Bring Your Own Data (BYOD)

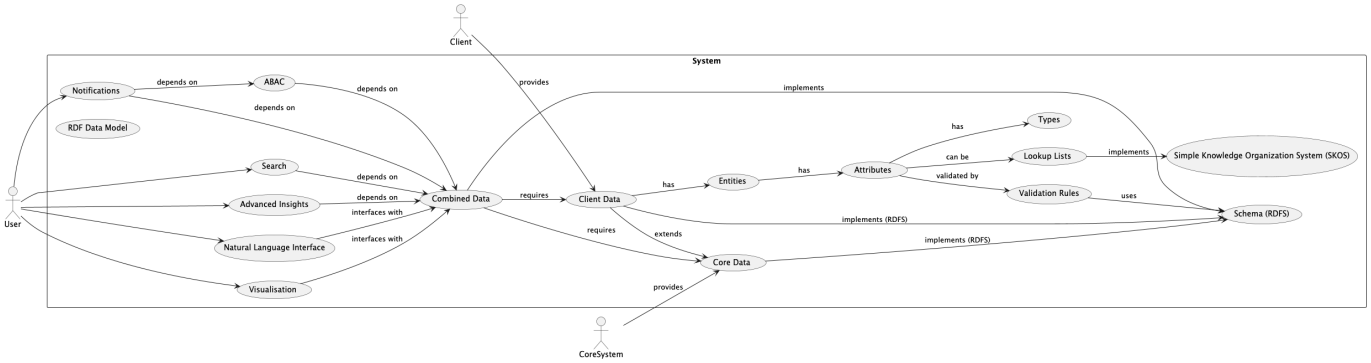
Architecture Design Specification (ADS)

