Enterprise Design, Operations and Computing with AI Agents: Accountability using DSL

Zoran Milosevic[1][0000−0002−1364−7423] and Igor Dejanović[2][0000−0002−0414−1455]

[1] Deontik, Australia zoran@deontik.com
[2] Faculty of Technical Sciences, University of Novi Sad, Serbia igord@uns.ac.rs

Abstract. LLM-powered AI agent systems are bringing new perspectives to enterprise design, operations, and computing (EDOC), particularly in environments where agents can act autonomously and independently, referred to as agentic systems. These capabilities unlock new opportunities for automation, enabling agents to perform intellectually demanding tasks that were previously reserved for humans, while still allowing human oversight for key decisions. However, a critical challenge remains: ensuring clear accountability within these systems across humans and AI Agents, which may involve complex chains of authorization and delegation. As individual agents act independently, pinpointing responsibility becomes increasingly difficult. This paper proposes a novel solution to this problem: a domain-specific language (DSL) based on the ISO ODP Enterprise Language standard, precisely defining the roles and interactions between actors in the enterprise landscape. The DSL is implemented using textX, a contemporary tool-chain which provides rapid prototyping ecosystem. The aim is to provide user-friendly syntax while following the precise semantics of ODP enterprise language.

Keywords: AI Agents · DSL · Policy · Responsible AI · Obligations · ODP.

1 Introduction

AI Agents, including Generative Agent systems [22] are transforming the way enterprises function by automating complex tasks and enabling real-time decision-making. By leveraging Large Language Models (LLMs), these systems foster enhanced communication and collaboration between multiple intelligent agents. This collaboration unlocks the potential for novel and efficient solutions to complex enterprise tasks. However, a critical challenge remains: ensuring a clear chain of responsibility within these multi-agent systems. As these systems become more sophisticated, and individual agents exhibit increasing levels of agency (acting autonomously within their environment), pinpointing accountability for actions and decisions becomes increasingly complex.

While significant previous research has focused on communication protocols and coordination mechanisms within multi-agent systems even prior to their use of LLMs [4,9], ensuring a clear chain of responsibility has received less attention. Existing approaches often rely on centralized control structures

or pre-defined rules for task allocation, without considering the intricacies of social environment, such as accountability arising from the organisational or legislative rules. These methods do not provide flexibility in adapting to the dynamic nature of enterprise environments and the growing autonomy of agentic systems.

This paper addresses this critical gap by proposing a novel solution: a domain-specific language (DSL) developed to support enterprise design and operation activities, while placing special attention on the accountability concepts. This DSL can be used to express accountability rules for the enterprise which includes AI agents and multi-agent capabilities, and provide rapid implementation of required monitoring and enforcing capability. This DSL leverages the precise semantics of the ODP Enterprise Language (ODP-EL) standard [10], allowing for a clear and unambiguous definition of roles, responsibilities, and decision-making authority within a multi-agent system. The ODP-EL standard brings credibility but also pragmatic approach to building interoperable distributed systems and by utilizing this DSL, enterprises can design and deploy collaborative AI systems with a well-defined chain of responsibility, fostering trust, transparency, and legal clarity.

Next section provides an overview of related work. This is followed by a brief introduction of multi-agent and agentic AI systems, section 3. Key ODP Enterprise Language concepts are outlined in section 4. This is followed by the description of our DSL implementation of related ODP-EL concepts in section 5. Section 6 provides an overview of the integration of DSL tooling with one specific AI agent architecture and discusses some challenges from our implementation. Conclusions and directions for future work are summarized in section 7.

2 Related Work

The problem of designing enterprise structure and behaviour has been subject of many research and industry efforts. Much of the recent research contributions come from various enterprise ontology efforts, the most prominent being a series of proposals from the UFO community, of which UFO-L extension is particularly related to this work [8]. Another example is Open Digital Rights Language (ODRL) [23], which is used to represent permitted and prohibited actions over a certain asset, as well as the obligations required to be met by stakeholders. These ontologies and modelling languages provide an excellent conceptual foundations for expressing key concepts and relationships related to enterprise structure, and have provided insights into our current work.

Further influence comes from our earlier DSL efforts related to Business Contract Language (BCL) [15,17], which was based on the previous version of ODP-EL standard. BCL has similarity with our current DSL efforts, in terms of its focus on developing an implementable language. Current ODP-EL standard however provides more expressive set of concepts which includes a precise framework of expressing delegations and obligations in a way which are more

amenable for the distributed system implementations. This, along with the increasing requirements to better modelling of policy frameworks in digital health, where much of our recent industry efforts were involved, motivated us to use the latest version of he ODP-EL as a source of our DSL.

Much of the work above is related to the enterprise objects that model key entities in a system, of which computational agents represent subset of these. The agentic AI solutions are recent development and much focus there was on how to best structure an LLM-based application in terms of the roles undertaking dedicated tasks, with limited current success in supporting planning and collaboration [21]. AI agent architectures are currently concerned with the structure of single AI agents, leaving the communications among agents to the underlying LLM and relevant tools, to support the expression of agent's reflection, planning and collaboration, as demonstrated in the recent work on computational software agents that simulate believable human behavior [22]. Some earlier work has investigated the computational agents in a distributed environment, as for example the Siebog Multi-Agent System [19] where agents can be specified using ALAS DSL [24].

