Design and Development Log

This document contains the full version of the Design and Development Log, providing supporting material for the Research Through Design process we followed in our work.

2017-08-20
AwAg Platform

Define Awareness Agent Eclipse project
Taking a previous simple test application, we created a new Eclipse Java project to as the basis for an extensible platform that allows for modularity and multiple implementation types, with Platform, Service and Application sub-projects.

2018-09-26
AwAg Platform

Add Cognos reporting
Initial approach to extracting information from project, using a database-based reporting tool.

2019-07-19
AwAg Platform

Switch from JBot to jSlack
Our original Slack test code used a relatively limited library called JBot[a][b], designed to get you up and running quickly with Slack bots. This was fine for initially proving some concepts, but the full range of our design depended on being able to fully access the Slack API, so the project was transitioned to the full-featured jSlack API[c][d][e].

[a] https://github.com/rampatra/jbot
[b] https://blog.rampatra.com/how-to-make-a-slack-bot-in-java [https://perma.cc/ME5K-7AXW]
[c] https://tools.slack.dev/java-slack-sdk/guides/web-api-basics/
[d] https://perma.cc/KL7M-D8HX
[e] jSlack later transitioned to https://github.com/slackapi/java-slack-sdk, having been taken on with its original developer https://github.com/seratch by Slack itself

2019-07-25
AwAg Platform

Add Cloudant-based module configuration
To address the significant configuration needs of the design scope, and to facilitate multiple agents running in cloud-based environments, we transitioned the application to use a commercial CouchDB object store for configuration data.

2019-07-26
AwAg Acquire

Add Slack RTM support
To better support real-time monitoring of content in Slack, we added the now-legacy RTM API[a] to the platform, enabling us to listen for content more effectively.

[a] https://api.slack.com/legacy/rtm [https://perma.cc/QP7L-5FZB]

2019-09-24 *AwAg Platform*
First implementation of a Content Item
Initial testing used Slack content more or less natively, which was effective for the scope of the first prototype, but it became clear from our work that data abstraction and standardisation was needed to build a system that would be flexible and extensible enough to accommodate many diverse data sources. This point marked the start of developing the Content Item concept, in the first instance by applying a simple layer of abstraction over native Slack items.

2019-09-24 *AwAg Platform*
Add concepts of Base Operation and Service Operations
This was the start of a development branch that did not end up going anywhere. We initially designed the agent platform with the concept of several operation types - instructions for the agent to do something. While the support for this is still present, testing with real sources showed that a different type of control was needed - with what eventually became the administration side of the Interact service.

2019-10-15 *AwAg Augment*
Add the concept of Augmentation Items
We had tried several approaches so far to enhancing content to make it more easily actionable, with some issues arising: while it was certainly possible to summarise items or apply a classification, there was no organised framework within which this could sit, and the approach was not extensible. The previous successful introduction of the Content Item concept gave us an opportunity to address this issue in a structured way, by extending the CI structure to include Augmentations so that we could be both flexible about what enhancements we could add and also have a mechanism for abstracting this and managing/addressing the information.

2019-10-16 *AwAg Platform*
Backtrack on Operation changes introduced on September 24
These had proved impractical and not entirely useful in testing.

2019-10-16 *AwAg Platform*
Introduce a Consignment and Queue based approach
Moving away from the previous content/service operations approach, we added a wrapper for Content Items named Content Consignments (essentially just a collection of CIs) and added a queue system for these. This would then go on to form the design paradigm for intra-service content transfer in the Agent via the Exchange Service.

2019-10-17 *AwAg Augment*
Add simple text based explanation mechanism for Augmentations
First introduction of an explainability mechanism within the augmentation framework.

2019-10-19 *AwAg Platform*	**Reorganise modules into Acquire, Augment, Control etc.** While this development change was little more than a code refactoring at this stage, this was a significant design development. It marked the introduction of the concept of high level components within the Awareness Agent system model, with the creation of the Acquire and Augment module types to handle those logical operations. This was a result of initial work to expand the scope of these aspects of functionality - early testing showed us that a consistent modular approach was needed to allow for the future expansion of scope that we were looking for.
2019-10-19 *AwAg Platform*	**Introduce Triage service** Looking at the content flow through the prototype Awareness Agent, we realised that there was a need for a first point of contact assessment of incoming content - a triage process to borrow from medical terminology. We added the Triage service to act as the first port of call for all content coming from Acquire services, adding an ability to discard unwanted content based on simple heuristics before going on to Augmentation.
2019-10-21 *AwAg CI*	**Expand and improve metadata handling** Recognising a need to have more metadata information about content available further down the line, we improved the way that metadata is stored in individual CIs, and also added increased metadata at the consignment level.
2019-10-21 *AwAg Platform*	**Add Discard service** Before this point we had simply been dropping unwanted content by not passing it on to the next stage; however we recognised that there was utility in tracking discarded content - both from a scientific and application development perspective. By adding a service to which all unwanted items could be sent, we acquired to capability to easily record information about discarded items and also take any other actions appropriate at that point. This approach also had greater consistency with the overall CI queue-service flow model, with the Discard service fitting into a structured lifecycle for every CI.
2019-10-26 *AwAg CI*	**Add support for new Slack-specific metadata fields** We started formalising the concept of Extended Fields in the Content Item after testing showed us that we needed to be able to surface some source-specific information later in the CI lifecycle (for example at the point of augmentation or interaction). We recognised that this would be a generic need - sources other than Slack would need their own specific metadata - and we developed extended fields to fit this information within a standard CI structure in a manner consistent with the standard fields approach that we had already adopted.

2019-11-01 *AwAg CI*

Transition CI data to using JSON-LD

As a consequence of the development of extended fields, we looked again at the basic data structure of the Content Item, which had hitherto been entirely bespoke JSON. We recognised that JSON-LD[a] and the related concept of linked data notifications (LDN)[bc] provided us with an academically rigorous and supported mechanism for working with and sharing data from multiple sources, so we transitioned the internal CI structure to one compatible with JSON-LD.