Peter Verster

Table of contents

1. BYOD Function Implementation (MM)

1.1 Scenarios

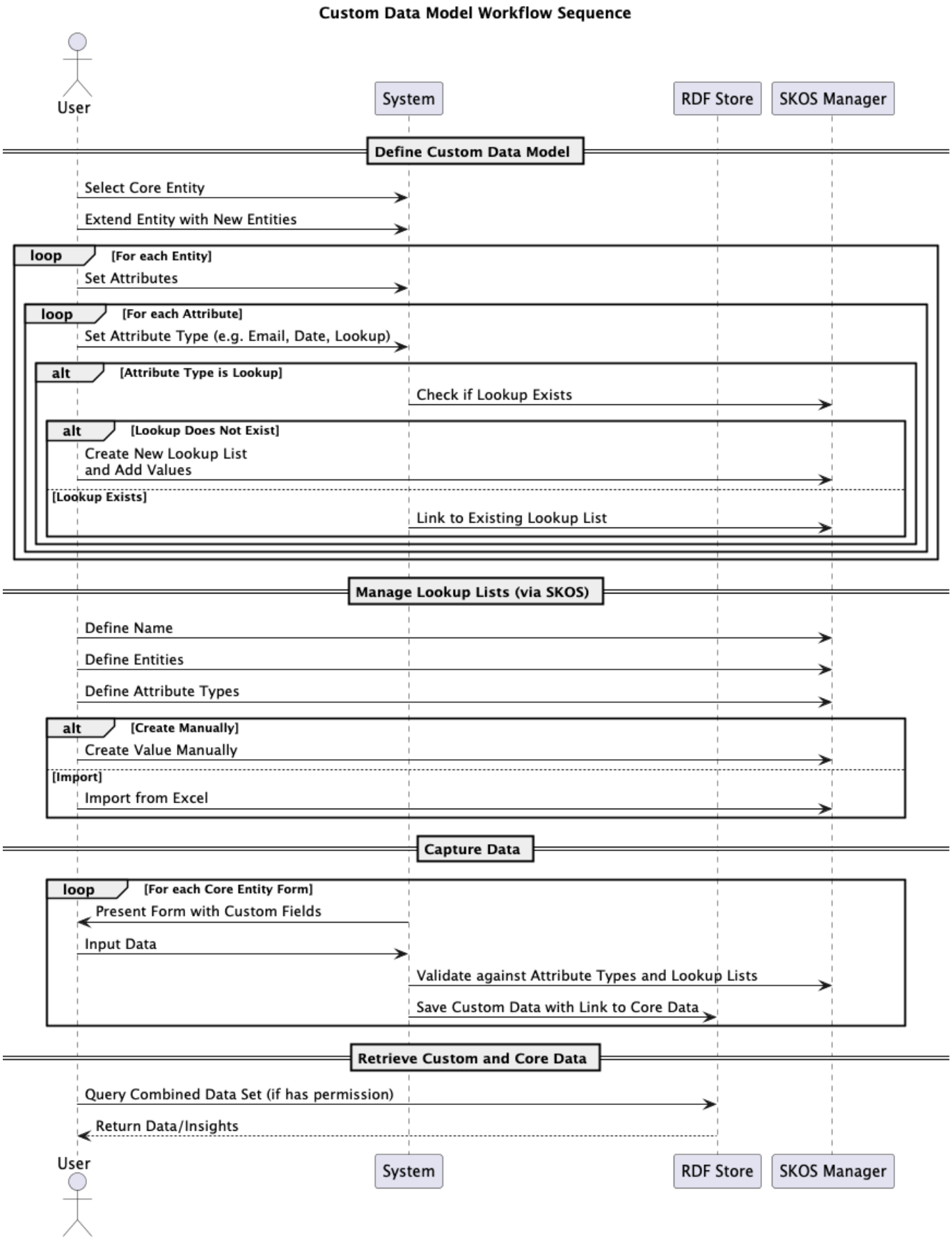



```
@startuml
left to right direction
!define RECTANGLE class

actor Client
actor CoreSystem
actor User

rectangle "System" {
    usecase "Combined Data" as UC_CombinedData
    usecase "RDF Data Model" as UC_RDF
    usecase "Client Data" as UC_ClientData
    usecase "Core Data" as UC_CoreData
    usecase "Schema (RDFS)" as UC_Schema
    usecase "Entities" as UC_Entities
    usecase "Attributes" as UC_Attributes
    usecase "Types" as UC_Types
    usecase "Validation Rules" as UC_Validation
    usecase "Lookup Lists" as UC_LookupLists
    usecase "Simple Knowledge Organization System (SKOS)" as UC_SKOS

    usecase "Search" as UC_Search
    usecase "Advanced Insights" as UC_Insights
    usecase "Natural Language Interface" as UC_NLI
    usecase "Visualisation" as UC_Visualisation
    usecase "ABAC" as UC_ABAC
    usecase "Notifications" as UC_Notifications
}

' Actors interacting with systems
Client --> UC_ClientData : provides
CoreSystem --> UC_CoreData : provides
UC_ClientData --> UC_CoreData: extends

UC_Validation --> UC_Schema: uses

' Dependencies for Combined Data
UC_CombinedData --> UC_ClientData : requires
UC_CombinedData --> UC_CoreData : requires
UC_CombinedData --> UC_Schema : implements

' System capabilities depending on Combined Data
UC_Search --> UC_CombinedData : depends on
UC_Insights --> UC_CombinedData : depends on
UC_NLI --> UC_CombinedData : interfaces with
UC_Visualisation --> UC_CombinedData : interfaces with
UC_ABAC --> UC_CombinedData : depends on
UC_Notifications --> UC_CombinedData : depends on
UC_Notifications --> UC_ABAC : depends on



UC_ClientData --> UC_Entities : has
UC_Entities --> UC_Attributes : has
UC_Attributes --> UC_Types : has
UC_Attributes --> UC_LookupLists: can be
UC_LookupLists --> UC_SKOS: implements
UC_Attributes --> UC_Validation : validated by
UC_ClientData --> UC_Schema : implements (RDFS)
UC_CoreData --> UC_Schema : implements (RDFS)

' User interactions
User --> UC_Search
```

```
User --> UC_Insights
User --> UC_NLI
User --> UC_Visualisation
User --> UC_Notifications
@enduml
```

1.1.1 Query Core and Custom Data Together (Custom Insights)

1.2 Process View



1.3 Outline Use Cases (Stories)

1.3.1 Create lookup lists -> SKOS

- A **top-level concept** (like "Material Type")
- **Narrower concepts** under it (like "Broadcast" and "Online")
- And even **narrower subtypes** under those (like "TV", "Cinema", "YouTube", "VOD")

Structure:

```
Material Type
├── Broadcast
│   ├── TV
│   └── Cinema
└── Online
    ├── YouTube
    └── VOD
```

