Role-Based Embedded Domain-Specific Language for Collaborative Multi-Agent Systems through Blockchain Technology

Orçun Oruç

TU Dresden, Software Technology Group, Nöthnitzer Straße 46, 01187, Dresden

Abstract. Multi-agent systems have evolved with their complexities over the past few decades. To create multi-agent systems, developers should understand the design, analysis, and implementation together. Agent-oriented software engineering applies best practices through mainly software agents with abstraction levels in the multi-agent systems. However, abstraction levels take a considerable amount of time due to the design complexity and adversity of the analysis phase before implementing them. Moreover, trust and security of multi-agent systems have never been detailed in the design and analysis phase even though the implementation of trust and security on the tamper-proof data are necessary for developers. Nonetheless, object-oriented programming is the right way to do it, when implementing complex software agents, one of the major problems is that the object-oriented programming approach still has a complex process-interaction and a burden of event-goal combination to represent actions by multi-agents. Designated roles with their relationships, invariants, and constraints of roles can be constructed based on blockchain contracts between agents. Furthermore, in the case of new agents who participate in an agent network, decentralization and transparency are two key parameters, which agents can exchange trusted information and reach a consensus aspect of roles. This study will take the software agent development as a whole with analysis, design, and development with role-object pattern in terms of smart contract applications. In this paper, we aim to propose a role-based domain-specific language that enables smart contracts which can be used in agent-oriented frameworks. Furthermore, we would like to refer to methodology, results of the research, and case study to enlighten readers in a better way. Finally, we summarize findings and highlight the main research points by inferencing in the conclusion section.

Keywords: Software agents, Domain-specific languages, Blockchain technology, Smart contracts, Role-based programming languages.

1 Introduction

Agent-oriented programming (AOP) can be considered as a subset of object-oriented programming by showing the state of an object with human-like features such as belief, desire, intentions, and goals. Moreover, an agent should be in the interaction with other agents, in this way, agents are able to play roles as human-being does. AOP specializes the object-oriented programming methodology by fixing state and modules (called agents) to consist of features that are coming from agent behaviors [1]. Besides, agents can handle the message passing between other agents internally.

Agent-based systems have changed their characteristics over the past few decades aspect of design, analysis, and implementation. Although one can find out the difficulty of exact definition in terms of the multi-agent system, multi-agent systems are used broadly in the application areas such as supply chain management, distributed systems, smart grids, robotic motion planning. Multi-agent systems are strongly dependent on contexts and roles. Each agent plays a role and it has a minimal set of attributes that represents the environment. Moreover, agents should have behaviors that are, in essence, related to implementing deterministic or non-deterministic behaviors of an agent that operate a role that can be in sequential order, cyclic order, or parallel order.

Agents must be in a relationship with external trusted parties to provide privacy and consistency in multi-agent systems. However, the trust was mostly provided by different logic interpretations and cumbersome ontological definitions in multi-agent systems. Blockchain technology offers a credible and private data pool that can be used with programmable contracts (smart contracts) as a shared database. As we solved the credibility problem with blockchain technology, we can enforce the data layer security, which is vital for data-driven multi-agent system communication, by implementing smart contracts on the application layer of the blockchain protocol. A smart contract is a piece of code that is stored on a blockchain by triggering coin-based transactions with saved data and which reads and writes data in a blockchain database [1]. In addition, smart contracts can ensure the testability of role features, secure transactions within the blockchain database, and separation of the business logic (model) and application logic (system architecture).

Role-based programming can be integrated into the agent concept, which is useful to reduce the complexity of the agent system design by categorizing the roles played by agents and describing the collaboration among agents [2]. We should consider equally grouping roles together in a collaborative relation or a compartment. Roles are essentially defining context-oriented software, which is an explicit data model of roles or objects that combine conditions, activated relationships of roles, and deactivate relationships of roles [3]. For instance, an agent may produce different results under different contexts, hence the given agent behaves differently in a specified environment or a collaborative agent simulation.

Compartments belong to the research of the Compartment Role Object Model (CROM) that establishes subtypes of natural types and relationship types between combined roles [4]. CROM combines the behavioral, relational, and context-dependent nature of roles in a common framework [4]. It is a research project that points out a framework for conceptual modeling that incorporates roles, graphical

[1] `https://www.coindesk.com/three-smart-contract-misconceptions`

modeling language, and a set-based formalization of roles, which has been conducted by TU Dresden Software Technology [4].

Roles can make the design of multi-agent systems easier by implementing the composition of role attributes, role invariants, role methods, and binding-interfaces. The difference between roles and objects is whether or not the roles can move hosts that exist in an environment [5]. Role-based software agents are related to context-aware multi-agent systems. Context is any information that is accessible to the program, where an entity is a person, place, or another agent that is considered relevant to the determination of behavioral variations [6]. Agents can dynamically collaborate with roles, create coalitions of trusted partners as an effective mechanism to communicate with service requestors, find services requested by them, and determine trusted services and provide services to the applicants without violating the privacy of the predefined environment [7]. A domain-specific language in an agent-oriented language can map abstraction of role-compartment to particular composition in an agent-oriented architecture. By adding design-by-contract language such as Solidity, agent-oriented language can assure role constraints regarding role-compartments.

1.1 Research Problem

Principally, multi-agent systems have been used as a software design methodology for software application problems over a few decades. Frameworks and agent communication languages that were proposed are still hard to understand and use effectively in a decentralized and centralized network. Lack of standardization in the area of analysis, design, and implementation increases software design complexity as we plan to deploy decentralized agents.

