IJWR

BitML: A UML Profile for Bitcoin Blockchain

Behrouz Sefid-Dashti[a], Javad Salimi Sartakhti[b*], Hasan Daghigh[c]

[a] Electrical and computer engineering department, University of Kashan, Kashan, Iran,: b.sefiddashti@grad.kashanu.ac.ir

[b] Electrical and computer engineering department, University of Kashan, Kashan, Iran; salimi@kashanu.ac.ir

[c] Faculty of Mathematical Science, University of Kashan, Kashan, Iran; hasan@kashanu.ac.ir

ABSTRACT

Blockchain is a technology that enables distributed and secure data structures for various business domains. Bitcoin is a notable blockchain application that is a decentralized digital currency with immense popularity and value. Bitcoin involves many concepts and processes that require modelling for better comprehension and development. Modelling is a technique that simplifies and abstracts a system at a certain level of detail and accuracy. Software modelling is applied in Model-Driven Engineering (MDE), which automates the software development process using models and transformations. Domain-specific languages (DSLs) are languages that are customized for a specific domain and offer intuitive syntax for domain experts. To address the need for specialized tools for Bitcoin blockchain modelling, we propose a novel Unified Modelling Language (UML) profile that is specifically designed for this domain. UML is a standard general-purpose modelling language that can be extended by profiles to support specific domains. A meta-model is a model that defines the syntax and semantics of a modelling language. The proposed meta-model, which includes stereotypes, tagged values, enumerations, and constraints defined by Object Constraint Language (OCL), is defined as a UML profile. The proposed meta-model is implemented in the Sparx Enterprise Architect (Sparx EA) modelling tool, which is a widely used tool for software modelling and design. To validate the practicality and effectiveness of the proposed UML profile, we developed a real-world case study using the proposed meta-model and conducted an evaluation using the Architecture Tradeoff Analysis Method (ATAM). The results showed the proposed UML profile promising.

Keywords*— *Meta-model, UML profile, Bitcoin, Blockchain, OCL, Domain-specific language.

1. Introduction

Blockchain is a revolutionary technology that offers enormous advantages for various business domains. It enables the creation of decentralized applications that run on a distributed network of nodes, without the need for intermediaries or central authorities. Blockchain applications can provide transparency, trust, security, and efficiency for various transactions and processes. One of the most prominent and pioneering applications of blockchain is Bitcoin, a digital currency that operates on a peer-to-peer network and uses cryptographic techniques to ensure its security and validity. Bitcoin has attracted significant attention from researchers, developers, investors, regulators, and the general public. However, Bitcoin also involves many complex concepts and processes that need to be modeled for better understanding and development.

Modeling is a technique that is used in many fields to share ideas, reduce complexity, align different viewpoints, and provide abstractions of a system at some level of precision and detail. Modeling is also a prerequisite for developing blockchain-specific software engineering best practices [1] and modeling profiles, as well as relevant methodologies and reference architectures [2]. Models are used in model-driven engineering (MDE), which is an approach that aims to provide feedback on model's correctness prior to development [3], help reduce complexity, focus on software development goals, and leave other aspects aside. In addition, Domain-specific languages (DSLs) are languages that provide domain-specific primitives which not only ease model development

http://dx.doi.org/10.22133/ijwr.2024.422357.1191

Citation: B. Sefid-Dashti, J. Salimi Sartakhti, H. Daghigh, " BitML: A UML Profile for Bitcoin Blockchain", *International Journal of Web Research*, vol.6, no.2,pp.1-18, 2023, doi: http://dx.doi.org/10.22133/ijwr.2024.422357.1191.

*Coressponding Author

Article History: Received:25 June 2023; Revised: 22 October 2023; Accepted: 29 October 2023

but also present intuitive syntax for domain experts, and the possibility of code generation for narrow domains [4].

Moreover, complementary models might be used to provide important insights into some complex phenomena. Unified Modeling Language (UML) is a graphical language standardized by the Object Management Group (OMG) for modeling software-intensive systems, and is executable, at least in part [5]. While different UML diagrams are complementary and appropriate for different aspects of software systems, UML provides extensibility mechanisms to define meta-models known as UML profiles [6-7]. A profile is a lightweight mechanism to extend the UML standard [6] and defines domain-specific or platform-specific elements, connectors, and diagrams which aim to facilitate the modeling of applications for people interested in that domain or platform.

Various studies have been carried out on blockchain modeling, but few of them have focused on Bitcoin-specific modeling. For example, Bartoletti and Zunino [8] formalized the bitcoin contract by defining a calculus; their work introduces formalism to validate defined contracts but its usage requires prior knowledge of formal methods and underlying mathematical constructs. Moreover, in contrast to our work, their work concentrates on validation instead of abstraction and simplification that can foster software development in the bitcoin application domain. Rocha and Ducasse [9] used a UML class diagram to model smart contracts; they used a class with "chain" icon to represent the smart contracts, but they used neither stereotyping (i.e. stereotypes, tagged values, and constraints) nor comprehensive combinations of connectors (e.g., dependencies, composition, and aggregation). Bollen [10] applied fact-based modeling to provide conceptual model of Hyperledger Fabric using fact definition and rule validation, but his model requires a significant cognitive load to cross through annotated diagrams and the provided tables. The integration and orchestration [11] of blockchain-specific services with other organizational services have also been studied [12]. Vingerhouts et al. [12] used i* modeling notation and UML Use Case and Sequence diagrams for requirement engineering and modeled contract interactions in lieu of detailed design. Despite these attempts at blockchain modeling, most of them are modeling instead of meta-modeling and to the best of our knowledge, there is no bitcoin-specific meta-model supported by tools, and this study is the first one to have aimed at assisting application development using profiling.

The motivation for this study stems from the observation that Bitcoin, as a prominent and pioneering application of blockchain technology, involves many complex concepts and processes that need to be modeled for better understanding and development. However, there is a lack of adequate and effective modeling tools and techniques that can capture the essence and specifics of Bitcoin applications. Existing studies on blockchain modeling have mostly focused on general aspects of blockchain or other platforms, such as smart contracts, Hyperledger Fabric, or Ethereum. Moreover, most of these studies are modeling instead of meta-modeling, and do not provide tool support or automated conformance checking. Therefore, there is a need for a domain-specific and platform-specific meta-model that can facilitate the modeling of Bitcoin applications, and that can be implemented in a widely used modeling tool, such as Sparx EA.

This study aims to build on extant research to design and evaluate a new viable meta-model for Bitcoin applications. Hence, in this study, we build on UML, OCL, and bitcoin's ontology to propose a UML profile which aligns models with bitcoin's ontology and automates evaluation of conformance. The former is achieved by defining relevant stereotypes, tagged values, and enumeration inside our profile, while the latter is achieved by defining relevant OCL constraints. Our profile has been implemented in the Sparx EA modeling tool.

We believe that our study contributes to the advancement of knowledge and practice in the field of blockchain and Bitcoin modeling, by providing a novel and useful meta-model that can assist developers and researchers in creating and analyzing Bitcoin applications. We also hope that our study inspires further research on the topic, such as extending the profile to cover other aspects of Bitcoin, or applying the profile to other blockchain platforms or applications.

The remainder of this paper has been structured as follows: Section 2 briefly introduces some concepts upon which our approach is built. Our proposed UML profile is presented in Section 3. In the next section, a case study investigated and modeled by the proposed UML profile is presented. Section 5 elaborates the conducted evaluation. Finally, the paper is summed up in Section 6, and conclusions and future works are discussed.

2. Background

2.1. Bitcoin and Blockchain

Blockchain is a distributed ledger that does not rely on a central node of control. It records cryptographically signed, irrevocable, and auditable events (i.e. transactions) that are shared across participants who can independently store, verify, and audit information over a peer-to-peer network. `and each block contains some transactions and uses

a hash value [13] to refer to a previous block. This makes blockchain traceable and transparent. When the majority of participants agree and append a block of transactions to their local copy of blockchain, the content of that block becomes immutable [14].

There are different types of blockchain, depending on the level of access and control. A public blockchain allows anyone to join, and participants are anonymous with equal rights to access and validate transactions. A private blockchain is controlled by a single organization that decides on membership and assigns the roles that each node can play in the blockchain. A consortium blockchain, also known as federated blockchain, is controlled by a group of organizations that collaborate to find solutions. A hybrid blockchain is a combination of a private and a public blockchain. It is controlled by a single organization, and transactions are made and stored privately, but they can be made public and verified by public blockchain members. Finally, emerging fourth and fifth generation blockchains use microprocessors, mobile devices, and Artificial Intelligence (AI) to improve security and scalability, and assist mining by features such as AI based consensus algorithms. Bitcoin is powered by a public blockchain. The rest of this sub-section explains the bitcoin blockchain in more detail.

Bitcoin is secured by mining through significant computational efforts and consensus through the commitment of the majority of blockchain nodes. Mining is a process that enhances the security of bitcoin by introducing the computational effort required for adding a new block to the blockchain [15]. Miners are rewarded with new coins generated in each new block, and they receive a transaction fee from each transaction included in that new block.

To achieve these rewards, miners have to compute a cryptographic hash function several times, until they find a block hash value that is smaller than a predefined threshold called Target Value. The resulting solution is called hashcash, which is a Proof-of-Work (PoW) system invented by Adam Back in 1997 [16] that ensures that issued coins are backed by significant computational efforts. The computed hash and all contained transactions are evaluated by all nodes of the bitcoin network, and each node adds the confirmed block to its local copy of the blockchain. When the majority of nodes participating in the bitcoin network add the new block to their local copy of blockchain, consensus is achieved. Miners use Nonce to compute hash values. Nonce is a random 32-bit (4-byte) number stored in a bitcoin block header to achieve a block hash that is smaller than Target Value (i.e. a hash value that has enough leading zeros). Target Value is used to adjust mining difficulty. In addition to the 4-byte nonce, miners can use 8 bytes of the coinbase transaction as an extra nonce field [15].