Example in Turtle (RDF)

```
@prefix skos: <http://www.w3.org/2004/02/skos/core#> .
@prefix clearcast: <http://clearcast.co.uk/schema#> .

# Concept scheme
clearcast:MaterialTypeScheme a skos:ConceptScheme ;
    skos:prefLabel "Material Types"@en .

# Top-level concept
clearcast:MaterialType a skos:Concept ;
    skos:prefLabel "Material Type"@en ;
    skos:inScheme clearcast:MaterialTypeScheme .

# First-level children
clearcast:Broadcast a skos:Concept ;
    skos:prefLabel "Broadcast"@en ;
    skos:broader clearcast:MaterialType ;
    skos:inScheme clearcast:MaterialTypeScheme .

clearcast:Online a skos:Concept ;
    skos:prefLabel "Online"@en ;
    skos:broader clearcast:MaterialType ;
    skos:inScheme clearcast:MaterialTypeScheme .

# Second-level children
clearcast:TV a skos:Concept ;
    skos:prefLabel "TV"@en ;
    skos:broader clearcast:Broadcast ;
    skos:inScheme clearcast:MaterialTypeScheme .

clearcast:Cinema a skos:Concept ;
    skos:prefLabel "Cinema"@en ;
    skos:broader clearcast:Broadcast ;
    skos:inScheme clearcast:MaterialTypeScheme .

clearcast:YouTube a skos:Concept ;
    skos:prefLabel "YouTube"@en ;
    skos:broader clearcast:Online ;
    skos:inScheme clearcast:MaterialTypeScheme .

clearcast:VOD a skos:Concept ;
    skos:prefLabel "VOD"@en ;
    skos:broader clearcast:Online ;
    skos:inScheme clearcast:MaterialTypeScheme .
```

Bonus Tips:

- Use `skos:broader` and `skos:narrower` to create hierarchical relationships.
- In UI dropdowns, you can render this as nested menus or tree selectors.
- Use `skos:prefLabel` for display, and use the URIs as stored values.

1.3.2 Concept: Link Property to SKOS Concept

You have a `Card`, and it has a custom property `clearcast:materialType`.
Instead of just using a plain string (e.g. `"TV"`), you link it to a **SKOS Concept URI**, e.g. `clearcast:TV`.

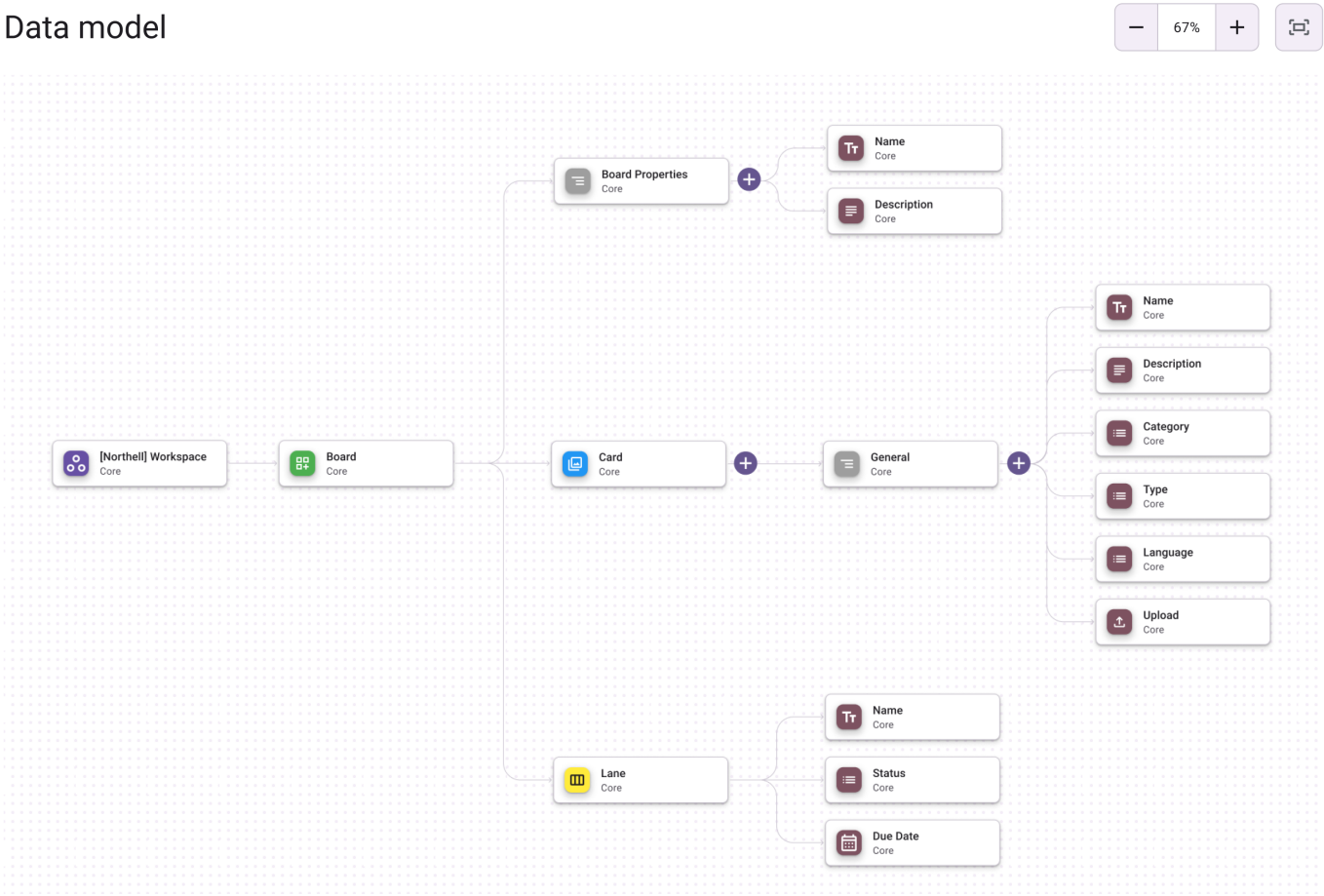
Example in Turtle (RDF format)

```
@prefix flow: <http://northell.com/media-magic/flow#> .
@prefix clearcast: <http://clearcast.co.uk/schema#> .
@prefix skos: <http://www.w3.org/2004/02/skos/core#> .
@prefix ex: <http://example.org/instance#> .

# Instance of a Card
ex:Card123 a flow:Card ;
    clearcast:materialType clearcast:TV .

# TV is a concept in the SKOS scheme
clearcast:TV a skos:Concept ;
    skos:inScheme clearcast:MaterialTypeScheme ;
    skos:prefLabel "TV"@en ;
    skos:definition "Television broadcast format"@en .
```

