ONLI OS

CLOUD OPERATING SYSTEM

ONLI®
GENERATION 2

Onli is a new way to store value move it around and keep it safe.

Editor
DHRYL ANTON

Art Directors
Mark Eternites, MBA

Content Writter
PETER J. HAXEL
DHRYL ANTON
MICHAEL MCFALL

Photography
SHUTTERSTOCK
UNSPLASH
UI DEISGN BY DHRYL ANTON

Issued By
THE ONLI CORPORATION

Notice

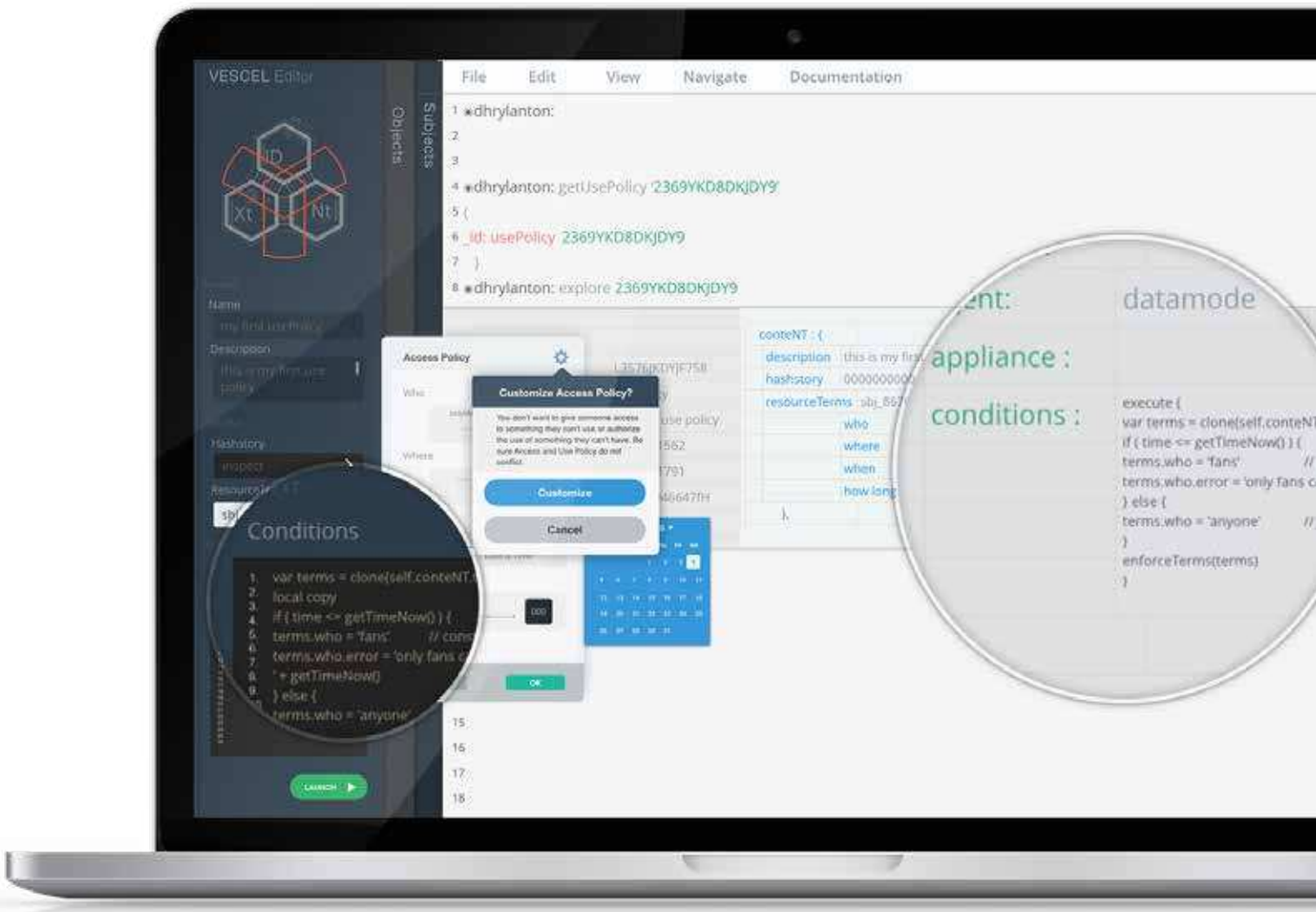

VESCEL Editor
File
Edit
View
Navigate
Documentation
Objects
Subjects
Name
Description
Hashstory
Resources
Access Policy
Customize Access Policy?
Customize
Cancel
Conditions
var terms = clone(self.conteNT
local copy
if (time <= getTimeNow()) {
terms.who = 'fans'
terms.who.error = 'only fans
' + getTimeNow()
} else {
terms.who = 'anyone'
LAUNCH
1 dhrylanton:
4 dhrylanton: getUsePolicy '2369YKD8DKJDY9'
5 (
6 _id: usePolicy '2369YKD8DKJDY9'
7)
8 dhrylanton: explore 2369YKD8DKJDY9
conteNT : (
description
hashstory
resourceTerms
who
where
when
how long
15
16
17
18
ent:
appliance :
conditions :
datamode
execute (
var terms = clone(self.conteNT
if (time <= getTimeNow()) {
terms.who = 'fans'
terms.who.error = 'only fans
} else {
terms.who = 'anyone'
}
enforceTerms(terms)
}

TABLE OF CONTENTS

#onlios #forward

FORWARD

In the summer of 2010, Dhryl Anton, Michael McFall and Katherine Anton were trying to use web technologies to build an exchange for non-performing assets. Non-Performing assets are credit instruments like loans that are in default or at risk of defaulting. NPAs are sold at a discount by financial institutions to qualified investors. The nature of these transactions require privacy and the strictest measure of confidentiality. While NPAs negatively affect institutions financially, the damage to their brand from information leaks was a greater risk. Security and confidentiality was therefore the highest priority in selecting a technology. After many heated arguments between the founders as to why security was such a problem with the web they realized that while computers have changed in the last 60 years, the paradigm used to store data had not.

