PROMPT-RESPONSE (PNR) CLOUD COMPUTING -INTENTIONS AND CONTEXT

Pronab Pal

Keybyte Systems, Melbourne, Australia

ABSTRACT

In today's cloud-native multi-core environments, the deployment and runtime performance of applications often take precedence over the visibility and agility of business logic. While this prioritization ensures optimal responsiveness, it can inadvertently create barriers to real-time analysis and rapid business adaptability. Traditional approaches necessitate reproducing issues in development environments and implementing additional tracing mechanisms, leading to delays in problem resolution and business evolution.

This paper introduces the Prompt and Response (PnR) computing model. This paradigm shift addresses these challenges by maintaining a clear representation of intention flow and object data throughout the application lifecycle. The PnR system enables real-time analysis and intelligence derivation in production environments, transcending the limitations of container boundaries and module isolation. By representing every input and result of each module associated with distinct intentions within the PnR framework, we create a unified and traceable computational space called Intention Space.

This approach allows for precise identification and analysis of specific modules referred to as just 'Design Chunks', regardless of their distribution across single or multiple containers or boundaries. We explore the architectural patterns of PnR transformations, illustrating how this model aligns with and extends current computing paradigms while offering a more flexible and transparent approach to managing complex, distributed systems.

This paper aims to provide a computational foundation for implementing PnR systems, paving the way for more adaptable, analysable, and efficient cloud-native applications.

KEYWORDS

Prompt and Response, Designchunk, Response, Designchunk, Intentions, Objects, Cross Container Consistency, Input process identification, Output process Identification, Execution State, Identification, Execution State, Common Path Of Execution and Understanding, Intention Loop, Spaceloop, Intention Emission-Reflection-Absorbtion

1. INTRODUCTION

Traditional computing models often struggle to integrate persistent storage and seamlessly manage context and state across various components and persistency modes within software at execution time. In the traditional approach, development time is viewed as a preparation for and detached from runtime, making it difficult to map into a run-time model at development time. The Prompt Response Computing (PnR) computing model addresses these challenges by

David C. Wyld et al. (Eds): CAIML, ICAIT, ITCSE, CRYPIS, NLCA – 2024
pp. 65-82, 2024. - CS & IT - CSCP 2024 DOI: 10.5121/csit.2024.141806

providing a unified framework encompassing transient and persistent data and media assets, facilitating a more coherent and efficient computational paradigm.

PnR computing builds the foundation of 'Intention Space', which we introduced in a previous publication [1]. Intention Space elevates coding as a design act and makes software execution intent-driven within the boundary of the Design Chunks. This allows the treatment of any custom art, media, and code as part of designs in Design Chunks. At the same time, multiple Design Chunks are joined through intentions at execution time.

In this paper, we shall give a formal description of the computational space created to show this. We shall restate and define those concepts first and build a formal model of PnR computing so that it captures the entire computational foundation for building Apps in any language through some transformation on the set of selected PnRs.

We shall illustrate the cognitive foundation with a small example in Golang. We shall also explore the underlying rationality behind the existence of PnR from the perspective of internet maturity over the last decade and the power of multi-core processors. Finally, we shall discuss the range of scenarios where PnR can be applied effectively.

2. DEFINITIONS

An Intention Space, as introduced in [1], can have a name identifying it. An intention space may have one or more Domains, each identified by a unique name. A Domain has a set of prompt responses (PnRs), represented as JSON, where the JSON name acts as the prompt or a question and whose value is a response or an answer, paired by a trivalence True/False/Undecided. Generally, can be represented as {"q.." :["aa","Y/N/U"]}

2.1. PnR: The Pharse-Pair-Truths & Dynamic Logic

A PnR as a JSON , has a name and value. In Intention Space, a PnR is treated as a template with unique names within a domain, both at design time and runtime - At any stage in a defined computation process, a PnR can also be seen as either true or false or undecided regardless of the value carried by the individual PnR.

While each PnR is identifiable by the "name" portion of the JSON object in the PnR,the value portion has two parts in an array; the first element of which is treated as an answer to the name, treated as a question, and the second element which is a trivalence,can have a value 'Y' (for 'yes') , "N" (for 'no') or 'U' (for 'Undecided'), while if the trivalence portion is not present , it is assumed as 'Y' ; For example, in a scenario where a visitor is served a food dish from the menu, the PnRs might look like this:

{ "what is your name": ["_",'U'] }, { "what is your name": ["panda"]} , "do you have veg or non veg": ["veg"] }, { "dinner for": [["panda" , "dish selected,"Vegetable-Biryani"]}, { "dish made": ["Vegetable-Biryani","N"] }, { ""dish presented": ["Vegetable-Biryani","U"]}

Thus, the PnR can be equivalent to parameter values in traditional computation and the contextual pre/post true /false conditions of a particular computation unit or comprehension unit the Design chunk represents. This duality allows the PnRs to be represented by the business and the programming discipline in a contextually congruent fashion.

Also, we shall see that this association of the trivalence with the value makes it possible for PnR data to drive logic (e.g., if the condition or for loop) at execution time rather than forcing it to be hardcoded in code at design time. Thus, dynamic control over data flow and operations allows a Design chunk to work as an independent module that is not bound to a particular logic statement.

2.2. The App

An App in Intention Space is a bunch of modules (i.e. Design Chunks) that bring in some actionable functions together to cater for some real-world situations. However, the reach or scope of an App is quite limited within a small portion of the Domain. E.g if ' my Health' is a Domain, My fitness App and My Medical records App are within the 'my Health' Domain. In Intention Space, multiple apps within a single domain share the domain PnRs.