Why This Matters

- Your UI can show a dropdown by querying `skos:Concept` in `clearcast:MaterialTypeScheme`.
- Users pick from this list.
- The selected **URI** (like `clearcast:TV`) is saved as the value.
- This allows for **controlled vocabularies**, multilingual labels, definitions, and compatibility with SPARQL-based systems.

1.3.3 Extend Core Schema (entities and attributes) -> RDFS ()



1.3.4 🔄 Strategy Pattern in Media Magic Flow

The Strategy Pattern allows the behavior of a class to be selected at runtime. In the context of your **kanban-style system**, this means:

- **Core entities** like `Card` can have **different behaviors or roles**, e.g., `VideoCard`, `ScriptCard`, etc.
- These **specialisms** are *interchangeable*, allowing dynamic configuration without altering the core logic.
- Each **specialism** adds domain-specific functionality.

✅ Functional Enhancements via Strategy Pattern

- `VideoCard`: adds video format metadata, preview rendering
- `ScriptCard`: supports rich text editing, versioning, and language tagging
- `StoryBoardCard`: image sequences, transitions
- `BriefCard`: links to global guidelines and references

Each specialism extends `Card` but with unique, pluggable behaviors.

1.3.5 🧠 RDFS + SKOS Schema (with Namespace)

Let's define: - **RDFS** for your data model - **SKOS** for taxonomy (e.g., types of Cards) - Namespace:

- `northell:` → `http://northell.com/`
- `mm:` → `http://northell.com/media-magic#`

- `flow:` → `http://northell.com/media-magic/flow#`
- `skos:` → `http://www.w3.org/2004/02/skos/core#`

