IT Project Guidance

Design: Modelling  
Users, Persons, Channels & Locations

Version:

0.2

## Purpose

To support the development of more maintainable, flexible and resilient data schemas by disambiguating the differences between Addresses, Channels and Locations.

## Synopsis

Addresses are forms of communication Channels. Locations are distinct from Addresses, and both are distinct from the Person and Persona they are providing information about.

## Contents

[Purpose 1](#_Toc194674266)

[Synopsis 1](#_Toc194674267)

[Contents 2](#_Toc194674268)

[Introduction 3](#_Toc194674269)

[Background 3](#_Toc194674270)

[Users and Persons 3](#_Toc194674271)

[Channels 4](#_Toc194674272)

[Location 5](#_Toc194674273)

[Personas and Identifiers 4](#_Toc194674274)

[Social Domain, Entities & Value Objects 6](#_Toc194674275)

[Further Discussion 7](#_Toc194674276)

[Conclusion 9](#_Toc194674277)

[Appendices 11](#_Toc194674278)

[Appendix A - Document Information 11](#_Toc194674279)

[Versions 11](#_Toc194674280)

[Images 11](#_Toc194674281)

[Tables 11](#_Toc194674282)

[References 11](#_Toc194674283)

[Review Distribution 11](#_Toc194674284)

[Audience 11](#_Toc194674285)

[Structure 12](#_Toc194674286)

[Diagrams 12](#_Toc194674287)

[Acronyms 12](#_Toc194674288)

[Terms 12](#_Toc194674289)

# Introduction

All business systems have users. And whether users, customers, employees, suppliers, or learners, almost all systems involve people who need to be contacted. This often leads developers to model a 'Person' entity, then attach one or more Address value objects directly to the Person, typically for default billing and shipping purposes. This model is so common that it seems unquestionable — and yet it is incorrect and has never reflected the true, evolvable mechanics involved in modelling users, people, communication Channels, and Location.

## Background

The "Contact + 2 Addresses" model originates from early development texts that introduced these patterns for simplicity and educational purposes, and as such were not concerned with real-world use cases or evolvability. Even if they thought it could be used, it was from a different era. At the time, PO Boxes were less prevalent, phones were landlines, faxes were standard, and there were no Amazon, AliExpress, or Temu delivery services — and certainly no WhatsApp or social media. However, the assumptions baked into those simplistic educational models have persisted long afterwards, showing up in production systems. For those reasons alone, a relook at the model would be appropriate.

## Modelling Approach

A mature modelling discipline, Domain-Driven Design (DDD), encourages us to begin by identifying and separating distinct domains of concern — for example, distinguishing between the **System Domain**, where we represent system-level agents like authenticated services or background processes, and the **Social Domain**, where we model people and their social relationships. Only once the domains are separated can we define the entities and value objects that make sense within each, with clear boundaries and responsibilities.

This is what we aim to demonstrate in the rest of this document. It is not simply a data modelling correction, but an example of how clearer thinking at the domain level leads to more adaptable, consistent, and future-aligned systems.

## Users and Persons

All systems have system Users. For a number of reasons - including performance - User records are kept as "thin" as possible. Whether the User record contains a system display name, or that is saved in a separate User Profile (i.e. User Preferences) object, is a design choice.

All users have to be contacted at some point — if only to notify them of system downtimes, other state changes, or policy updates. But it would be a design mistake to associate contact details directly to the User.

Instead, each User is associated with a separate Person. The logic is that a system may need to send notifications to Persons who are not yet Users. For example, a current User may invite a non-user Person to accept a Role within the system and in doing so become a User. Since the User doesn’t yet exist — and it would be incorrect to create a User record for a non-user, even in a disabled state — best practice is to maintain a distinct Person record.

As for how to contact Persons, the default mechanism of the web is email. But it’s not the only contact channel.

## Personas and Identifiers

Although this document does not attempt to fully model Personas and Identifiers, it is worth briefly noting their relevance. A Person may present different personas in different contexts — for example, at work, at home, or across different life stages. Even within the same group, personas can change (e.g. pre- and post-marriage). Each persona may involve distinct contact channels, names, or identifiers.

Identifiers themselves (e.g., National Student Number, Business Number, Military ID, or Teaching Registration Number) are issued by a Group that is of type 'System'. These are formal, system-issued identifiers. In contrast, names (e.g. given and family names) are often pre-system identifiers — they come from family or cultural traditions and may later be recorded in civil registries.

