ISSN: 3068-1332

Research Article

Journal of Applied Language Learning

From Prompt-Response to Pulse-Driven Intentions: The Intention Pulse Transfer Protocol (IPTP) For Release and Absorption of Pulses

Pronab Pal*

IntentixLab Keybyte Systems, Melbourne, Australia June 2025

***Corresponding Author**
Pronab Pal, IntentixLab Keybyte Systems, Melbourne, Australia.
E-Mail: pronab@keybytesystems.com

Submitted: 2025, Jul 16; **Accepted:** 2025, Aug 22; **Published:** 2025, Aug 26

Citation: Pal, P. (2025). From Prompt-Response to Pulse-Driven Intentions: The Intention Pulse Transfer Protocol (IPTP) For Release and Absorption of Pulses. *J App Lang Lea, 2*(2), 01-08.

Abstract
Building upon our previous work on Prompt-Response (PnR) Computing, this paper introduces a refined computational model that separates logical state from data payload through the concepts of Pulse and Response. We present the Intention Pulse Transfer Protocol (IPTP), an asynchronous communication protocol that enables distributed coordination of semantic states across cloud-native environments. This evolution addresses the conceptual coupling limitations of the original PnR model while maintaining its core benefits of semantic traceability and business logic alignment. The paper demonstrates how Pulses serve as declarative truth conditions, Responses carry structured payloads, and Intentions act as field-gated activation mechanisms. A practical NodeJS implementation illustrates the protocol's application in user authentication scenarios, showing how IPTP can coexist with existing client-server architectures while providing enhanced semantic coordination capabilities.

Keywords: Intention Space, Pulse-Signal Architecture, Distributed Systems, Semantic Computing, Cloud-Native Applications

1. Introduction: Revisiting the Prompt-Response Model

In our previous work on Prompt-Response (PnR) Computing, we introduced a model for capturing runtime context and state using Prompt-Response pairs encoded as trivalent JSON objects [1]. This approach offered a unified way to align business logic with application behavior, addressing the foundational gap in software engineering regarding the representation and management of human intentions as first-class citizens in code [2].

However, our practical implementation and system testing revealed a conceptual limitation: the tight coupling of truth-state (logic) and payload (data) within single PnR entities. This monolithic structure hindered composability and created challenges in distributed environments where semantic state and data payloads may need to travel different paths or be processed by different components.

This paper presents a refined model that separates these concerns into two distinct elements: Pulse and Response. Simultaneously, we elevate Intention to become the central unit of control and communication. This foundational separation offers improved testability, composability, and declarative flow control particularly important in cloud-native and Large Language Model (LLM)-integrated systems where semantic understanding and data processing often occur at different scales and locations.

The contributions of this work include:
1. A refined computational model separating logical state from data payload
2. The Intention Pulse Transfer Protocol (IPTP) for asynchronous semantic coordination
3. A practical demonstration of field-based activation semantics
4. Integration patterns for existing cloud infrastructure

2. Theoretical Foundation: Pulse as Logical Primitive

2.1. Pulse: The Atomic Unit of Semantic State

A Pulse represents the smallest semantic signal unit in our computational model. It expresses a truth condition using trivalent logic: Yes (Y), No (N), or Undecided (U). This design draws from three-valued logic systems and Speech Act Theory, where utterances carry both propositional content and performative force [3,4].

Unlike traditional boolean variables that exist in program memory, Pulses are field-addressable semantic statements that act as declarative filters or triggers for subsequent operations. In Intention Space, we recognize the existence of a Pulse as a possible state around an observation, reference, or result. The trivalent nature allows it to carry data in the form of Response while contributing to computational logic in any of these states.

Formally, a Pulse can be defined as:
Pulse: = (name: String, TV: {Y, N, U})
Where name is a natural language phrase expressing a semantic condition, and TV represents the trivalent state.

2.2. Response: Structured Payload Decoupled from Logic

While a Pulse represents what is known, a Response conveys what is produced or referenced. A Response is optionally attached to a Pulse and may carry structured data, computational results, or references to external resources.

This separation of concerns aligns with established patterns in distributed systems where control plane and data plane operations are handled differently [5]. It enables different Design Chunks to process or reuse responses based on pulse truth values without being coupled to specific data formats or processing requirements.

The representation of Results as separate from the Pulses carrying them has profound implications for how we instruct computational systems. Traditional computing instructs machines in terms of data values and control flow, while Intention Space instructs through semantic Pulses that carry business meaning.

