Aspect	Object Synchronizer Pattern	Publish Subscribe Pattern	Pass the Baton Pattern	Behavioural Synchronization Pattern
Synchronization Mechanism	Decouples synchronization from functionality, supports various policies.	Research lacks exploration into novel aspect-oriented synchronization mechanisms, specifically in isolating concerns from concrete classes.	Lack efficiency in highly contended scenarios due to its sequential unblocking approach, potentially leading to increased contention and performance degradation.	Utilizes an internal thread for operation execution, simplifies synchronized access to shared resources.
Modularity	Provides encapsulation, modularity, extensibility, and reuse of synchronization policies.	Insufficient investigation into modular synchronization approaches, overlooking benefits of separating logic into reusable aspects.	Exhibit limited modularity as it focuses primarily on managing concurrency within a single critical section, potentially leading to difficulties in separating concerns and maintaining code clarity.	Enhances modularity by separating operation execution from invocation, simplifies maintenance.
Performance	May introduce overhead due to increased number of classes and objects.	Gap in understanding performance implications of aspect-oriented synchronization, particularly in terms of runtime overhead.	Suffer from performance bottlenecks, especially under heavy loads, due to its reliance on sequential unblocking and potential contention issues within the critical section.	Improves performance by simplifying synchronization and reducing overhead associated with synchronization policies.
Scalability	May face challenges in handling high contention scenarios efficiently.	Limited research on scalability of aspect-oriented synchronization patterns for evolving system requirements.	Face challenges in scaling effectively to accommodate increasing numbers of concurrent processes or threads, as its sequential unblocking approach may introduce scalability limitations and contention overhead.	Offers scalability by facilitating concurrent access to shared resources and efficient resource management.
Flexibility	Offers customization of synchronization policies but may not be optimal for all scenarios.	Inadequate examination of flexibility in adapting aspect-oriented synchronization to diverse application needs.	Lack flexibility in adapting to changing system requirements or concurrency patterns, as its design primarily focuses on managing access to shared resources through a rigid sequential unblocking mechanism.	Provides flexibility through customizable object behaviours and policies, adaptable to varying concurrency requirements.
Maintenance	Requires careful management of synchronization code within objects, potential for complexity and errors.	Lack of research on maintenance advantages of aspect-oriented synchronization patterns.	Pose maintenance challenges over time, particularly in large-scale systems, as its design may lead to complex and tightly coupled code structures, making it harder to implement changes or updates.	Simplifies maintenance with a clear separation between operation execution and synchronization, easier debugging, and troubleshooting.
Reusability	Allows for independent reuse of synchronization code and functionality but may introduce code tangling.	Research gap in exploring reusability benefits of encapsulating synchronization logic in aspects.	Offer limited reusability potential across different components or systems, as its design is tailored specifically to manage concurrency within critical sections, potentially limiting its applicability in diverse contexts.	Enhances reusability through encapsulation of object behaviours, promotes code reuse without coupling synchronization to functionality.