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

Version: 1.4

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, tightly coupled systems. This document clarifies the distinct roles of identity providers and consuming systems, and proposes best practice 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

Clarifying Types of Attributes

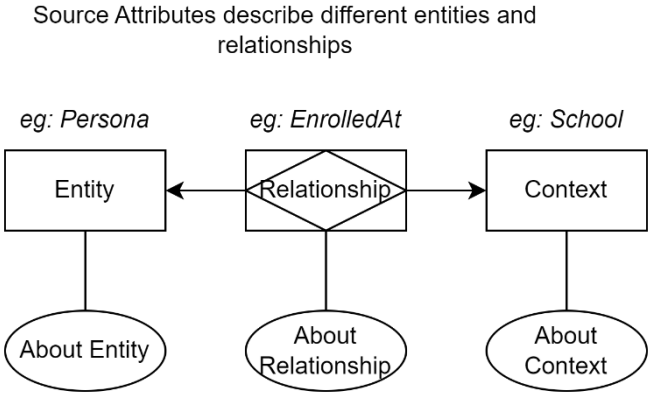

Figure 1: Different types of attributes

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.

Each System Has Its Own Users

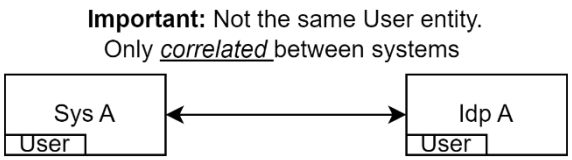

Figure 2: Each System has its own Users.

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 by System A beyond the initial session establishment.

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

Authorisation is Local

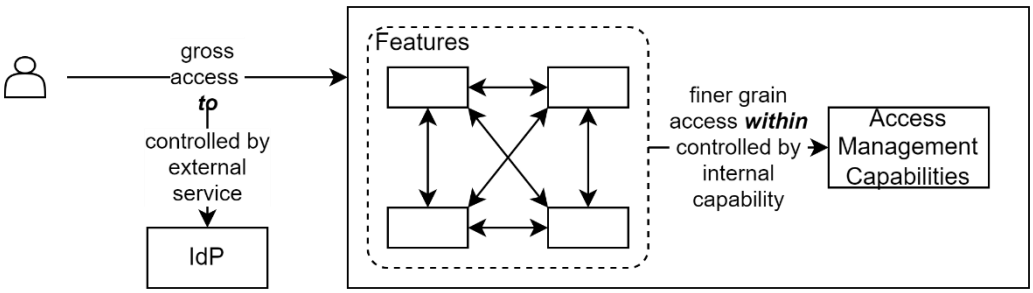

Figure 3: Access To is distinct from Access Within a system

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.

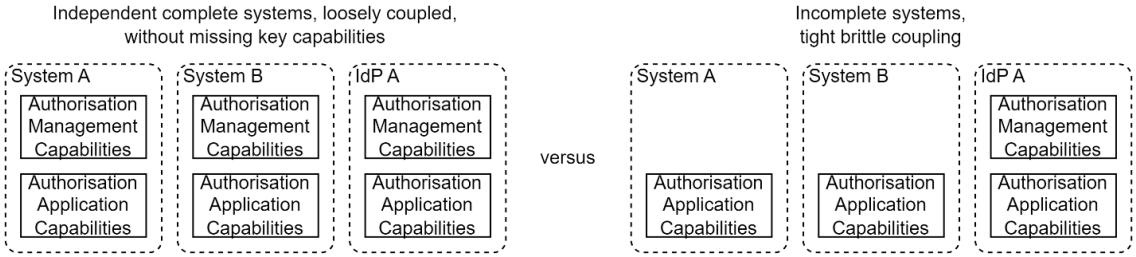

Figure 4: Authorisation Management and Implementation is Local

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.