2.3. Field of Pulses: Bounded Semantic Collections

A Field is defined as a bounded collection of Pulses that together represent a coherent semantic state. Collections of Pulses always maintain a subjective stance as assemblages of semantic statements. Intention Space treats computation as a journey through the transformation of these subjective spaces.This concept extends recent work on context-aware computing and semantic web technologies, providing a runtime representation that maintains semantic coherence while enabling distributed processing [6,7].

3. Intention: Declarative Communication Handle

An Intention is a named semantic action that serves as both a declarative test and a routing envelope. It is triggered only when its associated signalâ €" a set of required Pulsesâ €" matches the current field state. This design ensures that all communications in the system are semantically grounded and contextually appropriate.

3.1. Asynchronous Signal Composition

Although at design time an Intention carries a specified set of Pulses, we treat Design Node participation in Intention Space as an asynchronous process. This allows Pulses to be transmitted incrementally, enabling graceful handling of network partitions, partial failures, and progressive data collection scenarios common in distributed systems [8].

For example, in a typical authentication scenario, the Intention" Log me in" may be designed to carry the Pulses:

1. {'user name present': 'Y'}
2. {'password present': 'Y'}
3. {'User Responded': 'Y'}

Each Pulse carries its respective reference data and can be released from the user device over a spread of time, while the target object accumulates these signals and reflects the Intention only when the trigger activation condition is fulfilled.

4. The Intention Pulse Transfer Protocol (IPTP)

4.1. Protocol Architecture

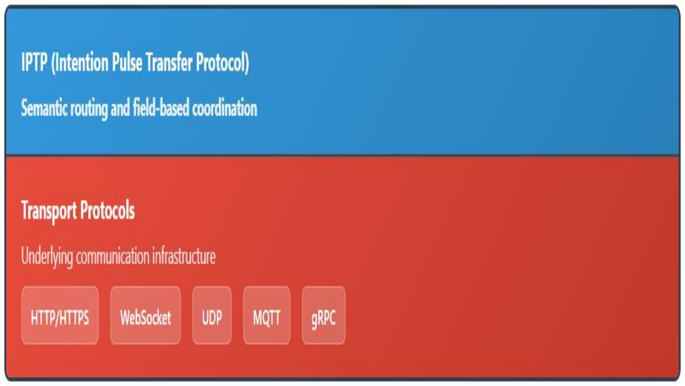

IPTP (Intention Pulse Transfer Protocol)
Semantic routing and field-based coordination
Transport Protocols
Underlying communication infrastructure
HTTP/HTTPS
WebSocket
UDP
MQTT
gRPC

Figure 1: IPTP Protocol Stack

The Intention Pulse Transfer Protocol (IPTP) provides asynchronous communication of semantic triggers over existing network infrastructure. Unlike traditional protocols where both control and data are embedded in the payload, IPTP introduces a clear separation of concerns:

Intentions : Carried in protocol headers for declarative routing
Signals : Transmitted as structured payload for semantic processing

This architectural decision enables objects to handle intention routing declaratively while processing signals through field-based semantic operations. IPTP operates above transport protocols [Figure 1], enabling semantic coordination through intention-based routing with signals as structured payload.

Traditional Port-to-port communication is stateful, not fixed: In TCP, a connection is identified by a 4-tuple (source IP, source port, destination IP, destination port) . Only the destination components must be predefined—the destination IP (10.0.0.50) and listening port (443) are fixed. The source components are dynamically allocated: while the source IP (192.168.1.100) identifies the client machine, thesource port (54321) is ephemeral —allocated by the OS and discarded after the connection closes.

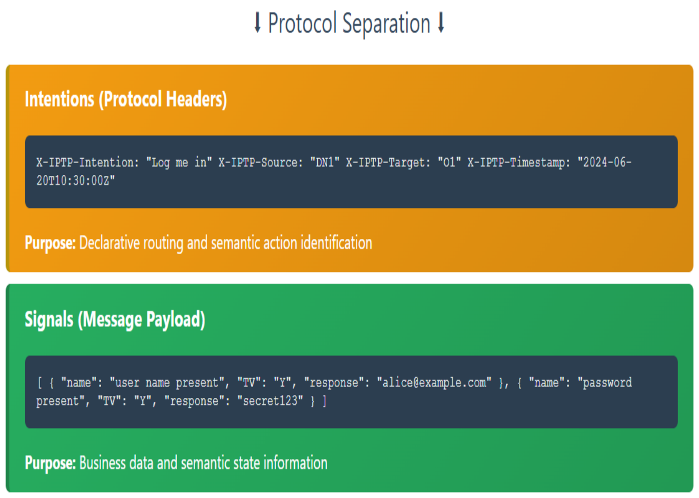
Thus traditional port-to-port communication, while stateful during connection lifetime, provides no persistent expression of business intent. Source ports ,being ephemeral—dynamically allocated and discarded—leaving no trace of the semantic purpose behind machine-to-machine interactions. This creates a fundamental gap:**the computational "what" (data flow) is divorced from the business "why" (intent)** . IPTP bridges this gap by embedding intentions in protocol headers, creating persistent semantic identity that survives beyond individual connections. Where traditional networking provides transient 4-tuples, IPTP provides**semantically grounded communication paths** where business understanding coincides with execution state throughout the interaction lifecycle.

