SCTE CABLE-TEC EXPO 22
SEPTEMBER 19-22 • PHILADELPHIA, PA

2022 Fall Technical Forum
SCTE • CABLELABS • NCTA

Advanced Workflows for Onboarding

A Technical Paper prepared for SCTE by

Peter Cline
Principal Engineer II
Comcast
1800 Arch St. Philadelphia, PA
215-917-2645
Peter_cline@comcast.com

Priyasmita Bagchi
Sr. Manager Software Engineering
Comcast
1800 Arch St. Philadelphia, PA
priyasmita_bagchi@cable.comcast.com

Nirav Dave
Principal Architect II
Comcast
183 Inverness Dr W, Englewood, CO
720-236-5223
Nirav_Dave@cable.comcast.com

Table of Contents

List of Figures

1. Introduction

Every brilliant network deserves a brilliant onramp, one which makes it simple and easy for customers to get quick, ready- access to the services for which they are paying. This first interaction with services and products will leave a lasting impression that will be difficult to change if it isn't positive. At Comcast, teams are intently focused on ensuring that this onboarding or first-time user experience (FTUE) is frictionless and positive for our customers. We look to minimize customer interactions stemming from difficulties with onboarding and to direct as many folks as possible into the nself-install installation route. This paper examines how we are using cloud native such as workflow orchestrators and Functions as a service (FaaS) to realize this goal. We will examine how previous paradigms employed for onboarding provided a foundation for the new workflow orchestration architecture presented here and helped propel us in that direction. From the perspective of software development, we wish to develop platform services that are robust, highly observable, scalable, quick and easy to modify and deploy, while providing the best customer experience.

2. Onboarding Process

The process of onboarding IP (Internet Protocol) gateway devices has evolved with changing technologies and business opportunities. Customers used to rely heavily on technicians to help onboard their equipment. Interactive Voice Response (IVR) systems were available to help customers who needed it. There has long been a web interface used by both customers and technicians to facilitate the onboarding process. With the advent of mobile applications, new business opportunities arose. Comcast introduced mobile applications and recognized the potential of the superior user interface there to improve the onboarding experience. This was also about the time when our advanced xFi gateways running the Resource Development Kit – Broadband (RDK-B) firmware were introduced. These gateways would necessitate changes to the existing onboarding process, as additional backend services were now involved. Initially only IP gateway devices that were leased to customers ran the RDK-B firmware and consequently our focus was solely on these devices. Onboarding functionality was implemented as part of the single Application Programming Interface (API) supporting most of the functionality for what was then called the xFi app. This paradigm served us well for a long time.

The success of the onboarding process in the xFi app encouraged the business to seek additional opportunities. Soon discussions were underway about how we could support customers who chose to bring their own device, so-called customer owned and managed (COAM) devices, when subscribing to Comcast High Speed Data (HSD) service. A growing percentage of Comcast broadband customers opt for this route and ideally their onboarding experiences are as consistently positive as those experienced by customers who lease gateways from Comcast. Concurrently, software engineers began to recognize the drawbacks of operating software as large applications performing many different functions and the era of microservice architectures dawned. As we contemplated adding support for COAM devices to the xFi application, we began examining how we could leverage the benefits of the emerging microservice architectures at the same time.

The decision was made to build the COAM activation as a collection of FaaS components that would then be orchestrated by an external workflow engine. Separating concerns in this way affords us a host of benefits which we shall examine in detail.

Onboarding requires asynchronous execution of a series of tasks to activate service, update core device configurations such as wireless fidelity (wi-fi) radio credentials, provision or update status of the device in a central device repository, and conditionally apply other device configurations. All of these functions could potentially be performed by

discrete functional units as suggested by microservice architectures. The decision was made to build the COAM activation as a collection of FaaS components that would then be orchestrated by an external workflow engine. Separating concerns in this way affords us a host of benefits which we shall examine in detail.

The COAM onboarding workflow was successfully implemented in this fashion and introduced into the xFi app. We were now able to provide the superior mobile application onboarding experience to both leased and COAM customers, so when the business value of facilitating onboarding of gateways for customers in Multi Dwelling Units (MDUs) via the mobile application became apparent we were well positioned to tackle that work.

3. Leased Gateway Orchestration

The introduction of leased gateway orchestration to the xFi mobile application was a large success. We were able to gain more insight into how the onboarding process was performing, identify opportunities to improve the customer experience and the orchestration of all the requisite back-office processes. Most significantly, customers could onboard their devices out-of-band, meaning without being connected to the wi-fi network broadcast by the IP Gateway itself. This opened many opportunities for an improved user experience in the xFi app and facilitated the increased use of Self-Install Kits (SIKs) for IP Gateway onboarding. SIKs meant fewer technicians visiting homes to facilitate the onboarding process. The leased gateway onboarding functionality was part and parcel of the platform API supporting the xFi mobile application. This single large software application was in line with how most folks were building software and made it easy to deploy and operate. Figure 1 illustrates this leased gateway workflow orchestration.

