The Industrial Internet of Things
Volume G2: Key System Concerns

IIC:PUB:G2:V1.0:PB:20180807

CONTENTS

FIGURES

TABLES

1 OVERVIEW

1.1 INTRODUCTION

The Industrial Internet Consortium (IIC) originally published a single reference architecture document entitled *Industrial Internet Reference Architecture* (IIRA, V1.7)[1] to help system architects define Industrial Internet of Things (IIoT) architectures consistently. This initial IIC technical report described the several viewpoints and the various functional domains needed to evaluate industrial internet systems. It also defined common concerns that cannot be assigned to a viewpoint or functional domain. Addressing these concerns requires consistent analysis across the viewpoints and concerted system behaviors among the functional domains and components, ensured by engineering processes and assurance programs. We call these *key system concerns*.

1.2 PURPOSE

The purpose of this document, *Industrial Internet of Things Volume G2: Key System Concerns* (IIKSC) is to highlight key system concerns in industrial internet systems as special topics and provide additional analysis on them to assist system architects and others involved in designing and implementing IIoT architectures to be fully cognizant of their importance in such activities. It comprises the collective thinking of the membership of the Industrial Internet Consortium and its various work streams and testbeds.

1.3 SCOPE

This volume comprises the key concerns from the original *Industrial Internet Reference Architecture* version 1.7 that have not subsequently been addressed elsewhere in the *IIC Industrial Internet of Things* volumes. These key system concerns were deliberately removed from the second version of the *Industrial Internet of Things Volume G1: Reference Architecture*[2] and moved to this volume as a placeholder for the content until standalone volumes for each respective key concern are published. The contents of this document will be deprecated as those individual volumes of the *IIC Industrial Internet of Things* are published.

1.4 STRUCTURE

This volume is organized as follows:

- Chapter 1: Overview
- Chapter 2: Key IIoT System Concerns
- Chapter 3: Safety
- Chapter 4: Resilience

[1] See [IIC-IIRA2015]
[2] See [IIC-IIRA2017]

- Chapter 5: Integrability, Interoperability and Composability
- Chapter 6: Data Management
- Chapter 7: Intelligent and Resilient Control
- Chapter 8: Dynamic Composition and Automated Integration

1.5 AUDIENCE

This volume is primarily intended for IIoT system architects. We assume the reader is familiar with architecture concepts, architecture frameworks and reference architectures including architecture views and domains. This volume also has value for plant managers, IT managers, business managers and others who want to understand how the marriage of OT and IT enables the promised benefits of the IIoT.

1.6 USE

This document is intended to be used by system architects as a supplement to *IIC Industrial Internet of Things: Volume G1: Reference Architecture* to become aware of, and address, IIoT key system concerns as they develop their architectures and deploy their systems.

1.7 TERMS AND DEFINITIONS

The following terms and definitions are key to understanding this document:

Architecture: fundamental concepts or properties of a system in its environment embodied in its elements, relationships, and in the principles of its design and evolution

Architecture framework: conventions, principles and practices for the description of architectures established within a specific domain of application and/or community of stakeholders

Architecture view: work product expressing the architecture of a system from the perspective of specific system concerns

Architecture viewpoint: work product establishing the conventions for the construction, interpretation and use of architecture views to frame specific system concerns

Composability: capability of a component to interact with other components in recombinant fashion to satisfy requirements based on the expectation of the behaviors of the interacting parties

Composition: result of assembling a collection of elements for a particular purpose

Concern: interest in a system relevant to one or more of its stakeholders

Cross-cutting concern: concern that affects the whole system and thus may impact multiple viewpoints of the architecture

Cross-cutting function: function that may be applied and realized across multiple functional domains of the architecture to address cross-cutting concerns

Information technology: the study or use of systems (especially computers and telecommunications) for storing, retrieving and sending information

Integrability: the capability to communicate with each other based on compatible means of signaling and protocols

Interoperability: the capability to exchange information with each other based on common conceptual models and interpretation of information in context

Key system concerns: those concerns paramount to ensure successful architecting and deployment of successful IIoT systems

Operational technology: hardware and software that detects or causes a change through the direct monitoring and/or control of physical devices, processes and events in the enterprise

Safety: the condition of the system operating without causing unacceptable risk of physical injury or damage to the health of people, either directly, or indirectly as a result of damage to property or to the environment

Security: property of being protected from unintended or unauthorized access, change or destruction ensuring availability, integrity and confidentiality

Resilience: ability of a system or component to maintain an acceptable level of service in the face of disruption.

1.8 CONVENTIONS

Given that the document is non-normative, all 'must', 'may' and 'should' statements are to be interpreted as English language and not as in RFC 2119.[1]

1.8.1 TYPOGRAPHICAL AND LINGUISTIC CONVENTIONS AND STYLE

Terms that require definition are rendered in *italics*. (As the usage immediately preceding demonstrates, italics may also be used as example, or for emphasis.)

Generally, only the first use of the term is italicized. However, when a term can be read in its usual English language mode, the first use of the term may be italicized as the discussion becomes technical. In the first example below, "safety" and "security" are used informally. In the second, it introduces a definition.

"Among the key system characteristics that must be considered, safety is perhaps the most important, followed by security."

[1] See [IETF-RFC2119]

> "*Safety* is the condition of the system operating without causing unacceptable risk of physical injury or damage to the health of people, either directly or indirectly, as a result of damage to property or to the environment."

1.8.2 ABBREVIATIONS

See Appendix B for a full list of key abbreviations used in this document.

1.9 RELATIONSHIP WITH OTHER IIC DOCUMENTS

This document fits in the IIC Technical Publication Organization, see Figure 1-1.

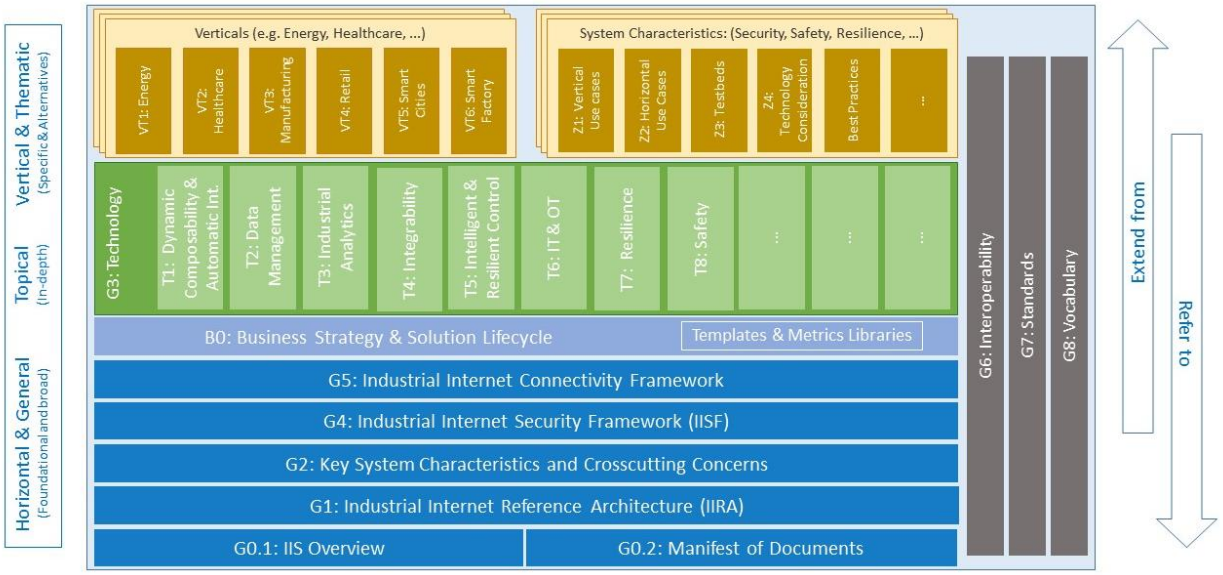

Figure 1-1: IIC Technical Publication Organization

2 KEY IIOT SYSTEM CONCERNS

There are a number of concerns unique to designing, building, deploying and operating an Industrial Internet of Things System (IIoTS), several of which are elevated beyond normal systems design concerns. We briefly describe each to provide a general overview and understand the context of their inter-relationships.[1] Each is explained in more detail in the subsequent chapters or in the other IIC Industrial Internet of Things volumes as footnoted.

Safety highlights a number of important considerations for safety in IIoTSs.

Security, Trustworthiness, and the relationships amongst them and Safety, Resilience, Reliability & Privacy, along with all relevant information related to this key system concern on how to secure IIoTSs end-to-end can be found in the Industrial Internet Security Framework.[2]

Resilience presents a few ideas on how to establish a resilient system, in reference to some of the learning from the military programs and operations.

Integrability, *Interoperability* and *Composability* suggests the direction in which IIoTSs' components should be built to support the dynamic evolution of components, including self-assembling components. It also serves as a unifying reference for some of the topics, such as Connectivity, Data Management, Dynamic Composition and Automatic Integration, all of which are to follow.

