BULLETIN of Cherkasy State Technological University

Journal homepage: https://bulletin-chstu.com.ua/en

Vol. 30 No. 2. 2025

UDC 004.4:658.5
DOI: 10.62660/bcstu/2.2025.33

The method of structured transformation of a business process activity diagram into a context map of domain-driven design

Serhii Moskovko*
Postgraduate Student
Vinnytsia National Technical University
21021, 95 Khmelnytske Shose Str., Vinnytsia, Ukraine
https://orcid.org/-0001-4376-0187

Roman Kvyetnyy
Doctor of Technical Sciences, Professor
Vinnytsia National Technical University
21021, 95 Khmelnytske Shose Str., Vinnytsia, Ukraine
https://orcid.org/0009-0001-4376-0187

Abstract. The relevance of the study lies in the need to implement unified methods for the structured formation of context maps, capable of reliably and accurately representing business processes during the design of domain-driven information systems. The aim of the work was to develop and substantiate a systematic method for transforming business process activity diagrams into context maps, which optimises the modelling process and enhances the accuracy of interaction representation within the domain. To achieve this goal, methods of formal analysis of activity diagrams, structured identification of bounded contexts, grouping of interrelated actions, and establishing types of interactions between them were applied. The main results of the study consisted of creating a phased method that allows for the identification of key elements of business processes, their integration into logically consistent contexts, and the definition of inter-contextual relationships without the use of complex notations. The proposed approach enables the extraction of bounded contexts based on the analysis of activity diagrams, the identification of dynamic dependencies between actions of different participants, and the formation of a coherent context map. The effectiveness of the method was demonstrated using the example of the restaurant business, where it clearly identified interactions between customer service processes, the kitchen, inventory management, and payment processing. This contributed to a transparent distribution of responsibilities, improved system complexity management, accelerated design, and enhanced consistency between technical implementation and business requirements. The proposed approach also improved mutual understanding between developers and domain experts, as it provides clear boundaries of responsibility and a more adaptive system architecture. Furthermore, the results confirmed the scalability of the method to other industries, indicating its universality and wide applicability. The practical value of the work lies in the possibility of scaling the method for application in various domains, making it a useful tool for business analysts and software architects when designing complex systems and improving the effectiveness of architectural decision-making

Keywords: information systems; business process analysis; unified modelling language; model transformation; bounded context definition

Article's History: Received: 26.12.2024; Revised: 03.05.2025; Accepted: 16.06.2025.

Suggested Citation:
Moskovko, S., & Kvyetnyy, R. (2025). The method of structured transformation of a business process activity diagram into a context map of domain-driven design. *Bulletin of Cherkasy State Technological University*, 30(2), 33-43. doi: 10.62660/bcstu/2.2025.33.

*Corresponding author

INTRODUCTION

This study examined the problem of constructing context maps in the process of domain-driven design (DDD) for complex software systems, which is an important direction in modern software development approaches. The DDD methodology, initiated by E. Evans (2003) and expanded upon by V. Vernon (2013), laid the foundation for modern domain modelling. Its conceptual evolution was demonstrated by the works of C. Zhong *et al.* (2024), S. Kapferer & O. Zimmermann (2022), and J. Sangabriel-Alarcón *et al.* (2024), where DDD was applied to microservices, strategic mapping, and automated context validation. However, one of the key challenges remains the creation of a context map capable of systematically and accurately representing interactions between different parts of the domain. The lack of a clear methodology for forming context maps complicates mutual understanding among project participants and can lead to a disconnect between business requirements and technical solutions. A proper description of interactions and demarcation of responsibilities within the domain will contribute to improving software quality and reducing risks associated with incorrect interpretation of business logic. A balanced approach to modelling complex systems is critically important for projects that scale dynamically and constantly evolve.

Some researchers, notably C.E. da Silva *et al.* (2022), proposed the BPM2DDD method, which algorithmically identifies domains in Business Process Model and Notation (BPMN) models and automatically generates bounded contexts. Analysis showed high formal accuracy, but it must be acknowledged that the complexity of BPM notation limits the participation of business experts without deep process training. In contrast, Unified Modelling Language (UML) activity diagrams remain an intuitive and accessible modelling tool, actively used for formalising business logic in the architecture of modern information systems. According to the research by M. Waseem *et al.* (2021), UML diagrams – particularly activity and use case diagrams – remain one of the most common modelling tools even in microservice-based projects. Furthermore, the study by S. Khoshnevis (2023) highlighted the advantages of UML diagrams as an input format for automated transformation of business logic into structural system components, closely related to bounded contexts in DDD.

Significant contributions to strategic contextual mapping were made by S. Kapferer & O. Zimmermann (2020): their work presented the Domain-Specific Language (DSL) tool Context Mapper, which formalises bounded contexts and their interaction types in the form of machine-processable models. The study demonstrated how the generation of interaction schemas significantly simplifies the evolution of microservice architecture and reduces the risk of semantic disconnects between teams. However, existing approaches predominantly ignore the possibility of using simplified models at the initial stage, accessible to business analysts and domain experts, which reduces the effectiveness of their involvement in design.