There are different types of mining. A miner may perform mining operations with no help (i.e. solo mining), use cloud resources (i.e. cloud mining), or join a mining pool composed of a large number of miners (i.e. pooled mining). These types of mining use different protocols that are included in our proposed meta-model. Bitcoin blockchain stores financial transactions and defines a coin as a chain of digital signatures [17]. Each transaction encodes the transfer of money between participants and includes at least one input, at least one output, and a transaction fee. An exception in this case is the first transaction of each block, which is a special transaction that generates a new coin and is called coinbase transaction [15].

A transaction output specifies the number of Satoshis to be transferred and provides a locking-script that indicates the next owner of the coin(s). A transaction output can be locked by any equation defined by the Script language of bitcoin [15], but most of them (i.e. Pay-to-Public-Key (P2PK), Pay-to-Public-Key-Hash (P2PKH), Multi-Signature (multisig), and Pay-to-Script-Hash (P2SH) [15]) use an Elliptic Curve Digital Signature Algorithm (ECDSA) as proof of ownership [18]. A transaction input consists of two fields: the address of an Unspent Transaction Output (UTXO) and an unlocking-script, which is checked against the locking-script of the referred transaction output. A transaction fee is collected by mining nodes. The fee is not explicitly stated in a transaction and is defined as the difference between the sum of transaction inputs and the sum of transaction outputs [15].

2.2. UML Profiling

Different notations and meta-modeling frameworks [19, 20, 21] and extension of these meta-models [4, 7, 22, 23, 24] exist. Meta Object Facility (MOF) [20] is a well-accepted framework developed by OMG, and approaches model development in multiple levels of abstraction (e.g., a four-layered metamodel architecture including meta-meta-models, meta-models, models, and user objects). Subsequent layers allow navigation from an instance to its meta-object (its classifier) and vice versa. UML itself has been defined by MOF. Profiles are lightweight mechanisms to extend the UML standard [6]. This section describes UML profiling and related concepts which are used to define our proposed profile for bitcoin.

A profile is a mechanism for customizing UML to meet the needs of a particular context. A profile can be defined for a platform that is being targeted (e.g., Java EE or .NET) or a domain with which one is working (e.g., financial or blockchain) [25]. UML

profiles define a concise dialect of UML for a specific family of applications (e.g., UML profile for wireless sensors [7], electronic and electrical waste [22], big data [26], aerospace systems safety [27], hazard mitigation [23], and publish/subscribe paradigm [24]) and are composed of Stereotypes, tags, and constraints [28].

Stereotypes explicitly specify that an element has a special intent or role in a model [25, 28]. A stereotype is shown using guillemots at either end of the stereotype name, as in «stereotype_name». However, they can be substituted by angle brackets, as in <<stereotype_name>>. In Sparx EA, stereotype definition and implementation are denoted by specifying the name of the stereotype between and before two angle brackets, as in <<stereotype_name>> and stereotype_name <<>>, respectively.

Tagged values define information needed by a stereotype to perform its responsibilities [28]. They are meta-attributes which show some properties of model elements such as stereotypes [6]. To encourage reuse, UML 2.0 restricted declaration of stereotypes and tagged values to UML profiles [25].

Finally, constraints restrict model elements. They are defined in the profile but evaluated in the model. Constraints are defined by OCL, a formal yet easy-to-understand expression language for specifying constraints, which allows values to be checked but not changed [25]. OCL is helpful in creating the metamodel of a language, which is a description of all the concepts that can be used in that language and includes all meta-classes of that language and the relationships between them [29].

Figures 1 and 2 show an example of stereotype definition and usage, respectively. In the first figure (Figure 1), an extension arrow with a solid arrowhead pointing from ExampleStereotype stereotype to Class meta-class depicts that the ExampleStereotype stereotype which is tagged with ExampleTaggedValue tagged value can be applied to classes. Figure 2 depicts a stereotyped class containing an attribute, an operation, and two tagged values. This example also includes an OCL constraint. The mentioned OCL invariant states that the firstAttribute attribute of ExampleClass class must be greater than zero.

In this paper, the stereotypes, tags, and constraints are represented by UML standard notation, and the proposed UML profile was implemented using Model Driven Generation (MDG) Technologies [30] feature of Sparx EA modeling tool. We defined OCL constraints on our proposed metamodel to allow greater integrity of application models.

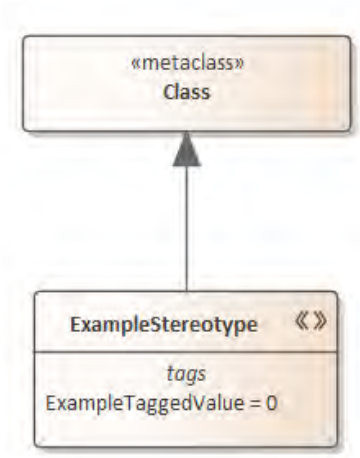


Figure. 1. Example of stereotype definition

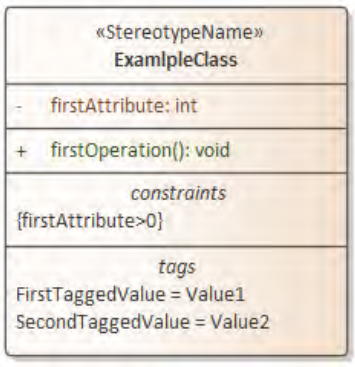


Figure. 2. Example of stereotype usage

3. Proposed Meta-Model

This section describes our UML profile and its implementation in Sparx EA modeling tool. The proposed profile is called Bitcoin Modeling (BitML) and is defined using UML notation. This notation is common for profile definition with some exceptions that are considered irrelevant.

3.1. Profiling Definition

BitML includes two types of diagrams, i.e. a Transaction Processing diagram and a Network diagram. As profiles extend UML, our proposed diagrams are intended to be used along with UML standard diagrams. For example, BitML stereotypes can be used in both a Transaction Processing diagram and a UML sequence diagram to show both application structure and its runtime behavior.

A Transaction Processing diagram is defined as an extension of UML class diagram and provides stereotypes, tagged values, enumerations, and constraints to model both on-chain and off-chain transactions in a way consistent with bitcoin

ontology. Network diagram is defined as an extension of the UML deployment diagram and provides stereotypes, tagged values, enumerations, and constraints to model different bitcoin node types and protocols. Figures 3 and 4 depict profile elements (i.e. stereotypes, tagged values, and enumerations) and an exemplar set of constraints on the proposed UML profile, respectively. Furthermore, Figures 5 and 6 depict how the proposed connectors (i.e. Spend connector, Uplock connector, PBKDF2KeyStretching connector, HMAC-SHA512 connector, and RIPEMD160HashOfSHA256Hash connector) are defined and constrained, respectively.

As shown in Figures 3 and 5, the profile is composed of 42 stereotypes (16 for classes, 14 for attributes, eight for operations, and five for connectors), 23 tagged values, six datatypes, and a set of constraints. Proposed meta-model includes the following enumerations:

ScriptType: This enumeration defines five values.

1. Pay-to-Public-Key (P2PK)
2. Pay-to-Public-Key-Hash (P2PKH)
3. Pay-to-Script-Hash (P2SH)
4. Pay-to-Witness-Public-Key-Hash (P2WPKH)
5. CustomScript.

The first four values correspond to the most common script types in bitcoin, while the last value reflects the custom script development capability [15] which made bitcoin an extendable and programmable form of currency.

TransactionPosition: A transaction may be either on-chain or off-chain, which will be executed on or out of the blockchain, respectively. An off-chain transaction will be executed outside of the blockchain, but its execution is bonded to some blockchain data, and its results will be saved on the blockchain too. This enumeration includes two values for on-chain and off-chain transactions.

PayToScriptHashType: Regarding Pay-to-Script functionality, a mining node may support either Pay-to-Script-Hash (P2SH) or CheckHashVerify (CHV). P2SH or CHV correspond to BIP-16 or BIP-17, respectively.

CommunicationProtocol: Bitcoin nodes communicate over the Bitcoin P2P protocol. In addition, some miners and mobile wallets communicate over the Stratum protocol. Finally, a pool miner may communicate over a specialized mining pool protocol. This enumeration defines these three values.

HashCashFucnctionType **and** ***HashCashVerificationFunctionType***:

These enumerations correspond to Hashcash. As mentioned, Hashcash is a PoW system. Depending on the hash function that is used, there are three Hashcash variants (i.e. SHA-1, scrypt hash function, and double SHA-256). Bitcoin uses Hashcash with double SHA256 hash.