Agent communication languages such as KQML, KIF, FIPA-ACL, AgentSpeak, and major agent deployment frameworks JADE [8], JADEX [9], JASON [10], GOAL [11], JACK Framework [12], JaCaMo [13], 3APL [14], and 2APL [15] do not offer any solution for privacy, security, and trust at the level of deployment of agents. Furthermore, a variety of these languages creates a burden for language mapping. Moreover, previous solutions have no practical design-by-contract approach so as to establish the goals and actions of agents.

Researchers who deal with multi-agent systems still offer limited modeling solutions for the aforementioned problems. A domain-specific modeling language that merges general-purpose language, agent communication language, and blockchain-based design-by-contract language can make the developers' and researchers' life easier and we can map roles and goals at the analysis phase to the deployment phase through smart contract language.

Thus, current challenges of the programming aspect of the multi-agent systems lead us to create a new approach and a solution as we named role-based blockchain-enabled domain-specific language for collaborative multi-agent systems.

To conclude up regarding the research problem, we have defined research questions(RQ) as below:

- *RQ1*: Can a domain-specific language that comprises the main features of agent communication language, agent framework, and smart contract language be created?
- *RQ2*: How can roles, goals, and compartments be implemented with the domain-specific language?

1.2 Motivation and Challenges

The main motivation of this study is to create a goal-driven (so-called cognitive) agent-oriented language with blockchain technology to provide goals, desires, and intentions in multi-agent systems. The current challenges of programming in multi-agent systems lead us to create a new approach and solution in order to solve the aforementioned problems in the Introduction section.

- Multi-agent systems or swarm management should assign trust and privacy levels for new agents that consist of roles, goals, and plans to increase efficiency in the network.
- Multi-agent systems should ensure trust and privacy in data-driven domains. A human operator or an external participant should see it as a black-box process.
- System planning with the belief-desire-intention reasoning engine suffers the vulnerability of critical decisions. Such decisions may be capabilities, role assignment between agents, limitations of follower agents, and leader agents while changing positions.
- Protection against malicious agents is dependent on mostly language virtual machine environments. A developer should know the specifications of a language that relies on a virtual machine such as Java, which is a cumbersome and error-prone task. If an agent is allowed to communicate with external agents, the smart contract can alleviate the complexity of the security task.

1.3 Outline of Objectives and Contributions

Goal: The main goal of the present study is to implement a domain-specific language to demonstrate role constraints, types, invariants, and relationships using design-by-contracts, which can be done by external blockchain programming language such as smart contracts, with a secured and trusted environment for software agents.

Objectives:

- Identify existing roles from Compartment Role Object Model (CROM) in the context of compartments.

- Mapping from natural types, role types, compartment types, and relationship types to the data structures to a smart contract language such as Solidity. We would like to write a code generator from a general-purpose language to create a smart contract language so that we will have a common language with the agent-oriented programming language.
- Implementing role-definition to agent containers because entities can be bounded to devices on which agents are able to move, create, and deploy.
- Defining collaborative goal-driven roles for agents themselves that reside in agent containers.
- At the implementation phase, we defined the above-mentioned roles' and goals' specifications in a smart contract language for design-by-contract to combine with an agent-oriented programming framework.
- If we have enough time in the course of Ph.D., we will deploy a real multi-agent system such as a multi-agent unmanned air vehicle or robotic arm collaboration.

We will contribute to different aspects inducting from different research areas such as smart contract programming in the blockchain, multi-agent system development frameworks and communication languages, and decentralized agent networking. We have listed our conceivable contributions in this study:

- We introduce a new role-based agent-oriented domain-specific language that is capable to use blockchain technology at the application layer through smart contracts.
- We will evaluate the new language with existing agent-oriented languages aspect of performance, usability, fault tolerance, adaptability, and cost of communication between agents.
- We will reduce the overhead of software agent design, analysis, and implementation for the belief-desire-intention framework by proposed domain-specific language.

2 Background

Although software agents are not a new concept, there has not been found specific definition regarding what exactly should be. In essence, an agent can be either physical, software-based, or a combination of them. This kind of feature brought us to define that the software agents must be in a new category. In this study, we will implement software agents with role-oriented programming that works with role-constraints, role-invariants, role-relationship, and compartments using smart contracts in blockchain technology.

Role-based systems are autonomic systems, which means that the role-based multi-agent systems design planned capabilities and collaborative skills to delegate tasks to components [16]. Roles are an abstract concept of objects that can be transferable between software agents; however, liveness is much longer than an object. Moreover, roles can have compartments that consist of states of roles, contexts in a software agent, and events. A software agent can connect with other software agents. While an agent is transferring a message to other agents, it should have two unique features which are:

– **Self-awareness**: States and context can be adaptable according to the environment. Software agents should adjust their contexts according to the environment.
– **Self-configuring**: When a new software agent has joined into the network, the agent should configure and reconfigure itself.

Context-dependency refers to the self-awareness and self-configuring definitions to provide agent awareness in a dynamic and high-flexible environment in multi-agent systems. Roles can be assigned to a specialized context and one can use multiple contexts in multiple compartments. Since contexts are strictly bounded by runtime evaluation, design-by-contract can be more useful than test-driven development which is the compile-time metaprogramming feature.