Between 2020 and 2024, a number of works were published, some of which directly addressed the issue of forming context maps. H. Vural & M. Koyuncu (2021) demonstrated that the correct definition of bounded contexts in a microservice architecture reduces the cognitive complexity of code and accelerates evolutionary changes. However, the authors emphasised the absence of clear rules for transitioning from business processes to contexts. In a broader overview, J. Sangabriel-Alarcón *et al.* (2024) highlighted that practitioners either fragment the domain excessively, leading to unnecessary dependencies, or leave contexts too large, hindering development. C. Zhong *et al.* (2024), based on a statistical cross-section of a large number of companies, showed that early formation of a context map reduces the number of changes in service contracts by 23% in the first year, but researchers note that typical teams lack methodologies that combine the clarity of UML with the strict rules of DDD. In this same context, the research by H. Vural & M. Koyuncu (2021) complemented the results of S. Kapferer & O. Zimmermann (2020), demonstrating that formalised DSL solutions are effective only if the initial models are sufficiently accurate, which is often lacking in the pre-design analysis phase.

The aim of the article was to develop a new method for transforming UML activity diagrams into a context map within DDD, which will simplify the initial modelling of complex software systems and enhance the effectiveness of collaboration between business analysts and developers. It was necessary to analyse existing approaches to building context maps within Domain-Driven Design, paying particular attention to methods based on BPMN, and to evaluate their limitations. The next step involved developing and substantiating a method for systematically transforming UML activity diagrams into a DDD context map, suitable for application in the initial stages of modelling. To verify the effectiveness of the proposed method, it was planned to use a typical domain from the restaurant business to identify key bounded contexts and the relationships between them. Following this, a comparative analysis of the developed method with other approaches was planned to identify its advantages and disadvantages regarding effectiveness and flexibility. Finally, prospects for the further development of the method were identified, including possibilities for automation, the use of CASE tools, and integration with modern technologies.

MATERIALS AND METHODS

The research was based on an approach that involves the systematic transformation of business process activity diagrams into a context map within the framework of DDD. The material for analysis consisted of UML activity diagrams, which were manually created based on the primary business requirements of the

restaurant domain; during modelling, completeness of scenarios, the presence of all key actors, and the display of main control and data flows were taken into account. This domain is characterised by the presence of various participants (clients, cooks, waiters, suppliers, payment systems), as well as extensive interaction between them, which allows for a comprehensive demonstration of the proposed method's effectiveness.

Preliminary identification of key domain events was conducted using the Event Storming method, proposed by S. Gadiyar (2024), which ensured rapid detection of cause-and-effect relationships in the process. The study used a five-stage methodology for transforming a UML activity diagram into a context map within the framework of DDD. A schematic representation of this methodology is provided in Figure 1.

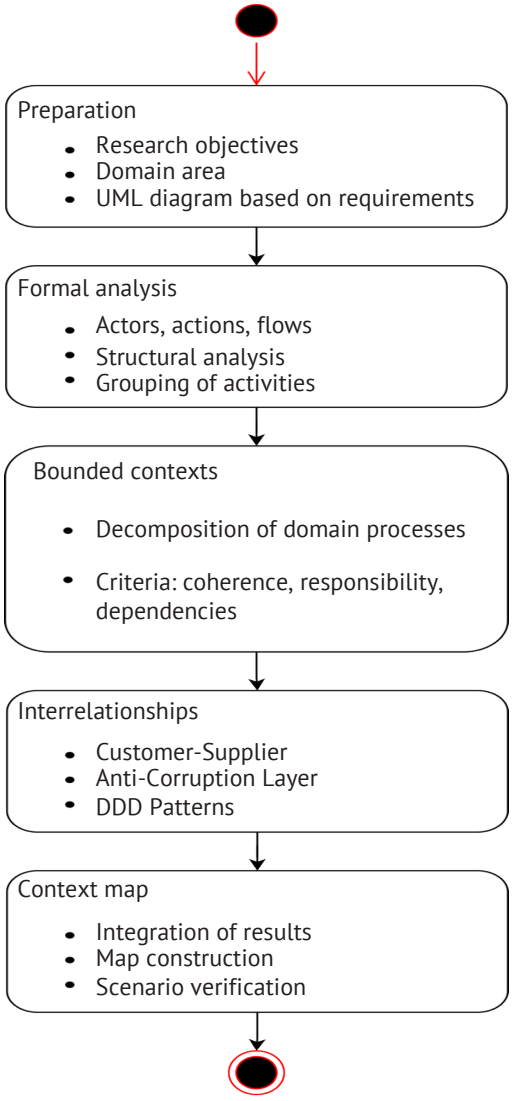

Figure 1. Diagram of the phased transformation of an activity diagram into a context map
Source: developed by the authors

A detailed description of the actions performed at each stage is provided below. During the preparatory stage, the research objectives were formulated, which involved developing a systematic approach to forming a context map for complex business systems, particularly within the restaurant service domain. The restaurant business was chosen as the domain area due to its clear structure, the presence of multiple actors with distinct areas of responsibility, and the typicality of its processes. The UML activity diagram used in the study was created based on an analysis of primary business requirements, formulated through an Event-Storming session (Gadiyar, 2024). When creating the diagram, consideration was given to the completeness of process coverage (customer service, food preparation, inventory management, payment processing), the participation of all key actors (client, waiter, cook, supplier, accountant, payment system), and the presence of control points for data transfer and management. Table 1 provides a generalised description of the main elements used in the study during the transformation of business logic into a context map.

