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

Claudio Marxer
University of Basel, Switzerland
claudio.marxer@unibas.ch

Christian Tschudin
University of Basel, Switzerland
christian.tschudin@unibas.ch

ABSTRACT

Recent realizations of the ICN principle organize content in a hierarchical namespace. We argue that this addressing mode has shortcomings because a single document could be part of several data collections. For instance, Joe's record of his New York Marathon run might be published as a content object with the name `/repo/events/NYmarathon/record1234` but would also fit into `/repo/users/Joe/record1234`. Even further, the content of documents can be very multifaceted such that not all details –e.g. spacial coordinates, timestamps, rankings– can be made available on the name surface.

In this paper we show that CCN-style networks enriched with active elements, i.e. content processing/producing entities, can overcome these problems. Exemplarily, we adopt relational data modeling concepts to organize named data and deploy Named Function Networking to implement content addressability which goes beyond the scope of pure, i.e. passive, CCN.

CCS CONCEPTS

• **Information systems → Relational database model**; **Relational database query languages**; • **Networks → In-network processing**; **Network protocol design**; **Cloud computing**;

KEYWORDS

ICN, CCN, NDN, NFN, in-network processing, relational data management, namespace engineering, addressability, query language

ACM Reference format:
Claudio Marxer and Christian Tschudin. 2017. Improved Content Addressability Through Relational Data Modeling and In-Network Processing Elements. In *Proceedings of ICN '17, Berlin, Germany, September 26–28, 2017,* 7 pages.
DOI: 10.1145/3125719.3125735

1 INTRODUCTION

A Content-Centric Network is also a database. Or, more specific, a key-value store: Base elements are data chunks (values) which are organized in a hierarchical namespace (keys). The query interface is a lookup mechanism which mediates chunk-wise data access. From the information system community we can learn that choosing the right data model is a crucial step in application development. Unfortunately, on CCN-level there is no choice to organize data in another manner than in the aforesaid "hierarchical key"-value-model. For many use cases this has shortcomings:

– Even though not supported by CCN, domain-logic often requires that a single document is part of many collections. For instance, assume Joe wants to publish a document which comprises data (time-location track, ranking, …) about his participation in the New York Marathon 2017. Different prefixes are in question but only one can be chosen:
  - `/personal/joe/2017/events/...`
  - `/personal/joe/marathons/...`
  - `/organizer/NYmarathon/2017/ranking/1/...`
  - `/organizer/NYmarathon/2017/joe/...`

– Not all characteristics of a document can be made visible on the CCN name level and must therefore be excluded when defining the "name surface". Even few characteristics which should be integrated into a content name can lead to overlong identifiers. Beyond that, there is usually also implicit information which is relevant for certain access patterns, but which can't be part of the "name surface" (e.g. from time-location track: length, average speed, trans-boundary?). Also, for privacy reasons it makes sense to minimize the information exposed in content names.
– Name-based solutions for data integrity [9] and confidentiality [10][7] grant access rights and signing authority according to name patterns (schemata). Granting such rights for a document which could be available under multiple prefixes makes right management tedious. If characteristics should be taken under consideration which are not even visible on the "name surface", name-based security solutions are limited.
– The CCN network layer organizes access of known documents but lacks of effective index structures to discover all data chunks within a certain collection (sub-tree).

In this paper we show that with appropriate data modeling and in-network processing elements, the accessibility and addressability of content can be improved. We illustrate this along a use case in which records of marathon events, athletes and participations are available. We show that it is possible to:
– discover all records of a certain type (e.g. all athletes).
– handle a record collection (e.g. athletes) by different attributes like "birthday" or "home country" (or combinations).
– selectively address some attributes within a record or a collection of records (e.g. omit "home country" from an athlete's record).
– combine different types of records (e.g. fusion of an athlete's record with the record of a participated event).
– do combinations of these, like: First and last name of all Swiss participant who joined a certain marathon event.

In Section 2 we introduce Content-Centric Networking, Named-Function Networking and the basics of relational data modeling. In

ICN '17, Berlin, Germany

DOI: 10.1145/3125719.3125735

Sections 3 and 4 we present an adoption of relational data management to CCN and a fitting NFN-based query language. Section 5 is about general-purpose processing beyond simple data querying. Management of meta-data is treated in Section 6. Before we conclude in Section 8, a discussion as well as future work is stated in Section 7.