Connectivity discusses a foundational aspect of the Industrial internet—how to connect the numerous components (sensors, controller, and other systems) together to form IIoTSs. All relevant information related to this key system concern can be found in the Industrial Internet Connectivity Framework.[3]

Data Management concerns the basic approaches for exchanging and management of data among the components in IIoTSs.

Analytics concerns the transformation of vast amounts of data collected in an IIoTS into information that be used to make decisions and optimize systems. All relevant information related to this key system concern can be found in the Industrial Internet Analytics Framework.[4]

Intelligent and Resilient Control presents a conceptual model and some key ideas on how to build intelligent and resilient control.

Dynamic Composability and Automatic Integration concerns flexible adaptation to optimize services as environments change and to avoid disruptions as components are updated.

[1] Note that as of publication several of the key IIoTS system concerns listed herein have already been published as standalone volumes of the Industrial Internet of Things and are so annotated. However, they are retained in this list for completeness.

[2] See [IIC-IIoTSF2016]

[3] See Industrial Internet Connectivity Framework

[4] See [IIC-IIAF2017]

3 SAFETY

The industrial internet and the systems it comprises manage a variety of safety-critical processes. Safety must be considered and analyzed throughout the lifecycle of an IIoTS. Depending on the operating domain, regulatory requirements may mandate that a target safety assurance level be established for IIoTSs using a risk assessment process. While there are existing safety standards that may apply to IIoTSs under development for different domains (e.g., nuclear, rail, medical, automotive, process, maritime, machinery, and industrial process control), many are based on the fundamentals established in ISO 61508[1] *Functional Safety of Electrical/Electronic/Programmable Electronic Safety-related Systems* and do not explicitly address safety issues related to the cross-cutting concerns, architecture, integration and overall lifecycle of IIoTSs. A complete description of techniques for establishing and meeting IIoTS safety goals is beyond the scope of this document, but some are presented here.

Safety is an emergent property of the system in question and that has two major implications for systems engineering:

- Safety is not compositional: safety of every component in the system does not *necessarily* imply safety for the system.
- One cannot predict the safety of a system in a situation without first predicting the behavior of the system in that situation.

Therefore, IIoTS design must focus on not only mandating general notions of safety but also providing mechanisms that enable systems integrators to measure, predict and control the behavior of the system. For systems integrators to ensure safety, they must understand the intended behavior of the system and at the same time employ mechanisms that can constrain unintended behavior. Safety can be addressed either passively (e.g. by adding guards around a process to make sure nothing escapes the guarded area) or actively (e.g. by adding components that adjust the systems behavior to assure it is safe). Mechanisms include, but are not limited to:

Support for independent functional safety features: A functional safety feature is a feature the rest of the system relies upon to ensure safe operation. Examples include airbags in automobiles, ejection seats in military aircraft and the automatic shutdown system in nuclear reactors. It is not possible to prescribe specific functional safety features in general because a functional safety feature for one system or context may result in unsafe behavior in another. That being said, each safety-critical IIoTS must implement the functional safety features necessary to its safety requirements and usage context. Architecturally, functional safety features should be isolated and independent of the rest of the system to the maximum possible extent. This simplifies system safety validation and allows system integrators to mitigate costs associated with ensuring safe system behavior.

Well-defined, verified and documented interfaces: The system components used in IIoTSs must have well-defined, verified and documented interfaces. Systems integrators can use these

[1] See [IEC-61508]

specifications, and evidence that demonstrates that components conform to its interface, to make predictions about the emergent behavior of the composite system. Interfaces of concern include software, such as APIs, and relevant physical and processing characteristics that include the resource usage requirements and how the component will behave when used in its intended environment. Component manufacturers should make available any evidence used to verify that a component conforms to its specification. This helps system integrators predict how two or more components may interact when composed.

Enforceable separation of disparate functions and fault containment: Component manufacturers cannot provide complete assurance of component behavior because testing cannot cover all eventualities. Additionally, system integrators may opt to control costs by using components that come with less assurance providing those components will not be used to support a safety critical function.

Systems integrators must ensure that low assurance components will not negatively affect safety critical system functions. Thus, IIoTS design must have mechanisms to enforce separation between disparate functions and components. The enforcement mechanism must isolate faults and prevent *unintended* interactions between different system components. Examples of unintended actions include:

- a software component stealing CPU resources from another,
- a software component corrupting the data or instructions of another,
- a device on the network becomes a “babbling idiot” and preventing other components from communicating in a timely manner,
- a can of liquid on a conveyer spilling and causing the floor near the machine to become slippery, causing a mobile robot to lose traction and
- a motor drawing more power than expected causing a brownout affecting other devices on the same branch circuit.

Runtime monitoring and logging: Engineering is a human activity, and our knowledge of engineering is constantly improving. System failures, when they occur, should be looked upon as an opportunity to learn more. Mechanisms for gathering and preserving the episodic chain of events that has led to a failure may be useful to determine the underlying causes of a particular failure incident. Runtime monitoring and logging is one approach to gather and preserve such information.

In addition to supporting post-accident forensic activities, runtime-monitoring and logging can help *prevent* accidents. Runtime monitoring can detect if the system under scrutiny has entered or is trending towards an unsafe state and generate an alert. Some systems are equipped with special safety functions that automatically activate a safety mode in response to this alert. These safety modes are designed to either drive the system to a safe state, or prevent the system from entering unsafe states in the first place (e.g., the automatic shutdown system of a nuclear reactor). The runtime monitor can trigger such mode changes.

3.1 RELATIONSHIPS WITH OTHER CONCERNS

Safety interacts with other system concerns.

The role of reliability and resilience in safety-critical systems: Neither 'reliability' nor 'resilience' imply system safety. Indeed, there can be reliable systems that are unsafe in that they reliably perform unsafe behavior, and there can be unreliable systems that are safe. Even so, reliable and resilient infrastructure is useful and sometimes necessary to support the safety-critical functions of a larger system. Examples of features that support reliability and resiliency include fault tolerant computation and communication, replicated communications, distributed consistency protocols, adaptive control algorithms, and 'hardened' components, such as radiation-hardened CPUs and memory. Unfortunately, increased reliability may increase cost. This can be mitigated at the architecture level: The IIoTS could partition the infrastructure to allow for separation between the infrastructure used by safety and non-safety functions. The safety functions reside on a high-reliability (or high-resilience, and high-security) partition while non-safety functions are deployed on a less reliable (but cheaper) partition.

The relationship between safety and security: Often, system safety requirements impose system security requirements. Sometimes safety depends on the presence of a security feature. For example, if a platform cannot protect application code from unauthorized modification, malicious actors can corrupt safety-critical control algorithms and drive the system into an unsafe state. Sometimes, safety depends on the absence of a security feature. For example, one may actually want unauthorized or unauthenticated users, such as emergency responders to have the ability to initiate emergency shutdown procedures. Safety and security requirements (and their possible implementations) must be carefully balanced.

Implications of dynamic composition and automated interoperability for safety: Traditionally, safety-critical systems have been designed, manufactured and integrated by a single systems integrator. This model of integration allows the systems integrator to ensure the safety of their system by decomposing system safety requirements to sub-systems in a top-down manner and by verification and validation to ensure that no unsafe interactions were introduced during integration. Dynamic composition and automated interoperability functionality enable two models of system integration, each with their own set of implications for safety and the IIRA.[1]

Accelerated traditional: Here systems integrators still design, manufacture and integrate the system prior to its delivery to the customer. The systems integrator understands the top-level system safety requirements and the collection of specific system components that will be composed to comprise the system. In this model, dynamic composition and automated interoperability features, such as active inter-component interface checking, can reduce integration effort by the systems integrator *if* those features can be trusted. If the dynamic composition and automated interoperability features cannot be trusted, then it would be possible to compose components with incompatible interfaces, and other integration verification activities must be performed by the systems integrator to ensure the system components

[1] See [IIC-IIRA2017]

interact properly with each other. Therefore, if dynamic composition and automated interoperability features are used to support safety-critical functions, those features themselves must be verified and validated to the same level of assurance as the safety critical function.

User assembled: Here component manufacturers market a variety of interoperable components. Users can buy those components and compose them into a system designed to meet the user's specific needs. Users effectively act as the integrator of these systems. Unlike the traditional integration model, here the integrator (the users) would not have the necessary engineering expertise or resources to ensure the safety of the composite system. Indeed, the user may not even have a comprehensive understanding of the top-level safety requirements for the composite system. Instead, the dynamic composition and interoperability features of both the IIoTS infrastructure and IIoTS components must be designed to enforce safe system integration.

4 RESILIENCE

Resilience is more than just recovering quickly from pressure. To be resilient is to be able to take "bitter circumstance in stride" and still "get the job done." It might cost more or not be done as well had less (intentional or unintentional) adversity been present, but it will be done. Resilience is a superset of fault tolerance—and very much related to autonomic computing notions of self-healing, self-configuring, self-organizing and self-protecting.[1,2]

No other institutions are more involved directly in bitter or adversarial circumstances than the military. No other institutions have a greater dependence on resilience of its organizations and operations to survive and to succeed. Therefore, the current thinking of the military on resilience and the lessons they have learned in the past will inform us on how to better effect resilience within industrial internet systems.