Data on most computers are stored using a hierarchical model. Hierarchies are an organizational structure where every entity in the organization, except one, is subordinate to a single other entity. In computers, this subordination translates to control of all that is within the domain of sub-entities. The nature of a hierarchy is that it is arranged in order of rank. Security in hierarchies is therefore about controlling “access”, but there will always be a root that has rank over everything within its domain. Hierarchies also don’t scale well. As a result certain realities emerge. To be reliable computers must make a copy of data so that it is within their domain. This “quirk” of hierarchies is what the Web is built on. It is from this insight that Onli OS (CloudMoDe) was born.

The world of computing has changed our lives in many ways. Computing used to be a tool experienced while sitting at a desk. Now, in 2010, technology goes with us everywhere. Soon even the word office will sound like something your father went to. We are moving from the desktop era of computing to the distributed era of computing, The next phase of the information revolution has already begun. Everything is becoming more connected. It is time to make a shift in thinking, from desktop mode to cloud mode.

Onli Os (CloudMoDe) is a new kind of cloud that consists of a Semantic File Storage System, a Content Management System and a Content Delivery Network, all working as one secure computing service. At the heart of Onli Os (CloudMoDe) is an elegant ground breaking technology that revolutionizes how you store and organize data. We call the process, making your data “intelligible”.

Our vision is to inflate a new kind of space on the internet. A place where data is private-by-default. The mission is to give content creators control over their content. We see the company as a utility company that makes a special kind of data: Intelligible Data. We believe that data, much like electricity in the Second Industrial Revolution of the late 19th century, is powering a new era in human development. Intelligible Data can be used to create new disruptive models of computing in any industry. Models that don’t just disrupt the status-quo. They make the old models obsolete.

Supplemental November 05 2020
Along the way we discovered that the intrinsic value of intelligible data is the ability to quantify uniqueness. From the wikipedia: “ /In mathematics and logic , the term “uniqueness” refers to the property of being the one and only object satisfying a certain condition. This sort of quantification is known as *uniqueness quantification* or *unique existential quantification*./ “ Uniqueness Quantification as defined here is the ability to add a property to data that maintains a single global state across a network of connected devices. This makes it possible to create something on a computer of which there can only be one.

#onlios #thesis

THESIS

in·tel·li·gi·ble

/in'telijəbəl/able to be understood; comprehensible.

1. DEFINITION

Onli Os (CloudMoDe) makes data intelligible. We do that by putting it through a proprietary manufacturing process Let's begin by understanding what we mean by the term "intelligible . We learn from philosophy that in order for something to be intelligible it must possess at least four specific properties:

***Syntax**

-syntax is the way in which things are put together.

it must possess a useful structure.

*** Semantics**

-it must be able to define the relationships between things

*** Perspicuity**

from Latin per- + -spicere: per; the stem of perspicere to look or see through and, the combining form of specere to look; inspect.

It must be able to be reached and inspected.

*** Lucidity**

it must be capable of being used, heard or seen clearly.

Software is often used to perform tasks that relate to the real world. In so doing it must model “real world”, objects such as products, suppliers, customers, and processes such as recording orders, scheduling delivery and tracking sales. A Data Model is a description of the objects and processes together with their properties and relationships as would be represented in software. A Data Model explicitly determines the structure and organization of data. Since the context of our inquiry is software and therefore what we are working with is a model, our first task then is to come up with a model that is robust enough to include the properties that make something intelligible.

When we transpose these concepts into the context of a data model, we get four specific and defined qualities that our model of data must possess in order to be intelligible.

* Syntax
    * It must have a consistent structure that holds throughout the model

* Semantics
    * It must have a uniform way to model relationships, which tend to be recursive.

* Addressability -Perspicuity
    * It must be able to be reached and inspected, which we will call addressability.

* Accessibility-Lucidity
    * It must be capable of being used, heard or seen clearly, which we will define as accessibility.

We now have a working definition of what the concept of intelligible is in our context. In order for data to be intelligible we must imbue it with these properties.

2. THE SYNTATIC MODEL

syn·tax

/'sin¸taks/a set of rules whereby something is put together or an analysis of this.

From the philosophy of Quantumism we learn that to form knowledge of a thing, we must first ask of a thing what is it. In other words, we must solve the problem of identity.

Data modeling is arguably the most boring discipline in the world of computer science. Code is far more exciting. Graphical representations and UI are downright sexy. Yet the data model is the very foundation of all computing. From the conceptual to the logical to the physical, models determine how data is written, stored, read, retrieved and used. The data model you use has a significant effect on cost, performance, security and usability.

A data-model is a structure into which you put data and from which you derive meaning. Eg: let's say we have information: "The color of the sky is blue". We need to store this information in a manner it can be retrieved, now it is changed into data. How we store data has a structure. 1=[the color], 2=[of the], 3=[sky], 4=[is] 5=[blue]. This affects how we retrieve it. We ask for the blocks 1 through 5. Implied in that request is consecutive order, that order or organization must also be stored. If we store the data without the "order" then when we request the data and get back 3=[sky], 4=[is], 1=[the color], 2=[of the], 5=[blue]. When we translate this into information, the meaning you derive from 'Sky is the color of the blue", which is completely different from what we stored, which was /"The color of the sky is blue"./ Computers are machines and need specific, detailed and clearly defined instructions. The order or organization of the blocks into which to put the data, is actually, the most important piece of "information" in the example. The "structure" determined the output and that profoundly affects the information from which meaning is derived. How you organize data, the structure, the model, is the foundation of computing.

/"Organization is what gives potentiality to a thing" -The Philosophy of Quantumism/
Consequently the first step on our journey is to get a sense of the most fundamental organization, the atomic unit, the basic building block, the syntax that is to say the

elements that can be put together to form our system.

The Onli Os (CloudMoDe) Atomic Unit

The fundamental building block of Onli Os (CloudMoDe) is called an Element. An element has an entity (the class), an attribute (the naming part) and a value (the expression). At the bottom is an E-A-V model, which makes it so you can implement Onli Os (CloudMoDe) on almost any object data store.

