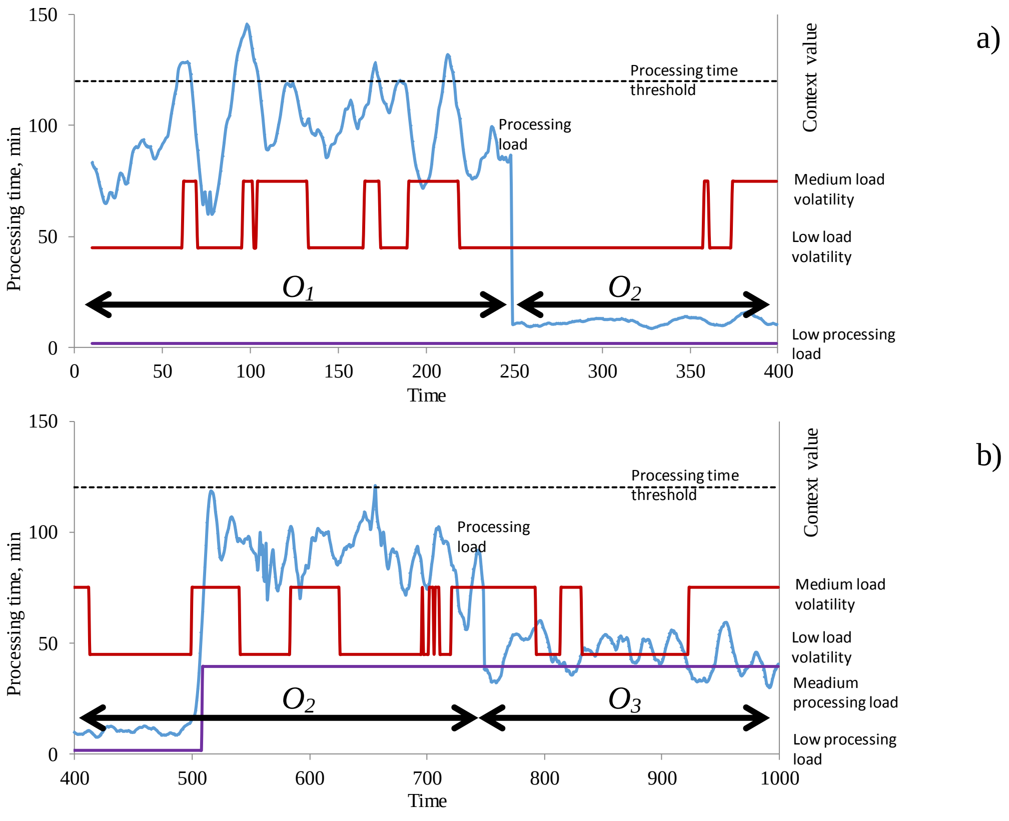

Figure 6. Dynamics of simulated delivery results and configurations used: a) EXP1 and b) EXP2

5 Related Work

Related work can be found in the areas of selection of packaged applications and version management. The multi-objective methods for selecting packaged applications allow for comprehensive evaluation of offerings by different vendors [5]. The selection is based on the generic set criteria and the alignment of these criteria with business needs is not ensured. Importance of explicitly accounting for various business and operational goals in selection of commercial-of-the-shelf applications is emphasized in [17]. Capilla et al. [18] describe methods for designing different versions of business software depending on the application context. These works focus on software architecture and design issues rather than the software management decisions. Evolutionary development in software engineering [11] concerns the operational level evolution of atomic development requirements. A value-driven approach to design of service-oriented systems is proposed by Thomas and vom Brocke [19]. They use modeling to identify various options for designing services and creating orchestrations. The cost-benefit analysis of various options is conducted considering investment in different layers of Service Oriented Architecture (SOA).