Military *Command-and-Control* (C2) has four main functions:

Mission planning is either *strategic* (what resources and programs need to be in place to handle expected major events in the long term), or *tactical* (what resource can be deployed in response to an event that is expected to occur, and what is the overall impact if the resource is diverted from other tasks). Generally, tactical planning is done by units like ships or battalions and by units at higher levels such as a carrier group or a division) for larger engagements. Tactical planning focuses on a set of objectives such as the enemy's next moves and planning for those that pose the deadliest threat. For IIoTSs, this is the difference between how we *plan to fail* (create systems that are robust to expected kinds of failures and have sufficient resources to recover), and how

[1] Fault tolerance traditionally addresses system internal faults caused by a bug, hardware failure or some kinds of internal error states. Resilience has a bigger scope in that it focuses on harmful elements external to the system, often introduced by an adversary, that tend to be unpredictable and unforeseen by the system's designers.

[2] Sometimes called self-optimizing (as part of self-CHOP), but optimization is often too strong a concept—we will accept suboptimal but improved capabilities as part of recovery.

we *enable recovery* (create subsystems that are aware of their own performance and can adjust how they operate, particularly in light of their peers.)

Situation awareness is knowing what needs to be known about a situation in relation to the tactical plan. *Situation understanding* is the larger contextual picture: why things are as they are.

Resource management balances competing interests and concerns. It balances resources between threats that are current and imminent and those that are future and potential. For an IIoTS, it addresses questions such as how much computational resource to expend against detecting a security incursion vs. increasing replication of a service to assure that some copy will be able to compute a needed result in time.

Decide and assess is the execution arm and the part that measures what has been done. If one sets out to neutralize an enemy emplacement, places the order and completes an air strike on the target, the next question is whether it has in fact been neutralized. This of course has implications for next steps, which could be further targeting or determining the target is too hardened and change the plan. For an IIoTS, there will be a constant balance of trying to decide if the current result is 'good enough' or if additional resources need to be expended to improve it (such as additional sensor readings to reduce uncertainty, confirmation dialogues to reduce risk of unintended action), or suggesting an in-service part be scrapped after a minor failure as it is more likely to trigger a major failure.

The military generally sorts responsibilities into administration, intelligence, operations, logistics and communications, all under a common commanding officer. These groups may be replicated at several levels of command, so there may be division-level intelligence as well as battalion level (or fleet vs. ship).

Military orders between levels of command have a specific syntax, including a number of sections that must be addressed, but the most important aspect for resilience is the notion of *commander's intent.*[1] This is used as part of the mission-planning process where the commander sets up what the mission is about. The commander's intent is important to resilience when used operationally, as it enables an isolated unit at any level devoid of communication to still have a chance of knowing what to do even if the extant plan fails.


Example

For instance, if all we have is the order to neutralize hill 73, then we must continue until hill 73 is neutralized or we run out of men. If we instead know that we are told this in the context of getting a clear shot at bunker AAA which is blocked by hill 73, then a unit which is (temporarily) under independent command can look at alternatives; perhaps all that is needed is to suppress the enemy or divert them on hill 73 to give the unit the opportunity to take the shot on AAA. So, commander's intent pushes decision making down to the level that is best able (in terms of having

[1] Of particular importance, "what are the *N* most likely things the adversary may do?" "What are the *M* most dangerous things they may do?" Planning generally has to address all of these contingencies, the longer one has for planning, the larger *N* and *M* can be—which means that the unit can be ready for a wider variety of unfolding circumstances.

> the best information at the best time) to make the decision. Then when the mission is completed, the unit can attempt to reintegrate, report what they did while out of contact and allow the plans to be changed to address the new situation.

This approach handles uncertainty and failures of communication within 'decide and assess,' which is probably the most time-sensitive part of command-and-control. When executed well, there is a tremendous amount of flexibility and local negotiation possible even with 'disconnected' units to 'get the job done'.[1]

Establishing a global information grid (a military cloud) to provide information to the frontline commanders wherever they were has proved to have unintended consequences. For example, providing this information has offered a way to run battles directly from the command headquarters, overriding intermediate command. The military has moved in the other direction in recent years partly because such remote control of battle hurts resiliency. Similarly, moving industrial control to the cloud may also hurt resilience—not only does the network itself create an attack surface and a point of failure, but the information available at the scene will always be greater than that which can compressed into the pipe. For resilience, we should instead think of how to improve local decision-making through network services—without introducing new dependencies such as using the cloud for higher-level management and perhaps permission, but not low-level control.

Here then is a list of lessons we can learn from the military C2 structure and doctrine:

Expect to be disconnected from authority. Mechanisms must be in place to allow the mission to succeed, so some level of decision making on the edge is a requirement.

In an IIoTS, control elements for critical operations must not be dependent on network availability.

Good decisions are not made in a vacuum. Communicate commander's intent so that units in the field understand show their actions fit the bigger picture. The ability to alter plans locally provides more flexibility and resiliency.

The implication for IIoTSs is that local control elements must know more than just their own part of the plan. They must have a bigger picture of what they are responsible for that allows them to reconfigure their operation and maintain mission-level performance when under stress.

Peer-to-peer communication is more important than hierarchical communication. Changing plans and developing new tasks requires the disconnected units to engage in all parts of command-and-control jointly with their neighbors so they can jointly succeed within the constraints of the commander's intent. Once that intent (and an initial plan based on the strategically available resources) is communicated, little more needs to be said from higher chain of command until the mission is completed.

[1] The exact amount depends on the service—marines have a lot more doctrinal flexibility than army units do for instance.

In IIoTSs, this suggests that components must be autonomous, and able to act independently based on the plan and information from other independently operating components nearby.

Take advantage of the hierarchical network to optimize all parts of command-and-control. Do not use the connectivity, when available, to centralize decision making but distribute information to ensure that whole network becomes aware of changes to local plans so they can get an early start on changing too.

In IIoTSs, this suggests that components must be aware of the behavior of other components.

Build a system that does not need the network to work—it only needs it to optimize. This is a given in the 'fog of war'.

In IIoTSs, this partly follows from being able to run disconnected. However, some functions such as safety should never be compromised just because of a network failure.

Delegate authority but not responsibility. Delegate sufficient authority to the agents to get the tasks done, but assume full responsibility to ensure the tasks are done right.

In IIoTSs, mechanisms must be in place to find the right function for the job, and to validate that it did what it said it would do. It is important, for example, that orchestration elements not just suggest a process, but also suggest how the process can be validated and monitored.

Data without context can never become information. Pushing data around without context is not actionable. Context is hard to transmit as most context is the unwritten aspect of the circumstances and how the data was collected. Again, this points to the criticality of 'man on the spot' processing—local to where the context actually is, enabling new techniques like learning to discover local phenomena that can help the particular instance of the problem being solved (rather than the much harder problem of inducing broad general rules).

Plans do not survive first contact with the enemy. Many plans are reworked every time we learn something new. To have a plan is not to have a set of instructions for every situation but to ensure training is in place to handle every conceivable contingency. Battles will always be dynamic, what endures is the ability to see patterns, react quickly and get inside the enemy's OODA loop.[1]

Control systems observe (by reading sensors), decide (using a comparator) and act (using actuators). There is no 'orient' function. In IIoTSs, a resilient control architecture must be able to notice and discover when it is in an unexpected situation (i.e. orient itself), and then work to get itself back into a reasonable operating band, with the cooperation and collaboration of its peers.

Plan and prepare. Current military thinking tries to go beyond 'react and respond'. This goes back to mission planning. We must both plan to fail and enable recovery. We must also capture the lessons learned to aid future recovery. Recovery of prior operational capacity can mean changes

[1] "Observe, Orient, Decide, Act" Col. Boyd's brilliant insight into how fighter pilots operate. If one can execute one's OODA loop faster than the adversary can, they will usually win the battle because they can react to unfolding circumstances and in fact create unsettling circumstances faster than the adversary can. (This has also been applied to business management.)

to tactics, techniques and procedure ('doctrine'). That is how the *organization* learns rather than just individuals.

In IIoTSs, analytics can detect both imminent failure of a component, but also circumstances extant across a fleet of components when a component failed. This enables global learning.

U*pward communication is often more important than downward.* Mechanisms must be in place to communicate knowledge up the chain of command based on actual incidents, including what has been tried and failed.

In IIoTSs, we have to ensure the kinds of properties driven down to the autonomous edge devices are policies rather than plans. That is, they include advice about how to make choices in difficult situations, rather than specific courses of action. The implication of this is that edge devices are dependent on having appropriate computational elements for the complexity of the kinds of *reactive plans* (Wikipedia)[1] they are expected to implement.

I was just following orders. This is never a legitimate excuse in the military.

In IIoTSs, each component to the extent possible must make sure it does not violate local safety doctrine, even if it means ignoring direct orders from chain of command.[2] The 'unit on the spot' is on the spot both in the physical and legal sense.