> *Note:*
> *AAD groups are suitable to manage access **to**, not access **within.***

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

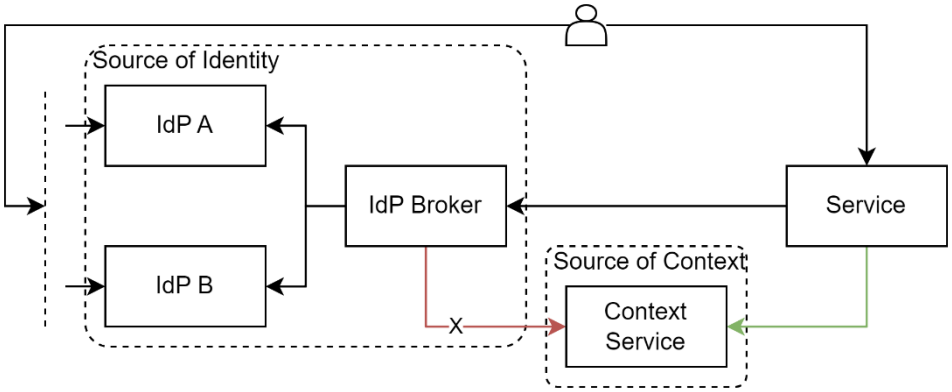

Figure 5: retrieve identity and context from separate, appropriate, sources

It may be claimed that brokering improves overall system responsiveness or reduces integration overhead. However, this potential advantage hides disadvantages.

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 3rd party data inside the IdP 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.

Additionally, the authority to release such information will be misattributed. While the source system may permit access to specific data by the consuming service (in this case, the IdP), that authorisation typically does not extend to the end user being authenticated. Unless the IdP replicates the source system's authorisation logic, it may expose data that the original system would have withheld—resulting in a silent breach of access boundaries and policy intent.

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.

Sustainable Approach – BaseLine

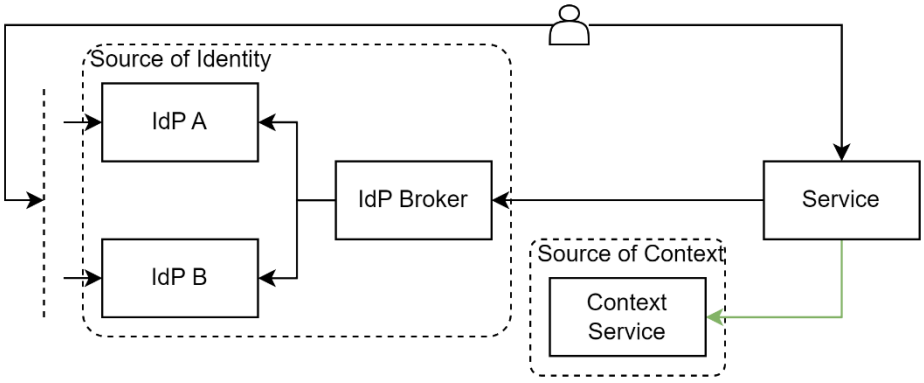

Figure 6: retrieve identity and context from separate, appropriate, sources

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 safe 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.

Route Management

A recommendation would be to recognise an API Gateway should be interjected in between the source context service and the consumer. This would permit redirection, even restructuring the message as required, to a new service without requiring service consumers be reconfigured to call the new service.

Sustainable Approach – Extended for Identity Resolution Over Time

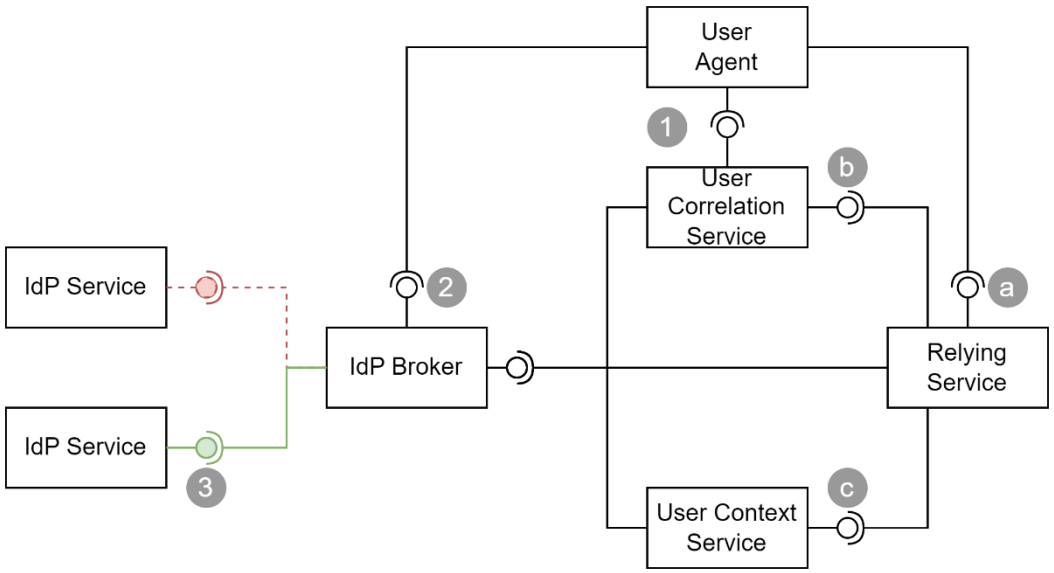

Figure 7: User Correlation Service

Modern federated identity systems can reliably confirm who someone is within a single organisational context.

