Architectural Code in the Catlair Entity Model

Ruslan Cherkas Igor Cheledinov

2025-04-19

Contents

1 Introduction **1**

2 Definitions **1**

3 Theory **2**

4 Practice **2**
4.1 First Entity 3
4.2 Model Development 3
4.3 Contexts 4
4.4 Links 4
4.5 Entity Descriptions 6

5 Applicability **7**

6 Appendix **8**

Notes

1. https://doi.org/10.5281/zenodo.15250894 `https://doi.org/https://doi.org/10.5281/zenodo.15250894`
2. Permanent document link [ru]: `https://github.com/johnthesmith/catlair-archcode/blob/main/export/catlair-archcode-ru.pdf`
3. Permanent document link [en]: `https://github.com/johnthesmith/catlair-archcode/blob/main/export/catlair-archcode-en.pdf`
4. The article is published under the license: `https://creativecommons.org/licenses/by-sa/4.0/` CC BY-SA 4.0

1 Introduction

This article presents a concept for describing the architecture of information systems within a minimal set of rules.

The concept is based on the use of an entity hierarchy. The description is universal and minimalist. It includes: an entity index, context-dependent properties, and relationships.

2 Definitions

The concept operates with the following specific terms:

1. Entity — a named abstraction representing an object, concept, role, or any other phenomenon from the real world in an information system. Entities form hierarchies, can have properties, and can be interconnected.

   In the text, the term entity is denoted as `entity`, and the set of entities is denoted as `entities`.

2. Entity property — a description of the entity that defines its characteristics and features. An example of an entity property in architecture might include a name, description, quantitative indicators such as volume, etc.

   In the text, the term property is denoted as `prop`, and the set of properties is denoted as `props`.

3. Entity relationships — a means of defining dependencies or interactions between entities.

   In the text, relationships are denoted as `link`, and the set of relationships is denoted as `links`.

4. Contexts — views on the properties and relationships of entities that are specific to different participants and situations, depending on the environment. An example of a context could be the description language, differences in perception of the entity from the business logic side and the developer's side, and the level of detail in the diagrams.

   In the text, the term context is denoted as `context`, and the set of contexts is denoted as `contexts`.

3 Theory

The concept is based on the following principles:

1. Any concept in an information system should be described as an entity.
2. For each entity, it is necessary to minimally define:
   - the fact of existence;
   - the type of the entity.
3. All other descriptions and properties of the entity should be considered secondary and context-dependent, namely:
   - the relationship between entities;
   - the attributes or properties of the entity.

4. The concept adheres to the principle of separation of concerns, and therefore separates the description of entities and their relationships from the implementation of actions with them and their interpretation for specific purposes.

4 Practice

For practical examples, YAML syntax is used.

4.1 First Entity

1. Entities use a tuple relationship in key-value format, where the key is the entity identifier, and the value is the identifier of its type.

   This corresponds to a relational representation with two attributes:

   - `id` - the entity identifier;
   - `type` - the identifier of the entity type (which, in turn, is also an entity).

2. Since the entity type is mandatory, and no type exists yet, the first entity should self-type in the `entities` collection:

```
entities:
    entity: "entity"
```

   Note: The self-typing tuple creates a new domain. Their number is unlimited, and each can develop independently.

4.2 Model Development

1. Let's define several entities that will be used later in the description of the architecture.

```
entities:

    # Basic architecture entities

    # Component (services, state storages, etc...)
    component: "entity"
    # Service as a component
    service: "component"
    # Database as a component
    db: "component"
    # Agent capable of acting
```

```
    agent: "entity"
    # We defined the user as an agent
    user: "agent"
    # We defined the client as a user
    client: "user"

    # Let's add several architecture-specific entities:

    # Define the backend service
    my-backend: "service"
    # Define the database
    my-db: "db"
    # Define internal user Alice
    alice: "user"
    # Define client Bob
    bob: "client"
```

2. Thus, through successive declarations, diverse entities have been defined, which will be used later.

4.3 Contexts

1. The description of an entity differs for participants depending on the environment. Such differences are defined by the context.
2. A simple example is the description of an entity in a human-readable form in different languages.
3. A context may not only be a language, but also any specific perspective on entities from various groups, departments, divisions, or tasks.
4. Let's add several contexts.

```
entities:

    # Context as an entity
    context: "entity"

    # Language as a context
    lang: "context"
    # To-do plans as a context
    todo: "context"
    # As-is description as a context
    asis: "context"
```

```
    # Russian language
    ru: "lang"
    # English language
    en: "lang"
```

4.4 Links

1. After adding entities, their interrelationships should be considered. This is done using the `links` section.

2. Relationships are described as typed directions from one entity to another, where the type is also an entity.

3. For relationships, it is possible to specify multiple contexts, which allow displaying relationships in different situations. The context is optional. The absence of a context is interpreted at the presentation level.