However, the problem of expressing policy constraints over agent's actions in distributed systems is currently not addressed and our contribution is to provide architectural positioning of the formalism of deontic and accountability concepts to be integrated with agentic developments, leveraging mature and pragmatic framework from the ODP-EL.

3 Agent AI systems

3.1 Agent AI architectures

Agent AI systems have recently emerged as a vehicle of making better use of LLM models in support of specific, AI enabled enterprise tasks. They allow complex tasks to be decomposed into smaller units, i.e. actions, such as in complex activities of writing software or providing financial or health related advice to consumers. Each of these actions can in turn be implemented by a separate, dedicated agent. The actions can include prompting of LLM models in an iterative fashion, invoking tools for specific functions by a single agent, or multiple agents. Some architectures, such as crewAI [20] include mechanisms for delegating actions to other agents who would execute more specific tasks or for farm-out other tasks for resource allocation reasons. A key new quality here is to replace the current's LLM's zero-shot prompt with a sequence of steps undertaken by a single or multiple agents, sometimes referred to as agent workflow, which as experience shows, produces better results than the zero-shot approach. This task decomposition also influences the properties of the agent AI architectures, recently described in terms of the four design patterns [21], namely:

– Reflection, where a LLM examines its own work to come up with ways to improve it, which may involve creating its own memory stream for this[22].

- Tool Use, where a LLM exploits tools such as web search, code execution, or other functions to help it gather information, take action, or process data.
- Planning, where a LLM creates and executes a multi-step plan, e.g. writing an outline for an essay, then doing online research, then writing a draft.
- Multi-agent collaboration, where multiple AI agents work together, decomposing a complex task and discussing and debating ideas, to come up with better solutions than a single agent would [22].

There are a number of variants of how these patterns can be implemented. In some cases the focus is on their communication and collaboration, with limited autonomy [26,20], while at the other end of spectrum is supporting autonomous decision-making [22]. It should be noted that the LLM-based AI agent architectures are currently not supporting implementations in distributed environment.

3.2 Policy Constraints Considerations

Ethical, regulatory, and policy considerations are crucial aspects to consider when designing any AI system, including the Agent AI systems. These can be summarised in the context of the following requirements:

Ethical Considerations

- Bias and Fairness: it is essential to ensure AI systems are trained on unbiased data and designed to avoid discriminatory outputs. Techniques like fairness checks and diverse datasets are crucial.
- Transparency and Explainability: this is referred to a need to understand how AI systems reach their decisions; for example, in agentic workflows, this might involve explaining the actions and choices of AI agents, and for generative agents, transparency in content generation is important.
- Human Control and Oversight: While agentic AI and generative agents exhibit agency, it is crucial to maintain human control over their actions and outputs. Clear guidelines and safeguards are necessary.

Regulatory and Policy Constraints

- Data Privacy: The Agent AI systems might involve handling personal data. Regulations like GDPR [5] need to be followed when collecting, storing, and using such data.
- Accountability: Assigning responsibility for the actions of AI systems is crucial, such as identifying who is legally accountable for decisions made by agentic AI systems or the outputs of generative agents.
- Safety and Security: Security measures are essential to prevent malicious actors from manipulating AI systems. Safety measures should ensure that AI agents do not pose a risk to users' physical or psychological well-being.

Enterprise Design and Operations Considerations Ethical considerations, regulations, and policies should be integrated from the early stages of the design process for agent AI systems, and the experience with the operations of such system can also inform better integration of policy considerations [16].

Our approach to these requirements is to consider well developed semantics from the ODP-EL in a technology neutral way so that it can be integrated with various architecture approaches. While ODP standard has provided many inputs to various distributed systems and architectures [13], less is known about the expressive power of the ODP Enterprise Language [10], its foundation in deontic concepts, and pragmatic translation of these into implementable software artifacts. Our approach is to express these foundational concepts in a programming language independent framework using modern DSL technologies and tooling, such as Xtext [27] and textX [3], as will be discussed in section 5.

3.3 Human and AI Agent interactions — tracing accountability

As enterprise applications become more complex, they increasingly involve a mix of human and automated actors, including AI agents, as introduced above. While AI's replacement of human actors may not change the essential behavioural characteristics of the tasks performed, this integration raises critical accountability questions, particularly regarding legal responsibility in human-AI interactions.

Humans can take on two roles in the world of AI [17]. They can be AI creators, designing agents to achieve specific goals and deliver value, as for example developers structuring crewAI application. Alternatively, they can be users who engage existing agents to perform tasks on their behalf, potentially even delegating decision-making authority. Delegation is the first class concept in the principal-agent relationship in economics and law, where one party hires another to act on their behalf. We note that it is also possible for AI agents to delegate their tasks to other AI agents, potentially passing authorisation that was created by their originating principals.

The concepts of authorization, delegation, principal and agent are some of the key accountability modelling concepts defined in the ODP-EL standard [10] and built based on the precise expression of behavioural constraints over actions of objects in the system, and we will use them as a basis for our DSL. We note that the ODP-EL, as other ODP languages, are defined in an abstract way, without commitment to any notation [13], although the ODP family of standards include separate, UML based expression, referred to as UML profile for ODP [2]. Next section provides a summary of some of the key ODP-EL concepts which we use within our DSL.

4 Key ODP Enterprise Language concepts

4.1 Community

The ODP Enterprise Language (EL) defines the organizational, business and social context in which an IT system is designed, deployed and operated. The

main structuring concept here is that of a community, which is a grouping of interested parties to collaborate and satisfy their own and the objective of the community. The objective is defined as a "practical advantage or intended effect, expressed as preferences about future states"[10], emphasising the need to express the objective in measurable terms.