Table 1. Key elements of UML activity diagrams

UML element	Description
Action	The basic unit of the diagram, representing a single step performed by a user or system.
Control Flow	Arrows indicating the sequence of action execution.
Initial/Final Node	Define the start and end of the process respectively.
Fork/Join Node	Used for parallelising or merging execution flows.
Decision Node	A branching point where the process divides depending on a condition.
Object Node	Displays objects (data) that are passed between actions.
Swimlane	Sections that divide actions among process participants, indicating who is responsible for each action.

Source: developed by the authors

At the formal analysis stage of the activity diagram, key actors and the corresponding activities they perform were identified. Grouping was based on the analysis of functional commonality, target outcomes, and sequences of actions. The key criterion here is the presence of a clear business objective that each group of actions realises. The methodological basis for this stage was the principle of transitive unification of actions into service layers, as outlined by O. Özkan *et al.* (2023). This forms the prerequisite for distinguishing bounded contexts – an autonomous part of the system with its own internal logic. At the stage of identifying bounded contexts, the processes of the restaurant business were decomposed based on the identified groups of activities. The criteria for defining contexts were the internal coherence of actions, clear boundaries of responsibility, and the minimisation of external dependencies. For the transitive unification of commands into service layers, the approach by O. Özkan *et al.* (2023) was used, which allowed related functions to be consolidated into a single logical whole.

At the stage of establishing inter-contextual relationships, the directions of interaction between the identified contexts were determined. A comparison of connection types with known interaction patterns (Alfadel *et al.*, 2020; Farshidi *et al.*, 2020) was applied. The determination of critical synchronisation points was carried out considering the queue-time/flow-time ratio (Albuquerque *et al.*, 2020), and patterns (Valdivia *et al.*, 2020) were used for formal typification. In the final stage, a context map was constructed, where each of the bounded contexts is represented as a separate component, and the interrelationships between them as labelled lines indicating the type of dependency. To verify the integrity of the model, the Specification by Example methodology was applied, which allowed for ensuring the correctness and consistency of the interaction logic between system elements (Irshad *et al.*, 2022).

RESULTS AND DISCUSSION

In the process of developing the method, the main components of activity diagrams were analysed, which are a key means of visualising business processes within UML (Ambler, 2005; Ozkaya & Erata, 2020). This type of diagram allows for modelling sequences of actions, parallel flows, branching conditions, and also for defining the responsibilities of process participants using swimlanes. An activity diagram is intuitive and has a lower barrier to entry compared to the more complex BPMN notation, making it convenient for use in designing business processes. DDD is an approach to developing software systems that focuses on a deep understanding of the domain and modelling business processes according to its real needs (Evans, 2003; Vernon, 2013). DDD involves close interaction between developers and domain experts to create a model that accurately reflects the entities and processes existing within the domain. The main tool is a Ubiquitous Language, which both parties use to describe and interact with the domain.

A Context Map is an important DDD tool that allows for visualising the boundaries and interactions between different parts of the system, or so-called bounded contexts. Each bounded context represents a separate, clearly defined business area within which specific terms, models, and rules have a fixed meaning and are interpreted independently of other parts of the system. This helps to avoid ambiguities and conflicts in the interpretation of concepts, especially in large projects with multiple teams. Within a single bounded context, its own model is formed, which should not be directly used outside of this context without special alignment mechanisms (e.g., via anti-corruption layers or published languages). As an example in the restaurant business, the "Customer Service" context might contain the term "order", which means a list of dishes chosen by a visitor and is accompanied by a waiter's actions. At the same time, in the "Kitchen" context, the same term "order" would mean an internal technical task for the cook, which includes preparation instructions. Despite having the same name, these models are independent and correspond to different business needs, thus they should exist within separate contexts. Bounded contexts in DDD are classified into three main types depending on their role in the system (Kapferer & Zimmermann, 2020).

The Core Domain is the most important part of the system, reflecting key business capabilities and strategic

aspects. This is a critical area for the product's success, and it is here that the largest investment of resources in modelling has the greatest value. Unique competitive advantages are formed in this domain, so it requires the deepest elaboration and continuous involvement of domain experts. The Supporting Domain encompasses auxiliary contexts that support the core functions but are not strategically important themselves. They perform less critical roles but are necessary for the effective functioning of the system. Typically, these are technical or administrative components that facilitate the operation of the core domain (e.g., logistics or accounting). The Generic Domain includes standard contexts that use typical, generally accepted models, not specific to a particular business. They can be reused in different projects without significant adaptation or deep modelling. Examples might include authorisation modules, user registration, or message processing.

Contexts do not exist in isolation; various types of interrelationships can arise between them. It is important to clearly define these interrelationships to ensure a coherent and flexible architecture. Incorrectly designed interactions between contexts can lead to excessive dependency, duplication of functionality, or difficulties with scaling. Clear definition of boundaries and integration mechanisms help to avoid terminology conflicts, preserve module autonomy, and ensure system stability during changes. Research by A. Hlybovets & I. Paprotskyi (2024) demonstrated that correctly implemented fault tolerance at the microservice level significantly reduces the risk of such failures. The main types of interactions are shown in Table 2.

