Improved Content Addressability Through Relational Data Modeling and In-Network Processing Elements

Claudio Marxer
<claudio.marxer@unibas.ch>

Christian Tschudin
<christian.tschudin@unibas.ch>

Computer Networks Group · University of Basel · Switzerland

ACM ICN ’17, Berlin · September 27, 2017
Shortcomings of Hierarchical Content Organization

- Sometimes, a document can be part of multiple collections

  /personal/joe/2017/NYmarathon/time-location.gpx
  /personal/joe/marathons/NYmarathon/2017/time-location.gpx
  /organizer/NYmarathon/2017/ranking/1/time-location.gpx
  /organizer/NYmarathon/2017/joe/time-location.gpx

  → Different access patterns for a document require different ICN names.
Shortcomings of Hierarchical Content Organization

– Sometimes, a document can be part of multiple collections
  /personal/joe/2017/NYmarathon/time-location.gpx
  /personal/joe/marathons/NYmarathon/2017/time-location.gpx
  /organizer/NYmarathon/2017/ranking/1/time-location.gpx
  /organizer/NYmarathon/2017/joe/time-location.gpx

  → Different access patterns for a document require different ICN names.

– Unsupported access-pattern: Only tracks longer than 20 miles are relevant.
  → Data Packet has implicit characteristics which can’t be part of the ICN name
Shortcomings of Hierarchical Content Organization

- Sometimes, a document can be part of multiple collections
  
  `/personal/joe/2017/NYmarathon/time-location.gpx`
  `/personal/joe/marathons/NYmarathon/2017/time-location.gpx`
  `/organizer/NYmarathon/2017/ranking/1/time-location.gpx`
  `/organizer/NYmarathon/2017/joe/time-location.gpx`

  → Different access patterns for a document require different ICN names.

- Unsupported access-pattern: Only tracks longer than 20 miles are relevant.
  
  → Data Packet has implicit characteristics which can’t be part of the ICN name

- Limited descriptiveness of an ICN name affects granularity of name-based security policies

  - So, why not extensively descriptive names? Because: Overlong Names! Name Privacy!
  - Anyway, not all implicit information can be added to name (e.g. track length)
Wish List: An Access and Naming Scheme Should...

As the publisher, I can't know how all these future apps which want to access my information. I publish it in a general form and also provide some meta-data. This way, a third party can satisfy all the app-specific needs.

Hi, I'm a new app! I want to access your data grouped/filtered/.. by ...

Challenge accepted! As a cloud provider, I can satisfy on-demand needs if I have good recipes and sufficient raw data in place.

Security Mechanics: I must be able to reason about an on-demand request name. i.e. understand what information is named.
Wish List: An Access and Naming Scheme Should...

As the **publisher**, I can't know how all these future apps which want to access my information. I publish it in a general form and also provide some meta-data. This way, a third party can satisfy all the app-specific needs.

Hi, I'm a new app! I want to access your data grouped/filtered/... by ...

Challenge accepted! As a cloud provider, I can satisfy on-demand needs if I have good repicies and sufficient raw data in place.

Security Mechanics: I must be able to reason about an on-demand request name. I.e. understand what information is named.
Wish List: An Access and Naming Scheme Should...

As the publisher, I can't know how all these future apps which want to access my information. I publish it in a general form and also provide some meta-data. This way, a third party can satisfy all the app-specific needs.

Hi, I'm a new app! I want to access your data grouped/filtered/... by ...

Challenge accepted! As a cloud provider, I can satisfy on-demand needs if I have good repicies and sufficient raw data in place.

Security Mechanics: I must be able to reason about an on-demand request name. I.e. understand what information is named.
Wish List: An Access and Naming Scheme Should...

As the publisher, I can't know how all these future apps which want to access my information. I publish it in a general form and also provide some meta-data. This way, a third party can satisfy all the app-specific needs.

Hi, I'm a new app! I want to access your data grouped/filtered/.. by ...

Challenge accepted! As a cloud provider, I can satisfy on-demand needs if I have good recipes and sufficient raw data in place.