Party is defined as an enterprise object modelling a natural person or any other entity considered to have some of the rights, powers and duties of a natural person [10]. Examples of parties include enterprise objects representing natural persons, legal entities, governments and their parts, and other associations or groups of natural persons. It is important to highlight that parties are responsible for their actions and the actions of their agents.

Therefore, parties, such as IT system providers, service providers and customers, along with the automated systems that support their activities, can participate in a community. Note that the ODP uses the term active enterprise object to model an enterprise object that can be involved in some behaviour. So, community is defined through a community contract, which specifies roles in the community and their expected behaviour to be fulfilled by parties or IT systems (i.e. active enterprise objects), along the constrains on that behaviour. These constraints are typically expressed in terms of rules, such as permissions, prohibitions, obligations and authorisations. There can be also rules that apply across several roles in a community such as the constraints that support separation of duty policies.

A complete enterprise language specification would typically involved several communities, which can be nested or federated (this forming a larger community) and parties can fulfill roles in a number of communities. The use of community pattern supports design re-usability, so that many different parties can participate in the community in one or several instances of communities, instantiated from the community contract (as a template). A community can also evolve, by dynamically adding another role or policy rules.

4.2 Deontic tokens

The ODP-EL considers obligations, permissions and prohibitions, known as deontic concepts, as fundamental constraints over behaviour of parties and their semantics is grounded in the deontic logic and normative systems theories. However, with the focus on building implementable distributed systems, the standard takes a pragmatic approach to handling these constraints, such as of the actions of roles in the community, and thus the object that fulfill them. This is done through bringing the concept of a deontic token which encapsulates these deontic constraints as introduced in [14] and further specified in the standard [10].

The holding of the deontic tokens by active enterprise objects constrains their behaviour. This modelling approach provides a basis for manipulating deontic tokens, for example, passing them between parties to model delegations, and activation or de-activation of policies that apply to the active enterprise objects in the context of their enterprise interactions. Three types of deontic tokens

encapsulate deontic constraints. These are called: burden — representing an obligation, permit — representing permission, and embargo — representing prohibition.

In the case of a burden, an active enterprise object holding the burden must attempt to discharge it either directly by performing the specified behaviour or indirectly by engaging some other object to take possession of the burden and perform the specified behaviour. In the case of a permit, an active enterprise object holding the permit is able to perform some specified piece of behaviour, while in the case of an embargo, the object holding the embargo is inhibited from performing the behaviour.

It is to be noted that some actions, referred to as *performative actions*, change the state of the system, such as when one party has authorized another party to do actions on their behalf. In ODP-EL, these actions are referred to as *speech acts*, and they indicate when an action will modify the set of tokens held by the enterprise objects in questions [14,10]. This is the central theme allowing the description of the chain of obligations, permissions or prohibitions across the parties and active enterprise objects, such as AI agents, which we are using in this paper.

We note that the concept of deontic tokens have similarity with, and can be positioned on the widely used security tokens implementations, such as access tokens in Oauth2 [11], and with new OAuth 2.0 Token Exchange specification [12] providing further capabilities for secure exchange of tokens, including support for delegation, as will be discussed in section 6.2.

4.3 Accountability actions

While deontic constraints are important as a way of implementing constraints over system actions, such as for example in access control, or monitor obligations associated with contracts or compliance regulations, there is further benefit in providing high level of abstraction that are more directly related to the expression of social or organizational responsibility. For that purpose, a family of concepts for expressing responsibility is introduced, called accountability concepts. They support traceability of obligations in the overlapping and interacting communities that form the enterprise, allowing linking the rights and responsibilities of parties to the individual system actions and their consequences [13].

The concept of party is introduced above and it is significant to note that parties can have intentions and are accountable for their actions [10]. Those actions that involve accountability, identified by ODP-EL, are listed next.

Authorization is an action indicating that a particular behaviour shall not be prevented. Unlike a permission, an authorization is an empowerment. The fact that an enterprise object has performed an authorization is expressed by it issuing a required permit and itself undertaking a burden describing its obligation to facilitate the behaviour.

Delegation is the action that assigns something, such as authorization, responsibility or provision of a service to another object. The ODP-EL adopts the language from agency theory to refer to the delegated object as an agent, and to

a party that has delegated something (e.g. authorization or provision of service), as a principal. The agent is modelled as an active enterprise object that has been delegated something by, and acts for, a party (e.g. in exercising the authorization, carrying out responsibility). A principal is responsible for the actions of an object acting as an agent. We note that this object can in turn further delegate to another object, if authorised by the principal, thus forming another linked delegation. The first enterprise object in that chain of delegations is the party that is the root of accountability.

Commitment is defined as an action resulting in an obligation by one or more participants in the act to comply with a rule or perform a contract. This effectively means that they will be assigned a burden. Examples include commitments by clinicians to deliver safe, reliable and effective healthcare to patients.

Declaration is defined as an action by which an object makes facts known in its environment and establishes a new state of affairs in its environment. This can, for example, be performed by an AI system (or a party managing it), for example, informing the interested parties about the result of some analysis.

Evaluation is defined as an action that assesses the value of something, which can considered in terms of variables such as importance, preference, usefulness.

Figure 1 shows deontic concepts as primitive constraints over behaviour, and the accountability concepts as an abstraction built on top of deontic concepts.