2.2.1. Domain, Object, Intention, Design Chunks

A Domain can house several Apps. The components of the Apps are called the design chunks, and the App works by being driven by "Intentions". Intentions are uniquely identifiable strings in a domain, as are the Design Chunks. The Intention is either 'absorbed' by the Design chunks or 'emitted' by Design chunks to be received by Objects in the Domain. An object within a Domain in Intention Space is identifiable by its unique name string in the Domain. Objects can 'reflect' Intentions 'received' from other Design chunks or Objects, which other Design chunks or Objects can 'absorb' or 'receive' respectively. Thus the model brings the following verbs in place :
i.emits, ii. receive, iii. reflect, iv. absorbs; these verbs transform the 'metaval state' of PnRs while setting different responses in selected PnR sets.

Design chunks in Intention Space can hold any content, notes, media image, video, or functional code. Different Apps can share design chunks in the same Domain. An Object can hold and persist PnRs. Intentions carry PnR sets as payload between Design Chunks and Objects.
We follow a general diagramming convention: a Design chunk is a box, the object is an ellipse, and an Intention is a diamond, as in Fig 1.

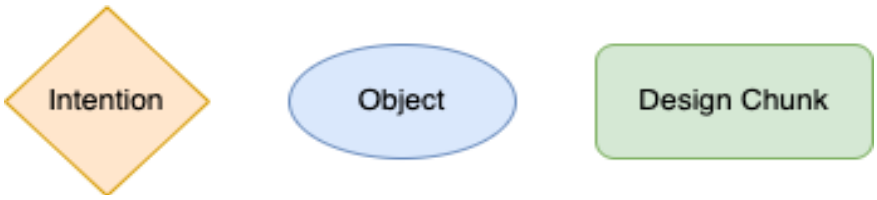

Fig 1. The Foundational operators

.

2.3. Basic Computation Model

Building on those Design chunks (DC), Intentions(I), and Objects (O), Intention Space builds a model of the unit of computation that uses Intention as a mechanism for transforming input conditions as expressed through a certain set of PnR attached to the design chunks. The smallest unit of computation is thus the sequence DC-I-O-I-DC, also referred to as the Link, the smallest Common Path of Execution and Understanding (CPUX). CPUX maintains the cognitive integrity of a sequence of Links that represents a sequence of PnR transformation, thus giving a business meaning like 'make a payment for a pair of shoes xyz'.

2.3.1. The 5-Tuple Unique Rule

The inherent data integrity maintained by Intention Space is that there can not be a duplicate of the 5-tuples DC-I-O-I-DC across all the CPUX in the domain. This means that the interim state of any computation is maintained for each tuple, so each object in the CPUX maintains the states and represents the mapping that goes with one Intention and its reflection between two Design Chunks irrespective of the computational characteristics of those Design Chunks. The reason for the uniqueness rule is that it makes each CPUX a unique string amongst all possible CPUX in the domain.

2.3.2. Intention are Carriers of PnR's Interaction

The computation thus progresses through intentions carrying the transformed PnR from one design chunk to another with the help of intermediate objects that work as the reference point for holding a state that the intentions act with. The objects also provide a point where the PNRs can be mapped and carried into new intentions, thereby holding the functions of mapping of PnRs attached to one intention to another set of PnRs, separate from the procedural computation that happens within Design chunks, as illustrated in Fig 2. This distinction is important to allow the Design chunks to have any procedural and interactive or I/O-based computation without any limitation from Intention Space. This allows the creation of Synchronicity Sets at design time as explained in Section 4.1. It also allows the configuration where the single Design Chunk holds both the starting set of PnR and the transformed set of PnR, as in Fig 3.

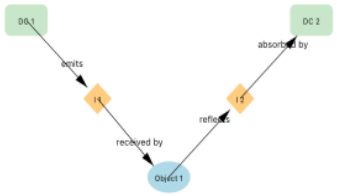

Fig 2. A Link , a smallest unit of extendable computation

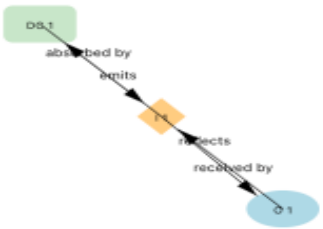

Fig 3. A closed computation by a Designchunk

2.4. CPUX: Common Path of Understanding and Execution

Taking a single unit of CPUX as a Link , DC-I-O-DC, in Intention Space, the sequence is repeated to extend a computational process to create a full-length CPUX suitable for the functionality as a use case of the App the Intention Space is used for. For example, a typical social chat App will have two CPUX that is active while two participants are engaged in a chat, or a sequence of actions performed in an App by e.g a click of a button can be treated as a CPUX ,'send a message' as in Fig 4.

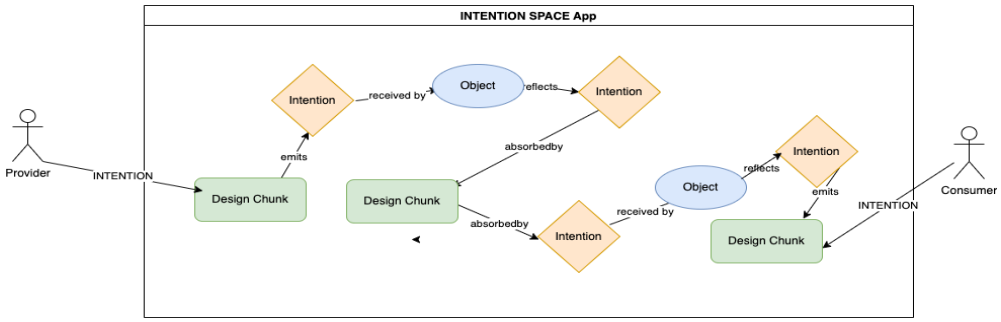

Fig 4. Any CPUX or Link repetition is uniquely identified in Intention Space

2.5. App in Domain, Intention Driven User Interaction

Any App in Intention Space can be considered a collection of CPUX, where a single CPUX is responsible for interaction with a human user. This is because a human user can only be represented by an Intention in the PnR model of computing; a human user needs two instances of intentions to interact with an App. This is significant from a privacy and security perspective as it needs registered Intentions to interact in each direction with the system , but we shall not get into that aspect of interaction in this paper.

