Digitally signed by
Technical Scientific
Library, TUM
Reason: I attest to the
accuracy and integrity of
this document

TECHNICAL UNIVERSITY OF MOLDOVA

FACULTY OF COMPUTERS, INFORMATICS AND MICROELECTRONICS

DEPARTMENT OF SOFTWARE ENGINEERING

DISTRIBUTED APPLICATIONS: MODELING CONCURRENCY USING UML

Course Notes



2026

CZU 004.75:004.434(075.8)
D 53

This work supplements the didactic and methodological support for the courses Distributed Applications Programming and Advanced Distributed Applications. These courses are included in the bachelor's and master's programs in Information Technology and Security and Software Engineering at the Technical University of Moldova. The content is based on the authors' research, including their doctoral studies. It has also been used as methodological support for bachelor's and master's students in the design and architecture of information systems.

Approved by the Department of Software Engineering, Minutes No. 3 dated 05.11.2026, and by the Council of the Faculty of Computers, Informatics, and Microelectronics, Minutes No. 3 dated 18.11.2026.

Authors:
Dumitru Ciorbă, assoc. prof. dr.,
Andrei Poștaru, dr.,
Ana Țurcan,
Viorel Munteanu, dr.,
Ernest Bîtcă

Reviewer:
Irina Cojuhari, dr., assoc. prof.

DESCRIEREA CIP A CAMEREI NAȚIONALE A CĂRȚII DIN RM

Distributed applications: modeling concurrency using UML: Course notes / Dumitru Ciorbă, Andrei Poștaru, Ana Țurcan [et al.]; Technical University of Moldova, Faculty of Computers, Informatics and Microelectronics, Department of Software Engineering.
– Chișinău: Tehnica-UTM, 2026. – 144 p.: fig., tab.
Aut. indicați pe verso p. de tit. – Bibliogr.: p. 141-143 (34 tit.). – 50 ex.
ISBN 978-9975-64-688-8.
Text: nemediat.
004.75:004.434(075.8)
D 53

ISBN 978-9975-64-688-8 **© UTM, 2026**

Table of contents

Introduction and course structure 6

1 Conceptual foundations 11

1.1 Concurrency, parallelism, and distribution 13
1.2 Entities and relationships for modeling concurrency 18
1.3 Core concepts of the UML modeling language 21

2 Forms of concurrency 29

2.1 Concurrency and multidimensional decentralization 31
2.2 Conceptual models for concurrency 35
2.3 General considerations for concurrency in UML 46

3 Concurrency in structural diagrams 51

3.1 Class diagram 53
3.2 Composite structure diagram 57
3.3 Component, package and deployment diagrams 60

4 Concurrency in behavioral diagrams 67

4.1 Use case diagram 70
4.2 Interaction diagrams 77
4.3 State and activity diagrams 89

5 Modeling concurrency-aware architectures 99

5.1 From local concurrency to architectural decisions 101
5.2 Architectural views and their consistency 104
5.3 Architectural strategies for managing concurrency 114
5.4 Architectural styles emphasizing concurrency 119