2 BACKGROUND

In this section we introduce Content Centric Networking and the Named Function extension. Further, the basics of relational data management are introduced.

2.1 Information-Centric and Named-Function Networking

In *Information Centric Networking (ICN)* content chunks are treated as base objects and identified with names which have no reference to network locations (servers). *Content Centric Networking (CCN)* [5] is a recent flavour of ICN which is implemented by several projects such as *Named Data Networking (NDN)* [2] or *CICN* [1] (formerly *CCNx*). These derivations have only minor conceptional differences but operate with incompatible packet formats. Whenever we refer to CCN, our explanations are adoptable to all derivations. Only if conceptionally relevant we point out differences.

In CCN, content names are organized in a hierarchical namespace (e.g., `/this/name/has/five/components`). Data chunks, so called *content objects*, are essentially a unique and unchangeable binding of certain payload to a name. This means, content objects constitute a self-contained peace of information without necessarily having some relation to a specific network location. An *interest packet* basically carries a name and is sent towards the network by a data consumer in order to express interest in the named content object. Since it is not relevant for this work how the network tries to spot and return the matching content object, we keep this transparent.

Named Function Networking (NFN) augments CCN with an in-network content derivation machinery [8]: Names are computation expressions which include data *and* function names. For instance, a data consumer can request the result of the named function `wordcount()` applied to the content object `poem.txt`:

```
/fct/provider/wordcount( /a/nice/poem.txt )
```

The content object with the name `/fct/provider/wordcount` contains bytecode of a JVM function which can be instantiated and called by responding execution locations. Since an expression is the network name of the result, a function call can also be the argument to another function call (function chaining):

```
/fct/prvdr/wordcount(/first/verse(/a/nice/poem.txt))
```

Such expressions instruct the network to either get hold of the corresponding final result (if already cached), or to let the network orchestrate its computation. In the latter case, NFN-enabled network nodes decompose (=reduce) the expression and recursively work through all sub-expressions. Once a (sub-) result is obtained, it is turned into an content object whose name is exactly the (sub-) expression itself. Also, these intermediate results are cached because they might be reused.

In the wire format, computation expressions are formulated in an extended version of λ-calculus [3] which additionally has a `call` operator to express a named function call. For instance, the first example above looks like (in λ-calc; not yet as a network name):

```
call 2 /fct/provider/wordcount /a/nice/poem.txt
```

The operator `call` is followed by the number of arguments which in our case counts the following function name and the name of the passed poem document (more arguments would be possible). By making use of function abstraction and application, the λ-expression can be rewritten such that the function name is prepended (λ is encoded as `@`):

```
/fct/provider/wordcount @x call 2 x /a/nice/poem.txt
```

Also, an argument's name could be prepended instead:

```
/a/nice/poem.txt @x call 2 /fct/providr/wordcount x
```

To gain routable CCN-names, these variants with prepended function or data names are mapped to the CCN namespace. To avoid confusion with name component separators, the character "`/`" within a name component is by convention written as "`|`":

```
/fct/providr/wordcount/@x call 2 |a|nice|poem.txt x
```

As this example shows, there exist several ways to map a λ-expression to a CCN-name. Either the function name but also an argument's name can be prefixed in order to guide the computation expression –capsulated in an interest packet– to an on-demand computation execution node. The decision influences whether the result is produced by an NFN-enabled node which is on the path to the data or to the function repository. Though, by name-rewriting, NFN accomplishes to influence the CCN-forwarding system, i.e. execution location spotting mechanism, without changing the core network protocols.

2.2 Relational Data Modeling

Proposed by E. F. Codd in 1970, the *relational data model* has become a well-known and perpetuated way to structure information [4]. The pivotal element of the relational data model are *relations*. Intuitively captured as tables like depicted in Figure 1, a relation is actually a set of certain tuples. Tuples are rows in the table-representation and columns are called attributes. Each column has an attribute name (top cell, in bold). The primary key, one or more attributes marked with $*$, must for each tuple be unique within a relation such that the rows are distinguishable and referable. Through a foreign key (marked with $\hat{}$), a tuple within one relation can refer to a tuple within another relation. For instance, the relation on the right side of Figure 1 refers through the foreign key `PID^` to a tuple in the relation on the left side or its primary key `PID*`, respectively.