Depicted in Figure 3, our profile includes 42 stereotypes, 16 of which (*BitcoinNode*, *Block*, *BlockHeader*, *Transaction*, *TransactionOutput*, *AbstractTransactionInput*, *TransactionInput*, *CoinbaseTransactionInput*, *LockingScript*, *UnlockingScript*, *EllipticCurveSignature*, *MnemonicCodeWord*, *Seed*, *PrivateKey*, *PublicKey*, and *PublicAddress*) extend UML class. Two types of transaction inputs are abstracted by the *AbstractTransactionInput* abstract class. The *UML Composition* relationship between *AbstractTransactionInput* and *Transaction* stereotypes indicates that each transaction requires at least one transaction input. Depicted by *UML Generalization*, *TransactionInput* and *CoinbaseTransactionInput* stereotypes are specializations of *AbstractTransactionInput* stereotype. The former represents a normal transaction input which includes three attributes that point to a UTXO, which will be unlocked by an instance of a class, which is stereotyped as *UnlockingScript*. The latter represents a coinbase input. Coinbase is the input of coinbase transaction which is the first transaction of each block and generates new coins as mining rewards. The *MinerPay2ScriptHashStandard* tagged value of the *CoinbaseTransactionInput* stereotype uses *PayToScriptHashType* enumeration to specify either P2SH or CHV, which is supported by the miner. Coinbase transaction input stores a block height and possibly an extra-nonce value which correspond to *BlockHeight* and *ExtraNonce* stereotyped attributes. Furthermore, the *Spend* connector, which is defined in Figure 5 and constrained in Figure 6, applies to instances of classes stereotyped as *TransactionOutput*, *AbstractTransactionInput*, *TransactionInput*, and *CoinbaseTransactionInput*, showing that a transaction output may spend either a coinbase transaction or a normal transaction. As Figure 6 shows, a tool-level constraint is defined to constrain classes stereotyped as *AbstractTransactionInput* and *TransactionOutput* as the allowed source and destination of the *Spend* connector, respectively. Hence, the *Spend* relationship can be drawn solely from classes stereotyped as *AbstractTransactionInput*, *TransactionInput*, or *CoinbaseTransactionInput* to classes stereotyped as *TransactionOutput*. In the same way, a constraint is defined to constrain classes stereotyped as *UnlockingScript* and *LockingScript* as the allowed source and destination of the *Unlock* relationship, respectively. In the same

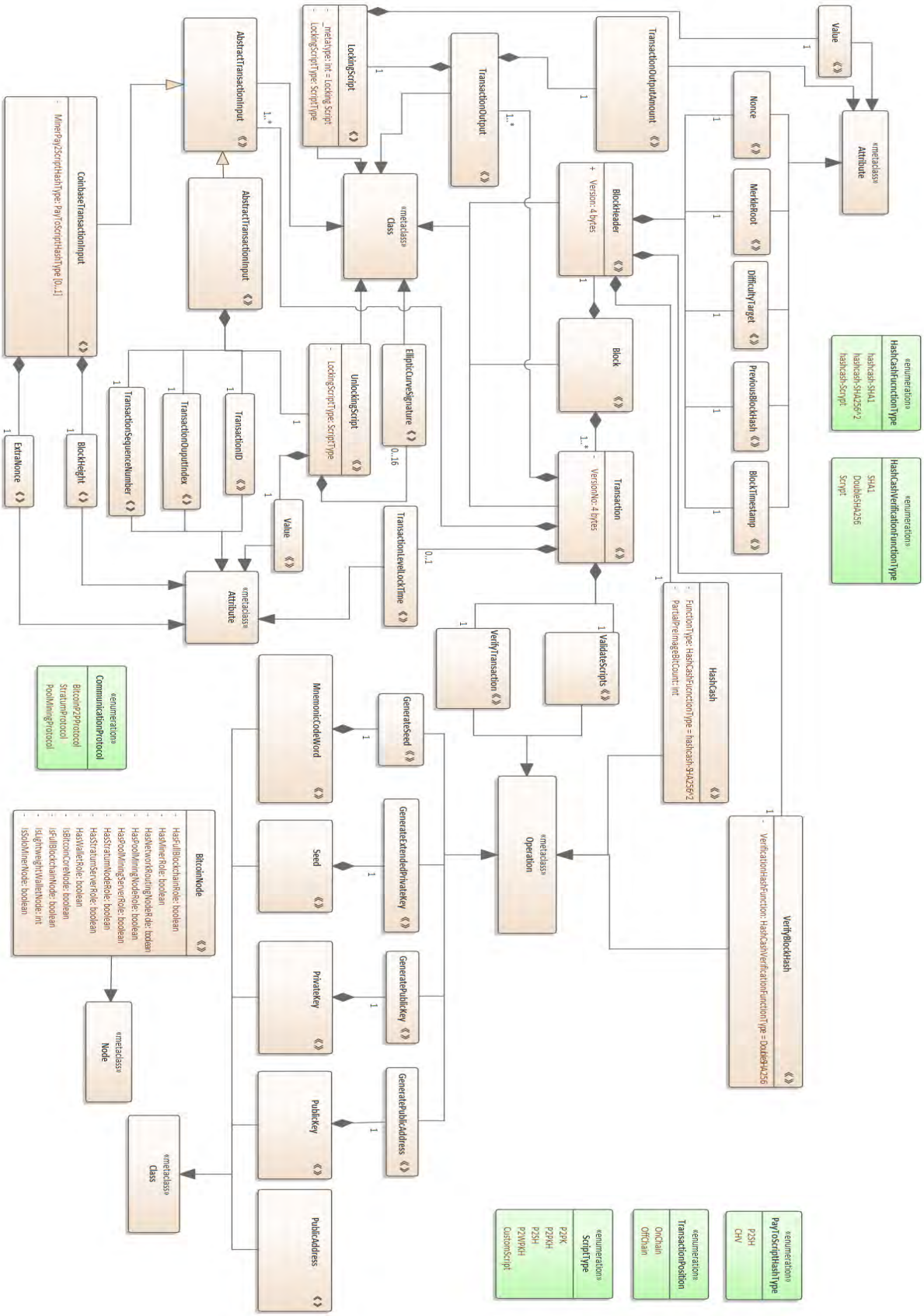
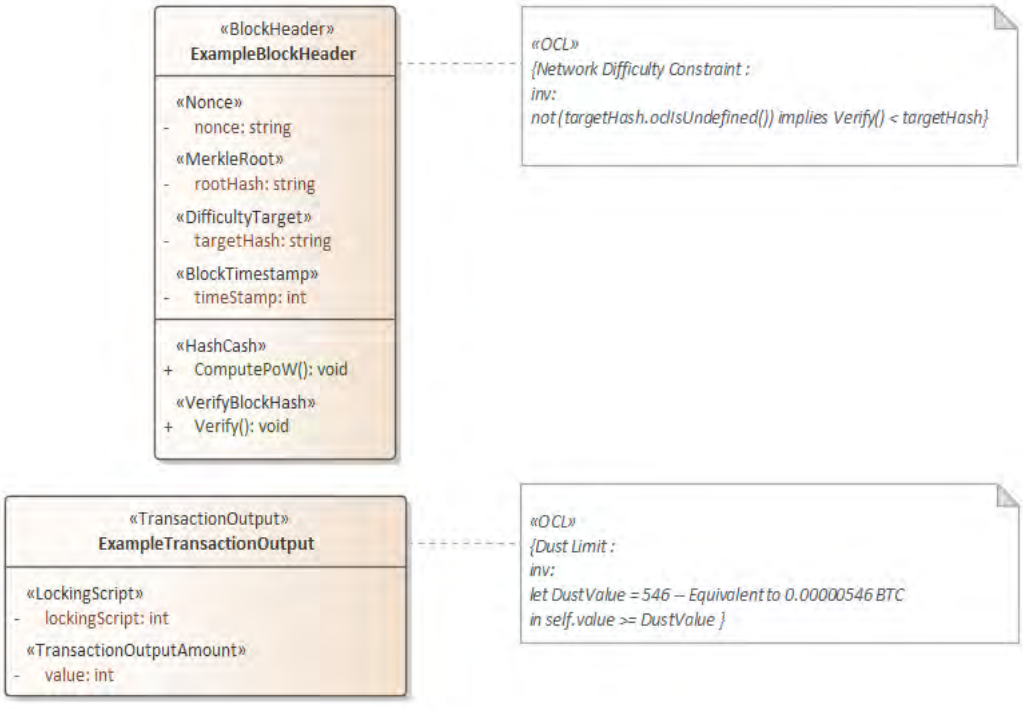
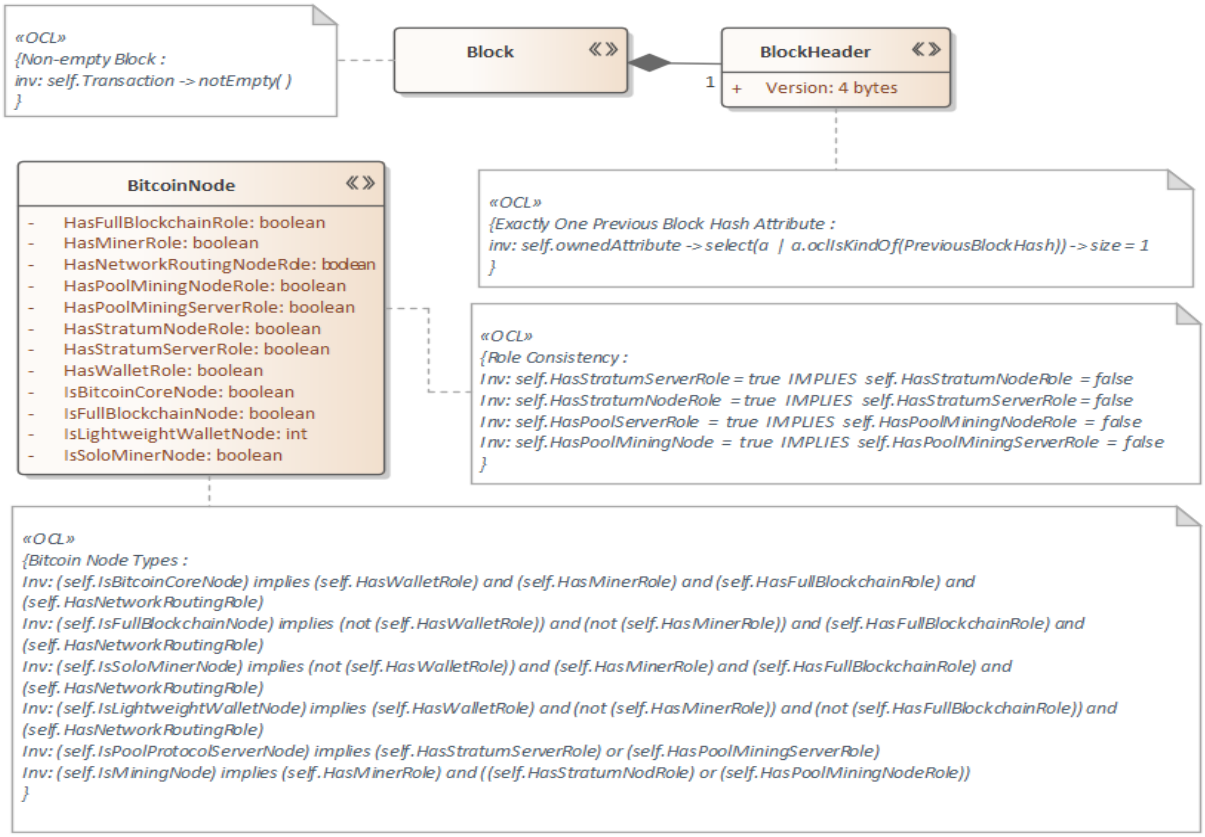