4. In general, relationships can be defined by a tuple of attributes, each of which may contain one or more entities:

   - `from` - the entity that is the source of the relationship;
   - `to` - the entity that is the direction of the relationship;
   - `type` - the type of the relationship;
   - `context` - an optional list of contexts for which the relationship is relevant;
   - `tuple` - an optional list of specific attributes of the relationship in key-value format;

5. When lists are used in the attributes `from`, `to`, `link`, or `context`, relationships are interpreted as a Cartesian product of values. This means that for each element from one attribute's list, separate relationships are created with all elements from the other list, while the remaining values are preserved.

6. Let's describe some entity relationships:

```
entities:
    # Relationship as an entity
    link: "entity"

# Define relationships between entities in the links section
links:
    -
```

```
    # Define that the service connects to the DB
    from: "my-backend"
    to: "my-db"
    link: "connect"
-
    # Define that the client connects to the service
    from: "client"
    to: "service"
    link: "connect"
-
    # Alice can read and insert data
    from: "alice"
    link:
        - "select"
        - "insert"
    to: "my-db"
    context: "right"
-
    # Bob has select permission for the service in the context
    # of rights for the "asis" concept
    from: "bob"
    to: "service"
    link: "select"
    context:
        - "right"
        - "asis"
-
    # It is planned that Bob will have creation and insertion
    # rights according to the "todo" concept
    from: "bob"
    to: "service"
    link:
        - "select"
        - "insert"
    context:
        - "right"
        - "todo"
```

7. The described method can include many different dependencies, including technical component connections, hierarchical subordination structures, component location placements, and more.

4.5 Entity Descriptions

1. Further, entities can be described with specific properties, also in the context of contexts.

2. In general, the properties of entities can be defined as a tuple of attributes:

   - `entity` - the entity for which the description is made, it can contain many entities;
   - `tuple` - a list of properties of the entity in key-value format;
   - `context` - an optional list of contexts for which the description is made;

3. Descriptions of entities are made in the "props" section:

```
props:
    # Human-readable description for an entity
    -
        entity: "entity"
        context: "ru"
        tuple:
            name:"Entity"
    # Description of Alice outside of context
    -
        entity: "alice"
        tuple:
            age:21
            weight:71

    # Descriptions of properties for Alice and
    # Bob in different languages
    -
        entity: "alice"
        context: "ru"
        tuple:
            name:"Алиса"
    -
        entity: "alice"
        context: "en"
        tuple:
            name:"Alice"
    -
        entity: "bob"
        context: "ru"
        tuple:
            name:"Боб"
```

```
-
    entity: "bob"
    context: "en"
    tuple:
        name:"Bob"
```

4. Similarly, any properties of entities can be described.

5 Applicability

1. The examples provided demonstrate the applicability of the Catlair entity model for describing architectural models.
2. The Catlair notation allows for a standardized description of the list of objects, components, and their interrelationships. Additionally, all of these can be represented from different perspectives, including chronological ones.
3. The model is applicable for describing organizational structures, access rights, and technical components.
4. The model can be used for the uniform generation of ER, BPMN, and C4 diagrams at all levels, while maintaining consistency in notation, with the format of presentation determined by the context.

6 Appendix

The appendix contains a compact listing of the examples described above for general understanding.

```
entities:
    entity: "entity"
    component: "entity"
    service: "component"
    db: "component"
    agent: "entity"
    user: "agent"
    client: "user"
    my-backend: "service"
    my-db: "db"
    alice: "user"
    bob: "client"
    link: "entity"
    context: "entity"
```

```
    lang: "context"
    todo: "context"
    asis: "context"
    ru: "lang"
    en: "lang"
links:
    -
        from: "my-backend"
        to: "my-db"
        link: "connect"
    -
        from: "client"
        to: "service"
        link: "connect"
    -
        from: "alice"
        link:
            - "select"
            - "insert"
        to: "my-db"
        context: "right"
    -
        from: "bob"
        to: "service"
        link: "select"
        context:
            - "right"
            - "asis"
        from: "bob"
        to: "service"
        link:
            - "select"
            - "insert"
        context:
            - "right"
            - "todo"
    props:
        -
            entity: "entity"
            context: "ru"
            tuple:
                name:"Сущность"
        -
            entity: "alice"
            tuple:
                age:21
```

```
        weight:71
-
    entity: "alice"
    context: "ru"
    tuple:
        name:"Алиса"
-
    entity: "alice"
    context: "en"
    tuple:
        name:"Alice"
-
    entity: "bob"
    context: "ru"
    tuple:
        name:"Боб"
-
    entity: "bob"
    context: "en"
    tuple:
        name:"Bob"
```

References

[1] EWD447 *On the role of scientific thought* https://www.cs.utexas.edu/~EWD/transcriptions/EWD04xx/EWD447.html

[2] Simon Brown.
Software Architecture as Code.
https://static.codingthearchitecture.com/presentations/saturn2015-software-architecture-as-code.pdf

[3] Cherkas R. Cheledinov I.
Архитектура сущностей Catlair.
https://github.com/johnthesmith/scraps/blob/main/ru/entities.md

[4] Mark Richards, Neal Ford.
Is "Architecture as Code" the Future of Software Design?
https://www.architectureandgovernance.com/applications-technology/is-architecture-as-code-the-future-of-software-design/