Whitepaper

April 2024

Experience-First Automation

Achieving experience-first networks using intent

Author: Grant Lenahan

In partnership with

Published by Appledore Research LLC • 44 Summer Street, Dover, NH. 03820

Tel: +1 603 969 2125 • Email: info@appledorerg.com• www.appledoreresearch.com

Original Publish date: 04-Apr-24

Cover image: Bing image creator/Robert Curran

CONTENTS

Experience-first automation: the solution to real business demands

Myriad business pressures drive all industries toward constant cost and agility improvements. Telecom is at a unique crossroads, where technological change and new opportunities, together with the digitization and cloudification of business, make automation both more feasible and more valuable than in times past.

Appledore has published several documents that demonstrate how the industry can increase its TAM by well over USD $200B and realistically capture almost half of those—but only if it adopts the most efficient principles of orchestration and operations. Most of the high-profile objectives (and services, such as digital ecosystems, 5G slices, and network-delivered security) all demand *automation* to achieve cost savings, *intent* to achieve simplicity and efficiency, and *advanced assurance methods* to ensure customer satisfaction. Moreover, the cost of power (and the associated cost of HVAC) is becoming one of the largest operational expenditures. In addition to being ecologically and politically important, intelligent automation is the only practical way to optimize the power-performance balance. These are not just nice-to-have technical features; they are non-negotiable enablers of revenue and margin growth.

This white paper will examine the business needs, operational promise, and **best practices for automating** large networks, both public (CSPs) and enterprises. In particular we want to zero-in on the importance of an often used but rarely understood term: **intent**. We will show how intent is essential for an efficient and flexible closed control loop. We will further assert that the goal of assurance and observability is, and always has been, to determine if intent is being achieved, and if not, why not? In many ways intent definitions and service assurance objectives are identical. We believe that leveraging this realization leads to several elegant points of integration that should be universally followed.

We will focus specifically on the IP networking[1] domain, both as a way to focus on something concrete and because it represents the focus for Juniper Paragon. Representative use cases will be stepped through to fully illustrate how theory translates to reality. We will conclude by discussing Juniper's newest Paragon Automation solution and how it incorporates these best practices.

Automation, if not specifically via intent and control theory, has been tried before, with limited success. This leads us to two important learnings we hope to reinforce:

1. Past automation (examples: legacy provisioning, early FAT-VM VNFs) has been half-hearted and implemented poorly (as complex, one-off, imperative spaghetti code). We must pay as much attention to *how?* (doing it properly) as *what?* (achieving one-off automation). Intent is critical to this success.
2. The time to implement intent is now. For the first time we have a growing critical mass of technologies that have been designed with automation in mind, using software and remote

[1] IP networking is its own domain yet must be carefully linked to the optical domain. In order to balance flows across optical paths and maintain HA (diverse paths), the IP controller must understand the optical topology in near real time. Juniper does.

control. Cloud-native CNFs, SDN (and SDN-like) enabled routers, 5G, and disaggregation, especially Open RAN, (to name a few) change the game. Specifically, they increase the scope of possible control, improve the scalability of controllable deployments, and they coincide with external opportunities drivers such as digital business/supply chains, and cloud delivery of infrastructure.

In summary—the time is now, the market need is now, and the technology is finally mature. If we get it right, we can break the cycle of costly, brittle, and expensive-to-maintain operations software.

Throughout the document we will link published research that allows readers to dive more deeply into topics we introduce.

Importance of the end-to-end IP networking domain

This paper will focus on the end-to-end IP Transport domain and the best practices for its automation. We will apply generic principles of automation and control theory, specifically to this domain (Layers 2&3).

The objective of IP domain automation, at its most basic, is to accept guiding parameters from a service order and fully automate the setup, testing, ongoing monitoring, and, when needed and possible, repairs to this service throughout its lifecycle. The related objective is for the automation system to make intelligent trade-offs and select the best solution, considering what may be conflicting inputs. Note that it must also choose, in real-time, from realistic options, considering things like capacity, availability, physical diversity, and congestion.

Moving information between two or more geographically distributed points is what telecom networks do. Unlike most other industries, and especially unlike hyperscale clouds, communications networks typically reach out to hundreds or even millions of endpoints, creating very different economics when compared to public cloud or aaS giants that have the luxury of consolidation into a few datacenters of huge scale.

Initially analog, the transformation of networks to digital is complete. Initially wired, networks are now composed of optical transport, carrying L2/L3 data either to LAN end points, cellular endpoints, or both. As we move from earlier technologies to 4G, 5G and 6G we are observing cell sizes decrease dramatically, which means many more cells, and a great deal more backhaul, mid-haul and front-haul.

All this boils down to an incredibly important but basic fact: the hybrid IP/optical domain is a ubiquitous building block of consumer access networks, enterprise broadband networks, the broadband core, and the cellular RAN—minus what will eventually be the last few hundred feet. It extends to millions of small end-clusters, cells, or LANs. It is also the fundamental building block across all services—existing and likely into the foreseeable future.

The centrality of the hybrid IP/optical domain to xHaul and network slicing increases the scope, complexity, and penetration of IP/optical networks dramatically. It also means that automation of the IP/optical domain is a fundamental building block for all other network deployments. It pays to get automation right since everything will depend on it, and it will be re-used repeatedly in complex network services – especially in 5G, for xHaul. It is a single foundation for nearly everything.