Table 2. Types of interactions between bounded contexts

Type of interaction	Description	Features of interaction
Customer-Supplier	One context supplies data or services, the other consumes them.	The supplier is obliged to maintain the service; the consumer depends on the reliability and stability of the Application Programming Interface (API).
Conformist	The consumer adopts the supplier's model without changes.	Reduces integration complexity, but limits flexibility for future changes.
Partnership	Contexts closely collaborate to achieve a common goal.	Both sides are mutually dependent; high coordination of actions is required.
Open Host Service	A context publishes a service, accessible to several other contexts.	Provides shared access to functionality without tight integration.
Published Language	Interaction via a standardised language or protocol, agreed upon between contexts.	Reduces dependency, facilitates scaling and interaction with multiple systems.
Anti-Corruption Layer (ACL)	The downstream context is isolated from the upstream context using an adaptation layer.	Ensures translation between models; protects the internal model from external influences.

Source: developed by the authors

The main components of a context map in DDD (Vernon, 2013) play a crucial role in forming a clear system architecture and ensuring consistency between its parts. A context map serves as a visual tool demonstrating the structure of interactions between autonomous modules – bounded contexts – and defines their relationships, areas of responsibility, and shared components. The first and central element is the bounded context, which has already been described above. The importance of this component lies in its ability to avoid semantic conflicts and ensure the autonomy of teams working on different modules. Dividing into contexts helps prevent confusion and mixing of meanings. The next critical element of the map is the Relationships between bounded contexts. These relationships describe exactly how individual parts of the system interact, who is the supplier, and who is the consumer of data or services.

Types of interactions, such as Customer-Supplier, Conformist, Partnership, Open Host Service, or Published Language, define the level of dependency, coordination, and shared responsibility between contexts. Clear modelling of such interrelationships helps avoid excessive integration dependencies and ensures system flexibility when changes occur in specific modules. The third important component is the Shared Kernel – a set of models, objects, or functions that are used by several bounded contexts simultaneously. Its main purpose is to ensure consistency and reduce duplication of business logic in cases where complete isolation of contexts is impractical or too costly. A shared kernel requires careful management, as changes within it can affect all related contexts. However, in cases where contexts have closely related logic, a shared kernel becomes an effective mechanism for code reuse and maintaining term consistency.

A context map helps divide complex systems into independent areas, which reduces the complexity of interactions between different components and provides clear boundaries of responsibility. This is important for constantly evolving systems, as it allows changes within one context to be easily managed without disrupting the operation of others. Creating a context map within

DDD involves the sequential execution of a series of analytical stages, each of which plays an important role in forming a correct architectural model.

At the initial stage, bounded contexts were identified and grouped into corresponding subdomains. This involves detailed identification of the actors represented in the UML activity diagram, as well as the activities these actors perform. Identification was based on an analysis of swimlanes, which allow for the visual distribution of responsibilities between users and system components. Particular attention was paid to establishing the sequence of actions that form logical process blocks. In the next step, an analysis of the goals and expected outcomes of each process participant was carried out. This approach allows for understanding the motivation of each actor, as well as determining which results are critical from a business value perspective. This analysis serves as the basis for identifying bottlenecks and priority elements of the system. It is in this context that the identification of critical process points and potentials for its optimisation becomes possible, contributing to the qualitative separation of logic by domain boundaries.

Next, an investigation of interactions between actors was conducted. The primary focus was on identifying the roles of supplier and consumer in each specific segment of the process, which allows for the classification of dependencies between participants. Data flows and their directions are analysed to determine the directions of integration and coordination between contexts. This is critical for understanding precisely how information is transferred between system parts, as well as for identifying areas of potential duplication or conflicts. Following this, semantic grouping of activities within logical business functions was carried out. Based on the resulting groups of functionally related activities, bounded contexts are identified. It is important not only to ensure the internal coherence of each context but also to minimise its dependencies on other parts of the system. This provides greater autonomy, simplifies testing, accelerates independent deployment, and reduces the risks of cascading changes. At this stage, bounded contexts were also integrated into broader subdomains. Such consolidation is based on the relatedness of functions and conformity to the overall business structure, and also allows for classifying contexts as core, supporting, or generic – in accordance with DDD principles.

Further analysis involved identifying internal relationships between bounded contexts. For this purpose, data exchange paths between previously defined groups of actions were analysed, their functional interdependence and frequency of interactions were evaluated. Based on this information, specific communication channels were identified, their roles (supplier/consumer) were formalised, and the direction of information flows is established. The concepts of upstream and downstream contexts are applied to model who initiates the interaction and who receives it, which is crucial for maintaining system stability in changing conditions. The final analytical step is to assess the nature of the interrelationships between contexts. Here, it is appropriate to apply the DDD typology, which includes models such as Customer-Supplier, Conformist, Partnership, Anti-Corruption Layer, Open Host Service, and Published Language. Each type of interaction implies a different level of integration, isolation, and coordination, and the selection of the appropriate model depends on the requirements for flexibility, independence, and scalability. The analysis of these interrelationships allows for the justified formation of a context map, which not only visualises the system's architecture but also serves as a means of strategic management for its development.

Based on the conducted analysis, a detailed list of bounded contexts, subdomains, groups of related actions was obtained, and the interrelationships between them were established. As a result of the preceding stages, context map within DDD was created. This map serves as the basis for visualising and understanding the interaction between different parts of the system. Furthermore, an example of applying the method for transforming an activity diagram into a domain-driven context map using the restaurant business as a case study was provided. The activity diagram for the restaurant business is presented in Figure 2.

