IT Project Guidance -- On Cross-System User Identity Attribute Resolution

Version: 0.1

Purpose

This document defines architectural and operational guidance for managing user identity and attribute data across systems. It distinguishes between identity providers, local user records, and the appropriate handling of contextual or entitlement attributes. It addresses common integration errors, proposes boundary-respecting patterns, and clarifies principles for secure, extensible, and maintainable user data flows in federated environments.

Synopsis

Many organisations operate in multi-system environments where user identity and entitlements are distributed across platforms. Misunderstandings around identity token design, attribute ownership, and user lifecycle responsibilities lead to fragile, overcoupled systems. This document clarifies the distinct roles of identity providers and consuming systems, and proposes defensible patterns for retrieving user attributes via trusted APIs instead of enriching tokens. It advocates for separation of identity and context, consistent user correlation, and system-local ownership of authorisation logic.

Contents

Purpose and Audience

This document provides design guidance for managing user identity and attribute resolution across systems. It clarifies the distinct responsibilities of identity providers (IdPs) and consuming systems, reinforces trust boundaries, and outlines correct practices for resolving user attributes in multi-system environments. The guidance is written for architects, integration designers, security officers, and system owners who are responsible for user lifecycle, authentication, and role management.

Scope

This guidance applies to systems that authenticate users through an external identity provider (IdP) and require additional attributes to authorise or personalise their behaviour. It covers the separation of concerns between authentication and authorisation, cross-system user correlation, and the retrieval of identity and context data. It is relevant to both cloud-native and hybrid environments, and assumes the presence of federated identity practices such as OIDC or SAML.

Background

In organisations of any meaningful scale, it is uncommon—and operationally unwise—for all systems to run on a single platform. For reasons of business continuity, commercial independence, vendor lock-in, and service diversity, most ecosystems are composed of distinct systems handling different aspects of the business. Each of these systems has value to contribute and receives users through a variety of interfaces and workflows.

Even if an organisation were to adopt a single-platform strategy—relinquishing architectural independence and binding its business services to one vendor's tooling—it cannot escape the fundamental reality: no vendor offers a complete solution for all services. The IT landscape evolves too quickly, with innovation, market disruption, and sector-specific needs outpacing any one platform's capacity to keep up. Relying on a single platform narrows choice, introduces vendor lock-in, and increases the cost and complexity of future change. It limits how effectively an organisation can adapt to user needs, market pressures, and service innovations. Even if such a strategy were pursued, many services—particularly those integrated externally or provided through sector partnerships—would still fall outside the chosen platform. The integration challenge remains.

To unlock that value, systems often need to share user-related *context* information: for example, a learning platform may need to know what class a student is in, or a health portal may need to know what clinic a user is associated with. However, attempting to centralise or preload this data into the IdP is a design error that leads to a range of problems—technical, operational, and architectural.

Establishing Foundations: Identity, Roles, and Context

Each System Has Its Own Users

In federated systems, each system has its own user records. Even when using a shared Identity Provider (IdP), the consuming system maintains its own internal user representation.

The IdP is also a system in its own right. It has users—its users—and its primary function is to expose - provide - a user's *identity* (excluding credentials) attribute data related to them. Hence Identity Provide

The consuming system *also* has users—its own records, profiles, and entitlements. While these two users relate to the same persona, these two system user entities are not the same; they are only linked by a trustable correlation, not by shared instance or state.

What actually happens when a user attempts to access System A is they are redirected to an IdP, which authenticates them and returns a token containing some of the user's attributes. System A then validates this token to trust it and uses any IdP user information embedded within it to in turn locate or create a local System A user record. Once created, this user record, in a different system, is distinct from the IdP's.

From that point on, System A manages its own system user record internally, using its own mementos to continue session user information (cookies or headers). The IdP's token is *not* reused beyond the initial session establishment.

Local access, authorisation, personalisation, and auditing are based on System A's internal data—not the IdP.

Clarifying Types of Attributes

Not all attributes are equal in purpose, trust boundary, or source of authority. A clear distinction must be made between attributes that identify a person (who they are), and those that describe a relationship (what they are to something else).

The most stable and reusable identity tokens carry only identifier-level attributes—such as subject ID, username, or system-scoped persistent identifier. These are sufficient to establish a session and perform secure lookups elsewhere. In some cases, attributes like name, preferred name, or date of birth may also be included, but these should be treated as soft identifiers, not definitive sources of truth.