The idea of usability of the blockchain technology for multi-agent systems will be tested and implemented with this study. Commonly, trust and privacy should be provided by external components, application programming interfaces, or other intrusive technologies. In this study, we want to implement the application layer of blockchain so that one can easily employ a multi-agent system in a non-intrusive secure decentralized computing platform.

Design-by-Contract (DbC) is a software testing and correctness methodology. Principally, it uses preconditions, postconditions, asserts statements, and invariants. However, general-purpose language-based creates heavyweight code dependency while realizing the design-by-contract approach. A domain-specific language for this purpose is an elegant way to implement unit testing with test-driven development. In the aspect of the multi-agent systems, contracts may have states and changeable contexts so that developers can apply the design-by-contract into the blockchain-based domain-specific language. Actually, this programming approach is called defensive programming, because the application is responsible for figuring out what has occurred an error in postconditions or preconditions. For instance, when an agent planned a goal in a collaborative relationship, we should decide assumptions (precondition) and the effect of these assumptions (postcondition) that are valid. In this case, the effect should be the agent's goals. If the procedure that has been defined as a precondition is executed correctly, then it will terminate successfully to complete the given goal as achieved. Normally, there are a couple of ways to do it, but we will use Solidity stateful blockchain language to do so.

Embedded domain-specific language stands for incorporating a domain-specific language in a general-purpose language such as Java, Scala, or Kotlin. Despite that it restricts language extensibility, in this study, we will use the advantage of a host language such as annotation-based code generation, runtime, or compile-time metaprogramming.

Last but not least, even though the solution consists of different types of languages such as an agent communication language, general-purpose language, and smart contract language, we believe that the embedded domain-specific language can ensure the design and analysis, and implementation layer compact in terms of agent deployment. We will reduce the complexity of the design, analysis, and implementation layer as much as possible and also maintain existing agent-oriented programming languages at the implementation layer.

3 Methodology and Expected Outcome

In the Ph.D. journey, we will focus on the different domain-specific language approaches. To this end, we will use the simplified methodology called Prometheus [17] methodology as shown in 1. We will separate the research into layers to create a simple application.

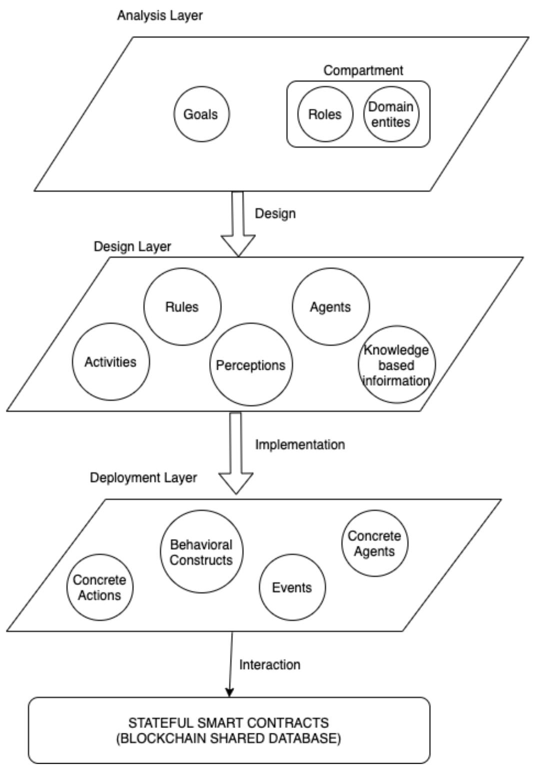


Fig. 1. Concept of Role-based Blockchain-Enabled Agent Programming

In the Analysis Layer, we should define domain entities and roles regarding the application. This will be likely a planning phase for large-scale applications when each agent involves in the network, there will be a dynamic model to create roles, goals, and domain entities. This step is a necessity to abstract computation behavior for realizing as software agents or programs. In the analysis layer, we need to implement the role-object pattern for agent applications because role-oriented languages have no direct connection with object-oriented applications.

We will design the agent features without considering the environment that is uncertain, unpredictable, time-sensitive, and highly dynamic. Having identified these abstraction characters of the planned environment, let us give a desirable definition of agents to perform in the implementation layer. The design and Analysis layer is a kind of requirement engineering phase because we should understand and specify the requirements of the given environment through a dynamic model. As shown in Figure 1, we can assign rules, activities, and perception features with an agent object so that we can implement role-oriented programming features such as constraints and relationships between agents. Agent rules can be assigned with beliefs (Information about the environment) and desires (agent's wishes), which means that the agent can define objectives by starting from an internal state to accomplish a goal.

Deployment of a software agent contains fundamental features such as communication languages, software components, users, hardware elements connecting to software agents. The deployment layer is closely related to the implementation layer of software agents, which is why practical frameworks follow up the design pattern that provides a recurring solution in a particular design problem. Design patterns such as Belief-Desire-Intention (BDI) or Reactive Agent Frameworks have no restrictive specifications and it has not been implemented with the design-by-contract approach. However, most of the agent-oriented practical frameworks have followed the BDI approach, but agent frameworks do not have to be dependent on the BDI approach. Agents should be synchronized with behaviors that show concurrent operations such as atomicity, thread prioritization, and lock-based synchronization.

