Implementation of Composable Enterprise in an Evolutionary Way Through Holistic Business-IT Delivery of Business Initiatives: Real Industry Use Case

Ivka Ivas
Faculty of Electrical Engineering and Computing, University of Zagreb, Croatia

Keywords: Enterprise Architecture, Composable Enterprise, Business Composability, Business-IT Alignment, Business Capability Map, Value Stream, Holistic Value Delivery, EA Model.

Abstract: Service composability, introduced by service-oriented architecture (SOA), is a design principle that encourages the design of reusable services that themselves also consist of reusable services. In domain driven design (DDD), which inspired microservice architectures, the scope of composable service design is interpreted as a software solution domain, while the problem domain lies in the detached business world. This results in IT solutions that are often redundant at the enterprise level or tend to be composable only within a specific enterprise IT ecosystem as a result of the design without understanding the business domain or how the new solution fits into the overall delivery and enterprise architecture. On the other hand, it is not uncommon for company´s "business", motivated by revenue increase, to push frequent deliveries of business changes, putting pressure on company´s IT to implement quick fix solutions that only solve immediate business problems. All this leads to inconsistent and redundant software systems that increase the complexity of the organization and result in higher maintenance costs and less flexibility in implementing future changes. As a solution, this paper proposes Composable Enterprise, a business-IT approach for architecting the enterprise that introduces Business Composability and a holistic understanding of the enterprise. Business Composability is a business-IT-aligned service abstraction that starts with the notion of first applying service composability to business assets (business capabilities) to achieve the scale and pace required to realize business changes. The purpose of this paper is to provide a methodology for implementing Composable Enterprise in large, complex organisations, not as a massive, enterprise-wide rationalization and consolidation initiative, but in an evolutionary way through the joint and holistic business-IT delivery of business initiatives. The application of the proposed methodology is illustrated using a real-industry use case.

1 INTRODUCTION

In large organisations IT has traditionally been seen as a servant of the "business", existing to fulfil business needs, with the "business" deciding on the why, what and when and IT on the how. This mindset led to late involvement of IT in the process of business initiative delivery, resulting in quick fix solutions implemented only to meet immediate needs of a particular business initiative. Which led to increased complexity, higher maintenance costs, higher costs of future investments and less flexibility to implement future changes. Earlier IT engagement or a longer cycle time allowing IT to take enough time for a "proper" design does not solve the problem. When designed without understanding of the business domain and without holistic understanding on how the new solution fits into the overall delivery and enterprise architecture (EA), solutions that are designed to be modular and composable only from IT perspective tend to be redundant at the enterprise level or composable only within a specific IT ecosystem. Which also leads to increased complexity, higher costs, and less flexibility to implement business changes. Widely adopted agile practices do not help either, sometimes even the opposite, as agile practices encourage frequent deliveries of business changes, which puts pressure on IT to implement quick solutions. The root cause of this situation is a cultural split to "business" and IT, both existing as two separate words pursuing their own interests. The goal of enterprise IT is to lower costs, which can be achieved through reusability, modularity, and

Ivas, I.
Implementation of Composable Enterprise in an Evolutionary Way Through Holistic Business-IT Delivery of Business Initiatives: Real Industry Use Case.
DOI: 10.5220/0012728400003690
Paper published under CC license (CC BY-NC-ND 4.0)
In *Proceedings of the 26th International Conference on Enterprise Information Systems (ICEIS 2024) - Volume 1*, pages 397-408
ISBN: 978-989-758-692-7; ISSN: 2184-4992
Proceedings Copyright © 2024 by SCITEPRESS – Science and Technology Publications, Lda.

composability. Where enterprise "business" has interest in higher revenues for which they need frequent deliveries of business changes. And in large complex organisations this cannot be a separate, two-world journey, as digital products, and services that those organisations deliver to their customers are highly dependent on and intertangled with the underlying IT (Orlikowski, 2010).

The objective of this paper is to reconcile these two worlds in large, complex organizations with Composable Enterprise mindset. Composable Enterprise is a new paradigm introduced by Gartner that enables flexibility of business change through composability and modularity that starts with business assets and is achieved as a joined business-IT venture (Panetta, 2020). This purpose of this paper is to propose a methodology for implementing Composable Enterprise in an evolutionary way through holistic business-IT delivery of business initiatives and to illustrate the implementation of the methodology using a real industry use case. The paper is the continuation of the announced research on strategy realisation through implementation of Composable Enterprise as part of the BASE framework (**B**usiness **A**rchitecture-based EA Framework for **S**trategic Alignment of the **E**nterprise) (Ivas, 2023). As authors works as enterprise architect in real industry, application of the proposed methodology for implementing Composable Enterprise will be carried out as part of her regular business-support EA work.

2 BACKGROUND AND RELATED WORK

This section will provide theoretical background on terms and definitions related to SOA, microservices and Composable Enterprise. Section also presents results of focused literature research on development method for implementing Composable Enterprise in an evolutionary way.

