DOI: https://doi.org/10.15276/ict.03.2026.04
UDC 004.41.045:004.056

Index-based routing model for multi-facet proxy architectures in EVM blockchains

Artem A. Buchikhin[1)]
ORCID: https://orcid.org/0009-0006-9562-1273; artembuchihin@gmail.com
Viktor O. Boltenkov[1)]
ORCID: https://orcid.org/0000-0003-3366-974X; vaboltenkov@gmail.com. Scopus Author ID: 57203623617
[1)] Odesa Polytechnic National University, 1, Shevchenko Ave. Odesa, 65044, Ukraine

ABSTRACT

Multi-facet proxy architectures address the limitations of monolithic smart contracts by distributing protocol logic among multiple implementation contracts while preserving a common external interface and storage context. Existing standardized architectures primarily use function selectors as routing identifiers, coupling facet selection with behavioral identification and consequently introducing selector management, collision considerations, and function-level routing metadata. This study analyzes these limitations and introduces an index-based routing model that represents facet identity independently from function identity. The proposed model appends a routing index to calldata outside the Application Binary Interface-encoded function call, allowing the router to select a facet directly while preserving conventional function selectors for compatibility with existing Ethereum interfaces and tooling. The model is applied to an upgradeable multi-facet proxy architecture specified by the proposed standard and realized by the Cento Proxy reference implementation. The resulting architecture provides facet-level routing granularity, removes selector-specific facet management and selector-collision constraints, supports selector-independent facet entry points, and reduces the routing-related implementation surface. The reference implementation is evaluated against the Diamonds reference implementation under equivalent conditions using gas and bytecode measurements. The results demonstrate substantial reductions in facet-management and introspection costs, while maintaining near-identical standard routing performance and introducing only a small overhead for protocol-to-protocol routing caused by on-chain construction of routing metadata. The complete protocol also requires substantially less deployment cost, while the reference implementation contains considerably less runtime bytecode. The study demonstrates that separating facet identity from function identity provides an alternative routing abstraction with measurable architectural and efficiency benefits while remaining compatible with the established Ethereum Virtual Machine ecosystem.

Keywords: Cento; multi-facet proxy; index-based routing; facet identity; ERC-2535; function selector; routing model; smart contract; EVM; blockchain; Ethereum; proxy architecture; smart contract architecture; ERC-8349

For citation: **Buchikhin A. A., Boltenkov V. O. "Index-based routing model for multi-facet proxy architectures in EVM blockchains". *Informatics. Culture. Technology*. 2026; Vol. 3 No. 1(3): 48–57. DOI: https://doi.org/10.15276/ict.03.2026.04**

The study aims to analyze architectural limitations of existing standardized multi-facet proxy architectures in EVM (Ethereum Virtual Machine) blockchains, introduce an index-based routing model as an alternative approach to representing a target facet identity, evaluate its compatibility with the existing EVM ecosystem and its architectural trade-offs, and measure the reference implementation of the proposed routing model against implementations of existing routing models under identical conditions.

Relevance. In EVM blockchains, smart contracts, or hosted executable programs, have been conceived as immutable pieces of logic whose unwavering execution is guaranteed by the system. However, when supporting complex protocols (projects), developers encountered continuous problems that could not have been foreseen at the development stage: bug fixes, security improvements, feature extensions, and adaptation to evolving standards [1], [2], all while having to preserve the storage context and interface identity (contract address), which are directly tied to the immutable bytecode of the smart contract. Inability to patch threat vectors in time has led to the most famous hacks in Ethereum's history [3], [4], [5]. Consequently, the emergence of mechanisms that permit controlled modification or extension of deployed protocols under constraints of the EVM is crucial for sufficiently complex and long-lived applications in particular, and for decentralized ecosystems overall.

The proxy design pattern, enabled by the *DELEGATECALL* opcode introduced in EIP-7 [6], is one of the design patterns that separates storage from application logic. At its core, it defines a contract (Proxy) that forwards calls to another contract (Implementation) while preserving the execution and storage contexts of the Proxy contract. It allows developers to swap the address of the implementation contract, changing only the logic executed by subsequent calls. But despite the clean mechanism, the breadth of its applications has led to a wide variety of architectural patterns [7], [8].

As a separate architectural constraint, historically imposed by the EVM, EIP-170 established a 24,576-byte limit on deployed runtime bytecode [9]. This limitation affects all deployed protocols, regardless of whether they seek modularity for separation of concerns and cheaper upgradability or require more space for the logic under a common storage context and interface identity. Although proposals have considered replacing or relaxing this constraint [10], [11], [12], the code-size bound remains unchanged and continues to pose a challenge for sufficiently large protocols. Application-level patterns have consequently provided workarounds, and while the fallback-extension pattern is relevant to this topic, multi-facet proxy architectures provide a more mature model for investigation as they explicitly face the challenge of determining which facet (one implementation contract among many active contracts in a protocol) should execute a given call.

