From Objects to Agents and Beyond: Stabilizing Structures in Evolving Coordination Contexts

Ulrike Steffens[1,*,†]

[1]University of Applied Science Hamburg, Berliner Tor 7, 20099 Hamburg, Germany

Abstract

Recent advances in agentic AI have renewed discussions about the coordination of increasingly autonomous software systems. This paper argues that such challenges are not unprecedented but represent another step in the historical evolution of coordination complexity in software engineering. By examining object-oriented software engineering, service-oriented systems, enterprise architecture management, and platform-based software systems, recurring patterns of coordination and the emergence of stabilizing structures are identified. Based on this historical perspective, agentic systems are interpreted as a new coordination context in which coordination increasingly shifts to runtime. The paper concludes that many current approaches to governing agentic systems can be understood as the next generation of stabilizing structures developed to manage increasing coordination complexity.

Keywords

Agentic AI, software engineering, coordination complexity

1. Introduction

Recent advances in agentic AI have sparked discussions about the implications of increasingly autonomous software systems [1]. In contrast to traditional AI applications that primarily generate predictions or content, agentic systems are expected to pursue goals, make decisions, coordinate activities, and interact with other agents as well as organizational resources. As a consequence, questions of coordination, governance, and controllability have become central topics in both research and practice.

This paper argues that agentic systems introduce a new *coordination context*. We use this term to describe socio-technical settings in which multiple autonomous or semi-autonomous entities interact while pursuing individual or shared goals and require mechanisms to coordinate activities, decisions, resources, and responsibilities. Although the technological realization of such contexts has changed considerably over time, many underlying coordination concerns have repeatedly reappeared throughout the evolution of digital information systems.

To investigate this observation, this paper adopts a historical perspective by revisiting four exemplary coordination contexts from software engineering and information systems: object-oriented software engineering, service-oriented systems, enterprise architecture management, and platform-based software systems. While these contexts differ substantially in their objectives, abstraction levels, and implementation technologies, they all introduced new forms of coordination complexity. In response, their respective communities developed concepts and practices that helped to make this complexity manageable, including design principles, architectural patterns, governance mechanisms, and quality models.

Current discussions on agentic AI similarly emphasize mechanisms intended to stabilize increasingly autonomous systems, including guardrails, policies, human oversight, and architectural principles. Throughout the remainder of this paper, we use the term *stabilizing structures* as an umbrella term for such mechanisms, including design principles, governance concepts, architectural constructs, quality models, and runtime controls that contribute to making increasing coordination complexity manageable.

PNSE'26, International Workshop on Petri Nets and Software Engineering, 2026
ulrike.steffens@haw-hamburg.de (U. Steffens)
https://www.haw-hamburg.de/ (U. Steffens)
0009-0006-7487-970X (U. Steffens)
© 2026 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

CEUR Workshop Proceedings ceur-ws.org ISSN 1613-0073
published 2026-08-03

CEUR-WS.org/Vol-4236/keynote2.pdf

Rather than proposing additional stabilizing structures, this paper offers a historical perspective that helps interpret their emergence. We argue that the development of stabilizing structures is not unique to agentic systems but follows a recurring pattern that can be observed throughout the evolution of digital information systems whenever new forms of coordination complexity arise.

The contribution of this paper is therefore twofold. First, it proposes a historical perspective that relates current developments in agentic AI to recurring coordination challenges already observed in previous coordination contexts. Second, it derives a conceptual research perspective by discussing how this historical pattern may inform future work on coordination, governance, and controllability in increasingly autonomous systems.

2. A Historical Perspective on Coordination Complexity

The evolution of digital information systems has repeatedly been accompanied by increasing coordination complexity. As systems became larger, more distributed, and more adaptive, software engineering communities were challenged not only by technical implementation problems but also by the question of how activities, decisions, resources, goals, and responsibilities could be coordinated effectively.

Rather than providing a comprehensive historical survey, this section revisits four exemplary coordination contexts that illustrate recurring patterns in the evolution of digital information systems. The selected examples neither represent a complete chronology nor an exhaustive classification of software engineering approaches. Instead, they were chosen because each introduced a distinct form of coordination complexity and prompted its respective community to develop new stabilizing structures.

The purpose of this historical perspective is therefore not to compare technologies but to identify recurring relationships between increasing coordination complexity and the emergence of stabilizing structures. This recurring pattern provides the conceptual basis for the subsequent discussion of agentic systems.