Figure 3. Proposed UML profile (excluding connectors' meta-model



(a)



(b)

Figure. 4. (a) Example metamodel level OCL constraints; (b) Example application level constraints

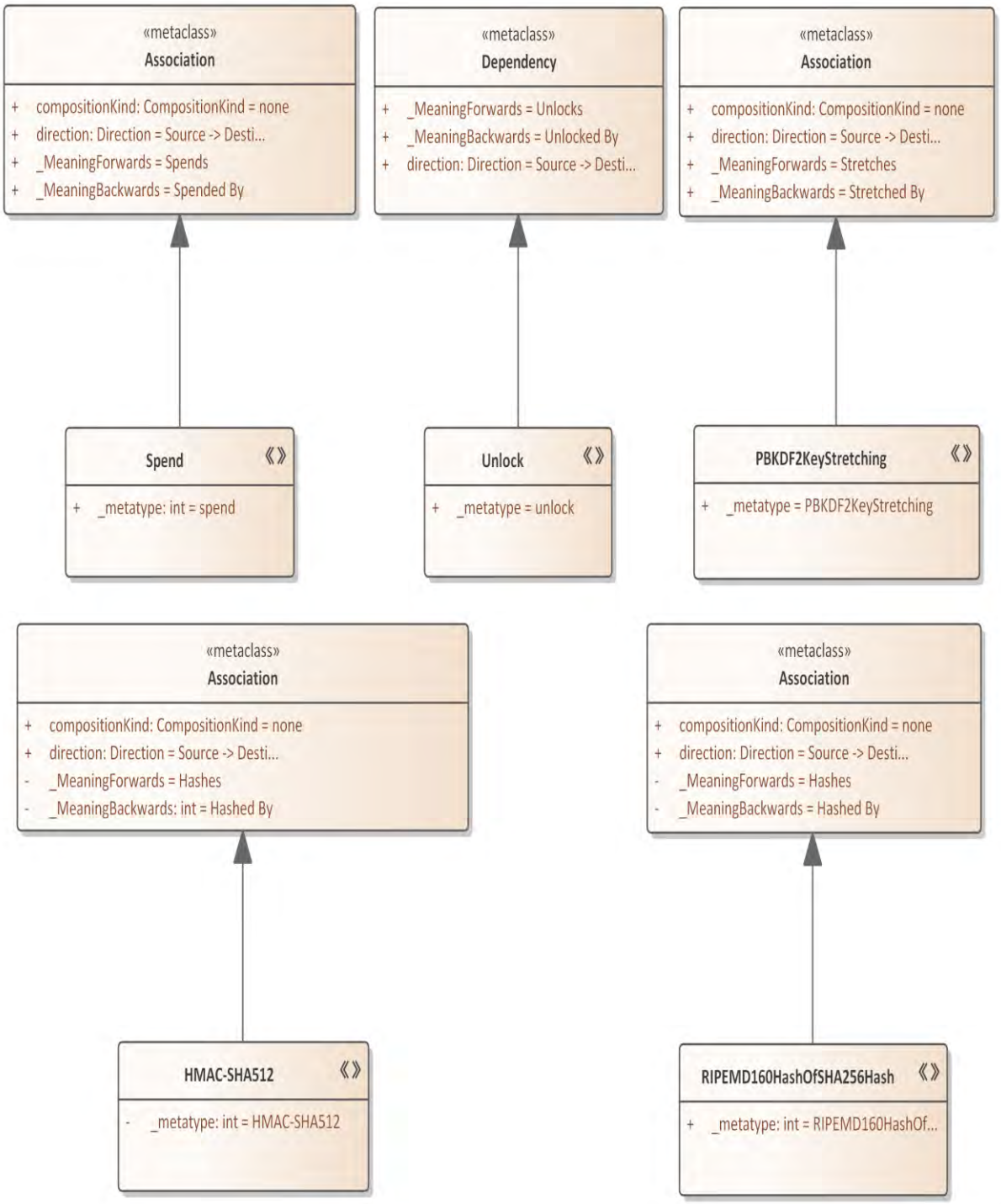


Figure. 5. Connector definition meta-model

manner, the *Unlock* relationship can be drawn solely from classes stereotyped as *UnlockingScript* to classes stereotyped as *LockingScript*. As Figure 5 depicts, the *Spend* connector extends the *UML Association* and shows that transaction inputs will provide proof of ownership (i.e. unlocking script) of the referenced UTXOs. The *Unlock* relationship extends the *UML Dependency* and is self-explanatory. In the same way, the remaining connectors (i.e. *PBKDF2KeyStretching*, *HMAC-SHA512*, and *RIPEMD160HashOfSHA256Hash* connectors) are defined in Figure 5 and constrained in Figure 6 and are used for bitcoin key management capabilities that may be used by a wallet or an

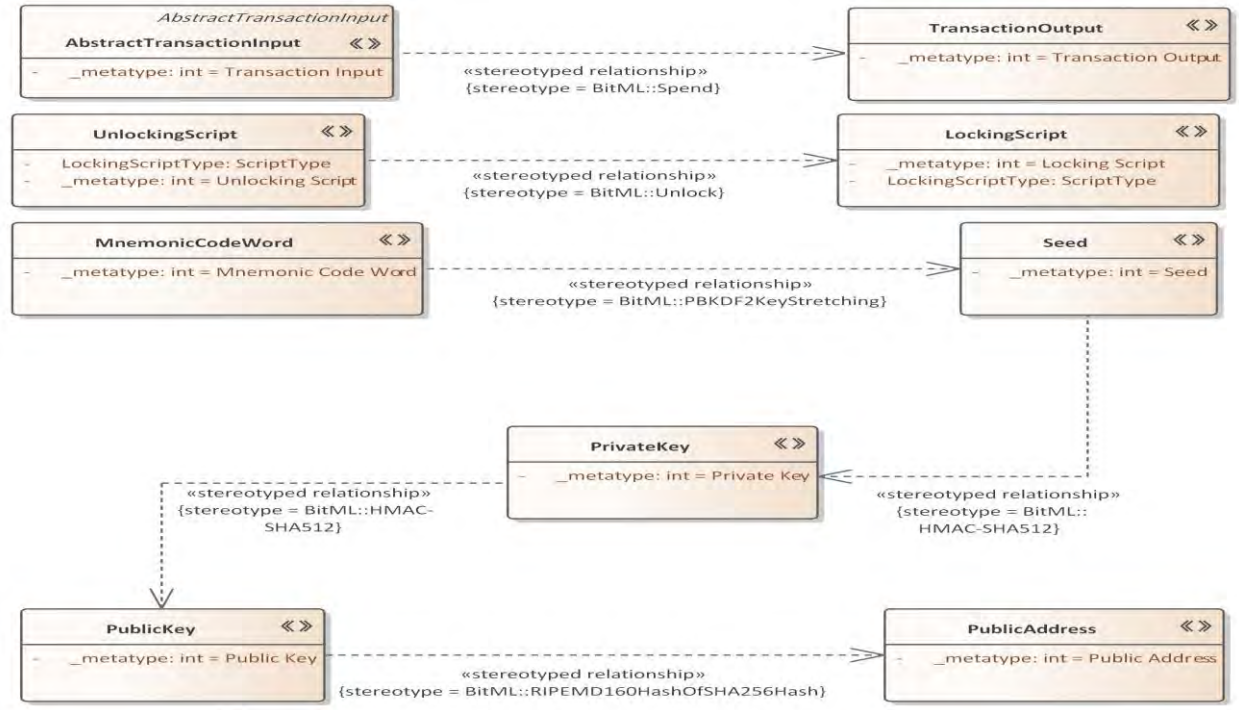


Figure. 6. Connector usage constraint definition

exchange to access bitcoin blockchain and/or generate new transactions (here wallet term is used to refer to Hierarchical Deterministic (HD) wallets as the most common type of bitcoin wallets). The *PBKDF2KeyStretching* connector can be used to model generating *seeds* from *mnemonic code word*s that are presented by *Seed* and *MnemonicCodeWord* stereotypes. Bitcoin uses 12-to-24-word mnemonic phrases. A wallet may use a mnemonic code word to generate a seed, and user private keys will be derived from that seed. The *HMAC-SHA512* connector models HMAC function which uses SHA512 hash function and its usage in bitcoin wallets is twofold: to derive a private key from a seed and a child private/public key from a parent private/public key. Finally, the *RIPEMD160HashOfSHA256Hash* connector models the double hashing process used by Bitcoin to generate public addresses from public keys. Initially, the SHA256 hash of the public key is computed, and the result is then hashed using the RIPEMD160 hash function. This double hashing is represented by the *RIPEMD160HashOfSHA256Hash* connector and is employed to calculate a public address: PublicAddress = Ripmed160(SHA256(PublicKey)).

MnemonicCodeWord, *Seed*, *PrivateKey*, *PublicKey*, and *PublicAddress* stereotypes along with the last three described connectors (i.e. *PBKDF2KeyStretching*, *HMAC-SHA512*, and *RIPEMD160HashOfSHA256Hash* connectors), provide key management capabilities from which applications such as wallets and exchanges may benefit.