3 RELATIONAL NAMED DATA

In this section we introduce *Named Relations*, *Relation Schemata* and *Relation Type Schemata*. A named relation is an addressable chunk of relational data while a relation schema defines the structure of such a document. Both are linked together in a relation type schema.

3.1 Named Relations

A *named relation* is a set of special tuples which are carried by a content object. Figure 1 shows two examples. The relation `/repo/people` qualifies per tuple a person, i.e., state its name, home country and assigns a unique ID. The relation `/repo/events` provides information on who participated which marathon events. Each tuple

states the attendance of a certain person in an event. Either consists of the event's name, the participant's ID and a unique ID for the particular participation. The examples show that the top row of each relation gives some information about the shape of the tuples, highlight unique tuple IDs (marked with *) and references to IDs of other relations (marked with ^). It is thinkable that further relations supplement the picture. For instance, another relation could state running time and ranking for each participation or provide further details about events.

PID*	Name	Home
1	Alice	US
2	Bob	CH

EID*	PID^	Name
1	2	NYmarathon
2	1	ParisMarathon
3	2	NYmarathon

Figure 1: Named Relations. Left: `/repo/people`. Right: `/repo/events`

3.2 Relation Schemata

The comprehensive description of a named relation is published as a separate document which is called a *relation schema*. If intended to be an application-independent data exchange format, relation schemata are likely published by standardization bodies. Figure 2 shows the relation schemata for the named relations introduced in Figure 1. Each line describes one attribute: Attribute name and its data type are separated by a colon. Attribute names are marked with * if they constitute a primary key. Foreign keys are marked with ^ and further contain a pointer to the referring relation schema. As defined in the relation schema `/ietf/relations/people` (left, line 1), an integer value named `PID` constitutes the primary key. The relation schema `/ietf/relations/events` (right, line 2) defines a foreign key to refer to event participants `PID`s from a people relation.

```
PID*:Int
Name:String
Home:String
```

```
EID*:Int
PID^:Int -> /ietf/relations/people
Name:String
```

Figure 2: Relation Schemata. Left: `/ietf/relations/people` Right: `/ietf/relations/events`

3.3 Relation Type Schemata

Per publisher of named relations there exists a *relation type schema (RTS)* which assigns a relation schema to each named relation. Figure 3 shows an example. Each rule captures certain documents according to a name-based pattern and states the according relation schema document. In our example, the first two rules (lines) link the relations from our use case to their relation schemata. The following two lines show that a rule can also capture multiple relations (grep style pattern matching) which should all be linked to a certain relation schema.

Repositories must make their relation type schema available under a canonical name. For simplicity, we stick with the name `/repo/rts` for the RTS in our use case.

```
# published as /repo/rts
/repo/people            ->   /ietf/relations/people
/repo/events            ->   /ietf/relations/events
/repo/circuits/[^/]+    ->   /ietf/relations/circuits
/repo/circuits/old/.*   ->   /ietf/old/relations/circuits
```

Figure 3: Relation Type Schema: `/repo/rts`

4 QUERY LANGUAGE

When it comes to accessing named relational data, it is in many cases dissatisfying that pure CCN merely allows to address data at document-level. What if a data consumer is just interested in some parts of a named relation? Or in information spread over several documents? In this section we introduce a NFN-based query language to address information in an in-relation and cross-relation style. Our solution allows to request results such as: What are the names of all Paris Marathon participants? Which are the countries from which the individuals listed in the people database come from?

4.1 Basic Operations

Research in the late 1960s has shown that very few basic set operations are sufficient to define a rich relational algebra [4]. Accordingly, we identify a couple of named functions which connect or reduce named relations. Since these functions always derive new relations from existing ones, proper combination (chaining) allows to address a rich set of selections inside and cross named relations. Inspired by the common language in the databasing field, we call these functions `join`, `restrict` and `project`.

Joining Relations. Joining two relations means to unite them into a single relation. The new relation is the combination of the joined relation's tuples where the values of primary and foreign keys are equal. As the following example shows, `join` takes the names of two relations (and a short name).

