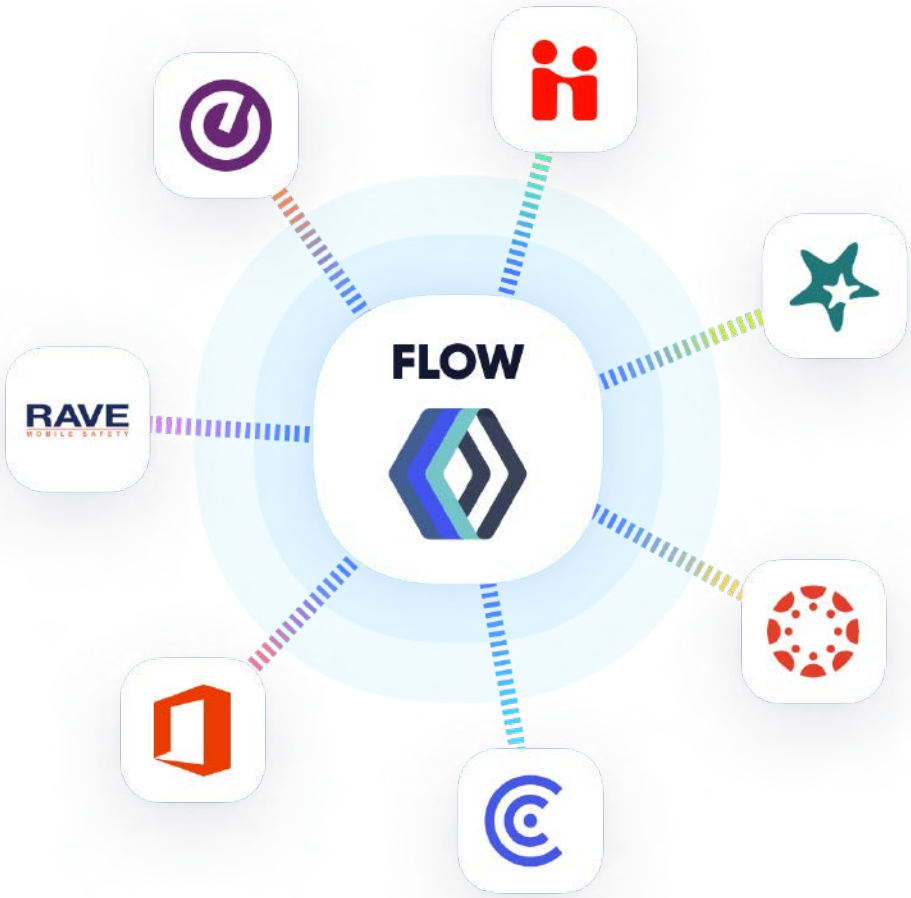
FLOW MIDDLEWARE

What is FLOW?

FLOW is an integration engine. That's had a lot of names over the years: Middleware, Enterprise Service Bus, IPaaS (Integration Platform as a Service), etc. That means it's a system which knows how to read data from a number of different systems and write them to other systems. It is a flexible engine which allows you to read data from one system, transform it into another shape (perhaps redacting sensitive information, or enriching it with information from other sources, etc) before putting it into another place.



Why Is That A Good Thing?

You have a number of different systems, and they're all for specific purposes. They each have their own data, and they'd each be better if they knew some of each others' data. Your LMS is better when it knows in what courses your users are enrolled.

Your SIS is better when it knows how your students are performing in the LMS. Your student success system is better when it knows how active your students are on your forums. Moving information between those systems increases the value of each of those systems. Having a centralized point of control keeps all that data movement under your control, and within your oversight.

Why not just have each system individually connect to the other systems it needs, in a point-to-point manner? Well, firstly, while there's a mathematical complexity reason (see appendix B, if you're interested), which will help you understand why an integration engine can reduce effort and cost, it's mostly so that we can keep all the integration logic in one place, defined in one language, and providing a single window over the data flows.

What Can FLOW Do?

FLOW is used to share data amongst your systems. This can take on many forms, which means FLOW can do lots of things, and connect lots of systems together in various configurations. For example, FLOW is currently used for:

Reports

Extracting information from LMS, SIS, Analytic, Social systems, and more, to generate reports which combine information for a more complete view of the users (e.g. a report of student success indicators from multiple systems for a student) .