The first step in building a contextual map of the restaurant business was to identify key actors and related activities, which were recorded in the diagram (Fig. 2). Analysis of the participants in the process made it possible to determine which roles (both human and technical) are involved in the business process and what functional actions they perform. In particular, roles such as customer, waiter, cook, supplier, payment processing system, and accountant were covered. For each of them, a typical set of activities was identified that represent their functions within the service process: from the customer's arrival to the confirmation of payment by the accountant.

Further analysis was aimed at uncovering the motivations and expectations of each actor. At this stage, the goals of each role were defined – for example, client satisfaction, ensuring the quality of prepared dishes, accurate financial reporting, etc. – as well as the expected outcomes of participation in the business process. This allowed for identifying the strategic significance of individual functions and understanding which aspects of the process hold the highest business value and influence the overall user experience. Particular attention was paid to analysing the interactions between actors. It was established that supplier-consumer relationships exist between the roles, defining functional dependencies within the system. For instance, the client consumes services provided by the waiter, and the latter, in turn, relies on the results of the cook's work. Such chain dependencies allow for clear outlining of information flows and responsibilities between parts of the system. Data flow analysis enabled the identification of control directions and the

boundaries within which services or requests are transferred. The next logical step was the conceptual grouping of activities within higher logical units – business functions. Activities that were related by purpose or outcome were combined into four main business functions: customer service, food preparation, inventory management, and payment processing. This grouping was critical from the perspective of forming bounded contexts, as it allowed for the identification of autonomous functional areas with a high degree of internal coherence.

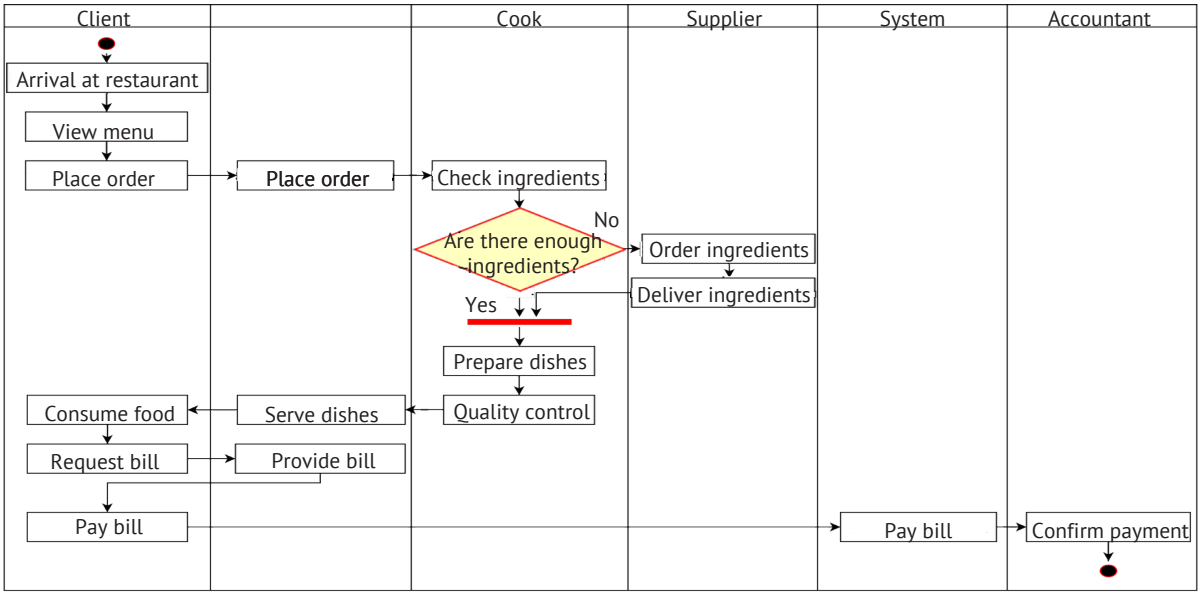

Figure 2. Activity diagram of the restaurant business

Source: developed by the authors

Based on the formed functional groups, the bounded contexts were defined: "Customer Service", "Kitchen", "Inventory Management", and "Payment". Each of these demonstrates a clearly expressed autonomy, which allows for minimising coupling between contexts and reducing potential risks during changes. Further analysis allowed these contexts to be classified by subdomain types according to DDD approaches. The "Customer Service" and "Kitchen" contexts were assigned to the core subdomains, as they represent key business value. In contrast, "Inventory Management" and "Payment" were classified as supporting subdomains, because, although important for system functioning, they do not carry strategic uniqueness.

In the next stage, inter-contextual relationships were modelled based on previously identified relationships between actors. The "Customer Service" context interacts with the "Kitchen" context using a Customer-Supplier model, where "Kitchen" supplies prepared dishes, and "Customer Service" consumes them. A similar dependency is observed between "Kitchen" and "Inventory Management", although an additional mechanism – an anti-corruption layer – was applied here, which allowed for isolating the model of one context from changes in another. Interaction between "Customer Service" and "Payment" was also modelled as Customer-Supplier with elements of an ACL, as the payment processing process is critical but should not affect the internal logic of the main service. The final stage was the construction of the context map, which visualises the derived bounded contexts, their interrelationships, integration types, and directions of dependencies (Fig. 3).