2.1 About SOA and Microservices

The roots of Service Oriented Architecture (SOA) go back to 90s (Krafzig, 2010) in time when "Software crisis" from the previous decades was considered to be over. "Software crisis" is a phenomenon that emerged in 60s and referred to poor software performance that was causing failed software projects (Dijkstra, 1972, Randell, 1996). New programming languages as Smalltalk, C++ and later Java were seen as remedy for these ills from the past, as they introduced structured programming, modularity, and Object Orientation (OO). OO paradigm introduced abstraction with focus on the "what" instead of the "how", with caller invoking the exposed interfaces to use the service without worrying about the service internal "how" details (Jana, 2006). This sorted out isolated application, but the focus than shifted to end-to-end process chains and application landscapes supporting them which introduced integration of application landscapes as the next big challenge of IT industry. As solution for this, SOA developed in three phases. In the first phase, the aim was to solve the problem of point-to-point integration of distributed applications with middleware technologies such as CORBA (Krafzig, 2010), which proved to be unreliable and complicated to use and caused unmanageable middleware complexity. The second phase therefore introduced Enterprise Application Integration (EAI) solutions (Linthicum, 2000). Although EAI was introduced to reduce complexity, it is now considered a disaster because it enriched the middleware with auxiliary technical services, as centralized data transformation or transaction monitoring, which further increased application landscape complexity. As a final solution, in the third phase, service-oriented architectures (SOA) were developed as a continuous improvement process at company level. SOA introduced a holistic view of enterprise IT architecture and the idea of enterprise consisting of composable services (provided as modules of enterprise monolith applications) that are accessed via standardised web interfaces and used as building blocks for business processes (Krafzig, 2010). Introducing service composability design principle, SOA envisioned composable services as reusable modular services consisting of other reusable modular services. SOA also introduced service orchestration with Enterprise service bus (ESB), a central service integration unit providing capabilities as integration, routing, orchestration, message transformation or service monitoring. The ESB approach led to centralization of business logic, making the ESB as a central enterprise monolith a bottleneck and a major pain point for many organizations (Shadija et al., 2017). The promised remedy came in the form of microservice architecture. Microservice architecture, initially introduced as a concept in 2011 (Dragoni et al., 2017), is a distributed application consisting of independently developed, deployed, and managed fine-grained services that communicate via lightweight protocols as REST (Shadija et al., 2017). Microservice architecture introduced service choreography, meaning that instead of central orchestration unit as in SOA, microservices themselves take responsibility for interacting with the environment, i.e., in a microservice architecture business processes are embedded into the services (Cerny, 2017). This limited flexibility in design, adjustment, and visibility of end-to-end business processes, as well as difficulty of managing microservice architectures, as a large unorganised set of fine-grained, headless services, made unrealistic the implicit expectation that microservice architectures would be sufficient to meet all the needs of large complex organizations (Xiao et al., 2016). Furthermore, following Domain driven design (DDD), microservices should be defined within a so-called bounded context, a boundary of a model within a specific domain. The purpose of this concept was to encourage a design that focuses on delivering business functionality rather than just focusing on code reuse and decomposition (Evans, 2003). However, very fuzzy definition of DDD´s bounded context and the separation of business problem space and solution space led to interpretion that DDD domains (or subdomains) belong to "business" and its problem space, while engineers define the bounded context as the solution space-boundery of the software they design (Tune, 2020). The next chapter will introduce Composable Enterprise as a new remedy for illnesses of large complex organisations, aiming to bring bunded context of IT solution space close to the business problem space and DDD domains (or business capabilities).

2.2 Composable Enterprise

The term Composable Enterprise was introduced by consultancy company Gartner in 2020, which defined it as "an organization that can innovate and adapt to changing business needs through the assembly and combination of packaged business capabilities" (Gaughan et al., 2020). Composable Enterprise is an enterprise envisioned as modular and highly adaptable to business changes. This is achieved through Composable Business, an enterprise business made up of a combination of easy interchangeable and pluggable "lego-like" components. As Composable Enterprise embraces the API economy, a business model built around delivering enterprise services to customers through APIs (Basole, 2019), pluggability in Composable Enterprise is enabled through APIs exposed by its composable components. The key principles of Composable Business include ensuring holistic system view at the business level (and not just at the technology level) and applying modularity to business assets to achieve flexibility for implementing business change (Dessus, 2021). Modularity, defined as the extent to which a system may be divided into smaller components, is an important concept for reducing enterprise complexity by breaking enterprise to independent parts, hiding the complexity of each part behind abstraction and API interfaces (Baldwin and Clark, 2000). Modularity in Composable Enterprise is achieved through enterprise business capabilities, which are referred to as Packaged Business Capabilities (PBC) in Composable Enterprise. Business Capability Map (BCM), as a set of all enterprise business capabilities (PBCs), is envisioned as a tool for achieving Business Composability, a business-IT aligned service abstraction that starts from the notion of first applying service composability to business assets (or PBCs), to achieve the scale and pace needed for implementing business changes. Where PBCs would serve as building blocks both for composing the enterprise and encapsulating software components (Panetta, 2020). Furthermore, Gartner defines three building blocks of Composable Business: composable thinking, composable business architecture and composable technologies. Composable thinking promotes the idea that everything can be composable by encouraging the creative use of design principles throughout the enterprise to enable more flexible adaptable way to meet ever changing customer needs. Composable business architecture, with use of business capabilities delivering composable business processes (Heinig, 2022), ensures that the organization and its business are built as flexible and resilient. Where composable technology enables flexible information systems to support business which is achieved with modular system architectures (Panetta, 2020). An IT architecture that emerged to support building composable technologies in Composable Enterprise is so-called MACH architecture (MACH Technology, n.d.). MACH is a set of design principles that encourages building information systems as a set of pluggable, scalable, and replaceable IT components. MACH architectures promote the following concepts:

- ***M**icroservices-based*: developing microservices as independently developed, deployed, and managed components
- ***A**PI-first*: hiding internal complexity, and exposing functionalities through APIs that enable pluggability of PBCs
- ***C**loud-native SaaS*: leveraging hosting, elastic scaling, and automatic updating of cloud deployment models
- ***H**eadless*: the core of composable concept are headless, easy interchangeable components. At

IT level, this means that developed components should be loosely coupled, reusable modules that are agnostic to programming languages or frameworks and preferably oihave decoupled front-end and back-end.

A PBC solution should provide at least ***API-first*** and ***H****eadless*, meaning that the underlying technical components that enable PBC should be loosely coupled reusable modules with internal complexity being hidden behind a generic public API. To found theoretical background on Composable Enterprise and development method for implementing Composable Enterprise, author searched the literature with selection criterion "is composable enterprise mentioned as a concept". Table 1 lists the results of focused literature review (Brocke et al. 2009) which was carried out by searching the term "composable enterprise" on Google Scholar, Web of Science and ResearchGate, along with filtering relevant references in the found articles. As listed in table 1, Composable Enterprise is still very novel in scientific research with only a few scientific articles found dealing with the topic (Sunyaev et al., 2023, Scheer, 2023, Ćorić and Mabić, 2023, van Schalkwyk and Isaacs, 2023, Wang and Gao, 2022). As the search on academic papers resulted in just a few results, the professional literature was also searched using the same search term "composable enterprise" on Google. This search returned 37.900 results proving that Composable Enterprise is a very important technology trend. These results showed the mismatch between the academic work on enterprise architecture and practical enterprise architecture, as identified by Gampfer et al. (2018).

