DRAFT - IT Project Guidance

Design: Integrations – Overview of Common Approaches

Version:

0.2

## Purpose

To inform design decisions by summarising different integration approaches used between systems.

## Synopsis

While Integration APIs are often the most appropriate approach, systems may not offer them and/or environmental constraints force the use of alternate approaches.

## Contents

[Purpose 1](#_Toc160092161)

[Synopsis 1](#_Toc160092162)

[Contents 2](#_Toc160092163)

[Background 3](#_Toc160092164)

[Integration Approaches 4](#_Toc160092165)

[User Interface (UI) based Integrations 4](#_Toc160092166)

[Hypertext Links 4](#_Toc160092167)

[Advantages 4](#_Toc160092168)

[Considerations 4](#_Toc160092169)

[Disadvantages 4](#_Toc160092170)

[Application Programming Interface (API) based Integrations 5](#_Toc160092171)

[Webhooks 5](#_Toc160092172)

[Advantages 5](#_Toc160092173)

[Considerations 5](#_Toc160092174)

[Disadvantages 6](#_Toc160092175)

[Integration APIs 6](#_Toc160092176)

[Advantages 6](#_Toc160092177)

[Considerations 7](#_Toc160092178)

[Disadvantages 7](#_Toc160092179)

[Non-Integration APIs 7](#_Toc160092180)

[Advantages 7](#_Toc160092181)

[Disadvantages 8](#_Toc160092182)

[Non Interface Based Integrations 8](#_Toc160092183)

[Advantages 9](#_Toc160092184)

[Considerations 9](#_Toc160092185)

[Disadvantage 9](#_Toc160092186)

[Conclusion 9](#_Toc160092187)

[Appendices 10](#_Toc160092188)

[Appendix A - Document Information 10](#_Toc160092189)

[Versions 10](#_Toc160092190)

[Images 10](#_Toc160092191)

[Tables 10](#_Toc160092192)

[References 10](#_Toc160092193)

[Review Distribution 10](#_Toc160092194)

[Audience 10](#_Toc160092195)

[Structure 10](#_Toc160092196)

[Diagrams 10](#_Toc160092197)

[Terms 11](#_Toc160092198)

## Background

Integrations between services can be system user initiated or system agent initiated, using [user interfaces (UIs)](#Term_UI) or Application Programming Interfaces ([APIs](#Term_API)), transferring messages encoded in different encoding protocols (e.g., [SOAP](#Term_SOAP) or [JSON](#Term_JSON)), over different transfer protocols (e.g., HTTP/S), over transmission protocols (TCP/IP), over for invocation of procedures or operations resources.

Below are listed the advantages, considerations, and disadvantages of various approaches for consideration before making integration design decisions.

# Integration Approaches

## User Interface (UI) based Integrations

### Hypertext Links

The simplest integrations between services are user activated Hypertext Links from one view within the user interface of one web service to another view, within the interface of the same or another web service.

Common examples include integrating a discoverable and accessible page on a extranet or intranet corporate site to a project dedicated site.

Hypertext links are also used to connect from a [intranet|extranet] corporate site to a project’s dedicated static information site.

From there, links are used to connect to one of the following targets:

* A public view of the solution’s system
* A protected view of the solution’s system, which redirects it to its login form,
* An IdP service separate from the target service, passing information as a query param to indicate where to return to[[1]](#footnote-2).

#### Advantages

The simplest of integration approaches, requiring no encoding of timers, triggers, etc. to as the integration is initiated by a user.

#### Considerations

Quality Assurance of systems by testing user interfaces (e.g.: Tosca, Selenium, etc.) is notably brittle and difficult to maintain, and or achieve via automation pipelines.

#### Disadvantages

The Integrations are manually triggered by users so there is no certainty of the integrations happening regularly or at all.

The link information being encoded into the client system’s interface, the information transmitted (if any at all) never changes -- the only information changed between calls is the time of the event.

