OpenAutonomy Technical Overview

Open Autonomy Inc.
January 2014

Abstract:

As more communication networks, data, and applications are moved onto the internet, a problem of connecting these various services has arisen. While most providers are building centralized *walled gardens*, this limits interoperability, consumer choice, information security, and innovation potential. This document outlines a novel approach for connecting these services across the internet **without a centralized authority**.

Introduction:

Historically, users had complete control over the data they owned and the applications they used. This gave them strong security and control over who could access their data and how the information would be used. Additionally, it was trivial to combine many applications as part of complex work-flows through file types and interprocess communication facilities provided by desktop managers and operating systems. Even in simple environments, the file system operated as a universally shared medium for the user's applications and data.

As the internet has come to provide services which replace these local applications, the ability to control and protect information has not caught up, nor have the facilities to use these applications together. This problem is most apparent within the domains of **social networking** and **cloud storage** where these problems have caused security and privacy concerns for users and continue to limit the market penetration of internet-based applications which need access to the user's data.

OpenAutonomy is a protocol for connecting these services to their owning users while further allowing the users to selectively control the access of services and other users to their private data. The decentralized nature of the protocol also reaffirms the nature of the internet as a distributed system in that it **requires no central authority** to mediate this communication, nor its user authentication or trust mechanisms.

This paper outlines the high-level design of the protocol and how it can be used to solve these common problems.

Design Goals:

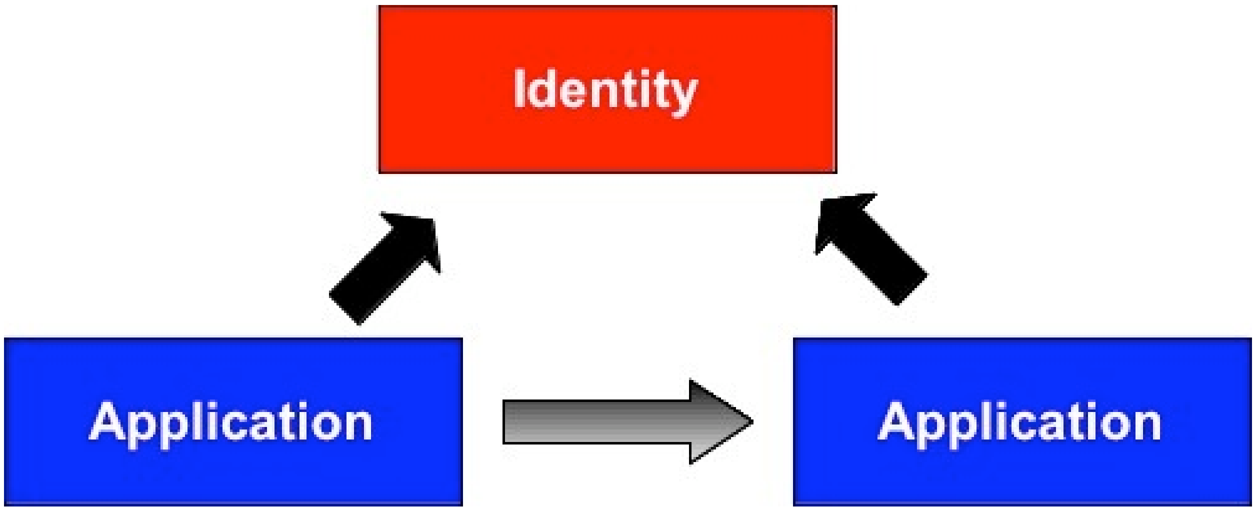
- There can be no dependence on any central service provider.
- There can be no assumptions made regarding where a service exists, relative to other services. This means that there is no assumption of monolithic servers, anywhere in the network.
- The system must be easily extended to support high-level interaction between applications which are not part of the reference implementation. This extension should not require modifying the reference implementation to be possible.
- The system must provide a way to mitigate or limit the intrusion of spam or untrusted third-parties, be these users or services.
- There must be no specific implementation language requirement.
- It must be possible to run a valid implementation on low-end hardware or in highly restricted environments. This means no dependence on long-lived or persistent server processes.
- Native or non-web applications must be able to consume services as first-class entities, although they may not be able to provide them.
- Communication between 2 applications, while potentially established by a trusted third-party, must not be required to pass through the third-party as this would be a bottle-neck.

High-level Outline:

OpenAutonomy has 2 concepts: **identity** and **application**. An identity represents a user and an application represents a service. Applications may be stand-alone, may provide services to other applications, consume services from other applications, or both. All applications have a *creator* identity. There is no requirement that any of these exist on the same physical server or are addressed at the same IP or domain name.

The interaction between applications and the identity flows similarly to a micro-kernel's services and name server: a consumer asks the name server for a producer and then begins direct communication with that process.