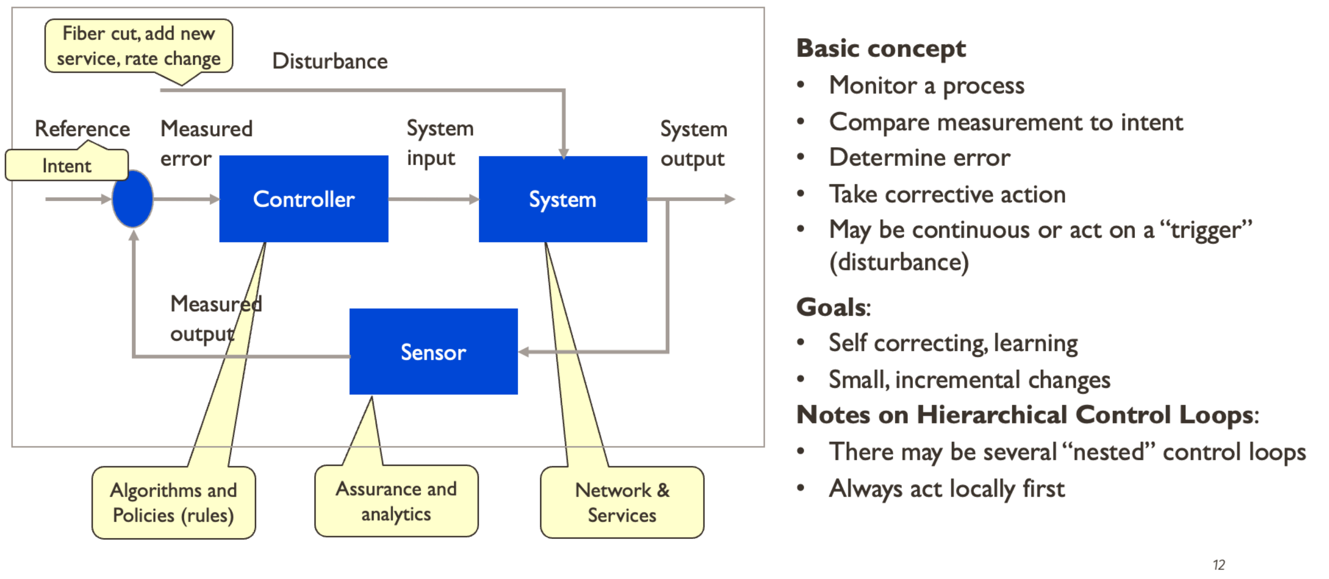
As an industry, we have decades of end-to-end thinking, but we must break free of such monolithic concepts, and achieve end-to-end networks and services through more modern and widely accepted principles. Such principles include self-managing technologies, abstraction of domains (including tech, regional, and administrative), and what is often referred to as domain-driven design. These principles are universally endorsed by organizations such as the CNCF, the TM Forum, and ETSI/3GPP. Treating each domain, in this case IP transport, as a self-managing island is not wrong; it is precisely right—so long as that island exposes real-time, intent-driven, abstracted APIs northbound.

Not all Automation is equal. "How" matters, and best practices are well-established

Most networking suppliers claim to have an automated solution for IP transport networks. And many do. But we want to underscore what Appledore has preached since 2016: *not all methods of automation are created equal*. There are well-established best practices for automation, and for closed control loops, that have been documented academically, and proven in many industries—from micro-electronics, to ballistics, rocket science, and robotics. The blueprint for all closed loops lies in control theory. Similarly, there are strong advocates with more specific proofs closer to our industry—in the recommendations of the CNCF, the TM Forum, and ETSI. Possibly the closest example of success is cloud-native in the hyperscale public cloud and its attendant SDN implementations.

The diagram below is widely used in control theory. It represents the generic concept of a control loop and can apply equally well to robotics, ballistics, or rocket guidance. We have overlaid network terms for context. Its simplicity (at least in concept) should be evident—there is a *single* loop, and a *single* reference for correctness. That reference is what we call **intent**. The controller is a combination of orchestration, intent-based algorithms, and aspects of AIOps. The corollary to having a single reference is that it can be used to constantly evaluate if the system is performing as intended. Moreover, given the abstract definition of intent, there is sufficient flexibility that *the same loop logic* can relocate or heal a system (e.g., an IP service or, in cloud-native, a workload or multi-workload application).

Figure 1: A Basic Control Loop Functional Diagram. Telecom terminology overlay



Basic concept
- Monitor a process
- Compare measurement to intent
- Determine error
- Take corrective action
- May be continuous or act on a "trigger" (disturbance)

Goals:
- Self correcting, learning
- Small, incremental changes

Notes on Hierarchical Control Loops:
- There may be several "nested" control loops
- Always act locally first

Source: Appledore Research

The best implementations pay dividends in the short-term and over their decades-long lives. Done right, control loops and automation simplify the exploding complexity in OSS stacks. Careful system design makes them elegant in their simplicity. Benefits include those listed in the table below.

