computers

Article

Access Control Verification in Smart Contracts Using Colored Petri Nets

Issam Al-Azzoni * and **Saqib Iqbal**

College of Engineering, Al Ain University, Al Ain 64141, United Arab Emirates; saqib.iqbal@aau.ac.ae
* Correspondence: issam.alazzoni@aau.ac.ae

Abstract: This paper presents an approach for the verification of access control in smart contracts written in the Digital Asset Modeling Language (DAML). The approach utilizes Colored Petri Nets (CPNs) and their analysis tool CPN Tools. It is a model-driven-based approach that employs a new meta-model for capturing access control requirements in DAML contracts. The approach is supported by a suite of tools that fully automates all of the steps: parsing DAML code, generating DAML model instances, transforming the DAML models into CPN models, and model checking the generated CPN models. The approach is tested using several DAML scripts involving access control extracted from different domains of blockchain applications.

Keywords: DAML; smart contracts; access control; Colored Petri Nets

Citation: Al-Azzoni, I.; Iqbal, S. Access Control Verification in Smart Contracts Using Colored Petri Nets. *Computers* **2024**, *13*, 274. https://doi.org/10.3390/computers13110274

Academic Editor: Paolo Bellavista

Received: 6 September 2024
Revised: 11 October 2024
Accepted: 18 October 2024
Published: 22 October 2024

1. Introduction

Smart contracts play a major role in distributed ledger and blockchain technologies. They are self-executing contracts that automate business workflows involving multiple parties without requiring a central authority to manage them [1]. They are used across blockchain platforms for applications including commercial and financial transactions, legal processes, data sharing, supply chain management, and the Internet of Things (IoT) [2–4].

DAML (Digital Asset Modeling Language) is a smart contract language designed to simplify the development of smart contracts [5]. Developed by Digital Asset [6], DAML focuses on modeling business processes and workflows. One of the key features of DAML is its ability to define complex workflows and relationships between participants in a straightforward manner. DAML is also designed to be blockchain-agnostic, meaning that it can run on various distributed ledger technologies without being tied to a specific blockchain [5]. These include Hyperledger Fabric [7], Corda [8], VMware Blockchain [9], and an enterprise version of Ethereum based on IBFT [10].

An important feature of DAML is its ability to specify authorization and access control policies using novel primitives that mimic the primitives of today's legal systems [5]. The authorization and access control policies are specified along with the data and smart contract code as part of the primitives. In DAML, every piece of ledger data are annotated with a set of owners and a set of controllers who are authorized to change the data based on the logic specified by the smart contract code. The aim of supporting such primitives is to have a secure smart contract language.

Having secure smart contracts is critical for adopting blockchain technologies [11]. These contracts manage access to assets that can be too financially valuable. Any vulnerabilities in smart contracts can lead to substantial financial losses and an exploitation by malicious actors. Furthermore, once deployed, smart contracts are immutable, meaning that fixing bugs or security flaws can be challenging. However, these smart contracts automate complex business processes that can be hard to verify. In fact, several vulnerabilities in smart contracts have been exploited resulting in major financial losses recorded in billions of dollars [12,13].

Hence, several researches have proposed methods for the formal verification of smart contracts (surveys can be found in [14,15]). These include model-checking [16], machine learning [17], and theorem proving methods [18,19]. Of particular relevance to DAML smart contracts, we believe that Colored Petri Nets (CPNs) can be effective in modeling the complex business workflows that DAML was designed to implement. Several methods based on CPNs exist in the literature for the verification of smart contracts (these are surveyed in Section 2); however, this paper is unique in proposing a CPN-based approach designed specifically for verifying DAML contracts. The approach is model-based, in which all intermediate steps are fully automated. Furthermore, the approach focuses on authorization and access control verification of DAML contracts based on the actual semantics of DAML authorization primitives.

The main contributions of this paper are as follows:

1. A new meta-model that defines the main authorization concepts in DAML contracts.
2. A toolkit for automatically parsing DAML contracts and generating the corresponding DAML models.
3. A toolkit for automatically transforming DAML models into equivalent CPN models, which can be subsequently verified using CPN Tools.
4. A complete approach for verifying authorization requirements in DAML contracts is presented and evaluated on several contracts.

The organization of the paper is as follows. Section 2 reviews the related literature. Section 3 provides the necessary background on smart contracts and CPNs. Example DAML contracts are presented and discussed in Section 4. Our approach is presented in Section 5 and an evaluation of the approach is discussed in Section 6. Finally, Section 7 concludes the paper and discusses future work.

2. Related Work

Several formal verification methodologies have been proposed to verify smart contracts. One of these methodologies verifies the contracts by translating the contracts' code to F* [20]. The methodology translates Solidity and EVM bytecode contracts to F*. The translated code is tested by F*-type checking, which identifies vulnerable parts of the code. The weakness in this approach is its inability to translate the entire Solidity syntax to F* as it only covers a portion of the whole Solidity syntax [16]. Another approach for formal verification is based on verifying the bytecode of the contracts with the help of the interactive theorem prover Isabelle/HOL [18]. The contracts are partitioned into blocks, which are then verified using the Hoare logic in the theorem prover. The approach is still not mature as it does not cover the entire syntax of Solidity leaving loops and message calls unsupported [18]. Model-checking approaches have also been proposed to verify smart contracts. In [21], a model-checking-based method has been proposed to verify the smart contract using the NuSMV model checker. The model works efficiently for simple to moderate size contracts but suffers inefficiency in verifying complex contracts due to the language limitations of NuSMV.

In [22], the authors propose an approach for the formal analysis of Ethereum smart contracts based on CPNs. The approach can be used to analyze potential security vulnerabilities that could exit during the smart contract execution. The input to the approach can be in the form of a Solidity source code or EVM bytecode. The approach builds a control flow graph (CFG) and then the corresponding CPN model is generated. The generated CPN models are then run in CPN Tools to complete the dynamic simulation and model checking. The approach is highly automated, however, it is based on Solidity contracts and does not look at authorization-related vulnerabilities in particular.

Rakkay and Boucheneb [23] present a formal technique to model and analyze RBAC using CPNs. The authors represent a Role-Based Access Control (RBAC) policy using a CPN and use CPN for validation of the RBAC-based security policy. Several features of RBAC are incorporated, including role hierarchy, role and user cardinality, and static and dynamic separation of duties relations. However, the presented technique is not automated.