Security Mechanics: I must be able to reason about an on-demand request name. I.e. understand what information is named.
Wish List: An Access and Naming Scheme Should...

As the **publisher**, I can't know how all these future apps which want to use my information. I publish it in a compartmentalized way and also provide some meta-data. This way, a third party can satisfy all the app-specific needs.

**On-demand composition of declarative names**

Hello, I'm a new app! I can satisfy on-demand data.

**Static Content Only**

***Declarative Name***'-to-***Production Recipe***'

Security Mechanics: I must be able to reason about an app's need for request name. I need to find what information is named.
Outline

– Problem Statement: Shortcomings of Hierarchical Content Organization

– Wish List for a Access/Naming Scheme

– Background: In-Network Processing & Named-Function Networking
  – ICN & Relational Data Modeling
    - Publisher Side: Data Organization
    - Consumer Side: Data Querying

– Comprehensive Example

– Conclusion
Named-Function Networking: Consumer’s View (Example 1)

**Classic ICN:** Distribution of named content (published)

lookup: /joe/NYmarathon/track.gpx

**Named Function Networking (NFN):** Generation of named content (on-demand)

lookup: /get/duration( /joe/NYmarathon/track.gpx )
Named-Function Networking: Consumer’s View (Example 1)

**Classic ICN:** Distribution of named content (published)

lookup: /joe/NYmarathon/track.gpx

**Named Function Networking (NFN):** Generation of named content (on-demand)

lookup: /get/duration(/joe/NYmarathon/track.gpx)
Named-Function Networking: Consumer’s View (Example 1)

**Classic ICN:** Distribution of named content (published)

lookup: /joe/NYmarathon/track.gpx

INTEREST[/joe/NYmarathon/track.gpx]

**Named Function Networking (NFN):** Generation of named content (on-demand)

lookup: /get/duration( /joe/NYmarathon/track.gpx )

INTEREST[/get/duration/@x call 2 x |joe|NYmarathon|track.gpx/NFN]

INTEREST[/joe/NYmarathon/track.gpx/@x call 2 |get|duration x/NFN]
Special NFN-capable nodes dissect the interest’s NFN name and orchestrate the result derivation.
Special NFN-capable nodes dissect the interest’s NFN name and orchestrate the result derivation.
NFN Expression = Network Name of the Result = Production Recipe

⇒ “On-demand Composition of Declarative Names” (Apps)
   High-level library of NFs allows to express domain-specific content needs.

⇒ “Declarative Name”-to-“Production Recipe” (Cloud Provider)
   NFN Expression with NF bytecode is the “program” to produce the content.

⇒ “Static Content Only!” (Publisher)
   NFN cares about dynamic/on-demand/derived content.

▷ “Descriptive Names” (Security Mechanics)
   Well, no per se clear what a NF does with the content.
   But tomorrow: Schematized Access Control for Data Cubes and Trees
Back to the Wish List...

**NFN Expression** = **Network Name of the Result** = **Production Recipe**

⇒ "On-demand Composition of Declarative Names" (Apps)
   High-level library of NFs allows to express domain-specific content needs.

⇒⇒ "Declarative Name"-to-"Production Recipe" (Cloud Provider)
   NFN Expression with NF bytecode is the “program” to produce the content.

⇒ "Static Content Only!" (Publisher)
   NFN cares about dynamic/on-demand/derived content.

⊄ "Descriptive Names" (Security Mechanics)
   Well, no per se clear what a NF does with the content.
   But tomorrow: Schematized Access Control for Data Cubes and Trees
Back to the Wish List...

NFN Expression $=$ Network Name of the Result $=$ Production Recipe

$\Rightarrow$ “On-demand Composition of Declarative Names” (Apps)
High-level library of NFs allows to express domain-specific content needs.

$\Rightarrow$ “Declarative Name”-to-“Production Recipe” (Cloud Provider)
NFN Expression with NF bytecode is the “program” to produce the content.

