ABFAB
Internet-Draft
Intended status: Informational
Expires: August 17, 2014

R. Smith
Cardiff University

February 13, 2014

# Application Bridging for Federated Access Beyond web (ABFAB) Usability and User Interface Considerations
## draft-ietf-abfab-usability-ui-considerations-00

## Abstract

The use of ABFAB-based technologies requires that the identities to be used to authenticate are configured on the client device. Achieving this requires software on that device (either built into the operating system or a standalone utility) that will interact with the user, and manage the user's identities and credential-to-service mappings. Anyone designing that software will face the same set of challenges. This document aims to document these challenges with the aim of producing well-thought out UIs with some degree of consistency.

## Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on August 17, 2014.

## Copyright Notice

Table of Contents

## 1. Introduction

The use of ABFAB-based technologies requires that the identities to be used to authenticate are configured on the client device. Achieving this requires software on that device (either built into the operating system or a standalone utility) that will interact with the user, and manage the user's identities and credential-to-service mappings. Anyone designing that software will face the same set of challenges. This document aims to document these challenges with the aim of producing well-thought out UIs with some degree of consistency.

## 2. Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

## 3. Terminology

Various items of terminology used in the document are heavily overloaded and thus could mean a variety of different things to different people. In an attempt to minimise this problem, this section gives a brief description of the main items of terminology used in order to aid with a consistent understanding of this document.

- NAI: Network Access Identifier - a standard way of identifying a user. See [RFC4282].

- Identity: In this context, an identity is a credential given to a user by a particular organisation with which they have an association. A user MAY have multiple identities - potentially multiple identities per organisation, or muliple identities from a set of organisations. The identity will consist of an NAI, alongside other information that supports authentication. Note that in other contexts the usual use of "identity" would match our use of "user", whereas the usual use of "identifier" matches our use of identity.

- Service: The thing that the user is attempting to authenticate to via ABFAB technology. See [TODO: Link to ABFAB-Use-Cases] for example use cases of what these services could be.

- Identity Selector: The mechanism by which the GSS-API acquires the identity to use with a particular service. An Identity Selector typically would allow the user to configure a set of identities along with service to identity mappings.





- Trust anchor: An authoritative source of verification of a particular service, used to allow authentication of a server using X.509 [TODO: link]. Typically a commercial CA to allow authentication via chain of trust, or a preconfigured non-commercial certificate.

## 4. Context

When using the ABFAB architecture to perform federated authentication to some service, when a user attempts to authenticate to an ABFAB secured application they will need to provide identity information that they wish to authenticate to that particular service with. This will happen through a process of the application calling the GSS-API, which will in turn gather the users credentials through whatever mechanism it has been configured to do so. We will call this mechanism the "identity selector" in this document (though note that this is not a recommendation on terminology for the mechanism).

The simplest way to achieve the desired effect would be a mechanism that simply takes the credentials from the currently logged in user (e.g. the Windows Domain Credentials) and uses those for all services that request authenticate through ABFAB. This approach gives

ultimate simplicity in terms of UI - i.e. it wouldn't have one - but the least flexibility. If there is ever to be a requirement for a user to use a different set of credentials for a service, then something more complex will be needed.

Where there is a requirement for multiple credentials to be supported, there are of course two methods that could be employed to configure identities and associated information:

- They could be configured manually by a user in a configuration file that could be edited by hand or some such simple mechanism, and read by the GSS-API mechanism. While this could work very well functionally, in practice only a small subset of users would be happy with - and able to - configure their identities in such a manner.

- They could be configured through some interactive mechanism. For ease of use this should have a simple UI, although a headless mode (i.e. a way of interacting with the identity selector without a GUI) may need to be supported.

When designing an identity selector with a UI (or indeed, with a headless mode), any implementer will share a common set of usability considerations inherent to the context. This document aims to explore these considerations, and provide advice and guidance on addressing them where possible.





## 5. Considerations around Terminology

Anyone designing an identity selector will have to grapple with choosing terminology that the average user has some chance of understanding. This terminology can split into a few main functional areas, as discussed next.

### 5.1. Identity

The first area where terminology is needed is around the identity/ identities of the user. Users are typically used to seeing a variety of terms for aspects of their identity in the federated sense, and an even larger variety in the wider internet sense. For example, in the federated sense some of these terms include "username", "login", "network account", "institutional account", "home organisation

account", "credentials", and a myriad of other such terms. However, NAI - the technically correct name for their identity in an ABFAB sense - is highly unlikely to be one of these terms that users are used to seeing.