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

Figure 4.a depicts the output which combines `/repo/events` and `/repo/people`. The attributes are renamed such that attribute names which occur in multiple relations become distinguishable (compare `event.Name` and `ppl.Name`). The former primary key of the people relation is marked with " such that further joining remains possible.

Restricting a Relation. A restriction of a relation is a selection of certain tuples according to a given criterion. In the following example the named function `restrict` extracts from the named relation `/repo/people` all people whose home country are the United States. The first argument specifies the name of the relation to restrict while the second one constitutes the criterion which a tuple must fulfill to be preserved.

```
/restrict( /repo/people, Home == 'US' )
```

Figure 4.b shows the result (black) and the erased tuples (gray).

Projecting a Relation. Projecting a relation means to take some attributes and erase all others. The following example calling the named function `project` takes from the named relation `/repo/events` the attributes `PID` and `Name`. The attributes to preserve are passed as second argument in square brackets.

```
/project( /repo/people, [PID,Name] )
```

In Figure 4.c the produced relation (black) as well as the erased parts (gray) are shown.

event.EID*	event.Name	ppl.PID"	ppl.Name	ppl.Home
1	NY Marathon	2	Bob	CH
2	Paris Marathon	1	Alice	US
3	NY Marathon	2	Bob	CH

(a) Joined Named Relations

PID*	Name	Home
1	Alice	US
2	Bob	CH

(b) Restricted Named Relation

PID*	Name	Home
1	Alice	US
2	Bob	CH

(c) Projected Named Relation

Figure 4: Basic Operations on Relations. (Erased parts in gray)

4.2 Chaining Example

Since `join`, `restrict` and `project` map relation(s) to one new relation, arbitrary function chaining is possible. Figure 5 illustrates an expression, i.e., name of an operation chain's result. The expression/name addresses all countries from which at least one person participated in the NY Marathon. All basic operations are incorporated in this chain.

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

Figure 5: Chained Operations on Named Relations: `/repo/..` → `join(..)` → `restrict(..)` → `project(..)` → `..`

5 GENERAL-PURPOSE PROCESSING

So far we have presented a query language to combine relations and/or extract specific portions. Further functions such as `sum`, `min`, `max`, `avg`, `count` (and also data/application-specific ones) are useful to query a new class of computation results: Who draws the highest wage among all employees? What is the wage bill of a company? How many employees earn more than the average wage?

Just as the three basic query functions introduced in the previous section, general-purpose functions process one or more relations and output another named relation. In this section we exemplarily describe some to the aforementioned, rather general, functions. However, more data/application-specific functions, for instance conversion of measurement units or timestamp formats, are also covered by these examples.

5.1 sum

The function `sum` accumulates all values in a certain column. As the following example shows, the arguments are a relation name and an attribute name.

PID*	Name	Wage
1	Alice	10000
2	Bob	1000
3	Charly	20000
4	Debby	1000

Figure 6: Employee-Wage Relation: `/repo/employee`

```
/sum( /repo/employee, 'Wage' )
```

The passed relation is shown in Figure 6 and the output is depicted in 7.a

5.2 max

The function `max` extracts the tuple(s) with the highest value in a certain column. The following example names the tuple (row) of the employee(s) with the highest income.

```
/max( /repo/employee, 'Wage' )
```

The resulting relation is depicted in Figure 7.b.

5.3 count

The function `count` identifies how many times the values in a certain column occur. The example identifies which amount occurs how many times on a pay roll.

```
/count( /repo/employee, 'Wage' )
```

The result is shown in Figure 7.c.

32000

(a) `sum(..)`

3	Charly	20000

(b) `max(..)`

1	10000
2	1000
1	20000

(c) `count(..)`

Figure 7: Derived Named Relations (tuples only, attribute names are transparent).

6 META-DATA FOR DERIVED RELATIONS

Not just primary but also derived named relations must be described by a relation type schema. It is obvious that not for every query a fitting relation type schema (RTS) can be pre-defined. This section is about how to produce according relation type schemata on-demand.