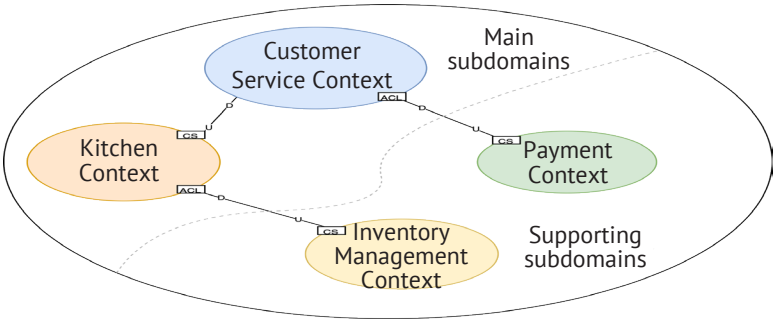

Figure 3. Context map of the restaurant business

Source: developed by the authors

It is important to involve domain experts to verify and refine the created context map. Their experience and knowledge of business specifics will help identify possible inaccuracies or gaps in the model, and ensure that all nuances of business processes are taken into account. This will contribute to the context map accurately reflecting the real needs and structure of the business. It should be noted that the context map may change during further analysis and development. In-depth study of the domain, changes in business requirements, or newly identified details may require a review of context boundaries and their interrelationships. The proposed method for transforming activity diagrams into a DDD context map is convenient for modelling business systems. The use of UML activity diagrams simplifies the process, as UML is a widely known and understood tool, facilitating collaboration between business analysts and developers.

Unlike the BPM2DDD method, which is based on detailed BPMN models and can be more complex for organisations without established BPMN usage practices, the approach proposed in this study is more flexible and accessible (Da Silva *et al.*, 2022). It allows for precisely defining bounded contexts and accounting for the nuances of business processes without excessive formalisation. Furthermore, this method emphasises the importance of involving domain experts and readiness to adapt the context map during the analysis process, which increases the accuracy and relevance of the model. Compared to BPM2DDD, the proposed approach ensures faster and simpler transformation of business processes into technical models, making it an attractive choice for many organisations.

The obtained results regarding the systematic transformation of UML activity diagrams into a context map for DDD should be considered in comparison with the research of other authors who also studied the problem of forming bounded context boundaries and integrating business processes into DDD. A comparative analysis made it possible to determine the extent to which the chosen approach is consistent with previous works and what the main differences are. First and foremost, the findings should be compared with the results of the study by C.E. Da Silva *et al.* (2022), which proposed a formalised transformation of BPMN models into domains within the BPM2DDD method. The author noted high accuracy in domain identification but emphasised that the complexity of BPMN complicates the involvement of specialists without in-depth expertise in process notation. In the presented approach, a similar goal – the formation of bounded contexts – was achieved, but by using more accessible UML activity diagrams. Compared to the BPM2DDD method, the presented solution benefits from a lower barrier to entry for teams, but it may lose out on the level of detail in formal process descriptions.

The views in the study by F. Gonzalez-Lopez *et al.* (2022), which proposed domain-oriented design of business process architecture, also turned out to be similar in their emphasis on business logic. The importance of a clear description of the domain before forming the system architecture was highlighted. The proposed approach also underscored the decisive role of initial business process analysis through UML models, where grouping activities by logical attributes contributed to the isolation of contexts. The difference lies in the emphasis on simplifying the initial modelling steps, whereas the study by F. Gonzalez-Lopez *et al.* (2022) aimed at deep integration of domain concepts in large-scale corporate systems.

Compared to the work of S. Kapferer & O. Zimmermann (2020), where the main focus was on creating specialised languages and tools for strategic contextual mapping, the obtained results are less "technologically specialised." The authors demonstrated the benefits of DSL for describing interactions between contexts, which reduced the risk of incorrect terminology interpretation. In this study, priority was given to the simplicity of UML diagrams, which did not require additional description languages. Such an approach proved effective for application in teams with varying levels of technical expertise, although it potentially yields in terms of formalisation level, as discussed by S. Kapferer & O. Zimmermann (2020).

Important parallels can be seen with the developments of C. Zhong (2024), which conducted a systematic analysis of DDD application in microservices. This study focused on how clear context boundaries help avoid duplication of functions and incorrect distribution of data between services. The obtained results confirmed the validity of these conclusions, as the formation of a context map at an early stage safeguarded against unnecessary fragmentation. However, that study was based on the analysis of existing real-world systems, whereas this work focused on a method for transforming activity diagrams into a contextual model.

The studies by H. Vural & M. Koyuncu (2021) were aimed at investigating the impact of DDD on the modularity of microservices, demonstrating that correct identification of bounded contexts defines the boundaries of service responsibilities. The obtained conclusions align with the authors' position on the crucial role of proper grouping of business processes in preventing duplication and terminology conflicts. At the same time, the works of these authors provided quantitative indicators evaluating system modularity, whereas the proposed methodology placed greater emphasis on qualitative analysis (logical grouping of actions and decomposition of processes).