These concepts extend beyond this paper but are important to consider when building systems that need to handle identity, continuity, or social roles across organisations and time. For more detail, see *IT Project Guidance - Data Schema Design: The Social Domain*.

## Channels

Each Person’s Persona can have a number of contact channels, which may either belong to the digital era or the pre-digital era.

In the digital era, most communication identifiers are representable as a simple trio of fields — Purpose, Type, and Identifier. Purpose might be 'Work' or 'Personal'. Type refers to the communication method — such as 'Mail' (SMTP), 'WhatsApp', or 'Mobile'. Identifier is the actual routable value — such as an email address or phone number. These combinations are sufficient to route a message or notification.

The primary remaining pre-digital era communication mechanism is the Postal Address. There have been attempts to define a unique identifier for each postal address, but to our knowledge these schemes have not been widely adopted or standardised. The postal address remains the only channel that requires multi-line formatting and structured components.

In both cases, the purpose of these channels is to provide a means for the system — or a person acting through it — to reach the intended recipient. Systems should therefore treat Channels as first-class concepts, with their own identifiers, types, purposes, and usage metadata.

Although increasingly rare, a single Channel can also be shared between people — such as a couple sharing a landline or a household sharing a mailing address. Systems should allow for the possibility of one Channel being linked to multiple Persons, especially in legacy or support scenarios.

## Location

There are services that permit determining the location of addresses, typically by interpreting the PostalCode and returning an approximate position. These are useful for estimating shipping feasibility or delivery cost. However, this doesn’t mean that an Address is the same thing as a Location. In fact, they are not reliable indicators of true physical location.

The most obvious limitation is a PO Box, which has no physical presence beyond the hosting post shop. Other examples include large university or industrial campuses, where the official street address may be hundreds of metres from the actual building in use. This makes it clear that an Address is not a reliable proxy for Location, and the two distinct concepts should not be conflated.

Instead, systems should treat Location as a distinct — and optional — object. A Location should use spatial identifiers appropriate to its purpose, such as latitude and longitude (GPS). This allows systems to reference or cache accurate, mappable positions independent of how an address is written.

Location is also nuanced. The Location of a Person or mobile object (like a vehicle) is inherently transient — changing frequently, and only valid for a point in time. Such Locations may be recorded as periodically updated caches. In contrast, the Location of a building or place is fixed and can be treated as permanent.

For this reason, it may be helpful to distinguish between two types of Location:

* **Transient Location** — used for people, vehicles, or mobile equipment, updated over time
* **Fixed Location** — used for physical buildings, sites, or spatial anchors

Both types of Location may be associated with a Person, Object, or Address — but should not be confused with the Address itself.

## Putting it Together: Social Entity Modelling

A domain defines the concepts and rules relevant to a specific area of interest. The Social Domain focuses on People, their Relationships with other People and their Roles within Groups.

Within the Social Domain, the core entity is **Person** that records birth facts (DOB, POB, DOD).

If the system has a business reason to track a person's location (admittedly very few systems do), a Person can have a single optional **Transient Location**, representing their last known or current physical presence. This value is updated as needed but is not considered permanent.

Each Person may have one or more **Personas**. A Persona represents the way a Person presents themselves in a particular context (i.e. Group) — such as a home context, professional role, or temporary affiliation.

A Persona will have identifiers issued from an external Group of type System of some kind (an education system, a school, etc.). While, technically, socially given and surnames are also identifiers, we just record them as separately as Names.

A Persona may record a Gender, which may be different from the underlying Person's biological sex.

Each Persona has a group of associated **Channels** — the ways that particular Persona can be contacted.

Each Channel has at least three key metadata properties:

* **Purpose**: such as 'Work' or 'Personal'
* **Type**: such as 'Email', 'Postal', 'Mobile', 'WhatsApp'
* **Identifier**: the value used to reach the person (e.g. phone number, email address).

If the Channel type is ‘Postal’, to provide queryability via SQL, then the Identifier should be left blank and an optional Address should be associated to the Channel. This provides the best intersection between the need for queryability while not loading Channels with unnecessary blank fields.

If the Type of a Channel is 'Postal' *and* the address is determinable (i.e. not a PO Box), then the Channel may additionally have associated to it a single **Permanent Location**. This separates the fixed spatial anchor from the communication route.