There is a chain of command. Communications between units must be validated before they are trusted. Obey no commands even if they are issued by a higher-level officer unless they are established in that unit's chain of command. Trust is established before it is needed, and is necessarily hard to change with very formal procedures in place for transfer between commands.

Similarly, IIoTSs components should be inherently distrustful of 'changes of ownership,' 'new doctrine,' or even out-of-cycle updates; it is important that they are mechanisms not only for verifying they come from a trusted source, but that they make sense now.

[1] See [WKPD-REPL]

[2] Other doctrine, such as security and privacy, may come into play as well, but safety is the most important as it is the hardest to recover from.

5 INTEGRABILITY, INTEROPERABILITY AND COMPOSABILITY

IIoTSs are assembled from many components from multiple vendors and organizations within a vendor. To be assembled into large systems, these multifarious components must demonstrate:[1]

- *integrability*: the capability to communicate with each other based on compatible means of signaling and protocols,
- *interoperability*: the capability to exchange information with each other based on common conceptual models and interpretation of information in context and
- *composability*: the capability of a component to interact with any other component in a recombinant fashion to satisfy requirements based on the expectation of the behaviors of the interacting parties.

Composability relies on and adds to interoperability and integrability. Integrated components may have a capacity to communicate with each other but there is no guarantee that they can exchange information correctly, let alone whether they would have the intuitively expected behavior. Interoperable components can exchange information correctly but there is no guarantee their behavior is predictable. To look at this in another way: if an integrable component is replaced with another integrable component, the system may stop functioning; if an interoperable component is replaced, the system may behave quite differently; if a composable component is replaced with another with similar specifications, the system behaves in the same way.


Example

Two people are integrable if both are able to speak and listen; interoperable if they speak the same language; and composable if they share similar culture and educational background and can collaborate for specific tasks.

Consider a person as a potential pilot in an airplane cockpit. The person is considered integrable with the airplane cockpit if she fits well in the seat, can view the front horizon, see the instrument readings and indicators, and reach to all the controls—this includes any physically fitting adult. The same person is interoperable with the cockpit if she understands the meaning of the instruments and the intended outcome of the controls—this includes any physically fitting enthusiast about piloting. The same person is composable with the airplane cockpit if she is trained for the model of the airplane so that she understands the meaning of the instruments in context and the behavior of the airplane when she exercises control over it. One appropriately trained pilot can replace another in operating a plane.

IIoTSs are large in scale and constructed from many types of components that are each evolving at an increasingly rapid pace. Components will change from being automatic (working by themselves with little or no direct human control) to autonomous (having the freedom to act

[1] This model is based on Toward a Family of Maturity Models for the Simulation Interconnection Problem [Page-2004]

independently) and they will need to be able to self-assemble. Integrability and interoperability are inadequate to meet the needs of such systems. We also need composability.

Human interaction and communication using natural languages has proven to be a robust and dynamic method for composability. This is evident from observing how well two strangers can communicate on the spot, with minimal preparation for integration and interoperation, and how well they form groups collaborating to complete large tasks (or gossiping on Facebook).

Willingly or not, component designers have different models of reality with different constraints and assumptions. Lacking telepathy or shared memory between designers of different components means models, constraints and assumptions are not easily communicated or shared. It is therefore difficult to support the higher levels of communications and interactions beyond the level of integrability.

Facing this challenge, we impose a mental framework for undertaking integrability, interoperability, and composability as different levels of communication or interaction, similar to that taken in *conceptual interoperability* (Wikipedia[1]), an idea grounded in simulation theory—how to make different parts of a simulation system interoperate.[2]

We treat each communication in terms of passing messages containing symbols, similar in concept to natural language. (We will use natural language for examples.) The manner of communication supporting integrability in messages is well understood, so we focus on interoperability and composability.

At the base level, we have vocabulary and syntax—grammar—rules on legal word order and the relationship to meaning. For example, “car race” and “race car” both use the same words, and the meanings of the words themselves to do not change, but the thing that is denoted by the two examples, which differ only in their syntax, is different. Syntax can be fixed, as they are for the order of entries in a form, or variable as they are in a sentence where grammar is expressed as a set of rules.

The next level up is semantics—the meaning of the words themselves. In most languages, each word can have more than one sense, which is taken up by context. For example, “safety” has one meaning in an industrial setting and another in football; we would not expect a safety engineer to worry about a defensive strategy in a game. Typically, in systems design we try to have each symbol have a unique context-free meaning, but this can also lead to excessive verbosity as well as inflexibility.

Database schemas tend to express the semantics of items in the structure of records, and these can be used as a kind of translation between different systems. For example, one database may talk about ‘names’ while another may segregate ‘last name’ and ‘first name’ but the meaning of

[1] See [WKPD-COIP]

[2] The considerations here are like those studied under linguistics and the philosophy of language—to ask how it is possible for humans to communicate, for us to know someone else’s meaning, and how we can learn new things without actually experiencing them through language, by, for example reading a book.

'last name' is culturally dependent, so translating between the two may depend on the culture of the person being denoted. In one culture, the 'last name' is given; in another, it is that of the family.

Therefore, to exchange information with natural languages, we need to share some basic vocabulary (how a word is interpreted in different contexts) and syntax in which the words can be arranged into structures. This semantic understanding in communication is the basis of interoperability.

Next, we have pragmatics, which is the meaning of a sentence *in context.* This generally depends on the conceptual model of the world. Pragmatics allows us to understand the import of an utterance on a particular occasion. In speech act (Wikipedia[1]) theory (how speech can be treated as a form of action), the base level is the 'locution' or what was actually said—the grunts in the utterance along with the syntax and semantics that are defined by the language (as opposed to the use in this instance). Above that is the 'illocution' or what the speaker intended to say: the pragmatics.[2] The final layer is the 'perlocution', which is the effect (intended or otherwise) on the other parties who hear the locution.

So, one way to think about communication is that we want to specify the illocution (what we intend to say) such that the perlocution (the effect on the listener) can be what we intend. ('Can be' rather than 'will be' because the speaker does not control the mental state of the listener, and the listener may not use the locution for the speaker's intended purposes—let's recall the 'invented the internet' meme among others.)

Therefore, to exchange information with natural languages to achieve the intended effect, we need:

- to share a common or similar world knowledge (the understanding of the natural world and culture),
- the conceptual model, to have the ability of comprehend the meaning in its context (locution and illocution) and
- some general expectation of the other party in their understanding of the information (illocution) and their reaction (perlocution)—behavior.

This pragmatic understanding is the basis for composability.

With this understanding of the different ways components can be assembled to form larger systems and how they are related to the different levels of understanding in communications, Table 5-1 identifies where each of these elements are addressed:

[1] See [WKPD-SPAC]

[2] Agent-based systems generally spill quite a bit of ink defining 'performative' speech acts (those that are performed by way of saying them, that is, when I say, "I promise you…," I have done something, namely made a promise!). See [FIPA/IEEE CALS2002]

Levels of Communication	Levels of understanding in communication[1]	Refer to
integrability	Technical	Connectivity (see Industrial Internet Connectivity Framework[2])
Interoperability	Syntax—getting the format of the messages right Semantics—getting the meaning of the symbols in the messages right	Connectivity (see Industrial Internet Connectivity Framework) and Data Management (chapter 6)
Composability	Pragmatics/illocution—interpreting what was intended by the sender	Intelligent and Resilient Control (chapter 7), Safety (chapter 3), and Dynamic Composition and Automatic Integration (chapter 8)

Table 5-1: Mapping of Levels of Communication to topics in this document

[1] The three levels of understanding in communication discussed here loosely map to the corresponding levels of interoperability as defined in the *conceptual interoperability* (Wikipedia) [WKPD-COIP], i.e. technical, syntactic, semantic and pragmatic understanding to technical, syntactic, semantic and pragmatic interoperability, respectively. We use the term understanding in place of interoperability here to avoid the confusion that might be otherwise caused by also using the term interoperability in the levels of communication (integrability, interoperability and composability). However, phrases such as semantic interoperability will be used in other sections of this document.

[2] See Industrial Internet Connectivity Framework

6 DATA MANAGEMENT

Industrial Internet Systems Data Management consists of coordinated activities involving tasks and roles from the usage viewpoint and functional components from the functional viewpoint, specifically:

- Reduction and Analytics
- Publish and Subscribe
- Query
- Storage, Persistence and Retrieval
- Integration
- Description and Presence
- Data Framework
- Rights Management

6.1 REDUCTION AND ANALYTICS

Sensors and other systems in an IIoTS produce extremely large amounts of data.[1] Transmitting all this raw data over the networks to a central data center is often unnecessary and prohibitively expensive, but insights contained in the raw data must not be lost.

Reduction and analytics can manage data by either reducing the volume or velocity without losing the value or the information content. It is analogous to lossy data compression, as the original IIoTS data is irretrievable.

Analytics summarize raw data and produce an approximation of the truth that is suitable for downstream communication, processing and storage, while *data sampling* and filtering are examples of data reduction techniques devoid of analytics. Data reduction and analytics services suggest a migration of computing, networking and storage resources from enterprise to edge systems.