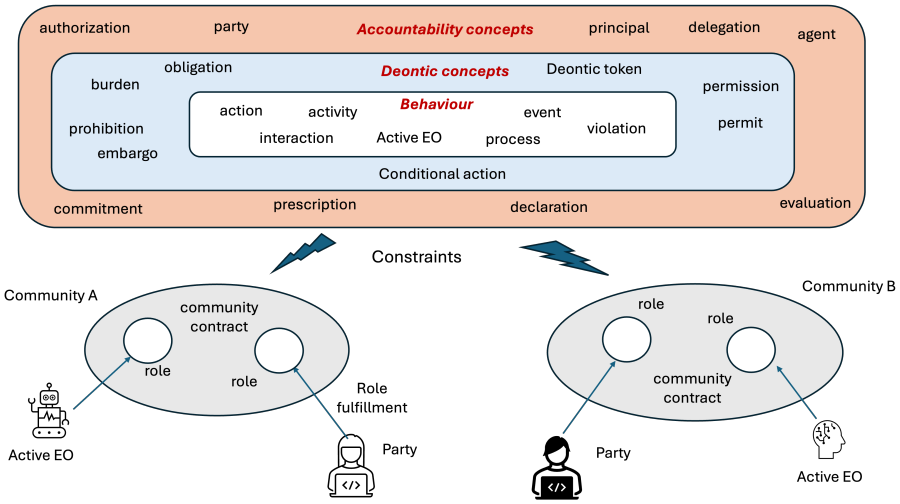

Fig. 1. Accountability and deontic concepts as behavioural constrains.

5 ODP-EL DSL

5.1 Motivations

There are many different type of multi-agent architectures, but common to them is the collaborative aspects of agents and their interactions. The reference architecture based on the ODP community template provides a precise framework for developing guidelines for enterprise design, operations and computing of complex systems which integrate Agent AI components. This is because the ODP community provides the expression of many type of organisational collaborations and interactions, including federations, which can be specified through community contracts. Such contracts also serve as a semantic foundation for a wide range of constraints associated with policies as described in 3.2.

Further, the expression of roles in community supports the description of an expected abstract behaviour to be fulfilled by entities with compatible behaviour. Initially this can be a human, but at a later stage such a function would be implemented by a system, modelled as an active enterprise object, such as an AI agent (not necessarily a LLM agent). A good example of this is a situation in pathology labs, where the pathology technicians analyse manually the blood test results, but much of that can be delegated to a clinical decision support systems (CDS).

This in turn allows for many different multi-agent proposals to be positioned in relation to the reference architecture. Such a reference architecture, benefits from the semantic precision and pragmatic decisions, developed through proven international standardisation processes, and can thus bring confidence to practitioners, system owners, architects and developers .

In our previous work we have proposed a computable policy framework for supporting privacy consent in healthcare [17], using the concept of consent community. This paper uses that framework as a basis for experimenting with the DSL to support a range of deontic and accountability concepts related to consent. The consent use case was chosen as it often includes many complex security level and (cross-)enterprise level rules which are important when designing and implementing interactions across healthcare providers from different organisational domains, with potential use of third-party services from AI vendors, while supporting policy preferences of consumers.

We begin with introducing general consent community, with the community roles of *Grantor*, the individual whose personal data is being requested, and *Grantee*, the individual or organization requesting access to data. The consent community also includes supporting roles for consent management services, including IT specific roles with no direct accountability (e.g. monitor) and policy specific roles which have accountability (e.g. consent policy maker).

The main action by *Grantee* is to submit data access request to *Grantor*, clearly stating the purpose of accessing data. The main actions by *Grantor* are to review and understand data access request details, decide whether to grant or deny access (i.e. authorization through issuing permits) and revoke (withdraw) access to their data at any time, as needed.

5.2 DSL tooling

In the development of the DSL presented in this paper we are using textX. textX is a meta-language for building Domain-Specific Languages (DSLs) in Python. From a single language description (grammar), textX builds a parser and a meta-model (a.k.a. abstract syntax) for the language. Building on top of the Python dynamic nature, textX works as an interpreter. Both the parser and the meta-model are built on the fly during run-time. This enables a quick round-trip from the change in the grammar to the working application.

There are two approaches to designing DSLs:

- Meta-model first approach (or Abstract Syntax first). This approach, also known as top-down, starts with the abstract syntax of the language, and the concrete syntax (or syntaxes) is defined later. This route is followed by so-called projectional editing environments where the Abstract Syntax Tree (AST) is manipulated directly by the user through projections that map ASTs to concrete syntaxes presented to the user.
- Concrete Syntax first. This approach, also known as bottom-up, starts with the concrete textual syntax specified by the grammar from which the meta-model is derived. This approach is popularized by the xText tool and is also used in the textX tool which we use in our implementation.

Both approaches have their pros and cons. A good overview of these approaches is given in [25]. We have chosen the second approach as we find it more suited to our development style and background. Also, this approach is directly supported by the textX tool.

5.3 Method

Our approach here is to design a generic consent architecture following the community contract template to specify key roles, such as grantor and grantee, and use this as a starting point for extending and reifying this generic consent for clinical care purpose and for clinical research purposes.

The approach is based on expressing the key ODP-EL concepts described above in terms of language constructs familiar to the domain experts involved in defining policy rules and constraints, see Listing 1. Ideally, this would involve subject matter experts from the legislative domain but also security policy experts, including those defining access control requirements.