4.2. IPTP Message Structure

An IPTP message consists of:

Protocol Headers:
X-IPTP-Intention: ”Log me in”
X-IPTP-Source: ”DN1”
X-IPTP-Target: ”O1”
X-IPTP-Timestamp: ”2024-06-20T10:30:00Z”

Payload (Signal):
[
{”name”: ”user name present”, ”TV”: ”Y”, ”response”: ”alice@example.com”},
{”name”: ”password present”, ”TV”: ”Y”, ”response”: ”secret123”}
]



```
X-IPTP-Intention: "Log me in" X-IPTP-Source: "DN1" X-IPTP-Target: "O1" X-IPTP-Timestamp: "2024-06-20T10:30:00Z"
```

Purpose: Declarative routing and semantic action identification

```
[ { "name": "user name present", "TV": "Y", "response": "alice@example.com" }, { "name": "password present", "TV": "Y", "response": "secret123" } ]
```

Purpose: Business data and semantic state information

Figure 2: Intentions traverse carrying signals

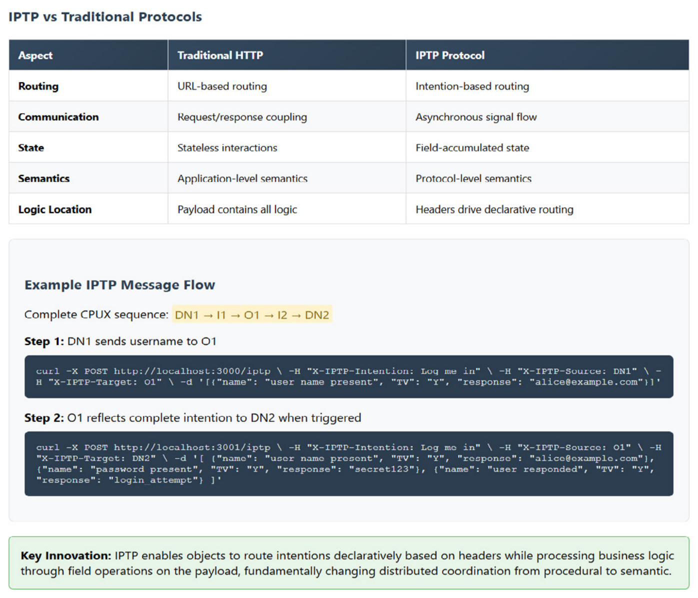
The key innovation shown in Figure 2 is the separation of control (intentions in headers) from data (signals in payload), enabling declarative routing based on semantic Pulses rather than procedural logic.

4.3. Declarative Intention Mapping

Objects register intention mappings that define how to handle incoming intentions:
{
incomingIntention: ”Log me in”,
triggerCondition: [
{ name: ”user responded”, TV: ”Y” }
],
outgoingIntention: ”Log me in”,
targetNode: ”DN2”
}
When an IPTP message arrives, the object:
1. Routes declaratively based on the intention header
2. Absorbs the signal into its field state
3. Evaluates trigger conditions against current field
4. Reflects intentions when conditions are satisfied

4.4. Protocol Semantics vs Traditional Networking

IPTP maintains business understanding while traditional protocol builds machine connections. [Figure: 3]


IPTP vs Traditional Protocols

Aspect	Traditional HTTP	IPTP Protocol
Routing	URL-based routing	Intention-based routing
Communication	Request/response coupling	Asynchronous signal flow
State	Stateless interactions	Field-accumulated state
Semantics	Application-level semantics	Protocol-level semantics
Logic Location	Payload contains all logic	Headers drive declarative routing