Also, it is not model-based and, hence, there is no metamodel that captures RBAC policies nor the verified business workflows.

The authors in [24] present an approach to model and analyze RBAC security policies using CPN formalism. In the approach, CPN Tools is used to analyze the generated CPN models. Several security properties about the RBAC security policy can be proven using the approach. Similar to the other related work presented earlier, this approach is not fully automatic and it is not a model-based approach.

In [25], the authors present a formal approach for the verification of Solidity smart contracts using CPNs. The approach is designed to check the absence of several kinds of vulnerabilities in Solidity smart contracts, such as integer overflow/underflow, reentrancy, self-destruction, timestamp dependence, and uninitialized storage variables. The vulnerabilities are formalized as Linear Temporal Logic (LTL) formulae that are subsequently verified using the Helena Petri net tool (https://lipn.univ-paris13.fr/helena/ (accessed on 11 October 2024)), which is a high level net analyzer available as a command line tool. The authors present an algorithm for transforming a Solidity smart contract into a CPN, but do not provide an implementation for automating the transformation.

The authors in [26] present an approach to design, develop, and verify secure smart contracts with a modeling tool-set that can be used to generate smart contracts before deploying them on blockchain. The approach is based on a Petri Net representation of the workflow that represents the business logic that a smart contract is supposed to implement. Petri net can be simulated and verified prior to the smart contract code generation.

Armando and Ponta [27] present an approach for formally modeling and analyzing authorization requirements in business processes. In their approach, model checking authorization requirements are used to separate the specification of the workflow from the associated access control policy. The approach is demonstrated using a loan origination process scenario featuring RBAC extended with conditional permission assignments and delegation. Petri nets are used to model the workflows.

In [28], the authors use CPNs to verify a crowdfunding Solidity smart contract. They demonstrate how the CPN model is able to discover logical vulnerabilities in the smart contract. The model checking capabilities of CPN Tools are utilized to discover attack scenarios.

He [29] presents an approach for modeling and analyzing smart contracts using predicate transition (PrT) nets. The approach is applied on smart contracts written in Solidity and that are deployed on the Azure Blockchain Workbench. The experiments conducted by the author show the applicability and suitability of PrT nets in modeling and analyzing smart contracts.

The authors in [30] propose a framework that incorporates model-driven engineering and formal verification into DAML smart contract life-cycle management. The framework utilizes CPNs for modeling the DAML smart contracts and verifying access control requirements. Our approach can supplement this framework by enabling a second round verification of the generated DAML smart contracts with a focus on identifying access control vulnerabilities. Furthermore, multi-party authorization is included in the CPN models in our approach, which we believe is lacking in their CPN models.

Compared with the aforementioned related work, our approach is distinguished by being model-based. Utilizing models of DAML smart contracts, all subsequent steps in the verification of the contracts are fully automated. The focus of our approach is on verifying access control requirements that are specified using DAML's native authorization primitives.

3. Background

3.1. Smart Contracts

Smart contracts were proposed in 1996 for agreements, protocols, and agreed-upon policies between the two trading parties in a digital environment [31]. Their use in the blockchain technology has proven to be the driving force behind blockchain technology's success. The difference between a real-world contract and smart contract lies on the fact that the smart contract does not require a mediator for the agreement like a real-world contract.

The onus of fulfillment of the smart contract is on the execution of protocols by the network nodes on a decentralized blockchain. The code of a smart contract is usually in a high-level programming language, such as Solidity and Vyper, which is compiled into bytes and run by the Ethereum Virtual Machines (EVM) on the Ethereum based blockchain technology. The most significant use of the smart contracts is in cryptocurrency where the trading of the cryptocurrency is executed and recorded through the execution of these contracts. The autonomous nature of smart contracts reduces the overhead costs of mediatory services and commissions [32]. It also reduces the risk of breach or interference of the third party.

Due to transactions of large sums and the absence of a third party, smart contracts are vulnerable to attacks. There are a wide array of attacks and vulnerabilities that exist in the current smart contracts, which can hamper the adoptability of smart contracts on a wider level [33]. In recent years, these vulnerabilities have been exploited to plunder billions of dollars out of cryptocurrency [34]; hence, the security of smart contracts is imperative for its wider adoptability.

3.2. Colored Petri Nets

CPNs are a mathematical modeling language used for representing and analyzing systems where concurrency, communication, and synchronization are key aspects. CPNs extend classical Petri nets by introducing colors (data values) to tokens, which allow modeling systems in a more compact way. This capability makes CPNs suitable for applications in various domains, such as communication protocols and distributed systems.

A CPN is formally defined as a tuple $(P, T, A, \Sigma, V, C, G, E, I)$ [35], where:

- P is a finite set of places,
- T is a finite set of transitions, with $P \cap T = \emptyset$,
- A is a finite set of arcs, where $A \subseteq (P \times T) \cup (T \times P)$,
- Σ is a set of color sets,
- V is a set of typed variables that can appear in expressions in the CPN,
- C is a color set function that assigns a color set to each place,
- G is a guard function that assigns a guard to each transition, expressed as a boolean function,
- E is an arc expression function that maps each arc to an expression, and
- I is an initialization function defining the initial marking.

Tools such as CPN Tools provide simulation and validation functionalities, enabling a detailed analysis of systems modeled using CPNs. This makes CPNs a powerful and versatile tool for system design and analysis, promoting a comprehensive understanding of the system dynamics in complex environments.

4. Example DAML Contracts

Consider the following DAML contract (taken from A Simple Cash Model, https://docs.daml.com/daml/intro/4_Transformations.html#a-simple-cash-model (accessed on 11 October 2024)):

```
module IOU_EXAMPLE where

data Cash = Cash with
  currency : Text
  amount : Decimal
    deriving (Eq, Show)

template SimpleIou
  with
    issuer : Party
    owner : Party
    cash : Cash
  where
    signatory issuer
    observer owner
    choice Transfer
        : ContractId SimpleIou
        with
            newOwner : Party
        controller owner
        do
        create this with
            owner = newOwner
```

In DAML, a template defines a type of contract that can be created. It also defines who has the right to create it. Contracts are instances of templates. For example, the template *SimpleIou* defines a type of contract named *SimpleIou*. A contract contains data. A *SimpleIou* contract contains three fields: *issuer* and *owner* (both are of type *Party*) and a *cash* record. The template definition specifies that the signatory is *issuer* and that *owner* is an observer.