[a] `https://www.w3.org/TR/json-ld/`
[b] `https://www.w3.org/TR/ldn/`
[c] `https://csarven.ca/linked-data-notifications` [`https://perma.cc/7SVP-JPBV`]

2019-11-09 *AwAg CI*

Distinguish standard & extended fields with prefix

As part of the transition to JSON-LD we looked again at how properties (fields) in the Content Item were named. We had three categories of property: native (existing in the source data item), standard , and extended. We decided to take an approach of using Compact IRIs (CURIEs)[a], with the namespace `awag` mapped to Awareness Agent schema `http://parse.net/awag/0.1/` in the `@Context`. We gave standard fields (where they existed as independent properties) a shorthand namespace of `awag:std`, and extended fields a namespace of `awag:xtd`. We made a decision to use unchanged names for native properties from the source system rather than renaming or adding an additional namespace. There had been a number of ways that we might approach this, and we retrospectively discuss this further the Reflections section.

[a] `https://www.w3.org/TR/2010/NOTE-curie-20101216/` [`https://perma.cc/443S-8ELP`]

2020-09-28 *AwAg Acquire*

Add LDN module type

We added the basic framework for supporting LDN as an Acquire source via a client for an LDN inbox. We implemented our own local Solid server (nodeSolidServer[a] running on FreeBSD via PM2[bc]). We ultimately never progressed beyond the proof of concept stage for LDN Acquire support via Solid – although the technology was potentially useful, but did not have the level of actual content that we wanted to run a study.

[a] `https://github.com/nodeSolidServer`
[b] `https://www.npmjs.com/package/pm2` [`https://perma.cc/8ER6-KGAJ`]
[c] `https://gist.github.com/PieterScheffers/457769f2090c6b69cd9d`

2020-10-06 *AwAg Platform*

Use Juneau to serialise to/from Json

We were storing JSON objects in a commerical CouchDB application, using the supplied libraries to store objects (and implicitly serialise them) - we found that this was actually generating incorrect JSON for some of our objects after a round trip, so we moved to Apache Juneau[a] for this serialisation also, giving us more control over the process (we had already used Juneau extensively for JSON serialisation and REST service provision).

[a]https://juneau.apache.org/ [https://perma.cc/6ELF-TA3B]

2020-10-08 *AwAg CI*

Add tracking of Content Items

This was a necessary step in developing the mechanism for CI exchanging between agents - we added a persistent tracking mechanism to record CIs that the agent had already originated or seen, so that if offered back to us in a future exchange we could decline them (or take appropriate actions such as looking for new augmentations added by a partner agent)

2020-10-08 *AwAg Augment*

Add Augment engine and structure

Significant internal change to the organisation of the Augment process, adding a new engine to support it.

2020-11-08 *AwAg Exchange*

Add basic outbound exchange functionality via MQTT

The initial implementation of the Exchange component uses MQTT; we implemented code to manage the round trip serialisation of Content Items to/from binary data to be sent and received from MQTT.

2020-11-11 *AwAg Exchange*

Change MQTT serialisation to Kryo

We found in testing that serialising our complex Content Item structure to binary – and being able to successfully instantiate the returned deserialised item – is actually quite difficult. We tried here with the Kryo library[a] instead of the original approach of using native Java bytestream functionality.

[a]https://github.com/EsotericSoftware/kryo

2021-04-20 *AwAg Acquire & Interact*

Change to use Slack Conversations API

Slack made a change from using "Channels" terminology to referring to "Conversations" instead, so we reflected that change.[a][b]

[a]https://api.slack.com/docs/conversations-api [https://perma.cc/W7QM-5FK7]
[b]https://api.slack.com/methods/conversations.list [https://perma.cc/6Z72-XCZN]

2021-04-21
AwAg Exchange

Rework MQTT serialisation again
We had continued to experience problems with the Content Item lifecycle even with the Kryo library, so changed approach once more. This (successful) iteration uses Apache Juneau for object-JSON serialisation, and Apache SerializationUtils[a] for string-binary conversion. The Juneau approach had been a well tested feature of other elements of the agent so far, so we were able to get consistent object-string conversion using it, and the Apache serialisation tools worked reliably for the string-binary conversion.

[a] `https://commons.apache.org/proper/commons-lang/apidocs/org/apache/commons/lang3/SerializationUtils.html` [`https://perma.cc/UUR5-57VM`]

2021-05-14
AwAg CI & Interact

Add 'Prominence'
Adding the concept of 'prominence' a categorical code that can be assigned to a Content Item having possible values `LOW`, `NORMAL`, `HIGH`, `HIGHER`. We developed this as a way of surfacing augmentations: the prominence is a property at the Content Item level, and is effectively another Standard Field, but the mechanism for setting it is left open; the assumption is that the Augment engine or indeed later services can adjust prominence as they process the CI based on augmentations or other factors. Prominence can then be used by Allocate or Interact services to change how the CI is processed or displayed. The design rationale for this was that it is an easy to check standard way of indicating how prominent a CI should be. We introduced this in response to difficulties we experienced using augmentations for this task – they have the information, but not accessible in a standard way. Having Prominence available allows components to make high level decisions about a CI without knowing the internal augmentation structure.

2021-05-15
AwAg CI

Add permalink feature to Slack content item
Testing with user interface prototypes showed that the agent would benefit from being able to easily direct the user to the original item in context in the source system via a URL. In the case of Slack content, there was not such URL available directly, but we could construct one from the message content and current context within the Acquire service. We added this as a new extended Slack field (`PERMALINK`), but also mapped this as a Standard Field (`URL`) so that other types of CI can be used in the same way.

2021-06-08 *AwAg Interact*

Add Slack Service

This marked the start of adding two-way interactivity to the Slack Interact component, exposing functionality to receive events from Slack in response to user actions[a]. The first iteration of this was to add and test basic functionality; we would then go on to expand this and integrate with the Awareness Agent framework based on testing results.

[a]Initially using the now-legacy RTM API – `https://api.slack.com/legacy/rtm`

2021-06-20 *AwAg Interact*

Improve Slack interactivity

Initial testing of the Slack service was successful, so we expanded the types of interaction that were possible and genericised the process by adding a `SimpleSlackAction` object to the code. We used this to add a set of internal queues for Slack actions that were based on abstracted Slack request objects, with a separate request handler component to process these. We did this to take an approach consistent with the design ethos of the rest of the agent, which we had found in testing to be adaptive to change and different types of request.