Implementers of an identity selector will need to carefully consider their intended audience for both their level of technical capability and the existing terminology that they may have been exposed to.

Beyond terminology, careful thought needs to be given to the paradigm to use when presenting identity to users, as identities and services are abstract concepts that some users may not find is easily understandable. Implementers may wish to keep such abstract concepts, or may wish to examine attempts to map to real world paradigms, e.g. the idea of using "Identity Cards" that are held in the user's "Wallet", as used by Microsoft Cardspace.

## 5.2. Services

Terminology around services is likely to be less of a problem than identity, but it will actually depend on what the service is. For example, each service could be simply described as "server", "system", etc. But for simplicity just the word "service" will probably suffice.

## 5.3. Identity to Service Mapping

Depending on your perspective either each identity may be mapped to multiple services, or each service has multiple identities mapped to it. Thus any UI could present either perspective, or both.





# 6. Considerations around Management of Identities

One of the core features of an identity selector is the management of a user's identities. This section first looks at what information associated with an identity will need to managed, and then looks in detail at various usability considerations of this area.

## 6.1. Information associated with each Identity

There is firstly a minimal set of information that MUST be stored about each identity to allow ABFAB authentication to take place:

- NAI: The user's Network Access Identifier (see [RFC4282]) for this particular credentials. For example, "joe@example.com".

- Trust anchor: For the identity selector to be able to verify that the server it is going to talk to and attempt to authenticate against is the server that it is expecting, and that it is not being spoofed in some way. This is likely to be an X.509 certificate [TODO X509 ref], or a tuple of (trusted root certificate, servername in Subject or subjectAltName).

Next up is a small set of information that SHOULD be stored about each identity to allow the user to effectively select a particular identity:

- Credential: Whatever is used by the user to authenticate themselves with a particular NAI. What this will be will be dependent on the EAP method being used, but is likely to be something like a password or a certificate. Note that this is a SHOULD, rather than a MUST, because there are use cases where the user specifically chooses for this not to be "remembered".

- Friendly name for identity: To allow the user to differentiate between the set of identities represented in the Identity Selector. This should be editable by the user. The only restriction on this name is that it MUST be unique within that particular user's set of identities. For example: "My University", "Google Account", "Work Login", etc.

- Friendly icon for identity: To allow the user to differentiate between the set of identities they have they should be able to set an icon for that particular identity.

Finally, there is a set of optional information that MAY be stored about each identity that represent useful information for the user to have. Note that this list is not intended to be exhaustive; any implementer is free to add any more items to their identity selector





that make sense in their implementation.

- Password changing URL: The URL the user should visit should they need to change their password for this particular identity. For example, "http://www.example.com/passwordreset".

- Helpdesk URL: The URL the user should visit to get contact details for the helpdesk of the organisation that issued this particular identity for when the user encounters issues and needs help. For example, https://www.example.com/helpdesk.

## 6.2. Storage of Identity Information

Since some of the information that makes up the identity is sensitive in nature (e.g. containing passwords), then this information SHOULD be stored and accessed securely. This might involve ensuring the credential information is held in encrypted form on device and accessed using a passphrase. For deeper integration into the system, this could be done by using existing secure storage on the system such as Keychain on a Mac or the GNOME keyring on a GNOME based Linux device.

## 6.3. Adding/Association of an Identity

Users will have one or more identities given to them by organisations that they have a relationship with. One of the core tasks of an identity selector will be to learn about these identities in order to use them when it comes to authenticating to services on behalf of the user. Adding these identities could be done in one of three ways: manual addition, automated addition that is manually triggered, or automated addition that is automatically triggered. Each of these are discussed in more detail next.

Note that the term "association" or "addition" of an identity is used rather than "provisioning" of an identity, because while we actually are provisioning identities into the UI, provisioning is an overloaded term in the identity and access management space and could easily be confused with identity provisioning in the sense of the creation of the identity by the home organisation's identity management procedures.