Table 1: Overview of the most relevant literature on Composable Enterprise.

Title	Reference	Comment
Future of applications: delivering the composable enterprise. ID: G00465932	Gaughan et al. (2020)	Professional Gartner´s paper that elaborates on necessity of modernising application portfolio to become modular, flexible, and composable in order to support delivering business strategy at the pace of business change. No development method for implementing Composable Enterprise provided.
Gartner Keynote: The Future of Business is Composable	Panetta (2020)	Professional Gartner´s paper that elaborates on the concept of Composable Enterprise and Composable Business. No development method for implementing Composable Enterprise provided.
Method of Building Enterprise Business Capability Based on the Variable-Scale Data Analysis Theory	Wang and Gao (2022)	Scientific paper that mentions Composable Enterprise as a new architecture based on enterprise business capabilities. No development method for implementing Composable Enterprise provided.
Composable architecture: the latest trend in EA helping companies adapt and grow.	Bhatnagar (2022)	Professional paper that provides theoretical background on Composable Enterprise providing high-level method to engage in composable architecture.
Composable business processes – The journey towards a composable enterprise	Heinig (2022)	Professional paper that elaborates on composability of business processes assembled with business capabilities as building blocks. No development method for implementing Composable Enterprise provided.
Is composable enterprise the key to digital transformation?	Ćorić and Mabić (2023)	Conference article providing theoretical background on Composable Enterprise in the context of digital transformation. No development method for implementing Composable Enterprise provided.
The Composable Enterprise: Agile, Flexible, Innovative: A Gamechanger for Organisations	Scheer (2023)	A set of scientific articles on Composable Enterprise published as a book. Provides background information on the concept of Composable Enterprise with development method for implementing Composable Enterprise, but as an enterprise-wide rearchitecting approach.
The Future of Enterprise Information Systems.	Sunyaev et al. (2023)	Scientific paper that envisions enterprise processes made up of composable, interchangeable building blocks supported by corresponding enterprise information systems which enables "business to reassemble features dynamically and rearrange them as needed depending on external or internal factors". No development method for implementing Composable Enterprise provided.
Achieving Scale Through Composable and Lean Digital Twins.	van Schalkwyk and Isaacs (2023).	Scientific paper that mentions Composable Enterprise as an important topic for enterprises to adopt Digital Twins (concept of using the power of technology to make revolutionary improvements for the society). No development method for implementing Composable Enterprise provided.
The fundamentals of MACH	MACH Technology (n.d.).	Professional material that explains the MACH architecture as a set of design principles for implementing flexible, composable technology to support Composable Business. No development method for implementing Composable Enterprise provided.

The most comprehensive work found on the topic was the book "The Composable Enterprise: Agile, Flexible, Innovative: A Gamechanger for Organisations" (2023) by August-Wilhelm Scheer, in which the author proposes a Composable Enterprise development method for rearchitecting the whole enterprise. While this is a very valuable conceptual idea, this is unlikely to be implemented in large, complex organisations as such an enterprise-wide activity would require huge investment and would hinder regular business operations and delivery of new business changes. The purpose of this paper is to propose a different more cost-effective evolutionary approach with continuous implementation of Composable Enterprise as part of delivery of business initiatives.

Since most of the papers found only provide the theoretical background introduced by Gartner (Gaughan et al., 2020, Panetta, 2020), and none of them provide a development method for implementing Composable Enterprise in an evolutionary way, the author concludes that such a method has not been researched before

3 METHODLOGY

A prerequisite for implementation of the methodology explained below is the existence of some "good enough" version of BCM, a map of enterprise business capabilities (e.g., figure 2), and *Holistic value delivery* value stream (e.g., figure 4). *Holistic value delivery* is an enterprise value stream that illustrates all end-to-end critical steps that enterprise must undertake to deliver value to its customers, with the steps enabled by BCM business capabilities (Ivas, 2023). Methodology for implementing Composable Enterprise is as follows:

1. *Understand business drivers and objectives.* The first step is to understand the business background of the initiatives, i.e., the rationale, objective, and scope from the business point of view.
2. *Understand holistic scope of the initiative*. The second step is about understanding which value streams steps and business capabilities from the *Holistic value delivery* are being affected by the initiative and how.
3. *Understand current situation.* Understand and sketch current solution (architecture) in scope.
4. *Understand situation and needs at enterprise level.* Identify if there are any components that can be reused or optimised by this solution at enterprise level, or if there are some other future initiatives with the same need.
5. *Design as API -first Headless PBC (preferably according to MACH).* If there is a need to implement a new service, you should preferably design it according to MACH principles. Otherwise, deliver business change by designing new or optimising existing monolith modules by *API-first* and *Headless* MACH principles.
6. *Implement business-IT aligned PBC solution and consolidate.* Implement business-IT agreed solution and, consolidate into the new solution any old solutions which implements the same functionality (business capability).

Diagramming tool that will be used for creating research artifacts is Archi, an open-source EA tool supporting ArchiMate EA language (The Open Group, 2022). The following section explains the metamodel of the EA model used as part of the proposed methodology.

3.1 EA Model

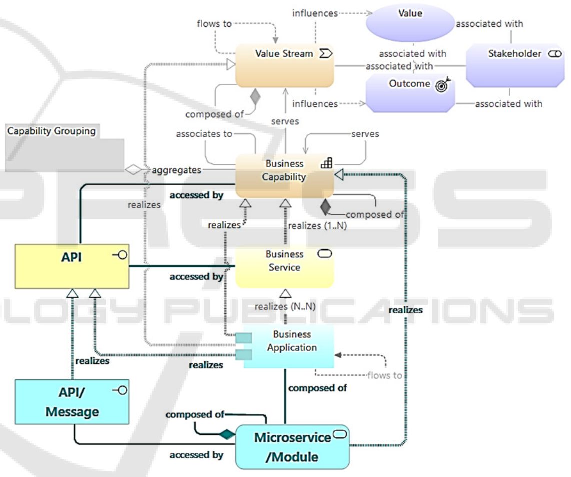


Figure 1: Metamodel of the EA model, newly added elements, and relationships in green.