Glossary 137

Bibliography 141

List of figures

Figure 1.1 – Interleaved concurrency, imperfect and perfect parallelism 15
Figure 1.2 – Decomposition of systems: entities..... 19
Figure 1.3 – Decomposition of systems: communication structures..... 20
Figure 1.4 – UML infrastructure, a hierarchy of definitions..... 22
Figure 1.5 – The redefinable elements *Classifier* and *Feature* 24
Figure 1.6 – Specification of behavior in UML 26
Figure 2.1 – Decentralization dimensions of *Enslow'*s model 32
Figure 2.2 – The two forms of the event 35
Figure 2.3 – Animation models..... 36
Figure 2.4 – Interaction models..... 39
Figure 2.5 – Interface models..... 43
Figure 2.6 – Levels and mechanisms for specifying concurrency..... 49
Figure 3.1 – Class diagram of the *Ornamental Garden* problem..... 56
Figure 3.2 – Composite structure diagram of the *Ornamental Garden* problem.. 59
Figure 3.3 – Deployment diagram and the components allocated..... 62
Figure 4.1 – Use Case diagram for the *Readers–Writers* problem..... 76
Figure 4.2 – Active objects and interactions in the sequence diagram..... 80
Figure 4.3 – Sequence diagram for the *Readers–Writers* problem 83
Figure 4.4 – Synchronization in the sequence diagram..... 84
Figure 4.5 – Sequence diagram for *the Producer–Consumer* problem 87
Figure 4.6 – Synchronization of operations in the sequence diagram..... 88
Figure 4.7 – Process with orthogonal (concurrent) states 91
Figure 4.8 – Mutual exclusion for a shared resource in an activity diagram..... 93
Figure 4.9 – Nondeterministic *object flow* through a *CentralBufferNode*..... 95
Figure 4.10 – Expansion region 96
Figure 5.1 – Alternatives for the *Ornamental Garden* problem..... 102

Figure 5.2 – Concurrency across complementary architectural views 109
Figure 5.3 – Service boundaries in an order-processing system 122
Figure 5.4 – Independent reactions to an order confirmation event 125
Figure 5.5 – Composite structure of the *Ornamental Garden* system 128
Figure 5.6 – Data dependencies and overlapping pipeline stages 131

Introduction and course structure

Modeling concurrency in distributed applications is a relevant subject for modern software development. As applications become increasingly complex and require distribution across a wide range of devices, the ability to identify and model concurrency becomes essential. These course notes focus on a single cross-cutting architectural concern—concurrency—and show how UML can be used to produce analyzable and implementable designs for distributed applications.

Incorrect handling of concurrency can lead to problems such as race conditions and deadlocks, which can reduce an application's performance and reliability. By correctly understanding and modeling concurrency, developers can design systems that address these challenges.

The UML language is a standard modeling language in the software industry and is often used to visualize, specify, build, and document the entities of a software system. Therefore, understanding how UML can be used to model concurrency in distributed applications is a valuable skill for any software developer.

Purpose and architecture focus

The purpose of these course notes is to build a practical understanding of modeling concurrency in distributed applications using UML, with an explicit architecture lens. The notes connect

conceptual foundations (what concurrency is and why it matters) to concrete UML mechanisms that let architects and designers recognize, express, and validate concurrent behavior in a distributed setting.

Learning objectives

At the end of this course, students will be able to:

- distinguish concurrency, parallelism, and distribution and explain their roles in software systems;
- identify concurrent execution units, shared resources, interactions, and synchronization constraints;
- use UML structural and behavioral diagrams to represent concurrency in distributed applications;
- analyze concurrency problems and select appropriate modeling techniques;
- relate local concurrency mechanisms and UML representations to architectural decisions concerning execution units, state ownership, communication, coordination, deployment, and consistency across architectural views.

The course notes are structured into five chapters, which develop the subject from conceptual foundations and individual UML mechanisms to the architectural analysis of concurrent systems:

1. *Conceptual foundations* - this chapter introduces the key concepts of concurrency, parallelism, and distribution. It also introduces the conceptual entities and relationships necessary for modeling concurrency, as well as the basic elements of the UML modeling language.
2. *Forms of concurrency* – in this chapter the concept of concurrency is explored, guided by the idea of multidimensional decentralization, the different conceptual models for concurrency, and how concurrency can be specified in UML.
3. *Concurrency in structural diagrams* – this chapter focuses on how concurrency is represented in UML structural diagrams, such as class, composite structure, component, and deployment diagrams.
4. *Concurrency in behavioral diagrams* – this chapter focuses on how concurrency is modeled in UML behavioral diagrams, including use case, interaction, state, and activity diagrams.
5. *Modeling concurrency-aware architectures* – this chapter integrates the concepts and UML mechanisms introduced earlier. It examines how architectural decisions determine execution units, responsibility distribution, state ownership, communication, and coordination. It also explains how structural, behavioral, coordination, and deployment

descriptions can be checked for consistency and illustrates these decisions through several concurrency-oriented architectural styles.