### 6.3.1. Manual Addition

Allowing users to manually add an identity is technically the easiest method to get this information, but it is a method that has the greatest usability drawbacks. Most of the information required is relatively technical and finding some way of explaining what each field is to an non-technical audience is challenging (to say the

least). This especially is the case for trust anchor information. Thus this method should be considered as a power-user option only, or as a fall-back should the other methods not be applicable.

When this method is used, careful consideration should be given to the UI presented to the user. The UI will have to ask for all of the information detailed in Section 6.1.

There are two points at which a user could manually add an identity:

- Asynchronously: the user could be allowed to, at any time, trigger a workflow of manually adding an identity. This represents the most flexible way of adding an identity since a user can perform this at any time. It does, however, have inherent issues when it comes to verifying the newly added identity - see Section 6.5.

- Just In Time: when connecting to a service which has no mapping to an existing identity, the user could be given an option to add a new one, as well as associating with an existing one. This presents a better user experience when it comes to verifying the newly added identity (see Section 6.5), however, represents a less direct method of adding an identity. Users who have not yet added the appropriate identity to their identity selector may find it difficult to understand that they must try to access a particular service in order to add an identity.

Of course, implementers could support both styles of identity addition to gain the benefits of both and give flexibility to the user.

One item worthy of discussion here is the area of verification of trust anchors. An Identity Selector SHOULD try to ensure that manual addition of an identity and checking of the relevant trust anchors is done as securely as possible, whereby users have to enter and confirm all trust anchor information, or be required to explicitly agree to an insecure configuration if this is not done properly.

### 6.3.2. Manually Triggered Automated Addition

One way to bypass the need for manual addition of a user's identity - and all of the usability issues inherent with that approach - is to provide some sort of manually triggered, but automated, provisioning process.

One approach to accomplishing this, for example, could be for an organisation to have a section on their website where their users

could visit, enter the user part of their NAI, and be given piece of provisioning data that contains much or all of the relevant identity





information for importing into the identity selector.

It is reasonable to assume that any such provisioning service is likely to be organisation specific, so that the Issuing Organisation and realm part of the NAI will be constant, as would be the trust anchor information. The user part of their NAI will have been input on the web service. The password could be provided as a part of the provisioning file or the identity selector could prompt the user to enter it.

Additionally, the user SHOULD be given the opportunity to:

- Supply or change the default friendly name for that identity - to allow the user to customise the identifier they use for that identity;

- Indicate whether or not the identity selector should always ask before using services with this identity - to customise the way in which the identity selector interacts with the user with this particular identity;

- Reject the addition of the identity completely - to allow the user to back out of the association process in an intuitive way.

In this case, trust anchors could be directly provided through the provisioning mechanism to help establish the trust relationship in a secure manner.

### 6.3.3. Fully Automated Addition

Many organisations manage the machines of their users using enterprise management tools. Such organisations may wish to be able to automatically add a particular user's identity to the identity selector on their machine/network account so that the user has to do nothing.

This represents the best usability for the user - who wouldn't actually have to do anything. However, it can only work on machines centrally managed by the organisation.

Additionally, having an identity automatically provided, including its password, does have some particular usability issues. Users are used to having to provide their username and password to access services. When attempting to access services, authenticating to them completely transparently to the user could represent a source of confusion. User training within an organisation to explain that automated provisioning of their identity has been enabled is the only way to counter this.





### 6.4. Modifying Identity Information

This process is conceptually fairly similar to adding an identity, and thus shares many of the usability issues with that process. Some particular things are discussed here.

#### 6.4.1. Manual Modification

An identity selector may allow a user to manually modify some or all of the information associated with each identity. The obvious item that MUST be allowed to be changed by the user is the password associated with the identity.

#### 6.4.2. Automated Modification

To ease usability, organisations may wish to automatically provide updates to identity information. For example, if the user's password changes, it could automatically update the password for the identity in the user's identity selector.

### 6.5. Verifying an identity

An inherent by-product of the ABFAB architecture is that an identity cannot be verified during the addition process; it can only be verified while it is in use with a real service. This represents a definite usability issue no matter which method of identity addition is used (see Section 6.3):

- If the user has manually added the identity (see Section 6.3) they may have gone through the whole manual process with no errors and so believe the identity has been set up correctly. However, when they attempt to access a service, they may be given an error

message, thus causing some amount of confusion.