Figure 1 shows extended metamodel of the EA model proposed in Ivas (2023). New elements added to the EA model to support the Composable Enterprise are *API*, *API/Message* and *Microservice/Module*. As listed in tables 2 and 3, to support this model extension, ArchiMate notation is used, as follows:

- element *API/message* added using ArchiMate´s *Application Interface* element
- element *Microservice/Module* added using ArchiMate´s *Application service* element
- element *API* added using ArchiMate´s *Business service* element
- *accessed by* relationship added between *Business capability* and *API* using ArchiMate´s *associate* relationship

- *accessed by* relationship added between *Microservice/Module* and *API/message* using ArchiMate´s *associate* relationship
- *composed of* relationship added between *Business Application* and *Microservice/Module* using ArchiMate´s *associate* relationship.

Table 2: Definitions of the newly added EA model elements.

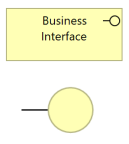
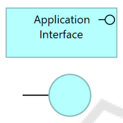
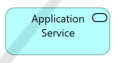
Element	Definition	ArchiMate Notation
API	API interface exposed by business capability to leverage API economy.	
API/message	Application interface in the form of API or message (message stream).	
Microservice/ Module	A functionality implemented as microservice or module of a monolith.	

Table 3: EA model elements relationships.

Source	Relationship type	Target	Description
Business capability	accessed by	API	ArchiMate´s *associate* relationship.
Microser. /Module	realizes	Business capability	Business capability as bounded context of *microservice /module* enables implementation of Business Composability. Relationship supported by ArchiMate.
Business application	composed of	Microser. /Module	ArchiMate´s *associate* relationship used.
Microser. /Module	composed of	Microser. /Module	To implement service composability principle in IT. Relationship supported by ArchiMate.
Microser. /Module	accessed by	API/ Message	ArchiMate´s *associate* relationship used.
API/ Message	realizes	API	Relationship supported by ArchiMate.
Business application	realizes	API	Relationship supported by ArchiMate.

4 RESULTS

The company used to implement the proposed method acts as an acquirer, a financial institution licensed to process credit and debit cards on behalf of merchants. In this company author works as enterprise architect and part of her regular work is supporting initiative gate approval process with early scoping of business initiatives. Early initiative scoping serves as an input for investment determination and initiative funding. One of those initiatives was "*generic surcharge*". The high-level scoping and design of the solution were carried out in collaboration between the author as EA, an author´s EA colleague (both belonging to IT part of the company), the surcharge product owner and the delivery manager (representing company´s "business"). For step 5, IT solution architects were consulted to help estimate and design the most cost-effective solution.

The results of applying proposed methodology are presented below. All data in presented diagrams are anonymised.

Step 1: Understand Business Drivers and Objectives.

Over the last 30 years payment processing industry has experienced a radical transformation that resulted with so called "Race to Zero". "Race to Zero" refers to the commoditization of payment processing services, where processing fees have fallen to almost zero due to increasingly competitive marketplace (Fitzgerald, 2023). Payment processors (acquirers) must therefore find other ways to generate revenue, for example by offering so-called value-added services. One of the value-added services offered by researched company is the surcharge, a service offered by acquirers to merchants who would like to additionally charge cardholders for the Merchant Service Charge (MSC) fee they pay to the acquirer for acquiring service. Surcharge as a feature is prohibited by EU legislation, meaning that surcharge can be used only for transactions made by Business Cards issued in the EU and all cards issued outside of the EU. Surcharge feature is available/allowed on the following card schemes: Mastercard, Visa, JCB, CUP, DI/DC, and Amex. Surcharge is identified during the purchase, after the card is read and prior to the authorisation with surcharge eligible check being based on the BIN-range of the card. The scope of initiative is Finland, Norway and Denmark, Sweden is out of scope due to local regulations. The company´s rationale for building centralised generic surcharge service is as follows:

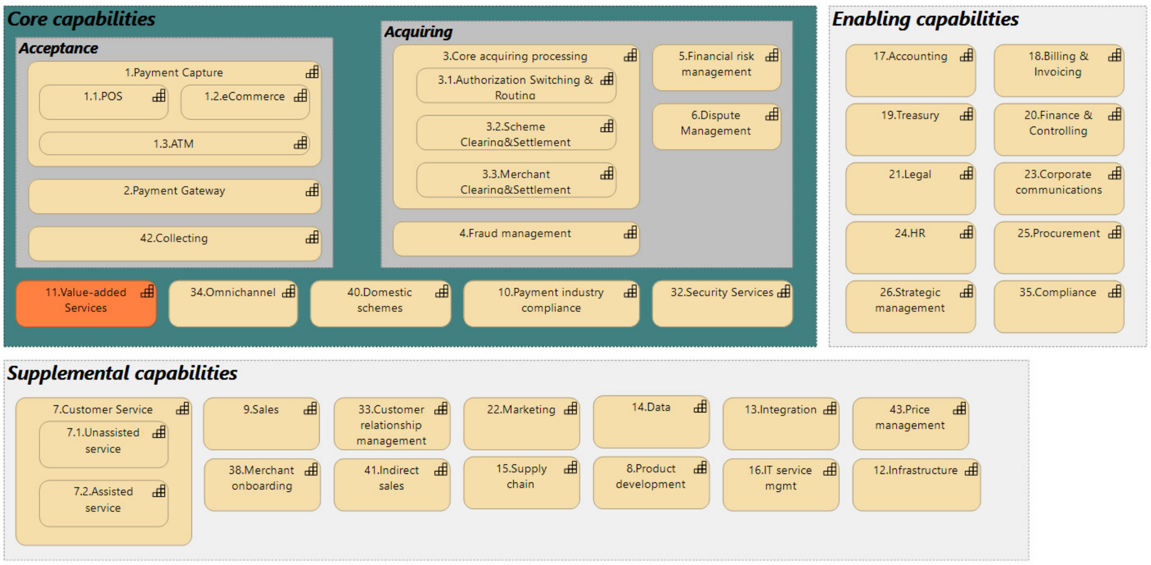


Figure 2: Acquiring high-level business capability map (surcharge is part of *11.Value added services*).