2022-01-04 *ML Service*

Initial creation of awagml

We added a new back-end service, referred to as "awagml", to initially support generating classifications for items. The first instance was a simple classifier to test this functionality - intially testing with both `MultinomialNB` and `SGDClassifier` (SVM) classifiers using scikit-learn [Pedregosa et al., 2011]. While the Awareness Agent itself is a Java application, we decided to implement a separate ML service in Python for the following reasons: 1) availability and quality of ML and associated packages in Python, 2) support for rapid prototyping, and 3) consistency with design principle of keeping back-end ML commoditised and separate.

2022-01-11 *ML Service*

Add `add_dataset_row`

Initial work to allow us to add training items to the ML model and refit on the fly via an API call.

2022-04-27 *AwAg Augment*

Implement Simple Classification Augmentation

We added a single value classification for CIs using the new `awagml` back-end service. This passes CI content to the service and gets a classification back, which is added to the CI as an Augmentation. This first instance had no concept of multiple ML models, with `awagml` supporting only a single model for classification.

2022-05-10 *ML Service*

Initial support for multiple data dictionaries

The first step to moving to a multiple model design was to add support for multiple data dictionaries at the back end, identifiable by ID. This allowed us to manage different datasets for different models.

2022-05-11 *ML Service*

Service initialisation improvements

We implemented a hierarchical data structure for model data persisted to disk, with automatic creation and fitting of data sets on startup.

2022-05-16 *ML Service*

Add multi-model support

Completed initial implementation of multi-model support at the back end.

2022-05-16 *AwAg UDML*

Add multi-model support

Adding support for passing a model ID to the `awagml` back end was the next step following the addition of support for multiple models to `awagml`. This would allow us to move from proving the concept of classification augmentations to supporting full multi-model usage.

2022-05-26 *AwAg UDML*

Add support for multi-model augmentations

Having added model ID support in the previous iteration, we now changed the processing so that models could be specified in the augment instance configuration rather than a relatively fixed ID. This allowed us to drive the process dynamically using the augmentation configuration.

2022-05-29 *AwAg UDML & Interact*

Add multi-channel output using classification to channel name map

We introduced a main element of User-Driven ML by implementing full multi-channel output in the Interact layer for augmented Content Items, where the model ID (classification name) and classification value are mapped to a Slack output channel. We made several user interaction design changes to enable this, changing from a single channel interaction paradigm to using multiple channels, with channels mapped to classification names and values. This required us to add support in the Interact configuration for the necessary mapping table in parallel to Augment configuration to apply matching augmentations.

2022-05-31 *ML Service*

Add support for multi-model classification training
Then process of training `awagml` had previously been a manual server-side command-line operation, where training text content was organised in directories for fitting and model use. This had been sufficient to prove the concept, but would not support the User-Directed ML process that we had in mind. We added an API to allow training requests to be made to `awagml` via REST calls, so that the Awareness Agent could directly initiate training events using its own 'live' data.

2022-05-31 *AwAg UDML*

Add support for multi-model classification training
Having successfully tested the process of applying multi-model classifications, we then looked to bring the whole model lifecycle within the scope of the Awareness Agent. We expanded our Slack interact handler to handle requests generated by the user interacting with published items, and added the necessary UI artefacts into those items themselves. With this we were able to test the full process of an item being classified, sent to a specific channel, then re-classififed into a different channel by the user, with a concomitant back end request to re-train the item in `awagml` using the newly added training service.

2022-06-01 *AwAg Augment*

Change so that Model ID can be independent of Augmentation Name
Our original implementation depended on a match between the augmentation name and the model ID to use for a given augmentation. We found that this was limiting flexibility: while there was generally a relationship between the two this was not a given, and we might also in future want to be able to assign the same back-end model to different augmentations. We changed the design to instead use a configurable mapping between the two.

2022-06-02 *ML Service*

Add model lifecycle methods
To support the full model lifecycle under dynamic control of the Awareness Agent we added back end services to create/delete models and to add or remove classifications from models.

2022-06-10 *AwAg Acquire*

Remove RTM and replace with Events API

The Slack RTM API[a] that we had hitherto been using to receive callbacks from Slack had been deprecated in favour of the Events API[b]. While this was a forced change it was also a good opportunity for design re-evaluation: the RTM API involved opening a websocket to listen for events, whereas the Events API is subscription based, where a client subscribes to events and the receives these to its configured URI as a HTTP REST call. This change allowed us to coalesce on two adaptable mechanisms for the Acquire service to use: a listener (acts as a service and receives HTTP calls, as is the case with the new Slack Events API) and a client (makes outgoing calls to a service on a scheduled or event-based basis). This meant that we were eventually able to design these mechanisms into a framework that could be applied to the majority of potential Acquire sources. The payload of REST calls to a listener type Acquire service would differ from source to source, but the service mechanism would be the same.

[a] `https://api.slack.com/legacy/rtm` [`https://perma.cc/QP7L-5FZB`]
[b] `https://api.slack.com/events` [`https://perma.cc/MTM2-G27X`]

2022-06-14 *AwAg UDML*

Add ML model management REST calls

Previously the `awagml` service relied on manual processes at the back end for model management, but we had enhanced this to provide a REST interface for these tasks (model creation, deletion, changes to classifications). We now adapted the agent itself to match this, bringing full control over the model lifecycle within its purview. This was an important step for user-empowerment, with the user now able to control models via the interface. It also had benefits for the agent deployment and integration processes, a parallel consideration.

2022-06-14 *ML Service*

Fix issues arising from testing

We addressed a number of minor issues that had become apparent during testing, such as fit errors under certain conditions.