The content supports two complementary reading paths.

By learning progression:

- Conceptual foundations (Ch. 1): concurrency, parallelism, distribution, modeling entities and basic UML concepts.
- Conceptual models of concurrency (Ch. 2): decentralization, execution, interaction, interfaces, and coordination.
- Structural modeling (Ch. 3): active elements, interfaces, components, resources, and deployment contexts.
- Behavioral modeling (Ch. 4): interaction, parallel execution, synchronization, critical regions, and nondeterminism.
- Architectural synthesis (Ch. 5): execution units, state ownership, communication choices, coordination strategies, architectural styles, and consistency across complementary views.

By architectural concern:

- Structure and allocation: Where are the active elements, components, resources, and execution environments? (Ch. 3 and 5)
- Behavior and interaction: Which activities may overlap, who communicates with whom, and is communication synchronous or asynchronous? (Ch. 2, 4, and 5)

- Coordination and state ownership: Who owns mutable state, where are conflicts resolved, and which ordering or mutual-exclusion rules apply? (Ch. 2, 4, and 5)
- Deployment and execution context: Which software elements share an execution context, and which may execute independently? (Ch. 1, 3, and 5)
- Cross-view consistency: Do structural, behavioral, coordination, and deployment models describe the same execution units, communication mechanisms, and synchronization rules? (Ch. 5)

Based on the topics covered in this course, the following prerequisites are outlined:

1. Basic knowledge of programming;
2. Understanding of the basic concepts of object-oriented programming[1];
3. Understanding of the basic concepts of software systems.

To master the material, students are expected to study independently and explore the topics beyond the examples presented in these course notes. When developing distributed applications, they are encouraged to treat concurrency as a distinct architectural concern and to model it explicitly using UML.

[1] Because UML is used to model object-oriented systems, students should be familiar with concepts such as classes, objects, inheritance, and polymorphism.

Bibliography

[1] F. Păunescu and D. P. Goleşteanu, Sisteme cu prelucrare distribuită şi aplicaţiile lor, Bucureşti: Editura Tehnică, 1993, p. 560.

[2] H. Georgescu, Programare concurentă. Teorie şi aplicaţii, Bucureşti: Editura Tehnică, 1996, p. 240.

[3] G. C. Hillar, Professional Parallel Programming with C#, Indianapolis, Indiana: Wiley Publishing, Inc., 2011.

[4] M. Ben-Ari, Principles of Concurrent and Distributed Programming, 2nd ed., Pearson Education, 2006, p. 361.

[5] I. Şora, "Textbook on Parallel Algorithms Course," "Politehnica" University of Timisoara, [Online]. Available: http://staff.cs.upt.ro/~ioana/calc_par/. [Accessed 08 Aug. 2026].

[6] P. H. Enslow, "What is „distributed" data processing system?," *Computer,* vol. 11, no. 1, pp. 13-21, January 1978.

[7] J. Duffy, Concurrent programming on Windows, Boston: Pearson Education, Inc., 2009.

[8] T. G. Mattson, B. A. Sanders and B. L. Massingill, Patterns for Parallel Programming, Addison-Wesley Professional, 2005.

[9] J. Ortega-Arjona, *Architectural Patterns for Parallel Programming,* London, 2006, p. 301.

[10] Object Management Group Inc., "OMG Unified Modeling Language, Version 2.5.1," 2017. [Online]. Available: https://www.omg.org/spec/UML. [Accessed 17 Aug. 2026].

[11] L. F. Andrade, J. L. Fiadeiro, J. Gouveia, G. Koutsoukos, A. Lopes and M. Wermelinger, *Coordination Patterns for Component-Based Systems,* Curitiba, PR, Brazil, 2001, pp. 29-39.

[12] P. Ciancarini and C. Hankin, Eds., Coordination Languages and Models, vol. LNCS 1061, Springer, 1996, p. 443.

[13] P. Ciancarini and H. Wiklicky, Eds., Coordination Models and Languages, vol. LNCS 4038, Springer, 2006.