6.2 PUBLISH AND SUBSCRIBE

Publish and subscribe is suited for exchanging data updates between loosely coupled components and allows the publish-subscribe framework to optimize the communication path between publishers and subscribers based on their requirements.

Publish-subscribe contributes to IIoTS reliability, maintenance and resilience by the decoupling of publishing and subscribing components in both location (location transparency) and time (asynchronous delivery). This decreases the likelihood of fault-propagation and simplifies incremental updating and evolution. Interactions on the receiver side can be periodic (time-driven) or responsive (event-driven), depending on the needs of the user. Asynchronous transfers

[1] Data quality monitoring and sensor health monitoring are common applications for data reduction and analysis.

can also handle IIoTS component failures such as a network failure on the data path by delaying rather than cancelling an ongoing data transfer operation.

Publish-subscribe naturally supports the following kinds of IIoTS data exchange.

Streaming data: Data is continually or periodically updated at fixed rates ranging from kilohertz frequencies to multi-second periods, requiring low latencies and jitter with best-effort reliability. Components often check for the reception of data on a periodic basis. When the volume of streaming data is exceedingly large, publish-subscribe offers key advantages for the large numbers of interconnected systems.

Alarm and event: Data is issued when detection of specific IIoTS system conditions occurs. This spontaneous publication requires the IIoTS system to provide at a minimum guaranteed at-least-once delivery. IIoTS alarm and event data should be delivered with low latency and high priority, and pre-empt lower priority data where needed to ensure critical alarms are transmitted within acceptable delays. Parallel processing of a topic by multiple subscribers is essential when large number of spontaneous alarms or events may arrive at once.

Command and control: Control algorithms or people change the behavior or state of IIoTS components by generating command and control messages. They are typically time sensitive and require delivery by a deadline to allow target IIoTS component to react in a timely manner. Spontaneous publication is the norm and it requires guaranteed, low-latency and high-priority delivery, pre-empting lower priority data where needed, to minimize response time.

Configuration: Configuration or policy data are exchanged to enable IIoTS components to adjust their algorithms and behavior. These data change slowly, and typically have low latency and low priority requirements. Persistence is essential to support information requirements of newly joining subscribers even when the original moment of publication is missed. The data may also need to persist beyond the lifetime of the original publisher.

Publish-subscribe serves these purposes:

- Reliable data flow from the edge to a data consolidation and aggregation tier, for example to a cloud-based data services platform.
- Scalable handling of a large, evolving number of data sources such as devices, as well as of a large number of data consumers.
- End-user, application-level data consumption often requires a subscription model, from data consolidated on a platform to application components.
- Reliable control flow from applications or management services to devices: Control commands can be multi-cast in a way that allows devices to get these commands whenever they are ready.

6.3 QUERY

IIoTSs employ two models to make queries. The *one-time query* model is associated with traditional databases and it fits well with the request-response pattern. The *continuous query* model is associated with data stream management systems and in-memory databases. It fits

naturally with a publish-subscribe pattern and is better suited to handle infinite and rapidly changing data streams and support real-time analytics.

IIoTSs use a combination of two styles to select a subset of data from a larger data set: *save data; run query* and its inverse, *save query; run data.*[1] Both query styles and models may apply at different levels in an IIoTS architecture, including at the device level.

Addressable devices may support direct queries (e.g. using WebSockets) in pull mode. Alternatively, device data may be pushed to a gateway configured with filtering rules that selects a subset of the device-generated data stream before acting as a data source to a higher-level data broker or data bus.

Query serves the following purposes:

- selection of a subset of device-generated data, either pulled by requests to addressable devices or pushed to a gateway running filters and
- selective, usage-centric access to consolidated data by end-users and analytics, possibly in the cloud.

6.4 STORAGE, PERSISTENCE AND RETRIEVAL

Storage, persistence and retrieval support many IIoTS functions:

Record supports defining and persisting a subset of IIoTS data in sequential order. Preserving time-stamping information supports ordering identification and reproduction between different data sets. Record data is typically not queried or reproduced as a time-series. Record is used for meeting record-keeping obligations, post-processing and analysis, replaying of system scenarios and related II use cases.

Replay supports retrieving a collection of IIoTS data previously recorded by replaying data-items in the order received. Replay supports creating simulation environments, regression tests and related II use cases.

Historian persists selected data for delayed time-series analysis.

Big data solutions support voluminous IIoTS Control Domain system data.

Storage, persistence and retrieval serve these purposes:

- creation of audit records for future auditing (record and historian),
- support for simulations and various forms of testing (record and replay) and
- reliable storage and scalable archiving (big data).

[1] Essentially, the difference is in the active element. In the former case, a knowledge base contains the data and a query is run against that data such as SQL. In the latter case, the query is fixed in a stream processor and the data is run through it to filter out anything from the stream that does not fit. The latter can be a better fit when the data being generated is too voluminous to store, however, once filtered the 'filtered out' data is lost.

6.5 INTEGRATION

Subsystems often have only partially compatible data models, so integration mechanisms between them are essential. An IIoTS Integration mechanism may use a wide variety of available integration mechanisms, including:

Syntactical transformation, which requires knowledge about the structure of the data and transformation rules in both IIoTS subsystems. *Presence discovery* (see *description and presence* below) partially addresses this requirement. Semantic compatibility is also required and can be achieved via an open standards-based metadata solution such as ISO/IEC 11179-5.[1]

Domain transformation, which converts a data domain based on one protocol to a data domain based on another.

Integration serves these purposes:

- enabling integration across various middleware and application components and
- supporting functions analogous to conventional Extract/Transform/Load (ETL), typically occurring in the first stages of data transfer, and preceding initial storage.[2]

6.6 DESCRIPTION AND PRESENCE

Description and presence enable components to discover the kinds, format, structure and metadata of available system data. Both use a variety of available mechanisms including query.

Presence allows components to discover which kinds of data are available using mechanisms such as query.

Metadata description enables components to obtain definitions of the structure of, and other information about, the present data.

Description and presence serve the following purposes:

- dynamic integration of new application components or middleware in a deployed IIoTS,
- addition of new types of devices with different data models and communication modes,
- design of a system management console applicable to various IIoTSs and
- composition of IIoTS with different data models.

6.7 DATA FRAMEWORK

Data frameworks provide users with insight into state and behavior of data exchange components by exposing diagnostic data, such as data update rates, number of discovered framework participants and detected message loss. Diagnostic data is similar to other data and therefore subject to all regular data mechanisms.

[1] See [ISO-11179-5]

[2] Extract, transform, load, a process in database usage and especially in data warehousing.

Monitoring and analyzing framework-associated information access is required such as data exposed by 'description and presence'. Framework mechanisms produce regular data that should be accessible via 'publish, subscribe and query'. The data provided enables creation of a dashboard for the data management framework that can track:

*Component presence discover*y identifies IIoTS component past or present framework participation.

Component activity monitoring monitors IIoTS component data such as update frequencies, throughput numbers, CPU load and memory usage.

Traffic monitoring monitors data flow characteristics such as data exchange volume, throughput, latencies, and jitters.

Data frameworks serve these purposes:

- design of a system management console applicable to various IIoTS and not specific to system technologies and components and
- ongoing deployed IIoTS testing and diagnostics

6.8 RIGHTS MANAGEMENT

IIoTS data rights management identifies and tracks data ownership. Rights management enables data owners to grant use rights, manage access based on the granted rights, and protect against unauthorized use. Rights management must be built on security functions but are clearly distinct from generic data protection and privacy.

Rights management serves these purposes:

- general data stewardship, in particular in case of consolidation and integration scenarios (between IIoTSs, or IIoTS integration with enterprise systems),
- out-sourcing of data-related functions of an IIoTS to third parties such as cloud providers and
- support for regulatory and compliance requirements.

7 INTELLIGENT AND RESILIENT CONTROL

7.1 MOTIVATION

The control model prevailing in industrial automation systems today tends to be localized in scope and reactive in response, such as the limited control-loop feedback mechanisms implemented by proportional-integral-derivative (PID) controllers. When we embark on the task of creating a control, even a simple PID controller, we must consider a number of system engineering factors in respect to the conditions, constraints on operation and the context. We then build a mechanism that takes some inputs to produce some outputs including engineering data values (voltages, temperatures etc.) and control signals to hardware, (opening or closing a breaker). Most of these factors are kept in the heads of the control (systems?) engineers and are thus a black box.

In IIoTSs, we intend to perform distributed rather than local control, and to make predictions about how the world will change as a result of control. Moreover, this control must be 'intelligent and resilient' so that it can operate within a dynamic and unpredictable environment, using a distributed, collaborative capability to sense, make sense of, and affect the world and so achieve the goals of the specific entity that is acting (the agent). However, to reason and make predictions about how other controllers will work, particularly in unpredictable circumstances and dynamic environments, etc., we need transparency into that black box—the head of the control engineer. We need to understand how those choices were made, and what models of the world, assumptions about the world, understanding of the actions an actuator can take, and so on, prompted those decisions.