- The Competition and Consumer Authority has made it clear that the current company´s surcharge setup for external PSPs is not compliant and have strongly urged company to replace it, with very high risk that they will soon mandate it out of service. The generic surcharge service will modernize company´s setup and eliminate the manual processes in the current setup (which is the reason why current setup is not compliant).
- By doing this build, company protects the revenue from its current external PSPs, enables easy onboarding of additional external PSPs, and makes it significantly easier for internal PSPs to enable surcharge.
- Non-financial benefits are:
  - License to play in hospitality: surcharge is seen as a prerequisite for having a viable proposition for hospitality.
  - Brand value: surcharge is an industry standard feature – not having it may hurt company´s brand.
- Financial benefits (quantifiable and non-quantifiable):
  - Revenue Protection: external PSPs are live on current setup. Approving this build will protect that revenue.
  - Incremental Revenue: additional PSPs can be added, both internal and external.
  - Running Cost and Error Efficiency: with current setup 3 days of manual work is required monthly, which is also an error-prone process.
  - Price Erosion Protection: surcharge is an important negotiation tool to maintain or increase company´s margins.
  - Cheaper enablement of surcharge on internal PSPs: company´s internal PSPs will have a more efficient way for surcharge enablement.

The information for this step is provided by P. Sand, surcharge product owner, and M. W. Lund, delivery manager.

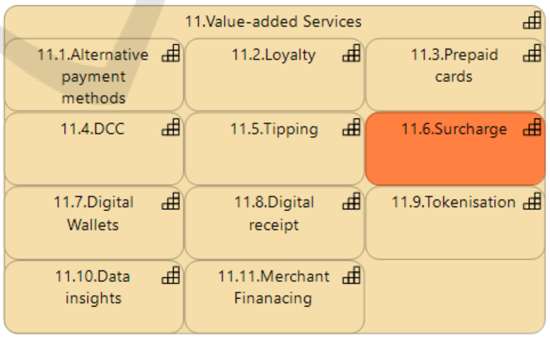


Figure 3: Decomposition of *11.Value-added services* (from the full business capability map).

Step 2: Understand Holistic Scope of the Initiative.

In business capability map, surcharge capability (*11.6.Surcharge*) belongs to *11.Value added services* (figures 2 and 3). As illustrated in company's *Holistic value delivery* (figure 4), *11.6.Surcharge* business

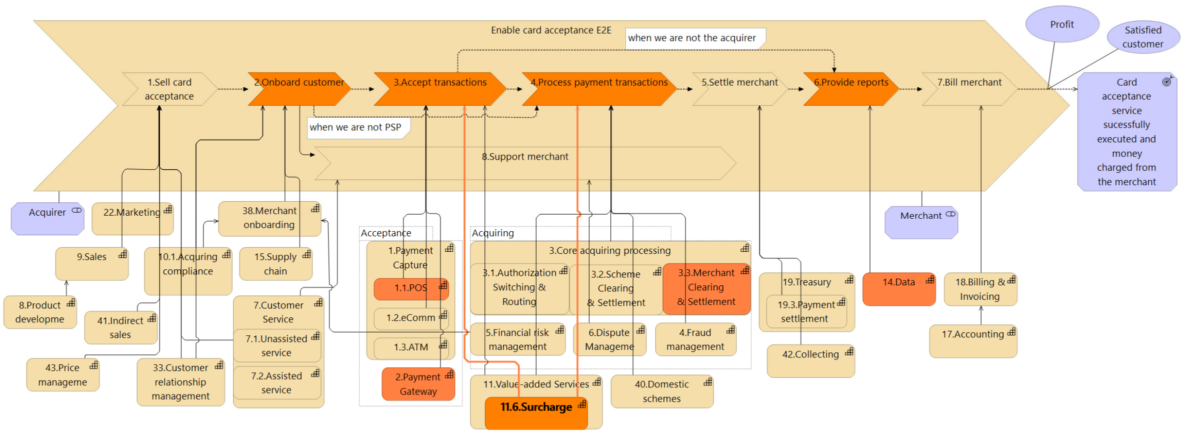


Figure 4: In orange affected by holistic surcharge product end-to-end delivery.

capability directly supports value stream steps *3.Accept transactions* and *4.Process payment transactions*. However, holistically scoping surcharge product delivery means taking into consideration all dependencies of the whole end-to-end value delivery process. Which means that the scope of the initiative are also the following value stream steps:

- *2.Onboard Customer*

PSP or payment gateway (business capability *2.Payment gateway*) is a system that routes payment transactions received by payment terminal to core acquiring systems. Internal and external PSPs will have to integrate towards the new generic surcharge API, first by getting registered on the API Marketplace (self-service portal) and fetching the API keys and then by adjusting their code to utilise the new API. Existing process of merchant onboarding is not affected.

- *3.Accept Transactions*

The operational call for surcharge lookup should be redirected to the new surcharge service, instead of existing implementations in internal and external PSPs. PSPs in scope are "PSP 1", "SoftPOS 3rd party PSP" and "3rd party PSPs" (*2.Payment gateway* business capability) that will fetch surcharge lookup for "POS payment terminal 1", "SoftPOS mobile app" and "3rd party POS payments terminals" (*1.1.POS* business capability). POS (point of sales) systems are used to support transactions in physical stores.

- *4.Process payment transactions*

3.3.Merchant clearing & settlement system must feed the new surcharge solution with additional parameter mapping to support generic service. The system that supports this and which falls within the direct scope of the initiative is the "Core Acquiring System 1".

- *6.Provide reports*

New "Generic surcharge service" should provide additional lookup statistics and error metrics.

Step 3: Understand Current Situation.

Figure 5 illustrates the current solution with several functional redundancies of surcharge calculation capability as follows:

1. "Surcharge calculation" implemented in "PSP 1" available only for "POS payment terminal 1" (physical POS terminal).
2. "Surcharge calculation" implemented in "SoftPOS 3rd party PSP" available only for "SoftPOS mobile app" (mobile phone terminal application)
3. "Surcharge calculation" implemented in "3rd party PSPs" available only for "3rd party POS payment terminals".