6.1 Derived RTS for Basic and General-Purpose Operations

Input and output relation of `restrict` are described by the same RTS. The RTS which describe input and output of `project` are quite similar: All attributes which are not explicitly selected in the function call's second argument, must be erased from the input-RTS in order to derive the output-RTS. The RTS of a `join`'s output must union both input-RTS according to the following rules: The derived RTS must include all attributes of both input-RTS. All attributes must be re-named such that it is clear from which input relation they origin. Primary key attributes remain primary and former foreign key attributes are marked with " instead of ^.

Depending in the derivation logic, a general-purpose relation processing function must also accompanied by an output-RTS. Some can be predefined (e.g. RTS for the relations in Figures 7.a and 7.c) while others must be derived from the input-RTS (e.g. for the relation in Figure 7.b).

6.2 Naming Conventions

One simple solution to name relation type schemata of non-primary, i.e., derived named relations, is to publish an accompanying RTS-derivation function for each basic and general purpose function. If for instance `/fct/restrict` and `/fct/count` name relation derivation functions, `/ftc/rts/restrict` and `/fct/rts/count` could produce derived RTS. The following two function chains then name a derived relation and the according RTS:

```
/fct/count( /fct/restrict( .. ) , .. )
/fct/rts/count( /fct/rts/restrict( .. ) , .. )
```

7 DISCUSSION AND FUTURE WORK

In this section we discuss our contribution and name topics for future work.

7.1 Interior and Exterior Content Considerations

We have introduced the pictorial term "name surface" to indicate that content objects expose certain information through data names but that there is also deeper, interior knowledge carried by the payload field. CCN's working assumption is that exterior surface information is sufficient to orchestrate content synchronization between data publisher and consumer. In this paper we question this assumption and argue that mechanisms are required which blur this firm boundary. This is useful because we think that a hierarchical namespace is not well suited to structure highly linked and multifaceted data. Our proposal is to introduce network entities which bridge the name surface with interior knowledge by actively looking into content objects and considering meta-data.

Our solution is declarative in the sense that primary content has not especially informative names but that consumers hold recipes to form à-la-carte names. It is the data consumers who "lift" arbitrary interior data characteristics to the name surface but the task of active elements to produce the queried content on-demand. The recipes to query à-la-carte content is given by the query language and document type schemata which expose the interior structure of content.

7.2 Network Model

Compared to state-of-the-art (TCP/IP) networking, CCN blends traditional application layer aspects into the network layer. With in-network computations like NFN and data querying languages on such a basis, we propose to go another step into this direction. Figure 8 shows one possibility to capture our proposal in a network model. The network layer is conceptionally subdivided into three sublayers:

– The bottom sublayer is constituted by common CCN which comprises services and responsabilities like hierarchical content naming, pull-based content lookup via interest and data packets, multicast/caching, forwarding/routing and congestion control. The interface to this *naming & lookup sublayer*, i.e. propagating an interest to receive content, can be used directly by application developers (as usually done today) but also constitutes the basis for the next-higher sublayers.

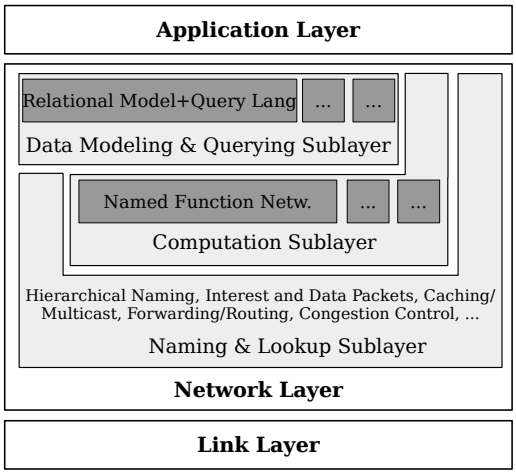

Figure 8: Network Model: Common CCN as a basis for in-network computations as a basis for in-network data modeling and query execution.

– The *computation sublayer* enables the subjacent content lookup service to return not just static, i.e. "pre-existing", content but also to deliver new content derived according to a given "recipe". The key idea to interface with the underlying naming & lookup sublayer is the encoding of functional computation expressions into hierarchical CCN names. In other words, a computation expression (in computation sublayer) constitutes the name of the result (in naming & lookup sublayer). As NFN shows, a computation sublayer can be deployed without changing the underlying naming & loopup sublayer (= CCN network). Still, tricks like expression/name rewriting, as performed by NFN-enabled nodes, allow to enhance pure-CCN features (e.g. routing for query optimization purposes). To the best of our knowledge, NFN-Scala [1] is currently the only implemented system which conceptionally fit into this sublayer. Still, other in-network computation solutions could be deployed in parallel.