Signatories of a contract in DAML are the parties whose authority is required to create the contract or archive it. Every contract must have at least one signatory. Observers on a contract are parties who can see that instance in the abstract ledger and all the information about it. For a *SimpleIou* contract, the *owner* is listed as an observer on the contract.

There is one problem with the *SimpleIou* contract. The contract is only signed by the *issuer*. The signatories of a contract are the parties who can create and archive contracts. Hence, if *Alice*, for example, gave *Bob* a *SimpleIou* for 100$ in exchange for some goods, she could just archive it after receiving the goods. There will be a record of such a transaction on the ledger, however *Bob* would need to resort to off-ledger means to get his money back. To make the *SimpleIou* contract safe for *Bob*, he needs to be added as a signatory.

To fix this authorization problem, the following updated *SimpleIou* contract adds *owner* as a signatory:

```
template SimpleIou
  with
    issuer : Party
    owner : Party
    cash : Cash
  where
    signatory issuer, owner
    observer owner
```

Now, to create a *SimpleIou*, authorizations by both the *issuer* and *owner* are needed. To collect the necessary authorizations, the Propose–Accept Workflow pattern [36] can be applied. This requires defining a proposal contract template as follows:

```
template IouProposal
  with
    iou : Iou
  where
    signatory iou.issuer
    observer iou.owner

    choice IouProposal_Accept
        : ContractId Iou
        controller iou.owner
        do
          create iou
```

The following DAML script demonstrates an example scenario to create a *SimpleIou* contract by collecting the necessary authorizations from *Alice* and *Bob*. First, *Alice* provides her authorization by creating and signing an *IouProposal* contract. Then, *Bob* adds his authorization by exercising the *IouProposal_Accept* choice. This would create a *SimpleIou* contract in the ledger, where *Alice* is the issuer and *Bob* is the owner.

```
  iouProposal <- submit alice do
    createCmd IouProposal with
      iou = Iou with
        issuer = alice
        owner = bob
        cash = Cash with
          amount = 100.0
          currency = "USD"

iou <- submit bob do
  exerciseCmd iouProposal IouProposal_Accept
```

In addition, a choice can have more than one controller. In this case, executing the choice requires the authorizations by all the controllers. For example, consider the following updated *Transfer* choice:

```
choice Transfer
  : ContractId Iou
  with
    newOwner : Party
  controller issuer,owner
  do
    create this with
      owner = newOwner
```

Now, the Propose–Accept Workflow pattern can be applied to collect the required authorizations before executing the *Transfer* choice. The full DAML code and test scripts can be found in the paper's GitHub repository at https://github.com/ialazzon/cpn_daml_paper (accessed on 11 October 2024). In addition, this repository contains all the source code and the test contracts presented in the paper.

5. Approach

The main problem this paper tackles is how to model the authorization workflow of a DAML smart contract in order to analyze such models and discover authorization problems. We propose a model-based approach that starts with a DAML smart contract and automatically produces a CPN that can be model checked and hence the authorization properties can be verified.

Our approach is composed of the following main steps:

1. Transform the DAML contract into a corresponding DAML model instance.
2. Transform the DAML model instance into a corresponding CPN model instance.
3. Transform the CPN model instance into a CPN.
4. Verify authorization properties on the CPN using simulation and model checking capabilities of CPN Tools.

All steps are fully automated in our approach. This section presents the DAML and CPN metamodels and discusses the application of our approach on the smart contracts presented in Section 4.

Figure 1 shows the DAML metamodel. The root element is an *SContract* representing a DAML contract. An *SContract* has a name and is composed of several *Templates*. A *Template* is associated with several observers and signatories (these are of type *Party*).

A *Template* is composed of several *Choices*. Each *Choice* can be associated with one or more controllers where a controller is a *Party*. A *Template* can reference another template (a single direction reference named *referenced_template*). Furthermore, a *Choice* can exercise another *Choice* (here, the referenced *Choice* is named *exercised_choice*). The reference *created_template* enables accessing the template containing a *Choice*.

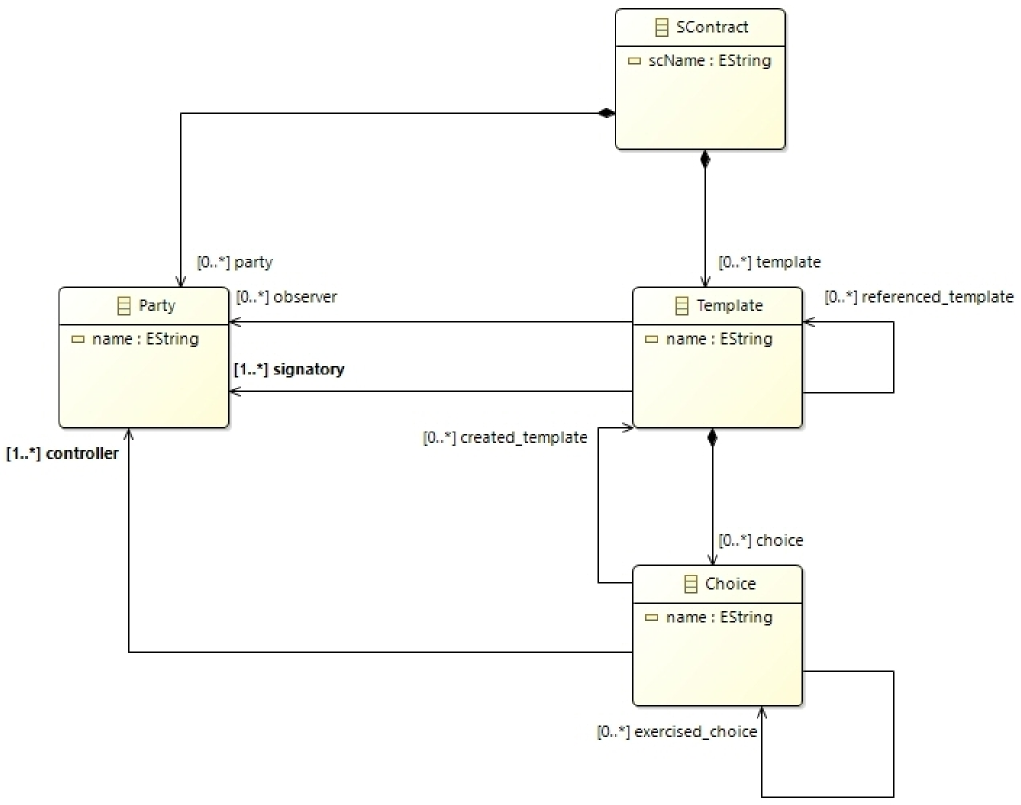

Figure 1. DAML metamodel.