[14] G. A. Papadopoulos and F. Arbab, "Coordination models and languages," in *Advances in Computers*, vol. 46, M. V. Zelkowitz, Ed., Academic Press, 1998, pp. 329-400.

[15] P. T. Eugster, P. A. Felber, R. Guerraoui and A.-M. Kermarrec, "The many faces of publish/subscribe," *ACM Computing Surveys,* vol. 35, no. 2, pp. 114-131, June 2003.

[16] D. G. Kafura and G. Lavender, "Concurrent Object-Oriented Languages and the Inheritance Anomaly," in *Parallel Computers: Theory and Practice*, T. Cassavant, P. Tvrdik and F. Plasil, Eds., Los Alamitos, CA: IEEE CS Press, 1995, pp. 165-198.

[17] D. M. Suciu, *Tehnici de implementare a concurenței în analiza și proiectarea orientată pe obiecte,* Cluj-Napoca: Universitatea Babeș-Bolyai, 2001.

[18] A. Burns and A. J. Wellings, Concurrent and real-time programming in Ada 2005, Cambridge, U.K.: Cambridge University Press, 2007.

[19] S. Gérard and I. Ober, *Parallelism/Concurrency specification within UML,* C. Atkinson, B. Douglass, S. Gérard, A. Moore, I. Ober, F. Terrier and B. Selic, Eds., 2001.

[20] P. Stevens, "UML and Concurrency," *Proceedings of the abstract state machines 10th international conference on Advances in theory and practice,* pp. 151-166, 2003.

[21] J. Magee and J. Kramer, Concurrency: state models & Java programs, Chichester, England: John Wiley & Sons, 1999.

[22] D. Ciorba, V. Besliu, A. T. Chronopoulos and A. Postaru, *Arch-pattern based design and aspect-oriented implementation of Readers-Writers concurrent problem,* vol. 15, Chisinau: Institute of Mathematics and Computer Science (Academy of Sciences of Moldova), 2007, pp. 338-353.

[23] J. Küster-Filipe, "Modelling concurrent interactions," *Theoretical Computer Science,* vol. 351, no. 2, pp. 203-220, 21 Feb 2006.

[24] M. W. Maier, D. Emery and R. Hilliard, "Software Architecture: Introducing IEEE Standard 1471," *Computer,* vol. 34, no. 4, pp. 107-109, Apr. 2001.

[25] N. Rozanski and E. Woods, Software Systems Architecture: Working with Stakeholders Using Viewpoints and Perspectives, Addison-Wesley Professional, 2005.

[26] International Organization for Standardization, "ISO/IEC/IEEE 42010:2022 Software, systems and enterprise — Architecture description," 2022.

[27] P. Kruchten, *Architectural Blueprints — The "4+1" View Model of Software Architecture,* vol. 12, 1995.

[28] A. T. Sampson, *Process-oriented patterns for concurrent software engineering,* University of Kent, 2008.

[29] J. Bacon and T. Harris, Operating systems: concurrent and distributed software design, Pearson Education, 2003.

[30] J. Lewis and M. Fowler, "Microservices," 25 March 2014. [Online]. Available: https://martinfowler.com/articles/microservices.html. [Accessed 23 Oct. 2026].

[31] M. Fowler, "Event Collaboration," 19 Jun 2006. [Online]. Available: http://martinfowler.com/eaaDev/EventCollaboration.html. [Accessed 25 Oct. 2026].

[32] G. Hohpe and B. Woolf, "Patterns and Best Practices for Enterprise Integration," 20 October 2003. [Online]. Available: http://www.eaipatterns.com/. [Accessed 26 Oct. 2026].

[33] G. Agha, *Actors: A Model of Concurrent Computation in Distributed Systems,* MIT Press, 1986, p. 190.

[34] Akka.io, "How the Actor Model Meets the Needs of Modern, Distributed Systems," [Online]. Available: https://doc.akka.io/libraries/akka-core/current/typed/guide/actors-intro.html. [Accessed 26 Oct. 2026].