ELEMENT =	Entity	Attribute	Value

Onli Os (CloudMoDe) then has a foundational layer on top of the entity-attribute-value triplet. Elements are organized into *Components*. Components are three specific and defined sets of elements:

* identiTY,

a set of elements that quantify what uniquely determines something,

/It answers the question what am I, whose am I, what identifies this object as this object ?/

* conteNT

a set of elements that quantify what makes up or describes a thing,

/It answers the question what does this object contain ?/

* conteXT.

a set of elements that quantify the role and use of the thing,

/it answers the question what is this object used for in what context ?/

Everything stored in Onli Os (CloudMoDe) is a Component as this is the atomic structure of Onli Os (CloudMoDe) .Everything that is stored in the datastore has an identity, content and context.

Component =	Entity	Attribute	Value
	identiTY	name	dhryl
	conteNT	phone	2125551212
	conteXT	ipAddress	123.456.78.890

This is the basic data model of Onli Os (CloudMoDe) . This basic data model forms a syntax - /literally, the elements that can be put together to form/, which everything is expressed.

This basic Syntax, this basic way of organizing things, it turns out, is a model robust enough to model almost anything. When you look around the natural world everything has a unique identity, it contains something and it has a context in which it is used. It is a simple idea when you first conceive of it. It is profound when you realize that something so simple is so robust.

Being able to adequately “model” the real world is ultimately the objective of computer science and this elegantly simple model gets us closer. All things have an identity, a set of attributes that makes a thing uniquely determined. All things contain properties, that is they have content. And, All things function within a context, in that it takes place in a setting, that is to say it is intended to be used for something.

3. THE SEMANTIC MODEL

Now we have to think differently about how to do what we are trying to do. We no longer have to abstract the hidden structures of a use case, which inevitably changes. We can think of things as “things” because all things have an identity, content, and a context. Therefore we do not need to model them they can be represented just as they are. We have a basic container into which we can put things and that leads us to pay attention to answering the more important question of how are things related.

se·man·tic

/sə'man(t)iks/a set of rules whereby something is put together or an analysis of this.

/When we use the word Semantic here, our context is the property of intelligibleness. Semantics is about a uniform way to model relationships./

What we mean by Meaning

When most people think semantics they think meaning. While this is an endlessly debated philosophical rhetoric, suffice it to say that, for the purposes of this book, we take the Quantumistic (philosophy of Quantumism) position. There is no such thing in the universe as meaning. Meaning exist only in the human (for now) cognition. Meaning is something consciousness “adds” to perception. Therefore, in an information science sense, there is no “meaning” to be derived from information. Information is bits of symbols that can be perceived, organized in a specific structure, and put through a process. Data is the representation of information. Data is what is stored. Information is what you see on the screen. Meaning is what you add to the information in order to take away what it tells you about the relationship between one thing and another. It is important to understand this philosophical difference. Consequentially, Semantic storage does not mean storing meanings or meaningful storage. It means possessing a way to model

the relationships between things, having a method to store those relationships, and that those relationships are stored separately from the thing itself. Quantumism is a philosophy that posits that things have no inherent meaning and as such no process can extract the meaning of something. The meaning is an emergent property of the model used to represent information. Therefore when we speak of semantics we are referring to, at the most fundamental level, the method for building and defining relations. A relation is an attribute that holds between objects, and as such usually describes an attribute that is an [element of] or [example of] another object. Therefore the relationship between things are itself an ontologically distinct entity. What we need from a computing system is not to store meaning but rather we need a way to represent the relation, then store and retrieve this relation separately from the thing itself.

Subject - the basic container

The fundamental unit of our system is not a file, it is an organization of data called a Subject. In computing systems you think of files and types of files. On the web you think of mime types. In Onli Os (CloudMoDe) there is only one class of file, so to speak, and that is a Subject.

A Subject is a Component and therefore is organized into identity, content and context. Subjects may contain files called a Primitive (a machine-encoded set of bits), or it may be “empty”, where the content is created after the fact by a function or operation and “filled”.

A Subject, which is the most basic and fundamental component replaces the concept of files. This Ternary System (identity, content, context) gives us a unified semantic ternary model of data. This elegantly simple organization yields a robust enough structure into which we can model any “real world” object.

Now that we have a means of containment we must define a method for modeling relationships and storing it as separate from the thing itself. Relationships fits within our syntax as it possesses an identity, content, and a context.

Objects

In information theory, the fundamental reason for storing something is to create and define a relationship. The most common relationship is that of representation. We therefore add an addressable layer of abstraction onto Subjects that can model this, which we call an Object.

An Object is a defined relationship. We create Objects by first requesting one from the DataMode process, which is a set of instructions for creating components and storing them.

First we define the Object’s identity (who am i). Next we define the content. The most common relationship is one of representation. To accomplish this we define the representation (the Subject used for depiction) and the derivative (the Subject the depiction is meant

to represent), and optionally a description of the Contents. Finally we define the Context (what am I used for). The context of a subject needs to be a flexible set for relationships we had not yet defined. We will therefore make the conteXT set extendable to accommodate future and as yet undefined uses of this object.

Stacks

You can also create relationships between groups of Objects. This is called a Stack. We create a Stack, much in the same manner as we do an Object, by first designating a representation (depiction) and defining the derivative (which in this case is a list of Objects), then, as this is a Component, we define it's identity (who am i), content (description of what it contains, depiction & derivative (the derivative is, in this case, a List of Objects), and context (what am I used for). Context must be flexible to accommodate future uses, it is therefore extendable.

Collections

We can create a relationship between Stacks, which we will call a Collection. We create a Collection, much in the same manner as we do a Stack, by first designating a representation (depiction) and defining the derivative (a list of Stacks), then define it's identity (who am i), content (description, depiction & derivative- List of Objects), and context (what am I used for). We will make the conteXT set of Elements extendable to accommodate future and as yet undefined uses.

uSer & aGent

Subjects, Objects, Stacks and Collections in a computing device are created by users. We will therefore need to create a special kind of stack called a uSer. The uSer will be stored and therefore must be expressed in our model's syntax; identity, content and context. A uSer is a special class of Object, stored in the UserMoDe service that manages users, and is defined as an owner of a set of Resources. A non-human actor is called an aGent.