In parallel, we developed the language grammar and meta-model using textX tooling. The grammar and meta-model are based on the semantics of the ODP-EL concepts as they were used in our consent use case, making sure that the meta-model is compliant with the ODP-EL while also adopting certain pragmatic decisions from the language designer perspective.

5.4 Results

This section shows parts of our DSL meta-model and fragments of our consent model created using our DSL.

Listing 1. Generic consent community contract

```
1  community contract genericConsent {
2     objective "Support consumers privacy consent preferences"
3
4     ... <snip>...
5
6     role grantee {
7          action consent_request(data: ConsentRequestData)
8                 emits consent_requested(data: ConsentRequestData)
9     }
10
11    role grantor {
12
13         action review_request(data: ConsentRequestData) [consent_requested(data)]
14                emits consent_reviewed(data: ConsentRequestData)
15
16         authorize give_consent(data: ConsentRequestData)
17             [consent_reviewed(data) && (now - data.subject.birth_date > legal_age)]
18         {
19             permit grant(consent: ConsentRecord) on grantee [this.time + permit_valid]
20
21             burden RespectPrivacy(consent: ConsentRecord) on grantee
22                 triggered by grant_trigger
23                 discharged by [this.time + privacy_valid]
24
25             burden StoreConsentRecord(consent: ConsentRecord) on consentAuthority
26                 discharged by storeConsent(consent: ConsentRecord)
27         }
28
29         declare withdraw_consent (consent: ConsentRecord) {}
30    }
31
32 }
```

Listing 1 shows a community contract implementing a generic consent, which is required in many different communities. This generic consent contract defines the roles of the grantee (line 6), representing parties who ask for consent, and the grantor (line 11), representing parties who are asked to give consent. Each role owns a set of actions that can be executed by a role filling object, if an associated guard expression is satisfied. The guard expression is a logical expression written inside square brackets.

When executed, an action can emit an event. For example, the action `consent_request` on line 7 emits an event `consent_requested` carrying `ConsentRequestData`. This event can be used inside a guard condition to allow an action call only if a specific event has occurred previously. On line 13, we see that the grantor can call the action `review_request` only if the `consent_requested` event has occurred.

In the body of an authorization action (lines 18-27), three deontic tokens are created: a permit `grant` given to the grantee and valid for a duration `permit_valid`, defined by the current policy setting; a burden on the grantee to respect privacy; and a burden on the consent authority to store the consent record.

Finally, the action of withdrawing consent is modelled as ODP-EL declaration action, through which the grantor notify parties and their agents in its environment about this decision (line 29).

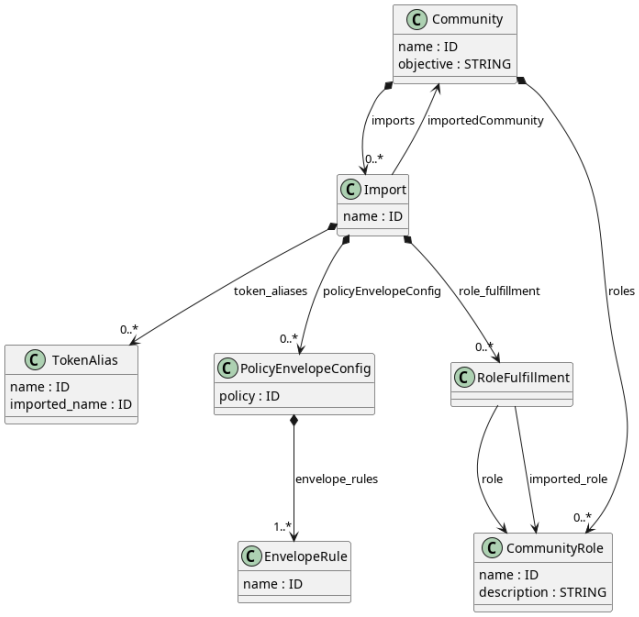

Fig. 2. Language support for community contract imports.

Listing 2. Importing generic consent to a community

```
community HealthCareConsent {
    # Importing and specializing consent authorization for HC community
    # Clinician and Consumer instantiate grantee and grantor respectively.
    # EHR (Electronic Health Record) with patient clinical data is different
    # from consentRecord
    import genericConsent as consumerHealthConsent
           Clinician fulfills grantee              # healthcare provider
           Consumer fulfills grantor               # healthcare consumer
           AccessEHR as grant                      # alias for grant permit
           grant_trigger.envelope = {
                one of [observation_performed, emergency_arrival]
           }

    ...<snip>...
```

In order to use this generic consent contract, the DSL has an import facility whose meta-model is shown in Figure 2. Using the import feature, a community can import generic consent and connect its roles and tokens with the roles and tokens from the generic community contract.

Listing 2 is an example of an import statement that imports `genericConsent` into `HealthCareConsent`, where the role of `Clinician` fulfills the generic role of `grantee`, while the role of `Consumer` fulfills the generic role of `grantor`. Additionally, `AccessEHR` is an alias for the generic permit `grant`. In this community, we model a scenario under which healthcare clinical research is conducted.

It should be noted that our DSL includes the expression of ODP policy concept, as also indicated in this import. ODP policy concept supports the

expression of variability of design choices, that can be anticipated at the design stage and changed at a later epoch. The options available are captured through policy envelope rules. Further details of policy concept are available in [13,10].