Analysis of existing multi-facet architectures. ERC-2535 (Diamonds) pioneered multi-facet proxies [13], [14]. Its routing model is selector-centric: the fallback function extracts the first four bytes of calldata as the function selector and uses it to identify the facet to which the call is delegated. The high-level view of the architecture is shown in Fig. 1.

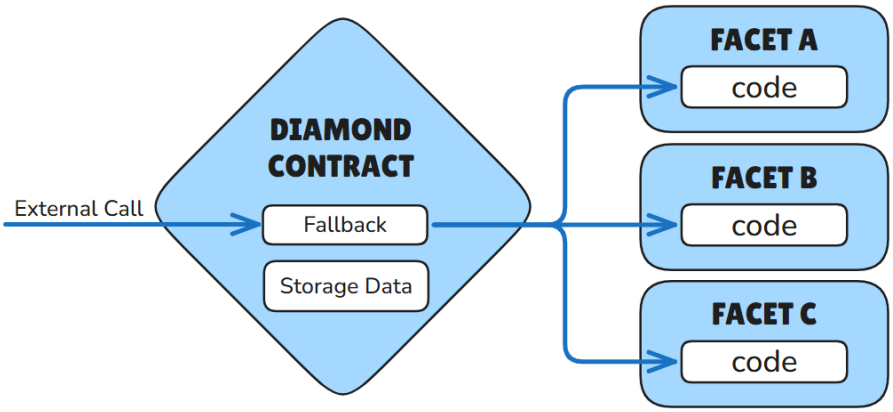


Fig. 1. **Architectural composition of ERC-2535**
Source: **compiled by the authors [13]**

Thus, the identifier collection of this routing model can be represented as a *mapping* from function selectors to target facets. The same relation is further reflected in the upgrade and introspection mechanisms of this pattern, where facet additions, replacements, and removals operate on collections of function selectors, while the Diamond Loupe exposes the selectors associated with each facet [13].

The choice of the first four bytes and their association with function selectors wasn't arbitrary. Under the Solidity ABI (Application Binary Interface) specification [15], the first four bytes of calldata contain the function selector, while the remaining calldata contains the function arguments encoded according to the ABI, using 32-byte words as its basic encoding unit (Fig. 2).

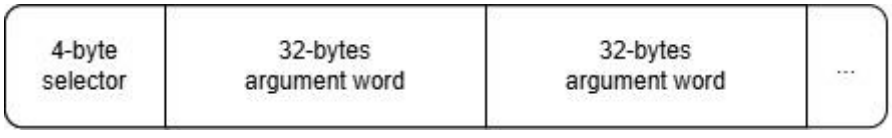


Fig. 2. **Standard general calldata structure**
Source: **compiled by the authors**

The selector is derived from the function signature and serves as a standardized behavioral identifier used by independently developed software, including external contracts and development tooling [15]. Consequently, reusing it for routing provides direct compatibility with the established Ethereum calling convention, but also makes facet selection dependent on behavioral identifiers.

ERC-8153 (Facet-Based Diamonds) changes the management model by making facets the primary objects of deployment and upgrade operations and allowing selectors to be discovered from the facets themselves. Nevertheless, its execution model remains selector-centric: the first four bytes of calldata continue to determine the target facet [16]. Thus, ERC-8153 can be considered a movement toward facet-oriented management while retaining selector-centric routing.

ERC-8167 (Modular Dispatch Proxies) takes the opposite direction by further simplifying the architecture around selector-based dispatch. Each delegate (facet) effectively implements only one function, making it the unit of modular dispatch [17]. The resulting model therefore fully embraces selector-centric routing rather than introducing an independent identity for a facet.

These architectures expose an important distinction between function identity and facet identity. Function selectors identify externally observable behavior and provide compatibility between independently developed software, whereas facet identities exist to determine which implementation module executes protocol logic. In selector-centric routing, these distinct concerns are represented through the same identifier, causing routing metadata to scale with exported functions rather than facets. Consequently, adding or modifying functionality requires corresponding selector management, independently developed facets must account for selector collisions, and upgrades performed at the facet level may require modifications to potentially large collections of selector associations.

The index-based routing model (Cento) introduced in this study separates these identities. Instead of inferring the target facet from the function selector, it represents facet identity explicitly as a routing index appended to calldata outside the ABI-encoded function call (Fig. 3).

Fig. 3. **Calldata structure for the Cento routing model**
Source: **compiled by the authors**

The router uses the trailing index to identify the target facet and removes it before executing delegatecall, so the delegated facet receives conventional function calldata without the routing metadata. Therefore, the routing table is the logical identifier collection of this routing model which, unlike selector-based ones, has consequent identifier keys, which will later result in a different data structure implementing it.