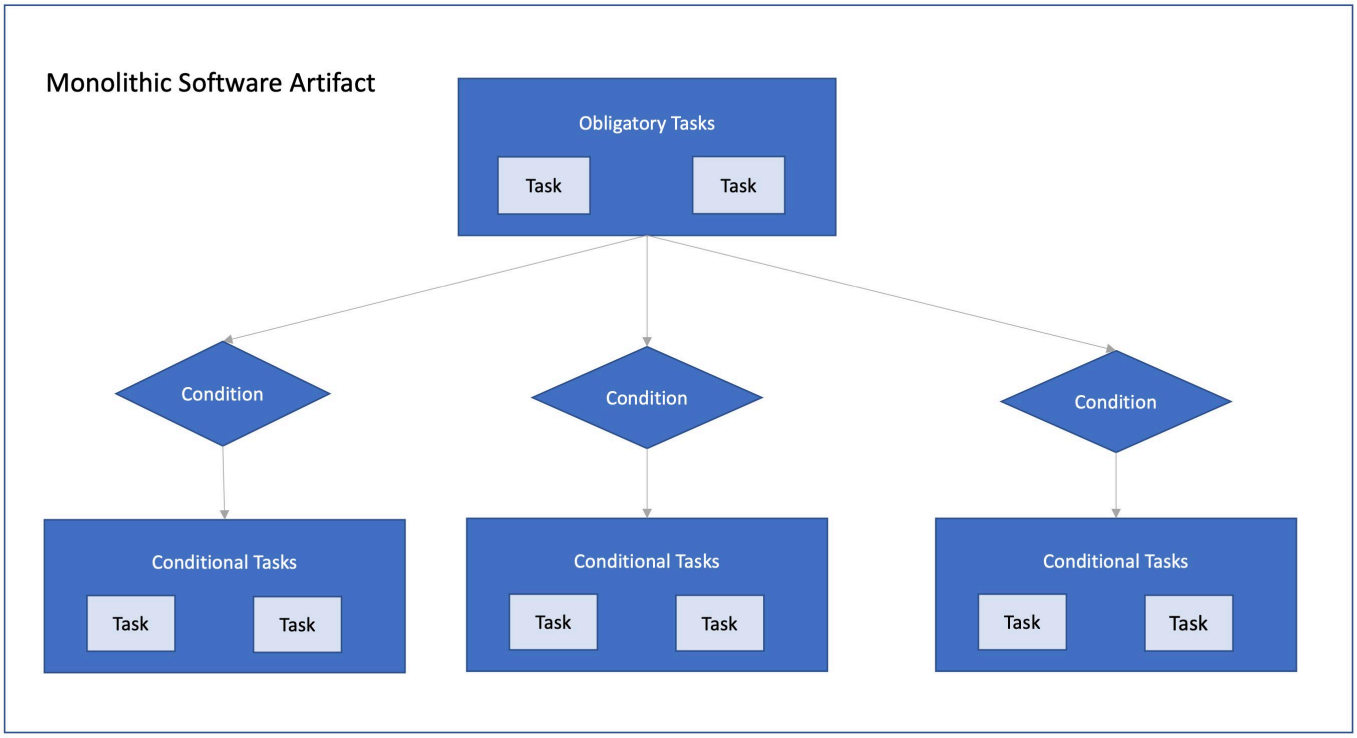

Figure 1 – Leased Gateway Workflow Orchestration

Our leased gateway onboarding implementation provided some great benefits as detailed here.

3.1. Observability

Because we were orchestrating the whole onboarding process from a single software workflow, we had great visibility into the whole process. A home-grown system for distributed tracing, named Money, which Comcast later open-sourced, allowed us to follow a request from the mobile application client through our orchestration to the various back-office services involved in the onboarding process. As long as the leased gateway flow was the only one supported this process worked well and gave us much insight into the onboarding workflow. As we added additional workflows for COAM and MDU devices we realized we had an opportunity to do better still and gain insight into how common tasks performed in aggregate across all the onboarding workflows.

3.2. Orchestration Modification

Once established, the leased gateway onboarding workflow changed infrequently. Within the xFi platform API, the code for the workflow was well encapsulated and could be modified when need be to support evolution of the onboarding process. The addition of the COAM workflow provided the impetus to extract the workflow definition from the code itself so that it could be managed and evolved independently of the code that implemented the business logic described by the workflow definition.