## Application Programming Interface (API) based Integrations

### Webhooks

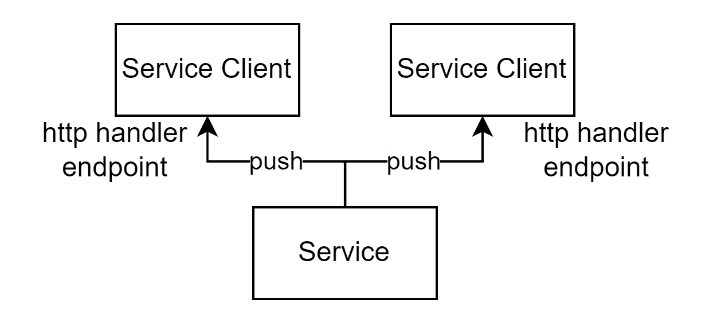


Figure 1: Web hooks based integration

With server offered [APIs](#Term_API) – REST for example -- a service client is required to repeatedly poll the server for latest changes by making a request to the server and analysing the response.

Polling increases traffic, reducing IO threads and bandwidth available to process user requests, and/or introduces delays when trying to reduce the bandwidth. If there are a thousand service clients, the above is multiplied by a thousand.

Webhooks remove the need from client-based polling by making it the server’s responsibility to notifying client(s) of changes, by making a request to the client’s registered webhook when data changes.

Webhooks are one-way publicly accessible integrations, avoiding the need for a user-session being established first.

Webhook calls are asynchronous and do return information to the server making the notification call to the client.

Note:  
The approach is classifiable as a [PUSH](#Term_PUSH) pattern, as opposed to a [PULL](#Term_PULL) pattern.

#### Advantages

Webhooks[[2]](#footnote-3) remove the need for cyclical polling, decreasing unnecessary traffic and IO consumption, as well as the delay before appropriate clients are notified of changes on the server.

#### Considerations

Webhooks for the most part should be POST HTTP verb based as they are providing information, not retrieving information, which would require a GET.

Webhooks may have response payloads, but it is less common than making the webhook appropriate for a non-acknowledged “fire and forget” call.

As with any connection between devices, especially across multiple networks, it should be secured using HTTP/S over TCP/IP domain-based TLS certificate.

The validity of payloads can be verified if a secret has been provided to the service so that it can develop a cryptographic hash signature of the payload.

For a web-based service to hook a webhook endpoint on a client system on a private network, a reverse proxy is used.

#### Disadvantages

Few systems are developed with [APIs](#Term_API), even less with webhook endpoints.

There is no industry standard or common pattern for the development of either webhooks or their payloads, so each can have their own quirks and associated development and testing costs.

Without acknowledgement messages, the server may believe that clients have been notified.

### Integration APIs

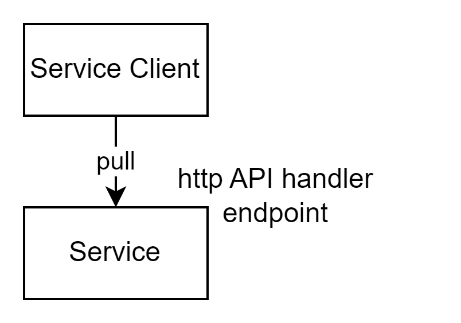


Figure 2: API integration approach

[Application Programming Interfaces (APIs)](#Term_API) are an interface provided by a system for another system to make requests for information or changes to be made[[3]](#footnote-4).

APIs protocols are generally developed in two general approaches: they are either designed to invoke Procedures or manage Resources.

[Remote Procedure Call (RPC)](#Term_RPC) based APIs permit the calling of an arbitrary number of commands of arbitrary signature, whereas Resource based [APIs](#Term_API) permit [Create, Read, Update, Delete (CRUD)](#Term_CRUD) operations on system Resources.

As a general observation, the complexity and flexibility of RPC based protocols (e.g.: [SOAP](#Term_SOAP)) have proven higher value between same environment components, while [Representation State Transfer (REST)](#Term_REST)’s adequate simplicity has proven to provide a higher value as the basis of APIs exposed to external 3rd party service consumers.

In the case of [REST](#Term_REST), remote clients are constrained to simple but effective use of one of four HTTP “verbs” (i.e., operations) on an identified resource. For example, if a remote client wishes to retrieve information, it submits a GET request to retrieve from the service a single or set of identified resources. If the remote system wishes to submit information or change or delete it, it changes the verb to POST, PUT or DELETE.