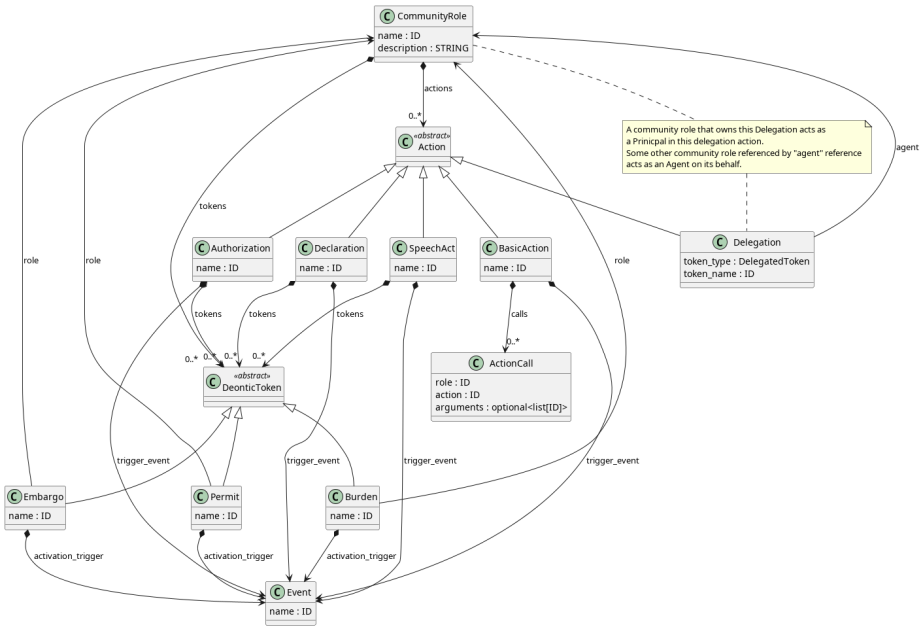

Fig. 3. Accountability and deontic language concepts.

Listing 3. Passing deontic tokens using delegation

```
1   delegate permit AccessEHR(consentRecord: consumerHealthConsent.ConsentRecord)
2     to RecommenderService
3         [consentRecord.grantorPreferences.thirdPartySharing]
```

An example of delegation of permit token in a `HealthCareConsent` community is shown in Listing 3. This delegation is specified as a part of `Clinician` role which makes the clinician a `Principal` in the delegation interaction. At the same time, the receiver of the token, a Clinical Decision Support (CDS) system named `RecommenderService`, becomes the agent of the clinician which acts on their behalf while the responsibility of the agent's actions are still on the clinician which initiated the delegation (Figure 4). The role of the agent is an action-level role, i.e. the role is valid only during the interaction between the clinician and the CDS service.

The principal of the delegation has implicit right to withdraw the permit token at any moment. This principal-agent relation is modelled by the `agent` association between `Delegation` and `CommunitRole` concepts in the meta-model

in Figure 3. The principal is implicitly the owner and initiator of the delegation action, in this case the object fulfilling the `Clinician` community role.

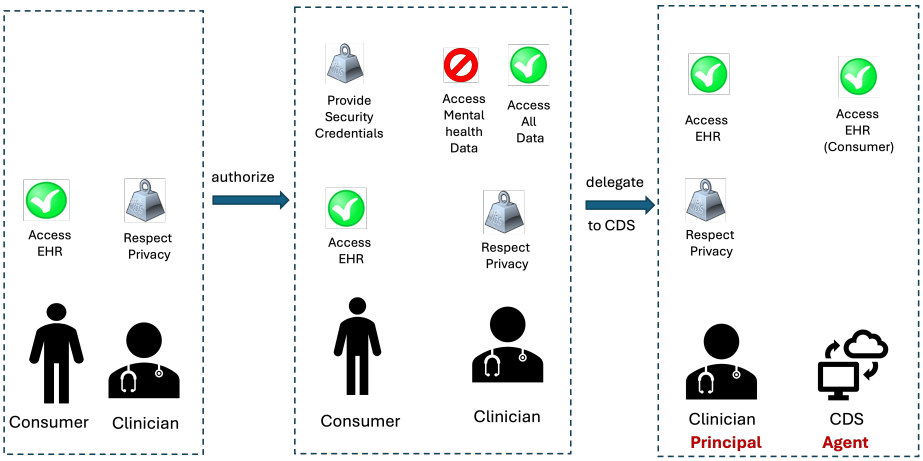

Fig. 4. Token passing and chain of responsibility

Current version of the grammar/meta-model is available on GitHub under MIT license (https://github.com/igordejanovic/ODP-EL-textX).

6 Supporting Agent AI architectures

The DSL proposed provides a generic approach to expressing deontic and accountability constraints over the actions of humans and AI agents as special type of active enterprise objects, and supports modelling chains or responsibility.

6.1 Integration architecture

One way of integrating our DSL with AI Agent architecrure is shown in Figure 5. Our DSL tooling includes a compiler which generates Python code from the DSL specification, which then runs in a Python runtime supporting the management of deontic tokens. This includes the monitoring of token passing and state change as a result of speech acts performative actions.

On the other hand, the LLM based agent architectures bring their own constraints which determine the best integration approach with our DSL tooling. For example, crewAI applications are concerned with structuring LLM applications in terms of AI agents which can collaborate, according to the constraints stated in their configuration definition.