This structure allows systems to model rich, flexible identities that adapt to real-world complexity — allowing people to participate in different groups under different names or contact points, while maintaining a clear internal model of who they are and how they can be reached.

An example diagram of a starting point for the entities described is as follows:

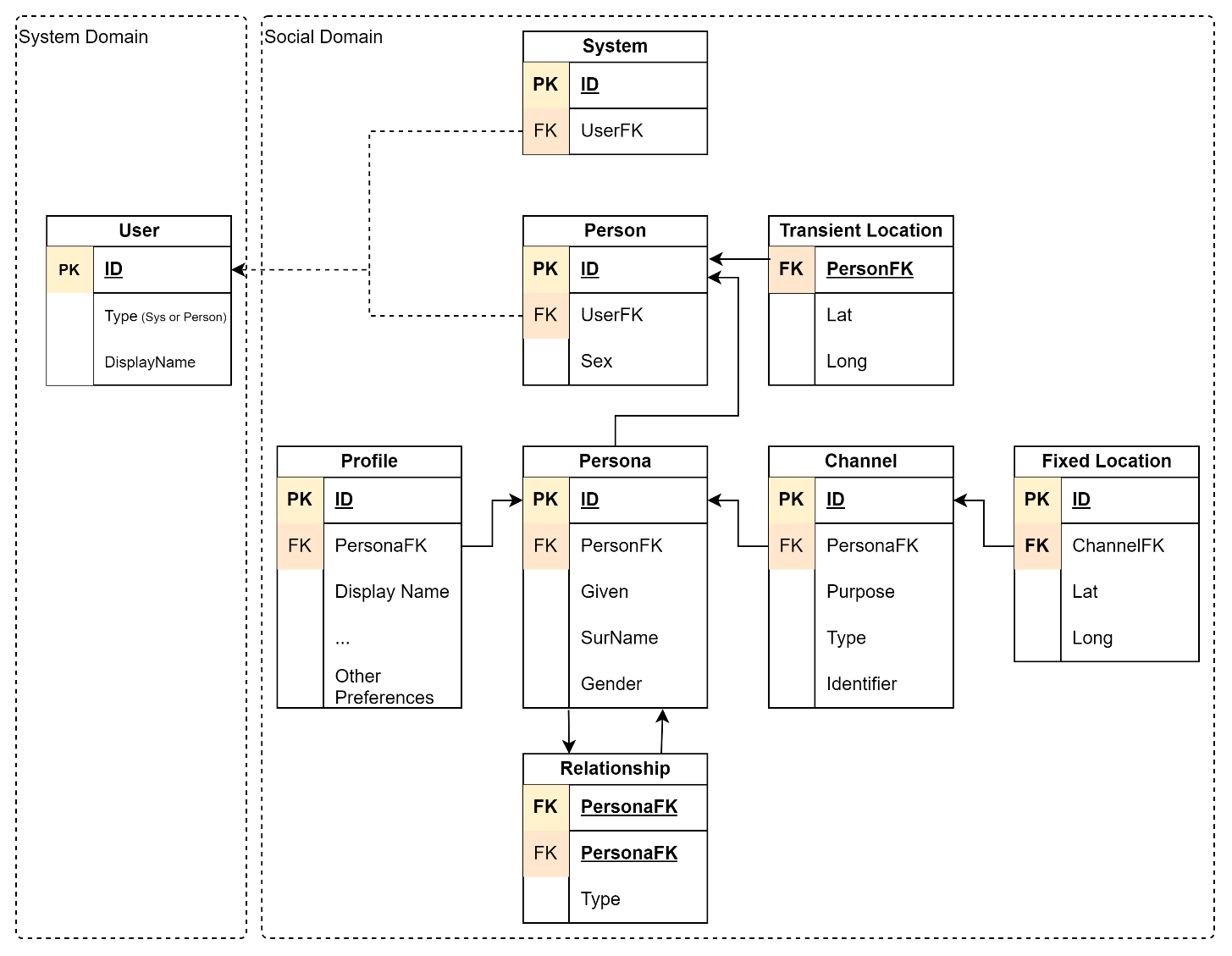


Figure 1: Example Social Domain Entities

## Further Discussion

A critical and related concern is the outdated expectation that database tables must mirror business entities one-to-one. This approach, common in early data-driven application development, encouraged stakeholders to write SQL directly against relational schemas. However, contemporary architecture has shifted toward **interface-driven design**, where APIs define the system boundary and internal storage models are implementation details, not public contracts. This aligns directly with Domain-Driven Design (DDD), which advocates for expressing clear domain boundaries and isolating underlying mechanisms — rather than exposing conflated or legacy representations that may still be used by business users. APIs are now the equivalent of database views: they can flatten, rename, or project information according to business-specific contexts, while preserving flexibility and internal consistency. This allows back-end models to evolve without requiring consumer refactoring or data migration. This decoupling allows system designers to optimise for change, performance, and clarity without sacrificing interoperability or reporting needs.