Benefit	Elaboration
Agility	Rapid, error free network configuration and service fulfilment.
Simplifies automation and self-healing	Intent allows the system to use the ***original orchestration method*** (algorithms, configuration logic) to make ongoing corrections. It provides the flexibility to make such changes and allows the same algorithms to perform healing without specific healing logic.
Reduce ongoing integration costs	Abstraction of self-managing domains means that integration points are unaffected by internal updates and improvements (unless changes demand different input). Intent means that detailed implementation changes don't affect other systems.
Reduce ongoing maintenance costs	Similar benefits as accrue to integration.
New savings, benefits	Power-savings and other costs demand constant state management, which is ***only cost and operationally feasible at scale with closed loop automation***—and these costs are rising. Moreover, intent gives a construct that can be used for any desired outcome or constraint, including future innovations.
Cheaply extendable (technology, services, vendors)	Model (YANG, other) driven implementation of intent and logic means that new services and products are defined (and therefore may be added to the system) with new models on-the-fly. The rest of the orchestration and solution-finding algorithms (intent compiler) remain unchanged.

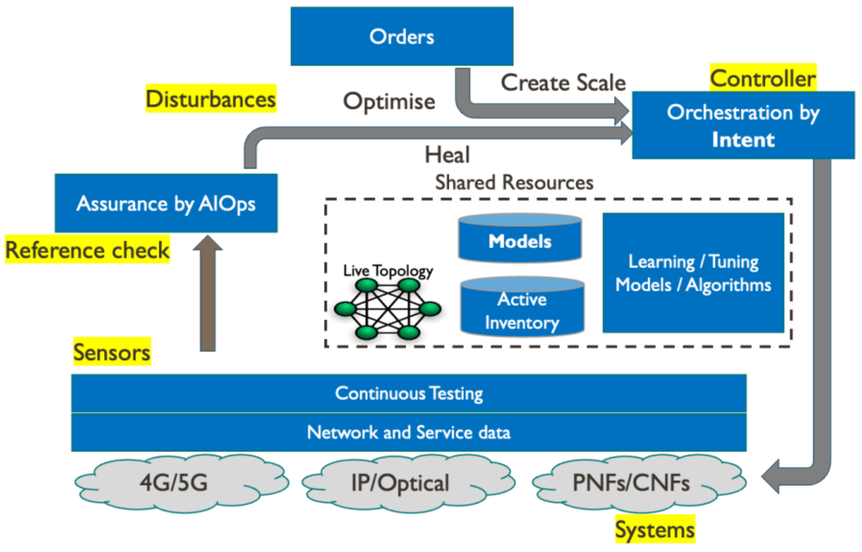
The principles of control theory may be simple, but they are easy to get wrong if one ignores the basic principles. Witness our recent past: over the past decade, telecom struggled to introduce virtualization, automation, and closed loops. The early versions (MANO-driven) proved unsatisfying. They worked poorly, slowly, and did not deliver the intended agility. The reasons were simple to an outside observer: the telecom industry did not implement proven best practices. After years of clunky half-measures ("Fat VMs",[2] anyone?), the industry has reset and begun adopting cloud-native, an ongoing process that also demands continuous test and incremental upgrade (CI/CD/CT) methods. Those same principles (testing at various stages, differential analysis, and changes) are essential to most automation loops.

Next, we will focus on those best practices (specifically applied to telecom), and in particular on one best practice that is foundational to many other best practices: the elusive concept of **intent**, and intent-based orchestration and networking. Interested readers can look for our upcoming Market Outlook research report: *Principles of Successful Automation*.

Theory applied to reality: Best practices in an IP transport domain control loop

The diagram below is Appledore's telecom-specific adaptation of a control loop. This diagram comes directly from our published research. It calls out best practices and emphasizes the fact that these best practices are **interdependent and form a system;** one cannot pick and choose and expect success. Our diagram uses terms that ought to be familiar to telecom network management professionals.

Figure 2: Best Practices for Closed Loop Automation, tailored to telecom environments.



Key points:

- One loop (one set of algorithms) replaces 3 stacks for Create, Heal, Scale/Optimize, based on common data and methods (logic)
- Process includes, and takes feeds from, observability and assurance
- Several loops exist
- Leverage intelligence from AIOps / Analytics / Continuous testing in addition to basic fault data

Source: Appledore Research

[2] For those who are not familiar with the term, "Fat VMs" was shorthand for a single, large virtual machine that had in effect a hand crafted VNF. No intent existed. Healing meant re-implementing the entire (huge) network application. Scaling meant replacing it with a smaller or larger hand-designed version. Typically, these were images.

One loop, yet many . . . say what?