2023-02-21
AwAg
Augment

Add item summarisation based on OpenAI/ChatGPT
This was the first instance of being able to exploit the revolution in Large Language Models such as ChatGPT. The agent up to this point had used a very simple technique for producing a shortened summarisation of content items for display: it would truncate long items, as well as removing any HTML content. We were able to enhance the augmentation processing, swapping out the original Summarise augmenter with a new one that passed the CI data to the OpenAI API and asked for a summary, which it then included as the summary augmentation (in the case of a failure, we used the original technique as a backstop). This greatly improved the utility of the presented CIs but importantly it also vindicated the design approach we had taken: we demonstrated that the summary augmentation could be seamlessly swapped out, taking advantage of a powerful commodity ML service that had not been available to us at project inception.

2023-02-28
AwAg
Acquire &
Interact

Separate Slack Acquire and Interact functions
The support for using Slack had grown somewhat organically, resulting in an implementation that did not have a clear demarcation between using Slack as an Acquire and as an Interact source. As a consequence of the Acquire formalisation work that we had been able to do as a result of switching to the Events API, we were able to complete a separation of these logical functions.

2023-03-03 *AwAg Platform*

Support for Slack workspace installs

Prior to this, the agent had still required manual installation into a workspace[a][b], with the administrator manually creating and assigning tokens within Slack and then adding these to the agent configuration. This was obviously not a sustainable approach, and it limited both agent and user autonomy. We added support for the Slack OAuth 2.0 authentication and installation flow[c], exposing additional services for Slack to make related calls to the agent, and the internal processes to obtain and store the API tokens issued in this process. Our work to logically separate the Acquire and Interact components of the Slack implementation meant that we were also then able to install the Acquire Listener and the Interact service independently – to different workspaces as required. This was an important feature, because it meant that a Listener could be installed to multiple Slack workspaces with limited read-only authorisation scopes, while only the Interact service needed greater permissions. The design concept for this was that the listener should be installed to workspaces that the agent's owner was a member of (such as company workspaces) without the need for elevated privileges purely to acquire content, while the Interact instance would be installed to a workspace that the owner controlled, dedicated just to the Awareness Agent interaction process. This addressed various potential security and deployability concerns that had we had identified during the course of prototyping.

[a] `https://slack.com/help/articles/217626298-getting-started-for-workspace-creators`
[b] `https://perma.cc/A7AH-R8PW`
[c] `https://api.slack.com/authentication/oauth-v2` [`https://perma.cc/ER4A-H5PS`]

2023-03-10 *AwAg Platform*

Add an applicationIdentifier/contextIdentifier

While working on the process of setting up a study to test the Awareness Agent we realised that there were a number of technical points where there was a conflict if multiple agents shared the same Java JVM (i.e. deployed to the same application server). In particular the use of singletons for internal queues and other items. We implemented a named application identifier to be used internally to address this.

2023-03-16 *ML Service*

Add support for multiple agents

To better support the study process, we added support for multiple agents to the back end. The initial test version had not used any form of client identification, which was obviously not suitable for scaling up.

2023-03-27
AwAg
Acquire

Initial work to add RSS as an Acquire source

We added RSS as a second type of Acquire source, with a Client type acquire service configured to pull one or more RSS feeds on a scheduled basis[a]. This was a validation of the approach that we had designed for Acquire Services and Content Items generally, as it showed that we were able to relatively seamlessly add in a different type of source that not only had different content but also a different mechanism for acquiring content. Iterative testing of this required a few minor changes to be made – mainly in relation to the particular content properties of RSS items – but importantly we found these to be non-breaking changes for the existing Slack Acquire implementation, and we were able to seamlessly use the Slack Interact implementation to output RSS-sourced items.

[a]Using Quartz Scheduler in `cron`-type mode: `https://www.quartz-scheduler.net/documentation/quartz-3.x/tutorial/crontriggers.html` [`https://perma.cc/EP6D-R2RZ`]

2023-04-13
AwAg
Platform

Refactor `platform.database` to `platform.data`

This was a minor refactoring operation, but reflected a significant design development. Previously `platform.database` had housed only the CouchDB client, but we had now decided to start introducing a general concept of a data layer to the agent platform, supported by a new back end service, “awagdata”. We would then go on to use `awagdata` to support many tasks related to the smooth operation of the agents (configuration storage, object persistence, logging) and the study implementation (content item capture, recording interactions with ML services for repeatability and so on).

2023-04-14
ML Service

Better handling of empty data & error logging

Testing of the model lifecycle – in particular with models in early stage of training – had exposed a few errors at the back end with how data was loaded and fitted.

2023-04-14
Data Service

Add data service for recording summaries

As part of our testing process we had identified that summary quality was inconsistent, with some items being better summarised by OpenAI than others. To gather more information on this, we decided to add the facility to record the input and output of the summarisation process, so that issues could be reproduced and re-tested after the fact. To do this, we added a new service to the back end, `awagdata`, initially supporting only this functionality, with data persisted in a local SQLite database. We would later go on to expand this data platform for many other uses.

2023-04-16
AwAg Interact

Implement Summarisation Feedback
Testing with users showed that summarisation quality was not always perfect; in some cases – particularly where the source content was short – the ChatGPT model would summarise badly or even fabricate summary text[a]. We added a feature to the UI to allow the user to give feedback on the quality of the summarisation; this would capture the item in question, its content and the user's rating of the quality of the summary text. The captured information was then stored using the `awagdata` service for later use.

[a]See also Emsley [2023]

2023-04-16
Data Service

Add services to record and retrieve summarisation feedback We expanded the data service to allow feedback from the end user on summary quality to be recorded and retrieved, by adding services `record_summarisation_feedback` and `get_summarisation_feedback`.

2023-04-21
Data Service

Add services to add and supply "dummy text"
Our first look at synthetic data was the enhancement to `awagdata` to allow what was initially referred to as "dummy text" to be stored & retrieved using services `add_dummy_text` and `get_dummy_text`. This text was simulated message content (to be generated elsewhere) that could be served sequentially one item at a time, grouped by a notional "topic". The design idea was that a future simulation service could periodically call this and get a series of pre-canned simulated "messages" for a given topic, channel or discussion. By pre-generating the content and storing in the data service this way, we could ensure good performance and reliability, as well as the ability to pre-vet content.