In parallel with this modelling shift, it's important to acknowledge a broader architectural evolution. Historically, search was treated as a database concern — with users or administrators querying tables directly using structured queries. Today, search requests are no longer performed against primary database records, but are routed to dedicated external search services. These services support advanced, fuzzy, and context-aware lookup capabilities.

As part of the save event, records are tokenised and enriched (e.g. with phonetic or semantic tokens), and indexed into the search service. Later searches query this index and return references (typically by Type and ID) to the matched entities. This decoupling from direct database search allows for more flexible data storage — such as storing postal addresses as JSON fragments — without impacting findability.

However, storing data in a structured, relational form remains important in contexts where systems need to expose queryable APIs. Standards like GraphQL or OData enable clients to filter entities by specific fields — such as PostalCode, State, or Channel Type — in a predictable and schema-bound way. This is why a structured Address object is introduced under the Channel: not primarily for internal logic or end-user search, but to support API-driven filtering and integration use cases, without compromising flexibility or performance.

While this paper is specifically about bringing reusable flexibility and clarity to a single entity in the Social Domain — a Person, their communication channels, and optional locations — there are a number of further concepts that could be followed up on.

One such concept is that of **Systems as Users**. In increasingly automated and service-oriented architectures, it's not just people who access systems — external systems themselves may also act as Users. These systems may authenticate, receive notifications, hold Roles, and participate in workflows. They may even have designated Channels for system-to-system communication (e.g. webhooks or API endpoints) and represent organisational presence within the same user-space model. In such cases, the User entity may include a simple UserType field (e.g. 'Person', 'System') to avoid coupling it tightly to either source model. This allows the system to resolve the acting subject efficiently without embedding knowledge of its structure directly into the User. Authentication and lookup logic can then dereference the User by type and ID, leading to either a Person or a System entity. Their participation should still be cleanly separated from the Person model.

Consider whether you need an auditable collection of transient Locations to keep a historical record of where a Person or Object has been. For example, this may be relevant for regulatory, security, or operational reasons in health, education, or logistics.

A Persona can also support one or more Profiles, used to store context-specific data. These include examples like a HealthProfile, EducationProfile, or CareerProfile. Profiles may reflect data collected or relevant to a specific life period, and it is possible to associate different Profiles to different Personas — for example, a pre-marriage and post-marriage identity, or a gender transition. Some or all of this data may be made shareable by consent.

This paper does not attempt to explore all facets of social identity, continuity, or legal representation — such as delegated authority, guardian roles, or digital agents. However, it provides the foundation for modelling individuals in a way that aligns with both lived experience and contemporary system demands.

You may also want to consider supporting **Versioning**, especially for objects like Profiles, Channels, and Personas. Versioning allows systems to retain previous states over time — whether for audit purposes, rollback, regulatory compliance, or historical context. This is closely aligned with the idea of auditability.

Similarly, introducing a formal **State** model can help control object lifecycle. States such as Draft, Review, Rejected, Accepted, Released, Endorsed, Refreshed, Merged, Removed, or Restored allow your system to track the trustworthiness, visibility, and actionability of any given item. State and Versioning together form a resilient approach to modelling change and continuity without loss of history or meaning.

Finally, in domains where identity verification is critical, consider the idea of **Endorsement** — where trusted authorities can assert or certify facts that make up a Persona or its associated Profiles. For example, a government agency could endorse a verified legal name; an educational institution could endorse a degree; or a licensing body could endorse a teaching qualification. These assertions may also be consent-based and revocable. While powerful, this concept is far beyond the scope of this paper — but worth consideration in extended identity models.

## Implementation Considerations

When translating this conceptual model into a working system, certain implementation choices can help preserve flexibility, security, and alignment with domain-driven principles.