As shown in Figure 1, agents may have interaction with a database in order to keep data in persistent storage. Blockchain technology provides smart contracts (programmable code snippets that work in the blockchain database) and database functionality in a deterministic way. Determinism means that a copy of a particular blockchain database should work in the exact same way in another environment. The database can accept stateful (Turing complete) or stateless (non-Turing complete) smart contracts to operate transactions from agent applications. Once an agent triggered action, the action performs a transaction into the blockchain database. Security and privacy of agent smart contracts can be provided by Merkle trees that present zero-knowledge proof and verifiable data structure.

Agent-Oriented Libraries and Frameworks can be applied to role-object pattern in order to connect between agents's and roles' world. In this proposal, we implement role and natural types in a stateful contract with a contract wrapper as below:

4 Proposed Solutions

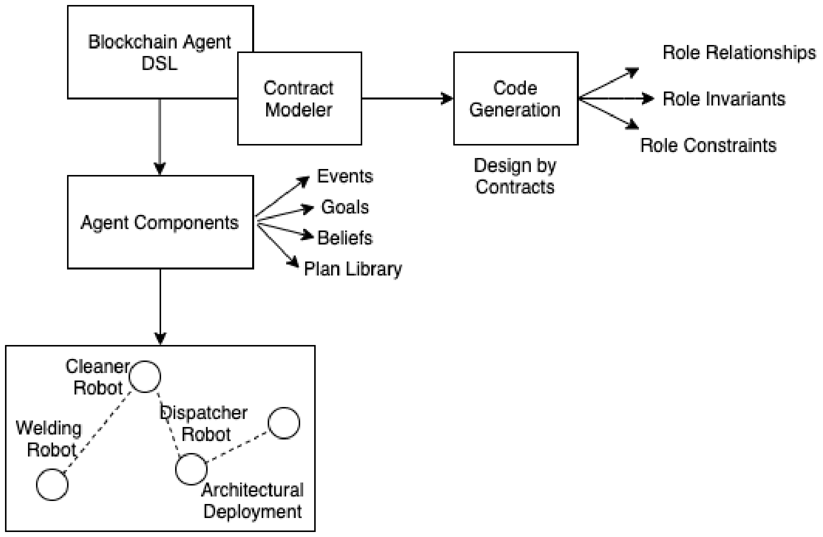
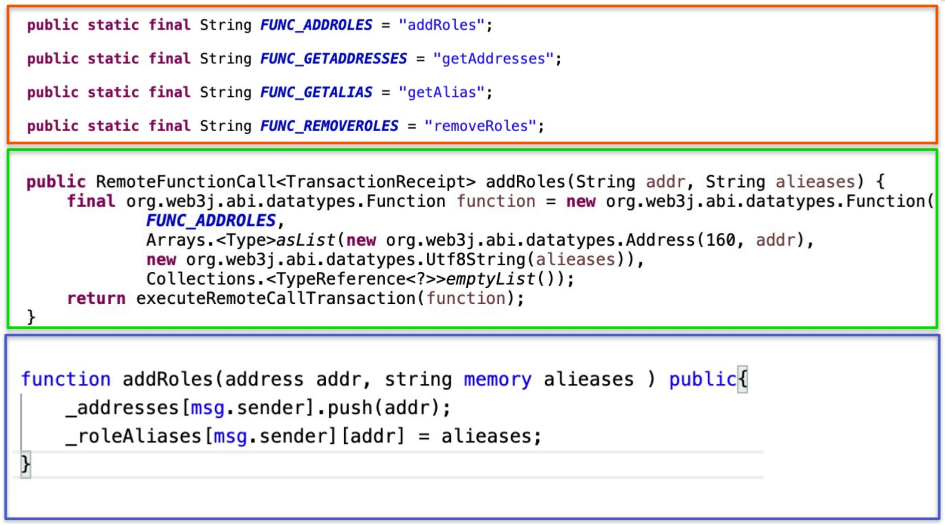


Fig. 2. System Architecture of Role-based Blockchain Enabled Language

In order to ensure all kinds of functionalities in Figure 2, we create a domain-specific language runtime interpreter with existing technologies. In the first step of implementation of the proposed language, we would like to focus on a stateful contract language, which is called Solidity, Agentspeak with Jason, and parser generator such as Another Tool for Language Recognition (ANTLR). We are planning to do language translation with various technologies such as ANTLR so that we will create the Abstract Syntax Tree to transform into the Concrete Syntax Tree. Due to the nature of agent-oriented programming, it seems to have a necessity of runtime-metaprogramming, which is allowing us to generate code from meta-objects at the runtime, we might have a solution with compile-time metaprogramming. In this case, one of the biggest challenges is to select general-purpose language because the language can identify annotations either is in a runtime-time or compile-time role type checking.