2.1. Object-Oriented Software Engineering

Object-oriented software engineering introduced a new way of structuring software systems. Objects became the primary units of decomposition and interaction. This enabled developers to build increasingly complex systems but also introduced new forms of coordination complexity. As systems grew, object interactions, inheritance hierarchies, and dependencies became increasingly difficult to understand and maintain. Common symptoms included highly coupled systems, deep inheritance structures, God objects, and the well-known “Big Ball of Mud” architecture.

The software engineering community responded by developing a variety of stabilizing structures. Modeling languages such as UML provided common representations for discussing system structure [2]. Design principles, later popularized through the SOLID principles [3], constrained how object-oriented mechanisms should be applied. Design patterns, popularized by the *Gang of Four* [4], captured proven solutions to recurring design problems and provided a shared vocabulary for discussing software architectures. Rather than limiting the flexibility of object-oriented programming, these concepts helped developers manage the increasing coordination complexity introduced by increasingly complex object-oriented systems.

2.2. Service-Oriented Systems

Service-oriented systems shifted software engineering from coordinating objects within a single application to coordinating distributed services across technical and, in many cases, organizational boundaries. Independent services could evolve separately, be deployed and scaled independently, and be reused in different business processes. While this increased flexibility, it also introduced new coordination complexity. Services had to interact reliably despite distribution, heterogeneous implementations, changing availability, and organizational autonomy.

The software engineering community responded by developing a range of stabilizing structures that addressed different aspects of distributed coordination [5]. Service contracts established explicit interfaces and interaction semantics between services. Orchestration and choreography provided complementary approaches for coordinating distributed service interactions, while governance frameworks helped manage service lifecycles, ownership, versioning, and compliance across organizational boundaries. Together, these concepts helped organizations coordinate increasingly distributed systems without sacrificing interoperability or maintainability.

2.3. Enterprise Architecture Management

Enterprise Architecture Management (EAM) extended coordination beyond software systems to the alignment of business capabilities, organizational structures, projects, applications, and technology landscapes. In contrast to previous coordination contexts, Enterprise Architecture Management focused less on coordinating software artifacts themselves than on aligning continuous organizational and technological change. As organizations became increasingly digital and interconnected, managing dependencies between business and IT evolved into a central coordination challenge.

The EAM community responded by developing stabilizing structures, including architecture principles [6], enterprise architecture frameworks [7], modeling languages such as ArchiMate [8], and capability-based planning approaches to guide decision making, support organizational and technological alignment, and ensure long-term coherence.

2.4. Platform-Based Software Systems

Platform-based software systems introduced a different form of coordination complexity. Rather than developing individual software solutions, they sought to provide reusable capabilities that could be configured and adapted to diverse application contexts. This shifted coordination from software components alone to the relationship between generic platform functionality and application-specific requirements. As a result, flexibility, reuse, quality assurance, and context-specific adaptation had to be balanced simultaneously.

The platform engineering community responded by developing stabilizing structures such as reference architectures, quality models, parameterization concepts, and platform governance. The DaFne platform illustrates how these structures support configurable reuse and context-specific adaptation while preserving quality and consistency [9].

Many of the objectives pursued by platform-based systems — including specialization, contextual adaptation, reusable capabilities, and explicit quality assurance — reappear in current discussions on agentic AI. While platform-based systems primarily achieved adaptation through predefined architectures, parameterization, and configuration, agentic systems increasingly shift adaptation and coordination into runtime.

3. Agentic Systems as a New Coordination Context

Across the four coordination contexts discussed above, a recurring pattern becomes visible. Each context introduced a new form of coordination complexity that could not be addressed by technical implementation alone. Instead, the corresponding communities developed stabilizing structures that provided shared abstractions, principles, governance mechanisms, and quality criteria for managing increasingly complex socio-technical systems. Although these structures differed substantially in their concrete realization, they fulfilled a similar purpose: enabling coordination without eliminating the flexibility introduced by the respective technological developments.

This historical perspective motivates the following question: should agentic systems be understood as an entirely new class of coordination problems, or do they represent another coordination context that follows a familiar evolutionary pattern? While the technologies underlying agentic AI differ

fundamentally from previous software paradigms, many of the coordination concerns discussed in the previous section appear familiar.