Addressing this document’s needs to cover 3rd party integration, only REST is discussed further.

Note:  
The approach follows a [PULL](#Term_PULL) as opposed to a [PUSH](#Term_PUSH) pattern, providing more specific control to the recipient as opposed to the provider.

#### Advantages

Unlike Webhooks, services that provide [REST](#Term_REST) based endpoints are not dependent on being able to access the remote client system.

Being HTTP/S based, requests can be passed through a [Web Application Firewall (WAF)](#Term_WAF) for additional defence in depth.

A [REST](#Term_REST) based API architecture is well known to current market resources, and a lot of tooling[[4]](#footnote-5), patterns and standards[[5]](#footnote-6) has been developed to ease development, testing.

Other than [OData](#Term_OData) there is no standard or accepted pattern for the development of qualifiers to verb based actions. For example, the approach to common operations of filtering, projecting, sorting, and paging are left to individual choices.

#### Considerations

The system’s integration [API](#Term_API) should be constrained to HTTP/S as clients will be in a different network, presumably crossing the public web network space.

Information transfer across public networks adds latency, as does the distance between services.

[OData](#Term_OData) fully adheres to [REST](#Term_REST) guidelines and is an OASIS based standard, developed by coordinated effort of multinationals such as Microsoft, SAP, Red Hat, IBM, Huawei, etc.

#### Disadvantages

The simplicity of [REST](#Term_REST) is the basis of requiring workarounds: operations on resources need to be thought of as resources in their own right: “operation requests” resources.

The use of standards is preferable to novel solutions to the issue of qualifiers – but while there is the [OData](#Term_OData)/ISO/IEC 20802-1:2016 standard, it has lost some mindshare to Facebook’s non-standards based open-source protocol, GraphQL. Service designers who wish to attract 3rd party buy-in are forced to consider offering [APIs](#Term_API) in both flavours.

### Non-Integration APIs

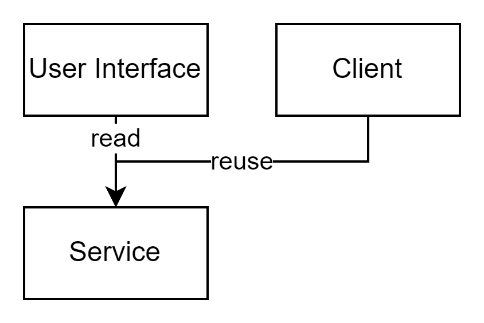


Figure 3: UI API Reuse Approach

While a system may provide APIs, they may not be intended for integration purposes. For example, APIs may have been developed to support [Asynchronous JavaScript and XML (AJAX)](#Term_AJAX) requests for data only without requiring a refresh of a system’s webpage(s) -- a common requirement of [Single Page Apps (SPAs)](#Term_SPA).

[API](#Term_API)s for the purpose of backing [User Interfaces (UIs)](#Term_UI) differ those required by Integration in that they generally are constrained to a single tenancy or context associated to the current session.

#### Advantages

Separating user interfaces from system logic improves maintainability, while decreasing server resource consumption as well as IO. The user interface is the most visible hence “perishable” tier or component of a service that depreciates in respect to user expectations as technologies improve. By being a separate code base, it provides an efficient way of being refreshed without requiring change to the system’s logic.

#### Disadvantages

As there is no method provided to develop request resources irrespective of tenancy or content, a service client must work around this by repeatedly signing in and out – a resource intensive and time-consuming operation – to cycle through contexts, all to make a single request within each of them.

The above relies on a single user account being made a member of all contexts.

To move through a list of contexts, the service client is required to be provisioned beforehand with a list of identifiers of these contexts and a single account has been a member of all of them. Alternatively, the service client must be provisioned with both a full list of context identifiers and credentials for multiple accounts, one for each context. Both approaches raise security concerns, although to different degrees.

The list of context identifiers and/or users must be kept current by some refreshment process to not miss changes.