Figure 2 shows an instance of the DAML metamodel, which corresponds to the DAML contract presented at the beginning of Section 4. This instance is generated automatically in our approach using the Eclipse Modeling Framework (EMF) [37]. The smart contract defines a single template (named *SimpleIou*) with a single choice (named *Transfer*). The template's signatory is the party named *issuer* and its observer is the party named *owner*. The choice's controller is the party *owner*. Note that this model instance is persisted in the XMI (XML Metadata Interchange) format.

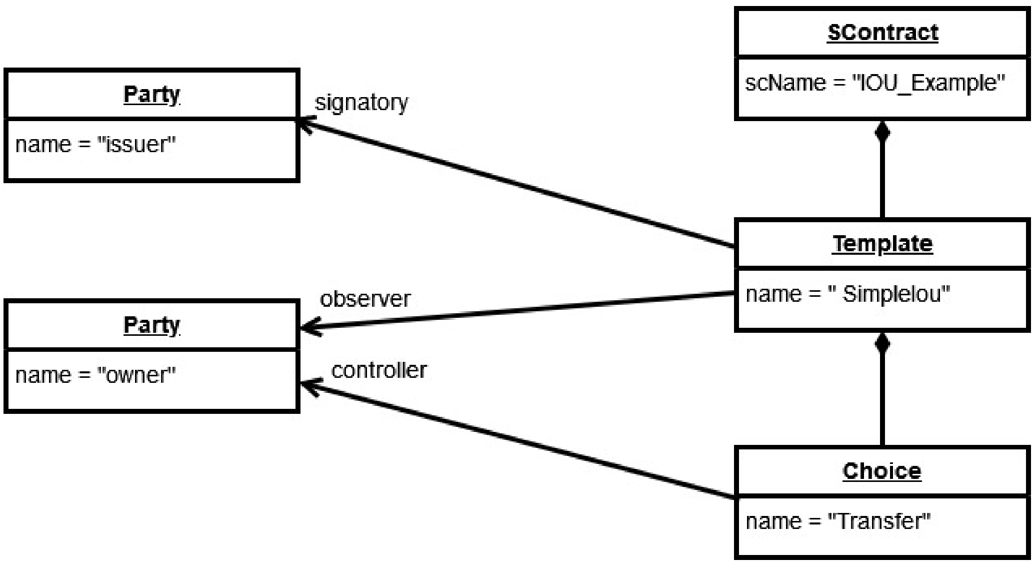

Figure 2. An instance of the DAML metamodel.

For the CPN metamodel, we utilize the metamodel from the CPN Tools toolkit [38]. The toolkit includes a plug-in to create CPN Tools files from EMF. In addition, the toolkit provides capabilities to layout and serialize EMF-compatible CPN models into a CPN Tools XML format. Also, the toolkit defines a metamodel for CPNs which we utilize for creating CPN model instances from DAML models. The CPN metamodel can be found in the CPN Metamodel. https://github.com/abelgomez/cpntools.toolkit/blob/master/plugins/io.github.abelgomez.cpntools/model/cpntools.ecore (accessed on 11 October 2024).

Figure 3 shows the CPN generated by our approach for the first contract presented in Section 4. For the updated contract that has two signatories, Figure 4 shows the generated CPN.

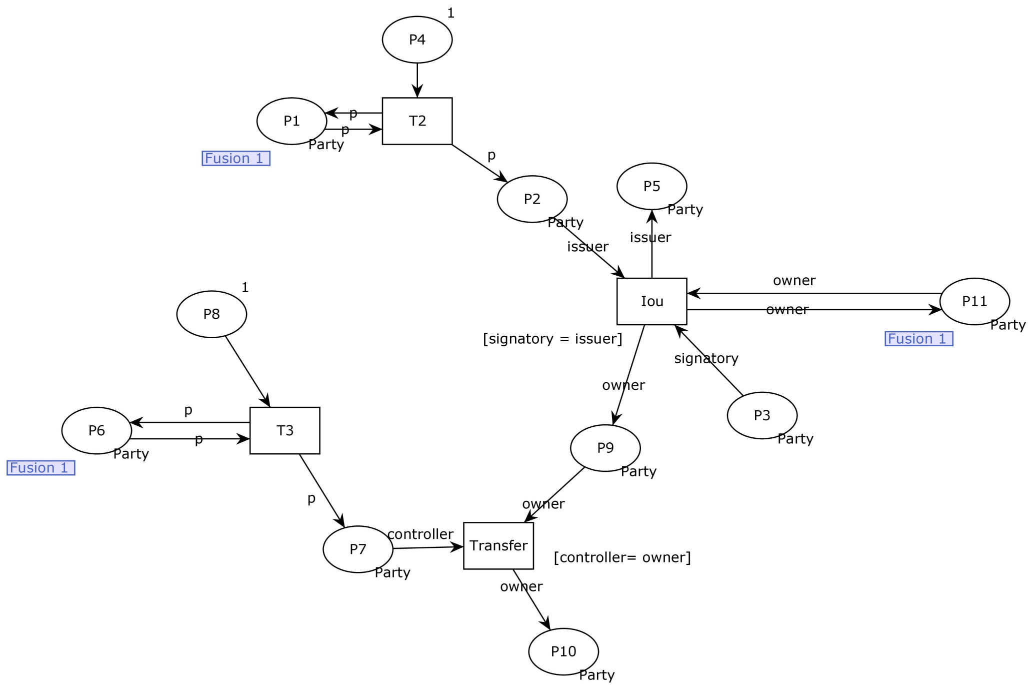

Figure 3. The CPN corresponding to the first *IOU_EXAMPLE* DAML smart contract in Section 4.