In contrast, contextual attributes describe the user's relationship to a system, organisation, or role. Examples include whether someone is a student *at* a particular school, a teacher *of* a class, or a caregiver *for* a child. These are not identity attributes—they are context attributes. Their truth and change frequency depend on the business system that governs the relationship.

It is incorrect to preload identity tokens with such contextual data. Not only does this violate separation of concerns, but it also creates brittle, stale, and hard-to-audit dependencies. Instead, these attributes must be retrieved via controlled API calls to the systems that authoritatively manage those relationships.

Authorisation is Local

It is important to distinguish between authorisation *to use* a system and authorisation *within* a system. The same term is often used to describe both, which has led to persistent confusion in system design.

An Identity Provider (IdP) can confirm that a user is who they claim to be, and provide access *to* a system—but it is the system itself that must determine what the user is allowed to do *within* it. These scopes are not interchangeable.

The logic for both assigning permissions and authorising actions must be embedded in the system delivering the business function, not delegated to external identity infrastructure.

Using IdPs to manage permissions leads to confusion and fragility. Every system should determine its own permission, role, and access logic. A label like 'teacher' may mean something different in a learning system than in a reporting portal or finance application. Role logic must be tailored, maintained, and auditable within the consuming system.

Misusing Other Services to Provide Authorisation

Maybe the flawed perspective came from back in the day when administrators used Active Directory's Organisation Units (OUs) for a different purpose than they were designed for. OUs were intended as a way to organise devices and users by their place within an organisation—not as a role management mechanism. However, with a desire to manage permissions and lacking a dedicated system for roles, these OUs were often appropriated as proxies.

But an OU doesn't distinguish between managers and managed, or capture role-specific behaviour. In some environments, this led to the proliferation of OUs such as 'GroupXYZ_Managers' and 'GroupXYZ_Managed' in an attempt to represent roles. Even though this became common practice in the era of AD, and continues to some extent in AAD through the use of Groups (again, not Roles), that widespread use does not validate it as sound design.

It is worth noting that Active Directory Security Groups came closer to expressing access logic. They were routinely used to grant access to resources like file shares, applications, and email distribution lists. However, Security Groups still lacked the richer semantics of roles: they did not capture purpose, context, or behavioural expectations, and they had no inherent logic for scoping, time-bounding, or chaining entitlements. Security Groups were static collections of users, and the consuming systems had to interpret them in their own way, often inconsistently. As a result, while they served practical needs, they did not amount to role management in the architectural sense.

In Azure Active Directory (AAD), traditional OUs no longer exist, and Security Groups have been replaced by cloud-based Group constructs. These Groups are commonly used to manage access to applications and resources but still do not equate to roles. They remain flat membership containers with no native semantics for behavioural logic, contextual scope, or time-based entitlements. While they may be integrated into access control flows, they require consuming systems to interpret and act on them explicitly. Thus, Groups in AAD continue the pattern of being helpful for managing "access to" surfaces—but fall short of true "role within" modelling, which must still be defined within the system consuming the identity. The IdP’s token is not reused beyond the initial session establishment. Local access, authorisation, personalisation, and auditing are based on System A's internal data—not the IdP.

Identity (only) Brokering

A secondary source of confusion arises when identity brokering is introduced. In federated environments, an intermediary broker may authenticate users by delegating to one or more upstream IdPs—for example, a national education IdP that federates to cloud-hosted school-based identity services. While brokering authentication is valid and necessary, the concept is sometimes stretched beyond its design, to brokering other information, as well as context and authentication.

It may seem convenient to treat the broker as a bridge not only for identity authentication, but also for aggregating or injecting user attributes from other systems. This conflation is problematic. An identity broker is not a general-purpose integration hub. Its role is to negotiate identity assertions—not to serve as a conduit for cross-domain attribute resolution.

Confusing these responsibilities leads to architectural drift: tokens become overloaded with external context, systems become dependent on indirect data sources, and the clarity of identity boundaries is lost. The result is fragile integration, poor traceability, and an increased surface area for data breaches. In federated systems, each system has its own user records. Even when using a shared Identity Provider (IdP), the consuming system maintains its own internal user representation.

Context Belongs Where It Is Created

Systems sometimes preload tokens with context—such as enrolment, roles, or organisational affiliation—that originates from other systems. These attributes are not identity. Nor are they attributes of the Persona - they are instead describing a persona’s *relationship* to a Context.

Being not attributes of an identity they should not be managed by an Identity Provider. Instead, they should be resolved at runtime via deliberate calls to the system that owns them.

Embedding such data in tokens violates boundaries, reduces auditability, and disperses entitlements. Instead, consuming systems should fetch what they need from source systems using authorised, traceable interfaces.