In an App, multiple CPUX are not connected by a common Design chunk, which gives CPUX their individual identity. But how such individual CPUX can interact with each other if they don't share any design chunk? The answer to that question is the 'Space loops' that comes inbuilt with Intention Space as outlined in the next section.

2.6. Execution for any App in Intention Space , Intention Loop and Space Loop

The overall application flow logic and architecture of any App,i.e a collection of CPUX, can be summarised below:

Any execution process in Intention Space is driven by the names created by Design Chunks, Objects, Intentions, and PnRs; each CPUX represents an execution path, and the PnRs carry the values specific to that execution instance state and its progression from one design chunk to another. The Intentions static string participates in the emit and reflection paradigm.

i. a design chunk is a black box function that may or may not have its own IO, but that process stays outside Intention Space
ii. design chunk execution is governed by a pre-condition as a set of PnRs
iii. A Design chunk does not call/invoke any other Design chunk except receive/update PnRs from App run time. The App run time is a process run time which can be of two kinds , Intention Loop and Space Loop .
iv. A Design chunk produces a post condition, which is merged with the App run time PnR set.

A single design chunk does not explicitly trigger the next design chunk; the intention and its attached PnR sets transfer the PNR set from one design chunk post condition to the reach of the next.

So, the trigger mechanism that starts the new design chunk after the execution of one is finished is the 'Intention Loop' that is active for every CPUX. The Intention loop facilitates a sequence of

design chunks within a single CPUX to work sequentially. However, the application, by design, has multiple CPUX. So there is another independent loop called Space Loop to start (if pre-condition allowed) each CPUX. The Intention Loop goes through each design chunk is ready for execution is sequence , but the logic built in through PnR's this sequentially is not a requirement; the sequence helps with the cognitive meaning of the set of Design chunks. For example, Design chunks 'book a bus ticket from A to B ', followed by 'booking a plane ticket from B to C', which can be executed in any order unless some specific PnR precondition set makes the execution in sequence. Two CPUX can be quite independent, e.g. 'Find balance in the Account' and 'Book a Tour' are independent CPUX that the App may need to execute simultaneously . This can be done through Space Loop, which checks whether the first Design Chunk of every CPUX is readu to execute and trigger it if that is the case.

This is illustrated by the diagram below :

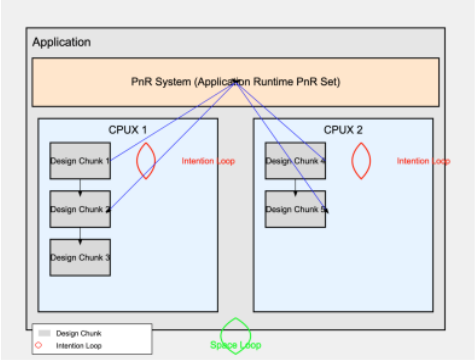

Fig 5. Architecture of App in Intention Space

2.7. Cross Application Communication

Both the Intention Loop and the Space loop check the readiness of the Design chunk in their path through the pre-condition check associated with the pre-condition associated with the Design Chunk. This is referred to as synchronicity check. While synchronicity check is done for Application run time the same design philosophy can be extended to the Domain runtime covering multiple Apps in the Domain as they all share the same PnR set. This is summarised in the Fig 6.

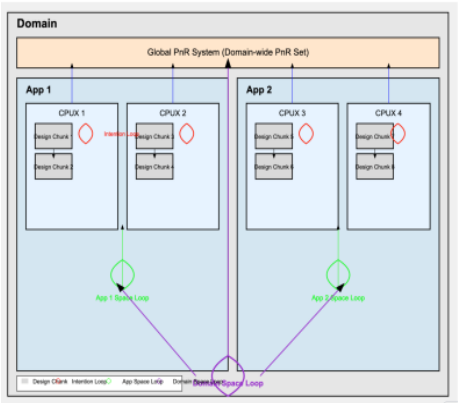

Fig 6 Cross Application Synchronicity

3. FIBONACCI GOLANG SEQUENCE EXAMPLE

In the following Fibonacci sequence GO example, we encapsulate the structure DC1-I1-O-I2-DC2 so that the Intention loop works through it. The task here is to generate the fibonacci numbers between two integers and then find the average of the numbers generated.

We have two loops in this scenario. The Intention loop is responsible for finding the Fibonacci number and it comprises of following Design chunks.

Design Chunk 1 (DC1): Start Sequence
Intention 1 (I1): Setup first two members in the Fibonacci sequence
Object (O): Fibonacci sequence
Intention 2 (I2): Find the next in the Fibonacci sequence
Design Chunk 2 (DC2): Find Next Fibonacci Number
Intention Loop: Manages to execute the design chunks with pre-condition (sync Test) checks within a CPUX.
Space loop Triggers the first design chunk in each CPUX.

The Go Lang code for the Fibonacci generation and running average can be found here Fibonacci generator wirh running average in go

4. PNR TRANSFORMATIONS IN PNR COMPUTING

In PnR computing, each Design Chunk operates as an independent software component, executed based on the state of the PnR (Prompt and Response) within its operational scope through preconditions. A Design Chunk is triggered by the Intention Loop, which initiates its execution. However, the Intention Loop remains agnostic to the specific computations or I/O processes that the Design Chunk may perform. The loop simply waits for a completion signal from the Design Chunk before proceeding.