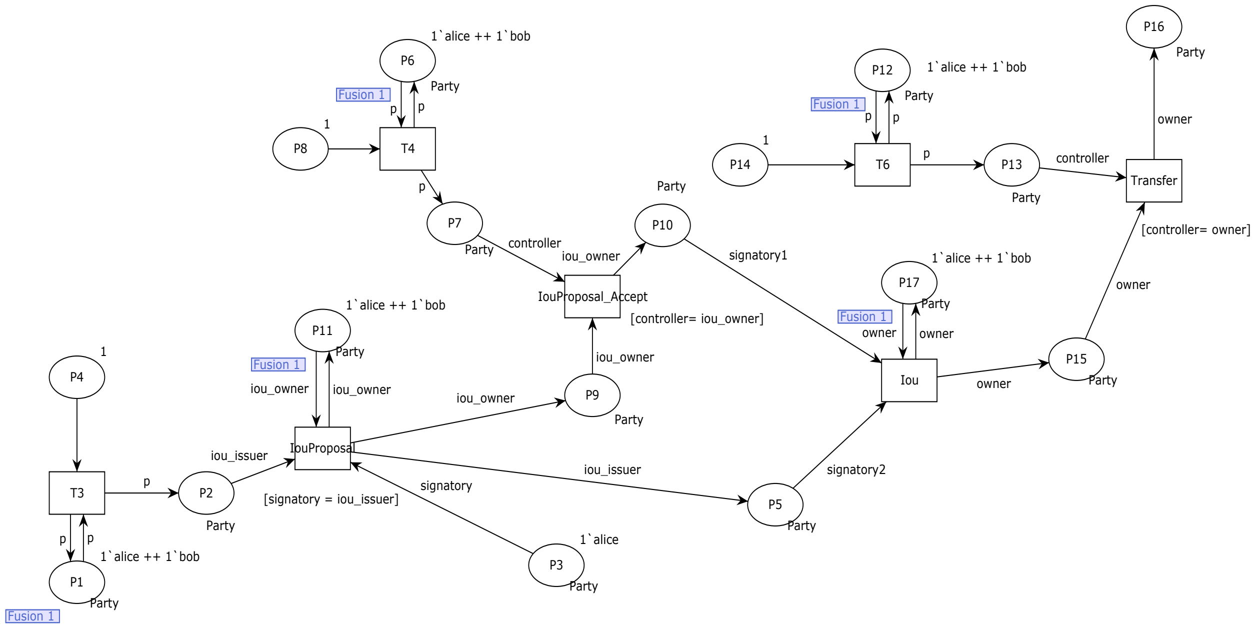

Figure 4. The CPN corresponding to the second *IOU_EXAMPLE* DAML smart contract in Section 4.

The first CPN, shown in Figure 3, has two main transactions: *SimpleIou* and *Transfer*. Firing the transaction *SimpleIou* represents a creation of the *SimpleIou* contract, while firing the transaction *Transfer* represents a successful invocation of the *Transfer* choice. The CPN indicates that any party can instantiate the contract as a signatory while specifying an owner. Later, the owner party can invoke the *Transfer* choice at any any time. In the second CPN shown in Figure 4, the authorizations of two signatories (labeled *signatory1* and *signatory2*) are required for the creation of an *Iou* contract. To collect the necessary authorizations, the Propose–Accept Workflow pattern can be applied as captured in the CPN model. Thereafter, once an *Iou* contract is created, the owner party can invoke the *Transfer* choice.

6. Evaluation

In this section, we present the results of using the Calculate state space tool in CPN Tools to generate and analyze the state spaces of the two CPNs presented in Section 5.

Table 1 shows the state space information statistics reported by the Calculate state space tool for the CPNs in Figures 3 and 4 (named *IOU_EXAMPLE 1* and *IOU_EXAMPLE 2*, respectively). In both cases, the number of seconds reported by the tool is zero indicating very short state space generation time. Also, the full state space was generated in both cases.

Table 1. State Space Information Statistics.

	***IOU_EXAMPLE 1* CPN**	***IOU_EXAMPLE 2* CPN**
Number of Nodes:	17	59
Number of Arcs:	24	124

Consider the first DAML contract whose CPN is shown in Figure 3. We are interested in answering the following question: assuming that the party *Alice* signed the *SimpleIou* contract, which parties can invoke the *Transfer* choice?

To answer this question, we initialize the marking of place *P3* to a single token $1'\ alice$. In addition, we initialize the marking of the fusion set $Fusion1$ to two tokens: $1'\ alice$ and $1'\ bob$. Here, *Bob* represents a party distinct from *Alice*. Then, the full state space can be generated using the Calculate state space tool in CPN Tools.

The next step involves analyzing the generated state space. We are interested in determining all paths starting from the initial marking that end at a marking of the CPN (i.e., a node in the state space), such that the transaction *Transfer* is enabled and hence can fire. To do so, we define the following function in CPN ML language:

```
fun FindMarkingsCanFire () : Arc list
= PredAllArcs(
fn a => st_TI(ArcToTI(a)) = "Page’Transfer 1")
```

This function returns a list of all arcs in the state space that represent the firing of the transition $Transfer$. Then, by extracting the source node of each returned arc using the CPN ML function *SourceNode*, we can obtain the markings in which transaction *Transfer* is enabled.

Figure 5 shows a partial state space generated for the *IOU_EXAMPLE 1* CPN. The node corresponding to the initial marking is labeled 1. The nodes labeled 12 and 15 represent the target markings. These are the markings in which transaction *Transfer* is enabled. They were obtained using the CPN ML queries described above. Using the Calculate state space tool, we can trace back all predecessor nodes to find the paths from the initial marking.

To answer our stated question and by examining the state space, there are two target markings labeled 12 and 15. The target marking 12 represents a state of the contract where the owner is defined to be *Alice*, while the target marking 15 represents the state where the owner is defined to be *Bob*. Hence, the owner defined in the contract may invoke the *Transfer* choice as dictated by the DAML rules.