Common Misconceptions About Brokering

Some argue that brokering improves speed, availability, or caching. However, these assumptions rarely hold under scrutiny. An Identity Provider is designed to rapidly authenticate users—not to act as a query router for arbitrary attributes. Adding indirect lookups to third-party systems via the broker introduces potential latency and unpredictability. Without rigorous load testing, it is unclear whether such designs would introduce thread exhaustion, blocking calls, or subtle degradation in system responsiveness. The result may be hard-to-diagnose availability issues.

Caching such data inside the IdP in an attempt to solve for performance introduces other risks. Since the IdP has no direct triggers from the upstream source systems, any cache refresh window is arbitrary. Some data will always be stale—yet the consuming systems cannot reliably tell what is out-of-date or how critical the change might be.

Others argue that using the broker avoids point-to-point integration. This too misses the mark. While point-to-point proliferation is a concern, it is not the IdP’s responsibility to fix it. An API Gateway, service mesh, or other dedicated interface mediation layer can provide controlled indirection, payload transformation, and lifecycle stability without overloading the identity layer.

Turning the identity broker into a generic lookup engine undermines trust boundaries, escalates operational risk, and obscures data ownership. It is not a sustainable approach.

Sustainable Approach

A sustainable approach to identity integration separates identity authentication from context acquisition. Identity Providers confirm who someone is. Consuming systems decide what that person can do. Authorisation logic must be embedded where it is exercised. Contextual attributes must remain governed by the systems that create and maintain them. Brokering, while essential, must not become a vector for uncontrolled attribute injection. Sustained integrity requires discipline in how we assign, share, and resolve user information across system boundaries.

The identity token should contain only what is strictly required to identify the user—typically a stable, scoped identifier. Once the system starts a session, it retrieves contextual or role-specific attributes directly from the authoritative systems via secure, auditable APIs. These attributes might include enrolments, group memberships, roles, or other associations. This ensures each consuming system defines its own logic, retrieves only what it needs, and maintains clean boundaries.

These roles and attributes are applied within the consuming system, and that system is responsible for updating or revoking them if they change. Where authorisation sensitivity is high, systems should implement revalidation checkpoints or limit session lifespans accordingly. Systems must also handle cases where attribute services are unavailable or return incomplete data. In such cases, the system should apply defensible defaults or restrict access until sufficient context is retrieved.

This architectural guidance is realised in a specific, repeatable implementation pattern, described in **Appendix B**. That appendix outlines the deferred attribute resolution design, where contextual attributes are retrieved after session establishment. For this to function correctly, systems must know which types of attributes are suitable for identity tokens and which must be deferred. That classification is provided in **Appendix C**. To support actual API integration, consuming systems must also be authorised to query attribute sources. The technical steps and flows to do so securely using OAuth 2.0 Client Credentials are described in **Appendix D**.

Together, these appendices operationalise the principles described above.

This model supports reuse, auditability, and long-term sustainability of federated identity. It avoids brittle integrations, improves traceability, and aligns with the principle of least privilege.

Conclusion

Identity and access responsibilities must be clearly partitioned. Identity providers confirm who someone is. Consuming systems decide what that person can do. Authorisation logic must be embedded where it is exercised. Contextual attributes must remain governed by the systems that create and maintain them. Brokering, while essential, must not become a vector for uncontrolled attribute injection. Sustained integrity requires discipline in how we assign, share, and resolve user information across system boundaries.

Appendices

Appendix A - Document Information

Authors & Collaborators

- Sky Sigal, Solution Architect

Versions

0.1 Initial Draft

Images

Tables

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.

DDD

: Domain Driven Design

GUI: Graphical User Interface. A form of UI.

ICT: acronym for Information & Communication Technology, the domain of defining Information elements and using technology to automate their communication between entities. IT is a subset of ICT.

IT : acronym for Information, using Technology to automate and facilitate its management.

UI : User Interface. Contrast with API.

Terms

Refer to the project's Glossary.

Application Programming Interface : an Interface provided for other systems to invoke (as opposed to User Interfaces).

Capability : a capability is what an organisation or system must be able to achieve to meet its goals. Each capability belongs to a domain and is realised through one or more functions that, together, deliver the intended outcome within that area of concern.

Domain : a domain is a defined area of knowledge, responsibility, or activity within an organisation or system. It groups related capabilities, entities, and functions that collectively serve a common purpose. Each capability belongs to a domain, and each function operates within one.