2023-04-24
AwAg Sim

Add `data.awagdata.simulation` & `modules.genus.simulate`
Our focus had by this point shifted towards supporting the study process, and we started to add support for synthetic data within the Awareness Agent, adding code to publish simulated (synthetic) content to Slack workspaces. This uses the `awagdata` service at the back end to serve up synthesised content to a set of channels. While this functionality is not strictly part of the Awareness Agent proper – it exists for the purpose of studying the agent – including it in the agent codebase was a pragmatic decision, as it could take advantage of existing modules and functionality.

2023-05-05
Data Service

Added combined dummy text generation
We added the ability to generate simulated "dummy" texts within the data service itself so that this process could be automated, by adding the service `generate_and_add_dummy_text`.

2023-05-10 *AwAg Augment*
Make summarisation less specific to news articles
The original GPT prompting used to generate CI summary augmentations was worded in such a way that it was more suited to summarising news articles. We modified this to make this more generic. We should also note that this was not a code change; prompt wording is picked up from text template files that can be more easily modified.

2023-06-12 *AwAg Eval*
Add ChatGPT/OpenAI evaluation implementation
This was the second major addition related to the study, supporting synthetic evaluation. Again this is not a core feature of the Awareness Agent design but is included for pragmatic reasons. The new feature passes the contents of augmented Content Items to the OpenAI API, asking it to evaluate the quality of the augmentation classification - in effect OpenAI marking `awagml`'s work. This was integrated into the agent's CI flow, with copies of CI's being siphoned off at the Allocate stage to be evaluated. The `awagdata` platform was used to store the evaluations.

2023-06-14 *AwAg Eval*
Attempt to coerce OpenAI to only output the requested JSON
The OpenAI API proved slightly unreliable; our design for evaluation was to ask OpenAI to output its evaluation as a JSON document conforming to a supplied schema. It would usually do so, but sometimes would respond in plain text or using an incorrect JSON schema. We had several iterations working on OpenAI prompts to make this more reliable.

2023-06-15 *Data Service*
Add evaluation data recording
In response to initial testing of the Evaluate functionality in the agent, we added the back-end facility to record the evaluation data via a new service, `record_evaluation_data`. This records the details of what was evaluated, how (OpenAI settings) and what the response was. The intent was to facilitate the recording of this information to support both debugging and the study.

2023-06-23 *AwAg Eval*
Add support for Perspectives
We introduced a concept called Evaluation Perspective, which was an attempt to add another dimension to how we view and run evaluations. The concept was that a perspective was a second way of looking at the evaluation, by asking a slightly different question of OpenAI. For example "has this item been classified correctly?" is one perspective, but we could in theory ask slightly different questions. The new perspective data structure supported the automatic flow of this, with different perspectives being defined in the agent configuration and then automatically passed to OpenAI. However, we found success was limited, mainly because we were not able to devise a convincing alternative question to the default.

2023-06-25 *Data Service*

Add Flow Monitor

Our service-queue design for the Awareness Agent meant that the flow of Content Items through the system should be clearly defined with known entry and exit points. We had found in testing that some items were in practice not appearing in the Interact UI when we had expected to see them. To help address these issues, and to provide general visibility on the internal workings of the agent, we added the Flow Monitor, recording each contact of a CI with a service or queue in `awagdata`. This meant that we could track the progress of individual items or aggregate flows and identify bottlenecks and other issues.

2023-06-26 *AwAg Eval*

Add Available Classifications to enable better evaluation

Previously, our evaluation requests to OpenAI did not list the available options that `awagml` had chosen from when it had made its classification; to improve the quality of evaluation we added this information to the evaluation request. This was relatively easy to do technically, as our design for the Single Classification Augmentation Item already included this information.

2023-06-28 *AwAg Eval*

Add `promptQueryJson` to `openaiApiCompletionsQueryInfo`

We had previously started using `awagdata` to record the evaluation operation and response for each evaluation request. In order to give us better reproducibility we added the raw prompt query JSON sent to OpenAI to the recorded data.

2023-06-29 *AwAg Platform*

Add state tracking & ability to disable Simulate & Acquire

In order to facilitate efficient testing we added the ability to selectively enable and disable services via the UI; we had found that the 'always on' nature of the services resulted (ironically) in information overload with items being generated and processed when we might want to focus on another aspect of testing. This feature allowed as to easily control this.

2023-07-04 *Data Service*

Add support for batching of evaluations

We added back end data support to facilitate a process of running evaluations in batch that we anticipated adding to the core agent code.

2023-07-09 *AwAg Eval*

Add batch execution of OpenAI evaluation queries

Our very first implementation had fired off a single evaluation request per Classification per Content Item per Perspective. This was quickly found to be very expensive in terms of token usage, as each request required significant prompting text, and adding more classifications to evaluate grew the demands substantially. We changed to a structure early on that would treat each CI as a single request, processing the evaluations for each classification. However, this was still not entirely efficient, so we added a facility for items to be combined and batched. That is, a single prompt request would include the common instructions and a certain number of combined CI's to evaluate. The idea was to reduce the amount of common text per item processed. However, this was at the cost of larger individual queries and higher complexity and we hit token limits with the GPT-3.5 models available to us at the time[a]. We had mixed results with this, finding that combining too many items caused failures due to a number of factors - such as token per request limits being hit, and also a higher chance of the request receiving a malformed response. We found that 3 items per request was a reasonably safe number.

[a]The `gpt-3.5-turbo-1106` model supported a maximum context window of 16,385 tokens, which a batched request of over 5 items could easily hit, while the more capable `gpt-4` had not initially been available for API use

2023-07-11 *Data Service*

Add evaluation failure reporting web services

We added services to capture evaluation failure data to `awagdata` in order to support parallel work in the agent platform.

2023-07-11 *AwAg Eval*

Add evaluation failure recording

We had added a specific mechanism for recording evaluation failures to `awagdata`, which we now incorporated into the Evaluation engine. Failures may be caused by an invalid response from OpenAI, network issues, running out of OpenAI quota for example. By testing for and recording which evaluations failed we gained the ability to re-run or debug them in a targetted manner.