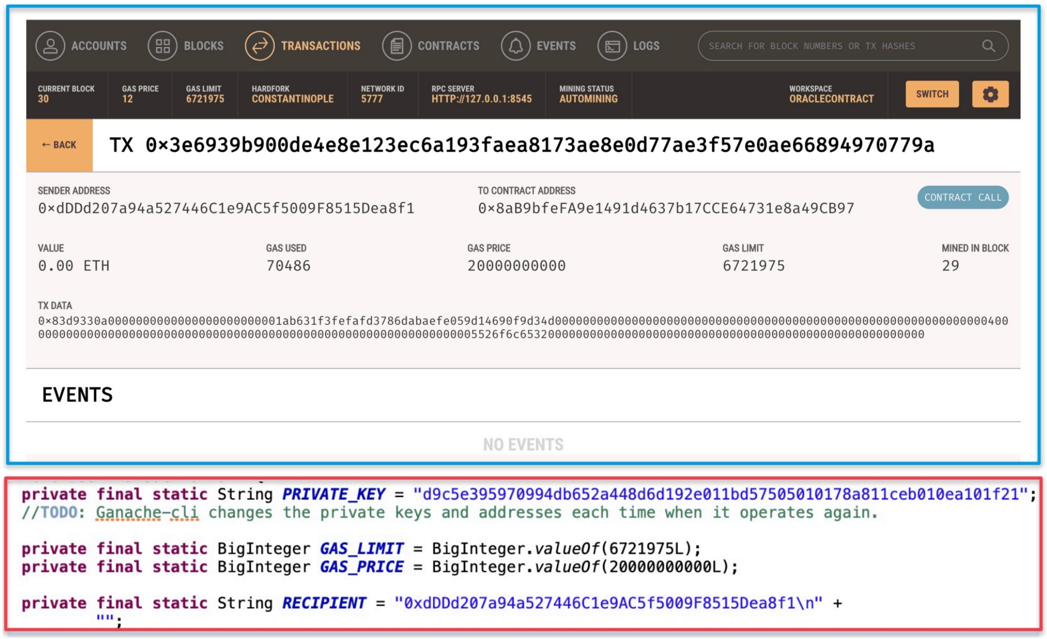


```
public static final String FUNC_ADDROLES = "addRoles";

public static final String FUNC_GETADDRESSES = "getAddresses";

public static final String FUNC_GETALIAS = "getAlias";

public static final String FUNC_REMOVEROLES = "removeRoles";
```

```
public RemoteFunctionCall<TransactionReceipt> addRoles(String addr, String alieases) {
    final org.web3j.abi.datatypes.Function function = new org.web3j.abi.datatypes.Function(
            FUNC_ADDROLES,
            Arrays.<Type>asList(new org.web3j.abi.datatypes.Address(160, addr),
            new org.web3j.abi.datatypes.Utf8String(alieases)),
            Collections.<TypeReference<?>>emptyList());
    return executeRemoteCallTransaction(function);
}
```

```
function addRoles(address addr, string memory alieases ) public{
    _addresses[msg.sender].push(addr);
    _roleAliases[msg.sender][addr] = alieases;
}
```

Fig. 3. Code Snippet from Role-Object Pattern in a Stateful Smart Contract



```
CURRENT BLOCK  GAS PRICE  GAS LIMIT  HARDFORK        NETWORK ID  RPC SERVER             MINING STATUS  WORKSPACE
30             12         6721975    CONSTANTINOPLE  5777        HTTP://127.0.0.1:8545  AUTOMINING     ORACLECONTRACT

TX 0×3e6939b900de4e8e123ec6a193faea8173ae8e0d77ae3f57e0ae66894970779a

SENDER ADDRESS                                TO CONTRACT ADDRESS
0×dDDd207a94a527446C1e9AC5f5009F8515Dea8f1    0×8aB9bfeFA9e1491d4637b17CCE64731e8a49CB97

VALUE      GAS USED    GAS PRICE      GAS LIMIT    MINED IN BLOCK
0.00 ETH   70486       20000000000    6721975      29

TX DATA
0×83d9330a0000000000000000000000001ab631f3fefafd3786dabaefe059d14690f9d34d00000000000000000000000000000000000000000000000000000000000000400
000000000000000000000000000000000000000000000000000000000000005526f6c6532000000000000000000000000000000000000000000000000000000

EVENTS

NO EVENTS
```

```
private final static String PRIVATE_KEY = "d9c5e395970994db652a448d6d192e011bd57505010178a811ceb010ea101f21";
//TODO: Ganache-cli changes the private keys and addresses each time when it operates again.

private final static BigInteger GAS_LIMIT = BigInteger.valueOf(6721975L);
private final static BigInteger GAS_PRICE = BigInteger.valueOf(20000000000L);

private final static String RECIPIENT = "0xdDDd207a94a527446C1e9AC5f5009F8515Dea8f1\n" +
        "";
```

Fig. 4. Data from the Merkle Hash Tree in the Blockchain

The red rectangle In Figure 3, the address aliases that have been specified for smart contract functions need to call contract wrappers that were written in a general-purpose language. The blue rectangle in Figure 3 shows a procedure from a stateful smart contract that adds address and provides constraints checking. The green rectangle in Figure 3 demonstrates a contract wrapper in a general-purpose language in order to control contract address from Java language. In this example, we would like to simulate role attributes by adding role types into a smart contract. Role types and natural types can be represented in the object-oriented data structures and we can use smart contract addresses in order for reaching out to contracts in the blockchain consensus network. Smart contracts provide trust and security because the data will be shown as below in Figure 4. TX value represents a transaction in the blockchain that has been processed by one of the smart contracts. Private keys are assigned to accounts as shown in Figure 4 and developers can use accounts like a shared memory to realize limited concurrent applications. In the end, all values are in a Merkle tree through the hashed data structure. Moreover, one of the important security features of a distributed system is the single point of failure can be prevented by gas costs. A typical gas cost consists of an operation and a transaction cost that can prevent the consumption of general system resources to the end.

5 Limitations

In this section, we will list our limitations regarding the process of writing the thesis.