1.3.6 🧩 RDF Schema Definition (Turtle Syntax)

```
@prefix northell: <http://northell.com/> .
@prefix mm: <http://northell.com/media-magic#> .
@prefix flow: <http://northell.com/media-magic/flow#> .
@prefix skos: <http://www.w3.org/2004/02/skos/core#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

# Core Classes
flow:Workspace a rdfs:Class .
flow:Board a rdfs:Class .
flow:Card a rdfs:Class .
flow:Lane a rdfs:Class .

# Core Properties
flow:name a rdf:Property ; rdfs:domain rdfs:Resource ; rdfs:range xsd:string .
flow:description a rdf:Property ; rdfs:domain rdfs:Resource ; rdfs:range xsd:string .
flow:category a rdf:Property ; rdfs:domain flow:Card ; rdfs:range xsd:string .
flow:type a rdf:Property ; rdfs:domain flow:Card ; rdfs:range xsd:string .
flow:language a rdf:Property ; rdfs:domain flow:Card ; rdfs:range xsd:string .
flow:upload a rdf:Property ; rdfs:domain flow:Card ; rdfs:range xsd:anyURI .
flow:status a rdf:Property ; rdfs:domain flow:Lane ; rdfs:range xsd:string .
flow:dueDate a rdf:Property ; rdfs:domain flow:Lane ; rdfs:range xsd:date .

# Specializations (Strategy Pattern)
flow:VideoCard a rdfs:Class ; rdfs:subClassOf flow:Card .
flow:ScriptCard a rdfs:Class ; rdfs:subClassOf flow:Card .
flow:BriefCard a rdfs:Class ; rdfs:subClassOf flow:Card .
flow:StoryboardCard a rdfs:Class ; rdfs:subClassOf flow:Card .

# SKOS Specialism Taxonomy
mm:CardTypeScheme a skos:ConceptScheme ; skos:prefLabel "Card Specialisms"@en .

mm:VideoCard a skos:Concept ;
    skos:inScheme mm:CardTypeScheme ;
    skos:prefLabel "Video Card"@en ;
    skos:definition "Card used for video asset handling and metadata." .

mm:ScriptCard a skos:Concept ;
    skos:inScheme mm:CardTypeScheme ;
    skos:prefLabel "Script Card"@en ;
    skos:definition "Card used for managing scripts with language and versioning support." .

mm:BriefCard a skos:Concept ;
    skos:inScheme mm:CardTypeScheme ;
    skos:prefLabel "Brief Card"@en ;
    skos:definition "Used for creative briefs and global guideline linkage." .

mm:StoryboardCard a skos:Concept ;
    skos:inScheme mm:CardTypeScheme ;
    skos:prefLabel "Storyboard Card"@en ;
    skos:definition "Visual planning card for storytelling and shot sequencing." .
```