The crewAI Agents are tightly linked to the crewAI runtime and the Python's interpreter, performing their goal-oriented reasoning using LLMs interactions,

and actions using configured tools. Computational agents there are not aware of deontic tokens and requirements imposed by our DSL and runtime.

This means that the best integration approach with our DSL is through the use of our runtime created from the DSL to monitor the life-cycle of deontic tokens (Figure 5). This is our generic approach to support deontic token functionality to other systems, which also requires exposing the deontic tokens management via an API. This is also an approach we adopted in linking with the crewAI system, where our API is accessed by crewAI Agents using a provided tool. This has required adding the tool functionality to crewAI system, to support querying and handling of tokens through our token management API.

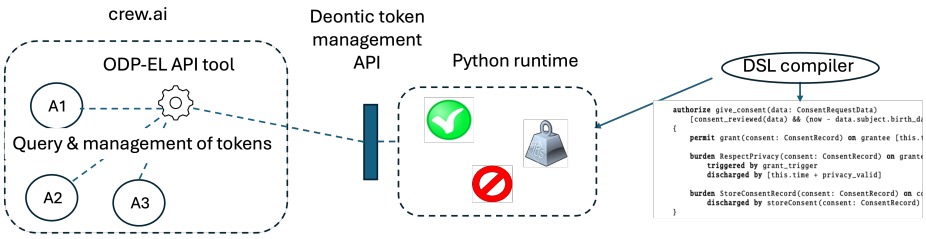

Fig. 5. DSL integration with Agent AI architecture: example

6.2 Discussions

Applying a generic DSL for ODP-EL to LLM-based agents presents several challenges. Firstly, their nondeterministic nature (that are dealt by agent architectures through the mechanism of reflection and continual plan updates) would suggest initial adoption of pessimistic enforcement strategy for violation of their accountability actions. It is also possible to support optimistic approaches but these would require sophisticated monitoring mechanisms, which are particularly difficult when monitoring obligations [13]. These mechanisms may rely on human oversight, in a similar way when monitoring behaviour associated with obligations in business contracts [15], or use of AI explainability tools.

The nature of deontic tokens, reflecting their different deontic modalities, requires different mechanisms for their handling in distributed systems. A permit typically benefits its holder, who needs to keep and present it when accessing resources that require possession of the permit. For example, a ticket to a football game is issued to the buyer, who must present the ticket at the stadium entrance. Thus, permit tokens can be distributed to their owners. It is to be noted that permits have broader scope than access tokens developed for granting access to specific resources or APIs in web applications, using OAuth2.0 authorisation protocol, the specific example of which is JSON Web Token (JWT) [11].

On the other hand, burden and embargo tokens are associated with parties that have no incentive to keep or present them. In fact, these parties have an

incentive to dispose of these tokens. A good example would be a traffic ticket. Therefore, these tokens must be stored in a central repository where interested parties with the appropriate credentials can verify whether the entities they are interacting with are constrained by any of these tokens.

These general considerations need to be taken into account when integrating with potential future distributed, agent AI architectures.

7 Conclusions and Future Work

This paper proposes a new method for formalising and tracking responsibility in complex enterprise systems involving both humans and AI agents. We achieve this by creating a domain specific language based on ODP-EL concepts. This standard-based language aims to support current and future AI architectures.

Our approach leverages the strengths of ODP-EL while utilizing modern DSL tools for faster development, deployment, and ongoing management of the system. Our emphasis on the precise expression and implementation support for accountability constrains over actions of the parties, and the associated chain of responsibility, is one of the first formal, yet pragmatic frameworks amenable to the contemporary software engineering practices, including user-oriented expressions of their requirements. We demonstrate its effectiveness with a use case for managing consent requirements in a digital health privacy consent use case.

Our implementation efforts have identified several challenges, arising from the stochastic properties of LLM systems as well as difficulties with monitoring obligations in distributed systems. We will monitor future developments in Agentic systems, and will accordingly update our DSL integration patterns with such architectures.

Our current language supports a set of key ODP-EL concepts, as driven by our digital health consent use case. Our plan is to implement several other concepts such as evaluation and prescription and do a full evaluation with users through several use cases. For example, we are considering applying our DSL to specific industrial applications, including digital twins [18], while leveraging our previous work on the monitoring of obligations in business contracts [15].

We also plan to develop integration of our DSL tooling with specific distributed architecture in health, such as FHIR [6,7]. This can be supported through a Mediator component that would encapsulate the logic for interacting with both the policy engine, which implements our policy language and the FHIR server, which includes a FHIR Consent Resource, similarly to what we used for integration with crewAI. We will also be investigating whether the deontic and accountability concepts can be mapped onto various FHIR Resources and workflow patterns, as a way of enhancing business process modelling with policy constraints, in initiatives such as eRequesting in Australia [1].