Catalog

The computing system itself also needs to operate Resources. We differentiate this from uSer created Resources. We call this a Catalog. A Catalog is defined as resources used by a set of operations and/or functions. A Catalog will be stored in the the data store and is therefore a Component, which means it is expressed in our model's syntax; identiTY, conteNT and conteXT. A Catalog is a resource (like subject, object, etc) where the uSer is not a "person".

General Classification

In order to be robust our semantics must differentiate between a task and the target of a task. There are three classifications of "things" in Onli Os (CloudMoDe) 's semantics.

* Resources -where you store data.

Resources are addressable containers where data is stored. There are six classes of Resources: Subjects, Objects, Stacks, Collections, uSers,

aGents and Catalogs.

* Operations -a set of instructions for the computing system for doing things with resources.

* Functions -a set of instructions for the computing system.

Appliances & Trays

In a client-server architecture the client is a piece of software that requests a service or resource from another program (like a server). Cloud Computing refers to the sharing of resources to achieve coherence, similar to a utility (like the electricity grid) over a network. An Appliance is something that plugs into a grid. In this case a grid of data, compute and memory resources. The client side is called an Appliance and the server side is called a Tray. You take data put them into cloud mode (make them intelligible) they are then resources. A operation then puts them on a Tray. Functions may be performed on them. Another operation then streams them to an Appliance.

Benefits

By using a unified semantic model of the physical and logical schema we can achieve a significant conservation of efficiencies, where conservation means /to hold constant during an interaction or process)/and efficiencies / mean the ratio of work/effort to the energy/ resources supplied to it/.
Requirements gathering and analysis in data modeling are reduced to identifying where attributes should be placed in the model, instead of creating a new structure of attributes and relationships with each application. The implementation phase (data storage, logical-physical mapping and translation) becomes a simple coding exercise, placing the attributes in the appropriate location in the model. Testing and verification of the data model prior to integration testing is greatly simplified also, requiring only the verification of data using existing (and proven) test software. Consistent semantics also reduces the entropy inherent in data storage systems that result from competing taxonomies.

A unified semantic model, in an of itself, has a tremendous impact on the value of a data model.

4. ADDRESSIBILITY

addressibility

/ə-'dre-sə-bəlthe capacity for a thing to be targeted and reached./

/In order for data to be intelligible it must possess the quality of being perspicuous. Perspicuity [pur-spi-kyoo-i-tee] is a word that denotes the measure of clearness resulting from inspection. We interpret it in the context of data to denote the capacity of being reached, as any measure of clearness resulting from inspection can only come from the capacity to be reached. We can therefore, taking some artistic license, translate perspicuity as the quality of being addressable./

The reason you store data is to query it, make a selection and retrieve it. In order to retrieve something stored it must be addressable. In order to be addressable it must be uniquely identifiable and there must be a method of addressing it, a language for querying.

addressability

In Onli Os (CloudMoDe) each component is uniquely identifiable with the standard GUID method (part of it's identity). The robustness of our models makes it possible to do more than address an entity but we can also directly, in a consistent manner, reach the entire spectrum of the syntactic model; from a single attribute to a single element, from a set of elements to an entire component. This contributes directly to a conservation /(to hold constant during an interaction or process)/of efficiencies /(the ratio of work/effort to the energy/resources supplied to it)/. This is where, you can reach what you want more efficiently and learn from your interactions enough to get the same result while using less resources. The secret to this efficiency comes from having a simplified semantic model and results from the very nature of the models used to store and structure data.

Semantic not hierarchical

A hierarchical file system requires users to interact with a hierarchical classification of their files or data. In this sense, the user of a hierarchical file system is confronted with creating and maintaining a consequent taxonomy (vocabulary and naming convention). The context of the user constantly changes and so does the taxonomy. It becomes harder to remember where to look for files in the hierarchy. The hierarchical model impairs the capacity to reach what you want. This is more than simply a user problem. Computer processes require the ability to reliably access a piece of data or file. Consequently should it require such from an external process, it must communicate with said system. In a hierarchical system it must know "where" that file is, and since each system has it's own taxonomy, the only way to guarantee that it knows where the file is, is to make a copy of that file. This is an inescapable consequence. A semantic approach, knowing what to look for, and searching for it, leads to a better user interface, in that way it works the way the human brain does. When you download something from the web, you often can't remember where you put it. Most people find it easier to find something on the Internet - through Google - than they do on their hard drives. The semantic approach to storage and retrieval can revolutionize user interfaces but has an even more profound impact on addressability between computing systems. If a computing system "knows" where something is stored and can reliably predict, through standardized taxonomy, that it will always be in the same place, then it needn't make a copy to guarantee quality of service.

Impact on search

The depth of granularity inherent in our models also makes search more elastic. Granularity is the level of depth within the data structure. Elasticity is defined as the degree to which a system is able to autonomically adapt to changes. Elasticity, within a system, is the key ingredient for the conservation of efficiencies. E.g, recently accessed searches by a user are more likely to require additional searches within the same set therefore it is more efficient to retrieve an entire set of elements. However the opposite is true if the request originates from a computing process, which almost always only require the value of a specific Element. Therefore if you make the simple observation of distinguishing who is making the request, a user or a computing process, you can "conserve", that is use less resources, in this case make less round trips to the storage mechanism. Increasing the level of granularity leads to elasticity. In other words, you can ask for, or address, what you want more precisely.

Technology Choices

Addressability in computing goes beyond simply having an address to a piece of data. It is a concept that extends out to the technology choices of the enterprise. When you choose a

database, you are not choosing one piece of technology but three : storage technology, data model, and query language.
E.g., if you choose MySql, you have a choice of storage engines, the two most common are MyISAM and InnoDB, a relational data model, and the SQL query language. If you choose MongoDB you are choosing the MongoDB distributed storage engine, a document data model, and the MongoDB API. These are three distinct layers of technology. In most systems they are tightly coupled and, to varying degrees, are inseparable.

Not all storage technologies and data models make sense in every context. Often different parts of a single application call for different choices in each layer. Graph databases, document databases, column-oriented, row-oriented, etc., each have different requirements. This creates a dilemma: either select new databases for each need, which is often incompatible technology to support the required variety of data model, or try to crowbar all the data into a single database. Probably nothing has more of an impact on the degree to which data is addressable than technology choices.