In the bitcoin network, nodes play different roles, including *wallet*, *miner*, *full blockchain*, *network routing*, *stratum node*, *stratum server*, and *pool server* [15]. Our profile introduces a *BitcoinNode* stereotype, 12 tagged values, and a *CommunicationProtocol* enumeration which may be used in *BitML Network* diagrams to model application deployment and communication over the bitcoin network. These roles are specified as tagged values for the *BitcoinNode* stereotype, representing distinct functionalities. However, not all combinations of these roles are allowed. For example, no nodes are allowed to play *stratum node* and *stratum server* roles at the same time as defined by OCL constraint presented in Fig 4 (a). Furthermore, common combinations of the mentioned roles are defined as node types. For example, a node that serves as a wallet, miner, maintains the full blockchain, and handles network routing is commonly referred to as a *bitcoin core* node, or a *reference client* node. As another example, a node that fulfills both wallet and network routing roles is termed a *lightweight wallet* node. The corresponding OCL constraints are illustrated in Figure 4-a. In addition to *BitcoinNode* constraints, two other meta-model level constraints are provided in Figure 4-a. The first avoids empty block creation, and the second mandates the use of exactly one previous block hash attribute. Finally, application-

level constraints may be utilized, as illustrated by two examples in Figure 4-b. The first constraint ensures that the block hash of each valid block is smaller than the network's difficulty target value, while the second enforces a dust limit of 546 Satoshis.

3.2. Implementation

The proposed profile has been implemented in the Sparx EA modeling tool. Figure 7 shows an example snippet of XML generated by *Sparx MDG technology* for the defined UML profile. It shows that the *Unlock* relationship and the *UnlockingScript* stereotype extend the *UML Dependency* and *Class* meta-class, respectively. Furthermore, it demonstrates the applicability of the *Unlock* relationship on *UnlockingScript* stereotype and the definition of the *LockingScriptType* tagged value and its possible values.

As previously mentioned and illustrated in Figure 8, the *Transaction Processing* diagram extends the *Logical* diagram (i.e., *UML class* diagram), while the *Network* diagram extends the *Deployment* diagram. Figures 9 and 10 showcase the definition and presentation of the toolbox designed to provide access to BitML class stereotypes, BitML enumerations, and BitML connectors.

In this context, a class stereotype refers to a stereotype that extends the *Class* meta-class. Each element (i.e., class stereotype, enumeration, or connectors) is linked to the *BitML toolbox* by defining a tagged value for the extended toolbox. As depicted in Figure 8, the *BitML toolbox* is associated with the *Transaction Processing* diagram and *Network* diagram through the *toolbox* attribute of the *Diagram_Logical* and *Diagram_Deployment* meta-classes, respectively.

The appearance of the defined toolbox shown in Figure 10 aims to ease access to defined stereotypes, enumerations, and connectors. Users have the option to utilize the provided toolbox interchangeably or directly apply stereotypes to the base elements (i.e., class or node).

It's worth noting that attribute/operation stereotypes have not been directly bound to the toolbox because they are not utilized directly; instead, they are accessed through their associated class stereotypes. In this context, attribute and operation stereotypes refer to stereotypes that extend the *Attribute* and *Operation* meta-classes, respectively.

4. Case Study

To validate its practical utility and efficacy of proposed UML profile, we turn to a compelling case study: the development of a fast payment system powered by Bitcoin's robust infrastructure. This study transcends mere demonstration; it acts as a rigorous testing ground, revealing the strengths and potential limitations of our profile in a real-world context.

Through the lens of meticulously crafted diagrams, we delve into the heart of the fast payment system. These diagrams, crafted using our UML profile, serve as a testament to its capabilities. They reveal:

- Precise representation of Bitcoin transactions: Witness how our profile seamlessly captures the intricate dance of data within the Bitcoin network, from transaction initiation to confirmation.
- Clear communication between system components: Observe how our profile facilitates efficient interaction between diverse system elements, ensuring smooth processing and rapid transaction finalization.
- Enhanced developer clarity and understanding: Designed to empower developers with a profound understanding of the system's inner workings.

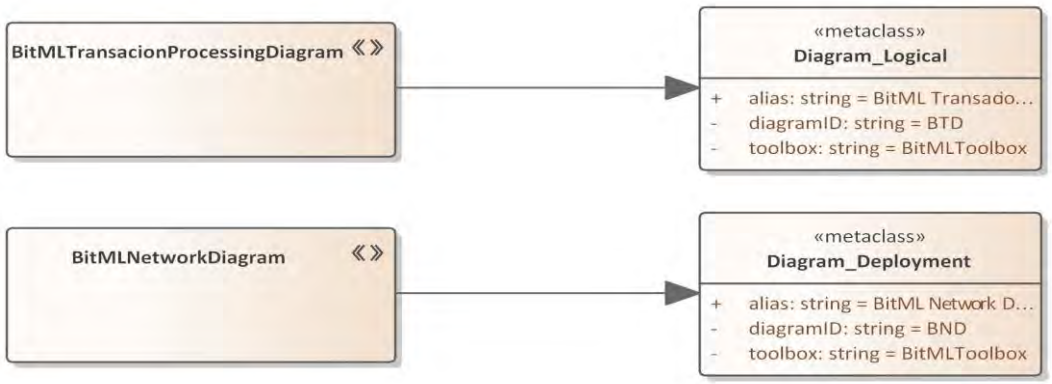
By dissecting these diagrams, we not only showcased the practical application of our profile but also embarked on a critical appraisal of its effectiveness. We meticulously analyzed its ability to:

- Handle the nuances of fast payment protocols: Discover how our profile adapts to the unique demands of high-speed transactions within the Bitcoin network.
- Maintain transparency and traceability: Explore how our profile ensures clarity and integrity throughout the transaction lifecycle, fostering trust and security within the system.
- Identify potential areas for improvement: Through this rigorous analysis, we uncover valuable insights that can further refine and strengthen our profile, paving the way for even more powerful applications in the future.

Let's delve into the case study: Since a secure bitcoin payment requires at least 6 block confirmations, bitcoin is not suitable for payments that rely on quick transaction confirmation. Bitcoin fast payment algorithms [31, 32] aim to expedite the payment process while mitigating the risk of double spending. A sub-set of these algorithms facilitate a large number of off-chain transactions recorded by a small number of on-chain transactions. Therefore, the challenge is to either reuse, combine, or devise a new blockchain consensus algorithm to expedite

```
<Stereotype name="Unlock" metatype="unlock" notes="" cx="0" cy="0" bgcolor="-1" fontcolor="-1" bordercolor="-1"
          borderwidth="-1" hideicon="0">
    <AppliesTo>
        <Apply type="Dependency">
            <Property name="_MeaningForwards" value="Unlocks"/>
            <Property name="_MeaningBackwards" value="Unlocked By"/>
            <Property name="direction" value="Source -&gt; Destination"/>
        </Apply>
    </AppliesTo>
</Stereotype>
<Stereotype name="UnlockingScript" alias="Unlocking Script" metatype="Unlocking Script" notes="" cx="0" cy="0"
         bgcolor="-1" fontcolor="-1" bordercolor="-1" borderwidth="1" hideicon="0">
    <stereotypedrelationships>
        <stereotypedrelationship stereotype="BitML::Unlock" constraint="BitML::LockingScript"/>
    </stereotypedrelationships>
    <AppliesTo>
        <Apply type="Class">
            <Property name="isActive" value=""/>
            <Property name="_HideUmlLinks" value="false"/>
        </Apply>
    </AppliesTo>
    <TaggedValues>
        <Tag name="LockingScriptType" type="enumeration" description="" unit="" values="P2PK,P2PKH,P2SH,P2WPKH" default=""/>
    </TaggedValues>
</Stereotype>
```

Figure. 7. An example of generated XML snippets



Figur 8. Diagram definition meta-model

primary blockchain payments. In a companion research project, we designed a fast payment application, implemented it using the NBitcoin library[1] and tested it on the Bitcoin Testnet[2]. Presented UML profile has been applied to model this application. This section presents one of these models along with defined constraints to illustrate how BitML can be used to model applications belonging to bitcoin application domain.

To this end, we utilized a bond transaction to create a specific UTXO, which can be employed for off-chain payments and is resistant to double spending. A refund transaction, spending that UTXO in its entirety after a specified period, was defined with a transaction-level lock time. Additionally, the fast payment application has the capability to generate settlement transactions, facilitating the transfer of funds from the fast payment account to merchants' accounts. A settlement transaction can yield one or two outputs: one for transferring bitcoins to the intended recipient and optionally, another output for returning the remaining bitcoins in change back to the user's wallet. Each settlement transaction spends the UTXO of either a bond transaction or a change-back settlement transaction.