The separation treats protocol routing and interface compatibility as independent concerns. Protocol Routing identifies the target facet by its routing index, while Interface Compatibility associates conventional Ethereum function selectors with the appropriate routing index. The latter is necessarily implemented as an adapter because routing metadata changes the complete call format from the perspective of an existing interface. Although adding an index during off-chain call formation incurs negligible computational overhead, deployed protocols have long-established interaction formats that external protocols and applications may not be able or willing to modify. Function selectors consequently retain their established role as behavioral and compatibility identifiers, while routing indices provide dedicated facet identifiers.

Application and evaluation scope. The index-based routing model is not inherently limited to upgradeable proxy architectures. It can be applied to any multi-facet architecture in which a dispatch mechanism selects one implementation contract from a collection of active facets. For objective evaluation, however, this study applies the model to upgradeable multi-facet proxy architecture, allowing its routing properties to be compared with an established architecture under

equivalent conditions. ERC-2535 is selected as the primary reference because it is the established Final ERC for multi-facet proxies and provides a complete reference implementation suitable for comparative measurement. ERC-8153 and ERC-8167 are therefore considered in the architectural analysis but are not used as the primary quantitative baseline.

The evaluated architecture follows the conventional deployment model in which the proxy and its facets are deployed as separate contracts, while the routing identifier collection is maintained in the router storage. Other architectural dimensions that are not intrinsic to the routing model are kept equivalent where applicable, allowing the measurements to reflect the difference in routing representation.

The resulting architecture is specified by ERC-8349 (Index-Based Multi-Facet Proxy) [18], whose distinctive property is the routing index being a single byte, providing 256 possible facet identifiers. Its reference implementation, referred to as Cento Proxy, applies the index-based routing model to an upgradeable multi-facet proxy while preserving the common properties required from this class of architectures: a stable external contract address, shared proxy storage, facet-level implementation, controlled addition, replacement, and removal of facets, and introspection of the active routing configuration. It consequently serves as the experimental subject for evaluating the proposed routing model rather than as a definition of the routing model itself. The architecture itself exhibits a set of architectural properties that extend beyond gas consumption.

Architectural consequences. Since a facet is identified by a routing index rather than by the selectors of the functions it provides, adding a facet does not require assigning its function selectors to the routing collection. Likewise, replacement or removal of a facet can be performed by changing its single routing association without modifying the selectors of the functions implemented by that facet. Routing metadata therefore follows changes in the set of facets rather than changes in the set of externally callable functions.

Selector uniqueness is no longer required for facet composition. In a selector-centric architecture, independently developed facets may expose identical function selectors, requiring the resulting protocol to account for selector collisions when composing them [19]. Under index-based routing, selectors remain identifiers of externally callable behavior, while facet selection is performed through an independent identifier. Different facets can consequently contain functions with identical selectors without competing for the same routing entry, provided that their respective routing indices are distinct.

Because routing is performed at facet-level granularity, functions that do not require a conventional function selector can also be associated with individual facets. In particular, a facet can provide *receive()* and *fallback()* behavior without requiring these entry points to be represented as ordinary selector-to-facet associations, as neither special function is identified by a conventional four-byte function selector. This allows the introduced routing model to preserve the distinction between function-level interface identification and facet-level execution selection even for calldata-free or otherwise non-standard entry points.

The distinction also affects the gas cost of batch transactions containing multiple calls to the same facet. In selector-centric routing, each function selector requires access to its corresponding routing entry, so calls to different functions of the same facet may independently incur the cost of accessing previously cold selector-specific routing state. In index-based routing, functions implemented by the same facet share a common routing index and therefore a common routing entry. Once accessed on the first call, subsequent calls to the same facet can access the already warm routing entry, regardless of which function is invoked. Therefore, the cost of warming routing state is incurred per facet rather than per selector, reducing the routing overhead of transactions containing multiple calls to the same facet.

Finally, separating routing metadata from behavioral identifiers reduces the amount of function-specific state that must be considered when auditing routing correctness. A selector-centric architecture requires the consistency of potentially large collections of selector-to-facet associations to be considered during protocol deployment and upgrades. An index-based architecture instead concentrates this relationship at the facet level, while the compatibility layer determines which

conventional selectors are associated with each routing index. This does not remove the need to audit the routing mechanism or upgrade authorization, but shrinks the surface of the object whose correctness must be established from individual function associations toward the set of active facets and their identities [20].