Example IPTP Message Flow

Complete CPUX sequence: DN1 → I1 → O1 → I2 → DN2

Step 1: DN1 sends username to O1

```
curl -X POST http://localhost:3000/iptp \ -H "X-IPTP-Intention: Log me in" \ -H "X-IPTP-Source: DN1" \ -
H "X-IPTP-Target: O1" \ -d '[{"name": "user name present", "TV": "Y", "response": "alice@example.com"}]'
```

Step 2: O1 reflects complete intention to DN2 when triggered

```
curl -X POST http://localhost:3001/iptp \ -H "X-IPTP-Intention: Log me in" \ -H "X-IPTP-Source: O1" \ -H
"X-IPTP-Target: DN2" \ -d '[ {"name": "user name present", "TV": "Y", "response": "alice@example.com"},
{"name": "password present", "TV": "Y", "response": "secret123"}, {"name": "user responded", "TV": "Y",
"response": "login_attempt"} ]'
```

Key Innovation: IPTP enables objects to route intentions declaratively based on headers while processing business logic through field operations on the payload, fundamentally changing distributed coordination from procedural to semantic.

Figure 3: IPTP: A Semantic Protocol

4.5. Field Operations in IPTP Context

Design-Time Guarantees
By enforcing that:

• All Intentions carry known signal definitions
• All valid signals are pre-registered in the SignalRegistry

…we ensure runtime environments never need to perform semantic signal reconstruction or ambiguity resolution. This directly contributes to:

• Scalability across distributed CPUX instances
• Predictable latency in field-based matching
• Auditability via hash-based tracing
Algorithmic Summary
Given:

```
type Signal struct {
  Pulses []Pulse
  Hash   string
}
```

A Signal's hash is computed as:

```
GenerateSignalHash(signal Signal) string {
  // Normalize names, sort by name + TV, then hash the string
}
```

At runtime:

```
DoesSignalMatchFieldByHash(signal, field) {
  return GenerateFieldHash(fieldSubset) == signal.Hash
}
```

Motivation and Computational Benefit

Without signal hashing, runtime signal matching would require recursive set comparisons between Pulses in the incoming signal and the current field. This is computationally expensive (O(n log n) worst case for sorting-based matching).

With hashing:
• Field states are transformed into field hashes
• Incoming Signals carry their design-time precomputed hash
• Field matching becomes a hash equality check, reducing runtime complexity to O(1)

In Intention Space, each Signal—defined as a subset of trivalent Pulses—is assigned a unique design-time hash. This hash is computed using a deterministic algorithm that normalizes Pulse names, sorts them by name and TV, and concatenates them into a canonical string before hashing with MD5. This ensures:

• All valid Signals in an Intention Space are enumerable and finite
• Each Signal's semantic signature can be validated with constant-time comparisons at runtime

4.6. Signal Hashing and O(1) Field Matching

The separation of intentions and signals enables two clean field operations:

FieldAbsorb(signal): Processes payload independently of routing concerns, integrating pulses into semantic field state.

FieldTrigger(intention, conditions) : Evaluates header-specified intentions against field conditions declaratively, without procedural logic.

This separation allows objects to be configured with intention mappings at design time while processing signals dynamically at runtime.

While most implementations compute runtime field hashes for subsets corresponding to known signals, Intention Space also supports a stronger optimization. When all relevant field states can be enumerated at design time, their full-field hashes can be precomputed and stored as valid match conditions.

This eliminates the need for extracting pulse subsets at runtime. Instead, the runtime system:
1. Computes a hash of the entire current field
2. Looks up this hash in a precomputed map:
fieldHash → [triggeredIntentions]

This yields pure O(1) field-triggered activation, transforming the signal matching problem into a single hash lookup. This approach is especially useful in CPUX designs with finite and enumerable semantic states.