However, they provide no guarantees about identity continuity across organisations or over time. This limitation is particularly acute for industry consultants and education, where learners and staff transition between institutions regularly, and each institution may operate its own Identity Provider (IdP) using different domains, providers, and subject identifiers.

The first Sustainable Approach (now Option A) separates identity from context and promotes attribute resolution via source systems. This remains necessary.

However, it is not sufficient on its own. We must also account for the fact that the same Persona may sign in later under a different IdP, with different identifiers, and that those changes are often invisible to the systems consuming the authentication tokens.

Why Identity Resolution is a Distinct Need

Identifiers such as sub (subject ID) and email are generally scoped to the IdP issuing them. When a person transitions to a new institution, their identity token will change:

- A new iss (issuer)
- A new sub (subject identifier)
- A new email, typically the suffix scoped to the organisation
- A User Principal Name (UPN), typically used in AAD, based on current email

None of these allow reliable correlation to earlier logins unless external continuity is established.

Identity Providers rarely supply long-lived identifiers such as national student numbers (NSNs) or government-issued IDs in their claims. And even if they did, embedding them in tokens would violate the very design principles this document defends.

The Role of a Continuity Service

Sustainable design therefore requires a new architectural role: a **User Continuity Service**, separate from any individual IdP or consuming system. Its sole purpose is to record and expose the linkage between different identities over time—each represented by an iss:sub tuple—and associate them with a durable internal User record.

This service must:

- Accept new identity linkages, either inferred or user-initiated
- Store multiple DigitalIdentity entries per User
- Support query by iss:sub, email, or other soft identifiers
- Return a system-scoped UserId for use by consuming systems

- Enforce security, traceability, and consent in all operations

This service does **not** provide authentication. It is not an IdP, nor a profile service. It is a correlation registry, providing continuity and lookup—nothing more.

User-Assisted Identity Linking

Where authoritative correlation is not available (e.g. no NSN, no enrolment record), and no continuity service exists yet, user-assisted linking may provide a workable fallback. However, the common assumption that a user can "log in with their old identity" is fragile. In many education environments, once a user leaves an institution, their previous login method (e.g. old school email) is disabled.

To address this, systems should instead:

1. Allow users to authenticate with their **current** IdP (e.g. new school)
2. Provide a "Link to prior account" option in their settings
3. Prompt the user to enter a known prior email address or identifier
4. Send a verification challenge (e.g. email with nonce) to that prior address
5. If the user still controls it, they return and confirm the linkage
6. If the user does not control it (account closed), validation can be done by a third party.

Where successful, the system links the new identity (iss:sub) to the previous User record, creating an enduring association.

If the user requires assistance—due to age, disability, or other support needs—the process should allow for delegated facilitation by an authorised caregiver, teacher, or support person, using clear consent and audit controls.

If the prior email is no longer accessible (account disabled or closed), linkage must rely on backend resolution services—or validation by a trusted third party able to confirm continuity, such as a sector body, registry service, or support desk process authorised to mediate identity claims. If no such process is available, the identity may remain unlinked. At no point should this process involve speculative token enrichment.

This user-driven flow maintains auditability and trust while avoiding brittle inference. It can supplement—but not replace—the need for a system-assisted resolution layer.

Risks of Inaction and Misapplied Alternatives

If a continuity service is not available, systems must either accept broken identity chains or attempt unreliable matching on soft attributes like name and date of birth. These patterns degrade quickly at scale, introduce privacy risk, and cannot support role-sensitive authorisation.

Some architects respond by enriching identity tokens with long-lived attributes. This practice is **specifically discouraged**. It violates separation of concerns, introduces un-auditable trust assumptions, and limits system evolution.

Moreover, enriching the broker with correlation logic introduces an unresolved responsibility gap. The broker is neither the authoritative Identity Provider nor the relying system responsible for authorisation. If mismatches occur, no party has full visibility or accountability. Incorrect linkages can result in inappropriate access, misrouted data, or silent merging of unrelated user records—problems which are difficult to detect and often impossible to reverse.