Experimental realization and its evaluation. The reference implementation represents the routing identifier collection as a static array in router storage, with each position corresponding to one possible routing index and containing the metadata required to resolve the associated facet. Occupancy of this identifier space is tracked separately through a bitmap, allowing the implementation to determine whether a routing index is occupied without traversing the collection. Thanks to the routing index being restricted to a single byte, the bitmap is implemented as a *uint256*, which is a data structure that consists of precisely 256 bits. This allows the occupancy state of the entire identifier space to be read or modified through a single storage access. The bitmap therefore represents the state of the identifier space, while the static array represents the routing table itself. This separation is an implementation choice of the reference implementation and is not intrinsic to the index-based routing model.

Following the reference implementation of ERC-2535, the router is implemented as an immutable execution-dispatch component responsible for extracting the routing index, resolving the corresponding facet through the routing table, removing the routing metadata from calldata, and performing delegatecall. Facet management and observability are implemented through facets operating in the shared proxy storage context.

ERC-8349 remains subject to further changes during the development of the standard, and the reference implementation may therefore change together with it. The implementation evaluated in this study is fixed to a specific repository commit, which defines the exact implementation state used for the reported measurements. Later changes to the implementation or the specification do not affect the results presented here, while implementations based on later revisions may differ from the evaluated version.

The experimental evaluation was conducted using Foundry under identical execution conditions for both the ERC-2535 reference implementation [13] and the Cento Proxy implementation. Gas consumption was measured using *vm.snapshotGasLastCall()*, which records the gas consumed by the most recent call. This method was selected because it provides consistent measurements across different execution conditions and avoids relying on gas estimates obtained before transaction execution. Although the measurement mechanism introduces a small call overhead, the same procedure is applied to both implementations, allowing the overhead to remain constant across the compared measurements.

The first group of measurements evaluates facet addition, replacement, removal, and core introspection operations. Upgrade benchmarks compare equivalent atomic multi-facet updates, with the evaluated example modifying between one and five facets, each exposing ten functions (Fig. 4a, Fig. 4b and Fig. 4c). Introspection is also measured for different numbers of active facets to observe how the cost develops as the protocol grows (Fig. 4d).

The measurements demonstrate the effect of changing the routing granularity from functions to facets. This results in savings of 87.6% for adding five facets, 77.9 % for replacing them, and 83.3% for removing them. The scale of improvement, combined with the bounded 256-facet identifier space, makes facet management independent of the number of functions implemented by each facet. In the evaluated implementation, even adding the maximum of 256 facets remains below 7 million gas, allowing the complete facet set to be managed atomically within a single transaction. In this respect, the index-based model enables larger atomic updates within a single transaction than selector-centric routing, whose operation cost additionally scales with the number of selectors associated with the affected facets. Introspection follows the same distinction: selector-centric routing exposes function-level routing relationships, while index-based routing represents the active facet set directly. As the number of facets increases from three to eight, with the core three facets containing substantially fewer functions, the measured introspection savings grow from 76.0 % to 90.6 %, reflecting the different amount of routing metadata that must be processed.