Execution Semantics: Field-Gated Activation

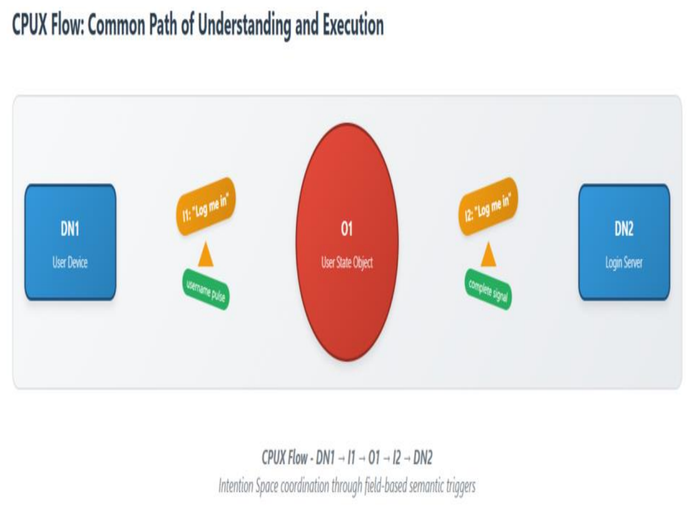
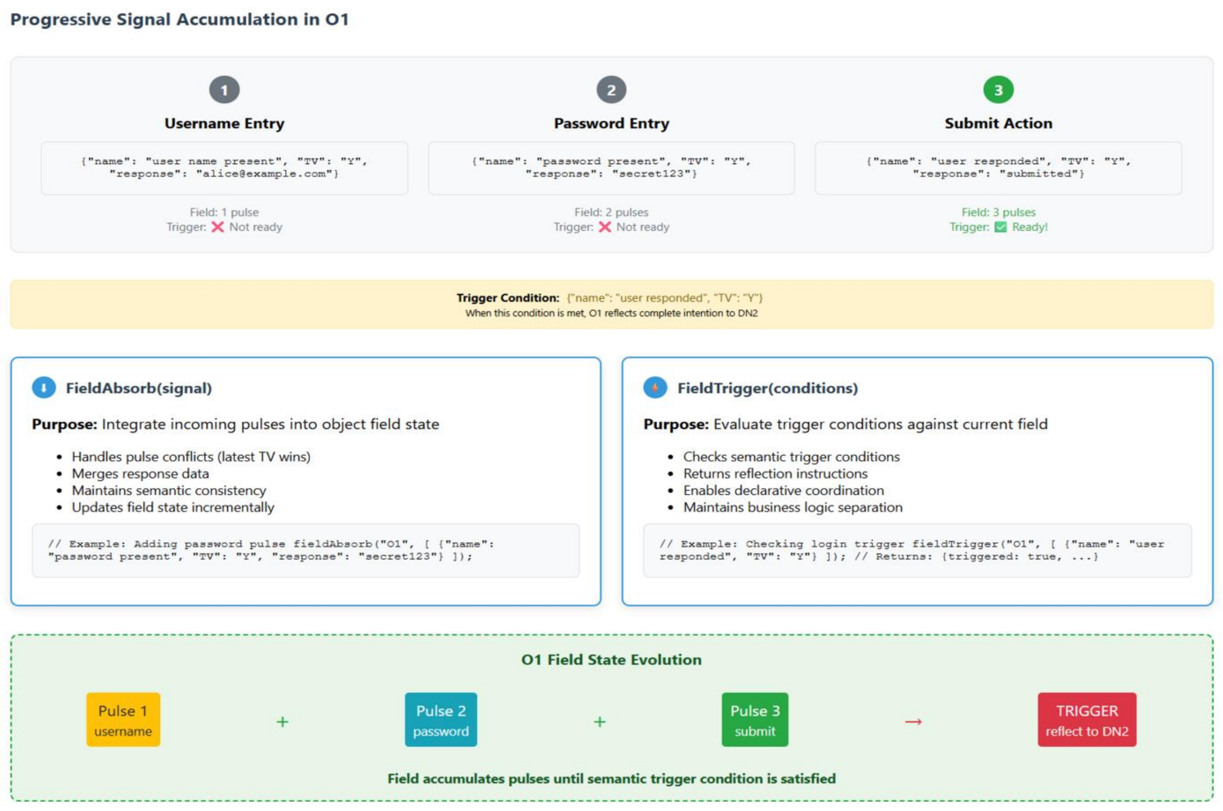

CPUX Flow: Common Path of Understanding and Execution
DN1
User Device
I1: "Log me in"
username pulse
O1
User State Object
I2: "Log me in"
complete signal
DN2
Login Server
CPUX Flow - DN1 → I1 → O1 → I2 → DN2
Intention Space coordination through field-based semantic triggers

Figure 4: Intention Space operates on the foundational principle that what is computationally executed remains directly aligned with business understanding throughout the process. A single unit of this alignment is a Common Path of Understanding and Execution (CPUX) (Fig 4)—a design-time sequence that ensures each execution step maintains direct correspondence between computational state and business understanding, whether within a single machine or The existence of Fields within CPUX sequences and Object states enables trigger mechanisms to be implemented declaratively, where semantic conditions determine execution flow rather than procedural logic.(Figure 5) . Each CPUX sequence maintains a unique identity through the constraint that no two sequences can share the same 5-tuple pattern (DN1-I1-O1-I2-DN2) within a domain. This uniqueness rule ensures that every computational path is distinctly identifiable and traceable, enabling precise state management and preventing ambiguous execution flows distributed across multiple machines.



```
{"name": "user name present", "TV": "Y",
"response": "alice@example.com"}
```

```
{"name": "password present", "TV": "Y",
"response": "secret123"}
```

```
{"name": "user responded", "TV": "Y",
"response": "submitted"}
```