In literature the adaptation and evolution of software systems to changing situations is also addressed in the field of "system maintenance". Much work in this field observes that systems are insufficiently prepared for quick adaptation and evolution, i.e., there is the criticism of poor flexibility [20]. According to [21], 65% of system modification costs are caused by adapting the system to changed requirements when the system already has been deployed to the client. Several approaches have been proposed aiming at more agile and flexible software systems, among them is shortening the software development process by improving the methods applied in this process [22]. In this context, [23] emphasizes that software engineering techniques are not sufficiently prepared for application areas, where system boundaries are not static, but are subject to continuous change.

Continuous Software Engineering (CSE) [24] develops methodologies, concepts and techniques for evolvable software systems. The central goal of CSE is producing long-living, evolution-capable software systems, which are of high quality and can be forward developed continuously [25]. An essential part of the CSE approach is to achieve consistency and transparency between (a) all artifacts of a software development process within a development cycle (e.g. specification, design, and implementation) and (b) the various forward development cycles of a software system and their modifications. This requires the identification of invariants and dependencies in order to predict the potential impact of initial and induced modifications.

Evolution of information systems also attracts substantial research work. Here, an important topic to be tackled is achieving more agile and flexible methods for information system development [26]. Furthermore, the implementation of more flexible systems based on component-based approaches is discussed [27]. [28] argues that there are different forms of flexibility in software development which can be analyzed from a control theory perspective. The method proposed relies on several existing methods. Goal modeling techniques [29] help to identify the relevant objectives. Business activity monitoring techniques allow measuring the process performance according to the specified goals and to identify significant changes in performance [30]. Context model techniques [31] help to identify relevant context factors and to represent their impact on process design. The software-service bundle selection method developed in this article extensively uses categorized context values. A similar approach is used by [32] to model context-driven business processes. Furthermore, there is related work in the field of service management from a service science perspective. In recent years, the perspective on what is characterizing a service has shifted from an intangible product to a more process-oriented focus [33]. A service process "… can be viewed as a chain or constellation of activities that allow the service to function effectively" [34 p. 68]. Existing work tries to understand service processes from three overall perspectives: input, transformation process and outcome [35]. In contrast to manufacturing-based production processes, also customers provide significant input in service processes [36]. This is clearly visible in our product-service bundle. However, this input is not only limited to one customer, but also multiple customers which is also acknowledged by service science [37]. The transformation process "entails the service delivery and consumption process, and involves customer participation in the service delivery/consumption process" [35 p. 1016]. For software-service bundles, we thus can divide this perspective into the continuum of service co-production [38] and consumption process flow. The latter is considered as a characteristic of services from a service operations management perspective. The final outcome of the service is determined by the service provider as well as by the service beneficiaries [39].

6 Summary and Future Work

The article presented and discussed the method for evolutionary development and configuration of software-services bundles. The article showed that the vendor-client collaboration for selecting the most suitable configuration of a given software-service bundle is feasible and represents value co-creation between the vendor and client. The capability concept and the CDD approach have been useful for this method to provide the common basis for defining software solution and explicitly representing relationships among performance, context, and solutions. The method so far is evaluated only using a simulation approach, and experiences from real-world application cases are needed for further validation.

There are multiple directions of further research. Updating the capability support matrix according to monitoring results is an important part of the method that requires further elaboration. Classification and machine learning methods can be used for these purposes. The decision to upgrade the solution is not an automated decision and usually involves a number of considerations (e.g., business relations) not covered by the evolutionary development method. Switching to a new configuration incurs additional costs. This factor also could be incorporated into the decision-making process to evaluate cost-benefit aspects currently not considered in the article.

In this article, process-oriented applications are considered. That could be generalized to other types of applications and services assuming that multiple alternative capability delivery solutions exist.

Another aspect which so far has not been sufficiently covered in our research is the business value of the overall approach, i.e., is it worthwhile to invest in context modeling, definition of capability matrix, software-service bundling, and evolutionary development? So far, our simulation primarily addresses the evolutionary development aspect. When it comes to the business value of context modeling, results from the Capability as a Service (CaaS) project indicate that context modeling and capability design result in transactional and informational benefits [40]. This experience from CaaS and the positive result of our simulation can be taken as indication that also the combination of both parts can be expected to have a positive business value. However, this needs a more thorough investigation in future work.