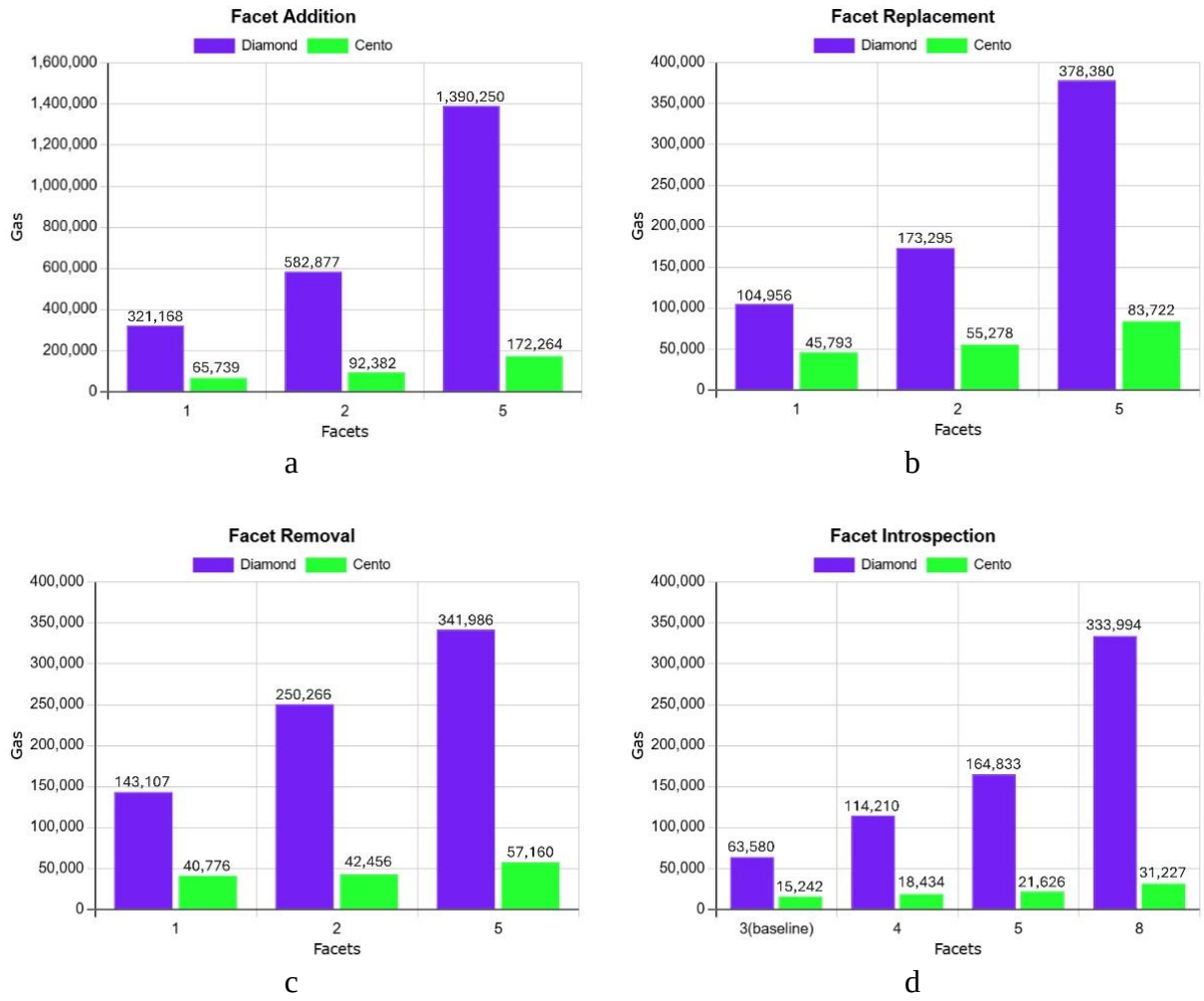


Fig. 4. **Title:**
a – Facet addition comparison chart; b – Facet replacement comparison chart;
c – Facet removal comparison chart; d – Facet introspection comparison chart
Source: **compiled by the authors**

Routing overhead was measured using identical protocol calls and derived as the difference between a direct call to a facet and a call through a router. Standard interface calls represent calldata formed externally, while protocol-to-protocol calls include the on-chain addition of the routing byte, as shown in Table 1.

Table 1. **Routing overhead comparison**

Configuration	*Diamonds*	*Cento Proxy*	*Difference*
External call	4,898 gas	4,873 gas	–25 gas (–0.5 %)
Call from another protocol	4,898 gas	4,909 gas	+11 gas (+0.2 %)

Source: **compiled by the authors**

The routing measurements show that the index-based model preserves near-identical execution cost while introducing a small context-dependent trade-off. Standard routing is 25 gas cheaper (0.5%) than ERC-2535, whereas protocol-to-protocol routing is 11 gas more expensive (0.2 %) because the routing byte must additionally be appended to calldata during execution. The latter overhead is therefore a consequence of constructing the index-based routing metadata on-chain rather than of the facet lookup itself. In both cases, the difference remains small compared with the savings observed in facet management and introspection operations.

The implementations were additionally compared by deployed runtime bytecode size (Table 2).

Table 2. **Reference implementation bytecode-size comparison**

Component	*Diamonds*	*Cento Proxy*	*Optimization*
DiamondCutFacet / FacetManager	4,232 B	1,938 B	54.2 %
OwnershipFacet / Ownership	542 B	454 B	16.2 %
DiamondLoupeFacet / Observability	2,565 B	1,625 B	36.6 %
DiamondInit	351 B	—	—
Diamond / CentoRouter	268 B	472 B	–76.1 %
Total runtime bytecode:	**7,958 B**	**4,489 B**	**43.6 %**

Source: **compiled by the authors**

The Cento Proxy reference implementation contains 4,489 bytes of runtime bytecode compared with 7,958 bytes for the ERC-2535 reference implementation. This reduction reflects the removal of selector-specific routing and management logic from the implementation. The router itself, however, is larger in Cento (472 bytes compared with 268 bytes), demonstrating that the reduction is not achieved by minimizing the dispatch component alone, but by reducing the amount of function-specific routing infrastructure required by the architecture as a whole. This also provides a measurable basis for the previously described reduction in routing-related audit surface.

Deployment benchmarks compare complete protocol initialization, including the router and essential facets (Table 3).