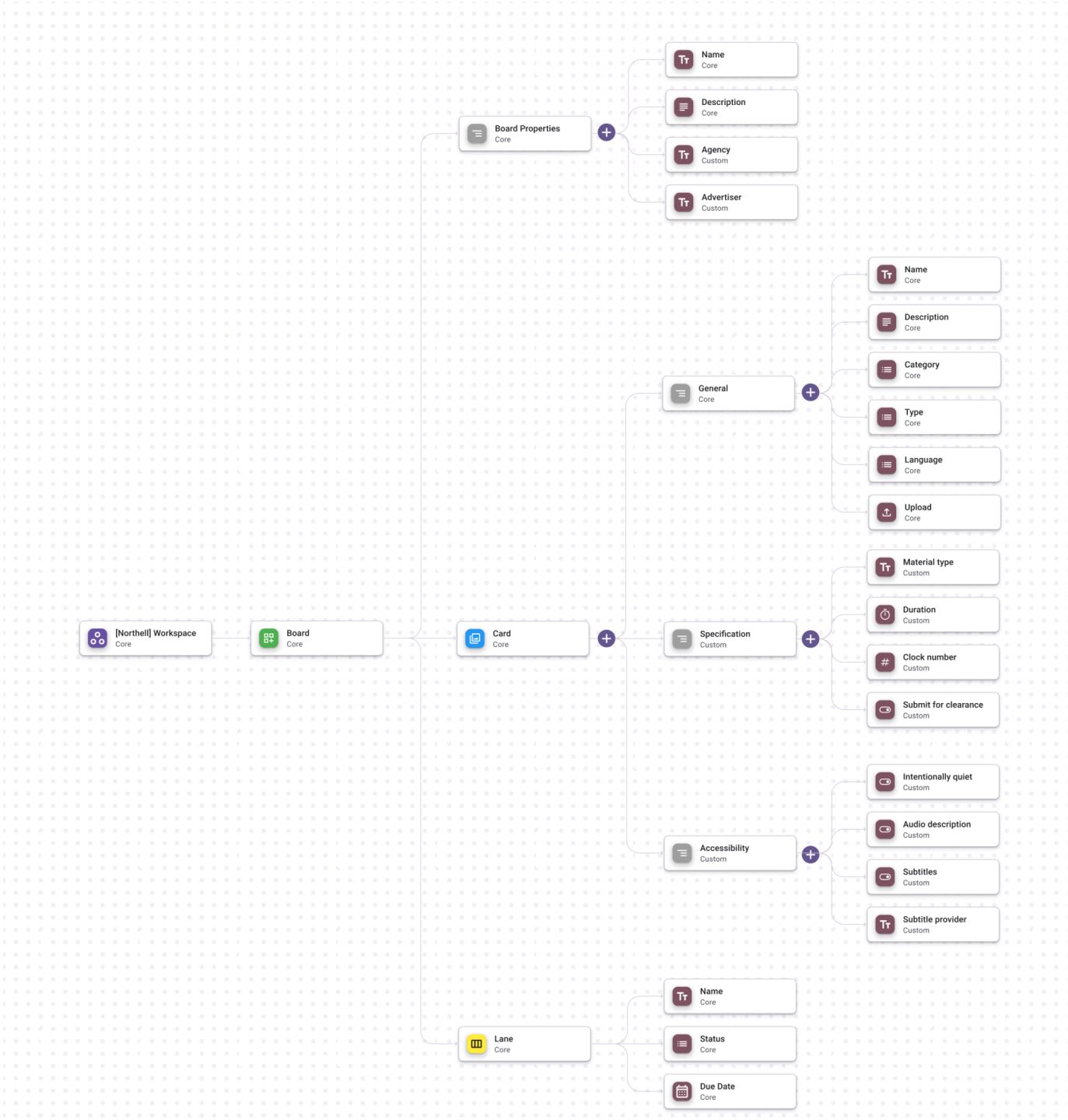
1.3.7 🧰 Implementation Notes

- **UI Picklists**: can query the SKOS taxonomy via SPARQL to build dropdowns for Card types.
- **Strategy Mapping**: in code, associate each SKOS concept with a class implementing the behavior (e.g., `VideoCardBehavior implements CardBehavior`).
- **Pluggability**: new types can be added just by defining new subclasses and SKOS concepts.

```
from rdflib import Graph, Namespace, RDF, RDFS, XSD, SKOS, Literal, URIRef

# Define Namespaces
northell = Namespace("http://northell.com/")
mm = Namespace("http://northell.com/media-magic#")
flow = Namespace("http://northell.com/media-magic/flow#")

# Create Graph
g = Graph()

# Bind namespaces for readability
g.bind("northell", northell)
g.bind("mm", mm)
g.bind("flow", flow)
```

```
g.bind("skos", SKOS)

# Define core classes
core_classes = ['Workspace', 'Board', 'Card', 'Lane']
for cls in core_classes:
    g.add((flow[cls], RDF.type, RDFS.Class))

# Define properties
properties = {
    'name': RDFS.Resource,
    'description': RDFS.Resource,
    'category': flow.Card,
    'type': flow.Card,
    'language': flow.Card,
    'upload': flow.Card,
    'status': flow.Lane,
    'dueDate': flow.Lane
}
ranges = {
    'name': XSD.string,
    'description': XSD.string,
    'category': XSD.string,
    'type': XSD.string,
    'language': XSD.string,
    'upload': XSD.anyURI,
    'status': XSD.string,
    'dueDate': XSD.date
}
for prop, domain in properties.items():
    g.add((flow[prop], RDF.type, RDF.Property))
    g.add((flow[prop], RDFS.domain, domain))
    g.add((flow[prop], RDFS.range, ranges[prop]))

# Define specializations
specializations = ['VideoCard', 'ScriptCard', 'BriefCard', 'StoryboardCard']
for spec in specializations:
    g.add((flow[spec], RDF.type, RDFS.Class))
    g.add((flow[spec], RDFS.subClassOf, flow.Card))

# Define SKOS ConceptScheme
g.add((mm.CardTypeScheme, RDF.type, SKOS.ConceptScheme))
g.add((mm.CardTypeScheme, SKOS.prefLabel, Literal("Card Specialisms", lang="en")))

# Define SKOS concepts
concepts = {
    'VideoCard': "Card used for video asset handling and metadata.",
    'ScriptCard': "Card used for managing scripts with language and versioning support.",
    'BriefCard': "Used for creative briefs and global guideline linkage.",
    'StoryboardCard': "Visual planning card for storytelling and shot sequencing."
}
for concept, definition in concepts.items():
    g.add((mm[concept], RDF.type, SKOS.Concept))
    g.add((mm[concept], SKOS.inScheme, mm.CardTypeScheme))
    g.add((mm[concept], SKOS.prefLabel, Literal(concept.replace("Card", " Card"), lang="en")))
    g.add((mm[concept], SKOS.definition, Literal(definition, lang="en")))

# Serialize the graph to a Turtle file
file_path = "/mnt/data/media_magic_flow_schema.ttl"
g.serialize(destination=file_path, format="turtle")

file_path
```