- If the user has had the identity provisioned into their identity selector, then there is a much greater chance of the identity information being correct. However, if any of the information is not correct, then there is the potential for confusion as the user did not add the information in the first place.

Also, if the identity information is incorrect the user may not know where the error lies, and the error messages provided by the mechanism may not be helpful enough to indicate the error and how to fix it (see Section 8).





### 6.6. Removing an Identity

This is fairly similar to adding or modifying an identity, and thus shares many of the usability issues with those processes. Some particular things are discussed here.

#### 6.6.1. Manual Removal

Allowing the user to manually delete an identity is probably the best way to achieve the goal. Any UI should allow for this option.

#### 6.6.2. Automated Removal

While automated removal of an identity is a way of achieving the goal without having to interact with the user, the consequence is that things may disappear from the user's identity selector without them realising.

### 6.7. Storing an Identity with or without credentials

Sometimes, a user may wish to have the identity they wish to use with a service stored by the identity selector, but not the credential (e.g. password) that goes along with that Identity. The consequence of this is that when a user attempts to authenticate to a service for which an identity, but no credential, is stored, then the user would

need to be prompted to manually enter the credential.

## 7. Considerations around Management of Service to Identity Mappings

A service to identity mapping tells the identity selector which identity should be used for a particular service. There is potentially a many-to-many association between identities and services since a user may wish to use one of their identities for many services, or more than one identity for a single service (e.g. if they have multiple roles on that service).

This potentially complex many-to-many association between is not easily comprehended by the user, and allowing the user to both manipulate it and control can be challenging. These obstacles are especially common when errors occur after an association has been made. In this scenario it is important that an identity can be disassociated with a service.

To further complicate the picture, users may wish for:

1. The identity to service mapping to be stored along with the credential, i.e. the user should always be authenticated to a particular service with a particular identity with no prompting.





2. The identity to service mapping to be stored but not the credential, i.e. the user should not be prompted to choose the identity for a particular service, but should be prompted to enter their credential for that identity.

3. The identity to service mapping to not be stored, i.e. the user should be asked which identity to use every time they authenticate to a particular service.

### 7.1. Associating a Service with an Identity

There needs to be a way for the user to create the service to identity association. It is advisable that this link be made only after the identity in question has authenticated with the service without any error.

There are a few ways this association could happen.

### 7.1.1. User-driven Manual Association

There are two ways in which manual association of an identity to a service could happen:

1. The user could manually associate a particular service with a particular identity using the identity selector before they first attempt to use the service. In order to do so, however, the user would need to know all the required technical details of that service beforehand, such as its GSS Acceptor Name.

2. On encountering a service new to the identity selector, the identity selector could pop up a dialogue box to the user asking if they would like to use an existing identity for this service (and might also allow them to create a new identity and use that).

### 7.1.2. Automated Rules-based Association

It would be beneficial from a usability perspective to minimise - or avoid entirely - situations where the user has to pick an identity for a particular service. This could be accomplished by having rules to describe services and their mapping to identities. Such a rule could match, for example, a particular identity for all IMAP servers, or a particular identity for all services in a given service realm. These rules could be configured as a part of the automated identity addition process described in Section 6.3.2 or Section 6.3.3





## 7.2. Disassociating a Service with an Identity

A user MUST be able to disassociate an identity with a service - that is, to be able to remove the mapping without having to remove the identity.

There should also be provision for the automated disassociation of an identity with a service for appropriate types of authentication failures.

## 7.3. Listing Services and Identities

A service listing should be considered in the identity selector which is both searchable and editable by the user.

## 7.4. Showing the Service that is requesting Authentication

When a user is attempting to authenticate to a service for the first time, there should be some indiciation given to the user as to which service is requesting authentication. In many cases, the service may be obvious (where the user has started the process of attempting to authenticate to a particular service), but in other cases this may not be obvious (e.g. if an authentication attempt is triggered by a timer or a specific event), and for this scenario some indiciation as to the requesting service is necessary.

## 7.5. Showing the Identity currently in use

It would be beneficial if, when using a service, the identity currently in use could be made visible to the user while he/she is using a specific service. This allows the user to identify which the identity is used with a particular service at a particular time (the user may have more than one identity that they could use with a particular service) - so that they can then disassociate the pairing.