The full list of context identifiers and/or user accounts must be iterated through, regardless of whether change has occurred on the server (most queries will return no new change or information).   
The approach is brittle as well as resource and time intensive.

## Non Interface Based Integrations

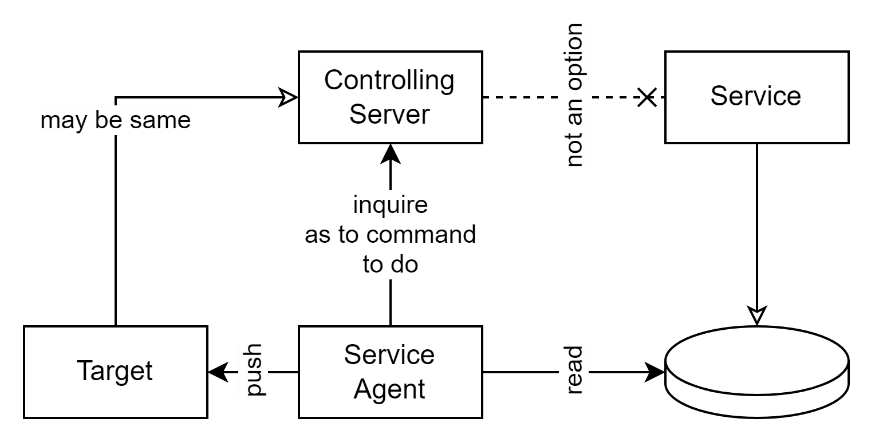


Figure 4: ETL Service Agent Approach

When the service requiring querying does not provide [API](#Term_API)s designed for integration, and an abuse of [API](#Term_API)s designed for [UI](#Term_UI) is too difficult to develop and brittle to operate effectively, then other options must be explored.

One option is – given sufficient access - to bypass the available interfaces, and query internal components directly.

For example, under all systems are datastores, relational in most cases, that can be accessed if a service agent can be installed on a device in the same network as the target system’s database.

A service agent can be a simple task runner following instructions provided to it when it was deployed, or a remotely controlled task runner running on instructions that are provided in real time from a controlling server.

Note:  
A remotely controlled service agent effectively changes the process from a [PUSH](#Term_PUSH) to a [PULL](#Term_PULL) approach, similar to what an integration API approach provides.

In both cases, communication between server and target destination, and/or communication between service agent and controlling server, if any, are required to be over secure channels.

Additionally, the service agent must identify itself using credentials. These credentials are required to be initially provided to the client in a secure manner, then persisted by it in a secure manner (e.g.: using an installed or default system encrypted keystore)[[6]](#footnote-7).

#### Advantages

The solution can work when appropriate integration [API](#Term_API)s are not available.

#### Considerations

#### Disadvantage

The installation of a runner and/or device is required first.

There is no convention or agreed pattern to follow, so each task requires its own design, development, testing. For example, a task may involve a series of steps from making one or more queries to the target database, dumping the results to a delineated format (e.g.: [CSV](#Term_CSV)), packaging them up, and transmitting them to a defined target location.

In some cases, the agent can be remotely controlled from a central service, which tells it when to run, and what to retrieve.

## Conclusion

While it is preferable to use Integration [API](#Term_API)s to communicate between systems, it is important to recognise that the option is not always available. Practicality requires the use of alternate solutions. Between using interfaces backing [API](#Term_API)s and direct database access to perform [Extract, Transform, Load (ETL)](#Term_ETL) is the next best option. In such cases, if there is a choice between performing the [ETL](#Term_ETL) operations via a scheduled task or a remotely controlled service agent, the later is again the preferred option.

Appendices

Appendix A - Document Information

### Authors & Contributors

Author: Sky Sigal, Solution Architect

### Versions

* 1. Initial draft.
  2. Glossary diagrams added.

### Images

[Figure 1: Web hooks based integration 5](#_Toc160095086)

[Figure 2: API integration approach 6](#_Toc160095087)

[Figure 3: UI API Reuse Approach 8](#_Toc160095088)

[Figure 4: ETL Service Agent Approach 9](#_Toc160095089)

### 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 |
| Sandy Britain, Enterprise Architect |  |
| Russell Campbell, Project Manager |  |
| Amy Orr, Data Domain Architect |  |

### 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.

### Terms

Refer to the project’s Glossary.