3.3. Workflow Testing and Deployment

While the leased gateway onboarding workflow was the only one we supported, testing and deployment was straightforward. Changes would be made, and the workflow would tested via a mixture of automated testing and manual testing using the client interface. As we began to consider COAM onboarding, we looked for opportunities to manage the two workflows separately so they could be tested and deployed independently and changes to one workflow would not necessitate any testing of the other if the changes did not apply.

3.4. Workflow Composability

In the monolithic orchestration in Figure 1, the conditional tasks shown are responsible for additional configurations that fall outside of the primary onboarding function as previously discussed. The gateway device is onboarded and functional from the customer's perspective after the primary tasks in the orchestration have been successfully completed. The advent of microservices architecture and the emerging architecture for COAM onboarding would provide the opportunity to treat these conditional workflow tasks as independent workflows whose execution could be done independently of the primary onboarding workflow. This separation would allow each of the workflow to be given the retry semantics they required and provide greater clarity as metrics could be separated and tallied individually for each workflow.

.

4. Workflow Orchestration Architecture

The introduction of COAM onboarding in the xFi mobile application gave us the opportunity to implement the architecture we'd been formulating, built upon an external workflow engine and independently deployable FaaS components. This paradigm decouples the workflow orchestration from the task implementation. An externalized workflow engine handles orchestrating tasks for which implementation isn't coupled to the workflow engine in any way. Tasks are implemented as discrete deployable units that are likewise free of dependency on each other.

The introduction of COAM onboarding in the xFi mobile application gave us the opportunity to implement the architecture we'd been formulating, built upon an external workflow engine and independently deployable FaaS components.

The implementation of a common task only needs to be completed once, and it joins a library of functionality from which engineers can draw as they build additional workflows. These workflows are expressed in a domain specific language (DSL) wherein the interactions between the tasks are described along with instructions about conditional execution of tasks, how to handle error conditions, and when to retry task executions. These workflow definitions are consulted by the workflow engine and used to determine which tasks need to be executed and in what order. This has allowed us to quickly support new device classes or types as they are introduced, and to build workflows that handle some ancillary concerns around gateway device configuration that often accompany onboarding, such as

the conditional enablement of gateway agents according to business and product requirements. This paradigm also allows for greater visibility into workflow performance so that potential issues can be identified and understood quickly and addressed with minimal disruption to our customers. Additionally, it provides deeper insight into how each task of the workflow is performing, both in the context of a single workflow and in aggregate across all the workflows in which it is used.

A high-level view of our new architecture looks like the image below, where service is used generically to refer to independently deployable functional units. In our specific implementation we rely on FaaS:

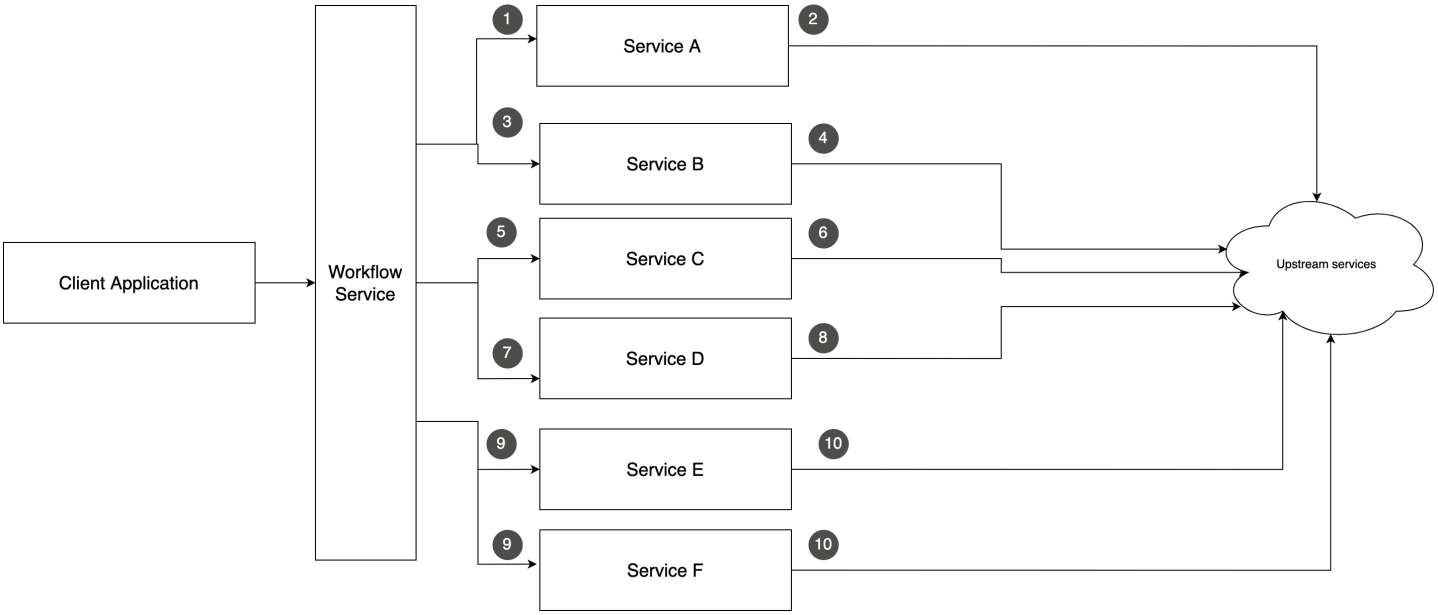

Figure 2 – Workflow Orchestration Architecture

4.1. Architectural Constructs

There are a few primary architectural constructs employed by our workflow orchestration architecture which are described here. From these basic building blocks, we can compose and execute new workflows, and reap other benefits, all of which we will examine in detail.

4.1.1. Task

A task is a discrete functional unit of work. In the onboarding domain this most often equates to a Hypertext Transfer Protocol (HTTP) interaction with an external service used to fulfill some specific, often repeated function. Examples include interactions to provision a gateway device in a centralized device repository, or to associate a gateway device with a set of configurations in a cloud database. Tasks should be generic enough to be reused by multiple workflows. If more than one workflow requires the same piece of functionality, they will ideally use the same task implementation to accomplish this work. The difficulty here lies in making tasks reusable without making them too large or generic. If tasks aren't granular enough, some of the benefits of the workflow orchestration architecture are lost, particularly the observability and state management functions we will look at shortly. In our architecture, task

implementations are typically done as FaaS. Serverless FaaS functions are cost-efficient as they only consume the computing resources they require. During periods of general inactivity, such as in the very early hours of morning when most people opt for sleep over onboarding of their gateway devices, little to no compute resources will be consumed. We've also done a good amount of work to standardize the request and response payloads each task uses so that communication between tasks and with the workflow engine is facilitated.

4.1.2. Workflow Definition

Workflows are defined in a DSL wherein the set of tasks comprising the workflow, the order of their execution, number of retries and retry semantics for each service, the inputs and output fields for the entire workflow, and the input and outputs required by each task are specified. The workflow definition may specify for each task, a fixed number of retries repeated at a fixed interval, an exponential back-off strategy in which each subsequent retry is delayed by an order of magnitude more time than the last, or even that no retries are warranted. The workflow can also specify that tasks be executed concurrently or serially and provide conditions that must be met before a task is executed. Exit criteria for a workflow may also be found in the definition. Certain task failures should result in termination of the workflow, while in other cases workflows may be able to continue after failure of a task that is optional or not essential to the overall workflow success. Our architecture uses a serverless cloud-based workflow engine as described in the next section. Each workflow definition can be managed via the cloud console or defined declaratively as JavaScript Object Notation (JSON) files and managed independently of the cloud console. This allows for automated deployment of workflow modifications. They can also be visualized using the tools provided in the cloud console.

4.1.3. Workflow Engine

Our architecture needs an engine to drive workflows. This engine reads workflow definitions to receive its marching orders and then executes the instructions defined in the workflow definition. It is the engine that orchestrates the task executions and applies the retry, concurrency, and conditional rules specified in the workflow definition. The workflow engine manage failures, retries, and parallelization as described in the workflow definition so developers needn't be concerned with these ancillary functionalities and can focus instead on where they can produce the most value, namely in implementing the business functionality required by the onboarding process. All this functionality was provided by the initial leased gateway onboarding, but now it is handled as a separate component that can orchestrate many workflows and focus on its primary functionality without being bogged down with the details of task implementations or the business logic embedded in the tasks. Since the tasks use standardized request and response payloads the workflow engine simply feeds the output from one task to the next task in the workflow.

4.1.4. Workflow Integrator

The workflow integrator is a component that provides a means for client interaction. It performs several key functions that fall outside the purview of the workflow engine, the tasks, or the workflow definitions. Chief among these is authorizing clients wishing to initiate workflows, mapping client requests to workflow definitions, starting workflows via the workflow engine, and reporting status on currently

executing workflows to clients who request it. The workflow integrator maintains a mapping of APIs to workflow definitions. This mapping will also include data about what inputs are required for each workflow. The client application makes a call to an API exposed by the workflow integrator which consults its mapping, authorizes the client, verifies proper workflow inputs have been supplied, and starts the Step Function State Machine that correlates to the client request.