All solutions are only available within their ecosystems, which means that decision to make the surcharge product available on some other type of internal or external POS terminal (to support physical stores) or eComm (to support online web shop transactions) requires their PSPs to develop their own surcharge solution. In addition, the current setup requires changes in both internal PSP, 3rd party PSPs and internal core acquiring system if the data exchange format changes. Furthermore, the second solution with "SoftPOS 3rd party PSP" is the recent implementation of functionality with the new "REST API" interface being built to serve only "SoftPOS 3rd party PSP". This means that there are now two interfaces for the same surcharge parameters data flow that need to be maintained. "SoftPOS" solution exemplifies traditional delivery where business initiatives tend to solve their own immediate needs without considering how the new solution fits to the overall enterprise architecture. Furthermore, company´s internal system "PSP 1" is a legacy monolith in which "surcharge calculation" is

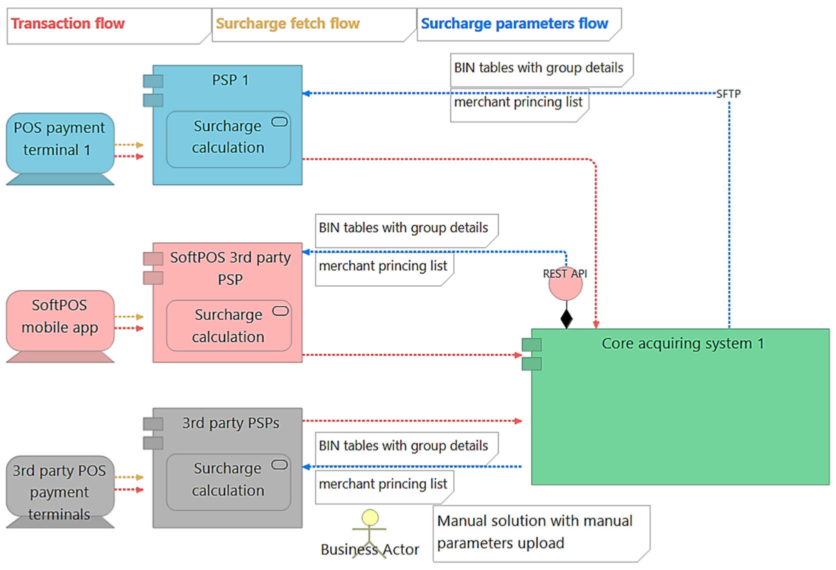


Figure 5: Current situation illustrating functional redundancies of Surcharge functionality.

implemented as a module. However, there are "surcharge calculation" services within the 3rd party PSPs (whose implementation is a black box for the company). Although these 3rd party solutions might be implemented as reusable and composable microservices inside their own 3rd party ecosystems, it does not change the fact that surcharge solution at enterprise level is inconsistent and redundant. This confirms Aleatrati Khosroshahi (2016) who states that without holistic high-level as-is overview, organisations tend to introduce new services to fulfil immediate business needs which leads to increased complexity (Aleatrati Khosroshahi, 2016).

Step 4: Understand Situation and Needs at Enterprise Level.

There is a need from a company´s "eComm PSP" to use this solution. It is also discovered that there is another POS terminal product that would like to have this solution available both for "Acquiring core platform 1" and "Acquiring core platform 2", which means that generic solution must be designed in such a way that any core acquiring platform can easily plug in and feed the new surcharge solution with needed parameters.

Step 5: Design as API-first Headless PBC Solution (Preferably According to MACH).

As illustrated in figure 6, acquiring and PSP agnostic target solution is designed according to following MACH design principles:

- ***M**icroservices-based*: the solution is envisioned as an independent, self-contained microservice. Figure 6 illustrates "Generic surcharge microservice" which would provide needed surcharge calculation functionality exposed for internal and external PSPs to consume. Service would calculate surcharge based on parameters received from core acquiring systems. Also, as figure 6 illustrates, following proposed metamodel (figure 1) "Generic surcharge microservice" is part of a *business application* "Value added services". According to proposed metamodel, microservices should always be grouped in a *business application* to enable their governance from the enterprise level (as microservices alone are too small and there are too many of them to be govern separately). In this case, newly introduced *business application* "Value added services" would also be used as a placeholder for the grouping of all future value-added service microservices.
- *API-first*: generic surcharge functionality would be exposed through generic "REST API" API application interface that would realise business "API" interface (figure 6). "REST API" would be designed with generic input and output parameters, without any internal identifiers exposed to the outside world. As showed in the figure 6, what "business" sees is the "API" interface of surcharge capability (*11.6.Surcharge*), which is publicly available and documented on company´s *API marketplace*, for PSPs (internal or external) to configure and use.

Furthermore, inbound "Surcharge parameters API" is also designed as generic, meaning that any core acquiring system can easily use it to share surcharge parameters without worrying about internal complexity of the "Generic surcharge microservice". This makes solution both PSP and acquiring agnostic.

- ***Cloud-native SaaS***: it is recommended to deploy the solution to one of the company's cloud providers to leverage cloud deployment model, providing storage, hosting, elastic scaling, and automatic updating.
- ***Headless***: Front-end presentation exposed via API interface with generic contract makes internal back-end logic black box for its internal and external PSPs and core acquiring systems.

As mentioned above, this initiative is part of the early gate approval process, which means that earlier steps were used to provide inputs for the cost estimation of the initiative. An external team with experience with microservice and cloud technologies, but without previous experience with company´s systems in scope, was asked to provide the cost estimate and timeframe for the implementation of the planned MACH solution (figure 6). This team's estimate was rejected as the estimated implementation and maintenance costs were too high and the timeframe offered was unacceptable (as important pre-sales enabler for hospitality, initiative was promised to be delivered as soon as possible).

This is why a joined business-IT trade-off decision was made to go instead for a more cost-effective solution with a shorter time-to-market by adjusting the existing surcharge solution in "PSP 1" developed by an internal team (figure 5). Also, the internal "PSP 1" team already had experience with the company's API marketplace. Figure 7 shows chosen target solution, which is, like the MACH solution (figure 6), both acquiring and PSP agnostic. Although this is not a long-term solution, as "PSP 1" as a legacy platform is likely to be replaced in the future, none of the internal and external PSPs or core acquiring platforms should be affected by a possible migration, as they only see the front-end presentation exposed via generic APIs. This chosen trade-off solution confirms Ford et al. (2022) who state that cost-effective time-to-market composable solutions could also be achieved using modular monoliths.