Upon completing its task, the Design Chunk transmits the resultant PnR to an Object,through the emited Intention, which can then distribute this data to other Design Chunks or Objects as needed through reflected Intentions , while also doing permitted declarative mapping between PnR sets, declared without any procedural code. This model aligns well with the architecture of modern cloud computing, where independent software chunks are prevalent in the form of Docker containers, serverless functions (such as AWS Lambda), client-side components, and server-side services.

Objects that hold the state in PnR computing can be likened to cloud storage systems, with the added flexibility of persisting state as required by the participating Design Chunks. This persistensy is crucial in ensuring that the state is maintained across different execution contexts, providing consistency and continuity in processing.

Moreover, the Design Chunks can comprise any open-source or proprietary software that underpins the Internet today. These chunks can be readily extended to incorporate the functionality necessary for emitting or absorbing Intentions, integrating seamlessly into the PnR computing model. This adaptability makes PnR computing a powerful framework for building robust, scalable, and context-aware cloud-based applications.

4.1. The Scopes of Computation in Intention Space

Individual Design chunks, CPUX (Intention Loops) and Space loops are different execution scopes in Intention Space, meaning the same PnR (i.e. identified by its name) can exist in these different scopes at any point during execution time. An App can have multiple Space loops. The Objects in Intention Space that reflects Intentions carrying PnR sets also gives a mapping facility of set-centric PnR mapping. Thses set-wise scope transfers are limited to scope transfers. This means that while PnR sets are passed between Design Chunks, their values and conditions remain unchanged during the transfer. These mappings' role is to transfer data across scopes without performing any computation, ensuring that all logical transformations are confined to the Design Chunks.

4.2. The PnR Interchange Protocol in Intention Space

With the generic process described above, it is possible to streamline the inter-Design chunk communications through some standardised function interfaces as listed below; these functional utility units work as the driver of PnR computing across design chunks.to simplify the inter-Desig nchunk communications through standardised function interfaces that define the flow of PnRs through the computational process.

As any Design chunk or CPUX works inside an implicit loop in a run-time context, each design chunk can define how and when it is executed through the pre-condition or the 'Synctest' function. As a Design chunk passes the Synctest, it is selected for execution; it allows some PnR from the runtime context, which will be an Intention loop, to be brought into its own set of PnR through the function 'Flowin' and after the execution of the Design chunk, the function 'Flowout' puts the resultant PnR into the runtime context.

i. Synctest. This function works with two sets of PnRs: one is Visitor, and the other is Gateman;
ii Flowin
iii Flowout
iv Mapping at Object . When an Object reflects an Intention, the PnR carried by the incoming Intention is mapped onto those carried by the reflected Intention through a sequence of mapping through the runtime PnR sets. These maps can take several forms, as listed below, i1 and i2 being the two subtests of PnR.

a. Union

Description: Combines the PnR sets from i1 and i2 into a single set, without removing duplicates. The resulting set will include all PnRs present in either i1 or i2.

Use Case: When you want to pass forward all available data without filtering or prioritizing any specific PnR.

Example:

i1: {"name1": ["value1", "Y"], "name2": ["value2", "N"]}

i2: {"name3": ["value3", "Y"], "name2": ["value4", "U"]}

Result: {"name1": ["value1", "Y"], "name2": ["value2", "N"], "name3": ["value3", "Y"], "name2": ["value4", "U"]}

b. Intersection

Description: Creates a PnR set containing only the PnRs that are present in both i1 and i2. The result set will include only those PnRs with matching names in both sets.

Use Case: When you need to ensure that only common data elements are passed forward, it is often used for filtering based on shared attributes.

Example:

i1: {"name1": ["value1", "Y"], "name2": ["value2", "N"]}
i2: {"name2": ["value4", "U"], "name3": ["value3", "Y"]}
Result: {"name2": ["value2", "N"]}

c. Difference (Subtract)

Description: Removes the PnRs from i1 that have matching names in i2. The result is a set of PnRs from i1 that do not have corresponding names in i2.

Use Case: This is useful when you want to exclude certain PnRs from the set, often to remove redundant or conflicting data.

Example:

i1: {"name1": ["value1", "Y"], "name2": ["value2", "N"], "name3": ["value5", "Y"]}
i2: {"name2": ["value4", "U"], "name3": ["value3", "Y"]}
Result: {"name1": ["value1", "Y"]}

d. Union with Overwrite

Description: Combines the PnR sets from i1 and i2, but if there are matching PnR names, the value from i2 overwrites the value from i1.

Use Case: When updating existing PnRs with new values while retaining non-conflicting PnRs from both sets.

Example:

i1: {"name1": ["value1", "Y"], "name2": ["value2", "N"]}
i2: {"name2": ["value4", "U"], "name3": ["value3", "Y"]}
Result: {"name1": ["value1", "Y"], "name2": ["value4", "U"], "name3": ["value3", "Y"]}

e. Union with Preserve

Description: Similar to Union, but if there are matching PnR names, both PnRs are preserved, creating a list of possible values for that name.

Use Case: When you need to track multiple possible values for a PnR name, without losing any data.

Example:

i1: {"name1": ["value1", "Y"], "name2": ["value2", "N"]}
i2: {"name2": ["value4", "U"], "name3": ["value3", "Y"]}
Result: {"name1": ["value1", "Y"], "name2": [["value2", "N"], ["value4", "U"]], "name3": ["value3", "Y"]}

f. Match Only

Description: Select only the PnRs from i1 that have matching names in i2, but it carries over the values from i1.