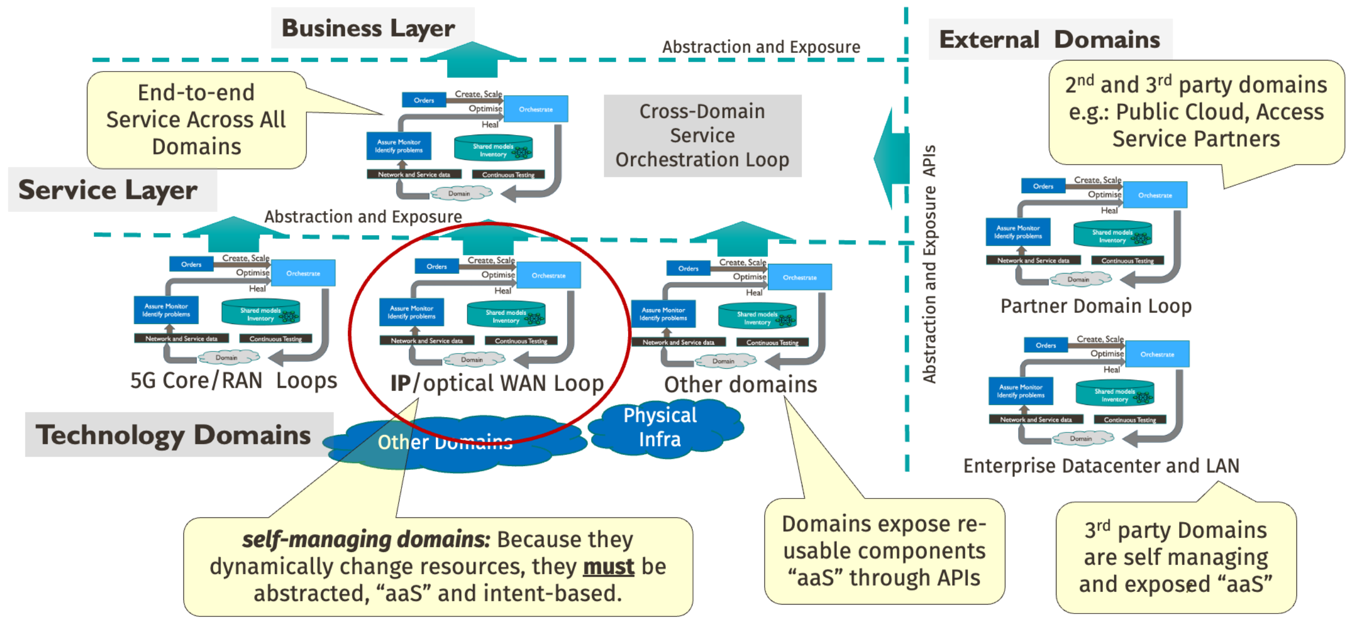
There can be an apparent contradiction between the principle of having one loop across all business processes, but many loops in a network. However, both are true (see figure 3, below). In most cases, there will be an automation loop in every technology domain and another loop at the cross-domain service orchestration layer that brings them together. Each of those loops replaces three business processes that were previously siloed: fulfillment, assurance, and capacity management. In the modern world, these are all orchestrated by the same entity and are called configuration, healing, and scaling—to use the cloud-native terminology popularized by the CNCF.

A single loop for fulfilment, healing and scaling. One of the most critical best practices, and one that clearly demands thinking differently, is that a *single orchestration method* must be used for the three major life cycle processes of fulfillment, assurance/healing, and capacity expansion/scaling. There can be absolutely **no difference** between the logic that creates a service initially, heals it in the case of failure, or scales the service as necessary. In effect, healing is re-fulfilling a service, but with the failed component removed from the solution set. Scaling is re-fulfilling as well, with larger input parameters. The consequent advantage is simplicity and ease of future maintenance. It further means that the results of AI or other sensor input can be introduced as intelligence, without modification.

Independent, self-managing domains. A second critical principle is that the control loop is not monolithic; rather each domain—typically a technology specific domain—is an independent self-managing entity. This allows experts in radio to design radio control algorithms and experts in IP to implement IP control algorithms. The models and algorithms will be necessarily different. And yet ultimately, this is very much an *end-to-end* approach, with cross domain service orchestration linking domains together as necessary to effect an end-to-end service. Each self-managing domain must expose northbound as-a-Service APIs. Why is this critical? Because the definition of that service is intent; and by defining it as an abstracted service, flexibility is left for the domain to heal scale and fulfill independently.

One exception: because the IP transport domain is so foundational to telecom, sometimes an entire (fixed broadband) service may be represented within the IP/optical domain.

Figure 3: Many domains exist. Each must be a self-managing loop and implement intent


Business Layer
Abstraction and Exposure
External Domains
End-to-end Service Across All Domains
Cross-Domain Service Orchestration Loop
2nd and 3rd party domains e.g.: Public Cloud, Access Service Partners
Service Layer
Abstraction and Exposure
Abstraction and Exposure APIs
Partner Domain Loop
5G Core/RAN Loops
IP/optical WAN Loop
Other domains
Technology Domains
Other Domains
Physical Infra
Enterprise Datacenter and LAN
self-managing domains: Because they dynamically change resources, they must be abstracted, "aaS" and intent-based.
Domains expose re-usable components "aaS" through APIs
3rd party Domains are self managing and exposed "aaS"
Orders
Create, Scale
Optimise
Heal
Orchestrate
Assure Monitor Identify problems
Shared models Inventory
Network and Service data
Continuous Testing
Domain

Source: Appledore Research

Intent. At the core of this loop, and difficult to illustrate, is intent. Otherwise known as declarative logic, the input (order) to intent-based orchestration is abstracted to SLA parameters, exposed over a service-oriented API. This input is the intent and, in effect, constrains the implementation to choices that meet that intent. Within the orchestrator, intent algorithms differ from traditional imperative logic. Rather than execute a pre-determined implementation, intent-driven automation has the flexibility to allow technology-savvy[3] algorithms to select any location, resources, etcetera that can fulfill the desired intent. This flexibility is critically important in the long-term elegance, maintainability, and viability of the control loop. Specifically, it allows healing and scaling the flexibility to select an alternative implementation to circumvent a failure or meet increased capacity demand. Well-crafted intent logic is equally capable of setting up a service and healing it. Interested readers can dive into our twin market outlooks on intent: here and here.