$\Rightarrow$ “Static Content Only!” (Publisher)
NFN cares about dynamic/on-demand/derived content.

$\Rightarrow$ “Descriptive Names” (Security Mechanics)
Well, no per se clear what a NF does with the content.
But tomorrow: Schematized Access Control for Data Cubes and Trees
Back to the Wish List...

**NFN Expression** = **Network Name of the Result** = **Production Recipe**

⇒ “On-demand Composition of Declarative Names” (Apps)
   High-level library of NFs allows to express domain-specific content needs.

⇒ “Declarative Name”-to-“Production Recipe” (Cloud Provider)
   NFN Expression with NF bytecode is the “program” to produce the content.

⇒ “Static Content Only!” (Publisher)
   NFN cares about dynamic/on-demand/derived content.

⇒⇒ “Descriptive Names” (Security Mechanics)
   Well, no per se clear what a NF does with the content.
   But tomorrow: Schematized Access Control for Data Cubes and Trees
Example: Relational Data Organization (1)

**Named Relations:** Structured Information

<table>
<thead>
<tr>
<th>PersID*</th>
<th>Name</th>
<th>Home</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Alice</td>
<td>US</td>
</tr>
<tr>
<td>2</td>
<td>Bob</td>
<td>DE</td>
</tr>
<tr>
<td>3</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

/repo/people.table

<table>
<thead>
<tr>
<th>EventID*</th>
<th>PersID^</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>NYmarathon</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>ParisMarathon</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>NYmarathon</td>
</tr>
</tbody>
</table>

/repo/events.table
Example: Relational Data Organization (1)

**Named Relations:** Structured Information

<table>
<thead>
<tr>
<th>PersID*</th>
<th>Name</th>
<th>Home</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Alice</td>
<td>US</td>
</tr>
<tr>
<td>2</td>
<td>Bob</td>
<td>DE</td>
</tr>
<tr>
<td>3</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

/repo/people.table

<table>
<thead>
<tr>
<th>EventID*</th>
<th>PersID^</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>NYmarathon</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>ParisMarathon</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>NYmarathon</td>
</tr>
</tbody>
</table>

/repo/events.table

**Relation Schemata:** Meta-Data

```
PersID*: Int
Name: String
Home: String
```

```
EventID*: Int
PersID^: Int -> /ietf/relations/people.schema
Name: String
```

/ietf/relations/people.schema

/ietf/relations/events.schema
Example: Relational Data Organization (2)

How to assign schemata to relations? Relation Type Schema (RTS):

<table>
<thead>
<tr>
<th># published as /repo/rts</th>
</tr>
</thead>
<tbody>
<tr>
<td>/repo/people.table</td>
</tr>
<tr>
<td>/repo/events.table</td>
</tr>
<tr>
<td>/repo/circuits/[~/]+</td>
</tr>
</tbody>
</table>

Typing: Mapping named relations (name patterns) to schemata documents.
Example: Basic Query Operations

DB Research: Few basic set operations are sufficient to define a rich relational algebra.

→ **restrict:** /named/fct/restrict( /repo/people.table, Home == 'US' )
→ **project:** /named/fct/project( /repo/people.table, [PersID,Name] )
→ **join:** /named/fct/join( /repo/events.table as 'event',
                               /repo/people.table as 'people' )

<table>
<thead>
<tr>
<th>PersID*</th>
<th>Name</th>
<th>Home</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Alice</td>
<td>US</td>
</tr>
<tr>
<td>2</td>
<td>Bob</td>
<td>DE</td>
</tr>
</tbody>
</table>
Example: Basic Query Operations

DB Research: Few basic set operations are sufficient to define a rich relational algebra.

- **restrict**: `/named/fct/restrict( /repo/people.table, Home == 'US' )`

  → **project**: `/named/fct/project( /repo/people.table, [PersID,Name] )`

- **join**: `/named/fct/join( /repo/events.table as 'event', /repo/people.table as 'people' )`

<table>
<thead>
<tr>
<th>PersID*</th>
<th>Name</th>
<th>Home</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Alice</td>
<td>US</td>
</tr>
<tr>
<td>2</td>
<td>Bob</td>
<td>DE</td>
</tr>
</tbody>
</table>
Example: Basic Query Operations

DB Research: Few basic set operations are sufficient to define a rich relational algebra.