It is also interesting to compare the approach to domain decomposition highlighted in the work of J. Sangabriel-Alarcón *et al.* (2024), where it was noted that strategic contextual mapping and precise definition of bounded contexts contribute to better system scalability and reduced technical debt. The obtained results confirmed this thesis, as transparent context boundaries facilitated working with domains that change over time. At the same time, the risks of oversaturating the

system with overly granular contexts, which can complicate coordination between teams, were considered. The presented methodology included a criterion for minimising inter-contextual dependencies, but the method prioritised preliminary business analysis, whereas the scholar's work favoured a multi-step iterative verification of context boundaries based on real data.

A comparison with the results of previous studies demonstrated that the field of integrating DDD with business process models is evolving in several directions. Some researchers focused on more formalised transformation mechanisms, using, for example, BPMN, specialised DSLs, and algorithmic modularity criteria. Others, however, emphasised a simplified initial analysis that fosters better mutual understanding among teams with varying levels of technical expertise. Ultimately, it is the balance between formal depth and the accessibility of the proposed methods that determines success in domain decomposition and the formation of a flexible, scalable architecture within DDD.

CONCLUSIONS

Throughout the study, a phased methodology was developed that effectively combines business analysis with technical DDD practices, allowing for the clear and prompt identification and description of key operations within the restaurant business domain, all within their respective bounded contexts. In the initial stage, business requirements and primary process models were thoroughly analysed, and key participants and their main activities were identified. The subsequent stage involved a formal analysis of UML activity diagrams, which included systematising actions based on functional commonality, business objectives, and interdependencies. This enabled the separation of several autonomous bounded contexts and the formation of a clear understanding of the nature of interactions between them, serving as the foundation for creating a coherent context map.

The research results demonstrated the practical effectiveness of the proposed method and confirmed that combining business analysis with UML diagrams significantly simplifies and accelerates the modelling of complex systems at early design stages. The use of intuitive UML diagram tools ensured the accessibility and transparency of the process for both business analysts and software developers, fostering improved communication and mutual understanding between multidisciplinary teams. The proposed method not only allowed for clear definition of responsibilities between system parts but also significantly reduced the risks of semantic conflicts and errors arising from ambiguous interpretations of business requirements. Furthermore, this approach can contribute to building flexible and scalable architectural solutions that can adapt to changes in business processes without substantial costs for system rework and adaptation. However, it is important to note that the practical implementation of the method in this study was conducted using the example of a single domain – the restaurant business – which requires additional verification of the method's universality and scalability in other industries and business contexts. For further refinement of the method, it is recommended to conduct similar studies in other domains, such as financial services, healthcare, logistics, and so forth.

In future research, a promising direction is the automation of the process for transforming UML activity diagrams into context maps using modern CASE tools and software platforms. This can significantly enhance the efficiency and accuracy of initial modelling, reduce time expenditure, and facilitate the integration of this method into existing software development processes. The integration of the method with cutting-edge technologies, such as machine learning and artificial intelligence, holds promise for further enhancing business process analysis and enabling the automatic identification of key contexts and interrelationships.

ACKNOWLEDGEMENTS

None.

FUNDING

None.

CONFLICT OF INTEREST

None.

REFERENCES

[1] Albuquerque, F., Torres, A.S., & Berssaneti, F.T. (2020). Lean product development and agile project management in the construction industry. *Revista de Gestão*, 27(2), 135-151. doi: 10.1108/REGE-01-2019-0021.

[2] Alfadel, M., Aljasser, K., & Alshayeb, M. (2020). Empirical study of the relationship between design patterns and code smells. *PLoS ONE*, 15(4), article number e0231731. doi: 10.1371/journal.pone.0231731.

[3] Ambler, S.W. (2005). UML activity diagrams. In *The elements of UML™ 2.0 style* (pp. 113-131). Cambridge: Cambridge University Press. doi: 10.1017/CBO9780511817533.011.

[4] Da Silva, C.E., Gomes, E.L., & Basu, S.S. (2022). BPM2DDD: A systematic process for identifying domains from business processes models. *Software*, 1(4), 417-449. doi: 10.3390/software1040018.

[5] Evans, E. (2003). *Domain-driven design: Tackling complexity in the heart of software*. Boston: Addison-Wesley.

[6] Farshidi, S., Jansen, S., & van der Werf, J.M. (2020). Capturing software architecture knowledge for pattern-driven design. *Journal of Systems and Software*, 169, article number 110714. doi: 10.1016/j.jss.2020.110714.

[7] Gadiyar, S. (2024). Leveraging event storming for enhanced product development in FinTech: A collaborative approach to simplifying complex business processes. *Journal of Mathematical & Computer Applications*, 3(5), article number E146. doi: 10.47363/JMCA/2024(3)E146.