– Our proposal in this paper is to introduce another *data modeling & querying sublayer* on these content lookup and computation services/sublayers. As we have exemplarily illustrated by means of the relational data model, this sublayer is concerned with (meta-) data organization and offers an information querying service to the application layer. While (meta-) data organization is directly implemented on top of the service offered by the naming & lookup sublayer, the query execution part is based on the computation sublayer. We consider relational data management as just one way to organize and query content. Other data models known in the database field but also data structures as used in computer programming might be potential candidates for coexisting on the data modeling & querying sublayer.

[1] https://github.com/cn-uofbasel/nfn-scala

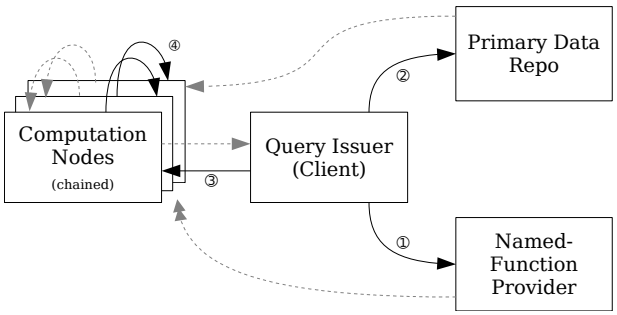

Figure 9: Trust relationships and logical data flow: "A → B" means that A trusts B to offer correct data, bytecode or results. "A ⇢ B" means that B consumes data provided by A.

7.3 Security, Privacy and Trust

Trust Model. Since NFN introduces function providers and computation nodes, the trust model is more complex than in common CCN. Figure 9 shows the logical components (boxes) and their trust relationships (solid arrows). When formulating a query, the issuer directly trusts the providers of all incorporated functions to deliver bytecode which operates according to the specification (relationship ①). Also, the issuer trusts the provider of all processed named relations to deliver correct primary data (relationship ②). When requesting the result of a query, the issuer trusts the computation node which immediately produces the final result (relationship ③). This trust relationship is twofold: First, the issuer is confident that the computation node does his portion of the work properly. Second, it is trusted that the computation node incorporates intermediate results from others only if he assesses the producing node as trustworthy (relationship ④). Thus, trust in computation nodes is transitive. This means that trusting one computation node implies to recursively trust all computation nodes which are trusted by at least one trusted computation node.

We think, ensuring trusted provenience (authenticity) and integrity of named relations, functions and results via cryptographic signatures is a trivial task if a proper identity management system is in place. This is a general topic for future work in the CCN field. A weak point, so far, is the foundation for (transitive) trust in computation nodes. We think this must be tackled for NFN (and other distributed in-network computations) in general. Solutions might be technical (e.g. verifiable computing) or non-technical (e.g. service level agreements [6]).

Access Control. In this paper, we evince increasingly structured and annotated named information together with a declarative query language. Since the query format is built into the CCN namespace, we gain highly descriptive content names which are an ideal foundation for name-based access control (NAC) [10] solutions. With *schematized access control* [7] these authors described a variant of NAC in which access for a content object is granted to a certain principal (e.g. a user or a group) iff the content's name is caught by a name-pattern matching rule in the principal's capability schema. One novelty of that work is that pattern-matching rules operate not just on name component but also on λ-calculus level. Therefore, schematized access control - adjusted to named relations - allows to to exhaust all information exposed by the descriptive NFN-based queries. We think a proper integration of this work with schematized access control allows a data owner to formulate fine-grained access-control policies such as: A certain principal is allowed to access the personal record of an employee only if the birthday is removed (or technically: if the query includes a project function which excludes the attribute "birthday"). Showing feasability is a topic for future work.