Our specific implementation of the workflow orchestration architecture looks something like this:

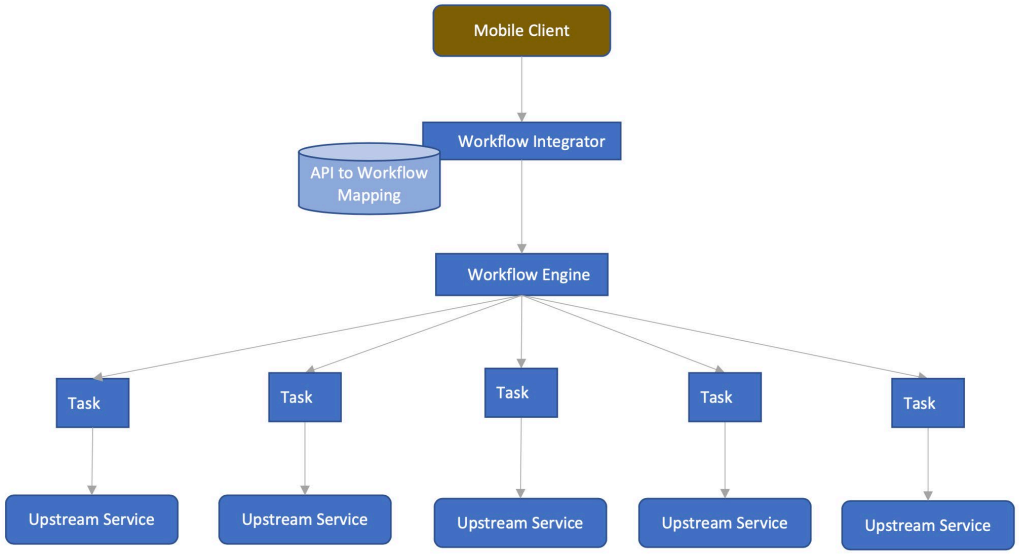

Figure 3 - Workflow Orchestration Architecture Implementation

4.2. Benefits

The workflow orchestration architecture has delivered handily on its promise. It is currently in use for both COAM and MDU onboarding workflows. Here we examine this benefits in detail.

4.2.1. Metrics and observability

With the modular units in our new architecture, logs are emitted to their own buckets in our log aggregator, making it easy to aggregate logs for a given task, or get details of how a given task is performing. Additionally, the workflow engine itself produces a wealth of data about the workflows it has executed. These metrics have been collected and exposed via dashboards in various observability tools, supplying great insight into workflow executions, and allowing us to find and address issues promptly.

4.2.2. Reusability

Separating orchestration from business logic implementation in the services has helped us reuse the same tasks for different onboarding experiences. For example, most onboarding experiences would involve associating cloud-based configuration with a gateway device. Since each task is concerned only with its specific business logic and does not have other dependent services, we can use the same configuration-to-device-association task in multiple workflows without the need for duplicated efforts.

4.2.3. Development agility

Given that we can reuse tasks very efficiently, writing new workflows, in the best-case scenario, has been reduced to writing a new definition file with references to the existing tasks that have already been developed and deployed. This makes it easy to update an existing workflow. As an example, if an existing workflow needs to change the order of the tasks it executes, the change is limited to changing a workflow definition file and deploying it to production with no code change involved. Conversely, shared tasks can be updated to implement some universal change, like a new endpoint or authentication mechanism for an upstream service or a tweak in the business logic without having to modify the workflow definition.

4.2.4. Ease of programmatic client integration

The clients who call our platform API which today include several back-office processes, and web interfaces in addition to the original mobile application client, have an easy intuitive point of integration. All workflows are initiated through API endpoints exposed by the workflow integrator and specified using well understood open-sourced standards like OpenAPI. We are able to generate the client code needed to interact with the platform API thereby eliminating a significant chunk of work the client development organization would otherwise need to undertake. Additionally, since clients do not interact with the workflow engine directly, workflows can evolve independently of and transparently to the clients so long as the responses provided or inputs required don't change. This allows for the tasks to add more features and functionality without impacting clients. If the contract between the workflow integrator and client application stays intact, there is no change needed on the client application and hence no updated version of the app to be released. Mobile application evolution is complicated by the fact that customers we wish to support may still be using older versions of the application. Being able to drive down new features to customers without a client application update helps us provide a more frictionless experience.

4.2.5. State management

With the new workflow-based architecture, the workflow service tracks the state of the workflow as it orchestrates the calls between multiple tasks. This allows clients to resume from the last successful task execution in an earlier attempt; customers need not start the entire flow from the beginning and repeat work they already completed. With the modular breakup in the new architecture, customers can resume from the place they had left off in their previous attempt and not repeat the steps that they had already done.