From a service science perspective, the co-production of services has to be seen as a continuum which "can vary from none at all to extensive co-production activities by the customer or user" [41, p. 8]. When specifying service processes, it thus has to be highlighted which tasks are performed by which entity of the service system or service system network. For software-service bundles, modeling of the service process with explicit distribution of tasks between vendor and client could be a way to further optimize gathering of relevant data. This will be part of future work.

References

[1] D. Miller, "The Problem of Solutions: Balancing Clients and Capabilities," Business Horizons, vol. 45, no. 2, pp. 3–12, 2002. Available: https://doi.org/10.1016/s0007-6813(02)00181-7

[2] B.W. Boehm, "Software Risk Management: Principles and Practices," IEEE Software, vol. 8, no. 1, pp. 32–41, 1991. Available: https://doi.org/10.1109/52.62930

[3] A. Taudes, M. Feurstein and A. Mild, "Options Analysis of Software Platform Decisions: A Case Study," MIS Quarterly, vol. 24, no. 2, pp. 227–243, 2000. Available: https://doi.org/10.2307/3250937

[4] K. Pohl, G. Böckle and F. Van der Linden, "Software Product Line Engineering: Foundations, Principles, and Techniques," Springer-Verlag Berlin Heidelberg, 1st edition, pp. XXVI–467, 2005. Available: https://doi.org/10.1007/3-540-28901-1

[5] A.S. Jadhav and R.M. Sonar, "Evaluating and Selecting Software Packages: A Review," Information and Software Technology, vol. 51, pp. 555–563, 2009. Available: https://doi.org/10.1016/j.infsof.2008.09.003

[6] H.H. Olsson and J. Bosch, "Towards Continuous Customer Validation: A Conceptual Model for Combining Qualitative Customer Feedback with Quantitative Customer Observation," in 6th Int. Conference on Software Business, ICSOB 2015, LNBIP 210, pp. 154–166, 2015. Available: https://doi.org/10.1007/978-3-319-19593-3_13

[7] J. Grabis and K. Sandkuhl, "Selection and Evolutionary Development of Software-Service Bundles: A Capability Based Method," in 28th Conference on Advanced Information Systems Engineering, CAiSE 2016 held in conjunction with ASDENCA, Lecture Notes in Business Information Processing, vol. 249, pp. 3–14, 2016. Available: https://doi.org/10.1007/978-3-319-39564-7_1

[8] A. Vigtil and H.C. Dreyer, "Critical Aspects of Information and Communication Technology in Vendor Managed Inventory," 2008. Available: https://doi.org/10.1007/978-0-387-77249-3_46

[9] J.E. Scott and L. Kaindl, "Enhancing Functionality in an Enterprise Software Package," Information & management, vol. 37, no. 3, pp. 111–122, 2000. Available: https://doi.org/10.1016/s0378-7206(99)00040-3

[10] S. Bērziša, G. Bravos, T. González, U. Czubayko, S. España, et al., "Capability Driven Development: An Approach to Designing Digital Enterprises," Business & Information Systems Engineering, vol 57, pp. 15–25, 2015. Available: https://doi.org/10.1007/s12599-014-0362-0

[11] I. Sommerville, "Software Engineering," Pearson, 10th edition, 816 p, 2015.