Table 3. **Reference implementation deployment cost comparison**

Component / Operation	*Diamonds*	*Cento Proxy*	*Optimization*
DiamondCutFacet / FacetManager	966,047 gas	469,459 gas	54.2 %
OwnershipFacet / Ownership	169,651 gas	151,009 gas	16.2 %
DiamondLoupeFacet / Observability	607,839 gas	403,832 gas	36.6 %
DiamondInit	128,959 gas	—	—
Diamond / CentoRouter	262,171 gas	369,939 gas	– 41.1 %
Bootstrap	311,051 gas	—	—
Total router-only cost:	**573,222 gas**	**369,939 gas**	**35.5 %**
Total complete cost:	**2,445,718 gas**	**1,394,239 gas**	**43.0 %**

Source: **compiled by the authors**

Despite its larger runtime bytecode, the Cento Router requires 369,939 gas to deploy, compared with 573,222 gas for the ERC-2535 Router, resulting in a 35.5 % reduction. This difference demonstrates that deployment cost is not determined solely by runtime bytecode size: the initialization procedure and the amount of state established during deployment also contribute to the resulting cost. The optimized initialization of the Cento Router therefore allows it to remain cheaper to deploy despite containing more runtime bytecode.

Complete protocol initialization shows an even larger reduction, requiring 1,394,239 gas for Cento Proxy compared with 2,445,718 gas for ERC-2535, a 43.0 % reduction. The two measurements represent different deployment scenarios: the complete-deployment result evaluates the cost of establishing the entire protocol, while the router-only result is relevant when facets are already deployed and a new router is installed around them.

Conclusions. The study analyzed the routing models of existing standardized multi-facet proxies and identified a persistent coupling between function and facet identities in selector-centric routing. Function selectors provide behavioral and interface compatibility across the Ethereum ecosystem, while facet identity determines the implementation contract to which protocol logic is delegated. The index-based routing model separates these concerns by representing facet identity

through an independent routing index while retaining conventional function selectors for interface compatibility.

Applied to an upgradeable multi-facet proxy, the proposed model changes routing and facet management from function-level to facet-level granularity. This removes selector-specific management and collision constraints, enables selector-independent facet entry points, reduces routing-related implementation surface, and changes the cost of repeated routing accesses from a selector-oriented to a facet-oriented model. These are architectural properties of the routing model and are therefore distinct from the gas optimizations measured in the reference implementation.

The experimental evaluation against ERC-2535 confirms that distinction quantitatively. Facet-management operations achieve savings of 77.9-87.6 %, core introspection achieves 76.0-90.6 % savings across the evaluated facet counts, and standard routing differs by only 25 gas (0.5 %), while protocol-to-protocol routing incurs 11 additional gas (0.2 %). The complete Cento Proxy implementation contains 43.6 % less runtime bytecode than the ERC-2535 implementation. Deployment also requires less gas in both evaluated scenarios: 35.5 % less for the router alone and 43.0% less for complete protocol initialization. The measurements therefore indicate that the observed efficiency does not result solely from optimizing the dispatcher, but from reducing the amount of routing state and function-specific management required by the architecture. Since the separation between facet identity and function identity is independent of the surrounding proxy architecture, the index-based routing model is a general approach that can also be applied to multi-facet architectures beyond the one demonstrated in this study.

Conflict of interests: Authors declare that they do not have conflict of interests in financial, personal, copyright or any other nature, which could have influenced the research, as well as the results published in this article

Financing: Research was carried out without financial support

Accessibility data: The simulation code used to obtain the reported results is publicly available in an open repository: [https://github.com/Arhemius/cento-proxy/tree/c41978470e1f29fd74da226f651a8c69e5a638bd]

Using artificial intelligence tools: During preparation of the manuscript, the tool of artificial intelligence (AI) called Grammarly was used for grammatical and stylistic text checks. All fragments processed by it were checked and edited by the author. AI was not used to generate text, charts, tables, evaluation results or formulation conclusions. Scientific provisions, results, and conclusions are entirely the intellectual contribution of the authors

REFERENCES

1. Salehi, M., Clark, J. & Mannan, M. “Not so immutable: Upgradeability of smart contracts on Ethereum”. In *Financial Cryptography and Data Security. FC 2022 International Workshops*. *Lecture Notes in Computer Science*. *Springer*. 2023. p. 539–554. DOI: https://doi.org/10.1007/978-3-031-32415-4_33.

2. Ebrahimi, A. M., Adams, B., Oliva, G. A. & Hassan, A. E. “A large-scale exploratory study on the proxy pattern in Ethereum”. *Empirical Software Engineering*. 2024; 29 (4): 81. DOI: https://doi.org/10.1007/s10664-024-10485-1.

3. Dhillon, V., Metcalf, D. & Hooper, M. “The DAO Hacked”. In *Blockchain Enabled Applications*. Cham: Apress, 2017. p. 67–78. DOI: https://doi.org/10.1007/978-1-4842-3081-7_6.

4. Palladino, S. “The Parity Wallet Hack Explained”. *OpenZeppelin Blog*. 2017. – Available from: https://blog.openzeppelin.com/on-the-parity-wallet-multisig-hack-405a8c12e8f7. – [Accessed: 22.08.2026].