We modeled this application using the UML profile proposed in this paper. Our algorithms were depicted using *BitML Transaction Processing* diagrams and corresponding UML sequence diagrams. The on-chain transactions of our fast payment application are shown in Figure 11. In addition to profile-defined constraints, Figure 11 introduces two application-level constraints. The first constraint, titled "Lock Time Validation,"

[1] https://github.com/MetacoSA/NBitcoin
[2] https://en.bitcoin.it/wiki/Testnet

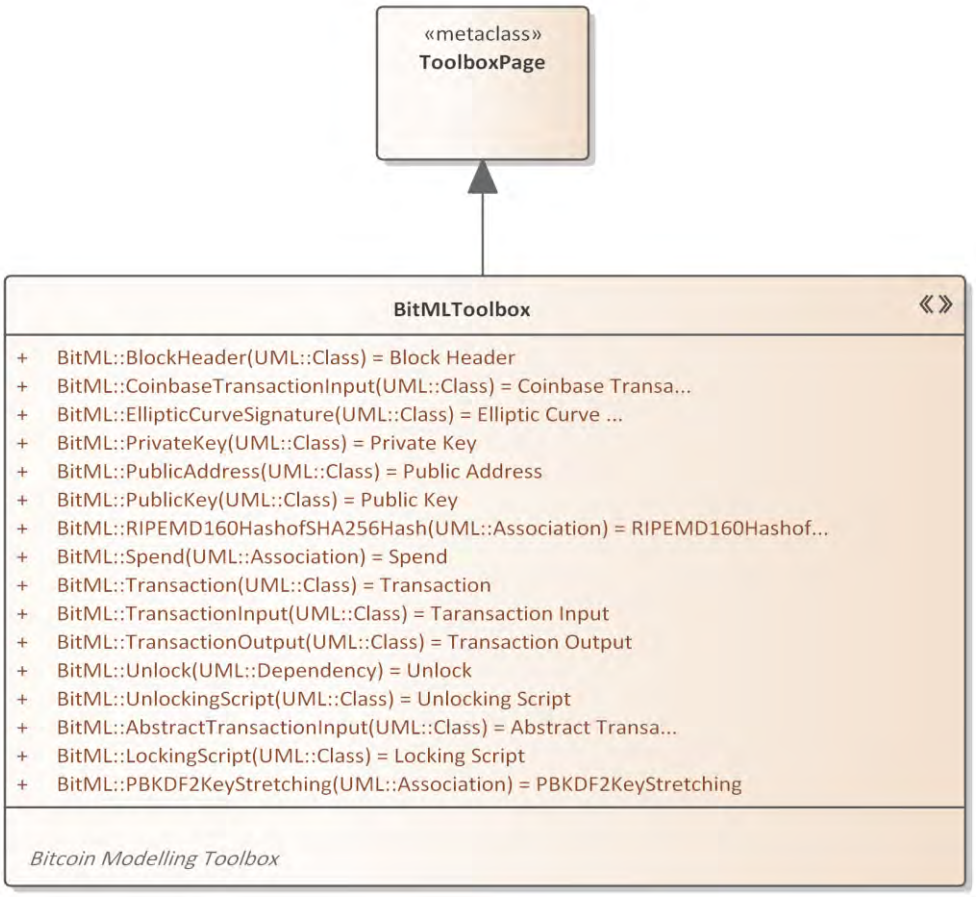


Figure. 9. Toolbox definition meta-model

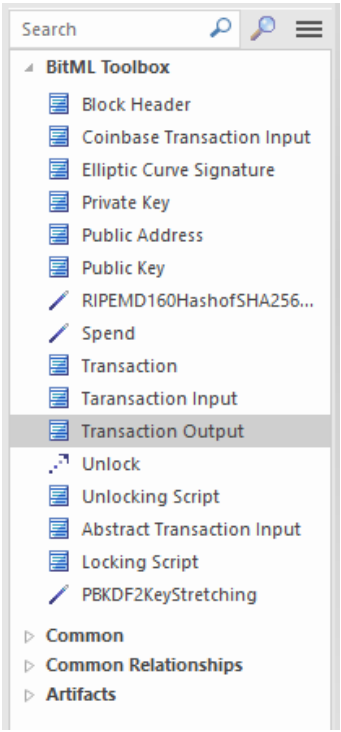


Figure. 10. Layout of the defined toolbox in Sparx EA

specifies that bond transactions must use zero transaction-level lock times, while refund transactions should use non-zero lock times. The second constraint, titled "Subsequent Transaction Constraint," ensures that each settlement transaction spends the UTXO of either a bond transaction or a change-back settlement transaction, and has either a transaction-level lock time of zero or a smaller transaction-level lock time than the lock time of the referenced transaction. This constraint ensures that the referenced UTXO cannot be spent beforehand. This model, exemplifying a BitML Transaction Processing diagram, was employed alongside other models in the analysis and design of the developed bitcoin fast payment application. It played a crucial role in helping us formulate and communicate a mathematically proven and algorithmically sophisticated solution, eliminating any potential ambiguity.

5. Evaluation

We utilized the ATAM framework [33] to assess the proposed UML profile for bitcoin application development. With its emphasis on tradeoffs, scenario-driven analysis, stakeholder collaboration, and a structured process, the ATAM framework

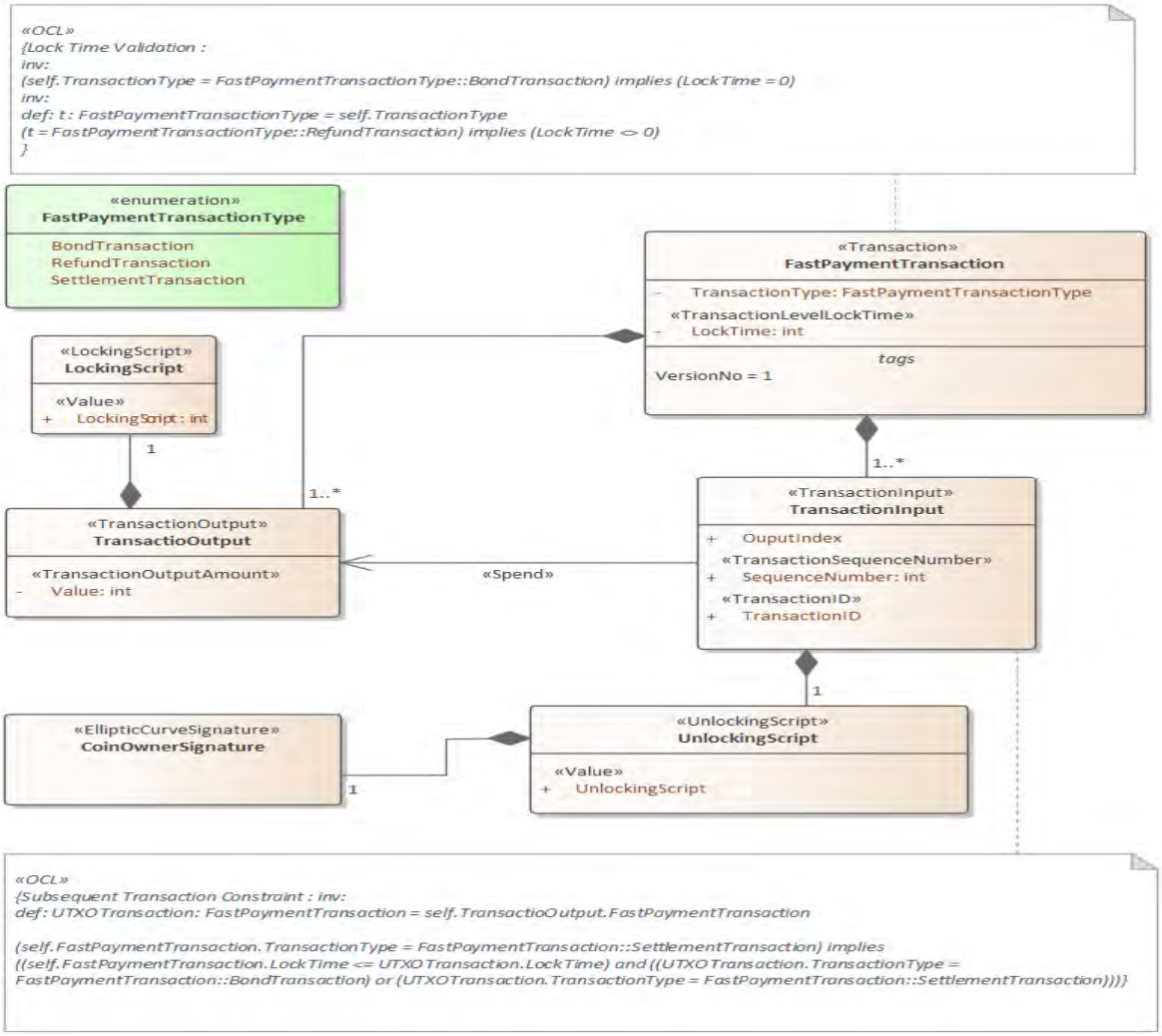


Figure. 11. An Example BitML Transaction Processing diagram for On-chain transactions of a bitcoin fast payment application

aligns well with comprehensiveness, usability, and domain relevance goals of the proposed profile. Three individuals with expertise in software architecture and four in the bitcoin blockchain actively participated in the evaluation. This section aims to provide a thorough assessment of the effectiveness of BitML UML profile for supporting bitcoin application development.

5.1. Architectural Drivers

To ensure our UML profile effectively addresses the multifaceted requirements of bitcoin blockchain modeling, we identified the following list of architectural drivers:

1) Domain Complexity and Conceptual Clarity: Addressing the inherent complexity of bitcoin and ease the development of clear conceptual models.

2) Modeling Precision and Detail: Ensuring precise and detailed representation of classifiers present in the processes and concepts of applications belonging to the targeted application domain.

3) Decentralization and Distributed Systems: Effectively supporting the decentralized nature and distributed architecture of the blockchain.

4) Security and Cryptographic Techniques: Accurately modeling and incorporating bitcoin's reliance on cryptography for both security measures and transaction validation.

5) Software Engineering Best Practices: Aligning with and contributing to blockchain-specific software engineering best practices, methodologies, and reference architectures.

6) Automation in Model-Driven Engineering (MDE): Facilitating automated model

transformations and potential for code generation (in future).