Step 6: Implement Business-IT Aligned PBC Solution and Consolidate.

After implementing the new trade-off solution (figure 7), plan is to first migrate "3rd party PSPs" and then also "SoftPOS 3rd party PSP".

5 DISCUSSIONS

As exemplified in the previous chapter, business initiatives often lead to inconsistent and redundant software systems when executed in a traditional way with IT serving to fulfil immediate business needs and when IT designs business-detached solutions

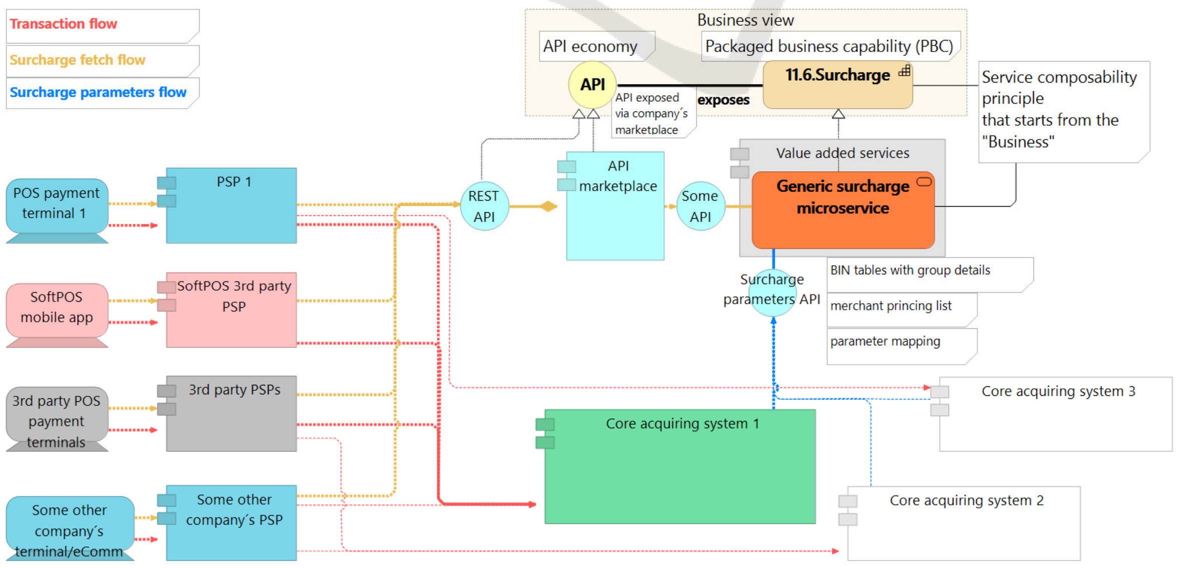


Figure 6: Target composable solution designed according to MACH principles.

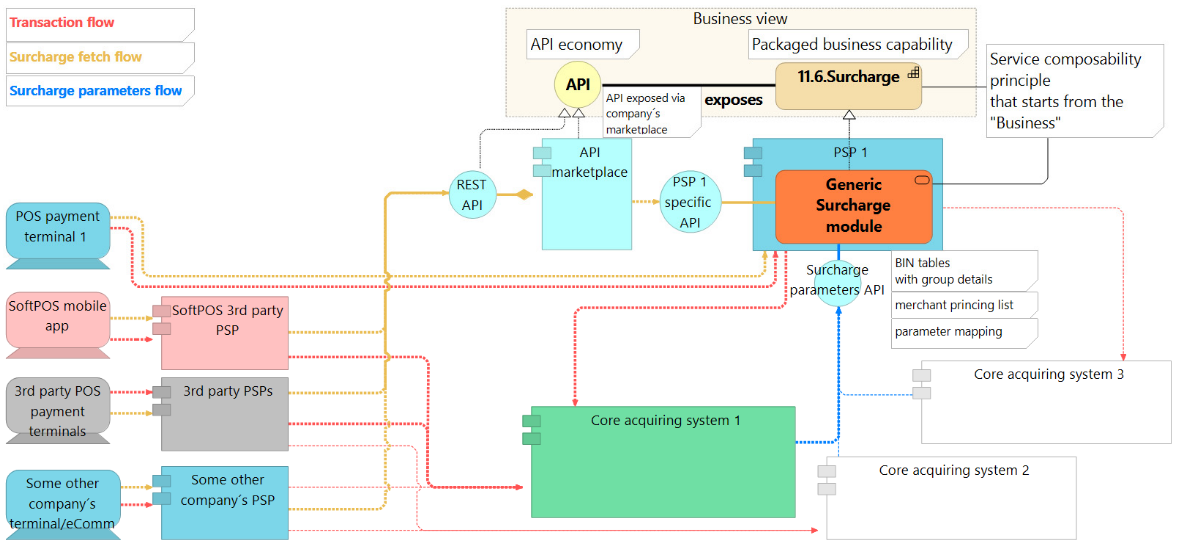


Figure 7: Trade-off composable solution implemented as a module of a monolith.

considering only domain of a specific ecosystem, without a holistic understanding of how solution fits into the overall delivery and enterprise architecture. Which increases the enterprise complexity and leads to higher maintenance costs, higher costs for future investments and less flexibility in implementing future changes. As solution, Composable Enterprise brings together "business" and IT introducing the mindset of composability at the enterprise level as a joint business-IT venture to enable ease of business changes with lower costs. Composable Enterprise is based on a holistic understanding of the enterprise and Business Composability, a service abstraction that starts from the notion of first applying service composability to business assets (business capabilities) to achieve the scale and pace needed for implementing business changes. Instead of massive enterprise-wide rationalization and consolidation initiative, this paper proposes implementation of Composable Enterprise in an evolutionary way through joined and holistic business-IT delivery of business initiatives. To support this thinking, paper has proposed methodology that has been implemented in a real industry where author works as an enterprise architect supporting "business" with early holistic scoping of business initiatives. The implementation of the proposed methodology has shown that "perfectly designed" IT solutions are not always the most suitable options when considered from all socio-techno-economic perspectives, illustrating how the same business-IT aligned objectives can also be achieved with modular monoliths. Future work on Composable Enterprise will incorporate value streams to explore how enterprise composability can be considered as part of business process design.