5. Editorial Team. “Compound Rekt”. *REKT.news*. October 2021. – Available from: https://rekt.news/compound-rekt. – [Accessed: 22.08.2026].

6. Buterin, V. “EIP-7: DELEGATECALL”. *Ethereum Improvement Proposals*, no. 7, November 2015. – Available from: https://eips.ethereum.org/EIPS/eip-7. – Accessed: 22.08.2026.

7. AL Amri, S., Aniello, L. & Sassone, V. “A review of upgradeable smart contract patterns based on OpenZeppelin technique”. *The Journal of the British Blockchain Association*. 2023. DOI: https://doi.org/10.31585/jbba-6-1-(3)2023.

8. Huang, Y., Wu, X., Wang, Q., Qian, Z., Chen, X., Tang, M. & Zheng, Z. “The sword of damocles: Upgradeable smart contract in Ethereal”. In *Proceedings of the 32nd IEEE/ACM*

International Conference on Program Comprehension (ICPC). New York: ACM. 2024. p. 333–345. DOI: https://doi.org/10.1145/3643916.3644426.

9. Buterin, V. “EIP-170: Contract code size limit”. *Ethereum Improvement Proposals*, no. 170, November 2016. – Available from: https://eips.ethereum.org/EIPS/eip-170. – [Accessed: 22.08.2026].

10. Zhou, Q. “EIP-5027: Remove the limit on contract code size [STAGNANT]”. *Ethereum Improvement Proposals*, no. 5027, April 2022. – Available from: https://eips.ethereum.org/EIPS/eip-5027. – [Accessed: 22.08.2025].

11. Cooper, C., Zhou, Q., Matt & Rakita, D. “EIP-7907: Meter contract code size and increase limit [DRAFT]”. *Ethereum Improvement Proposals*, no. 7907, March 2025. – Available from: https://eips.ethereum.org/EIPS/eip-7907. – [Accessed: 22.08.2026].

12. Rebuffo, G. & Adams, B. “EIP-7954: Increase maximum contract size [REVIEW]”. *Ethereum Improvement Proposals*, no. 7954, June 2025. – Available from: https://eips.ethereum.org/EIPS/eip-7954. – [Accessed: 22.08.2026].

13. Mudge, N. “ERC-2535: Diamonds, Multi-Facet Proxy”. *Ethereum Improvement Proposals*, no. 2535, February 2020. – Available from: https://eips.ethereum.org/EIPS/eip-2535. – [Accessed: 22.08.2025].

14. van Vulpen, P., Heijnen, H., Mens, S., Kroon, T. & Jansen, S. “Upgradeable diamond smart contracts in decentralized autonomous organizations”. *Frontiers in Blockchain*. 2024; 7: 1481914. DOI: https://doi.org/10.3389/fbloc.2024.1481914.

15. Solidity Team. “Contract ABI Specification”. *Solidity Documentation*. – Available from: https://docs.soliditylang.org/en/latest/abi-spec.html. – [Accessed: 22.05.2026].

16. Mudge, N. “ERC-8153: Facet-Based Diamonds [REVIEW]”. *Ethereum Improvement Proposals*, no. 8153, February 2026. – Available from: https://eips.ethereum.org/EIPS/eip-8153. – [Accessed: 22.08.2026].

17. Morriss, W. & Svarz, R. “ERC-8167: Modular Dispatch Proxies [REVIEW]”. *Ethereum Improvement Proposals*, no. 8167, February 2026. – Available from: https://eips.ethereum.org/EIPS/eip-8167. – [Accessed: 22.08.2026].

18. Buchikhin, A. “ERC-8349: Index-Based Multi-Facet Proxy [DRAFT]”. *Ethereum Improvement Proposals*, no. 8349, 2026. – Available from: https://eips.ethereum.org/EIPS/eip-8349. – [Accessed: 22.08.2026].

19. Kumar, G., Saha, R., Conti, M. & Buchanan, W. J. “LEAGAN: a decentralized version-control framework for upgradeable smart contracts”. *IEEE Transactions on Services Computing*. 2025; 18 (3): 1529–1542. – Available from: https://napier-repository.worktribe.com/output/4246545. – [Accessed: 22.08.2026].

20. Chen, J., Zhong, Z. & Zhou, Y. “USCSafe: Identifying permission vulnerabilities in upgradeable smart contracts”. In *Blockchain and Trustworthy Systems. BlockSys 2025. Communications in Computer and Information Science. Singapore: Springer*. 2026. p. 61–78. DOI: https://doi.org/10.1007/978-981-95-3480-7_5.