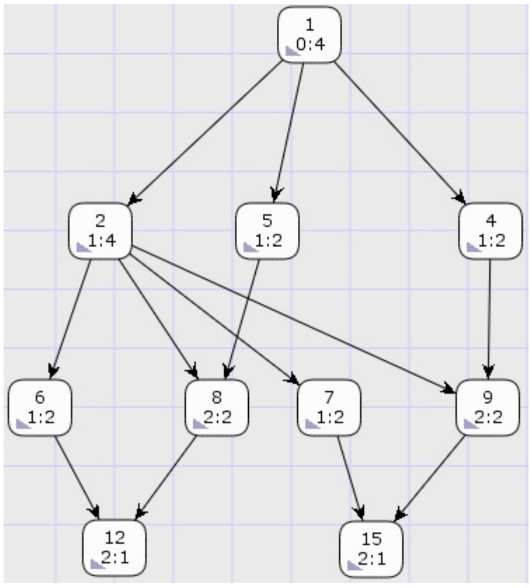

Figure 5. A partial state space generated for the CPN in Figure 3.

Furthermore, the state space graph generated by CPN Tools can be exported to Graphviz [39]. Graphviz is an open source graph visualization software. This can be done by using the *OGtoGraphviz* functions in CPN Tools (https://cpntools.org/2018/01/15/draw-state-spaces-with-graphviz/ (accessed on 11 October 2024)). To aid the users, we developed a Python 3 script that accepts the Graphviz file exported by CPN Tools as the as the input and produces a summary description of all simple paths starting from the initial node (marking) and ending at a final marking provided as an argument as the output. For each simple path, the Python script prints the information of the binding elements for the transitions corresponding to the contract creation and the choice execution. For example, the output when using the final marking 17, which is the next marking after firing the *Transfer* transaction at marking 15 (see Figure 5) is:

```
N1 -> N4 -> N9 -> N15 -> N17
Iou {owner=bob,signatory=alice,issuer=alice}
Transfer {controller=bob,owner=bob}

N1 -> N2 -> N9 -> N15 -> N17
Iou {owner=bob,signatory=alice,issuer=alice}
Transfer {controller=bob,owner=bob}

N1 -> N2 -> N7 -> N15 -> N17
Iou {owner=bob,signatory=alice,issuer=alice}
Transfer {controller=bob,owner=bob}
```

These are the possible execution scenarios that correspond to having *Alice* as a signatory and *Bob* as the owner. Similarly, the output corresponding to marking 16 represents execution scenarios when *Alice* servers as both a signatory and an owner of the created *Iou* contract. The Graphviz full state space graph is shown in Figure 6.

Now, we consider the second CPN, *IOU_EXAMPLE 2*, shown in Figure 4. In this case, the template *Iou* has two signatories: *issuer* and *owner*. Hence, the *Iou* contracts cannot be created by a single authorization from the *issuer* or the *owner*. The input DAML contract, in this case, applies the Propose–Accept Workflow pattern as discussed in Section 5.

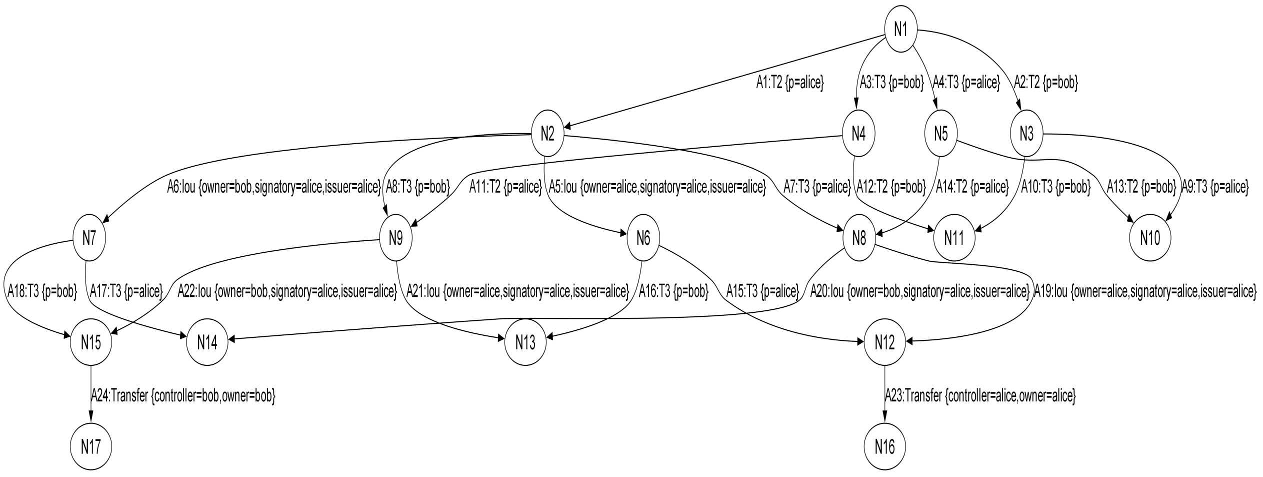
N1
A1:T2 {p=alice}
A3:T3 {p=bob}
A4:T3 {p=alice}
A2:T2 {p=bob}
N2
N4
N5
N3
A6:Iou {owner=bob,signatory=alice,issuer=alice}
A8:T3 {p=bob}
A11:T2 {p=alice}
A5:Iou {owner=alice,signatory=alice,issuer=alice}
A7:T3 {p=alice}
A12:T2 {p=bob}
A14:T2 {p=alice}
A10:T3 {p=bob}
A13:T2 {p=bob}
A9:T3 {p=alice}
N7
N9
N6
N8
N11
N10
A18:T3 {p=bob}
A17:T3 {p=alice}
A22:Iou {owner=bob,signatory=alice,issuer=alice}
A21:Iou {owner=alice,signatory=alice,issuer=alice}
A16:T3 {p=bob}
A15:T3 {p=alice}
A20:Iou {owner=bob,signatory=alice,issuer=alice}
A19:Iou {owner=alice,signatory=alice,issuer=alice}
N15
N14
N13
N12
A24:Transfer {controller=bob,owner=bob}
A23:Transfer {controller=alice,owner=alice}
N17
N16

Figure 6. The full Graphviz state space graph generated for the CPN in Figure 3.