7) Correctness and Validation Prior to Development: Supporting pre-development model validation, in accordance with bitcoin operational principles (through the use of defined OCL constraints).
8) Intuitive Syntax for Domain Experts: Offering intuitive syntax in the form of a domain-specific language.
9) Simplification and Focus on Development Goals: Capable of reducing complexity and concentrating on key development goals by providing clear abstractions within the targeted application domain.
10) Extension and Domain-Specific Modeling: Extending UML to define domain-specific elements, connectors, and diagrams.
11) Tool Compatibility and Integration: Ensuring compatibility and effective implementation in popular modeling tools such as Sparx Enterprise Architect.
12) Alignment with Bitcoin's Ontology and Automated Conformance Evaluation: Aligning models with bitcoin's ontology and automating conformance evaluation using OCL constraints.
13) Practical Applicability and Utility: Demonstrating usefulness in real-world application development contexts.
14) Standardization and Extensibility (UML Profiles): Adhering to UML standards while providing extensibility to address domain-specific needs.

5.2. Utility Tree

We utilized the following quality attributes and sub-attributes to conduct evaluation:

1) Correctness and Consistency: Ensures models accurately represent bitcoin's ontology and maintain consistency in conformance to relevant rules.
    a) Transaction validity:
        i) Conformance to transaction format and signature requirements.
        ii) Accurate representation of valid transaction types and fees.
        iii) Handling of edge cases and invalid transactions.
    b) Compliance with bitcoin protocols:
        i) Alignment with consensus mechanisms and rules.
        ii) Accurate modeling of network communication and message exchanges.
        iii) Adherence to relevant Bitcoin Improvement Proposals (BIPs).
    c) Internal consistency:
        i) Absence of contradictions or inconsistencies within the model.
        ii) Logical relationships between elements and constraints.
        iii) Traceability between model elements and Bitcoin concepts.

To maintain brevity, the remaining quality attributes are described more succinctly.

2) Understandability and Clarity: Models should be clear and easy to comprehend for both domain experts and developers, facilitating communication and collaboration.
    a) Syntax and notation:
        i) Familiarity and ease of use for target audience.
        ii) Conciseness and avoidance of ambiguity.
        iii) Alignment with established UML conventions.
    b) Documentation and examples
    c) Visual representation:
        i) Readability and intuitiveness of diagrams and model elements.
3) Completeness and Expressiveness: Ensures the profile includes sufficient elements and relationships to capture all critical aspects of the targeted application domain.
    a) Coverage of core bitcoin entities
    b) Modeling of decentralized aspects
    c) Extensibility and customization:
        i) Capability to extend the profile with additional elements for domain-specific needs.
4) Tool Integration and Support: Evaluates the compatibility of the profile with existing modeling tools and its effectiveness within those tools.
    a) Feature support:
        i) Availability of tools that fully implement the profile's features and constraints.

ii) Ability to utilize tool-specific functionalities for validation, simulation, or code generation.

b) Usability and performance:

i) Intuitiveness and ease of use of profile features within the chosen tool.

ii) Availability of error messages for validation issues.

Other quality attributes and sub-attributes, such as automated code generation and automated secure code generation, are beyond the scope of this paper and are dedicated to the future work of the authors.

5.3. Analysis

Table 1 presents evaluation scenarios, risks, and countermeasures. This sub-section highlights sensitivity points and tradeoffs identified in the ATAM evaluation. To maintain brevity of this section, descriptions are provided solely for the first sensitivity point and the first trade-off point. Sensitivity points are critical elements with significant impact on quality attributes, while tradeoff points require balancing competing attributes.

Exemplars of identified sensitivity points are provided below:

1) Cryptographic Modeling:
   - Sensitivity: Small changes or inaccuracies in modeling cryptographic processes can significantly impact the security aspect of the UML profile.
   - Impact: A minor error or oversight could lead to a substantial misunderstanding of bitcoin's security mechanisms.
2) Scalability Representation (application scalability)
3) Tool Compatibility (tool update adjustments)

Furthermore, the following are exemplars of tradeoff points identified in the conducted evaluation:

1) Complexity vs. Understandability:
   - Tradeoff: Balancing the need for detailed, accurate modeling of bitcoin processes against the need for the model to be understandable to non-technical users.
   - Decision: Classifier with different level of abstraction are provided in the proposed UML profile.
2) Model Flexibility vs. Standardization (domain-specific adaptations)
3) Security Detailing vs. General Usability (being both accurate and accessible to a broader audience)

In ATAM, pinpointing sensitivity and tradeoff points provides foresight into the ramifications of changes and design choices. This ensures that our UML profile is adaptable and effective for its intended use in blockchain application development.

Regarding the addressed risks, and the identified tradeoffs and sensitivity points, the evaluation showed that the proposed profile is promising. Furthermore, it has led us to implement improvements mentioned in Section 6 to foster further adoption by application development communities.

6. Conclusions and Future Works

Blockchain technology has become a cornerstone in various industries, featuring diverse blockchains such as public, private, consortium, hybrid, fourth generation, and fifth generation. Among these, Bitcoin, powered by its public blockchain, has gained prominence as a highly valued and widely used digital currency, spurring the development of applications on its blockchain. However, the intricacies of bitcoin application development present challenges, demanding specialized software engineering best practices.

In response to these challenges, our paper introduces a UML profile meticulously crafted for the bitcoin blockchain. This profile encompasses *Transaction Processing* and *Network* diagrams, 42 stereotypes, 23 tagged values, six datatypes, and a set of OCL constraints for automatic conformance evaluation. As bitcoin emerged at the convergence of distributed computing and mathematical cryptography, we advocate developing a UML profile as a pragmatic approach. This leverages both existing UML diagramming elements and profile-specific features to adeptly model the intricate nature of bitcoin application development.

The practical implementation of the proposed profile in Sparx EA not only confirms its applicability but also demonstrates its real-world effectiveness. The focal point of our case study was the development and modeling of a bitcoin fast payment application, showcasing the profile's capabilities in a tangible and applied context. Specifically designed for secure and rapid bitcoin payments, this application served as a robust testbed for evaluating the proposed UML profile.

A rigorous evaluation of the UML profile was conducted using the ATAM, providing valuable insights into its strengths and areas for enhancement. This evaluation, focused on the profile itself, offered a comprehensive understanding of its architectural

Table 1. Examples of Evaluation Scenarios, Identified Risks, and Countermeasures

Scenario No.	***1***	***2***	***3***
Scenario Name	Simple Bitcoin Wallet App	Decentralized Exchange (DEX) Platform	Bitcoin Integration with Enterprise System
Stimulus	Developer models a basic bitcoin wallet app with sending/receiving and key management functionalities	Team models a complex DEX platform with smart contracts, liquidity pools, order matching, and so on.	Team integrates bitcoin payment functionalities into an existing ERP system
Environment	- Development environment (chosen tool) - Target users (developers & domain experts)	- Development environment (chosen tool) - Target users (DEX development team) - Security requirements	- Development environment (ERP) - Target users (integration specialists) - ERP data structures and functionalities
Response	-Complete and clear model with sufficient detail for basic transactions . - Intuitive syntax for developers and domain experts. - Efficient implementation within the chosen tool.	Detailed and accurate model of the DEX architecture, including smart contracts, security protocols, and distributed communication - OCL constraints effectively validate model correctness and adherence to DEX principles	- Clear model of the integration with delineated responsibilities and data flows - Simplified abstraction level suitable for integration development and validation. - Demonstrated practical usage in a real-world enterprise scenario.
Architectural Decisions	- Level of detail in profile elements. - Syntax complexity for different user groups. – Tool compatibility and support.	- Profile capabilities for capturing DEX mechanics and smart contracts. - Automation support for DEX-specific aspects. - Validation mechanisms for complex interactions.	- Domain-specific modeling capabilities for bridging bitcoin and ERP systems. - Abstraction level appropriateness for integration context. - Practical utility and effectiveness in real-world scenarios.
Quality Attribute Impact	- Clarity and comprehensibility. - Developer productivity and ease of use. - Tool effectiveness and integration.	- Correctness and consistency. - Development efficiency and automation. - Security and reliability.	- Maintainability and ease of integration. - Development effort and resource optimization. - Practical applicability and value proposition.
Risk Rating	Medium	High	Medium-High
Mitigation Strategies	- Refine profile elements to provide appropriate detail for basic bitcoin concepts. - Offer alternative syntax options for varying user expertise.	- Model the system with the profile. - Enhance code generation capabilities. - Develop OCL constraints to validate complex DEX interactions and security protocols.	- Develop domain-specific model tailored to integration needs. - Provide adjustable abstraction levels to cater to different integration complexities. - Showcase successful case studies of real-world implementations.

tradeoffs, sensitivity points and addressed risks, and highlighted key considerations for further refinement. The results from the ATAM assessment was promising and contribute to the ongoing evolution of the UML profile, ensuring its continuous improvement and adaptability for diverse blockchain applications.

Further research can facilitate side-chain development, and we aim to extend this work to support side-chain modeling. The authors expect that this extension, along with the other mentioned benefits, will increase the number of rational incentives for the wider adoption of the proposed UML profile and will encourage researchers, practitioners, and decision-makers to conduct experiments. In addition, applications such as Colored Coin and fast payment algorithms broaden the use of bitcoin for non-fungible tokens (asset management) [34] and daily shopping [31, 32] respectively. While these applications can be modelled by the proposed meta-model, the proposed meta-model can be extended to define specific classifier to further assist modelling in these sub-domains. Finally, the authors aim to enrich the proposed UML profile by adding model transformation and automated code generation which are two important outcome of MDE.

Declarations

Funding

This research did not receive any grant from funding agencies in the public, commercial, or non-profit sectors.

Authors' contributions

BSD: Study design, software design and implementation, interpretation of the results, drafting the manuscript, revision of the manuscript;

JSS: Study design, supervision, interpretation of the results, drafting the manuscript, revision of the manuscript;

HD: Study design, supervision, interpretation of the results, drafting the manuscript, revision of the manuscript.

Conflict of interest

The authors declare that no conflicts of interest exist.

References

[1] S. Porru, A. Pinna, M. Marchesi, and R. Tonelli, "Blockchain-oriented software engineering: challenges and new directions", In *2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C)*, 2017, pp. 169-171, 10.1109/ICSE-C.2017.142.