A common counterargument is that enriching tokens simplifies vendor integration by reducing the number of systems each Relying Party must query. However, this convenience comes at the cost of architectural soundness, auditability, and long-term maintainability. What appears simple to the vendor introduces hidden complexity in governance, trust boundaries, and change management. It places correlation logic in a context (the token) where it cannot be verified, queried, or corrected after the fact. This may reduce short-term friction but invariably increases long-term risk.

It is important to recognise that the service needed to generate and maintain such enriched data must still be developed—regardless of where it is used. Whether tokens are enriched or attributes are retrieved, some system must be responsible for managing identity continuity. The only real difference is **who is expected to carry the risk**. Enriching the IdP token assigns responsibility to the broker, a party that cannot fulfil it. Delegating that task to the consuming service aligns responsibility with visibility, traceability, and system context. Complexity is not avoided by enrichment—only misassigned.

If identity continuity is a requirement—and it often is—it must be solved by designing for it explicitly, not retrofitting it into an authentication flow.

This approach introduces non-trivial design effort, but it reflects the reality that continuity of identity in federated systems is inherently complex. Attempting to conceal that complexity through token enrichment does not remove the need for correlation—it only buries it in a place that is unauditable, unauthorised, and unsustainable. Designing explicit resolution pathways makes the complexity visible, traceable, and manageable, supporting secure and evolvable systems over time.

What Relying Parties have to do

To use this service, relying parties (RPs)—such as SaaS platforms, LMS tools, and other service providers—must participate in a simple but structured integration model.

This does not require handling the full complexity of correlation logic, but it does require a few clearly defined responsibilities:

1. **Authenticate via IdP**: The RP receives a token including iss, sub, and email (or UPN).
2. **Attempt Identity Resolution**: If the incoming iss:sub is not recognised, the RP queries the **User Continuity Service**, providing the current claims (e.g. email, iss:sub).
3. **Interpret Response**:
    - If the continuity service confirms a match to an existing UserId, the RP should create a new DigitalIdentity record under that user.
    - If there is no match, the RP may prompt the user to initiate a linking flow, or treat it as a new user (according to local rules).
4. **Avoid Direct Inference**: RPs must not attempt to match users on unverified combinations of name, DOB, or email unless explicitly delegated.
5. **Maintain Internal Ownership**: Once a local User record is established, all subsequent access, role logic, and auditing remains the responsibility of the RP system—not the IdP or continuity service.
6. **Handle New Users Gracefully**: If the continuity service confirms there is no match—and the user does not attempt to initiate a linking process—the RP should treat the identity as a legitimate new user. This includes creating a new User record, assigning any context roles as appropriate (e.g. from enrolment services), and proceeding with onboarding. This distinction between unlinked and new is essential: lack of correlation is not an error, but a fork in the logic path that must be anticipated and handled explicitly.
7. **Retrieve Context Separately**
   Any additional attributes—such as enrolment, roles, or group membership—must be retrieved via dedicated API calls to authoritative context services, as described under Sustainable Option – Baseline. These attributes must not be inferred from token contents or reused across sessions without validation.

This model minimises vendor-side complexity while preserving trust, visibility, and long-term maintainability. Enrichment of tokens is not required. What is required is one additional API call to a continuity resolver at the moment identity correlation is uncertain.

This model minimises vendor-side complexity while preserving trust, visibility, and long-term maintainability. Enrichment of tokens is not required. What is required is one additional API call to a continuity resolver at the moment identity correlation is uncertain.

Correct Implementation is Key

Implementation according to the OIDC specification is critical to obtain expected trustable outcomes.

For long serving systems required to disambiguate users with potentially the same email, refer to Appendix E.

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

1.0 Initial Release

1.1 Added information about **sub** attribute.

1.2 Adjustments based on feedback by Alan Heward

1.3 Addition of Appendix F based on feedback by Mike O'Connor

1.4 Diagram for Appendix F

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

Alan Heward, Senior Advisor, Digital Assurance	
Mike O'Connor, Enterprise Architect (Security)	
Harry Nguyen, Lead Cloud Engineer	

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.