Use an ORM to manage internal storage. This not only reduces SQL injection risks, but also reinforces that database structure is an implementation detail — not something to be accessed or reasoned about directly. Readability of the underlying storage is no longer a design goal.

Favour a **Code-First** development approach to database schema generation, rather than database-first or model-first approaches. This ensures the schema evolves from domain logic and not from storage expediencies.

Avoid exposing internal entities directly via your APIs. Instead, define and version **Data Transfer Objects (DTOs)**. This separation protects contracts, allows multiple concurrent API versions, and supports a longer transition path for consuming clients.

Where dynamic querying is a business need, expose queryable endpoints using **OData** or other standards. This reduces the overhead of delivering bespoke filter logic. If suitable, **GraphQL** can also be supported to meet developer expectations and provide fine-grained query control.

Expose all APIs through a **Gateway**, decoupling external-facing logic from internal services and enabling centralised policy, monitoring, and access control.

Finally, avoid assuming that integration platforms offer a universally suitable abstraction layer. While they can simplify internal orchestration or legacy system connectivity, they are often misapplied as end-to-end design solutions. Many integration platforms lag behind current development practices, offer limited flexibility, and can introduce long-term complexity if relied on for core domain logic or external service contracts.

## Conclusion

This paper has demonstrated how Domain-Driven Design (DDD) encourages a more disciplined approach to modelling — beginning with the identification of distinct domains and working inward toward clearly bounded entities. In doing so, we avoid encoding today's assumptions or use cases directly into storage, which often become tomorrow's limitations.

By treating identity, location, and communication as separate but related concepts within the Social Domain, we de-conflate roles and responsibilities that are often oversimplified in traditional systems. This clearer separation allows us to model real-world complexity in a way that is both expressive and maintainable.

Modern systems benefit from shifting toward interface-driven design — where APIs, not databases, define the contract with the outside world. This enables systems to evolve behind the scenes without sacrificing external interoperability. Search has been externalised and improved, and structured data is retained where necessary to support filtering, integration, and administrative needs.

We have also shown that higher-value modelling increases reusability and longevity. The data model outlined here encapsulates communication mechanisms spanning centuries — from the analogue postal service carried by horse, to today's digital-first messaging platforms like WhatsApp — without requiring structural change. That’s not just extensibility — that’s resilience.

The result is a more flexible, future-proof system — one that accommodates both the human and technical realities of identity in the modern era, and does so without locking us into outdated assumptions. This approach supports evolvability, contextual accuracy, and long-term integrity.

We encourage architects and designers to move beyond legacy models and toward a domain-aware, interface-driven mindset that better serves the systems — and people — of the future.

Appendices

Appendix A - Document Information

Authors & Collaborators

* Sky Sigal, Solution Architect

### Versions

* 1. Initial Draft
  2. Minor flow adaptions

### Images

[Figure 1: Example Entity Diagram 7](#_Toc194747252)

### Tables

**No table of figures entries found.**

### References

**There are no sources in the current document.**

### Review Distribution

The document was distributed for review as below:

|  |  |
| --- | --- |
| Identity | Notes |
|  |  |
|  |  |
|  |  |

### Audience

The document is technical in nature, but parts are expected to be read and/or validated by a non-technical audience.

### Structure

Where possible, the document structure is guided by either ISO-\* standards or best practice.

### Diagrams

Diagrams are developed for a wide audience. Unless specifically for a technical audience, where the use of industry standard diagram types (ArchiMate, UML, C4), is appropriate, diagrams are developed as simple “box & line” monochrome diagrams.

### Acronyms

API

: [Application Programming Interface](#Term_ApplicationProgrammingInterface).

### Terms

Address

: an old-school communication Channel of type “Postal”.

Channel

: a communication channel with a Person or Group.

Enrolment

: an example of a type of role – a Subscription - between a Person and a Group (e.g. of type School, which is of type System).

Gender

: the gender that a Persona identifies with. Not necessarily equal to their *Person*’s *Sex*.

Group

: a collection of (RASCI) Roles filled by People for temporary durations of time.

Location

: the Lat, Long (and optional Altitude) of a Person, Object or Address of a Group.

Person

: a physical person (at a single Location) that has one or more Personas.

Persona

: a presentation of a person within a *milieu* (ie, Group of some kind). Personas have Names and communication Channels.

Role

: the relationship between a Persona and a Group

Sex

: the biological attribute of a Person. See Gender.

User

: a system or human user of a system via its UIs.