2023-07-25
AwAg Eval

Add support for newer `/chat/completions` API for Evaluation

At this time, OpenAI moved its current API from “Completions” to “Chat Completions”[a], a change made since we had started work on OpenAI evaluations. This was a significant change for us, and actually very helpful. One change introduced as part of it was to formally support JSON as a response format - we had previously requested a JSON response in the prompt text, with imperfect results. With the new API we could specify a defined response schema, which we could then parse. The changes made to the request side were also significant; we could move from constructing a single text prompt to using a more structured array of message JSON. This helped to a degree with our process of passing a mix of JSON and text in the evaluation request, allowing more structure. However, we noted it was still short of a full JSON request format where the API would be sent a schema-based request rather than being told this on each occasion. The change to an array of system/user messages in the OpenAI request also opened a design possibility to us, as it made it easier to make the request process more modular in terms of request construction and variation. We would go on to use this later.

[a] `https://help.openai.com/en/articles/7042661-moving-from-completions-to-chat-completions-in-the-openai-api` [`https://perma.cc/KBZ7-MGPP`]

2023-09-07
Data Service

Add support for tags and for recording/fetching raw evaluation items

We made a number of additions to `awagdata` to support new evaluation-related functionality in the agent platform, by allowing data items to be stored associated with text tags, and adding new services to store and retrieve raw evaluation items (the CI data required to perform an evaluation operation).

2023-09-07
AwAg Platform

Add support for recording tags (in evaluation etc.)
Tag support was a major study-focussed change that emerged from the first test study iterations. The main driver for this was data management and organisation of tests. We needed to support concepts such as "test run 1" or "evaluation run 1" so that we could identify which data (content items, evaluations etc.) were associated with each run. This would enable us to clearly define the start and end of different study phases, and also allow us to run multiple different evaluations against the same content and identify different runs. We did this by adding the concept of tags. Each write of an item to `awagdata` would be accompanied by one or more tags, plain text strings. Use source of the tags varied: we could set a "current tag" (or multiple current tags) in the Acquire service via the UI; each incoming CI would the be tagged with these. Similarly for evaluations we added support for input tags (evaluate only those CIs that match the supplied tag) and output tags (i.e. store the evaluation output in awagdata with these tags attached). This allowed us a great deal of control and precision in the evaluation and general study administration process.

2023-09-12
AwAg Eval

Add on-demand processing for evaluations based on recorded evaluation items
Adding tag support had given us the ability to change our approach to evaluation processing. Our previous design for evaluation was to incorporate this step into the CI flow; however we had found this was not optimal. Sometimes we were not interested in evaluating during some types of tests for example (each evaluation execution has both a monetary and performance cost attached to it). We also wanted the ability to run evaluations in batch after changing something, such as the prompting text. To achieve this we added the ability to enable/disable the background Evaluate service as we had previously done for Simulate and Acquire, and also added a Slack command based facility to run evaluations in batch on demand. At the back end we added an `awagdata` facility to record all Content Items as they passed through identified by the currently active set of tags. We could then run on-demand evaluation batches against a set of CIs defined by the passed tag, and we were also able to identify and distinguish the output of these runs from the added output tags. We actually found that this was our preferred way of running evaluations, as it gave us in our role of study administrator much more control.

2023-09-18 *AwAg Eval*
Add Evaluate Explore interaction via Slack UI
We needed a mechanism to allow the study administrator to examine evaluations interactively and our first iteration of this was based on the Slack Interact UI, adding an Evaluate Explore feature. This could be used to return lists of evaluation results to the user as Slack messages in the Slack UI. Testing showed this was functional but not as easy to use as desired, so we noted that we would need to revisit this area later.

2023-09-22 *Data Service*
Add new object store to replace Cloudant/Bluemix
We added our self-implemented JSON object store, named SPOSS, to the `awagdata` layer, allowing us to persist objects locally with good performance, simplicity of access and certain functionality tailored to our specific requirements.

2023-09-23 *AwAg Platform*
Remove dependency on IBM Bluemix/Cloudant
We had previously been using the free tier of this commercial CouchDB database, but the T&C's changed so that it was no longer free for our level of usage. We stopped using it and transitioned to our own SPOSS object store service.

2023-10-02 *AwAg UI*
Add `awagUi` implemented in Angluar
We had not been satisfied with the first attempt of evaluation exploration so we make the decision to design a dedicated Angular based UI for the study administrator to interact with evaluation results. We also added the ability for the user (study participant) to submit feedback data via this UI on the quality of the evaluations. This has `awagdata` as the back end, which required some additional services to be added for access to data for reporting and interaction in this way. We named this new web based interface "awagUi".

2023-10-04 *AwAg Eval*
Allow additional tags to be added when processing evaluations
The original approach to on-demand evaluation processing added only a single tag to evaluations from each run. Study testing showed us that we could in fact use more tags to give finer grained control over evaluation outputs, so we added this feature.

2023-10-04 *AwAg UI*
Add mode1/mode2 support to `awagUi` evaluation explorer
Initial testing of the evaluation feedback process with the pilot study participants showed some issues with the design of the UI; some concepts were not always clear and in some cases there was clutter in the UI that the user did not want to see. We approached this by designing two presentation modes, one with stripped down explanatory information and a differently organised layout. The user had the choice of switching between these.

2023-10-13 *ML Service*
Add models subdirectory for agents
We implemented segregation at the back end for data from different agents, so that we could more easily clean down and reset in testing.

2023-10-15 *ML Service*
Add support for model description
We added the ability to store/return a descriptive text for each model so that this information could be used by the agent evaluation process. We opted to do this in the back end because the description handling fits well into the model lifecycle.

2023-10-15 *AwAg Eval*
Add support for model description
Testing of the quality of evaluations showed cases where the OpenAI model was making clear mistakes about the meaning of classifications in the model that it was asked to evaluate. We addressed this by passing a model description text along with the classification request to `awagml` and incorporating this in the prompting for the request - which produced noticeably better results. We then added support for this throughout the agent system, providing a mechanism for the user to add and maintain descriptions for models.