5. Case Study: Mesh and Advanced Security Firmware Agent Enablement

After a comprehensive discussion of the evolution of onboarding processes and the benefits of the workflow orchestration architecture, we look at a specific example of how the move from one paradigm to the other improved an important business process. The onboarding process involves the enablement of firmware agents to support certain valuable features of our wi-fi product, specifically mesh networking and advanced security. In our initial leased gateway onboarding, if conditions were met indicating the need for agent enablement, this would be tried as an optional part of the onboarding orchestration. It was optional in the sense that should these task executions fail, these failures weren't reported as such to the client, but rather as warning that these portions of the workflow had not completed successfully. Because of the single orchestration, it was not possible to apply different retry semantics to this conditional agent enablement or to allow customers to resume the workflow at these optional tasks so that any failures were left to be dealt with by other external systems outside the context of onboarding. This was expedient in that it allowed customers to accomplish their primary goal of getting access to their HSD service and allowed any trouble in the ancillary configurations to be dealt with independently without requiring action on the part of the customer or delaying their use of the HSD service. Figure 4 illustrates this process.

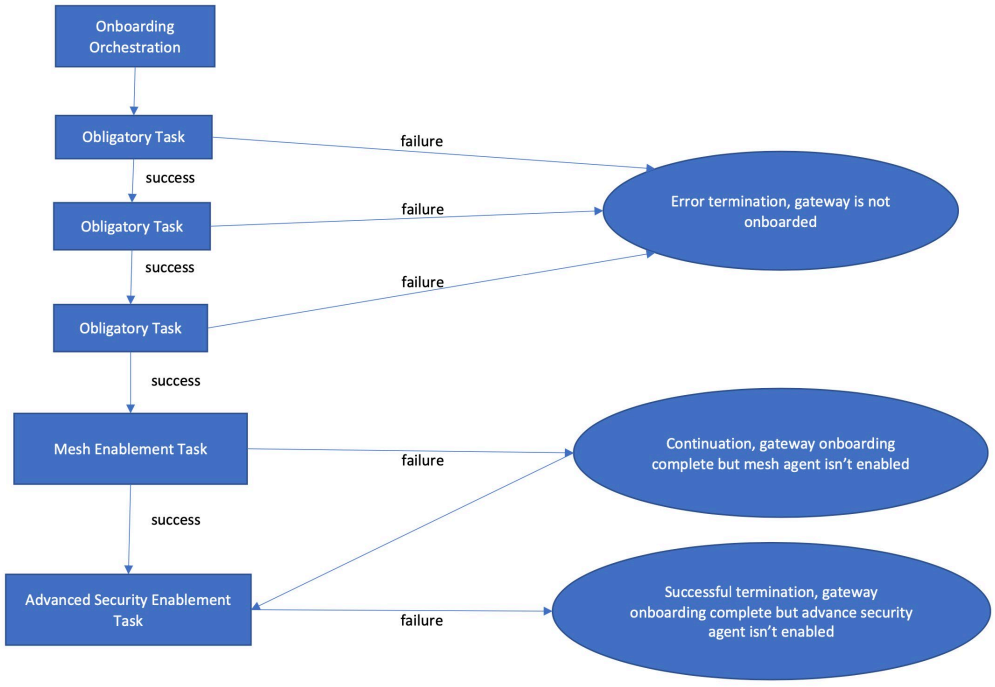

Figure 4 – Conditional Agent Enablement in the Monolithic Orchestration

The workflow orchestration architecture allows us to recognize the conditional enablement of agents as independent workflows, with their own retry semantics and definitions of success or failure. This still accomplishes the primary objective of avoiding a dependency on ancillary configuration before customers

can use their HSD service. The workflow orchestration architecture however, allows us to intelligently handle failures with the ancillary configurations and address them without reliance on external services. When the primary onboarding workflow has successfully completed, it produces an event to a message bus which is then used to trigger the subsequent workflows to perform the agent enablement; the primary workflow is completely decoupled from the subsequent workflows that perform the agent enablement. Each can report success or failure on their own terms, and each can employ their own retry semantics. We can define appropriate retry semantics and ensure the enablement completes successfully. Further decoupling is achieved by inserting a message bus between the primary workflow and the agent enablement workflows. The primary workflow does not initiate the subsequent workflows directly. Each workflow has an initializer component that listens for events on the bus and initiates the workflow in response. This allows for externalized retries in addition to the ones defined for the workflow itself. This is particularly useful in a world where gateway devices may be activated prior to shipment to customers, but agent enablement requires the gateway device to be present on the network. The independence of these agent enablement workflows has also allowed us to introduce incremental improvements to them while leaving the primary workflow untouched. Figure 5 illustrates these improvements facilitated by the workflow orchestration architecture. Using the new workflow orchestration architecture for mesh and advanced security firmware agent enablement has resulted in better than 99.8% success for each of these processes.

Using the new workflow orchestration architecture for mesh and advanced security firmware agent enablement has resulted in better than 99.8% success for each of these processes.