Name Privacy. Highly descriptive names are beneficial for name-based/schematized access control (as showed in the previous paragraph) but also rise name privacy concerns. To overcome these, it will be necessary to identify the most privacy-relevant parts of a query and develop solutions of hide them. For instance, a query which contains hash values of attributes (instead of plain ones), hides some privacy-compromising information from relaying network nodes but does not substancially affect execution complexity, caching or NFN name-rewriting (NFN-specific routing). Because queries are generically constructed, it is potentially feasible to automate the translation to less verbose names (and reverse). In plain CCN where each application defines its own hierarchical name format, an general solution - which must also be compatible with any application logic - might be harder to achieve.

8 CONCLUSION

It is worth asking why traditional application concerns like data organization and processing should be detached and reinterpreted as a network service. We think application-wise data siloing and, if at all, purpose-specific data APIs do no longer describe today's mindset: In times of increasingly connected devices and applications, the management of information should be treated across application boundaries. In our opinion, an information-centric network which can manage structured, annotated and interlinked information is the right approach to build such a "global" integration point. With information we mean in the first instance "raw material" to satisfy on-demand needs but not application-specific selections nor content or data in a certain shape (data format, measurement units, currencies …). It should be a network service to bridge the gap between this "ground information" and the application's needs. This means, a query language must be in place to address application-specific content, i.e. selected information in a desired format.

With this mindset, we approached the Content-Centric Networking architecture along the classic questions edited in the database or information systems field: What are good models to structure certain information? What portions do data consumers need to access? How to address/query these portions? How to orchestrate query execution in an effective way? We argued that the pure CCN network-layer is in database terms a distributed key-value store and claimed that other data models (e.g. relational data) are not effectively captured. Our contribution to investigate this issue is twofold: Conceptionally, we described a network model which aims to support rich structuring and querying of information inside a content-centric network. On the practical side, we exemplarily described a relational schema and a NFN-based query language. A key insight of this work is that a "data modelling layer" on its own is not sufficient because also active, processing elements are required to support application-specific on-demand querying.

ACKNOWLEDGMENT

We thank Gareth Tyson (Queen Mary University of London) for comments during the revision process.

REFERENCES

[1] "Community ICN" (CICN) Project. *https://wiki.fd.io/view/Cicn.* 2017.
[2] "Named Data Networking" Project. *https://named-data.net.* 2017.
[3] Alonzo Church. 1936. An Unsolvable Problem of Elementary Number Theory. *American Journal of Mathematics* 58, 2 (April 1936), 345–363.
[4] E. F. Codd. 1970. A Relational Model of Data for Large Shared Data Banks. *Commun. ACM* 13, 6 (June 1970), 377–387. DOI: https://doi.org/10.1145/362384.362685
[5] Van Jacobson, Diana K. Smetters, James D. Thornton, Michael F. Plass, Nicholas H. Briggs, and Rebecca L. Braynard. Networking Named Content. In *Proceedings 5th International Conference on Emerging Networking Experiments and Technologies (CoNEXT '09).*
[6] Claudio Marxer, Michal Krol, Balazs Faludi, and Dennis Grewe. 2017. *Escrowed Service Agreements as the Foundation of Trust in Named-Function Networking.* Discussion Report. GI-Dagstuhl IoT Hackathon 2017.
[7] Claudio Marxer and Christian Tschudin. 2017. Schematized Access Control for Data Cubes and Trees. In *Proceedings of the 4rd ACM Conference on Information-Centric Networking (ACM-ICN '17).* ACM, New York, NY, USA. DOI: https://doi.org/10.1145/3125719.3125736
[8] Manolis Sifalakis, Basil Kohler, Christopher Scherb, and Christian Tschudin. 2014. An Information Centric Network for Computing the Distribution of Computations. In *Proceedings of the 1st ACM Conference on Information-Centric Networking (ACM-ICN '14).* ACM, New York, NY, USA, 137–146. DOI: https://doi.org/10.1145/2660129.2660150
[9] Yingdi Yu, Alexander Afanasyev, David Clark, Van Jacobson, Lixia Zhang, and others. 2015. Schematizing Trust in Named Data Networking. In *Proceedings of the 2nd International Conference on Information-Centric Networking (ACM ICN '15).* ACM, 177–186.
[10] Yingdi Yu, Alexander Afanasyev, and Lixia Zhang. 2016. *Name-Based Access Control.* Technical Report. Internet Research Lab, UCLA.