1.3.8 Custom Clearcast Attributes




```
from rdflib import Namespace, Graph, RDF, RDFS, SKOS, XSD, Literal

# Define Namespaces
northell = Namespace("http://northell.com/")
mm = Namespace("http://northell.com/media-magic#")
flow = Namespace("http://northell.com/media-magic/flow#")
clearcast = Namespace("http://clearcast.co.uk/schema#")

# Create RDF graph
g = Graph()

# Bind namespaces
g.bind("northell", northell)
g.bind("mm", mm)
g.bind("flow", flow)
```

```
g.bind("clearcast", clearcast)
g.bind("skos", SKOS)

# Extend Card and Board with custom Clearcast metadata

# Clearcast additions to Board
board_custom_attrs = {
    "agency": XSD.string,
    "advertiser": XSD.string
}
for attr, datatype in board_custom_attrs.items():
    g.add((clearcast[attr], RDF.type, RDF.Property))
    g.add((clearcast[attr], RDFS.domain, flow.Board))
    g.add((clearcast[attr], RDFS.range, datatype))

# Clearcast additions to Card
card_custom_attrs = {
    "materialType": XSD.string,
    "duration": XSD.duration,
    "clockNumber": XSD.string,
    "submitForClearance": XSD.boolean
}
for attr, datatype in card_custom_attrs.items():
    g.add((clearcast[attr], RDF.type, RDF.Property))
    g.add((clearcast[attr], RDFS.domain, flow.Card))
    g.add((clearcast[attr], RDFS.range, datatype))

# Accessibility extensions
accessibility_attrs = {
    "intentionallyQuiet": XSD.boolean,
    "audioDescription": XSD.boolean,
    "subtitles": XSD.boolean,
    "subtitleProvider": XSD.string
}
for attr, datatype in accessibility_attrs.items():
    g.add((clearcast[attr], RDF.type, RDF.Property))
    g.add((clearcast[attr], RDFS.domain, flow.Card))
    g.add((clearcast[attr], RDFS.range, datatype))

# SKOS Concept Scheme for Material Types
g.add((clearcast.MaterialTypeScheme, RDF.type, SKOS.ConceptScheme))
g.add((clearcast.MaterialTypeScheme, SKOS.prefLabel, Literal("Material Types", lang="en")))

material_types = [
    ("TV", "Television broadcast format"),
    ("VOD", "Video on demand format"),
    ("Cinema", "Cinema release format"),
    ("Online", "Online or digital format")
]

for code, definition in material_types:
    uri = clearcast[code]
    g.add((uri, RDF.type, SKOS.Concept))
    g.add((uri, SKOS.inScheme, clearcast.MaterialTypeScheme))
    g.add((uri, SKOS.prefLabel, Literal(code, lang="en")))
    g.add((uri, SKOS.definition, Literal(definition, lang="en")))

# Serialize to file
file_path = "/mnt/data/clearcast_extension.ttl"
g.serialize(destination=file_path, format="turtle")

file_path
```

``` text

@prefix clearcast: http://clearcast.co.uk/schema# . @prefix flow: http://northell.com/media-magic/flow# . @prefix rdf: http://www.w3.org/1999/02/22-rdf-syntax-ns# . @prefix rdfs: http://www.w3.org/2000/01/rdf-schema# . @prefix skos: http://www.w3.org/2004/02/skos/core# . @prefix xsd: http://www.w3.org/2001/XMLSchema# .

clearcast:Cinema a skos:Concept ; skos:definition "Cinema release format"@en ; skos:inScheme clearcast:MaterialTypeScheme ; skos:prefLabel "Cinema"@en .

clearcast:Online a skos:Concept ; skos:definition "Online or digital format"@en ; skos:inScheme clearcast:MaterialTypeScheme ; skos:prefLabel "Online"@en .

clearcast:TV a skos:Concept ; skos:definition "Television broadcast format"@en ; skos:inScheme clearcast:MaterialTypeScheme ; skos:prefLabel "TV"@en .

clearcast:VOD a skos:Concept ; skos:definition "Video on demand format"@en ; skos:inScheme clearcast:MaterialTypeScheme ; skos:prefLabel "VOD"@en .

clearcast:advertiser a rdf:Property ; rdfs:domain flow:Board ; rdfs:range xsd:string .

clearcast:agency a rdf:Property ; rdfs:domain flow:Board ; rdfs:range xsd:string .

clearcast:audioDescription a rdf:Property ; rdfs:domain flow:Card ; rdfs:range xsd:boolean .

clearcast:clockNumber a rdf:Property ; rdfs:domain flow:Card ; rdfs:range xsd:string .
```

```
clearcast:duration a rdf:Property ; rdfs:domain flow:Card ; rdfs:range xsd:duration .

clearcast:intentionallyQuiet a rdf:Property ; rdfs:domain flow:Card ; rdfs:range xsd:boolean .

clearcast:materialType a rdf:Property ; rdfs:domain flow:Card ; rdfs:range xsd:string .

clearcast:submitForClearance a rdf:Property ; rdfs:domain flow:Card ; rdfs:range xsd:boolean .

clearcast:subtitleProvider a rdf:Property ; rdfs:domain flow:Card ; rdfs:range xsd:string .

clearcast:subtitles a rdf:Property ; rdfs:domain flow:Card ; rdfs:range xsd:boolean .

clearcast:MaterialTypeScheme a skos:ConceptScheme ; skos:prefLabel "Material Types"@en .
```