References

1. HL7 Australia - AU eRequesting Technical Design Group Home - HL7 Australia — FHIR Work Group - Confluence, https://confluence.hl7.org/display/HAFWG/HL7+Australia+-+AU+eRequesting+Technical+Design+Group+Home
2. ISO/IEC IS 19793, Information Technology — Open Distributed Processing — Use of UML for ODP System Specifications (2014), also published as ITU-T Recommendation X.906
3. Dejanović, I., Dejanović, M., Vidaković, J., Nikolić, S.: Pyflies: A domain-specific language for designing experiments in psychology. Applied Sciences **11**(17), 27 pages (2021). https://doi.org/10.3390/app11177823, https://www.mdpi.com/2076-3417/11/17/7823
4. Dorri, A., Kanhere, S.S., Jurdak, R.: Multi-agent systems: A survey. IEEE Access **6**, 28573–28593 (2018). https://doi.org/10.1109/ACCESS.2018.2831228
5. European Parliament, Council of the European Union: General Data Protection Regulation (GDPR) – Legal Text, https://gdpr-info.eu/
6. Fast Healthcare Interoperability Resources V5.0.0 (2023), http://hl7.org/fhir/R5/
7. Fast Healthcare Interoperability Resources: Consent (2023), https://build.fhir.org/consent.html
8. Griffo, C., Almeida, J.P.A., Guizzardi, G., Nardi, J.C.: From an ontology of service contracts to contract modeling in enterprise architecture. In: 2017 IEEE 21st international Enterprise distributed object computing conference (EDOC). pp. 40–49. IEEE (2017)
9. Hanson, J., Milosevic, Z.: Conversation-oriented protocols for contract negotiations. In: Seventh IEEE International Enterprise Distributed Object Computing Conference, 2003. Proceedings. pp. 40–49 (2003). https://doi.org/10.1109/EDOC.2003.1233836
10. ISO/IEC IS 15414, Information Technology - Open Distributed Processing - Enterprise Language 3rd edn (2015)
11. Jones, M., Bradley, J., Sakimura, N.: Json web token (jwt). Tech. rep., Internet Engineering Task Force (IETF) (2015), https://datatracker.ietf.org/doc/html/rfc7519
12. Jones, M.B., Nadalin, A., Campbell, B., Bradley, J., Mortimore, C.: OAuth 2.0 Token Exchange. Request for Comments RFC 8693, Internet Engineering Task Force (Jan 2020). https://doi.org/10.17487/RFC8693, https://datatracker.ietf.org/doc/rfc8693, num Pages: 27
13. Linington, P.F., Milosevic, Z., Tanaka, A., Vallecillo, A.: Building Enterprise Systems with ODP: An Introduction to Open Distributed Processing, 1st Edition. Chapman&Hall/CRC Innovations in Software Engineering and Software Development (2011)
14. Linington, P.F., Miyazaki, H., Vallecillo, A.: Obligations and Delegation in the ODP Enterprise Language. In: IEEE 16th International Enterprise Distributed Computing conference (2012)
15. Linington, P., Milosevic, Z., Cole, J., Gibson, S., Kulkarni, S., Neal, S.: A unified behavioural model and a contract language for extended enterprise. Data and Knowledge Engineering **51**(1), 5–29 (2004). https://doi.org/https://doi.org/10.1016/j.datak.2004.03.005, https://www.sciencedirect.com/science/article/pii/S0169023X0400031X, contact-driven coordination and collaboration in the Internet context
16. Milosevic, Z.: Ethics in digital health: A deontic accountability framework. In: 2019 IEEE 23rd International Enterprise Distributed Object Computing Conference (EDOC). pp. 105–111 (2019). https://doi.org/10.1109/EDOC.2019.00022

17. Milosevic, Z.: Enacting policies in digital health: a case for smart legal contracts and distributed ledgers? The Knowledge Engineering Review **35**, e6 (2020). https://doi.org/10.1017/S0269888920000089
18. Milosevic, Z., van Schalkwyk, P.: Towards responsible digital twins. In: Sales, T.P., de Kinderen, S., Proper, H.A., Pufahl, L., Karastoyanova, D., van Sinderen, M. (eds.) Enterprise Design, Operations, and Computing. EDOC 2023 Workshops. pp. 123–138. Springer Nature Switzerland, Cham (2024)
19. Mitrovic, D., Ivanovic, M., Vidakovic, M., Budimac, Z.: Siebog: An enterprise-scale multiagent middleware. Information technology and control **45**(2), 164–174 (2016)
20. Moura, J.: joaomdmoura/crewAI (Jun 2024), https://github.com/joaomdmoura/crewAI, original-date: 2023-10-27T03:26:59Z
21. Ng, A.: Four AI Agent Strategies That Improve GPT-4 and GPT-3.5 Performance (Mar 2024), https://www.deeplearning.ai/the-batch/how-agents-can-improve-llm-performance/
22. Park, J.S., O'Brien, J.C., Cai, C.J., Morris, M.R., Liang, P., Bernstein, M.S.: Generative agents: Interactive simulacra of human behavior (2023), https://arxiv.org/abs/2304.03442
23. Recommendation, W.: Odrl information model 2.2. https://www.w3.org/TR/odrl-model/ (2018)
24. Sredojević, D., Vidaković, M., Ivanović, M.: Alas: agent-oriented domain-specific language for the development of intelligent distributed non-axiomatic reasoning agents. Enterprise Information Systems **12**(8-9), 1058–1082 (2018)
25. Volter, M.: From programming to modeling-and back again. IEEE software **28**(6), 20–25 (2011)
26. Wu, Q., Bansal, G., Zhang, J., Wu, Y., Zhang, S., Zhu, E.E., Li, B., Jiang, L., Zhang, X., Wang, C.: AutoGen: Enabling Next-Gen LLM Applications via Multi-Agent Conversation. Tech. Rep. MSR-TR-2023-33, Microsoft (Aug 2023), https://www.microsoft.com/en-us/research/publication/autogen-enabling-next-gen-llm-applications-via-multi-agent-conversation-framework/
27. Language Engineering for Everyone! (2015), https://eclipse.dev/Xtext/index.html