Juniper is a strong proponent of intent. Kireeti Kompella, Juniper's SVP of architecture for Automated WAN, often speaks publicly of its importance, and of the importance of an ***intent compiler****, which productizes the types of algorithms we call for here and in the linked papers. His most recent talk may be seen here: keynote at AutoCon 2023.*

[3] One of the reasons that we believe that each technology domain requires its own dedicated control loop in software is that the models and algorithms are necessarily technology specific. The model of a radio and of an IP service will look very different. Similarly deep expertise in radio and deep expertise in IP (just to pick two prominent examples) typically exist in different organizations—even firms.

Model-Driven. In order for algorithms to be capable of handling a wide range of inputs,[4] data must be divorced from logic, yet defined in such a way that the logic/algorithms can interpret and implement it. The structure of this data is the model. A service model, for example, defines the intent parameters necessary for the algorithms to create a particular instance of that service. Implemented well, any service may be represented with the (extensible) model. Modeling avoids the complex changes necessary were they combined together in spaghetti code. Below, we present a simplified illustrative example of a transport service defined as intent. The reader should observe that these are the kinds of parameters typically captured in a service order and stored in a catalog. We present an example in the section specifically on intent below. IP transport services are typically modeled in YANG.

AIOps identifies deviations from intent, along with the root problem. In a control loop, assurance and AIOps have a different task: they identify existing or impending troubles, clarify the root cause, and present that data in a simple form that the intent-based orchestration method may rectify. AIOps becomes the intelligence regarding soft and hard troubles. In Figure 1, assurance/AIOps form the sensor and are the portion of the controller that identifies the underlying problem that must be corrected through healing or scaling.

You cannot pick one or two best practices: Collectively, they form a system

Despite having made a list of best practices, a careful look shows that **they are not independent**. In fact, it is awkward to implement any single attribute without the others. The single process vastly simplifies the ability for the control loop to operate and heal without additional code or diverging logic. Its guidance—whether to create or heal a service—is intent. Intent is specified in the model; and by separating data from logic allows the process to be easily extended to new services and types (no new logic or algorithms required, just a mode). Finally, domains must be independent in order to self-manage.

Intent and autonomous WANs managed by closed loops represent a major break with past norms for operations. However, executed collectively, these **best practices simplify the overall system**. This reduces the complexity of the code and its long-term maintenance, which has long been a burden in telecom.

Next, we will look inside intent and understand why it is powerful, and how to define it at the ideal balance between too much detail and too little specificity. The goldilocks of intent.

[4] Note that algorithms can only handle the data types that they are intended to. This is one reason why domain managers are often technology-specific. It is also why designers must maintain abroad view of what may be requested.

Intent and how to get it right.

Intent is a radical change to how NMS/OSS and automation are implemented.

Oversimplified, traditional methods specified branching logic and how a service or device (such as ports) should be configured. If a new service was added, new logic was required. In most cases, input to the process was very explicit, leaving no wiggle-room to select a different solution[5]. This was not considered inflexible, but rather reliable and predictable.

However, two main problems arose from these methods. First, they generated never-ending streams of maintenance and integration actions since nothing was truly abstracted; second, they lacked the flexibility to deal with a fault without starting over. Consequently, there was little to no ability to self-manage or self-maintain.

The **purpose of intent** is to define a satisfactory end-state with sufficient flexibility that intelligent algorithms, along with the truly up-to-date inventory, can make decisions during implementation and throughout the lifecycle of a service. We will provide examples that show why this is valuable later in this paper. A second benefit of this flexibility is that in the case of congestion or failure, a new implementation solution may be found and implemented automatically. This is called **self-healing**. A similar process exists to expand capacity, called **scaling**. Hopefully you can see that intent is critical to autonomy and self-management. You should also begin to see why a single orchestration method (or set of algorithms) is necessary for fulfillment, healing, and scaling. It all depends on intent.

Intent can be defined in a useful fashion or one that is less useful. Largely this comes down to the level of abstraction and, therefore, flexibility allowed. Let's illustrate this by considering extremes, neither of which is desirable. At the lowest level of abstraction, the intent specifies the output explicitly, which means in practice it is not much different to today's working practices. At the highest level of abstraction, intent provides only vague guidance to algorithms—which are typically sets of rules that the solution must meet.

Somewhere in the middle, we find a balance between flexibility and specificity. Fortunately, this almost always corresponds to the parameters specified in a good service definition/SLA. For the purposes of IP and optical transport services, the service definition (and therefore the *intent* specification) will certainly include parameters such as point of origin, point of termination, capacity, and quality metrics such as latency and BER. This leaves algorithms plenty of flexibility to choose between different implementation technologies and different paths.

Intent can also include less traditional but increasingly important parameters. Examples include geographical limitations such as prohibiting data from leaving the EU, cost objectives, and power efficiency objectives. With power becoming the largest cost in datacenters, this matters financially and environmentally.

[5] While it was never referred to as intent there were a few applications that did implement it—primarily path computing engines for IP and ATM, some of which existed long before the invention of SDN.