Received 14.07.2026
Received after revision 28.08.2026
Accepted 04.09.2026

DOI: https://doi.org/10.15276/ict.03.2026.04
УДК 004.41.045:004.056

Модель маршрутизації на основі індексів для мультифасетних проксі архітектур у EVM блокчейнах

Бучіхін Артем Андрійович[1)]
ORCID: https://orcid.org/0009-0006-9562-1273; artembuchihin@gmail.com

Болтьонков Віктор Олексійович[1)]
ORCID: https://orcid.org/0000-0003-3366-974X; vaboltenkov@gmail.com. Scopus Author ID: 57203623617
[1)] Національний університет «Одеська політехніка», пр. Шевченка, 1. Одеса, 65044, Україна

АНОТАЦІЯ

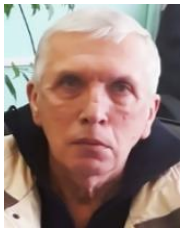
Мультифасетні проксі-архітектури усувають обмеження монолітних смартконтрактів шляхом розподілу логіки протоколу між кількома контрактами реалізації, зберігаючи при цьому спільний зовнішній інтерфейс і контекст сховища. Наявні стандартизовані архітектури переважно використовують селектори функцій як ідентифікатори маршрутизації, безпосередньо пов'язуючи вибір фасету з ідентифікацією функції та, як наслідок, зумовлюючи необхідність керування селекторами, врахування можливих колізій між ними та використання метаданих маршрутизації на рівні окремих функцій. У цьому дослідженні проаналізовано зазначені обмеження та запропоновано індексну модель маршрутизації, у якій ідентичність фасету відокремлена від ідентичності функції. Запропонована модель доповнює calldata індексом маршрутизації, розміщеним за межами виклику функції, закодованого відповідно до Application Binary Interface, завдяки чому маршрутизатор може безпосередньо визначати фасет, зберігаючи традиційне використання селекторів функцій для сумісності з наявними інтерфейсами та інструментами екосистеми Ethereum. Модель застосовано до оновлюваної мультифасетної проксі-архітектури, специфікованої запропонованим стандартом та реалізованої у зразковій реалізації Cento Proxy. Отримана архітектура забезпечує маршрутизацію з гранулярністю на рівні фасетів, усуває необхідність керування фасетами на основі селекторів та обмеження, пов'язані з колізіями селекторів, дає змогу використовувати точки входу фасетів незалежно від селекторів та зменшує обсяг програмного коду, пов'язаного з маршрутизацією. Зразкову реалізацію порівняно зі зразковою реалізацією Diamonds за еквівалентних умов із використанням вимірювань витрат газу та обсягу байткоду. Результати демонструють істотне зменшення витрат на керування фасетами та інтроспекцію при збереженні майже ідентичної вартості стандартної маршрутизації та незначних накладних витрат для маршрутизації між протоколами, спричинених формуванням метаданих маршрутизації на блокчейні. Повне розгортання протоколу також потребує істотно менших витрат, тоді як зразкова реалізація містить помітно менший обсяг виконуваного байткоду. Результати дослідження демонструють, що відокремлення ідентичності фасету від ідентичності функції дає змогу сформувати альтернативну абстракцію маршрутизації з архітектурними перевагами та вимірюваними перевагами ефективності, зберігаючи водночас сумісність із усталеною екосистемою Ethereum Virtual Machine..

Ключові слова: Cento; мультифасетний проксі; індексна маршрутизація; ідентичність фасету; ERC-2535; селектор функції; модель маршрутизації; смартконтракт; EVM; блокчейн; Ethereum; проксі-архітектура; архітектура смартконтрактів; ERC-8349

ABOUT THE AUTHORS

Artem A. Buchikhin - PhD Student of the Department of Information Systems, Odesa Polytechnic National University, 1, Shevchenko Ave. Odesa, 65044, Ukraine
ORCID: https://orcid.org/0009-0006-9562-1273; artembuchihin@gmail.com.
***Research field*:** blockchain technologies

Бучіхін Артем Андрійович - аспірант кафедри Інформаційних систем, Національний університет «Одеська політехніка», пр. Шевченка, 1. Одеса, 65044, Україна
***Галузь досліджень*:** блокчейн технології



Viktor O. Boltenkov - PhD, Associate Professor of the Information System Department, Odesa Polytechnic National University, 1, Shevchenko Ave. Odesa, 65044, Ukraine
ORCID: https://orcid.org/0000-0003-3366-974X; vaboltenkov@gmail.com. Scopus Author ID: 57203623617
***Research field*:** blockchain technologies; signal processing

Болтьонков Віктор Олексійович - кандидат технічних наук, доцент кафедри Інформаційних систем, Національний університет «Одеська політехніка», пр. Шевченка, 1. Одеса, 65044, Україна
***Галузь досліджень*:** блокчейн технології, цифрова обробка сигналів