Embed

Building new features into other systems, by building complex applications which can be embedded into other web applications through those systems' portlet interfaces.

Profiles

Reading information from LMS and SIS systems to create consistent profile information for students in other systems.

Notices

Consuming information from messaging systems and redistributing those notices through other channels, such as SMS, Email, Social networks, and Mobile Push Notifications.

Alerts + Tasks

Extracting information from LMS and SIS systems to generate alerts and tasks to keep users engaged and on-track.

Enrich

Enriching data flows to add extra information, or redact information. Enriching authentication flows (SAML, OAuth, CAS, etc) to add extra information, or redact information from the user claims which are given to the final services.

Widgets

Exposing user-specific information through dynamic widgets which can be embedded into other web applications using those systems' portlet injection interfaces, allowing users to use those systems as one-stop shops of capability, even though the functions they're performing are in different systems.

How is that different from other middleware?

FLOW has 4 primary priorities:

Higher Education Connectors

ONE

Most middleware (Mulesoft, Microsoft Biztalk, Oracle Service Bus, Dell Boomi, etc) is made up of a core engine and a number of connectors, each specialized to a particular 3rd party system or technology (Google Mail APIs, Apache Kafka queues, etc). While that's very useful, when there's no specific connector, it leaves the developer using generic technology connectors (HTTP, for instance) to connect to Higher Education systems (Ellucian Colleague SIS, Canvas LMS, etc), and that means the developer has to handle the idiosyncrasies of the 3rd party system themselves - authentication, pagination, error handling, etc. FLOW is built specifically for the Higher Education market, and our first priority is building connectors for Higher Education systems, so that the idiosyncrasies are handled.

Visibility

TWO

It's your data. You need to know where it's going, and what's going on with it. Higher education is full of legacy systems and complex rules, and most integration tasks become far more complex than predicted because the systems they're integrated with have ghosts of previous systems still inside them - datasets which follow rules which no longer exist in any current system, or rules which have to remain in play for as long as the last student who uses them is still on the books. It's important to be able to see how the data moves, when it's transferred from one system to another, and what happened when it moved from one to another.

You shouldn't need to be coincidentally online and watching when it happens. It should have logs available. It should be able to notify of events occurring. Importantly, you should be able to control how much or how little of that you want. Higher Education is full of data flowing in all sorts of directions, and you should be able to choose what you want to collect and be notified about.

Flexibility in the Hands of the User

THREE

You own the data. You should own the behavior too. That means you should be able to turn on and off synchronizations, at your convenience, not by logging a service desk ticket with the provider or consultant and hoping it gets picked up in the next week, but by logging into something you own and turning it on or off. You should be able to tweak the logic when your data changes (remember those pesky rules mentioned earlier? They're not just in the past - they're in the future too - new rules come into play, and you should have the power to change them to suit).

Data Only Where it Needs to Be

FOUR

Your data shouldn't need to be stored in places you weren't expecting. Data should only be transmitted when you're comfortable with that, and where you're comfortable. FLOW provides mechanisms for exposing views over data to other systems without transferring the data into those systems. FLOW provides mechanisms for abstracting systems from each other, so that they're only loosely coupled.

How Hard is it to Try?

FLOW is hosted on the cloud, so you don't need to worry about hosting and maintenance.

FLOW has secure connectors to your data center, using industry standard technologies, so you don't need to worry about that either.

FLOW has recipes/templates for common integrations, so you can set up your connections in a standard fashion quickly, and then you can take your time tweaking them to suit your institution's specific needs.

How Long Does it Take to Set Up?

We set up a lot of institutions on FLOW, and we normally work with them over a period of 6-12 weeks to set them up (though set up time of course has many variables).

At a high-level, we spend some time working with them on discovery - what systems do they have, what information do they want flowing where? This often takes a meeting or two, but usually we space those meetings out over a week or two so that they have time to think about the complexities of their domain.

When those have pre-existing templates/recipes, we provide those in a working environment where they can test and validate, and we help them tune the behaviours to their specific needs. This often takes a few meetings over two or three weeks, with testing usually taking place on both sides.