Converting intent into reality: algorithms and orchestration

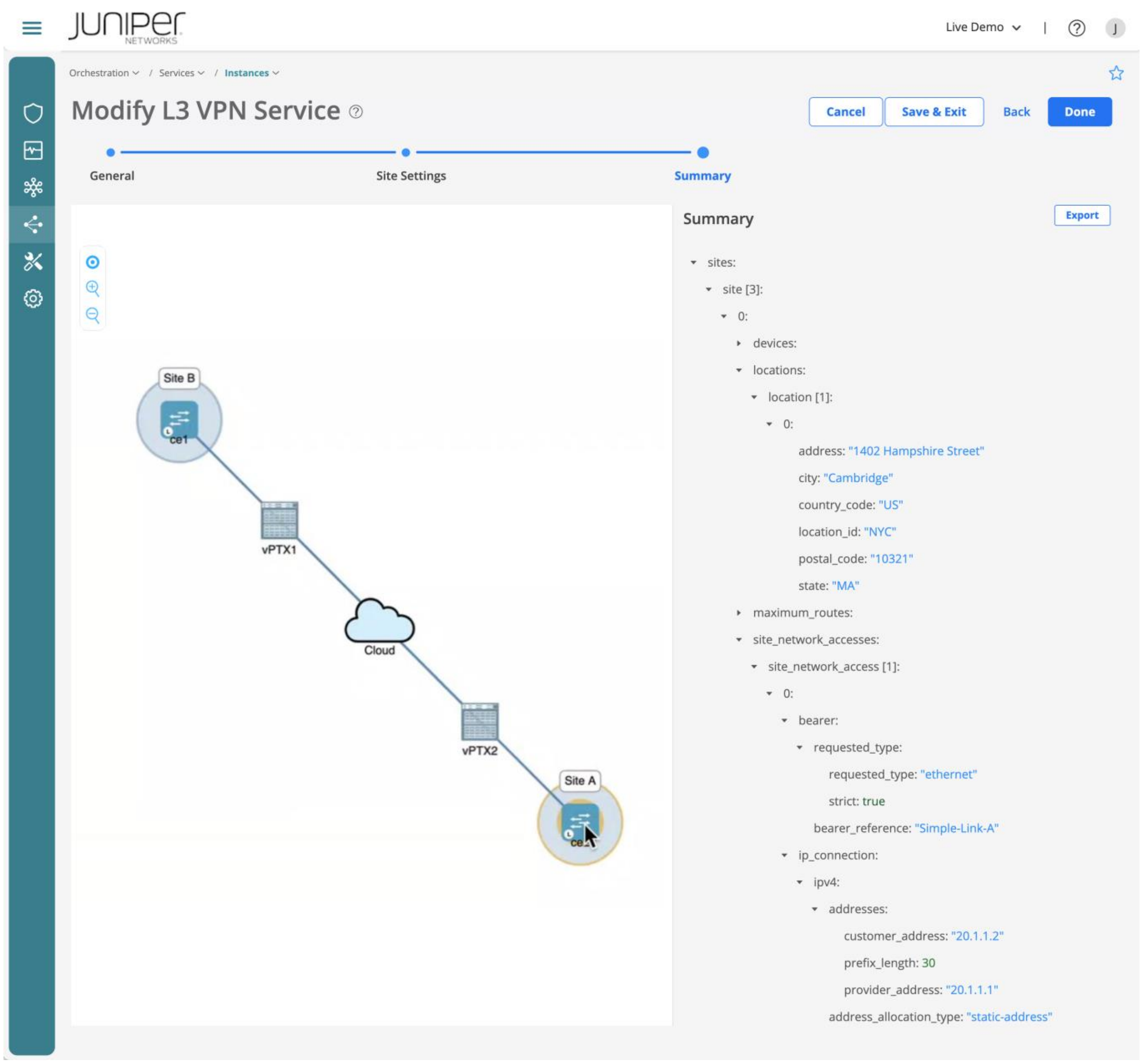
By capturing intent, we bound (i.e., put boundaries on) the solution and provide guidance to the algorithms (sometimes called the intent compiler) that must create a real instantiable service from these intents. Now let's turn our attention to the algorithms—the method by which we will perform this translation from general to specific.

The goal of these algorithms is essentially to understand the inventory of possible solutions (both physical inventory and configuration options) and to select from available inventory and options a solution that best meets all the intent requirements. This is a classic case of linear programming, and most commonly achieved through a combination of searching and rules. The first search and rule might eliminate all paths that do not begin at the specified origin. Subsequent rules could specify objectives for the termination point, speed, cost, latency, and power efficiency. Hopefully after progressing through this sieve, the remaining solutions fit all objectives. Note: having more than one solution means that if needed, a path may be relocated—for example, to avoid a failed port, fiber cut, or congestion point.

Intent definitions, along with rule sets can also be used to effect tradeoffs. One example that Juniper emphasizes is the ability to prioritize power-saving technologies. To make this more concrete, a user could choose to effect sleep on certain paths and ports, so long as the solution continues to meet latency and throughput goals—even if the margin is less generous. Alternatively, the user could choose for some high-performance apps not to allow such tradeoffs. Finally, another benefit of intent and automated healing: should traffic change such that the power-saving is impacting the latency or throughput SLA, intent would be violated, resulting in power-saving sleep being automatically turned off until the underlying congestion passed. This example underscores the many unanticipated potential use cases that could be automated. Appledore firmly believes that hundreds of similar examples will arise over the life of a system, proving the business and operational worth of good basic design.

Since even complex services are made up of many smaller, simpler, and common components (locations, path segments, speeds), well-designed intent algorithms can implement almost any service that may be modeled from those defined components and parameters. A new service may have more components, but if it has the same underlying parameters, it may be satisfied without any changes to the algorithms. Flexibility and extensibility are achieved.