- We will present examples regarding autonomous and collaboration features. In context with collaboration, supply chain simulation between participants would be enough. As for the autonomous feature, robotic motion planning can be simulated with our proposal.
- We will focus on the existing meta-model such as CROM for specifying communicative entity types. In the context of CROM research, we will follow the guideline regarding roles and compartments that have been specified before.
- We will develop an application based on a stateful smart contract language such as Solidity the following design-by-contract approach that interacts with different agents in the context of role-oriented programming.
- This research is limited to the KQML and FIPA agent communication languages and it does not comprise stateless blockchain language. Due to the nature of the stateless blockchain language, it does not purely suitable for the object-oriented approach.
- As for ontological representation, semantic heterogeneity between agent-oriented frameworks will not be taken into consideration. So we basically will handle existing ontologies and will not advance to ontology engineering.

6 State of the Art

When we conduct a literature review, we have been investigated the following two literature research questions (LRQ):

- *LRQ 1*: How can Blockchain and Multi-agent System improve each other?
- *LRQ 2*: How does role-based programming affect collaborative multi-agent systems?
- *LRQ 3*: Can the context of agents be an affiliated aspect of role-oriented programming through blockchain technology in multi-agent systems?

ALAADIN is one of the oldest metamodels to define models of organizations for agents and this model defines a very simple description of coordination and negation schema [18]. The authors determine that the role is an abstract representation of an agent or service function within a group. Groups are a set of features that behave as an atomic entity so that an agent dynamically joins, creates, or leaves groups [19].

When we focus on behavioral roles for agent interaction, (Cabri et. al. 2003) proposed that an agent system defines a role as a set of capabilities and expected behaviors. BRAIN is an approach that covers a role-based interaction model, where agents' interactions and behaviors are embedded in roles [19]. Moreover, they achieved and advise to realize agent-oriented features, separation of concerns, and reuse of solutions [20]. To describe agents semantically, they defined a language called XRole that exploits built-up definitions of roles. These definitions consist of name, description, addresses, role description, and contents of the agents with relational features such as *MinOccurs* and *MaxOccurs*. RoleSystem is an interaction infrastructure that implements the model of BRAIN [21]. Roles defined by XRole can be read by humans as well as by agents and tools [21]. The RoleSystem provides two main components which are: *reqRegistration*, to register an agent in the system with a specified role; *searchForRoleAgent*, to search for agents playing a given role between agents and server agents [21].

The planning capability of multi-agent systems is one of the key features that the blockchain should take care of it. After assigning roles, plan execution of the multi-agent systems should complete distributed ordering actions. To do so, a smart contract can be used which are essentially collections of distributed code and data representing some business logic that works with the blockchain distributed consensus protocols [22]. The main idea of this paper is to coordinate the steps of multi-agents through the smart contracts aspect of distributed plan execution. In this plan execution, multiple smart contracts can be used such as oracle contract, which is allowing to exploits data in the off-chain storage, or contract of preconditions and postconditions to provide the design-by-contract pattern.

Gaia is one of the methodologies at the design and analysis phases in multi-agent systems. The main goal of this methodology is to model multi-agent systems for an organization where different roles interact [19]. The Gaia methodology defines the features of roles as below:

– *Responsibilities*: They specify the functionalities of agents that play roles.
– *Permissions*: They are a set of rights associated with roles in which agents play.
– *Activities*: Internal computation of an agent. This does not take into consideration the relationship between agents.
– *Protocols*: This is related to interaction roles indicating agent-to-agent communication.

The role-based evolutionary programming (RoleEP) presents cooperative mobile agents to collaborate in achieving a common goal [19]. The authors of RoleEP state that an object becomes an agent by binding itself to a role that is defined in a dynamic environment [5]. The authors have defined the basic concept as below [5].

– *Environment:* An environment is composed of environment attributes, methods of environment, and roles.
– *Roles:* A role, which can move between hosts that exist in an environment, contains role attributes, role methods, and binding interfaces.
– *Objects:* An object, which cannot move between hosts, is composed of attributes and methods.
– *Agents:* An object or mental identity that binds itself with some roles and acquires traveling/collaboration functions.
– *Binding Interface:* A binding interface, which looks like an abstract method interface, is used when an object binds itself with a role.

Implementation of a domain-specific language may have metamodel design paths at the level of M1 (User Model), M2 (Unified Modeling Language), M3 (Meta Object Facility). For instance, AgentDSL is a domain-specific language for crosscutting concerns for agents, which is supporting aspect-oriented programming, and non-crosscutting concerns [23]. The authors of AgentDSL defines a code generator that maps abstractions.

7 Research Plan

During this research, we will try to answer the research questions that we have asked in the Research Problem.

– In the first year, we will deal with the design and analysis phase from the previous studies that have been conducted by various researchers from the department of Role-based software infrastructures for continuous-context-sensitive-systems (ROSI) at TU Dresden. Roles, compartments, negotiation, and collaboration parameters will be defined and domains of case studies may expand or narrow down.

- In the second year, the design of agent architecture, topology, and sample applications of the domain-specific language will be proposed. A prototype will be shown in accordance with the supervisorship' requirements.
- In the third year, the implementation layer will be completely finished, and then we will agree on a final version of the thesis with the supervisor. If we have enough time at this stage of the research, we will develop the implementation further for the practical solution with regard to the robotic applications.

At the end of our Ph.D. journey, it is believed that developers or experts can implement role-based agent-oriented applications in the blockchain network by having an embedded domain-specific language.

The complexity of this research can easily increase because a couple of approaches should be used in the end. However, we have limited the approach with collaborative software agents either can work on hardware solutions or enterprise applications.