2023-11-19 *Data Service*
Enhancements to data simulation
We made a number of changes to the data simulation back end, in particular item generation via OpenAI, which we updated to use the new `/chat/completions` API that we had previously adopted for evaluations. This gave us access to better OpenAI models, with improved quality of generated items.

2023-11-22 *AwAg Eval*
Add recording of classification actions
We had found during the pilot study phase that it would be helpful to maintain a record of the actions of the user to classify items – that is, where the user used one of the available UI mechanisms to perform a training event related to a content item (changing classification value or confirm correctness of a classification for example). We had not previously been capturing a log of these actions when they were performed, but realised that having this data would provide important experimental insight. A classification performed by the user is a de facto evaluation of the original classification by the human, mirroring the evaluation by the OpenAI-based evaluator. This would provide us with an additional mechanism for getting feedback data on the quality of the AI evaluation by comparing the agreement with the human and AI evaluations.

2023-11-26
AwAg Platform

Add/use non-volatile JSON Object Store to facilitate classification feedback

We had previously found that we needed to implement a workaround for limitations in the use of Slack for giving evaluation feedback related to the passing of state/context information. There were limits in the Slack API on how much data could be included in drop-down menu items, and there was a lack of ability to store arbitrary context data in items posted to Slack. We had addressed this by storing a data object – “Classification Feedback Item” in memory server-side and then including a reference ID to this in the posted item. Thus when the user made a change via UI action the submitted ID was used to look up the context information. However our original implementation for this used volatile memory to store these items - the flaw with this was that the published Slack items were not volatile, so there was the possibility of the user accessing these after the reference had been lost from volatile memory due to a server restart for example. We addressed this by instead persisting these objects to the SPOSS object store.

2023-11-28
AwAg UI

Change AGREE to CONCUR in `awagUi`

The pilot study user experienced some difficulty with the process of giving evaluations, as sometimes the wording was not clear to them. One case was when they were being asked whether they agreed with the evaluation of a particular classification. This is not the same as agreeing with the original classification (for example if the classification is incorrect then the human and AI would *agree* with each other that they *disagree* with the classification). We addressed this to an extent by using the word “Concur” instead.

2023-11-30 *AwAg UI*

Add training item display to `awagUi`

The pilot study exposed some limitations with the training process for the user-defined models when used in a bulk context (such as bootstrapping a new model or generating statistically valid amounts of data). When training within Slack we found that the user followed a process that was straightforward but inefficient and limited by UI speed. The user would view an item within its channel in Slack, then click on the classification drop-down, then select a different classification (to change classification) or re-select the same classification (to generate positive training). If reclassified, then they would need to wait for the back end service to process the action and move the item from its old to new channel before proceeding with the next one. If classifying a single item then this was sufficiently performant, but in the study the user would be asked to apply one classification per item per model – so if the user had 5 User-Directed ML models set up they would potentially need to repeat this process 5 times. It was too time consuming. We decided that significant action was needed to address this, and added a new training UI to `awagUi`. This UI was much faster and more responsive than going via Slack, and also let the user select corrected training classifications on every model defined for the CI in a single operation, significantly reducing the number of clicks and individual processing delays. This new UI would also then go on to be a basis for a secondary Interact UI.

2023-12-08 *Data Service*

OpenAI file handling

We had made a design decision based on our experiences so far to add the ability to train or fine-tune OpenAI models used for evaluation[a], so that we could compare the performance of trained vs untrained models. To support this we added a mechanism to generate, manage and upload training data files to OpenAI, and then to use these files to train models. To make the process manageable, we chose to handle the whole process within the `awagdata` layer. This includes generating training objects from recorded classification actions, storing these within our object store, using them to generate OpenAI files and managing the querying, execution and deletion of models on the OpenAI side. We took this approach because OpenAI provided only bare API support for these operations with no higher management ability; we had found that it was too easy to lose track of the various models and files on the OpenAI system, and not always easy to tell the provenance of these files. Our system tracks history, source and other information related to all artefacts that we create in OpenAI for easy end efficient management.

[a] `https://platform.openai.com/docs/guides/fine-tuning` [`https://perma.cc/CKJ2-LVVK`]

2024-02-18 *AwAg Eval*

Add Evaluation support to `awagdata`

Our original design for Evaluation was to have the processing done within the Java Awareness Agent application at time of CI generation; we later added the ability to batch this and run on-demand evaluations. During the pilot study we found that we had moved more and more towards on-demand tag based evaluation processing and rarely used the original mode. Because of this, we decided to replicate the evaluation functionality in the back end, so that it could be run entirely independently of the agent itself. This was consistent with the agent design concept, which does not include evaluation as a core function, and was also compatible with how we ran offline evaluations: all the data used to run these was already stored in the `awagdata` layer, and the outputs were also recorded there. Changing our evaluation engine to the `awagdata` platform meant that we could more easily run and manage evaluation jobs via simple web service calls. We also retained the code in the Java Awareness Agent, because the structured nature of the evaluation request object and the standardised prompt structure meant that all we needed to do was ensure that a few resources were kept synchronised between the two, such as schemas and prompt templates.

2024-02-18 *Data Service*

Add Fixit route

We had discovered during the pilot study run that a minor technical error had prevented the proper recording of evaluation items during the course of the study – they should have been recorded to the Data Service as they came in, but this was not happening. This would have been catastrophic for the study as it would prevent any evaluation being run on the data, which could not easily be re-captured. However, the necessary data to construct evaluation items after was in fact being recorded elsewhere within `awagdata`, so we were able to write code to extract this information and retrospectively construct the necessary evaluation items.

2024-02-19 *AwAg Eval*

Improve OpenAI prompting

In response to preliminary results in the pilot study we made several changes to the prompt text passed to OpenAI for evaluation requests. We found that we needed to be much more explicit about some of the things that we wanted the evaluator to do, and also needed to repeat some points in multiple ways to reinforce the message.

2024-02-20 *AwAg Eval*

Support Lightweight Mode option for evaluation

We had found that our OpenAI evaluation requests had been growing increasingly large, having an impact on both query performance and token usage. To address this we added a "lightweight" mode, where certain content could be omitted by setting a flag. For example, the schema describing the content item is not passed in this mode. The intent was to use this in conjunction with tags to run evaluations in each mode for comparison, while also using different OpenAI models at the back end. Part of this approach was also to use pre-trained (fine-tuned) OpenAI models, particularly in conjunction with lightweight mode, and compare results across models. The previous decision to introduce tags to the system made this practical.