[2] B. Sefid-Dashti, and J. Habibi, "A reference architecture for mobile SOA", *Systems Engineering*, vol. 17, no. 4, pp. 407-425, 2014, https://doi.org/10.1111/sys.21279.

[3] J. J. López-Fernández, A. Garmendia, E. Guerra, and J. de Lara, "An example is worth a thousand words: Creating graphical modelling environments by example", *Software and Systems Modeling*, vol. 18, no. 2, pp. 961-993, 2019, https://doi.org/10.1007/s10270-017-0632-7.

[4] J. Serna, N. A. Day, S. Esmaeilsabzali, "Dash: declarative behavioural modelling in Alloy with control state hierarchy", *Software and Systems Modeling*, vol. 22, no. 2, pp. 733-749, 2023, https://doi.org/10.1007/s10270-022-01012-1.

[5] F. Ciccozzi, I. Malavolta, and B. Selic, "Execution of UML models: a systematic review of research and practice", *Software and Systems Modeling*, vol. 18, no. 3, pp. 2313-2360, 2019, https://doi.org/10.1007/s10270-018-0675-4.

[6] B. M. Duc, "Uml superstructure: language definition and diagrams", In *Real-time object uniform design methodology with UML*, 2007, pp. 77-190, https://doi.org/10.1007/978-1-4020-5977-3_4.

[7] J. E. Plazas, S. Bimonte, G. D. Sousa, and J. C. Corrales, "Data-centric UML profile for wireless sensors: Application to smart farming", *International Journal of Agricultural and Environmental Information Systems (IJAEIS)*, vol. 10, no. 2, pp. 21-48, 2019, https://doi.org/10.4018/IJAEIS.2019040102.

[8] M. Bartoletti, and R. Zunino, "BitML: a calculus for Bitcoin smart contracts", in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, 2018, pp. 83-100, https://doi.org/10.1145/3243734.3243795.

[9] H. Rocha, and S. Ducasse, "Preliminary steps towards modeling blockchain oriented software", In *2018 IEEE/ACM 1st International Workshop on Emerging Trends in Software Engineering for Blockchain (WETSEB)*, 2018, pp. 52-57, https://doi.org/10.1145/3194113.3194123.

[10] P. Bollen, "A Conceptual Model of the Blockchain", In *OTM Confederated International Conferences On the Move to Meaningful Internet Systems*, 2019, pp. 117-126, https://doi.org/10.1007/978-3-030-40907-4_12.

[11] M. Anvari, M. D. Takht-Fooladi, and B. Sefid-Dashti, "Thrift service composition: toward extending BPEL", In *Proceedings of the international conference on smart cities and internet of things*, 2018, pp. 1-5., https://doi.org/10.1145/3269961.3269973.

[12] A. S. Vingerhouts, S. Heng, Y. Wautelet, "Organizational modeling for blockchain oriented software engineering with extended-i* and uml", In CEUR Workshop Proceedings, vol. 2749, pp. 23-34, 2020,

[13] B. Sefid-Dashti, S. J. Sartakhti, and H. Daghigh, "Brand New Categories of Cryptographic Hash Functions: A Survey", *Journal of Electrical and Computer Engineering Innovations*, vol. 11, no. 2, pp. 335-354, 2023, https://doi.org/10.22061/JECEI.2023.9271.598.

[14] H. Sheth, and J. Dattani, "Overview of blockchain technology", *Asian Journal For Convergence In Technology (AJCT)*, 2019, vol. 5, no. 1, pp. 1-3, 2019, https://asianssr.org/index.php/ajct/article/view/728.

[15] A. M. Antonopoulos, *Mastering bitcoin: programming the open blockchain*, O'Reilly Media, Inc, 2017.

[16] A. Back, "Hashcash-a denial of service counter-measure", 2002, available at: http://www.hashcash.org/papers/hashcash.pdf, last accessed October. 25, 2023

[17] S. Nakamoto, "Bitcoin: a peer-to-peer electronic cash system", *Decentralized Business Review*, 2008, http://dx.doi.org/10.2139/ssrn.3440802

[18] Z. Wang, H. Yu, Z. Zhang, J. Piao, and J. Liu, "ECDSA weak randomness in Bitcoin, " *Future Generation Computer Systems*, vol. 102, pp. 507-513, 2020, https://doi.org/10.1016/j.future.2019.08.034.

[19] D. Jackson, "Alloy: a lightweight object modelling notation", *ACM Transactions on Software Engineering and Methodology*, vol. 11, no. 2, pp. 256-290, 2002, https://doi.org/10.1145/505145.505149.

[20] Object Management Group, "OMG Meta Object Facility (MOF) Core Specification", *OMG specification*, 2019, Ver. 2.5.1, available at: https://www.omg.org/spec/MOF/2.5.1/PDF, last accessed October. 25, 2023.

[21] D. Varró, and A. Pataricza, "VPM: A visual, precise and multilevel metamodeling framework for describing mathematical domains and UML (The Mathematics of Metamodeling is Metamodeling Mathematics)", *Software and Systems Modeling*, vol. 2, no. 3, pp. 187-210, 2003, https://doi.org/10.1007/s10270-003-0028-8.

[22] A. Herrera, P. Lara, M. Sánchez, and J. Villalobos, "Metamodeling the e-waste domain to support decision-making", *The International Journal of Logistics Management*, vol. 32, no. 10, pp. 262-283, 2020, https://doi.org/10.1108/IJLM-01-2020-0070.

[23] B. Tenbergen, and T. Weyer, "Generation of hazard relation diagrams: Formalization and tool support", *Software and Systems Modeling*, vol. 20, no. 1, pp. 175-210, 2021, https://doi.org/10.1007/s10270-020-00799-1.

[24] A. Gómez, R. J. Rodríguez, M. E. Cambronero, and V. Valero, "Profiling the publish/subscribe paradigm for automated analysis using colored Petri nets", *Software and Systems Modeling*, vol. 18, no. 5, pp. 2973-3003, 2019, https://doi.org/10.1007/s10270-019-00716-1.

[25] R. Miles and K. Hamilton, *Learning UML 2.0: a pragmatic introduction to UML*, O'Reilly, 2006.

[26] D. Perez-Palacin, J. Merseguer, J. I. Requeno, M. Guerriero, E. Di Nitto, and D. A. Tamburri, "A UML profile for the design, quality assessment and deployment of data-intensive applications", *Software and Systems Modeling*, vol. 18, no. 6, pp. 3577-3614, 2019, https://doi.org/10.1007/s10270-019-00730-3.

[27] G. Zoughbi, L. Briand, and Y. Labiche, "Modeling safety and airworthiness (RTCA DO-178B) information: conceptual model and UML profile", *Software and Systems Modeling*, vol. 10, no. 3, pp. 337-367, 2011, https://doi.org/10.1007/s10270-010-0164-x.

[28] D. Pilone, and N. Pitman, *UML 2.0 in a Nutshell*, O'Reilly Media, 2005.

[29] J. Warmer, and A. Kleppe, *Object Constraint Language: Getting Your Models Ready for MDA*, Addison Wisely Professional, 2003.

[30] Sparx Systems, "User Guide - MDG Technologies", Enterprise Architect User Guide Series, Ver. 1.0, 2017, available at: https://www.sparxsystems.com/resources/user-guides/modeling/mdg-technologies.pdf, last accessed October. 25, 2023.

[31] S. Bartolucci, F. Caccioli, and P. Vivo, "A percolation model for the emergence of the Bitcoin Lightning Network", Scientific reports, Vol 10, No. 1, p. 4488, 2020, https://doi.org/10.1038/s41598-020-61137-5.

[32] C. Pérez-Solà, S. Delgado-Segura, G. Navarro-Arribas, and J. Herrera-Joancomartí, "Double-spending prevention for bitcoin zero-confirmation transactions", *International Journal of Information Security*, vol. 18, no. 4, pp. 451-463, 2019, https://doi.org/10.1007/s10207-018-0422-4.

[33] L. Bass, P. Clements, and R. Kazman, *Software architecture in practice*, 4th edn., Addison-Wesley Professional, 2021.

[34] S. M. H. Bamakan, N. Nezhadsistani, , O. Bodaghi, , and Q. Qu, (). "Patents and intellectual property assets as non-fungible tokens; key technologies and challenges", *Scientific Reports*, vol. 12, no. 1, pp. 1-13, 2022, https://doi.org/10.1038/s41598-022-05920-6.

Behrouz Sefid-Dashti received his B.S. and M.S. degrees in computer engineering from the Iran Information Technology Development and Islamic Azad University (Tehran North Branch), respectively. He has over 18+ years of experience in the field of software development, and is currently a PhD candidate of software engineering at the University of Kashan. His career and experience include software analysis, design and architecture, and blockchain development. His research interests include blockchain, software architecture, model development, and cryptography.

Javad Salimi Sartakhti is an assistant professor of artificial intelligence in the department of computer engineering at the University of Kashan, Iran. He obtained his B.Sc. degree in computer engineering from the University of Kashan and his M.Sc. degree in software engineering from the Tarbiat Modares University, Tehran, Iran, in 2008 and 2013, respectively. In January 2017, he obtained his Ph.D. degree in artificial intelligence at the Isfahan University of Technology. He ranked first among students of computer engineering in all three degrees. His main research interests are mechanism design and game theory, blockchain, machine learning, and Deep learning.

Hassan Daghigh is an associate professor of mathematics at the University of Kashan, Iran. He received his M.Sc. in mathematics (Commutative Algebra) at the University of Tarbiat Modarres, Iran. He got his Ph.D degree in mathematics (elliptic curves) at McGill university, Canada in 1998, under the direction of Henri Darmon. His research activities are now focused on number theory (elliptic curves, algebraic number theory) and applications in cryptography in particular lattice and isogeny based cryptography.