By employing models, either explicit or implicit, we can affect the desired intelligent control of the resources available to the agent and enable planning to bring the world to a state more acceptable to our interests. By making these modeling choices explicit, we improve the communication with the users of the system, and enable more advanced approaches to resiliency.

7.2 CONSIDERATIONS

The control engineer makes these choices based on a number of considerations, including:

Is the model of the world fully or partially observable? In a chess game, where the world is completely observable, the rules are known, so a legal move has a deterministic result. We may not be able to precisely determine the opponents move, we know it will be from a list of legal moves, and once it has been made we will know which move in the list has been taken. On the other hand, in a world that is only partially observable, such as many card games where cards are hidden, we are forced to infer world state based on the actions of the opponent. This affects our choices of the decision theory to use, the way we will model the world, the kinds of recovery strategies that are available to us after a fault, etc.

Are actions deterministic or probabilistic? When dealing with actions that can fail, such as a game of billiards, we are forced to consider not only the position of the balls we want to create if our

shot is successful (leading to the next shot we will take being easier), but also that if we are unsuccessful (leading to the next shot our opponent will take being harder). Many models presume sensing the world is 'free' in that it evolves continuously and outside of the influence of our decision-making system, but taking a reading may require effort in that we must move our sensor to observe something. That means that a world that might at one level of analysis appear to be fully observable is really only partially observable (because we don't have the processing capability to digest all of the sensor information we may be receiving) and also that what we think is the 'right' choice may not be, because we didn't see everything—the outcome of a particular action in a fully known state may be deterministic, but if we can't fully know the state—if some portion of it is uncertain, then we may want to say that the outcome of the action is uncertain (probabilistic) as well.

Can we plan all at once, or must we plan-to-plan? One way to deal with uncertainty is to defer planning until we have more knowledge—that is we can 'plan to plan' in that we create a partial plan that includes 'planning' as an action that will be taken under certain circumstances.


Example

I may not know the train schedule to New York City, so I have to plan to get the schedule before I buy the ticket. Since getting the schedule is insufficient to know which ticket to buy, I then have to plan to plan—decide now to postpone my decision as to which train to take, and thus the specifics of what I will do upon arrival. Or I may create a contingency plan, where I do all the work now iterating through every reasonable contingency, e.g., arrival before lunch, arrival after lunch, arrival after dinner, arrival after the subways are shut down, etc. It is a metacognitive action to decide which kind of plan I should create, but that decision can also be fixed at design time (i.e. the system will always generate a non-contingent plan, and if there are sufficient unknowns to prevent generation of such a plan, planning will fail with a list of unknowns to be satisfied).

Can we specify alternative methods to achieve goals? There's usually more than one way to skin a cat, and just as there are several possible routes to travel from point A to point B, by specifying multiple non-redundant ways to achieve a goal we build in a mechanism for the system to have backup strategies. For instance, while raising the house temperature is best achieved by turning on the furnace, a fireplace can be used, or the stove, or electrical heaters—all alternative methods that can be used if the furnace temporarily is non-operational.

Can we specify methods to reclaim or recover resources (particularly after casualty)? Such methods may be as simple as instructions for rebooting the network routers in case the network stops working, to strategies for reducing electrical usage to allow high current machinery to be started. Similar to the alternative methods, such an approach allows us to construct resilient systems—those that can reconstitute their capabilities after a failure. One can easily imagine resources having been assigned to processes (such as a database) to be in an unknown state during use, but should the process fail, we need to have a method to return those resources to some kind of known good state so they can be used again without having to wait for repair. In a database, this might be a rollback; in a nuclear power plant, this might be an orderly shutdown followed by a cold restart.

Do we want to learn and adapt to our inputs over time? Our main concern is application to dynamic situations - so the connection between the inputs and outputs may not be known perfectly at design time, and may change over time. For instance, if we are controlling both heat and humidity in a house, we may not know what kind of insulation the house has, and the system may not even know the time of the year it is (so if heating or cooling will be called for, if humidity will need to be added or subtracted, and at what rate). We will know that cooling will dehumidify. But presuming we have (unlike most residential systems) the ability to change the rate of cooling or adjust the balance between cooling and dehumidification, we may want to do so based on how the system has reacted in the past to such controls and furthermore be willing to readapt since seasonal changes will alter the response.

These considerations allow building appropriate models of and the relationships between the following that were previously mapped in the control engineer's head:

- the (relevant) world (context, environment, state of the universe, etc.),[1]
- action (both atomic and compound—e.g. a typical process that does something useful),
- communication (as a kind of action),
- intention (as of other agents),
- sensors,
- actuation and
- ethics (that is, those actions we must and must not do within a context).

7.3 FUNCTIONAL COMPONENTS

Figure 7-1: is a sketch of one way we might architect an intelligent control for a very dynamic environment. As a top-level decomposition, we have the following modules:

Deliberative and reactive planners: Long-horizon plans (typically called "deliberative") set goals. Short-horizon planners ("reactive") make satisficing real-time decisions (addressing resiliency) using a long-term plan to guide executing the plan in the current situation. Thus, even when the long-term plan is obsolete, i.e. we cannot execute the plan as written, the reactive planner must be able to modify it on the fly to fit the actual circumstances. The planners handle most physical or logical planning constraints. For example, you cannot put down something you are not holding and you cannot use two positives to make a negative.

[1] Modeling invariably involves abstracting away irrelevant detail. Deciding what is and what is not relevant is part of the job and risks of systems engineering.

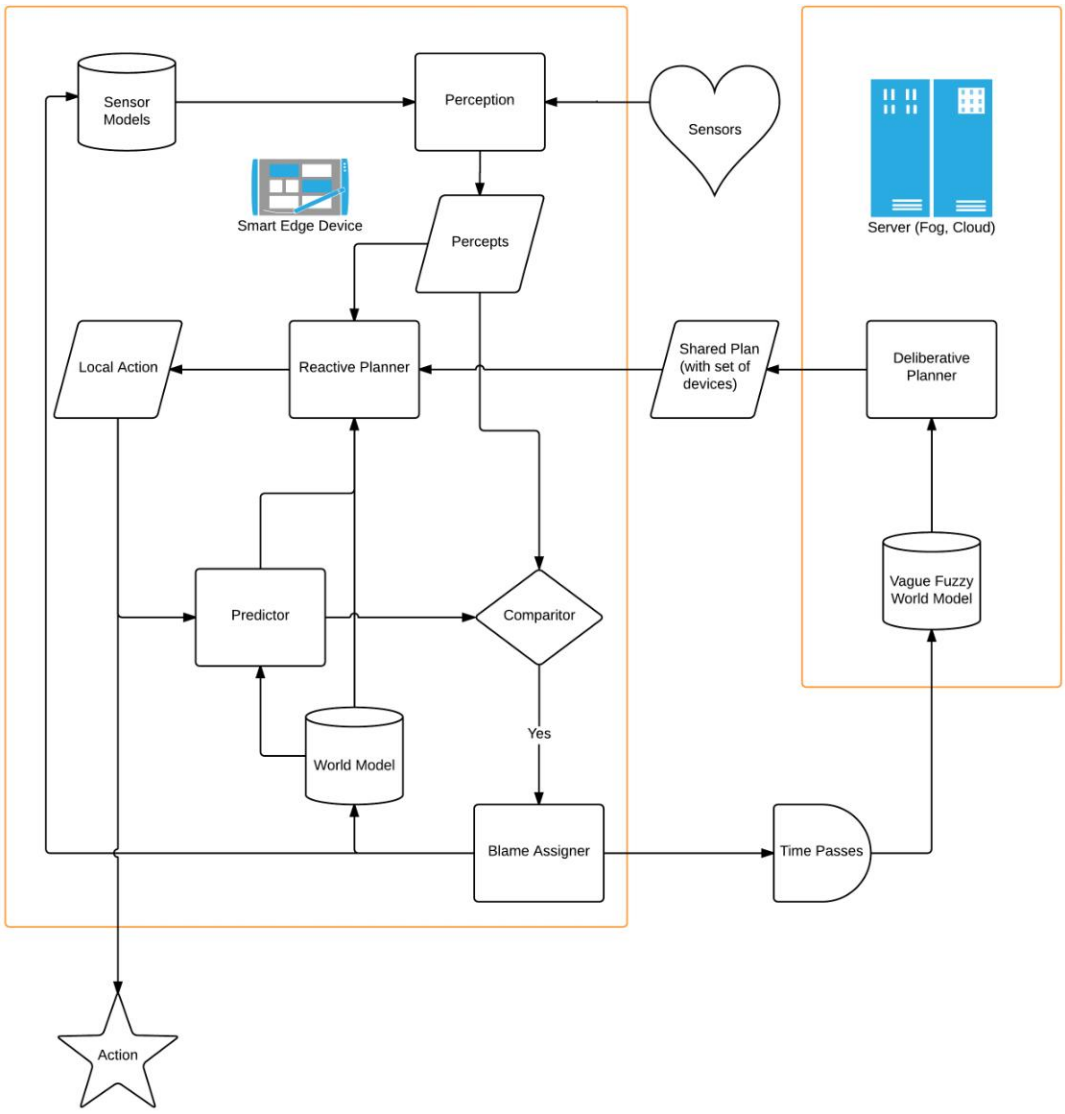

Figure 7-1: Intelligent Control Model

The deliberative planner needs more resources to establish a long-term plan, and is driven on a general understanding of the current world state, but is not the 'man on the scene.' That is the reactive planner, in the tight loop with the sensors and actuators making moment-to-moment decisions—guided by the long-term plan, but able to override actions—the long-term plan provides the moral equivalent of 'commander's guidance' while the reactive planner is the non-commissioned officer making tactical decisions under fire.