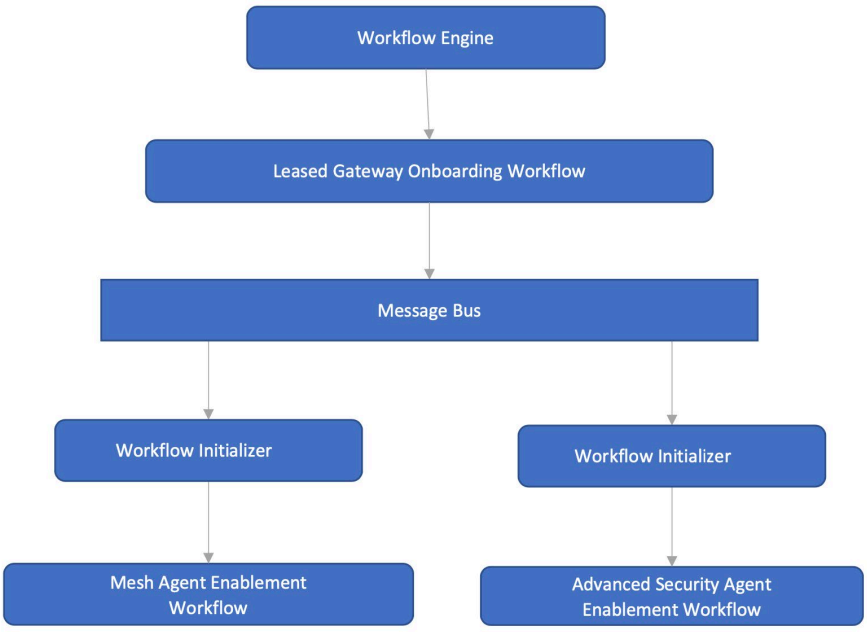

Figure 5 – Agent Enablement with the Workflow Orchestration Architecture

6. Unified Client Interface

As the onboarding process evolved, client interfaces needed to evolve along with them and means of interaction with the platform in presenting the onboarding experience to customers varied. Because of the nature of business process and software evolution, the differing onboarding processes discussed in this paper have and will continue to live concurrently for some time. This diversity in client interfaces is exacerbated by the independent evolution of business and product requirements for different channels, such as mobile clients used by customers versus the web interface used by technicians, or the tools employed by care agents to aid customers with onboarding trouble. Two disparate activation and onboarding platforms have consequently emerged, with some level of interdependency. As new products and devices are introduced, each channel needs to be modified to satisfy the latest requirements. While this has allowed each channel to deliver the appropriate experience, the workflow orchestration architecture gives us the chance to improve this situation. A single client interface for all activation and onboarding needs across all products and devices would be preferable. Having different platforms also increases the potential for customer experience inconsistencies and variance in how the different channels achieve the onboarding process. Figure 6 depicts the crisscrossing interactions that result from the current path.