Implementation

Our core philosophy requires that we separate storage from the data model and both from the query language. Since our structured object model makes a core departure from conventional data models, we use a custom domain specific language called a Natural API. This enables us to realize a significant gain in operational efficiencies. Remember, the entire point of storing data is to query and retrieve it. By separating “how and what” you do *with* data from “where and the manner” the data is stored, you can then choose the best technology for the right job.

Onli Os (CloudMoDe) is built with a distributed storage engine - a database in which storage devices are not all attached to a common processing unit such as the CPU, and controlled by a distributed database management system. Since the storage technology is separate then it may be stored in multiple computers, located in the same physical location; or may be dispersed over a network of interconnected computers.

The separation of the core layers; storage technology, data model, and query language, in addition to implementing this in a distributed storage mechanism, makes it possible to realize advances in efficiency, integrity, security and robustness.

5. ACCESSIBILITY

ac·ces·si·ble

/ək sesəb(ə)l/the capacity for a thing to be used or obtained.

/Simply because you can reach something, that is it is addressable, doesn't necessarily imply that you can obtain it or use it. This is especially true in computing. Accessibility covers a myriad of complexities that include authentication, authorization, accounting and we add one more classification use; control. /

Onli Os (CloudMoDe) is a semantic storage system and the storage technology is a distributed storage engine, this makes it possible to achieve conservations of efficiencies in the levels of accessibility that are simply not practical with hierarchal file systems. Probably the most important of which is that of instantiating a control system instead of a security system.

Structural Control

Hierarchal file systems (HFS), at its most fundamental level, uses an access security model based upon an access control list (ACL).

> */An access control list (ACL) is a table that tells a computer operating system which access rights each user has to a particular system object, such as a file directory or individual file. Each object has a security attribute that identifies its access control list. The list has an entry for each system user with access privileges. The most common privileges include the ability to read a file (or all the files in a directory), to write to the file or files, and to execute the file (if it is an executable file, or program). - Margaret Rouse whatis.com/*

When a request is made for an operation on a file, the operating system first checks the ACL for an applicable entry to decide whether the requested operation is authorized. A key issue in the definition of any ACL-based security model is determining how access control lists are edited, namely which users and processes are granted ACL-modification access. This security model

offers a moderate level of security so long as the file remains within the operating system. Once the file is outside the system the security model breaks. As already intimated in an HFS system, in order for computing processes to reliably access a file, it must make a copy of it, and as such the ACL security model is practically useless in a distributed environment.

Onli Os (CloudMoDe) does not have files in that sense therefore it abandons the ACL model. Onli Os (CloudMoDe) has a contextual structural controls mechanism. This contextual, capability-based security model, where the capability cannot be copied.

/A capability (known in some systems as a key) is a communicable, unforgeable token of authority. It refers to a value that references an object along with an associated set of access rights.- Wikipedia/

At the center of this model is the *usePolicy,* an organism /*organ*izations of *I*ntelligible *s*emantic *m*edia" as *organism*s or ism for short./ containing an explicit course of permitted actions for a referenced organism, and a set of instructions for a computing system that monitors and enforces the usePolicy /(see chapter on Uniqueness)./ The usePolicy is enforced by aGents and/or the Appliance (display client) . The usePolicy is attached to an instance of use rather than a file. The usePolicy controls use of a relationship not the actual resource, in the classic sense, and it can include binding clauses, which can be used as an additional layer of controls. aGentPolicy is a usePolicy used to store rules of how to apply the related objects. It is a component and has Identity, Content and Context and is therefore separately stored and is addressable. The term agentPolicy holds philosophically as it is a set of rules regarding the use of objects by aGents in a context, linguistically as a policy is a rule to be followed by agents and actually reinforces the structure of Onli OS in regards to the relationship between trays and agents; agents assemble data and trays serve the output of the assembled data. What you get therefore is not a concept of files and access control lists but an entirely different concept that we call Structural Controls.

This is a significant evolution in computing because it means that along with features such as encapsulation-(packing data in a modular structure into a single component) and unforgeable references , we now have specialized capabilities for every operation. A capability is an explicit course of action. Encapsulation means you can reveal elements of the identity of a thing in the context of one request or reveal only the elements of the content in another request. Unforgeable references means only a valid user can make a request and the request only reveals the

relationship without revealing the thing itself, thus it resists being forged by an authorized user. Capabilities are not transferable and only exist in a context thus they are also unforgeable.

This means you can reveal elements of the identity of a thing in the context of one request or reveal only the elements of the content in another request, and reveal the relationship between other things without revealing the thing itself. All this is already part of the system without writing a single line of code.

Structural Controls gives the data additional capabilities that translate into features such as true ephemerality (temporary existence of the bytes) and a robust set of remote controls are standard throughout Onli Os (CloudMoDe) rather than being brought about by exotic programatic constructs, which tend to be fraught with error. Consequently as this is already part of the system it is inherited throughout without writing a single line of code.

Security

Control emerges from communication in a context rather than being a table attached to a file. The difference is fundamentally about the difference between control and security. You can have security without control but once you have control you reduce the needs for security. Where control is defined as the ability to determine which specific set of actions a person gets to perform. And where security is defined as protection from destructive forces, and from the unwanted actions of unauthorized users.
Onli Os (CloudMoDe) separates the storage technology from other layers of computing, the system is able to achieve a far superior level of security through structural controls.

Security comes in three distinct but necessary and sequential processes called the AAA framework. Authentication, authorization, and accounting (AAA) is a framework for controlling access to resources, enforcing policies and auditing usage.

Authentication

The first process is Authentication.
Authentication is a method of identifying a user. Onli Os (CloudMoDe) includes a quantification method for uniqueness /(see chapter on uniqueness)/ that insures that only the user (which is unique) can make a request for authorization. This uniqueness is validated before the request for authorization is even considered. This adds a layer of security that renders the user object all but unforgeable.

In Onli Os (CloudMoDe) , the authenticating step is a two-cycle block-chain transaction that takes place out-of-band. In a two-cycle authenticate process, One cycle is used to perform confidentiality computations, and the second cycle is used to compute authenticity and integrity.