When those require new templates/recipes to be created, we usually create those over the course of a week, with a few back and forth interactions with the partner.

Our shortest implementation timelines have taken around 1 week. Our longest implementations are living relationships where the partner is finding new features they'd like, and new behaviours, sometimes implementing them themselves, other relying upon us to create new recipes/templates, but usually somewhere in the middle of those two extremes.

What do I Need to Know to Implement FLOW?

You need to know about your domain - what do you want to integrate? What systems are they? How do you create a trusted relationship in them (create credentials, generate tokens, whitelist access, etc)? What data is in there, and how is it identified? Does the system you want to send that data to have any shared identities (you might have a student id number which all the systems know, for instance)?

You need to know a little javascript - all of the definition of behaviour of flow is written in javascript and JSON. We think that's the most commonly understood computing language right now, with the widest audience, and we firmly believe that integration is hard enough without making you learn a few more languages as well.

You need to read a little about our patterns. Patterns are a necessary study for all integrations, because those 3rd party systems you're hoping to integrate together each had their own patterns, and you need to understand a little about all of that to make it fit together. FLOW supports a lot of different patterns, and you don't need to know them all, but you may need to understand a little about the ones which your chosen systems use. Fortunately, the recipes/templates can get you started there.

Appendix A: **Connectors**

Systems we currently integrate with FLOW:

Ucroo Campus
Ellucian Colleague
Ellucian Banner
Jenzabar
Canvas
Moodle
Blackboard
Touchnet
Rave
National Student Clearinghouse
Regroup
Rhino Fleet Tracking
Workday
BankMobile
OmniAlert
Handshake
Starfish
Jira
Github
Google Analytics
GhostInspector
Salesforce
Wordpress
Microsoft Exchange
Gmail
Google Calendar
Google Drive
Google Sheets

Technologies FLOW uses:

HTTP APIs
REST APIs
SOAP APIs
Oracle SQL
MySQL
Amazon Aurora
H2
PostgresSQL
Google CloudSQL
Google PubSub
Google BigQuery
Google Storage
Google APIs
Amazon S3
Amazon Billing
Amazon SQS
Amazon APIs
RSS
CAP
Atom
IMAP
SMTP
SCP
SFTP
SSH
Webcrawler
GraphQL
ICal
CSV
TSV
JSON
XML
JWT
OAUTH
SAML
CAS

Upcoming Systems

Ellucian PowerCampus
Desire2Learn
Sakai
Glassfish JMS

Appendix B: **Multi-System Complexity Comparison Between Point to Point and Hub and Spoke Patterns**

Imagine you have 2 systems, and they each need to speak to each other. In system A, you write logic to speak to system B, and in system B, you write logic to speak to system A. In this model, for 2 systems (A,B), you have 2 connections (A->B, B->A).

Now imagine you have 3 systems, and they each need to speak to each other. Now it becomes:

3 systems (A,B,C)
6 Connections (A->B,A->C,B->A,B->C,C->A,C->B)
Now imagine you have 4 systems, and they each need to speak to each other. Now it becomes:
4 systems (A,B,C,D)
12 Connections (A->B,A->C,A->D,B->A,B->C,B->D,C->A,C->B,C->D,D->A,D->B,D->C)

You'll notice that the connection count is represented by the algorithm:
(Systems * (Systems -1))

This is the point-to-point complexity algorithm

By contrast, when you add an intermediating hub, which the spokes connect to, the numbers change (in this case, I'm using "H" to represent the hub):

2 systems (A,B,H)
4 Connections (A->H,H->A,B->H,H->B)
3 systems (A,B,C,H)
6 Connections (A->H,H->A,B->H,H->B,C->H,H->C)
4 systems (A,B,C,D,H)
8 Connections (A->H,H->A,B->H,H->B,C->H,H->C,D->H,H->D)

The complexity algorithm changes to:
2 * Systems

So, when an organization has only 2 or 3 systems, the value of an intermediating integration engine doesn't appear important, but once you pass 3 systems, it becomes very valuable indeed.

FLOW MIDDLEWARE