- **restrict:** `/named/fct/restrict( /repo/people.table, Home == 'US' )`
- **project:** `/named/fct/project( /repo/people.table, [PersID,Name] )`

→ **join:** `/named/fct/join( /repo/events.table as 'event',
   /repo/people.table as 'people' )`

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>NY Marathon</td>
<td>2</td>
<td>Bob</td>
<td>DE</td>
</tr>
<tr>
<td>2</td>
<td>Paris Marathon</td>
<td>1</td>
<td>Alice</td>
<td>US</td>
</tr>
<tr>
<td>3</td>
<td>NY Marathon</td>
<td>1</td>
<td>Alice</td>
<td>US</td>
</tr>
</tbody>
</table>
Query Composition: Consumer’s View

“I need the list of countries from which at least one person attended the NY Marathon.”

The app composes the list’s network name:

```plaintext
/named/fct/project(
  /named/fct/restrict(
    /named/fct/join(/repo/events as 'event', /repo/people as 'people'),
    event.Name == 'NY Marathon'
  ),
  [people.Home]
)
```
Query Composition: Consumer’s View

“I need the list of countries from which at least one person attended the NY Marathon.”

The app composes the list’s network name:

```
/named/fct/project(
    /named/fct/restrict(
        /named/fct/join(/repo/events as 'event', /repo/people as 'people'),
        event.Name == 'NY Marathon',
    ),
    [people.Home]
)
```
“I need the list of countries from which at least one person attended the NY Marathon.”

The app composes the list’s network name:

```
/named/fct/project(
    /named/fct/restrict(
        /named/fct/join(/repo/events as 'event', /repo/people as 'people'),
        event.Name == 'NY Marathon',
    ),
    [people.Home]
)
```
“I need the list of countries from which at least one person attended the NY Marathon.”

The app composes the list’s network name:

```
(named/fct/project(
  (named/fct/restrict(
    (named/fct/join(/repo/events as 'event', /repo/people as 'people'),
      event.Name == 'NY Marathon'
    ),
    [people.Home]
  )
)
```

**Derivation Chain:** Primary Data → join() → restrict() → project() → Result
Stepping Through the Derivation Chain...

**Step 1**

### Event ID

<table>
<thead>
<tr>
<th>EventID*</th>
<th>PersID^</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>NYmarathon</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>ParisMarathon</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>NYmarathon</td>
</tr>
</tbody>
</table>

/repo/events

### Pers ID

<table>
<thead>
<tr>
<th>PersID*</th>
<th>Name</th>
<th>Home</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Alice</td>
<td>US</td>
</tr>
<tr>
<td>2</td>
<td>Bob</td>
<td>DE</td>
</tr>
</tbody>
</table>

/repo/people

/named/fct/join

Execution Step 1
Stepping Through the Derivation Chain...

**Step 1**

### /repo/events

<table>
<thead>
<tr>
<th>EventID*</th>
<th>PersID</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>NYmarathon</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>ParisMarathon</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>NYmarathon</td>
</tr>
</tbody>
</table>

### /repo/people

<table>
<thead>
<tr>
<th>PersID*</th>
<th>Name</th>
<th>Home</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Alice</td>
<td>US</td>
</tr>
<tr>
<td>2</td>
<td>Bob</td>
<td>DE</td>
</tr>
</tbody>
</table>

### Execution Step 1

```
/named/fct/join( /repo/events as 'event', /repo/people as 'people' )
```
### Step 2

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>NY Marathon</td>
<td>2</td>
<td>Bob</td>
<td>DE</td>
</tr>
<tr>
<td>2</td>
<td>Paris Marathon</td>
<td>1</td>
<td>Alice</td>
<td>US</td>
</tr>
<tr>
<td>3</td>
<td>NY Marathon</td>
<td>1</td>
<td>Alice</td>
<td>US</td>
</tr>
</tbody>
</table>

/\texttt{named/fct/join( } /\texttt{repo/events as 'event', /\texttt{repo/people as 'people' } )

\texttt{/named/fct/restrict}
Stepping Through the Derivation Chain...