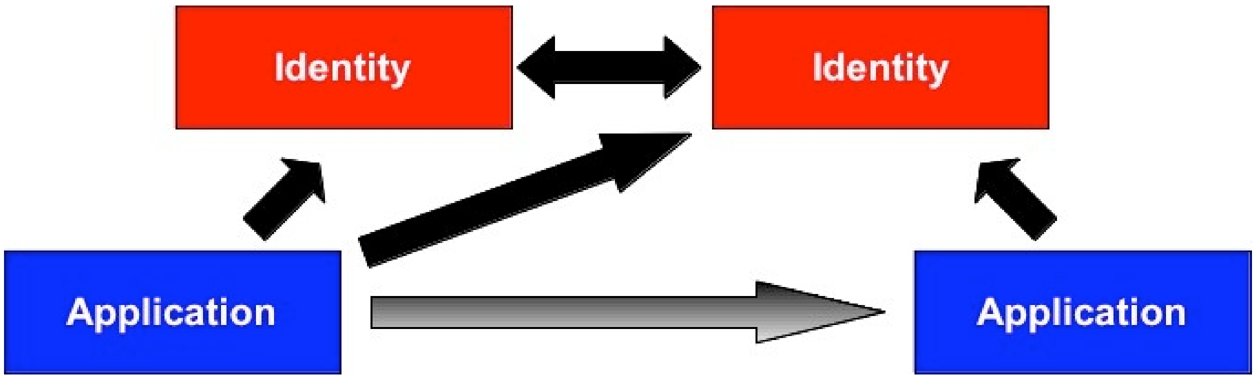
Applications **register services** they wish to provide with their creator identity (these services are identified by free-formed strings but typically follow the Java-style reverse domain name convention). Applications can **ask their creator identity** for a list of providers of the services they wish to consume, using these identifiers. Note that a service identifier is considered a known constant by both the provider and consumer.



An application communicates with another application of the same creator.

An **identity stores information regarding the applications** it has created. For each application, it stores the human-readable name the user gave it, any services it provides, and a hint describing how confidently the user trusts the application. Additionally, the identity **knows about other identities** the user has decided to trust. This list of trusted identities can be further split into groups with elevated or specialized trust.

Applications can also **request providers from other identities**, using its creator as a reference to communicate trust to this foreign identity. This allows for a rudimentary form of *transitive trust*: an identity trusts an application as much as it trusts the application's creator multiplied by the the degree to which the creator trusts the application. Note that this 1-level identity referral is the highest-order indirect trust provided by the protocol. There is never an additional mediating identity.



An application communicates with another application with a foreign creator.

Identity Protocol:

The identity protocol is designed as a very narrow XML-RPC interface. The reason for using this minimal interface is that it must not represent a complex or coherent object, for the purposes of application communication. All actions are expected to be simple query or update requests which return data or failure based on the internal state of the identity at the time of the request.

Application Protocol:

Applications must implement a very narrow XML-RPC interface related to some basic identity-application life cycle events.

Beyond this, applications must implement the respective interfaces expected by the services they advertise that they provide. The details of these interfaces are considered outside of the core protocol as they are free-formed. While the identity provides authentication tokens for the inter-application communication, it does not define how these tokens are to be passed, nor how they are to be interpreted once both sides have verified them.

Security:

The protocol is designed to operate across HTTP, either encrypted or unencrypted and **does not perform its own encryption**, beyond this.

Each application-identity or application-application communication channel is represented by a **public-private key pair** which is used to validate the message sender is who they say they are but **does not encrypt or sign the message**.

These public-private key pairs are sent during an initial hand-shake which is only ever done between an application and an identity. All inter-application communication uses keys initially provided by the identity when the communication was requested.

These key pairs **can be destroyed by either party** in a communication, at any time, so long as at least one message was accepted since the keys were created.

Simple Example: User Appearance

A simple example of an external application protocol being used to add functionality which would be limiting as part of the core identity is the user's appearance within the system.

Typically, a user is portrayed as a name and picture, potentially including other information regarding their interests or location, etc. Adding this information to the identity protocol would be problematic as it would require constant changes in order to adapt to changes in user interfaces or more complex environments. Simple examples being things such as picture resolution or format but more complex ones arising when one considers replacing the picture with a video or wanting to communicate richer meta-data such as employer, biographies, relationships, etc.

Due to the potential complexity of this issue, isolating it from the core identity removes an avenue of high volatility from that central component. It also allows for richer **domain-specific extensions** to be made in domain-isolated environments without needing changes to otherwise-reusable components.

The reference implementation **currently demonstrates this behaviour**: an identity is created with an application which defines its details. A server, given that it must render a user interface, asks the identity of a user for access to its details provider. It then contacts the provider application who, despite seeing that the server is untrusted, returns the public data detailing how the user should be described. The server uses this information to render their appearance but also uses it to populate a cache of searchable user details.

This does not assume that the user's identity or details application *lives* on that server. Correspondingly, the protocol doesn't treat this information as special which means that a different server could use the information differently. Further to that effect, a more sophisticated implementation of the details application could also implement future interfaces to view richer details while simultaneously implementing the legacy interface to provide seamless behaviour on legacy servers.

Conclusion:

This paper has provided a brief, high-level overview of the OpenAutonomy protocol and how it can be used to solve problems of both cross-server activities and issues of extensibility. This simple protocol is designed with cross-server interaction and future extensibility in mind, all without relying on a central authority controlling inter-application communication or user registration. This means that it can be deployed in virtually any environment and extended to solve virtually any problem, all without changing the central identity protocol.