[12] J. Grabis and J. Kampars, "Design of Capability Delivery Adjustments," in 28th Conference on Advanced Information Systems Engineering, CAiSE 2016 held in conjunction with ASDENCA, Lecture Notes in Business Information Processing, vol. 249, pp. 52–62, 2016. Available: https://doi.org/10.1007/978-3-319-39564-7_5

[13] J. Grabis and J. Stirna, "Advanced Context Processing for Business Process Execution Adjustment," in CAiSE 2015 Workshops, LNBIP, vol. 215, pp. 15–26, 2015. Available: https://doi.org/10.1007/978-3-319-19243-7_2

[14] T. Mens, "Introduction and Roadmap: History and Challenges of Software Evolution," in Software evolution Springer Berlin Heidelberg, pp. 1–11, 2008. Available: https://doi.org/10.1007/978-3-540-76440-3_1

[15] A. Schmidt, B. Otto and H. Österle, "Integrating Information Systems: Case Studies on Current Challenges," Electronic Markets, vol. 20, pp. 161–174, 2010. Available: https://doi.org/10.1007/s12525-010-0037-8

[16] K. Sandkuhl and H. Koc, "On the Applicability of Concepts From Variability Modelling in Capability Modelling: Experiences From a Case in Business Process Outsourcing," CAiSE 2014 Workshops, LNBIP, vol. 178, pp. 65–76, 2014. Available: https://doi.org/10.1007/978-3-319-07869-4_6

[17] C. Alves, X. Franch, J.P. Carvallo and A. Finkelstein, "Using Goals and Quality Models to Support the Matching Analysis During COTS Selection," COTS-Based Software Systems, Lecture Notes in Computer Science 3412, pp. 146–156, 2005. Available: https://doi.org/10.1007/978-3-540-30587-3_25

[18] R. Capilla, O. Ortiz and M. Hinchey, "Context Variability for Context-Aware Systems," Computer, vol. 47, pp. 85–87, 2014. Available: https://doi.org/10.1109/mc.2014.33

[19] O. Thomas and J. Vom Brocke, "A Value-Driven Approach to the Design of Service-Oriented Information Systems – Making Use of Conceptual Models," Inf. Syst. E-Business Management, vol. 8, no. 1, pp. 67–97, 2010. Available: https://doi.org/10.1007/s10257-009-0110-z

[20] R. France and B. Rumpe, "Model-Driven Development of Complex Software: A Research Roadmap," IEEE Computer Society, 2007. Available: https://doi.org/10.1109/fose.2007.14

[21] A. Al Kalbani and K. Nguyen, "Designing Flexible Business Information System for Modern-Day Business Requirement Changes," IEEE, vol. 2, 2010. Available: https://doi.org/10.1109/icste.2010.5608774

[22] J.-Y. Chung and K.-M. Chao, "A View on Service-Oriented Architecture," Service Oriented Computing and Applications, vol. 1, no. 2, pp. 93–95, 2007. Available: https://doi.org/10.1007/s11761-007-0011-2

[23] K. Bennett, P. Layzell, D. Budgen, P. Brereton, L. Macaulay and M. Munro, "Service-Based Software: The Future for Flexible Software," in 7th Asia-Pacific Software Engineering Conference, ASPEC 2000, pp. 214–221, 2000. Available: https://doi.org/10.1109/apsec.2000.896702

[24] H. Müller and H. Weber, Eds., "Continuous Engineering of Industrial Scale Software Systems," Dagstuhl Seminar #98092, Report No. 203, Schloss Dagstuhl, March 1998.

[25] M. Grosse-Rhode, R. Kutsche and F. Bübl, "Concepts for the Evolution of Component-Based Software System," Technical Report, Fraunhofer ISST, Germany, October 2000.

[26] P. Agerfalk, B. Fitzgerald and S. Slaughter, "Introduction to the Special Issue – Flexible and Distributed Information Systems Development: State of the Art and Research Challenges," Information Systems Research, vol. 20, no. 3, pp. 317–328, 2009. Available: https://doi.org/10.1287/isre.1090.0244

[27] V. Wulf, V. Pipek and M. Won, "Component-Based Tailorability: Enabling Highly Flexible Software Applications," International Journal of Human-Computer Studies, vol. 66, no. 1, pp. 1–22, 2008. Available: https://doi.org/10.1016/j.ijhcs.2007.08.007