Trigger Condition: {"name": "user responded", "TV": "Y"}
When this condition is met, O1 reflects complete intention to DN2

FieldAbsorb(signal)

Purpose: Integrate incoming pulses into object field state

- Handles pulse conflicts (latest TV wins)
- Merges response data
- Maintains semantic consistency
- Updates field state incrementally

```
// Example: Adding password pulse fieldAbsorb("O1", [ {"name":
"password present", "TV": "Y", "response": "secret123"} ]);
```

FieldTrigger(conditions)

Purpose: Evaluate trigger conditions against current field

- Checks semantic trigger conditions
- Returns reflection instructions
- Enables declarative coordination
- Maintains business logic separation

```
// Example: Checking login trigger fieldTrigger("O1", [ {"name": "user
responded", "TV": "Y"} ]); // Returns: {triggered: true, ...}
```

Figure 5: Object as State Accumulator

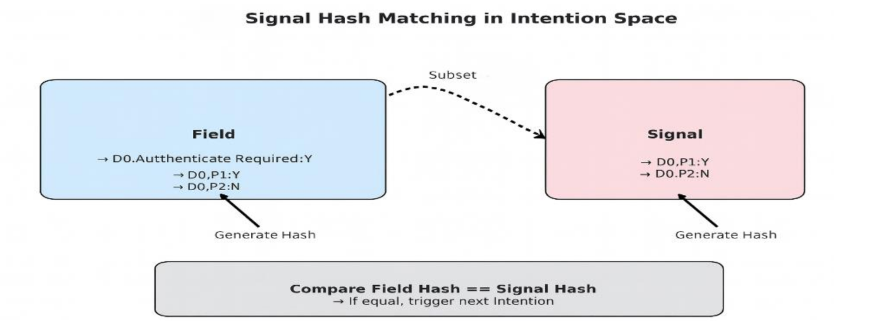

Signal Hash Matching in Intention Space
Subset
Field
→ D0.Autthenticate Required:Y
→ D0,P1:Y
→ D0,P2:N
Signal
→ D0,P1:Y
→ D0.P2:N
Generate Hash
Generate Hash
Compare Field Hash == Signal Hash
→ If equal, trigger next Intention

Figure 6: In Intention Space, execution follows a pull-based model rather than traditional push-based function calls. The Intention Loop waits until all required signal Pulses match the current field state, then activates the next Design Chunk. This approach provides several advantages:

1. Prevents premature computation by ensuring all preconditions are satisfied
2. Creates runtime traceability through explicit state transitions
3. Enables automatic test generation from signal maps and field states
4. Supports graceful degradation in distributed failure scenarios

5.1. Object Reflection Mechanism

Objects in Intention Space serve as semantic state machines that reflect Intentions based on field conditions. When an Object receives Pulses through an incoming Intention,[Figure 4] it:

1. Absorbs the Pulses into its internal field using FieldAbsorb
2. Evaluates trigger conditions using FieldTrigger for each potential outgoing Intention [Figure 5]
3. Reflects appropriate Intentions when trigger conditions are satisfied [Figure 6]
4. Maintains state consistency across partial signal arrivals

This mechanism enables sophisticated coordination patterns while maintaining the declarative nature of the computational model.

6. Implementation and Evaluation

6.1. NodeJS Reference Implementation

We have implemented a reference Object server in NodeJS that demonstrates the core IPTP concepts. The implementation handles the user authentication scenario described in Section 3.1, showing how traditional client-server patterns can be enhanced with semantic coordination. The implementation validates several key properties:

1. Asynchronous pulse accumulation maintains semantic consistency
2. Field matching provides reliable trigger activation
3. Integration with existing HTTP infrastructure requires minimal changes
4. Performance overhead is acceptable for typical cloud workloads

6.2. Integration with Existing Infrastructure

IPTP is designed to coexist with existing cloud infrastructure rather than replace it. Design Nodes can be implemented as:

1. Microservices with IPTP-aware interfaces
2. Serverless functions triggered by field conditions
3. Traditional web applications with semantic middleware
4. Container orchestrations with intention-based scheduling

7. Related Work and Positioning

Our work builds upon several established research areas while introducing novel semantic coordination mechanisms: Event-Driven Architectures: IPTP extends event-driven patterns by adding semantic matching and trivalent logic to event processing [9]. Dataflow Programming: The field-based activation model shares concepts with dataflow systems but operates at the semantic rather than data level [10]. Intent-Based Networking: Recent work on intent-based network management provides similar declarative approaches at the infrastructure level [11]. Semantic Web Services: W3C semantic web service specifications address similar semantic coordination challenges but focus on service description rather than runtime coordination [12].