2. 4+1 Architectural View Model

The **4+1 Architectural View Model** was introduced by Philippe Kruchten as a way to describe the architecture of software-intensive systems, using **five concurrent views**. Each view addresses specific concerns of different stakeholders (developers, users, testers, project managers, etc.), and together they provide a comprehensive blueprint of the system's architecture.

Here's a **comprehensive description of each of the 4+1 views**, along with the **UML diagrams** typically used for each:

2.1 +1. Scenarios / Use Case View (Stakeholders, Clients, Testers)

2.1.1 Focus:

- Illustrates typical **use cases or user stories**.
- Validates and illustrates the architecture using real-world examples.

2.1.2 Purpose:

- Acts as the **glue** between the four views.
- Helps ensure the architecture meets **functional requirements**.

2.1.3 Common UML Diagrams:

- **Use Case Diagram**: Captures user goals and system interactions.
- **Sequence Diagram**: Helps illustrate the realisation of use cases.
- **Activity Diagram**: Can be used to express complex workflows behind a use case.

2.2 1. Logical View (Designers & Developers)

2.2.1 Focus:

- Captures the system's **functional requirements**.
- Shows how the system is decomposed into classes, objects, and interfaces.

2.2.2 Purpose:

- To help developers understand the **design and internal structure**.
- To model the **object-oriented design** of the system.

2.2.3 Common UML Diagrams:

- **Class Diagram**: Shows classes, interfaces, attributes, methods, and relationships.
- **Object Diagram**: Depicts instances of classes at a specific time.
- **Sequence Diagram**: Shows interactions between objects over time (also part of Process View, but useful here for logic).
- **Communication Diagram** (formerly Collaboration Diagram): Focuses on object interactions and relationships.

2.3 2. Process View (System Integrators)

2.3.1 Focus:

- Addresses the **dynamic aspects** of the system.
- Shows how the system behaves at runtime and how components interact during execution.

2.3.2 Purpose:

- Models **concurrency, performance, and scalability**.
- Useful for identifying **threads, processes, and inter-process communication**.

2.3.3 Common UML Diagrams:

- **Activity Diagram**: Describes workflows and control logic.
- **Sequence Diagram**: Emphasises the time ordering of messages between processes or threads.
- **State Machine Diagram**: Captures the lifecycle of stateful components or systems.
- **Deployment Diagram**: When focused on execution nodes and runtime artefacts.

2.4 3. Development View (Programmers & Software Managers)

2.4.1 Also Known As:

- **Implementation View**

2.4.2 Focus:

- Describes the system from a **programmer's perspective**.
- Shows how the system is organised into modules, packages, components, and libraries.

2.4.3 Purpose:

- Helps manage source code and software configuration.
- Addresses software **build, reuse, and modularisation**.

2.4.4 Common UML Diagrams:

- **Component Diagram**: Illustrates how the system is divided into components/modules and their dependencies.
- **Package Diagram**: Groups related classes and interfaces into packages.

2.5 4. Physical View (System Engineers & Deployers)

2.5.1 Also Known As:

- **Deployment View**

2.5.2 Focus:

- Describes the **physical deployment** of software artefacts on hardware.
- Focus on **infrastructure**, such as servers, network nodes, and communication links.

2.5.3 Purpose:

- Models **system topology, installation, and distribution**.
- Addresses **availability, fault tolerance, and scalability**.

2.5.4 Common UML Diagrams:

- **Deployment Diagram**: Maps software artefacts (components, services) to hardware nodes.
- **Component Diagram** (when used to describe installed components).

2.6 Summary Table

View	Stakeholders	Focus	Key UML Diagrams
Logical View	Designers, Developers	Functionality & object structure	Class, Object, Sequence, Communication
Process View	Integrators, Performance Engineers	Runtime behaviour & concurrency	Activity, Sequence, State Machine, Deployment
Development View	Programmers, Managers	Software structure & components	Component, Package
Physical View	System Engineers	Deployment & hardware topology	Deployment, Component
Use Case (+1)	Stakeholders, Testers	Functional validation	Use Case, Sequence, Activity