[28] M. Harris, R. Collins and A. Hevner, "Control of Flexible Software Development Under Uncertainty. Information Systems Research, vol. 20, no. 3, pp. 400–419, 2009. Available: https://doi.org/10.1287/isre.1090.0240

[29] E. Kavakli, "Modeling Organizational Goals: Analysis of Current Methods," in Proc. the ACM Symposium on Applied Computing, SAC '04, pp. 1339–1343, 2004. Available: https://doi.org/10.1145/967900.968171

[30] J. Friedenstab, C. Janiesch, M. Matzner and O. Müller, "Extending BPMN for Business Activity Monitoring," in Proc. the Annual Hawaii International Conference on System Sciences, pp. 4158–4167, 2012. Available: https://doi.org/10.1109/hicss.2012.276

[31] H. Koç, E. Hennig, S. Jastram and C. Starke, "State of the Art in Context Modelling – A Systematic Literature Review," L.S. Iliadis, M.P. Papazoglou and K. Pohl, Eds., in Advanced Information Systems Engineering

Workshops – CAiSE 2014 International Workshops, Thessaloniki, Greece, June 16–20, 2014, LNBIP, pp. 53–64, 2014. Available: https://doi.org/10.1007/978-3-319-07869-4_5

[32] M. Born, J. Kirchner and J.P. Muller, “Context-Driven Business Process Modeling,” in Advanced Technologies and Techniques for Enterprise Information Systems, ICEIS 2009, Milan, Italy, pp. 17–26, 2009. Available: https://doi.org/10.5220/0002201500170026

[33] S.E. Sampson, “Visualizing Service Operations,” J. Serv. Res., vol. 15, pp. 182–198, 2012. Available: https://doi.org/10.1177/1094670511435541

[34] H.J. Bitner, A.L. Ostrom and F.N. Morgan, “Service Blueprinting: A Practical Technique for Service Innovation,” Calif. Manage. Rev., vol. 50, pp. 66–94, 2008. Available: https://doi.org/10.2307/41166446

[35] A.A. Yalley and H.S. Sekhon, “Service Production Process: Implications for Service Productivity,” Int. J. Product. Perform. Manag, vol. 63, pp. 1012–1030, 2014. Available: https://doi.org/10.1108/ijppm-10-2012-0113

[36] S.E. Sampson and C.M. Froehle, “Foundations and Implications of a Proposed Unified Services Theory,” Prod. Oper. Manag., vol. 15, pp. 329–343, 2006. Available: https://doi.org/10.1111/j.1937-5956.2006.tb00248.x

[37] S.S Tax, D. McCutcheon and I.F. Wilkinson, “The Service Delivery Network (SDN) A Customer-Centric Perspective of the Customer Journey,” J. Serv. Res., vol. 16, pp. 454–470, 2013. Available: https://doi.org/10.1177/1094670513481108

[38] T. Hilton and T. Hughes, “Co-Production and Self-Service: The Application of Service-Dominant Logic,” J. Mark. Manag., vol. 29, pp. 861–881, 2013. Available: https://doi.org/10.1080/0267257x.2012.729071

[39] J. Spohrer and S.K. Kwan, “Service Science, Management, Engineering, and Design (SSMED): An Emerging Discipline,” Int. J. Inf. Syst. Serv., vol. 1, pp. 1–31, 2009. Available: https://doi.org/10.4018/9781615209675.ch121

[40] J.P.C. Vega, “Supporting Organizational Induction and Goals Alignment for COTS Components Selection by Means of i*,” in Proc. the 5th International Conference on Commercial-off-the-Shelf (COTS)-Based Software Systems, ICCBSS '06, Feb. 13–16, IEEE, 2006. Available: https://doi.org/10.1109/iccbss.2006.28

[41] S.L. Vargo and R.F. Lusch, “Service-Dominant Logic: Continuing the Evolution,” J. Acad. Mark. Sci., vol. 36, pp. 1–10, 2007. Available: https://doi.org/10.1007/s11747-007-0069-6