Here, we are interested in determining all possible ways to instantiate an *Iou* template. We assume that the parties involved are *Alice* and *Bob*, and hence the fusion set is initialized to the marking $1'\,alice + +1'\,bob$. This multi-set has two tokens that correspond to *Alice* and *Bob*. We also assume that *Alice* is the signatory for the template *IouProposal*. We adapt the *FindMarkingsCanFire* function to return only the markings in which the *Iou* transaction is enabled. This is achieved by changing the name of the transaction *Page'Transfer* to *Page'Iou*. This returns a list of arcs, and by executing the *SourceNode* function on the returned arc numbers, the following nodes are returned: 40, 45, 50, 51, 52, and 53. By examining the markings of places *P5* and *P10* at these state space nodes, we determine that the *Iou* contract can be signed by *Alice* and *Bob* as the two signatories, or by *Alice* alone serving both as *signatory1* and *signatory2*.

All of the artifacts corresponding to the two scenarios presented above are available in the paper's GitHub repository. In addition, there are six more test contracts (with the corresponding generated DAML models and CPNs) included in the repository.

7. Conclusions

This paper has presented a model-based approach for the verification of authorization and access control requirements in DAML smart contracts using CPNs. Starting from a DAML contract's code, all intermediate models can be automatically generated, including the DAML and CPN models and the CPN itself. Subsequently, using CPN Tools, the CPN can be used to create and analyze the state space and identify specific authorization scenarios.

There are three main items for future work and extensions to this paper. First, the evaluation can be extended by analyzing DAML contracts deployed in real-life use cases. Second, a plug-in can be implemented in Eclipse to incorporate all the tools presented in the paper in one place for use by users. Third, currently the DAML meta-model does not incorporate DAML choice assertions. These are conditions that can be specified for a choice and act as preconditions that dictate whether the choice can be executed based on the state of the contract or the inputs provided to the choice. We believe that CPNs are able to model such assertions using transaction guards. Hence, our approach can be extended to support contracts that include choice assertions.

Author Contributions: Conceptualization, I.A.-A. and S.I.; methodology, I.A.-A.; software, I.A.-A.; validation, I.A.-A. and S.I.; formal analysis, I.A.-A. and S.I.; investigation, I.A.-A. and S.I.; resources, I.A.-A. and S.I.; data curation, I.A.-A.; writing—original draft preparation, I.A.-A. and S.I.; writing—review and editing, I.A.-A. and S.I.; visualization, I.A.-A.; All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: The original data and code presented in the study are openly available in GitHub at https://github.com/ialazzon/cpn_daml_paper (accessed on 11 October 2024).

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Mohanta, B.K.; Panda, S.S.; Jena, D. An overview of smart contract and use cases in blockchain technology. In Proceedings of the International Conference on Computing, Communication and Networking Technologies, Bengaluru, India, 10–12 July 2018; pp. 1–4. [CrossRef]
2. Lone, A.H.; Naaz, R. Applicability of blockchain smart contracts in securing Internet and IoT: A systematic literature review. *Comput. Sci. Rev.* **2021**, *39*, 100360. [CrossRef]
3. Hewa, T.; Ylianttila, M.; Liyanage, M. Survey on blockchain based smart contracts: Applications, opportunities and challenges. *J. Netw. Comput. Appl.* **2021**, *177*, 102857. [CrossRef]
4. Wang, S.; Yuan, Y.; Wang, X.; Li, J.; Qin, R.; Wang, F.Y. An Overview of Smart Contract: Architecture, Applications, and Future Trends. In Proceedings of the IEEE Intelligent Vehicles Symposium, Suzhou, China, 26–30 June 2018. [CrossRef]
5. Bernauer, A.; Faro, S.; Hämmerle, R.; Huschenbett, M.; Kiefer, M.; Lochbihler, A.; Mäki, J.; Mazzoli, F.; Meier, S.; Mitchell, N.; et al. Daml: A smart contract language for securely automating real-world multi-party business workflows. *arXiv* **2023**, arXiv:2303.03749.
6. Digital Asset. Available online: https://www.digitalasset.com/ (accessed on 11 October 2024).
7. Androulaki, E.; Barger, A.; Bortnikov, V.; Cachin, C.; Christidis, K.; De Caro, A.; Enyeart, D.; Ferris, C.; Laventman, G.; Manevich, Y.; et al. Hyperledger Fabric: A distributed operating system for permissioned blockchains. In Proceedings of the Thirteenth EuroSys Conference, Porto, Portugal, 23–26 April 2018; pp. 1–15.
8. Corda. Available online: https://corda.net/ (accessed on 11 October 2024).
9. VMware Blockchain. Available online: https://www.vmware.com/products/blockchain.html (accessed on 11 October 2024).
10. Saltini, R.; Hyland-Wood, D. IBFT 2.0: A safe and live variation of the IBFT blockchain consensus protocol for eventually synchronous networks. *arXiv* **2019**, arXiv:1909.10194.
11. Mense, A.; Flatscher, M. Security vulnerabilities in Ethereum smart contracts. In Proceedings of the International Conference on Information Integration and Web-Based Applications & Services, Yogyakarta, Indonesia, 19–21 November 2018; pp. 375–380.
12. Atzei, N.; Bartoletti, M.; Cimoli, T. A Survey of Attacks on Ethereum Smart Contracts (SoK). In *Principles of Security and Trust, Proceedings of the 6th International Conference, POST 2017, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2017, Uppsala, Sweden, 22–29 April 2017* ; Proceedings 6; Maffei, M.; Ryan, M., Eds.; Springer: Berlin/Heidelberg, Germany, 2017; pp. 164–186. [CrossRef]
13. Tang, X.; Zhou, K.; Cheng, J.; Li, H.; Yuan, Y. The Vulnerabilities in Smart Contracts: A Survey. In *Advances in Artificial Intelligence and Security, Proceedings of the 7th International Conference, ICAIS 2021, Dublin, Ireland, 19–23 July 2021*; Proceedings, Part III 7; Sun, X., Zhang, X., Xia, Z., Bertino, E., Eds.; Springer: Cham, Switzerland, 2021; pp. 177–190. [CrossRef]
14. Krichen, M.; Lahami, M.; Al-Haija, Q.A. Formal methods for the verification of smart contracts: A review. In Proceedings of the International Conference on Security of Information and Networks, Sousse, Tunisia, 11–13 November 2022; pp. 1–8.
15. Murray, Y.; Anisi, D.A. Survey of formal verification methods for smart contracts on blockchain. In Proceedings of the International Conference on New Technologies, Mobility and Security, Canary Islands, Spain, 24–26 June 2019; pp. 1–6.
16. Bai, X.; Cheng, Z.; Duan, Z.; Hu, K. Formal modeling and verification of smart contracts. In Proceedings of the International Conference on Software and Computer Applications, Kuantan, Malaysia, 8–10 February 2018; pp. 322–326.
17. Jiang, F.; Chao, K.; Xiao, J.; Liu, Q.; Gu, K.; Wu, J.; Cao, Y. Enhancing smart-contract security through machine learning: A survey of approaches and yechniques. *Electronics* **2023**, *12*, 2046. [CrossRef]
18. Ribeiro, M.; Adão, P.; Mateus, P. Formal Verification of Ethereum Smart Contracts Using Isabelle/HOL. In *Logic, Language, and Security: Essays Dedicated to Andre Scedrov on the Occasion of His 65th Birthday*; Nigam, V., Ban Kirigin, T., Talcott, C., Guttman, J., Kuznetsov, S., Thau Loo, B., Okada, M., Eds.; Springer: Cham, Switzerland, 2020; pp. 71–97. [CrossRef]
19. Yang, Z.; Lei, H. Formal process virtual machine for smart contracts verification. *arXiv* **2018**, arXiv:1805.00808. [CrossRef]
20. Bhargavan, K.; Delignat-Lavaud, A.; Fournet, C.; Gollamudi, A.; Gonthier, G.; Kobeissi, N.; Kulatova, N.; Rastogi, A.; Sibut-Pinote, T.; Swamy, N.; et al. Formal verification of smart contracts: Short paper. In Proceedings of the ACM Workshop on Programming Languages and Analysis for Security, Vienna, Austria, 24 October 2016; pp. 91–96. [CrossRef]
21. Nehai, Z.; Piriou, P.Y.; Daumas, F. Model-checking of smart contracts. In Proceedings of the IEEE International Conference on Blockchain, Halifax, NS, Canada, 30 July–3 August 2018; pp. 980–987. [CrossRef]
22. Duo, W.; Xin, H.; Xiaofeng, M. Formal Analysis of Smart Contract Based on Colored Petri Nets. *IEEE Intell. Syst.* **2020**, *35*, 19–30. [CrossRef]
23. Rakkay, H.; Boucheneb, H. Security Analysis of Role Based Access Control Models Using Colored Petri Nets and CPNtools. In *Transactions on Computational Science IV: Special Issue on Security in Computing*; Gavrilova, M.L., Tan, C.J.K., Moreno, E.D., Eds.; Springer: Berlin/Heidelberg, Germany, 2009; pp. 149–176. [CrossRef]