[8] Gonzalez-Lopez, F., Bustos, G., Muñoz-Gama, J., & Sepúlveda, M. (2022). Domain model based design of business process architectures. *Applied Sciences*, 12(5), article number 2563. doi: 10.3390/app12052563.
[9] Hlybovets, A., & Paprotskyi, I. (2024). Increasing the fault tolerance in microservice architecture. *Cybernetics and Systems Analysis*, 60(3), 480-488. doi: 10.1007/s10559-024-00689-0.
[10] Irshad, M., Börstler, J., & Petersen, K. (2022). Supporting refactoring of BDD specifications – an empirical study. *Information and Software Technology*, 141, article number 106717. doi: 10.1016/j.infsof.2021.106717.
[11] Kapferer, S., & Zimmermann, O. (2020). Domain-specific language and tools for strategic domain-driven design, context mapping and bounded context modeling. In *Proceedings of the 8th international conference on modeling and software development* (pp. 299-306). Valetta: SciTePress. doi: 10.5220/0008910502990306.
[12] Khoshnevis, S. (2023). A search-based identification of variable microservices for enterprise SaaS. *Frontiers of Computer Science*, 17(3), article number 173208. doi: 10.1007/s11704-022-1390-4.
[13] Özkan, O., Babur, Ö., & van den Brand, M. (2023). Refactoring with domain-driven design in an industrial context. *Empirical Software Engineering*, 28, article number 94. doi: 10.1007/s10664-023-10310-1.
[14] Ozkaya, M., & Erata, F. (2020). A survey on the practical use of UML for different software architecture viewpoints. *Information and Software Technology*, 121, article number 106275. doi: 10.1016/j.infsof.2020.106275.
[15] Sangabriel-Alarcón, J., Ocharán-Hernández, J.O., Limón, X., & Cortés-Verdín, M.K. (2024). Domain-driven design in microservices-based systems development: A systematic literature review and thematic analysis. *Programming and Computer Software*, 50(8), 742-770. doi: 10.1134/S0361768824700749.
[16] Valdivia, J.A., Lora-González, A., Limón, X., Cortés-Verdin, M.K., & Ocharán-Hernández, J.O. (2020). Patterns related to microservice architecture: A multivocal literature review. *Programming and Computer Software*, 46(8), 594-608. doi:10.1134/S0361768820080253.
[17] Vernon, V. (2013). *Implementing domain-driven design*. Boston: Addison-Wesley Professional.
[18] Vural, H., & Koyuncu, M. (2021). Does domain-driven design lead to finding the optimal modularity of a microservice? *IEEE Access*, 9, 32721-32733. doi: 10.1109/ACCESS.2021.3060895.
[19] Waseem, M., Liang, P., Shahin, M., Di Salle, A., & Márquez, G. (2021). Design, monitoring, and testing of microservices systems: The practitioners' perspective. *Journal of Systems and Software*, 182, article number 111061. doi: 10.1016/j.jss.2021.111061.
[20] Zhong, C., Li, S., Huang, H., Liu, X., Chen, Z., Zhang, Y., & Zhang, H. (2024). Domain-driven design for microservices: An evidence-based investigation. *IEEE Transactions on Software Engineering*, 50(6), 1425-1449. doi: 10.1109/TSE.2024.3385835.

Метод структурованого перетворення діаграми активності бізнес-процесів у карту контексту предметно-орієнтованого проєктування

Сергій Московко

Аспірант
Вінницький національний технічний університет
21021, вул. Хмельницьке шосе, 95, м. Вінниця, Україна
https://orcid.org/0009-0001-4376-0187

Роман Квєтний

Доктор технічних наук, професор
Вінницький національний технічний університет
21021, вул. Хмельницьке шосе, 95, м. Вінниця, Україна
https://orcid.org/0000-0002-9192-9258

Анотація. Актуальність дослідження полягає в потребі впровадження уніфікованих методів для структурованого формування контекстних карт, здатних надійно та точно відображати бізнес-процеси під час проєктування предметно-орієнтованих інформаційних систем. Метою роботи було розроблення та обґрунтування систематичного методу перетворення діаграм активності бізнес-процесів у контекстні карти, що оптимізує процес моделювання та підвищує точність відображення взаємодій у домені. Для досягнення цієї мети було застосовано методи формального аналізу діаграм активності, структурованого виявлення обмежених контекстів, групування взаємопов'язаних дій і встановлення типів взаємодій між ними. Основні результати дослідження полягають у створенні поетапного методу, який дозволяє ідентифікувати ключові елементи бізнес-процесів, об'єднувати їх у логічно узгоджені контексти та визначати міжконтекстні взаємозв'язки без використання складних нотацій. Запропонований підхід дає змогу виокремлювати обмежені контексти на основі аналізу діаграм активності, виявляти динамічні залежності між діями різних учасників та формувати узгоджену контекстну карту. Ефективність методу продемонстрована на прикладі ресторанного бізнесу, де за його допомогою було чітко ідентифіковано взаємодії між процесами обслуговування клієнтів, кухнею, управлінням запасами та обробкою платежів. Це сприяло прозорому розподілу відповідальності, покращило керування складністю системи, прискорило проєктування та підвищило узгодженість між технічною реалізацією та бізнес-вимогами. Запропонований підхід також покращив взаєморозуміння між розробниками та експертами предметної області, оскільки забезпечує чіткі межі відповідальності та більш адаптивну архітектуру системи. Крім того, результати підтвердили можливість масштабування методу на інші галузі, що свідчить про його універсальність та широку застосовність. Практична цінність роботи полягає у можливості масштабування методу для застосування в різних доменах, що робить його корисним інструментом для бізнес-аналітиків і архітекторів програмного забезпечення під час проєктування складних систем та підвищення ефективності прийняття архітектурних рішень

Ключові слова: інформаційні системи; аналіз бізнес-процесів; уніфікована мова моделювання; трансформація моделей; визначення обмежених контекстів