The confidentiality computation uses a Knowledge Factor - something the user knows to initiate the connection. The user enters a key, like a phrase, word or number. The hash value of this word plus the hash value of the last transaction is sent Out-of-band. Out-of-band means a communication that takes place over a network or channel separate from the primary network or channel. This communication is checked by the Validation Engine. This validates that the request for authentication is coming from a valid source.

The second cycle computes the authenticity of the user making the request by comparing a chain of blocks. A block consists of the hash value stored in the identity component of the user, it is something only the user would possess (ownership factor- something the user has). A typical transaction is a record of where the user logged in from, a token from the authenticator (an out-of-band process), and a token for the current use. Following authentication, a user must gain authorization for doing certain tasks.

Authorization

Following authentication, a user must gain authorization for doing certain tasks. In Onli Os (CloudMoDe) authorization is contextual and attached to a request. The ability to determine which specific set of actions a person gets to perform is recorded by the owner of the requested resource, in the UsePolicy, it is not in the system. Therefore authorization, emerges in a specific and defined request, is resource specific, may be dynamic and live or static and predetermined.

Accounting

Accounting is an inherent part of the system as every transaction is recorded as part of the identity of the user, or part of the identity of the resource requested. In addition to this there is a logging of session statistics and usage information, which is stored in the Validation Engine. The significance of this is that accounting processes covers the Inherence factor- something the user does, and this becomes part of the inherence factor - something the user has.

Mylar Architecture

In secure functions such as authentication, the system uses the MIT Mylar server architecture, where encryption and decryption all takes place in the client and not on the server. The traditional approach to securing user data on servers, is to accept data from a user then encrypt it before saving to a hard drive on a server. When the user requests the data, the server opens the file, decrypts it and then sends what has been requested. Any agent that gains control over a server, can get user names and passwords, and decrypt everything on it. In the MIT Mylar Model, even if the server was breached, the agent can't read the data files, as the only key that can decrypt it resides on the client (with the user).

The benefit of this architecture is that usernames and passwords are not stored on dataMoDe (server that streams Subjects) or objectMoDe servers (servers that streams Objects, Stacks, Collections, Catalogs). In the unlikely event that either an authentication service or a data storage server is breached there is no way to compromise the system. Accessibility in Onli Os (CloudMoDe) is literally a dynamic function that only takes place in a context where the credentials evolve with every transaction.

Benefit

All three factors, Knowledge (something the user knows), Possession (something the user has) and the user's history, which is a novel way to improve the Inherence factor - (something the user does), is all expressed in the authentication, two-cycle, process. The result of the two-cycle process and the architecture is a completely secure system. All without the heroic efforts required to make existing systems secure.

The difference between control and security
Onli Os (CloudMoDe) achieves a level of accessibility that leverage tremendous benefits from the technology. At its most fundamental level, the difference that makes the difference, is that Onli Os (CloudMoDe) has a "control" system, which in most cases greatly reduces the efforts required of the "security" system. Control emerges from a context and that means it is more than just the ability to discover and reach a thing, or access control. Think of security as something you put on the outside and control is a property a thing possesses. You can have security without control but once you have control, security takes on a different meaning.

6. MODE AND ISM

mode

/mōd/the state something is in

ism

\'i-zəm\: a distinctive doctrine, cause, or theory

/At Onli Os (CloudMoDe) we put data through our "manufacturing process", which makes data intelligible. We accomplish this by moving data through different modes (states or organizations): dataMoDe, objectMoDe, and Onli Os (CloudMoDe) ./

So far we learned that intelligibleness means having a specific and defined set of properties:

*** ### Consistent Syntax (Identity, Content, Context).**

*** ### Simplified Semantics (Subject, Object, Stack, Collection, Catalog, uSer, agentUser).**

*** ### Addressability (with a high level of granularity and elasticity).**

*** ### Accessibility -(authentication, authorization and accounting that include knowledge, possession and inherence factors).**

We showed how we apply this insight in the field of computing science. Our invention is a structured object model that expresses the properties that make something intelligible. We refer to these **"*organ***izations of ***I***ntelligible ***s***emantic ***m***edia" as ****organism*s*** or *ism* for short. The end result is a whole new way to store data, access it, move it around and keep it safe.

Modes

The nature of Organisms, their intelligibleness, require that you think about data and computing differently. You want to think of data as having states or "modes" at different stages in the process. We "make" data intelligible. We accomplish this by moving data through different modes (states or organizations): dataMoDe, objectMoDe, and Onli Os (CloudMoDe) . There is the storing step, the relation step, and the defining step.

The storing step dataMoDe

The system first generates a structured container that is addressable. It then takes the raw (primitive) file and make an entry in a specific set of Entity-Attribute-Value entries or Elements, which are organized into three distinct sets of Elements; identity, content, context. In this manner we create a single file class, called Subjects. When data is in this step it is in "data" mode. In this manner we create a single file class called Subjects.

The relation step : objectMoDe

The next step is the user then defines the relationship between this Subject and another Subject. The semantics are optimized for the most fundamental of relationships in computing, that of representation. The relationship itself is separately addressable. This gives the user a rich enough set of semantics to model complex relationships, called Objects. When data is in this step it is in "object" mode.

The defining step : cloudMoDe

The next step in process is to define accessibility. Resources (Subjects, Objects, Stacks, Collections, Catalogs and uSers) are all addressable at every layer. This gives it a high degree of granularity, which increases elasticity. Elasticity is the degree to which it can adapt. Granularity is the level of depth within the data structure.

The control step : usePolicy

n addition to this Onli Os (CloudMoDe) institutes a true contextual Structural Controls model, where the capabilities cannot be copied. A UsePolicy is an “Evolving” (see chapter on uniqueness) Organism that contains an explicit course of permitted actions. The explicit course of permitted actions is called terms. There is also a method for defining a relation of mutual dependence or action when processing a request for a referenced organism, as a binding clause, on the explicit course of permitted actions. These are expressed as a set of conditional statements (if-then-else-goto), called Conditions. Conditions permit the variance of Terms according to the circumstances in which a request exists or occurs. Different Use Policies can reference the same organism but authorize different sets of actions, thus producing different Terms depending on the context.