8. Future Work and Implications

This Pulse-Intention model establishes the foundation for several promising research directions:

8.1. LLM Integration

The natural language basis of Pulses enables direct integration with Large Language Models for dynamic Intent generation and semantic validation.

8.2. Distributed Testing

Field matching provides inherent test scaffolding, enabling property-based testing of distributed systems at the semantic level [13].

8.3. Declarative Orchestration

CPUX sequences could serve as declarative specifications for cloud orchestration, similar to Kubernetes manifests but with semantic grounding.

9. Conclusion

The separation of Pulse and Response, combined with the Intention Pulse Transfer Protocol, provides a practical foundation for semantic coordination in distributed systems. By maintaining the benefits of our original PnR model while addressing its compositional limitations, this work opens new possibilities for cloud-native applications that can reason about their own behavior. The key insight is that semantic state and data payload can be productively separated while maintaining their conceptual relationship through field-based coordination. This enables both human-understandable business logic and efficient distributed processing within the same computational framework. Future work will explore the implications of this model for automated testing, LLM integration, and declarative cloud orchestration, building toward more intelligent and adaptable distributed systems.

References

1. Pal, P. (2025). From Prompt-Response to Pulse-Driven Intentions: The Intention Pulse Transfer Protocol (IPTP) for Release and Absorption of Pulses. *Authorea Preprints.*
2. Pal, P. (2024). Human Intention Space-Natural Language Phrase Driven Approach to Place Social Computing Interaction in A Designed Space. *Authorea Preprints.*
3. Kleene, S. C. (1952). Introduction to metamathematics.
4. Austin, J. L. (1975). *How to do things with words*. Harvard university press.
5. Kurose, J. F., & Ross, K. W. (2019). *Computer networking: A top-down approach* (pp. 607967-5). Harlow, England Boston: Pearson.
6. Dey, A. K. (2001). Understanding and using context. *Personal and ubiquitous computing, 5*(1), 4-7.
7. Lee, T. B., Hendler, J., & Lassila, O. (2001). The semantic web. *Scientific american, 284*(5), 34-43.
8. Lamport, L. (2019). Time, clocks, and the ordering of events in a distributed system. In *Concurrency: the Works of Leslie Lamport* (pp. 179-196).
9. Hohpe, G., & Woolf, B. (2004). *Enterprise integration patterns: Designing, building, and deploying messaging solutions.* Addison-Wesley Professional.
10. Dennis, J. B. (2005, June). First version of a data flow procedure language. In *Programming Symposium: Proceedings, Colloque sur la Programmation Paris, April 9–11, 1974* (pp. 362-376). Berlin, Heidelberg: Springer Berlin Heidelberg.
11. Silvander, J. Wnuk, K. and M. Svahnberg, M. (2020).” Systematic Literature Review on Intentâ€?Driven Systems,” *IET Software*, vol. 14, no. 4, pp. 345-357.
12. Martin, D., Burstein, M., Hobbs, J., Lassila, O., McDermott, D., McIlraith, S., ... & Sycara, K. (2004). OWL-S: Semantic markup for web services. *W3C member submission, 22*(4).
13. Claessen, K., & Hughes, J. (2011). QuickCheck: a lightweight tool for random testing of Haskell programs. *Acm sigplan notices, 46*(4), 53-64.
14. Pal, P. (2025). "Design-Time Signal Hashing in Intention Space for O(1) Runtime Matching," Technical Note, Intentix-Lab. Available on request.

Appendix: NodeJS Implementation

This appendix provides a complete NodeJS implementation of the IPTP Object Server, demonstrating the FieldAbsorb and FieldTrigger functions as described in Section 5.1. The implementation shows how the user login scenario operates with incremental pulse transmission and field-based activation.

A.1. Core Field Operations

The implementation centers on two fundamental operations:FieldAbsorb Function : Integrates incoming pulses into an object's field state, handling pulse conflicts by maintaining the latest trivalent state and merging response data when multiple values exist for the same pulse name. FieldTrigger Function : Evaluates whether the current field state satisfies the conditions for reflecting a specific Intention. Returns reflection instructions when all required pulses match their expected trivalent values.