8 Case Study

Manufacturing scheduling is the process of assignment of timing for order, manufacturing, and delivery. So we should provide a good quality per unit and the number of units should be maximized per slot in the production line. At the same time, we need to minimize the waste of resource requirements and potential failures. Moreover, the designed system sometimes collaborates with human operators because they need to get involved in some complex problems by collaborating with robotic cells [24].

Another case study that we want to focus on the collaborative multi robots scenario. Let us assume we have two robot agents and a human agent. The human agent should work with two robot agents. The first robot agent will do actions picking material from an assembly line, finding the next slot, dropping the material, and moving towards a new position, respectively. So the second robot agent will just do action welding with the material into some raw good. Human-agent is going to check the material quality before welding it. Welding and moving can be goals for us and they need to have preconditions and postconditions. The actions of robots are pick(), find(), drop(), and goal of the robots can be the result of checking slots. We will put the goals and actions into the smart contract language with their data and then we will evaluate in terms of preconditions and postconditions. We can use these features in a domain-specific language that has been generated from an agent communication language, a general-purpose agent framework, and a stateful smart contract language.

9 Result of the Research Study

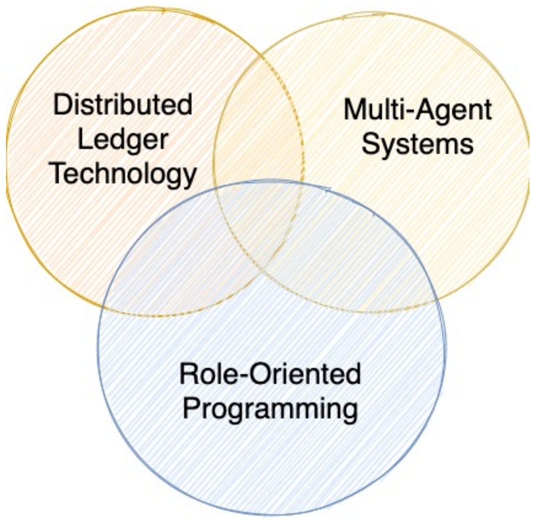


Fig. 5. Research Overview regarding different approaches

As readers can see in Figure 5, we are bringing together three fields, which are blockchain technology (BCT), multi-agent systems, and role-oriented programming, into a domain-specific language. As shown in Figure 5, we have demonstrated blockchain technology as distributed ledger technology. Aside from it being implementation-agnostic, a distributed ledger can be in a form of private, public, federated, or consortium networks.

In the distributed ledger part, we have focused on smart contract development that can give us the ability to develop an application in a decentralized environment. Multi-agent systems are suitable for the decentralized environment and they can use role-oriented attributes. Contract-oriented approach with smart contracts can benefit from role-orientation, which principally represents relations in a given model implementing an embedded domain-specific language. The language supports the following distinctive attributes:

- Agents that were created by the template language can connect to each other in a peer-to-peer manner.
- Agents can behave autonomous and partly proactive by having relationships between roles. In addition, applications that have programmed by the domain-specific language is easy to use by making certain of obtaining belief-oriented architecture and action-oriented program.

– Agents can collaborate and develop social behavior in the context of role-oriented programming by keeping the data secure with smart contracts.

10 Conclusion

The paper addressed the challenge of compelling trust and security in multi-agent systems and their role-oriented features by realizing smart contracts regarding blockchain technology (BCT). The main purpose of the research is to give a new approach to the intersection between smart contract programming, role-oriented programming, and agent-oriented programming. The course of findings among various research areas guides us to design a domain-specific language to contribute to the multi-agent system area.

– There is no common understanding in terms of multi-agent system methodology, analysis, design, or implementation. This increases the complexity of the research in the multi-agent system area.
– The limited number of domain-driven agent-oriented languages have been provided so that one can notice that multi-agent system research is most likely conceptual and it does not provide prototype and result-evaluated research.
– Synthesized metamodeling from scratch in different research areas can be ambiguous; thus, we believe that embedded domain-specific language with blockchain can solve most of the problems for multi-agent systems.
– Role-oriented programming with smart contracts is challenging because the choices of technology can affect the result of the study. For instance, stateful and stateless contracts are not advanced technologies that can employ all of the features of the object-oriented paradigm. Turing complete and non-Turing complete technologies will be scrutinized in future work.

In a nutshell, this paper presented a new significant role approach with smart contract programming implementing hash data structure and providing data security regarding roles. Presenting our approach will simplify the application development process for further researchers.

11 Future Work

In future work, a tool will be developed for a role-based multi-agent system. This tool includes an annotation processor and template-based code generator for agent behaviors. By selecting a general-purpose language, the system will be evaluated with qualitative and quantitative tools. Smart contracts will be generated through annotation processing with customized annotations and agents will be generated with a template-based code generator tool for one of the selected frameworks which have been presented in the introduction section.

Acknowledgements

The author would like to thank his supervisors, Prof. Dr. Uwe Aßmann, and Prof. Dr. Susanne Strahringer, for the patient guidance, encouragement, and comments they have provided to shape his doctoral vision. This work is funded by the German Research (DFG) within the Research Training Group Role-Based Software Infrastructures for continuous-context-sensitive Systems (GRK 1907, TU Dresden, Software Technology Group, Nöthnitzer Straße 46, 01187, Dresden).