When data is in this step it is in “cloud” mode. Organisms in “cloud” mode have Terms and Conditions, which are attached to a request not a file.

What does all this mean?

Intelligible data is an ontologically distinct entity. It is not just putting a file in a folder. You are building something or packaging data, in such a way that you are going to make it useable in a specific way and/or in multiple ways. It is a specific and defined organization of data that has the potential to power entirely new models of computing.

THESIS SUMMARY

Onli Os has a consistent syntax, a uniform taxonomy :

*** Element**

The fundamental building block of Onli Os (CloudMoDe) is called an Element. An element has an entity (the class), an attribute (the naming part) and a value (the expression).

*** Component**

Everything that is stored has a identiTY, conteNT, and a conteXT. This means that you always know what something is, where it is, how to query it, and where to look. This is the DNA of the system.

The semantics, type of things, in the system are

*** Subjects**

The most basic and fundamental component that replaces the concept of files. The Primitive could be present at the time of formation or “filled” after the formation of a subject.

*** Objects**

A Object is a defined relationship between Subjects. It is a Component and has identiTY, conteNT, and a conteXT. and is therefore separately stored and is addressable.

*** Stack**

A Stack is a defined relationship between Objects. It is a Component and has identiTY, conteNT, and a conteXT. and is therefore separately stored and is addressable.

*** Collection**

A Collection is a defined relationship between Stacks. It is a Component and has identiTY, conteNT, and a conteXT. and is therefore separately stored and is addressable.

*** uSer**

The human actor of the system. Subjects, Objects, Stacks, and Collections are owned by a uSer. It is a Component and has identiTY, conteNT, and a conteXT. and is therefore separately stored and is addressable.

*** aGent**

The machine actor of the system. Catalogs are owned by an aGent. It is a Component and has identiTY, conteNT, and a conteXT. and is therefore separately stored and is addressable.

*** Catalog**

A Catalog is defined as resources used by a set of operations and/or functions. Catalogs are Subjects, Objects, Stacks, and Collections, are owned by an aGent. It is a Component and has identiTY, conteNT, and a conteXT. and is therefore separately stored and is addressable.

*** Appliance & Trays**

The Client usually runs on another device and requests to use resources of the server. The client side is called an Appliance and the server side is called a Tray.

*** usePolicy**
A UsePolicy is a special object used to store the configuration of the capability, in Onli Os (CloudMoDe) 's Structural Controls system. It is a Component and has identiTY, conteNT, and a conteXT. and is therefore separately stored and is addressable.
*** agentPolicy**
A agentPolicy is a special object used to store the configuration of the how to use an object accessed by an agent. a special object used to store rules of how to apply the related objects, in Onli Os (CloudMoDe) 's Structural Controls system. It is a Component and has identiTY, conteNT, and a conteXT. and is therefore separately stored and is addressable.

A simplified syntax and a uniform semantics led us to make great leaps in addressability and accessibility. Most notably being able to instantiate an Structural Controls mechanism.

Different from the ground up
Onli Os (CloudMoDe) is based on a kind of Structured Object Data Model, a ternary (triplet) model, which is robust enough to model all the fundamental components of data.

What we learned in building it is that organization is really the first and most important step in computing.

We find that this premise is valid in the context of data.

Shift into Cloud MoDe (Onli Os
Onli Os (CloudMoDe) is something you shift your data into. By doing so you make data intelligible. Adding intelligibleness is a specific and defined process.

*** Organize** (using a model robust enough to use across conceptual logical & physical models)
* Store data using a uniform model of data -(distributed across machines) -/Syntax/
*** Define relations** - use representations to define relationships that can be stored -/ Semantics/
*** Optimize,** which increases efficacy in the way you get data in and get data out -/Addressability (API)
*** Secure ,** define how the data is to be used -/Accessibility (Authentication & Control)/

Once data has been through this manufacturing or evolution process we call it *Intelligible Data*. By adding intelligibleness we were able to forge an evolution of data, where evolution is defined as a natural change in the state of entropy in a system.

By making data intelligible, we now have a whole new way to store data, access it, move it around and keep it safe.

#onlios# #structural_controls

STRUCTURAL CONTROLS

At the most fundamental level, the web, is a network of documents, stored in a hierarchy, on a network of computers, the internet. The internet and the web are two very different things. That is a very important distinction to understand. How the web works, is that you create a document on a device and send it to another device, by making a copy of the document. The only reliable way to store and recall a document on a network of documents stored in a hierarchal file system on a network of computers, is to make a copy. This leads to a uniqueness quantification problem. How do you store and recall something that can be quantified as unique accross the network.

1 UNIQUENESS

uniqueness quantification

/In mathematics and logic , the term “uniqueness” refers to the property of being the one and only object satisfying a certain condition. This sort of quantification is known as *uniqueness quantification* or *unique existential

Onli Os (CloudMoDe) OS employs the use of an artifice called a UsePolicy that regulates the delivery and use of an organism (*organ*ization of *I*ntelligible *s*emantic *m*edia.). We call it Structural Controls and it allows the user to determine the explicit course of permitted actions for a referenced organism in a context.

Computer networks are designed to transmit packets of bytes between the computers on the network. Bytes are assembled, transmitted and replicated between computers. Once a collection of bytes has been transmitted from one computer to another, the original collection of bytes is no longer unique. There is now a copy and it must be indistinguishable from the original. This is called replication. This replicated copy must be stored and retrieved upon request. Replication is a fundamental capacity of a computing system. Replication ensures consistency, even between redundant resources. Access to a replicated resource when transparent to the external user evokes reliability. There must be no discernible difference between a replicated and non-replicated resource, thus in a failure scenario, the quality of service is not impacted. Replication ensures fault tolerance and in so doing avoids a single point of failure. Computers are by their very nature replication machines.

There is an added complexity when files are stored in a hierarchal file system. A hierarchy, which lacks consistency, a uniform taxonomy, a stable location and defined rank would be unreliable. The only way to make this kind of storage reliable is replication. By making a copy, a replicated resource can be reliably accessed.