Use Case: When you want to ensure that only specific PnRs (those present in i2) are passed forward, you want to retain their original values.

Example:

i1: {"name1": ["value1", "Y"], "name2": ["value2", "N"], "name3": ["value5", "Y"]}
i2: {"name2": ["value4", "U"], "name3": ["value3", "Y"]}
Result: {"name2": ["value2", "N"], "name3": ["value5", "Y"]}

g.Complement

Description: Produces a set of PnRs from i2 that do not appear in i1. It's the inverse of the Intersection, providing elements in i2 that are unique compared to i1.

Use Case: Useful for identifying and isolating new or additional PnRs introduced.

Example:

i1: {"name1": ["value1", "Y"], "name2": ["value2", "N"]}

i2: {"name2": ["value4", "U"], "name3": ["value3", "Y"]}

Result: {"name3": ["value3", "Y"]}

4.3. Cross-App Interactions

The concept of a Domain across which multiple Apps exist enables cross-app interactions and data sharing. Apps within the same domain can exchange PnRs and collaborate to achieve common goals or perform coordinated tasks. We can represent cross-app interactions as functions that operate on the PnRs of multiple apps: Cross App Interaction(App_i, App_j) = f(PnR_i, PnR_j) Here, f is a function that defines the interaction between App_i and App_j based on their respective PnR subsets, PnR_i and PnR_j. The function f can involve exchanging, merging, or transforming PnRs to facilitate communication and collaboration between the apps. Cross-app interactions can enable various scenarios, such as:

1. Data Sharing: Apps can share PnRs to exchange information, synchronize state, or coordinate actions.
2. Collaborative Processing: Apps can work together to perform complex tasks by leveraging each other's CPUXs and PnRs.
3. Workflow Coordination: Apps can coordinate their execution flows by triggering CPUXs in other apps based on specific PnR conditions.

The domain is a unifying entity that facilitates these cross-app interactions by providing a common set of PnRs that multiple apps can access and manipulate.

4.4. Real Life App Scenarios

Today, especially in developing countries, demand for functional multi-functional mobile Apps is on the rise as young start-ups look into implementing agile solutions in business processes, such as mobile apps used for an end-to-end customer experience for a customer walking into a shop starting from product selection, packaging, payment, warranty, insurance, delivery, maintenance primarily driven by a QRcode triggered through the App. The custom experience here is a physical presence in the shop accompanied by a business process step automation through the App.

The expectations from the startups from software developers are high. We need a more holistic approach to app development, deployment and scaling to meet the day's challenges. The domain-wide domain-wide PnR set, set, where each PnR is uniquely identifiable, is the start of this development scenario, which scenario, which may look like this :

```
[
  { "Customer entered shop?":[[ ],'UN'] },
  { "Item selected": "SmartphoneX" },
```

```
  { "Payment method": "Credit Card" },
  { "Payment processed", "answer": "Success" },
  { "Inventory updated", "answer": "Yes" },
  { : "Warranty registered", "answer": "Yes" },
  { "Customer satisfaction survey sent", "answer": "Pending" }
]
```

5. FORMAL DESCRIPTION OF PNR COMPUTING

Summarising the PnR computing model through Sets and Their Relationships.

A Domain represents the well-defined set of prompt-response pairs (PnRs) that can be used across multiple applications. Here, the domain is not an abstract concept as it implies in a traditional computing scenario; it is a unifying entity, a set, encompassing all possible PnRs within its scope along with the sets of DesignChunks, Intentions, and Objects.

A single PnR can be represented as a JSON {"q." :[aa, "Y/N/U"]}, where q is a string, aa is a primitive or any JSON or an Array, and the last item in the value is a trivalence that can take the value 'Y','N', or 'U' [for yes, no and undecided respectively]

Link: A Basic Unit Of Computation

A Link is a pair of design chunks joined by the unique occurrence of the intention-object-intention trio. It is the smallest unit of CPUX (Common Path of Understanding and Execution). A CPUX can be a sequence of Links of any particular length where the ending design chunk of a Link coincides with the starting domain chunk of the next Link.

The Sets Driving Computing Through Flow Compositions

Let DC be the set of all Designchunks, where each d∈DC is a unique string representing a design chunk.

Let I be the set of all Intentions, where each i∈I is a unique string representing an intention.
Let O be the set of all Objects, where each o∈O is a unique string representing an object.

A Flow F can be defined as a 5-tuple F=(d1,i1,o,i2,d2) where:
d1,d2∈D are design chunks.
i1,i2∈Ii1 are intentions.
o∈O is an object.

The tuple (i1,o,i2)(i1,o,i2) represents the unique intention-object-intention trio that joins the design chunks d1 and d2

A CPUX is a sequence C=(F1,F2,…,Fn)C=(F1,F2,…,Fn) of Flows Fk=(dk1,ik1,ok,ik2,dk2)Fk=(dk1,ik1,ok,ik2,dk2), for k=1,2,…,nk=1,2,…,n, where:
dk2=d(k+1)1dk2=d(k+1)1 for all kk such that 1≤k<n1≤k<n.

This means that the ending design chunk dk2dk2 of each Flow FkFk coincides with the starting design chunk d(k+1)1d(k+1)1 of the next Flow Fk+1.

Let C be the set of all CPUX (Common Path of Understanding and Execution) sequences, where each C=(F1,F2,…,Fn) is a sequence of Flows as defined above.
Let PNR be the set of all possible PnRs: Domain = {PnR_1, PnR_2, …, PnR_m} Here, each PnR_i represents a unique prompt-response pair within the domain. Each PnR, a JSON object, is uniquely identifiable within a Domain by its name portion.
Computation CDomain : A Computation Domain D is defined as the set that contains all the elements of the sets DC, I, O, F, C and PNR.