References

1. Y. Shoham, "Agent-oriented programming," *Artificial Intelligence*, vol. 60, no. 1, pp. 51–92, 1993. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0004370293900349
2. H. Zhu and M. Zhou, "Role-based multi-agent systems," *Personalized Information Retrieval and Access: Concepts, Methods and Practices*, 01 2008.
3. T. Kühn, M. Leuthäuser, S. Götz, C. Seidl, and U. Aßmann, "A metamodel family for role-based modeling and programming languages," in *Software Language Engineering*, B. Combemale, D. J. Pearce, O. Barais, and J. J. Vinju, Eds. Cham: Springer International Publishing, 2014, pp. 141–160.
4. T. Kühn, S. Böhme, S. Götz, and U. Aßmann, "A combined formal model for relational context-dependent roles," in *Proceedings of the 2015 ACM SIGPLAN International Conference on Software Language Engineering*, ser. SLE 2015. New York, NY, USA: Association for Computing Machinery, 2015, p. 113–124. [Online]. Available: https://doi.org/10.1145/2814251.2814255
5. G. Cabri, L. Ferrari, L. Leonardi, and F. Zambonelli, "A survey about role-based interaction proposals for agents," 01 2005.
6. B. Ferreira and A. M. Leitão, "Context-Oriented Algorithmic Design," in *7th Symposium on Languages, Applications and Technologies (SLATE 2018)*, ser. OpenAccess Series in Informatics (OASIcs), P. R. Henriques, J. P. Leal, A. M. Leitão, and X. G. Guinovart, Eds., vol. 62. Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2018, pp. 7:1–7:14. [Online]. Available: http://drops.dagstuhl.de/opus/volltexte/2018/9265
7. K. Wan and V. Alagar, "A context-aware trust model for service-oriented multi-agent systems," vol. 5472, 04 2009, pp. 221–236.
8. F. Bellifemine, G. Caire, and D. Greenwood, "Developing multi-agent systems with jade," *Developing Multi-Agent Systems with JADE*, pp. 1–286, 02 2007.
9. A. Pokahr, L. Braubach, and W. Lamersdorf, *Jadex: A BDI Reasoning Engine*. Boston, MA: Springer US, 2005, pp. 149–174. [Online]. Available: https://doi.org/10.1007/0-387-26350-0_6
10. R. Bordini, J. Hübner, and M. Wooldridge, *Programming Multi-Agent Systems in AgentSpeak Using Jason*, 10 2007, vol. 8.
11. K. Hindriks and J. Dix, *GOAL: A Multi-agent Programming Language Applied to an Exploration Game*, 03 2014, vol. 9783642544323, pp. 112–136.
12. M. Winikoff, *Jack^{TM} Intelligent Agents: An Industrial Strength Platform*, 01 2005, pp. 175–193.
13. O. Boissier, R. H. Bordini, J. F. Hübner, A. Ricci, and A. Santi, "Multi-agent oriented programming with jacamo," *Science of Computer Programming*, vol. 78, no. 6, pp. 747 – 761, 2013, special section: The Programming Languages track at the 26th ACM Symposium on Applied Computing (SAC 2011) & Special section on Agent-oriented Design Methods and Programming Techniques for Distributed Computing in Dynamic and Complex Environments. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S016764231100181X
14. M. Dastani, F. Dignum, and J.-j. Meyer, "3apl: A programming language for cognitive agents," 01 2003.
15. M. Dastani, "2apl: a practical agent programming language," *Autonomous Agents and Multi-Agent Systems*, vol. 16, no. 3, pp. 214–248, Jun 2008. [Online]. Available: https://doi.org/10.1007/s10458-008-9036-y
16. H. Zhu, "Role-based autonomic systems," *IJSSCI*, vol. 2, pp. 32–51, 01 2010.
17. R. Bordini, M. Dastani, and M. Winikoff, "Current issues in multi-agent systems development," vol. 4457, 09 2006, pp. 38–61.
18. *Aalaadin*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 169–179. [Online]. Available: https://doi.org/10.1007/3-540-26815-4_8
19. G. Cabri, L. Leonardi, L. Ferrari, and F. Zambonelli, "Role-based software agent interaction models: A survey," *Knowledge Eng. Review*, vol. 25, pp. 397–419, 12 2010.

20. G. Cabri, L. Leonardi, and F. Zambonelli, “Implementing role-based interactions for internet agents,” 02 2003, pp. 380– 387.
21. Cabri, Giacomo and Leonardi, Letizia and Zambonelli, Franco, “Brain: A framework for flexible role-based interactions in multiagent systems,” vol. 2888, 11 2003, pp. 145–161.
22. A. Shukla, S. K. Mohalik, and R. Badrinath, “Smart contracts for multiagent plan execution in untrusted cyber-physical systems,” in *2018 IEEE 25th International Conference on High Performance Computing Workshops (HiPCW)*, 2018, pp. 86–94.
23. U. Kulesza, A. Garcia, C. Lucena, and P. Alencar, “A generative approach for multi-agent system development,” vol. 3390, 05 2004, pp. 52–69.
24. A. Bauer, D. Wollherr, and M. Buss, “Human-robot collaboration: a survey,” *Int. J. Humanoid Robotics*, vol. 5, pp. 47–66, 2008.

Author

Orçun Oruç received M.Sc. from TU Chemnitz, and he graduated from Kocaeli University with a B.Sc. degree. Currently, he is pursuing his Ph.D. in Computer Engineering-Software Technology at the Dresden Technical University. His research interests include programming languages, multi-agent systems, role-oriented programming, natural language processing, decentralized, and distributed applications.