## 7.6. Multiple Identities for a Particular Service

An Identity Selector should be able to deal with the case where a user has multiple identities associated with a single service. For example, upon receiving a request for authentication to a service that multiple identities are configured for, ask the user which of the identities should be used in this instance.

## 7.7. Not using ABFAB for a Particular Service

There may be cases where a user does not wish to use ABFAB based authentication at all to a particular service, even though it is ABFAB enabled. To support this, the identity selector would have to





allow the user to choose not to use ABFAB when they attempt to authenticate to a service. It would be desirable if the user could also flag that this should be remembered.

## 8. Handling of Errors

All GSS-API calls need to be instantiated from the application. For this reason when an error occurs the user needs to be sent back to the application to re-initiate the GSS-API call. This can get tedious and cause the user to opt out of what they are trying to accomplish. In addition to this the error messages themselves may not be useful enough for the user to decipher what has gone wrong.

It is important to try and avoid error cases all together while using GSS-API as error messages and error handling can really effect usability. Another solution would be to alter the application to handle the errors as it is instantiating the GSS-API communication.

TODO: Lots more to discuss here...

### 8.1. Identity Association/Verification Errors

TODO: e.g. wrong password, bad trust anchors, etc. TODO.

### 8.2. Service Errors

TODO: e.g. identity is correct but no authorisation. TODO.

### 8.3. Other Errors.

TODO: e.g. network errors. TODO.

## 9. Handling of Successes

It is of course hoped that the identity selector will have to occasionally handle successes as well as errors. This section has some brief discussion about some areas you might want to think about.

### 9.1. Reporting Authentication Success on First Use of Identity

The first time an identity is used with a service, it would be good practice to visually indicate in some way that the process has been successful, in order that the user understands what is happening and is then prepared for future authentication attempts.

### 9.2. Reporting Authentication Success

On an on-going basis you may or may not wish to indicate visually to the user a successful authentication to a service. This relates to Section 7.5.

## 10. Other Considerations

This section briefly discusses other considerations that you might want to think about that don't fit in any of the other categories.

### 10.1. Identity Selector Taking Focus

When an ABFAB authentication request is triggered, and where it needs input from the user, the Identity Selector should take focus in some way so that it is clear to the user that they need to do something to proceed.

### 10.2. Import/Export of Credentials

For various reasons, an identity selector implementation might want to include functionality that allows for the export/import of identities and service to identity mappings. This could be for backup purposes, to allow a degree of mobility between identity selector instances, etc.

If providing this functionality, it would be advisable that the credential store that is the result of the export should be secure - encrypted and password protected - given the nature of the information.

## 11. Contributors

The following individuals made important contributions to the text of this document: Sam Hartman (Painless Security LLC), and Maria Turk (Codethink Ltd).

## 12. Acknowledgements

Jim, Stefan, David.

## 13. Security Considerations

TODO: Bad trust anchors, no trust anchors, phishing.

## 14. Privacy Considerations

TODO: See Arch for general discussion. Particular to UI - storing credentials on a particular device, mobility considerations.

## 15. IANA Considerations

This document does not require actions by IANA.

## 16. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC4282] Aboba, B., Beadles, M., Arkko, J., and P. Eronen, "The Network Access Identifier", RFC 4282, December 2005.

## Appendix A. Change Log

Note to RFC Editor: if this document does not obsolete an existing RFC, please remove this appendix before publication as an RFC.

Draft -04 to ietf draft -00

1. Adding brief discussion of identities vs identifiers (Ken).

2. Changing assumption about credentials having a password in favour of more generic text for other auth types.

3. Adding discussion of storage of identity information.

4. Added sections on dealing with multiple identities per service, remembering credentials, remembering not to use ABFAB.

5. Added small section on ID selector needing to take focus in some way.

Draft -03 to draft -04

1. Addressing various comments from Jim and Stefan.

Draft -02 to draft -03

1. Bumping version to keep it alive.

Draft -01 to draft -02





1. Completed the major consideration sections, lots of rewording throughout.

Draft -00 to draft -01

1. None, republishing to refresh the document. Other than adding this comment...

## Appendix B. Open Issues

Note to RFC Editor: please remove this appendix before publication as an RFC.

## Author's Address

Dr. Rhys Smith
Cardiff University
39-41 Park Place
Cardiff CF10 3BB
United Kingdom

Phone: +44 29 2087 0126
EMail: smith@cardiff.ac.uk