Because planning is a joint activity, and 'the plan' may not be visible to any particular agent or set of agents—since much of the plan is parochial and by the time it is observed by a (remote) agent, it will have been implemented or overtaken by events. A flexible mechanism for planning and implementing plans is therefore called for. One recommendation is a small but flexible reactive planner in the device itself and a deliberative (offline) planner provided as a service by a larger system or through remote service providers.

Predictor and precepts: Because plans may fail in an uncertain and dynamic world, we should expect that any particular agent's plan may fail, and that its models may make mistakes and fail. We therefore need *perception* to look at the relevant part of the world (driven by the plan) and for it to generate *percepts*—individual perceptions of interest to the agent.

The *Predictor* function informs the reactive planner what the likely outcome of planned actions will be, and through the *Comparator* can look at what actually happened as the result of taking an action. If there is no difference, the operation continues but when there is a difference, we invoke the *Blame Assigner* (see below). A predictor function predicts what the state of the world will be at some point in the future, given an action or lack of action by that agent. (We regard not taking an action as an action: waiting). We can break the predictor down into two components: one that uses the models to chain out a possible future, and another that learns from experience. A learning function also requires additional parameters, including at least the inertia (how long to wait until making a prediction) or entropy rate (how likely is the pattern of the next input to be different from the past).


Example

If we see the time series 0, 1, 1, 2, 3, 5, 8, 13: at what point do we react and say 'Fibonacci'—after the 2? The 3? The 13? At what point do we go back and make sure we are *still* seeing Fibonacci numbers? Every time? Every *xx* numbers? What if the pattern repeats? Stops and changes to some other pattern?

Blame assigner: Given a world model, a predictor and a plan, we can predict the likelihood our plan will succeed, and then amend the plan to increase the likelihood of success (in probabilistic models). But we may be 'surprised' when our action does not have the intended effect. The 'blame assigner' makes the decision as to why things aren't as we thought they would be by considering a number of possible scenarios to determine the component at fault when something goes awry. It could be bad sensing where the control did generate the expected effect, but our decision to act was based on incorrect sensor data, or the world was not in the state we thought it was when we selected the action. It could be because of a bad model in which we have the wrong effects or likelihood of effects from our action, or because of the conditions under which the action is effective were incomplete, etc. It could also be because of faulty action in which e.g. we intended to press button A, but actually pressed button B.

Ethical governor: An *ethical governor* (not in this diagram) might also be used to vet the action decisions to make sure that the agent does not perform any action it 'must not' perform (for, e.g., safety, security, or other reasons) and does perform any action it 'must' perform. It is a special deontic checker that validates that a course of action is within the scope of agreed upon ethics within appropriately negotiated, communicated, and represented community policies. The ethical governor must have the ability to override the agents' intentions. We give ethical rules special treatment because they do not tend to be an issue at the level of action selection but rather the overall plan pragmatics.[1]

[1] See Arkin: Governing Lethal Behavior in Autonomous Robots [Arkin-2009]

Security, safety and other models implemented by the ethical governor would be able to reject a request made by an operator or other agents. The autonomy implemented by an agent would always have 'final say' on accepting or rejecting a request.

Another task of the ethical governor is to determine when it is safe to dynamically change a device's behavior and/or performance through updating the device by deciding if the update is appropriate for the current circumstance and does not violate a safety or security constraint.

The agent should also store its own meta-information, so as to advertise its capabilities to the community, or to describe them in whole or in part in answer to a query about them.[1]

[1] Note that this does not require the agent to understand the meta-information, just be able to report it. The meta-information could then be parsed or interpreted by the receiver, through either pattern matching or general reasoning.

8 DYNAMIC COMPOSITION AND AUTOMATED INTEROPERABILITY

8.1 MOTIVATION

IIoTSs require secure, safe and scalable composition of many diverse components from a variety of sources, often with different protocols, to deliver reliable end-to-end services. Given that distributed industrial internet applications are intended to be responsive to dynamic environments and that related technologies and standards are rapidly evolving, resilient IIoTSs need to adapt flexibly to optimize services as environments change and to avoid disruptions as components are updated, upgraded and replaced. IIoTSs present new use-cases in distributed computing that will drive advances in information technology architecture.

Service orientation (Wikipedia[1]) defines a logical framework for thinking about exchanging capabilities and data via distributed services and the ability to compose services into high-order applications and business processes. Though implementations vary and evolve, in general practice, service compositions are models of statically connected components. The relationships between components are defined in advance. At run-time, "orchestration engines" merely execute the composition. The method of composition design does not provide for any adaptation in response to change in the environment or of the components themselves. This tight coupling makes the compositions brittle; change requires manual redesign and generation of a new model or else the service is likely to break. The approach is not scalable in dynamic IIoTS environments. Advance testing of such service compositions can only validate the model in a controlled environment that is not representative of real-world operations.

IIoTSs require a flexible method of composing services, so components can be dynamically integrated at run-time to enable adaptable services. Instead of static point-to-point connections, the demands of IIoTSs need semantic interoperability to support many-to-many connections. In this approach, compositions indirectly link components using metadata references to a domain information model. Compositions represent a set of references to the information model, rather than a fixed set of named components. By separating the model from the implementation, semantic service compositions support metadata-driven policy-controlled orchestration that interprets references at run-time to dynamically discover components and their connections, as well as their transport and transformation details and integrate them on-demand. This loosely coupled approach automates interoperability and enables policy-based optimization for flexible and dynamic IIoTSs that can adapt to change and re-configure network resources accordingly. Since semantically composed services have abstract contracts that are dynamically translated to concrete implementations, simulations can demonstrate the impact of real-time change and provide an execution trace that can be validated.

Many of these concepts were part of the original vision of Service Oriented Architecture (SOA).[2] In the early days of SOA, the software industry focused on static solutions that it could bring to market quickly to satisfy pressing business demand for distributed computing. While dynamic

[1] See [WKPD-SEOR]

[2] See: [Siegel] and [OASIS]

features were perhaps ahead of their time, IIoTSs now clearly require 'smart', adaptable composite applications. The demands of IIoTS drive change in several areas:

Situational awareness: Static solutions do not provide mechanisms for resolving possible incompatible assumptions about, for example, the operating environment, the deployment context, the interacting entities, and so on.

Workload diversity: Static compositions cannot change their stripes. In the real world, an end-to-end process may be a collaboration, choreography or orchestration with elements of various levels of autonomy. Many responses may be taking place simultaneously and would require flexible compositions in response to event.

Complex relationships: In IIoTSs any element may have peer, parent and child relationships and thus varying roles and perspectives. This applies recursively to the participating elements, which may represent complex systems and involve a network of sub-systems. In a static solution, as tightly coupled complexity goes up, resilience goes down.

Dynamic relationships: In IIoTSs relationships are constantly forming and un-forming, as in a 'friend-of-a-friend network'. The dynamics are changing constantly, creating on-the-fly activity and changing state among collaborating components. A self-forming composition may come into being when there is value in coordination, and then disengage or be uninvited from the group as events unfold.

In short, static, compartmentalized and centralized methods are not suited for dynamic optimization that concurrently satisfies multiple constraints for dynamic, diverse and distributed interactions expected in an IIoTS, and this prompts a shift from static models of integration and orchestration to "dynamic composition and automated interoperability".

8.2 CONSIDERATIONS

Practices and standards established today may enable or constrain future capabilities, so any approach must provide enduring value, and delay the need for expensive and time-consuming re-evaluation and re-design of the architecture.

By separating models from implementation, dynamic composition and automated interoperability supports a future-proofed IIoTS architecture that accommodates change by design (i.e. integration decisions are postponed to run-time to allow for optimization and adaptation).

We compose loosely coupled components using metadata references, contained in an information model that captures logical models of services based on abstract contracts. The abstract contracts decouple system capability and control from the details of the implementation and infrastructure complexity. Consequently, the system capabilities can be declaratively described in the form of policy to bring about the desired service, function, conditions, and so on, without having to understand low-level implementation details.

Since the abstract contracts do not explicitly define the implementation, they need to be interpreted at run-time by an agent that acts as an intermediary responding to events. Every

event is an opportunity for an agent to add value, not just to perform a static script, but to discover patterns, perform functions, run analytics and otherwise 'reason'.