Appendix E – Use Sub to disambiguate users

While email is conceptually sufficiently unique to disambiguate users, emails are insufficient for this purpose in long living, large user base, systems such as government systems for education. While sufficient to disambiguate two parallel learners with the same name (john.smith@schoolA.org versus john.smith@schoolB.org, or even john1.smith@schoolA.org and john2.smith@schoolA.org), emails may be reused over time (john.smith@schooA.org in 2010, and john smith@schoolA.org in 2020).

For this, the OIDC specification provides the solution:

- **OIDC Core 1.0 Specification – Section 2:**
  Subject Identifier
  A locally unique and never reassigned identifier within the Issuer for the End-User, which is intended to be consumed by the Client.
- **Section 5.1 – Standard Claims:**
  sub
  Subject - Identifier for the End-User at the Issuer.
  It is a **locally unique and never reassigned identifier** within the Issuer. It **must not be personally identifiable information (PII)** such as an email address.

However, a poorly implemented broker *can* (incorrectly) use email as the **sub**.

If that is the case, and email addresses are reassigned (e.g. two John Smiths reusing jsmith@school.nz), the sub becomes unstable—violating OIDC intent and making identity correlation unreliable.

Options to fix this include:

- Return sub to not being an email, **and** either make it:
  - Only Locally unique, while instructing clients use a composite key (iss+sub) to find identity records associated to a system user, **or**
  - Be globally unique (a UUID is satisfactory), while instructing clients use a composite key (iss+sub) *anyway* to find identity records associated to a system user (the addition of the iss prefix provides higher entropy), **or**
- Instruct brokered IdPs use a valid from/to (least preferred option, due to change management complexity).

Appendix F – IdP Migration and Sub Mismatch Handling

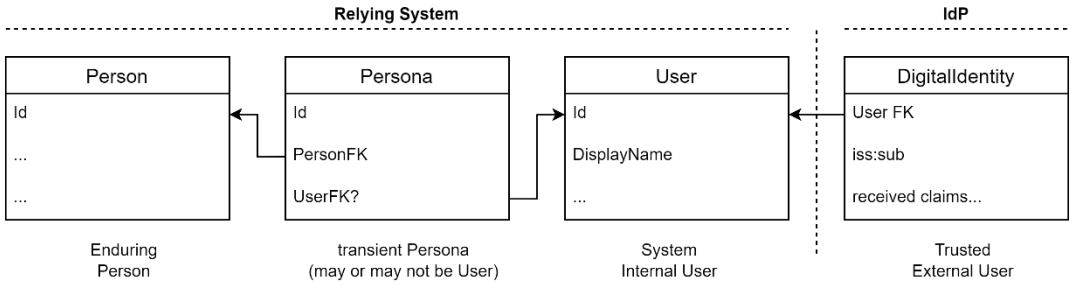

Figure 8: An internal User may be associated to multiple external identities

In federated identity architectures using OpenID Connect (OIDC), a common approach is to use a broker to manage authentication across multiple Identity Providers (IdPs). In education contexts, each school may use a different IdP (e.g. Google Workspace, Microsoft AAD), and this configuration may change over time. When a school migrates from one IdP to another, the `sub` claim—used as a stable unique identifier for the user—is not preserved. The `iss:sub` tuple, used as a composite identity key, is therefore no longer valid for returning users whose IdP has changed.

This presents a challenge for consuming systems that rely on `iss:sub` to resolve a known user. The new login will appear as a previously unseen identity, even for the same student, leading to potential account duplication or access denial. Historical DigitalIdentity records will still exist, but none will match the incoming identifier, and systems may not know how to reconcile them.

Recommendation

Systems should treat an unmatched `iss:sub` as a new login source, and initiate a secondary matching process to link it to an existing user. Specifically:

1. Upon receiving an unknown `iss:sub`, the system should attempt to find a User by searching existing `DigitalIdentity` records using the provided email address.
2. Where multiple results exist, the system **must** prefer the most recently modified User record, assuming this represents a current or recent enrolment.
3. If a match is confirmed, a new `DigitalIdentity` record should be created using the new `iss:sub`, and linked to the existing User record. This ensures that subsequent sign-ins resolve directly via the new identity.
4. Systems should preserve older `DigitalIdentity` entries to retain traceability of past authentications, but treat them as read-only.

This approach accommodates changing IdPs without requiring retrospective updates or manual reconciliation. It ensures that identity linkage is robust, gradual, and accommodates the reality that only currently enrolled learners are likely to authenticate. It

also supports graceful evolution over time, allowing systems to respond to federation changes without compromising access or trust.