REFERENCES

Aleatrati Khosroshahi, P., Beese, J., Matthes, F., & Winter, R. (2016). Causes and Consequences of Application Portfolio Complexity–An Exploratory Study. In *The Practice of Enterprise Modeling: 9th IFIP WG 8.1. Working Conference, PoEM 2016, Skövde, Sweden, November 8-10, 2016, Proceedings 9* (pp. 11-25). Springer International Publishing.

Baldwin, C. Y., & Clark, K. B. (2000). *Design rules: The power of modularity* (Vol. 1). MIT press.

Basole, R. C. (2019). On the evolution of service ecosystems: A study of the emerging API economy. *Handbook of Service Science, Volume II*, 479-495.

Bhatnagar, R. (2021). Composable architecture: The latest trend in EA helping companies adapt and grow. *Architecture & Governance magazine*

Brocke, J. V., Simons, A., Niehaves, B., Niehaves, B., Reimer, K., Plattfaut, R., & Cleven, A. (2009). Reconstructing the giant: On the importance of rigour in documenting the literature search process.

Cerny, T., Donahoo, M. J., & Pechanec, J. (2017, September). Disambiguation and comparison of soa, microservices and self-contained systems. In *Proceedings of the International Conference on research in adaptive and convergent systems* (pp. 228-235).

Ćorić, I., & Mabić, C. M. (2023, August). Is composable enterprise the key to digital transformation? 6th international scientific conference on digital economy DIEC, Tuzla, BiH

Dessus, C., 2021, Business Composability, https://www.slideshare.net/chdessus/business-composability (Accessed on 15/02/2024).

Dijkstra, E. W. (1972). The humble programmer. *Communications of the ACM*, *15*(10), 859-866.

Dragoni, N., Giallorenzo, S., Lafuente, A. L., Mazzara, M., Montesi, F., Mustafin, R., & Safina, L. (2017). Microservices: yesterday, today, and tomorrow. *Present and ulterior software engineering*, 195-216.

Evans, E. J. (2003). Domain-driven design: tackling complexity in the heart of software. Addison-Wesley Professional.

Fitzgerald, N. (2023). *Payment Processing Race to Zero.* https://www.linkedin.com/pulse/payment-processing-race-zero-noah-fitzgerald/ (Accessed on 13/02/2024).

Ford, N., Parsons, R., Kua, P., & Sadalage, P. (2022). *Building evolutionary architectures.* " O'Reilly Media, Inc.".

Gampfer, F., Jürgens, A., Müller, M., & Buchkremer, R. (2018). Past, current and future trends in enterprise architecture—A view beyond the horizon. *Computers in Industry*, *100*, 70-84.

Gaughan, D., Natis, Y., Alvarez, G., & O'Neill, M. (2020). *Future of applications: delivering the composable enterprise. ID: G00465932*

Heinig, M. (2022) Composable business processes – The journey towards a composable enterprise. https://www.linkedin.com/ pulse/composable-business-processes-future-enterprises-modu lar-heinig (Accessed on 18/02/2024).

Ivas, I. (2023). Introduction to BASE Enterprise Architecture Framework for Holistic Strategic Alignment of the Complex Enterprise. In *ICEIS (2)* (pp. 575-588).

Jana, D. (2006). Service oriented architectures–a new paradigm. *CSI Communications*, 12-14.

Krafzig, D. (2010). Serviceorientierte Architekturen (SOA). *Informationsverarbeitung in Versicherungsunte-rnehmen*, 163-174.

Linthicum, D. S. (2000). *Enterprise application integration*. Addison-Wesley Professional.

MACH Technology, https://machalliance.org/mach-technology (Accessed on 2024/02/15)

Panetta, K. (2020). Gartner Keynote: The Future of Business is Composable. https://www.gartner.com/ smarterwithgartner/gartner-keynote-the-future-of-busin ess-is-composable (Accessed on 12/02/2024).

Randell, B. (1996). The 1968/69 NATO software engineering reports. *History of software engineering*, *37*.

Orlikowski, W. J. (2010). The sociomateriality of organisational life: considering technology in management research. *Cambridge journal of economics*, *34*(1), 125-141.

Scheer, A. W. (2023). *The Composable Enterprise: Agile, Flexible, Innovative: A Gamechanger for Organisations, Digitisation and Business Software*. Springer Nature.

Shadija, D., Rezai, M., & Hill, R. (2017, September). Towards an understanding of microservices. In *2017 23rd International Conference on Automation and Computing (ICAC)* (pp. 1-6). IEEE.

Sunyaev, A., Dehling, T., Strahringer, S., Da Xu, L., Heinig, M., Perscheid, M., ... & Rossi, M. (2023). The Future of Enterprise Information Systems. *Business & Information Systems Engineering*, *65*(6), 731-751.

The Open Group. (2022). *ArchiMate® 3.2 Specification.* https://pubs.opengroup.org/architecture/archimate32-doc/

Tune, N. (2020). "Domain, Subdomain, Bounded Context, Problem/Solution Space in DDD: Clearly Defined" https://medium.com/nick-tune-tech-strategy-blog/domains-subdomain-problem-solution-space-in-ddd-clearly-defined-e0b49c7b586c#:~:text=In%20DDD%2C%20a%20subdomain%20is,most%20domains%20are%20a%20subdomain (Accessed on 18/02/2024).

van Schalkwyk, P., & Isaacs, D. (2023). Achieving Scale Through Composable and Lean Digital Twins. In *The Digital Twin* (pp. 153-180). Cham: Springer International Publishing.

Wang, A., & Gao, X. (2022, July). Method of Building Enterprise Business Capability Based on the Variable-Scale Data Analysis Theory. In *International Conference on Logistics, Informatics and Service Sciences* (pp. 267-278). Singapore: Springer Nature Singapore.

Xiao, Z., Wijegunaratne, I., & Qiang, X. (2016, November). Reflections on SOA and Microservices. In *2016 4th International Conference on Enterprise Systems (ES)* (pp. 60-67). IEEE.