2024-02-23 *Data Service*

Add standard vs split outputs

We found that many of the parts of prompts that we were sending to OpenAI had commonality for different functions (evaluations, training item generation etc.), so reorganised the back end code to approach this more efficiently and remove redundancies.

2024-02-23 *AwAg UI*

Add chat component to `awagUi`

To facilitate interactive testing, we added a simple chat UI so that the user[a] could enter some text which would be executed against the OpenAI model; this is functionality already provided by the ChatGPT UI[b], but that UI did not provide access to trained models, only to vanilla ones. By adding our own UI we could run arbitrary requests against our own trained models, also utilising our own system request prompt elements. This allowed us to more efficiently test model and prompt changes in an iterative fashion.

[a] In this case the user is the researcher rather than a study participant
[b] `https://chatgpt.com/`

2024-02-23 *Data Service*

Add subsets support

We had observed during the pilot study that the volume of content was too great for the participant to process all of it – in itself this was not unexpected and not a problem. However, we found that we would get more useful study results if the user acted on the same subset of items in each phase (initial classification and evaluation feedback), but it was important also to get a good distribution of items and not just have the user process in date order. To support this we added support for subsets to `awagdata`. We did this by adding the ability to add a set of subsets for the data, were we could define random subsets[a] of the recorded Content Items identified by tag and a subset percent label. So for example we might generate a subset of 25% of the items for `tagA`, which would be identified by the path `/subset/tagA/25`. The user-facing UIs could then be filtered to show only items in that pseudo-random subset.

[a]Using Python `random.sample()` to generate a pseudo-random sample

2024-03-06 *AwAg Eval*

Edit prompt content

Previous changes had improved evaluation quality, but we had found that sometimes OpenAI would not consider all classifications in its evaluation so we changed the prompting to encourage it to do so.

2024-03-14 *Data Service*

Add statistics and improve reporting

To improve the efficiency of result processing during and after then studies, we added a stats service, that would generate a set of statistics based on the evaluations and other data for a given study/tag, outputting as a JSON document containing a stats "package". Having such a package automatically generated would greatly improve the ability to generate updated statistics for each study instance.

2024-03-22 *AwAg Eval*

Address occasional incorrect property names in OpenAI response

We had found that sometimes OpenAI would return JSON with incorrect property names, such as switching from `camelCase` to `using_underscores`[a]. We implemented a workaround to check for and correct this. However we also noted that we experienced this more often when using Juneau's `ObjectMap.toString()` in Java to serialise items to JSON (which generated JSON that did not have quotes around property names). Changing the serialisation code to ensure that property names were always enclosed in quotes largely resolved this issue.

[a]`https://news.ycombinator.com/item?id=34525139` [`https://perma.cc/4F7Y-CUSD`]

2024-04-04 *AwAg UI*
Support for text-likert mismatch via partial feedback
We had found in some cases that the text description generated by OpenAI for its feedback did not agree with the Likert value that it had assigned (typically in this case they would be opposite to each other). This was a significant enough problem that we added an element to the evaluation feedback UI to allow the user to flag this easily, allowing us to capture data on the number of affected items.

2024-05-02 *Data Service*
Add agree/disagree exclusion
To improve the quality of data used to train models, we added a feature to allow the list of user classification actions to be used for training to be filtered to exclude those where the user agreed with the classification or those where the user had disagreed. We had found that the user was agreeing with the classification chosen by the trained models in the majority of cases (often a very large majority) so the training data sets had been very skewed towards entries where there was agreement. The training data would be more effective with a better balance of content [Dube and Verster, 2023], so that the model would be trained with a similar number of disagreements to agreements. Adding these filters meant that we could build up our evaluations in multiple steps, assembling a training set with a higher proportion of disagree actions than a single unfiltered pass would generate (i.e. oversampling 'Disagree').

2024-05-02 *Data Service*
Add dataset merging
To support the process of building up training datasets with varied compositions and using these to fine-tune models, we added a feature to allow the administrator to merge datasets – so for example two datasets that had been generated with different ratios of agree/disagree could be combined into one and then used to train an OpenAI model. This allowed us more control over the composition of fine-tuning datasets.

2024-05-08 *Data Service*
Work on evaluation job processing
Our evaluation processor was designed to process items that match a given tag and subset, that did not have a corresponding evaluation already recorded. If an error was encountered during evaluation (for example due to an OpenAI error) then no evaluation would be recorded – so when the evaluation was next run, the failed item would be retried. Due to a small percentage of failures in each run, we found that we needed to run a number of passes before all items were evaluated. We added code to make this process more manageable and transparent.

2024-05-23 *Data Service*
Add tabular statistics, broken down by classification
We improved the structure and content of the statistics output to better support the study analysis.

2024-05-27 **Excel stats pack export**
Data Service

We added the ability to automatically output a statistics package to Excel format for easy analysis, using `pandas` to output dataframes to Excel worksheets[a][b] and combine to multi-sheet formatted workbooks.

[a] https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.to_excel.html
[b] https://perma.cc/AN99-MZMP

Bibliography

[Dube and Verster, 2023] Dube, Lindani and Tanja Verster (2023). “Enhancing classification performance in imbalanced datasets: A comparative analysis of machine learning models”. In: *Data Science in Finance and Economics* 3.4, pp. 354–379. ISSN: 2769-2140. http://www.aimspress.com/article/doi/10.3934/DSFE.2023021.

[Emsley, 2023] Emsley, Robin (2023). “ChatGPT: these are not hallucinations – they’re fabrications and falsifications”. In: *Schizophrenia* 9.1, pp. 4–5. ISSN: 27546993. DOI: 10.1038/s41537-023-00379-4.

[Pedregosa et al., 2011] Pedregosa, F et al. (2011). “Scikit-learn: Machine Learning in {P}ython”. In: *Journal of Machine Learning Research* 12, pp. 2825–2830.