24. Kahloul, L.; Djouani, K.; Tfaili, W.; Chaoui, A.; Amirat, Y. Modeling and Verification of RBAC Security Policies Using Colored Petri Nets and CPN-Tool. In *Networked Digital Technologies*; Zavoral, F., Yaghob, J., Pichappan, P., El-Qawasmeh, E., Eds.; Springer: Berlin/Heidelberg, Germany, 2010; pp. 604–618. [CrossRef]
25. Garfatta, I.; Klai, K.; Graïet, M.; Gaaloul, W. Model checking of vulnerabilities in smart contracts: A Solidity-to-CPN approach. In Proceedings of the ACM/SIGAPP Symposium on Applied Computing, Brno, Czech Republic, 25–29 April 2022; pp. 316–325. [CrossRef]
26. Zupan, N.; Kasinathan, P.; Cuellar, J.; Sauer, M. Secure Smart Contract Generation based on Petri Nets. In *Blockchain Technology for Industry 4.0: Secure, Decentralized, Distributed and Trusted Industry Environment*; Rosa Righi, R.D., Alberti, A.M., Singh, M., Eds.; Springer: Singapore, 2020; pp. 73–98. [CrossRef]
27. Armando, A.; Ponta, S.E. Model checking authorization requirements in business processes. *Comput. Secur.* **2014**, *40*, 1–22. [CrossRef]
28. Liu, Z.; Liu, J. Formal Verification of Blockchain Smart Contract Based on Colored Petri Net Models. In Proceedings of the Computer Software and Applications Conference, Milwaukee, WI, USA, 15-19 July 2019; pp. 555–560. [CrossRef]
29. He, X. Modeling and Analyzing Smart Contracts using Predicate Transition Nets. In Proceedings of the International Conference on Software Quality, Reliability and Security Companion, Macau, China, 11–14 December 2020; pp. 108–115. [CrossRef]
30. Mustafa, I.; McGibney, A.; Rea, S. Smart contract life-cycle management: An engineering framework for the generation of robust and verifiable smart contracts. *Frontiers in Blockchain* **2024**, *6*, 1276233. [CrossRef]
31. Szabo, N. Smart contracts: Building blocks for digital markets. *EXTROPY J. Transhumanist Thought* **1996**, *18*, 28.
32. Swan, M. *Blockchain: Blueprint for a New Economy*; O'Reilly Media: Sebastopol, CA, USA, 2015.
33. Yi, X.; Yang, X.; Kelarev, A.; Lam, K.Y.; Tari, Z. *Blockchain Foundations and Applications*; Springer: Berlin/Heidelberg, Germany, 2022.
34. Parisi, C.; Budorin, D. DAO Security. In *Web3 Applications Security and New Security Landscape: Theories and Practices*; Springer: Berlin/Heidelberg, Germany, 2024; pp. 35–54.
35. Jensen, K.; Kristensen, L.M. *Coloured Petri Nets: Modelling and Validation of Concurrent Systems*; Springer: Berlin/Heidelberg, Germany, 2009.
36. The Propose and Accept Pattern. Available online: https://docs.daml.com/daml/patterns/propose-accept.html (accessed on 11 October 2024).
37. Eclipse Modeling Framework. Available online: https://eclipse.dev/modeling/emf/ (accessed on 11 October 2024).
38. CPN Tools Toolkit. Available online: https://github.com/abelgomez/cpntools.toolkit (accessed on 11 October 2024).
39. Graphviz. Available online: https://graphviz.org/ (accessed on 11 October 2024).