Below is an illustrative example of a service model, courtesy of Juniper Networks. The germane points are that the model defines relatively high-level needs – the ingress and egress points, service type requested, and performance parameters. Importantly, models can be extended to support additional constraints and objectives that intent algorithms may satisfy.



Summary

```
sites:
  site [3]:
    0:
      devices:
      locations:
        location [1]:
          0:
            address: "1402 Hampshire Street"
            city: "Cambridge"
            country_code: "US"
            location_id: "NYC"
            postal_code: "10321"
            state: "MA"
      maximum_routes:
      site_network_accesses:
        site_network_access [1]:
          0:
            bearer:
              requested_type:
                requested_type: "ethernet"
                strict: true
              bearer_reference: "Simple-Link-A"
            ip_connection:
              ipv4:
                addresses:
                  customer_address: "20.1.1.2"
                  prefix_length: 30
                  provider_address: "20.1.1.1"
                address_allocation_type: "static-address"
```

Source: Juniper Networks

By inference, we can see that healing or scaling can also be implemented by the same algorithms and rules (essentially a re-implementation of intent). However, the healing/scaling process can be simplified if the orchestrator first performs a **differential analysis** so that *only the parts that must change* are changed. In other words, focus only on the components of the model that are affected. We can think of this as a form of continuous deployment—changing only one segment, port, etc. not the entire service.

Finally, the model structures, parameters, and algorithms are *technology specific* and demand intimate knowledge of a particular technology. This is one of the reasons why Appledore advocates independent automation per technology domain and further advocates implementation by specialists in that technology. These specialists need not be the vendor of the equipment, but it is likely helpful.

Intent and active assurance—perfect together

So, through the magic of intent-based orchestration customers get exactly what they want. Or do they? How do we know? Was the service set up as intended? Along the way did something change? Do the assumptions in the algorithms match the perception of the customer?

This is why **intent and advanced assurance** methods are perfect together. Let's begin by returning to the fundamental concept of intent: part of the service's intent is its performance, expressed as an internal or external SLA parameter. This SLA/intent parameter should automatically drive assurance logic. For example, approaching that parameter might trigger a caution, crossing that threshold might trigger healing. Similarly, approaching a threshold might trigger deeper (automated, if possible) investigation. The point is that when a service is conceived, assurance metrics ought to be conceived and placed in the service model (this is the origin of DevOps!). The service model should drive not just initial fulfilment but also program the sensor in the control theory logical flow (as shown in Figure 1). Nothing new needs to be specified—in fact, further specification can only create confusion and inconsistency.

Active assurance, as opposed to methods based on passively collecting performance and fault data, has some unique capabilities that can be exploited in a next generation solution. Specifically, active assurance allows operators to:

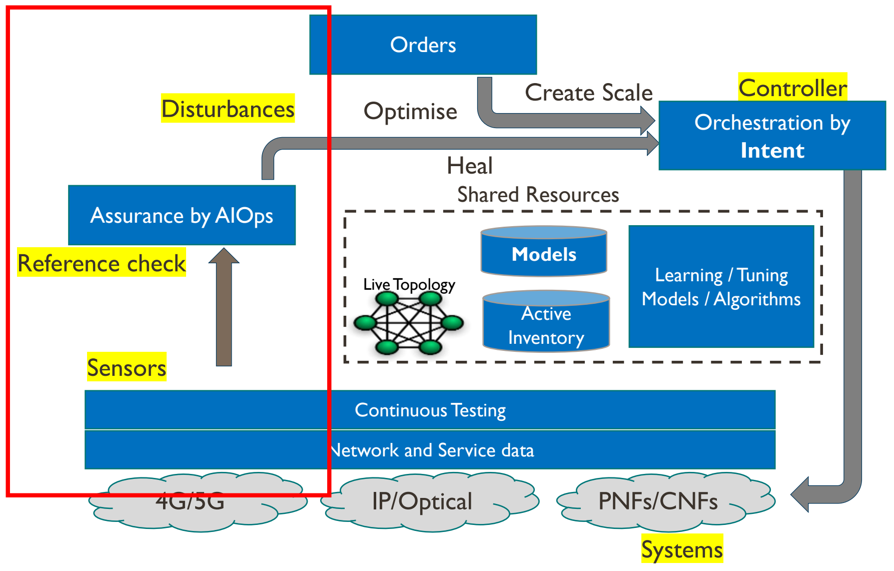
1. Perform tests on demand at any point in the service's lifecycle.
2. Run synthetic tests that closely simulate the user's experience.
3. Perform active tests at pre-determined points, both proactive and reactive.
4. Perform continuous testing during the in-service phase of a lifecycle.
5. Create targeted and interactive diagnosis scripts.

To get the most out of assurance, it should ideally be **integrated into lifecycle orchestration**. This enables network operators to prescribe testing at various points in that lifecycle. Such points include before service turn-up, or in reaction to a threshold crossing or other service level deterioration. One of the beauties that results from integrating active assurance with orchestration is that testing may be automated. This automation increases consistency, therefore lowering the chance of error, and also greatly reduces the cost of what would otherwise be labor-intensive work.

Figure 4 below highlights assurance and its role in the process, underscoring the role of service and resource models that drive not only orchestration but also aspects of assurance, threshold settings, etc.