Activities, decisions, resources, goals, and responsibilities remain central concerns in agentic systems. What changes is not the nature of these concerns but the way they are coordinated. Rather than being coordinated exclusively through predefined software structures, they become increasingly distributed across autonomous actors capable of reasoning, delegation, and adaptation during runtime [10]. A seemingly simple delegation scenario illustrates this shift. An agent receiving a task may decide to decompose it, delegate subtasks to other agents, or request access to organizational resources. Questions regarding authorization, accountability, responsibility, and oversight immediately arise. These questions are familiar from previous coordination contexts. However, they now emerge in environments where coordination itself becomes increasingly adaptive. The distinguishing characteristic of agentic systems is therefore not the existence of coordination problems but the increasing shift of coordination into runtime.

4. Discussion

Current research on agentic AI already proposes a broad range of mechanisms intended to stabilize increasingly autonomous systems. Guardrails, policies, human oversight, architectural principles, and governance mechanisms illustrate that the need for coordinating and controlling agentic systems is widely recognized [1, 11]. Rather than proposing additional mechanisms, the historical perspective developed in this paper offers a complementary interpretation of these ongoing developments.

Across the coordination contexts discussed in this paper, increasing coordination complexity was consistently accompanied by the development of stabilizing structures. Although these structures differed considerably in their concrete realization, they fulfilled similar purposes: they established shared abstractions, constrained design decisions, supported communication among stakeholders, and enabled increasingly complex socio-technical systems to remain manageable. From this perspective, current developments in agentic AI appear less as an exceptional phenomenon than as another instance of a recurring evolutionary pattern.

At the same time, agentic systems differ from previous coordination contexts in one important respect. Rather than coordinating predefined software artifacts or execution flows, they increasingly shift coordination itself into runtime. Autonomous agents may reason about goals, adapt their behavior, delegate tasks, and coordinate dynamically with other agents and organizational resources. Consequently, runtime mechanisms naturally become a central concern in current research and practice.

The historical perspective presented in this paper complements this discussion by focusing on a different question. Previous coordination contexts were accompanied over time by additional stabilizing structures that complemented immediate coordination mechanisms and supported the long-term evolution of increasingly complex systems. Whether similar structures will emerge for agentic systems, and what form they may take, remains an open research question.

From this perspective, future research may benefit from studying not only individual stabilization mechanisms but also the broader process through which software engineering communities develop shared abstractions, principles, governance concepts, and other stabilizing structures in response to new coordination contexts. Rather than predicting which specific structures will prove successful, this paper proposes a historical perspective that may help interpret these ongoing developments.

Declaration on Generative AI

The authors have not employed any Generative AI tools.

References

[1] Z. Xi, W. Chen, X. Guo, W. He, Y. Ding, B. Hong, M. Zhang, J. Wang, S. Jin, E. Zhou, et al., The rise and potential of large language model based agents: A survey, Science China Information Sciences 68 (2025) 121101.

[2] G. Booch, J. Rumbaugh, I. Jacobson, Unified modeling language user guide,(the 2nd edition), 2005.

[3] R. C. Martin, Design principles and design patterns, Object Mentor 1 (2000).

[4] E. Gamma, Design patterns: elements of reusable object-oriented software, Pearson Education India, 1995.

[5] T. Erl, SOA design patterns, Pearson Education, 2008.

[6] D. Greefhorst, E. Proper, Architecture Principles: The Cornerstones of Enterprise Architecture, Springer, 2011.

[7] J. Schekkerman, How to survive in the jungle of enterprise architecture frameworks: Creating or choosing an enterprise architecture framework, Trafford Publishing, 2004.

[8] M. Lankhorst, Enterprise architecture at work: modelling, communication, and analysis, Springer, 2013.

[9] A. Glass, K. Tokuç, J. R. Noennig, U. Steffens, B. Bek, Dafne: Data fusion generator and synthetic data generation for cities, in: KES International Symposium on Agent and Multi-Agent Systems: Technologies and Applications, Springer, 2023, pp. 99–108.

[10] M. Wooldridge, Reasoning about rational agents, MIT press, 2003.

[11] R. Gangavarapu, AI governance: preparing for the rise of agentic AI, in: Mastering AI Governance: A Guide to Building Trustworthy and Transparent AI Systems, Springer, 2025, pp. 111–119.