### Step 2

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>NY Marathon</td>
<td>2</td>
<td>Bob</td>
<td>DE</td>
</tr>
<tr>
<td>2</td>
<td>Paris Marathon</td>
<td>1</td>
<td>Alice</td>
<td>US</td>
</tr>
<tr>
<td>3</td>
<td>NY Marathon</td>
<td>1</td>
<td>Alice</td>
<td>US</td>
</tr>
</tbody>
</table>

```
/named/fct/join( /repo/events as 'event', /repo/people as 'people' )
```

**Execution Step 2**

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>NY Marathon</td>
<td>2</td>
<td>Bob</td>
<td>DE</td>
</tr>
<tr>
<td>3</td>
<td>NY Marathon</td>
<td>1</td>
<td>Alice</td>
<td>US</td>
</tr>
</tbody>
</table>

```
/named/fct/restrict( /fct/join( /repo/events as 'event', /repo/people as 'people' ),
                   event.Name == 'NYmarathon' )
```
Stepping Through the Derivation Chain...

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>NY Marathon</td>
<td>2</td>
<td>Bob</td>
<td>DE</td>
</tr>
<tr>
<td>3</td>
<td>NY Marathon</td>
<td>1</td>
<td>Alice</td>
<td>US</td>
</tr>
</tbody>
</table>

```
/named/fct/restrict( /fct/join( /repo/events as 'event', /repo/people as 'people' ),
          event.Name == 'NYmarathon' )
```

```
/named/fct/project
```

Execution Step 3
Stepping Through the Derivation Chain...

**Step 3**

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>NY Marathon</td>
<td>2</td>
<td>Bob</td>
<td>DE</td>
</tr>
<tr>
<td>3</td>
<td>NY Marathon</td>
<td>1</td>
<td>Alice</td>
<td>US</td>
</tr>
</tbody>
</table>

```
/named/fct/restrict( /fct/join( /repo/events as 'event', /repo/people as 'people' ),
    event.Name == 'NYmarathon' )
```

```
/named/fct/project
```

Execution Step 3

```
/named/fct/project( /fct/restrict( /fct/join( /repo/events as 'event', /repo/people as 'people' ),
    event.Name == 'NYmarathon' ), [people.Home] )
```
Conclusion

Limitations:

Static Content Only

"Classic ICN"

publisher

consumer

Offline Sync

"Classic ICN"

Offline Sync:
Namespace and Data Format Conventions (empirically)
Conclusion

Limitations: Static Content Only

Offline Sync: Namespace and Data Format Conventions (empirically)
Conclusion

New: Dynamic/On-Demand/Derived Content

"Classic ICN"

Publisher

Consumer

Offline Sync

NFn

"Classic ICN"

Named Function's Semantics their I/O Formats
Conclusion

**New:** Dynamic/On-Demand/Derived Content

**Additional Offline Sync:** Named Function’s Semantics their I/O Formats
Conclusion

Minimized Offline Synchronization

"Classic ICN"

Data Modeling & Querying

NFN

“Classic ICN”

publisher

consumer

Offline Sync
Conclusion

New: Minimized Offline Synchronization

Offline Sync: Relation Type Schema Names, Basic Query Functions (join, restrict, project)
Conclusion

Relational Modeling just as an example: Select a proper data model for your application and put it inside the network.
Q & A
Additional Slides
Trust Assumptions
Network Model

Application Layer

Relational Model & Query Lang.  ...

Data Modeling & Querying Sublayer

Named Function Networking  ...

Computation Sublayer

Hierarchical Naming, Interest and Data Packets, Caching/Multicast, Forwarding/Routing, Congestion Control, ...

Naming & Lookup Sublayer

Network Layer

Link Layer