A.2. Login Scenario Implementation

The implementation demonstrates the CPUX sequence: DN1 (user device) -> I1 -> O1 (user state) -> I2 -> DN2 (login server) The Object O1 is configured with trigger definitions that specify:
1. Intention name: ”Log me in”
2. Required signal: [{name: ”user responded”, TV: ”Y”}]
3. Target Design Node: ”DN2” (login server)

When the trigger condition is satisfied, O1 reflects the complete signal containing all accumulated user credentials to DN2.

A.3. IPTP Protocol Implementation

The implementation [full source code available in the repository: https://github.com/spicecoder/iptp_paper/blob/main/iptpserver.js] demonstrates proper protocol separation with intentions in headers and signals as payload:
/ * IPTP Protocol Endpoint *Intentions in headers,signal with payload */

app.post(’/iptp’, (req, res) => { // Extract IPTP protocol headers const intention = req.headers[’x-iptp-intention’]; const source = req.headers[’x-iptp-source’]; const target = req.headers[’x-iptp-target’];
// Signal is the request body (payload)
const signal = req.body;
// Route declaratively based on intention header
const result = processIPTPMessage(intention, source, target, signal); });
/** * Declarative intention mapping in objects */
const intentionMappings = [{ incomingIntention: ”Log me in”, triggerCondition: [{ name: ”user responded”, TV: ”Y” }], outgoingIntention: ”Log me in”, targetNode: ”DN2” }];

For test : . curl -X POST http://localhost:3000/iptp/intention -H ”Content-Type: application/json” -d ’{ ”intention”: ”Log me in”, ”source”: ”DN1”, ”target”: ”O1”, ”signal”: [{”name”: ”user responded”, ”TV”: ”Y”, ”response”: ”submitted”}] }**

The final request triggers the FieldTrigger condition, causing O1 to reflect the ”Log me in” intention to DN2 with the complete user credentials signal. This implementation demonstrates how IPTP enables asynchronous, semantic coordination between distributed components while maintaining field-based activation semantics as described in the main paper.
A.4 Proper IPTP Usage Example

Step 1: Username Entry

```
curl -X POST http://localhost:3000/iptp \
-H ”X-IPTP-Intention: Log me in” \
-H ”X-IPTP-Source: DN1” \
-H ”X-IPTP-Target: O1” \
-H ”Content-Type: application/json” \
-d ’[{”name”: ”user name present”, ”TV”: ”Y”, ”response”: ”alice@example.com”}]’
```

Step 2: Password Entry

```
curl -X POST http://localhost:3000/iptp \
-H ”X-IPTP-Intention: Log me in” \
-H ”X-IPTP-Source: DN1” \
-H ”X-IPTP-Target: O1” \
-H ”Content-Type: application/json” \
-d '[{"name": "password present", "TV": "Y", "response": "secret123"}]'
```

Step 3: Submit Trigger

```
curl -X POST http://localhost:3000/iptp \
-H "X-IPTP-Intention: Log me in" \
-H "X-IPTP-Source: DN1" \
-H "X-IPTP-Target: O1" \
-H "Content-Type: application/json" \
-d '[{"name": "user responded", "TV": "Y", "response": "submitted"}]'
```

DN2 Processing: When O1 reflects the intention to DN2, it sends:

```
# O1 -> DN2 via IPTP
curl -X POST http://localhost:3001/iptp \
-H "X-IPTP-Intention: Log me in" \
-H "X-IPTP-Source: O1" \
-H "X-IPTP-Target: DN2" \
-H "Content-Type: application/json" \
-d '[
{"name": "user name present", "TV": "Y", "response": "alice@example.com"},
{"name": "password present", "TV": "Y", "response": "secret123"},
{"name": "user responded", "TV": "Y", "response": "login_attempt"}
]'
```

A.5. Key Protocol Benefits

This proper IPTP implementation demonstrates:
1. Declarative Routing : Objects handle intentions based on headers without hardcoded logic
2. Semantic Processing : Signals processed through field operations independent of routing
3. Protocol Separation : Clear distinction between control (headers) and data (payload)
4. Design-Time Configuration : Intention mappings defined declaratively, not procedurally
5. Runtime Flexibility : Same object can handle multiple intentions through different mappings

The separation enables objects to be semantically aware routers that process business logic through field conditions rather than procedural code, fundamentally changing how distributed coordination is achieved