Apps within a Domain

Each app within a domain has a subset of PnRs drawn from the domain's PNR set. An app can be defined as a tuple consisting of its CPUXs and its specific subset of PnRs: App_i = ({$CPUX_1$_i, $CPUX_2$_i, …, CPUX_n_i}, {PnR_1_i, PnR_2_i, …, PnR_k_i}) Here, {$CPUX_1$_i, $CPUX_2$_i, …, CPUX_n_i} represents the set of CPUXs in App_i, and {PnR_1_i, PnR_2_i, …, PnR_k_i} represents the subset of PnRs specific to App_i, where each PnR_j_i ∈ Domain.

Intention Loop (IL)

An Intention Loop for a CPUX c is defined as:
IL(c) = (DC, <, P, R, T) Where

- DC is the set of Design Chunks in the CPUX
- < is a partial order on DC, representing the execution order
- P is the set of Preconditions (a subset of PnR)
- R is the set of Postconditions (a subset of PnR)
- T is a transition function: $T: P \times DC \rightarrow R$

The Intention Loop ensures that:

1. $\forall dc \in DC$, $Pre(dc) \subseteq P$ and $Post(dc) \subseteq R$
2. The execution of DC according to < transforms P into R
3. $T(P, DC) = R$ holds after the execution of all $dc \in DC$

Space Loop (SL)

A Space Loop for an App a is defined as:
$SL(a) = (C, \Gamma, \Pi, \Omega, \Phi)$

Where:

- C is the set of CPUXs in the App
- Γ is a directed graph representing CPUX dependencies
- Π is the set of App-level Preconditions
- Ω is the set of App-level Postconditions
- Φ is an orchestration function: $\Phi: \Pi \times C \rightarrow \Omega$

The Space Loop ensures that:

1. $\forall c \in C$, $P(c) \subseteq \Pi$ and $R(c) \subseteq \Omega$, where P(c) and R(c) are from IL(c)
2. The execution of C according to Γ transforms Π into Ω
3. $\Phi(\Pi, C) = \Omega$ holds after the execution of all $c \in C$

Intention Space (IS)

The Intention Space for a Domain d is defined as:
$IS(d) = (A, \Delta, PnR, \Psi)$

Where:

- A is the set of Apps in the Domain
- Δ is a hypergraph representing App interactions
- PnR is the global set of Prompts and Responses
- Ψ is a global state transition function: Ψ: PnR × A → PnR

The Intention Space ensures that:

1. $\forall a \in A$, $\Pi(a) \cup \Omega(a) \subseteq PnR$, where $\Pi(a)$ and $\Omega(a)$ are from SL(a)
2. The execution of A according to Δ maintains global consistency of PnR
3. $\Psi(PnR, A) = PnR'$ holds, where PnR' is the updated global state after execution

These formal definitions establish the inter-relationship between Intention Loops (at the CPUX level), Space Loops (at the App level), and the overall Intention Space (at the Domain level) while maintaining the integrity and consistency of PnR sets across all levels.

6. THE SYNCHRONICITY SETS

The concept of PnR detached from the function codes that use and change them creates a new field of establishing logical truth or untruth based on pure business knowledge which ultimately drives the design chunk or function execution and data flow in the application. Each Designchunk will have its own PnR set , CPUX will have its own subset of PnR; a collection of CPUX, an App, will form its own set of PnR. These multiple subsets of PnR and their concordance are referred to as Synchronicity in Intention Space. This is illustrated in Fig 7.

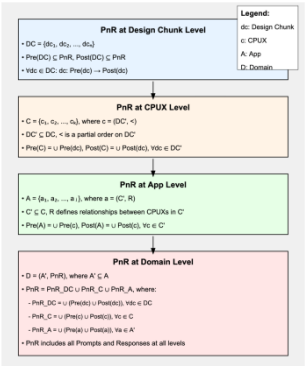

Fig 7 PnR subsets as Synchronicity Sets in Business

Synchronicity Test .

A Synchronicity function works with two sets of PnRs, one is a Visitor, and the other is a Gateman. So the idea of synchronicity is Gateman only checks for each of the PnR in its own set and finds the corresponding PnR in Visitor; if it can't find it , it is false; if it finds and the trivalence portion('Y','N','U') boolean do not match, result is false; if it finds and the trivalence

portion match, then the result is true. Thiis is illustrated in the JS code ,available here Syntest with gatekeeper and visitor

7. INTERNET MATURITY

Today, cloud-native development has become a cornerstone of modern computing, made possible by over 50 years of relentless progress in computing infrastructure. However, during the last 15 years, something significant has occurred in terms of CPU power increasing:

Increased Core Count:

Over the last 15 years, CPUs have seen a significant increase in cores, moving from predominantly single-core or dual-core processors in consumer devices to multi-core processors with 8, 16, or more cores.

Parallel Processing:

The rise of multi-core processors has allowed for parallel processing, where multiple tasks or threads can be executed simultaneously. This has led to substantial improvements in performance, particularly in applications that can leverage parallelism.

Despite these transformative changes, managing real-time interactions between disparate, manageable application software components at runtime remains a critical challenge. [2]

In today's processor environment, breaking up workloads into independent units orchestrated by an ever-running loop with central state management is a strategic approach that offers scalability, resilience, and efficiency. Drawing inspiration from Kubernetes, which has successfully orchestrated distributed systems, Intention Space opens up a new architectural style in distributed computing.

8. INTENTION SPACE ARCHITECTURAL PATTERNS

When Roy Fielding introduced the REST architectural style in his seminal work [3], he envisioned a scalable and resilient Internet architecture that could withstand unanticipated loads and malicious data. He advocated for "Anarchic scalability," where architectural elements continue functioning reliably, even when interacting with uncontrolled external components.