Figure 4: Assurance provides critical intelligence to the closed loop

Assurance & Analytics Inform The Loop



Key points:

- **Healing** modifies the original service to avoid congestion or failure
- Assurance and analytics are responsible for identifying the disturbance, finding the root cause, and **providing this intelligence** to the orchestration engine
- The **very same intent algorithms** ("compiler") can now find a new solution that avoids the trouble.
- **Active assurance** makes this an integral and continuous part of the life cycle.

Source: Appledore Research

In the end, the concept of a single orchestration method depends on the integration with assurance and the recognition that assurance has a new objective. Assurance now informs and guides the actions taken by the closed loop. Specifically, it identifies intent violations and the root cause, in a format that allows the closed loop to act. We believe that intent-based closed loops both open the door to new assurance methods and in fact demand them. Solutions that explicitly include a provision for assurance, whether from the same firm, from a partner, or from a competitor, implicitly recognize this fact. It's an interesting historical note that we published our first paper advocating this new role for assurance in 2016.

Real-world application

A new architecture with new concepts and new terminology can be abstract. We will introduce an example to make this more concrete and (hopefully) easier to grasp. The first example is for a point-to-point private line, with tight QoS/latency, specified throughput, and a minimal number of path sections (low latency, better reliability). An added twist is that we specify whether power optimization should be applied, constrained by meeting other SLA parameters.

The specification, in human readable tabular form, might look like the table below. The first column is the parameter. The second is the value for that parameter which could be a number, a location, Boolean, etc. The third and fourth indicate if the values are a) shared with the orchestrator for fulfillment and b) loaded in the assurance system(s) for monitoring and future active assurance. Note also that while we have revealed certain germane characteristics such as the fact that it will be path-optimized, no specification by the customer is necessary because it is an integral feature of the service type.

Parameter	Value	Fulfilment	Assurance
Service Type:	ELINE05 Latency and Path optimized PL	Y	Y
Customer Name:	BigBank Ltd.	Y	Y
A endpoint:	Standardized network and/or geographic location	Y	Y
B endpoint:	Standardized network and/or geographic location	Y	Y
Throughput:	100Mbps minimum, guaranteed	Y	Y
Latency:	11mS Maximum, end to end	Y	Y
Path:	Automatically selected to maintain all chosen values	Y	Y
Data Sovereignty:	May not leave EU	Y	Y
Errored Seconds:	<X (note - inferred from the service type ordered)	Y	Y
Power-saving:	Priority so long as Latency and throughput are maintained	Y	Y

In the example above, these intents and related identifying information would be shared with both the orchestration engine and the assurance systems from a common model. It is worth noting that there will be two models: a generic model of the service (typically YANG) that defines the parameters and applies to all instances and an entry in the customer instance inventory that is as-built. Since SDN networks may be dynamically and automatically maintained, this instance inventory must be real time.

Initially, the orchestration engine finds facilities, such as ports and paths, that can serve the endpoints and support the technical specifications required. For the last mile, this sometimes requires construction. Otherwise, where physical facilities exist, the intent-based algorithms (compiler) and orchestrator sift through possibilities to meet all criteria and, in some cases, make the final selection based on customer directed priorities. An example of a customer priority is primacy of latency versus primacy of power-savings (sustainability), so long as the minimums for both are met.

One of the more complex choices the intent-based orchestrator must make is **path selection**. In this case a path must be selected such that the resulting latency will be well within the specification—implying path hop minimization. You can also see that the data must remain sovereign and, for regulatory reasons, is not permitted to leave a particular jurisdiction – e.g., the EU).

Once a solution is found, it is desirable to confirm that it performs as intended. We anticipate that as solutions mature, pre-engineered automated test scripts will be executed before the service is turned over to the customer and billing commences. Furthermore, these intents may be used to identify potential or actual SLA violations through either passive assurance or periodic active testing.

Scenarios like this offer multiple benefits and better ensure positive customer experiences. They are proactive, such that modifications are made to remain within the SLA before the customer experiences any impairment. Finally scarce and costly human resources are saved at two points in the process: first, by averting trouble calls to human operators; second, through the automation of troubleshooting and resolution.

Summary

We believe in the need for intent and the need to do it well. We believe a properly designed, intent-based system, will pay dividends. Such a system makes automation simpler. It allows for many objectives (latency, power, cost, etc.) to be met simultaneously and provides for simple integration to achieve automatic healing and scaling. There are also secondary benefits. For example, to be intent-based, one must abstract and expose each domain and each service component. This provides the impetus for more elegantly designed OSS, or NAS. This, in turn, vastly reduces integration and maintenance costs going forward.

Perhaps the most important outcome is changing the paradigm of network operations from a focus on custom crafting and hardcoding the details, to a new paradigm. This new paradigm consists of setting objectives and relying on software, algorithms, and, in the future, ML and AI to keep things running and learn how to optimize. This is the often-claimed promise of AI, yet without intent, AI is not free to make the changes it deems desirable.

It is worth noting that Juniper replaced a perfectly serviceable orchestration component with a new, custom-built intent engine and orchestrator. Moreover, they chose to emphasize the importance of intent, which is not always well understood and, therefore, not the first choice of marketeers. This choice underscores the importance they place on intent as a forward-looking foundation for automation.

We expect to see a steady stream of enhancements and improvements to the Juniper Paragon portfolio over the years building on this foundation.

In partnership with

Insight and analysis for telecom transformation.

@AppledoreVision

Appledore Research

www.appledoreresearch.com

info@appledorerg.com

+1 603 969 2125

44 Summer Street Dover, NH. 03820, USA