Entity : an entity is a core object of interest within a domain, usually representing a person, place, thing, or event that holds information and can change over time, such as a Student, School, or Enrolment.

Function : a function is a specific task or operation performed by a system, process, or person. Functions work together to enable a capability to be carried out. Each function operates within a domain and supports the delivery of one or more capabilities.

Person : a physical person, who has one or more Personas. Not necessarily a system User.

Persona : a facet that a Person presents to a Group of some kind.

Quality : a quality is a measurable or observable attribute of a system or outcome that indicates how well it meets expectations. Examples include reliability, usability, and performance. Refer to the ISO-25000 SQuaRE series of standards.

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

User Interface : a system interface intended for use by system users. Most computer system UIs are Graphics User Interfaces (GUI) or Text/Console User Interfaces (TUI).

Appendix B – Deferred Attribute Resolution Pattern

This appendix provides the concrete implementation pattern for what was earlier described in this document as the "Sustainable Approach." While the earlier section articulates the architectural principle—that identity and context must be kept separate—this section shows how to realise it in practice. It provides a repeatable design that supports the sustainability, auditability, and clean system boundaries outlined in the core guidance.

A robust identity integration pattern separates authentication from contextual authorisation. It does this by keeping identity tokens minimal and resolving additional attributes only when needed, via secure, auditable API calls.

In this model, a user authenticates through an Identity Provider. The returned token contains only a scoped unique identifier. The consuming system uses this to locate or create its own internal user record.

Only after establishing a session does the system retrieve attributes such as enrolments, roles, or associations—based on its own logic. This is done by calling the appropriate external attribute or context services. What is fetched, when, and from where is the responsibility of the consuming system. This encourages independence, minimises coupling, and enhances traceability.

This avoids reliance on tokens preloaded with irrelevant or stale data. It allows every system to retrieve exactly what it needs, on demand, from a controlled source.

A common objection is that tokens from upstream IdPs or SaaS systems cannot be changed. This is correct—but irrelevant. If a system needs external information, it must fetch it explicitly, not rely on a hard-coded assumption that the token will be enriched.

Appendix C – Attribute Categories and Boundaries

Understanding attribute types is critical to avoiding misuse. Identity attributes describe the individual (or persona) and are appropriate for inclusion in identity tokens. These include:

- Persistent Identifiers (sub, sid, user_id)
- Legal and preferred names
- Date of birth, place of birth
- Identity provider and source domain

These attributes do not change often and define the authenticated subject.

In contrast, contextual attributes describe relationships or roles. They are not part of the user's identity, but rather their situation:

- Enrolment in a school or class
- Employment or teaching assignments
- Group membership or entitlements
- Location-based access rules or support eligibility

These attributes are owned by the systems that manage those relationships. They must be queried separately, and should not be embedded into tokens by the identity layer.

A third category includes identifiers in other systems (e.g. StudentIdAtSchool, EmployeeCodeInHRSystem). These are not identity attributes—they are foreign keys. Their management belongs with the system of origin.

Appendix D – Client Credential Access to External APIs

When a system needs to retrieve attributes after authentication, it must act as a trusted client of the attribute source. This is done using OAuth 2.0 Client Credentials Flow.

Prerequisite Configuration

Before any attribute resolution can take place, the consuming system must be registered with the attribute provider's authorisation server. This involves:

- Registering the client and issuing credentials: a client ID and secret, certificate, or key.
- Defining scopes: what endpoints and data the client is permitted to access.
- Storing credentials securely in the consuming system, ready for use at runtime.

This setup is done once and reused for subsequent operational flows. Credentials do not round-trip with the user.

Operations: End User Sign-In and Attribute Retrieval

1. The user signs in to the application via the Identity Provider. A token is returned containing a unique identifier (sub, sid).
2. The system uses this to find or create a local user record and begins a new session.
3. At this point, the system calls external APIs (e.g. a role or entitlement service), using its own client credentials to obtain an access token.
4. With the access token, it queries only the data it requires for that session.

Ongoing Requests and Data Caching

- The access token may be cached and reused for its validity period (usually short, e.g. 1 hour).
- Contextual data may be cached for the duration of the user's session if appropriate.
- There must be no assumption that cached data is still fresh beyond that point.

User Credential Refresh

- If a session is long-lived or requires revalidation of access, the system can force reauthentication with the Identity Provider.
- However, user tokens are separate from attribute service tokens; the system may independently refresh its own client token as needed.

This model ensures strong separation of concerns, traceability, and conformance to least privilege. Systems ask only for what they need, when they need it, using explicitly granted access rights.