Many of today's applications are still developed with a single-core CPU architecture mindset, mainly due to a lack of tooling. This presents a significant challenge in managing real-time interactions between diverse and independently manageable software components at runtime. Software practices take time to change. Today, some foundational issues Fielding identifies remain unresolved, even as cloud technology evolves. For instance, Fielding critiqued the introduction of HTTP cookies as an extension to the web protocol that conflicted with the stateless nature of REST, causing complications in managing application state across the web. He noted that "Cookie interaction fails to match REST's application state model, often confusing the typical browser application."

Roy Fielding also says, "Architectural design and source code structural design, though closely related, are separate design activities." The role of Designchunks in Intention Space fully supports this view.

8.1. Design Chunks in Intention Space

When Fielding wrote his paper, the infrastructure required to achieve the dynamic, real-time coordination of software components he envisioned was not entirely in place. However, the maturation of cloud technology has changed this landscape. Today's cloud-native environments provide the scalability, flexibility, and computational power to implement advanced models like the Design Chunk and Intention Loop. Loop. Intention Loop and Intention Ring point to developing business application software fitting to our processor environments.

The adaptation of Design chunks in a broader development phase can be enormous. It brings a much-longed-for alignment between screen designs and the whole App design philosophy. This topic will be covered in another paper.

8.2. Space Loop & Micro Service

Cloud technology enables distributed systems to interact seamlessly, handle vast amounts of data, and execute complex logic across geographically dispersed locations in real-time. This environment makes it feasible to move beyond the traditional request-response paradigm and adopt an Intention Loop as the primary driver of interactions. With its continuous evaluation of software components (Design Chunks), the Intention Loop aligns with the needs of modern, cloud-based applications, where the state is shared and synchronized across participants in a network rather than confined to individual client-server interactions.

Moreover, the Space Loop coordinates multiple Intention Loops across different CPUXs (Common Path of Execution and Understanding) between humans and machines, leveraging the cloud's ability to manage distributed processes efficiently. This orchestration ensures that the system remains responsive, scalable, resilient, manageable, and understandable even as the complexity of interactions increases.

This two-dimensional self-triggering mechanism, facilitated by the Intention Loop and Space Loop, can alleviate many of today's microservice composition challenges [4], particularly in scenarios involving client-server GUI interactions where the microservices are bound by the client and server request-response request-response irrespective of the whole App state. An App can have multiple Spaceloops. This provides a more integrated and fluid interaction model that paves the way for more cohesive and scalable cloud-native applications, addressing issues like state synchronization, process orchestration, and system resilience.

8.3. Design Chunks as Microservice Units

In the Intention Space model, Design Chunks represent self-contained units of functionality that execute based on predefined conditions. These chunks emit Intentions, which then guide subsequent actions within the system. The Intention Loop, enabled by the robust and scalable cloud infrastructure, continuously cycles through these chunks, ensuring that the system adapts dynamically to changes in context and state. At the same time, the Intention Loop waits for the current Designchunk's signal before it progresses in the loop. This allows the Designchunks to be responsible for any I/O operation in the application's primary process.

By leveraging today's cloud capabilities, the Intention Space model addresses the foundational gaps identified by Fielding and realizes a vision of the Internet where dynamic, context-aware interactions drive the system rather than static, predefined request-response cycles. This shift is crucial for building scalable, flexible, and contextually congruent systems in the cloud era. At the

same time, it gives us a pathway to delve into a Designchunk-based prototype for PNR flow in any environment or language, including full-stack development, which we shall explore in the future.

8.4. End-to-End Microservices, Inclusive of Front-End

A foundational characteristic of Design Chunk, as described in [1], is its ability to hold static media, functional code, or any other custom content. At the same time, Designchunk, a reference to resources that can be distributed across physical boundaries, allows simple virtualization of resources to usher in a new way of looking at user's device characteristics. The concept of microservices is embedded in the foundation of Intention Space, as the intention loop drives it, and the PnR structure allows any two Designchunks to share data and predicate logic, as long as they share a loop, either an Intention Loop or a Space Loop, that has shared a member Object. This approach allows the benefit of micro-front ends without its accompanying drawback [5] and, at the same time, allows much more managed sharing of resources in traditional microservice [6] ,[7]and still encourages the domain driven domain-driven design principle of small, autonomous service [8], where the focus is just not on scaling at deployment time but also bringing micro units at composition time. Using unique PnRs in the domain, expressed in free natural language, allows a uniform approach across multiple areas of concern.

9. DOCKER FOR CITIZEN SCIENCE

While multiple Design Chunks can operate within a container like Docker—which has become the de facto standard for containerization, whether running on a local machine or in the cloud—coordinating these independently running containers concerning timing, data exchange, and licenced data protection still requires custom solutions.

The Intention Loop can address this challenge by extending the coordination of independent Docker containers that are open to sharing data through object stores in the cloud a CPUX being a sequence of container in the same or different machines. At the same time, the space loop is not needed unless multiple CPUX in place. In this context, the Intention Loop would function as a ring executed on each local machine, interfacing with Intetion space server to manage the execution and interaction of local Docker containers. This approach enables collaboration among multiple independent units, such as citizen scientists working on different aspects of a simulation, allowing them to share and integrate their models effectively without needing to share custom data.

For instance, one scientist could focus on population statistics projection, while another develops a transport simulation. Together, they contribute to a comprehensive transport planning model for a region. A town planner could then vary the input PnRs from either model to explore different scenarios. In this scenario, the Design Chunks are individual Docker containers tied to local resources—not purely independent services—but capable of sharing model data effectively while adhering to time parameters.