**Asynchronous JavaScript and XML (AJAX)**

: a front-end development technology. A key technology for developing [SPA](#Term_SPA) based service clients.

**Application Programming Interface (API)**

: a system’s interface for remote access by another person. Preferably it authenticates authorises, audits, validates access.

**Create Retrieve Update Delete (CRUD)**

: an acronym for the primary operations on data stores.

**Character Separated Values (CSV)**

: a text-based information exchange format.

**Extract, Transform, Load (ETL)**

: an acronym for a common process of transferring information between systems, where one or both parties do not have an integration API.

**Graphical User Interface (GUI)**

: a form of a system’s [User Interface (UI)](#Term_UI). Traditionally [WIMP](#Term_WIMP).

**GraphQL**

: a often used but non-standards based open source approach to qualifying some [REST](#Term_REST) operations. See [OData](#Term_OData).

**IT**

: acronym for Information, using Technology to automate and facilitate its management. IT is a subset of ICT.

**ICT**

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

**JavaScript Object Notation (JSON)**

: a serialisation format for describing objects used for communicating between systems.

**Long polling[[7]](#footnote-8)**

: Long polling is used in real-time web applications to achieve near-instantaneous communication between clients and the web services. It is particularly useful in chat and messaging applications where real-time updates are crucial. When used by service agents behind firewalls it has the advantage that it is permitted to call out to the external command server, whereas the command server would be blocked calling into an agent.

**Minimum Viable Product (MVP)**

: A deliverable that meets the minimum set of functional and non-functional requirements expected by stakeholders.

**Missing Valuable Planning (MVP)**

: the state of Agile projects to which no project planning has been applied.

**ODATA**

: an international standards-based method of qualifying [REST](#Term_REST) operations (e.g., filtering, sorting, projecting, paging). Contrast with [GraphQL](#Term_GraphML).

**OpenAPI**

: a standard for describing [REST](#Term_REST) based interfaces in a machine-readable format.

**PULL**

: an integration approach where the service client requests what it wishes for from a service.

**PUSH**

: an integration approach where the service pushed to service clients without confirmation that the server desires it.

**Remote Procedure Call (RPC)**

: an API architecture pattern for [API](#Term_API)s that permits the invocation of Procedures/Functions. Contrast with [REST](#Term_REST), an [API](#Term_API) architectural pattern that provides methods to manipulate Resources instead.

**REsource State Transfer (REST)**

: a messaging protocol used to communicate between systems.

**Simple Object Access Protocol (SOAP)**

: a messaging protocol used to communicate between systems. See [REST](#Term_REST).

**Singe Page App (SPA)**

: a modern approach to web solutions where the whole site, minus data is transmitted to the browser, calling back for data as needed to be displayed. SPAs are noted for improving responsiveness while decreasing server resource utilisation as well as IO bandwidth, therefore cost.

**Textual User Interface (TUI)**

: a console form of [UI](#Term_UI).

**User Interface (UI)**

: an interface for use by system users (as opposed to a remote system). May be a [TUI](#Term_TUI) or [GUI](#Term_GUI).

**Web Application Firewall (WAF)**

: a security firewall specialised with analysing web-based HTTP/S traffic.

**Windows, Icons, Mouse, Pointer (WIMP)**

: an acronym for a type of [GUI](#Term_GUI).

1. To avoid design brittleness and maintenance requirements it is recommended that links avoid passing information. [↑](#footnote-ref-2)
2. Sometimes colloquially referred to as ‘reverse APIs’. [↑](#footnote-ref-3)
3. As opposed to a User Interface (text or graphics based) used to [↑](#footnote-ref-4)
4. E.g., Postman [↑](#footnote-ref-5)
5. E.g., OData, OpenAPI [↑](#footnote-ref-6)
6. Use of encrypted cleartext is not appropriate for sensitive data. [↑](#footnote-ref-7)
7. [What is Long Polling? | PubNub](https://www.pubnub.com/guides/long-polling/) [↑](#footnote-ref-8)