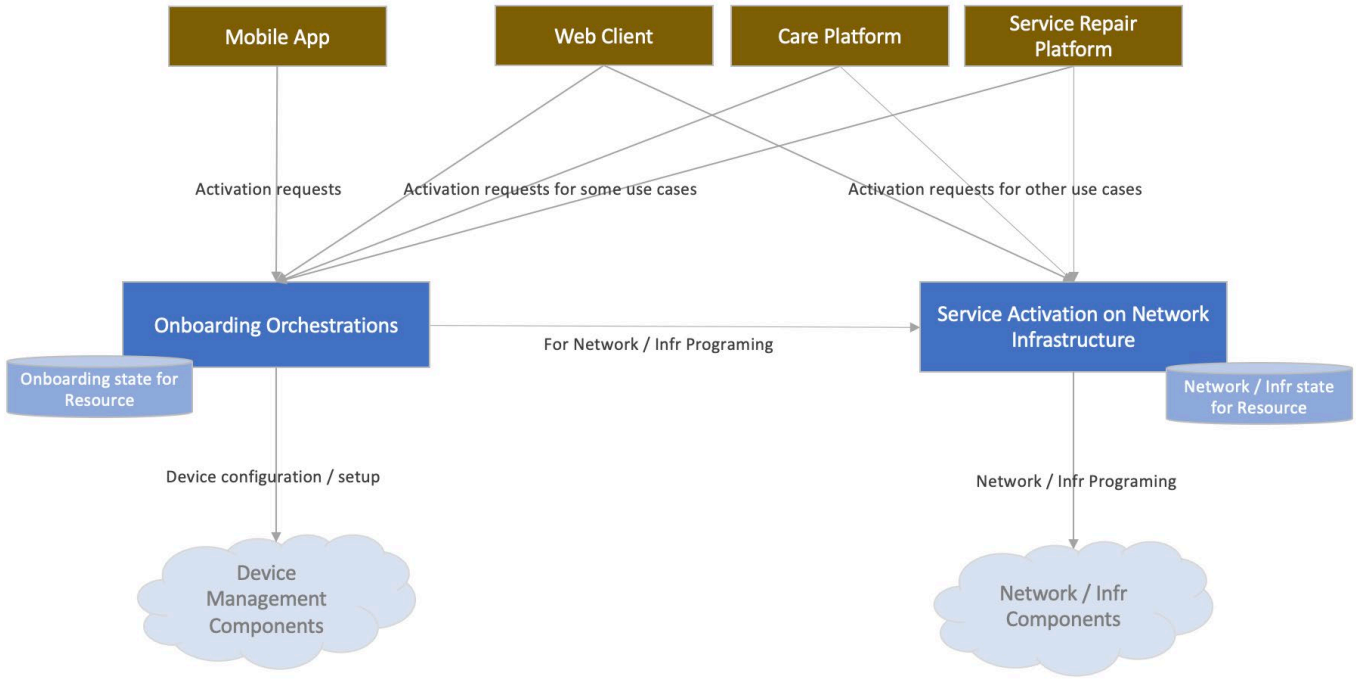

Figure 6 – Channel, Product Specific Client Onboarding Interfaces

To realize the opportunity that now presents itself, the teams responsible for these two platforms have carefully crafted a plan to launch one onboarding hub for all products and devices that can be used by all channels. While offering unified client interfaces, the hub will allow for channel-specific onboarding

considerations while relying on single components for common functionality. Figure 7 illustrates how the divergent platforms coalesce to provide the desired common client interface.

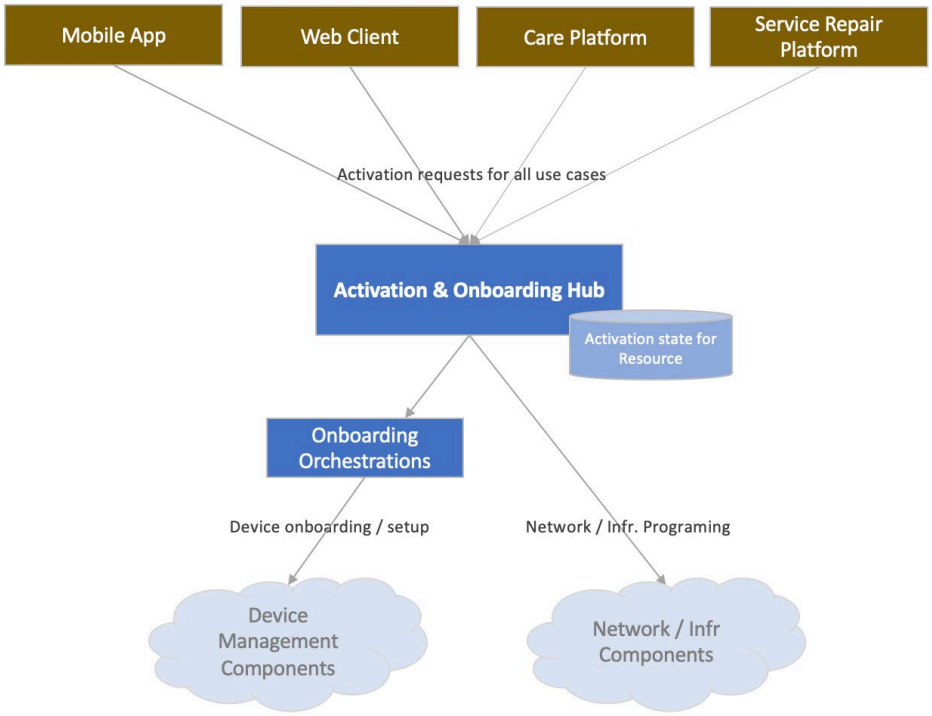

Figure 7 – One Onboarding Hub for all Channels and Products

7. Conclusion

The workflow orchestration architecture we have introduced for onboarding processes has paid great dividends. We have gained great insight into the functioning of workflows and the tasks of which they are comprised. We have a library of discrete functional units that are used to compose new workflows as required. We can quickly and easily modify these functional units apart from the workflow definition itself and vice versa. Workflow specifications can be read and understood apart from the code. We can implement different functional units in whatever programming language is most appropriate for the specific functionality they provide. All of this is helping us to deliver the best possible FTUE to our customers. We have seen steep improvements in onboarding success for COAM customers who can now leverage the Xfiniy App for onboarding. The process is continually evolving but we have the proper tools at our disposal to ensure the onramp to the network shines as brilliantly as possible.

Abbreviations

API	application programming interface
COAM	customer owned and managed
DSL	domain specific language
FTUE	first-time user experience
HTTP	Hypertext Transfer Protocol
IP	Internet Protocol
IVR	interactive voice response
JSON	JavaScript Object Notation
MDU	multi dwelling unit
Wi-Fi	wireless-fidelity