This approach brings enhanced observability and coordination between independent containers running across the globe, leveraging the PnR model to create a robust and scalable distributed computing environment. By doing so, the Intention Loop facilitates seamless collaboration, enabling researchers to integrate their work into a cohesive and dynamic system.

10.AI -LLM AND THE HUMAN INTENT

One core tenet of PnR computing is the uniqueness of the DC-I-O-I-DC (Design chunk-Intention-Object-Intention-Design chunk) occurrence within the entire domain. This approach effectively resolves many issues related to software code reusability and execution time ambiguity or traceability without delving into the style or content of the code itself. Composing the synchronizing sets that can be composed of business rules contributes to the cognitive integrity of the web App and allows the construction of meaningful test cases, too, by providing a common path of runtime execution path and test units [9]

While composing PnR sets for complex systems can be a meticulous task, using Large Language Models (LLMs) can significantly aid this process. With their capability to understand and generate natural language, LLMs can help compose effective PnR sets that cater to complex logic dependencies, making the system more adaptable to various contexts and domains.

10.1. The Role of Human Intent in Computing

In the broader design of Intention Space, Design Chunks emit and absorb Intents while users interact with the machine through Intents. This design allows Intention Space to represent the human user as a Design Chunk, recognizing that each user has unique behaviours and intentions. However, Intention Space remains agnostic to the specific behaviour of any individual user within a Design Chunk, focusing instead on how the particular intention interacts with the system's logic and processes. At the same time, Intentions and Object names, as natural language phrases, become the protocol of interaction while maintaining the DC-I-O-DC uniqueness.

The model allows us to view software as a network of Design chunks working together to enable interactions like writing an email, sending a message, receiving a message, or completing a bank transaction. What sets this apart is that recognized Intentions and Objects power these interactions. While this explicit use of natural language as a communication protocol between Design chunks does not automatically make the system accountable, it does interlace the power of LLMs between Design chunks. This makes the system much more traceable, enabling the construction of a dynamic, customized profile for each interaction. As a result, it provides better fidelity for identifying and managing offences, contributing to a safer and more responsible online environment. The approach opens up the possibility of a safer internet in society.

11. CONCLUSION ROAD AHEAD

The Intention Loop and PnR system open up a new frontier in distributed interactive computing—an evolution made possible by significant advancements in network capabilities, CPU power, and memory availability, the three core pillars of Cloud Computing.

PnR computing's most significant contribution lies in aligning user understanding of business processes with computing capabilities. Treating both as design chunks, enables Intention Space to weave design and media content with computing's analytical power.
This paradigm shift simplifies the development of applications on any platform and fosters a deeper, more intuitive interaction between humans and machines, leading to a much healthier, secured internet.

Here, as LLMs working with natural language phrases backed up by a database unique key and other integrity rules that an Intention Space offers can bring a semantic layer that helps align business goals and technology platforms at different flavours of technology (Android ,IOS,Web

)or within the different levels of technology (Network layer ,Application Layer,Domain Specific software), e.g., Intent-Driven Network Management [10] or [11]. In our future research and development, we shall incorporate these dimensions into Intention Space.

REFERENCES

[1] Pronab Pal. Human Intention Space - Natural Language Phrase Driven Approach to Place Social Computing Interaction in a Designed Space https://doi.org/10.5121/ijnlc.2024.13302
[2] Diane Coyle, Lucy Hampton, 21st-century Hampton, 21st-century progress in computing, Telecommunications computing, Telecommunications Policy, Volume 48, Issue 1,2024,102649, 1,2024,102649, https://doi.org/10.1016/j.telpol.2023.102649
[3] Roy Fielding https://ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf
[4] Martin Fowler https://martinfowler.com/articles/microservices.html
[5] Antunes et al.)(ar5iv)(exploring-microservice-based-frontend-architecture/).
[6] Team, F.BU. (2024). Microservices in the Cloud Native Era. In: Cloud-Native Application Architecture. Springer, Singapore. https://doi.org/10.1007/978-981-19-9782-2_1
[7] Nicolas-Plata, A., Gonzalez-Compean, J.L. & Sosa-Sosa, V.J. A service mesh approach to integrate processing patterns into microservices applications. *Cluster Comput***27**, 7417–7438 (2024). https://doi.org/10.1007/s10586-024-04342-5
[8] Benjamin Hippchen, Pascal Giessler, Roland Heinz Steinegger,Michael Schneider and Sebastian Abeck Designing Microservice-Based Applications by Using a Domain-Driven Design Approach Approach https://cm.tm.kit.edu/download/domain_driven_microservice-architecture.pdf
[9] Camilli, M., Bellettini, C., Capra, L. (2018). Design-Time to Run-Time Verification of Microservices Based Applications. In: Cerone, A., Roveri, M. (eds) Software Engineering and Formal Methods. SEFM 2017. Lecture Notes in Computer Science(), vol 10729. Springer, Cham. https://doi.org/10.1007/978-3-319-74781-1_12
[10] T. Hey, "INDIRECT: Intent-Driven Requirements-to-Code Traceability," 2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion), Montreal, QC, Canada, 2019, pp. 190-191, doi: 10.1109/ICSE-Companion.2019.00078.
[11] Silvander, J., Wnuk, K., & Svahnberg, M. (2020). Systematic literature review on intent-driven systems. IET Software, 14(4), 345-357. https://doi.org/10.1049/iet-sen.2018.5338

AUTHOR

Pronab is a long-term software architect and developer working as a freelancer at Keybyte Systems. Pronab leads a small group of developers at Keybyte Systems, developing modern multiplatform cloud native Apps. When he is not thinking about code, he spends time knowing and understanding the varied ways people live their lives worldwide.