Agents resolve all references to find-and-bind the right resources just-in-time by performing all necessary connections and transformations, and to provide an optimal, context-enhanced response. Moreover, these agents monitor the responses of the employed services to ascertain if they are, in fact, responsive in accordance to the contract and otherwise replace the underperforming services to achieve the desired end-goal.

To maintain Quality of Service (QoS) and Quality of Experience (QoE), applications in an IIoTS may require visibility to the state of the interacting elements and network resources so it can observe and react to changes in connectivity and faults. This requires information be aggregated and translated into a common abstraction so system management applications can respond to events with any necessary re-configuration of the infrastructure and service implementation. The same high-level abstraction supports change at the application level, with the same QoS and QoE implications, to provide for sustainability of the end-to-end business capability.

In short, dynamic composition and automated interoperability must allow for real-time, data-driven, policy-controlled integration of services, and systematically late-bound at run-time, rather than integrating them in advance at design-time.

This approach for IIoTSs, as a natural advancement in the evolution of system composition and service orchestration, has other significant benefits:[1]

Virtually centralized policy control: Security, business compliance and IT governance policies can be linked to abstract contracts addressing the historic challenge of managing consistent policy enforcement of system-wide concerns across diverse and distributed components.

Service adaptability: Since abstract contracts are not tightly coupled to any resources in advance, they can automatically evolve with updates and upgrades of underlying components without interrupting the operations.

DevOps productivity: Automating interoperability eliminates repetitive, error-prone and time-consuming integration work, accelerating service delivery and reducing cost.

8.3 FUNCTIONAL COMPONENTS

The key functional components required to support the dynamic composition and automated integration are:

Integration contract management:[2] The integration contract management functional component provides capabilities for managing abstract contracts for automated interoperability, including:

[1] For more details, see Semantic SOA makes Sense [Duggal-2014]

[2] Traditional SOA composition patterns, such as orchestration, collaboration and choreography can be supported through static integration contracts, using existing languages such as WS-BPEL. Languages for specifying more dynamic automated integration patterns may require new standardization.

- creation, query, update and deletion of abstract contracts for automated integration and
- management of policies that apply to dynamic compositions.

Dynamic composition: The dynamic composition functional component provides run-time capabilities for composing system elements, in adherence to abstract contracts for automated integration, including:

- monitoring the status of the distributed system,
- automated addition and removal of system components to a composition, in reaction to changes of system state and
- creation and deletion of links between the interfaces of composed components.

Annex A REVISION HISTORY

Revision	Date	Editor	Changes Made
V0.01	2016-08-31	Mark Crawford	Initial Release
V0.02	2018-01-10	Mark Crawford	Transition to IIC template
V0.03	2018-01-28	Mark Crawford	Update based on fine tuning of template compliance
V0.04	2018-02-28	Mark Crawford	Final Working Draft after Task Group Review & Approval
V0.05	2018-05-15	Mark Crawford	Incorporated Technical Editor Comments – ready for Working Group Review & Approval

Annex B ACRONYMS

ETL	Extract/Transform/Load
IETF	Internet Engineering Task Force
IIC	Industrial Internet Consortium
IIoT	Industrial Internet of Things
IIRA	Industrial Internet Reference Architecture
IT	Information Technology
SOA	Service Oriented Architecture

Annex C GLOSSARY

Industrial Internet Consortium (IIC)

an open membership, international not-for-profit consortium that is setting the architectural framework and direction for the Industrial Internet. Founded by AT&T, Cisco, GE, IBM and Intel in March 2014, the consortium's mission is to coordinate vast ecosystem initiatives to connect and integrate objects with people, processes and data using common architectures, interoperability and open standards.

Industrial Internet of Things (IIoT)

describes systems that connects and integrates industrial control systems with enterprise systems, business processes, and analytics.
Note 1: Industrial control systems contain sensors and actuators.
Note 2: Typically, these are large and complicated system.

Internet Engineering Task Force (IETF)

The Internet Engineering Task Force (IETF) develops and promotes voluntary *Internet standards*, in particular the standards that comprise the *Internet protocol suite* (TCP/IP). It is an open *standards organization*, with no formal membership or membership requirements.

Annex D REFERENCES

[Arkin-2009] Arkin, Ronald: Governing Lethal Behavior in Autonomous Robots, 1st Edition, 2009 May, Chapman and Hall/CRC, ISBN 978-1-420085945, retrieved 2018-01-25

[Duggal-2014] Duggal, Dave: Semantic SOA makes Sense! Dataversity, 2014-Aug-22, retrieved 2018-01-25
http://www.dataversity.net/semantic-soa-makes-sense

[IEC-61508] International Electrotechnical Commission: IEC 61508:2010 CMV, Commented version, Functional safety of electrical/electronic/programmable electronic safety-related systems, 2010, retrieved 2018-01-25
https://webstore.iec.ch/publication/22273
from
http://www.iec.ch/functionalsafety/explained/

[IETF-RFC2119] ETL Extract/Transform/Load

IETF, Bradner, S.: Key words for use in RFCs to Indicate Requirement Levels, 1997, retrieved 2016-06-29.
http://ietf.org/rfc/rfc2119.txt

[IIC-IIAF2017] IIC: The Industrial Internet, Volume T5: Analytics Framework, version 1.0, 2017-Oct-16, retrieved 2018-01-25
http://www.iiconsortium.org/industrial-analytics.htm

[IIC-IICF2017] IIC: The Industrial Internet, Volume G5: Connectivity Framework Technical Report, version 1.0, 2017-Feb-28, retrieved 2018-01-25

[IIC-IIRA2015] IIC: The Industrial Internet, Volume G1: Reference Architecture Technical Report, version 1.7, 2015-Jun-04, retrieved 2017-01-10
http://www.iiconsortium.org/IIRA-1.7.htm

[IIC-IIRA2017] IIC: The Industrial Internet, Volume G1: Reference Architecture Technical Report, version 1.8, 2017-Jan-31, retrieved 2018-01-25
http://www.iiconsortium.org/IIRA.htm

[IIC-IIoTSF2016] IIC: The Industrial Internet, Volume G4: Security Framework Technical Report, version 1.0, 2016-Sep-26, retrieved 2017-01-10
http://www.iiconsortium.org/IIoTSF

[ISO-11179-5] International Organization for Standardization: ISO/IEC 11179-5:2015, Information technology—Metadata registries (MDR) —Part 5: Naming principles, 2015 April, retrieved 2018-01-25
https://www.iso.org/standard/60341.html

[Page-2004] Page, Ernest H., Briggs, Richard, Tufarolo: Toward a Family of Maturity Models for the Simulation Interconnection Problem, 2004 January.
https://www.researchgate.net/publication/228811114_Toward_a_family_of_maturity_models_for_the_simulation_interconnection_problem, retrieved 2018-01-25
download at
https://www.researchgate.net/profile/John_Tufarolo/publication/228811114_Toward_a_family_of_maturity_models_for_the_simulation_interconnection_problem/links/0a85e531c842873fd0000000/Toward-a-family-of-maturity-models-for-the-simulation-interconnection-problem.pdf

[WKPD-COIP] Wikipedia: Conceptual interoperability, retrieved 2018-01-27
https://en.wikipedia.org/wiki/Conceptual_interoperability

[WKPD-REPL] Wikipedia: Reactive planning, retrieved 2018-01-27
https://en.wikipedia.org/wiki/Reactive_planning

[WKPD-SEOR] Wikipedia: Service-orientation, retrieved 2018-01-27
https://en.wikipedia.org/wiki/Service-orientation

[WKPD-SPAC] Wikipedia: Speech act, retrieved 2018-01-17
https://en.wikipedia.org/wiki/Speech_act

INDEX

AUTHORS AND LEGAL NOTICE

This document is a work product of the Industrial Internet Consortium Architecture Task Group, co-chaired by Mark Crawford (SAP) and Shi-Wan Lin (Thingswise).

Authors: The following persons contributed substantial written content to this document: Shi-Wan Lin (Thingswise), Mark Crawford (SAP), Bradford Miller (GE), Jacques Durand (Fujitsu), Rajive Joshi (RTI), Paul Didier (Cisco), Amine Chigani (GE), Reinier Torenbeek (RTI), David Duggal (EnterpriseWeb), Robert Martin (MITRE), Graham Bleakley (IBM), Andrew King (University of Pennsylvania), Jesus Molina (Fujitsu), Sven Schrecker (Intel), Robert Lembree (Intel), Hamed Soroush (RTI), Jason Garbis (RSA), Eric Harper (ABB), Kaveri Raman (AT&T) and Brian Witten (Symantec).

Contributors: The following persons contributed valuable ideas and feedback that significantly improved the content and quality of this document: Farooq Bari (AT&T), Tom Rutt (Fujitsu), Jack Weast (Intel), Lin Nease (HP), Ron Ambrosio (IBM), Omer Schneider (Cyber-X Labs), Pete MacKay (Wurldtech) and Lance Dover (Micron).

Technical Editor: Stephen Mellor (IIC staff) oversaw the process of organizing the contributions of the above Authors, Editors and Contributors into an integrated document.