Introducing the property of uniqueness would create a “hard” problem. Uniqueness, as we use it here, is more than a measure of distinction but rather it is a property of an object that can be quantified without further intervention. The problem of uniqueness or uniqueness quantification is where one seeks to persist an object in a computing environment used for communication, of which, there can only be one.

Implicit in communication is a requirement of precise replication. The capability the user requires from a communication system, is to send a message via a channel and for that message to be accurately reproduced. Replication is natural to communication. The problem is not replication itself but rather the lack of a method to quantify uniqueness.

The problem of uniqueness arises whenever we desire to model the natural world where there is a permanency to existence. Even two identical objects occupy different positions in space and time therefore their uniqueness can be quantified. This property of the natural world is something we take for granted. Quantifying that property in a representation system is therefore a hard data science problem in the computing environment.

Uniqueness quantification is a property that we desire to have in our system. Uniqueness is a property that, should the user desire it, be available as a mechanism one can employ, to

bestow upon an organization of intelligible semantic media (an organism) at a given point in time as to facilitate such a function, and this property should persist without further intervention.. There are many use cases for this. The most prevalent is to maintain the integrity of said organism. This kind of uniqueness will need to survive transmission, meaning that after transmission, the uniqueness of the original is retained.

Simply put, a distinctive requirement of Onli OS is to make a “thing” that is unique in the computing environment. Since we want this to be part of a communication system, necessary to our definition is that its uniqueness should survive transmission. Another vector of our requirements is that uniqueness should persist over time, without further intervention. In other words we are trying to make it so there is one and only one of a thing, which is a very useful property to endow upon an information organism.

Onli Os (CloudMoDe) solves the Problem of Uniqueness by implementing a novel system of methods and controls. Onli Os (CloudMoDe) instantiates a method of communication, where any change in the state of an organism (an organization of data) results in a transaction (an act, process, or instance of action) and such a transaction is immutably recorded as part of the identity of the organism. This capability enables us to build unique features into our data-store. It is therefore a good idea to have a working understanding of the concepts behind the Onli Os (CloudMoDe) Uniqueness Quantification Algorithm/

Solving the Problem of Uniqueness

The Philosophy of Quantumism posits that all communication has a target: one making a request (the requestor) and the target of the request (the receiver), therefore there are always two parties to a transaction. The act of communication, employing the Onli Os (CloudMoDe) Uniqueness Quantification Method, will result in a change in the state of the transmitted organism, which we call a transaction. A transaction is a consequence of a request. The Onli Os (CloudMoDe) Uniqueness Quantification Method, as a definable rule, also requires that every transaction generates a record and such a record is documented and immutably stored in the identity of the parties of the said transaction. This is easily facilitated, system-wide, as all organisms in Onli Os (CloudMoDe) have a consistent syntax or structure, that of an addressable store for identity, content and context. Transactions can therefore reside as a permanent property of the identity of the organisms. The Onli Os (CloudMoDe) method also institutes a method of recording transactions that uniquely identify substantial blocks of data and assemble them into a chain of blocks, which includes the previous state in the next block in the chain. This chain of blocks can be used to represent

the history of transactions. This gives us a method to capture the interactions of the organism.

There are two classifications of interactions. The first is an interaction that results in a 'copy' of the organism, and as such, the copying of an organism constitutes a change in the state of the organism and therefore generates a record, which is entered into that organisms identity. Another way to say this is that when an organism is copied, a transaction is entered into the organisms transaction history, uniquely identifying the organism that was copied and the organism that results from the copy transaction.

The second classification of a transaction is an interaction that results in the 'transfer' of the organism, where transfer is defined as a change in ownership of the organism, whereby the change in ownership results in a change in the state of the organism, and therefore a transaction.

Transfers can be accomplished in three different ways:

1) in-situ: where the owner property is changed in the organism itself,

2) by copy-delete: where a copy transaction is executed, the owner of the new copy is changed, and the original is deleted,

3) copy-extinct: where the original is flagged as extinct or invalid.

The Onli Os (CloudMoDe) Uniqueness Quantification Algorithm gives us a way to endow an organism with the property of uniqueness that: 1) maintains the integrity of the original 2) survives transmission and 3) maintains uniqueness over time. This is all done at the Data-Layer without requiring elaborate heroic efforts to implement, and thus it is available to any aGent acting on the organism, or appliance (client) retrieving said organism.

The Problem of Validation

Next we have a validation problem i.e. how to verify that each organism involved in the transaction is unique and a party to the transaction, where the transaction is a record of the changes in the state of the organism.

Upon being presented with the identity claims of an organism the system can use any variety of methods to compare transaction history, thus validating the uniqueness of the organism:

The trusted arbiter method has the properties of being centralized in that a single process on a single computer can be used to make the comparison.

Validation is the process by which the system will authenticate the claim of identity or uniqueness by an organism. Validation is critical to several functions.

1) Authentication, confirmation of stated identity
2) Authorization, allowing the performance of a set of actions
3) Accounting, to make an accounting for events.

Validation may be implemented in several different methods. Validation is a method that compares data in different systems to ensure it is consistent.

Use Policy Terms & Conditions
The Onli Os (CloudMoDe) Uniqueness Quantification Algorithm lets us :

1) create unique organisms (***organ***izations of ***l***ntelligible, ***s***emantic, ***m***edia).
2) Maintain the integrity of uniqueness by employing a method to validate the identity claims of organisms and transactions involving the organism.

Such uniqueness can now survive the transmission of the organism.

Given that we have the system that can verify and maintain the uniqueness of the organism as described above, we employ this to expand and impose our Structural Controls mechanism.

First we will designate this kind of organism where transaction history is recorded in the core organization of data: an Evolving Organism, as opposed to organisms that are Nescient organisms.

Only Objects (a model of relationships) can be “evolving’, as herein defined. Subjects are by their very nature, as a container, Nescient. It is the relationship “between” things that is most often the target of uniqueness. Eg: “My cigar” is a statement not about the uniqueness of the cigar, or, of the entity referred to as “my”. Rather it is a claim about the unique relationship between the entity referred to as the object “me” and the “cigar”.

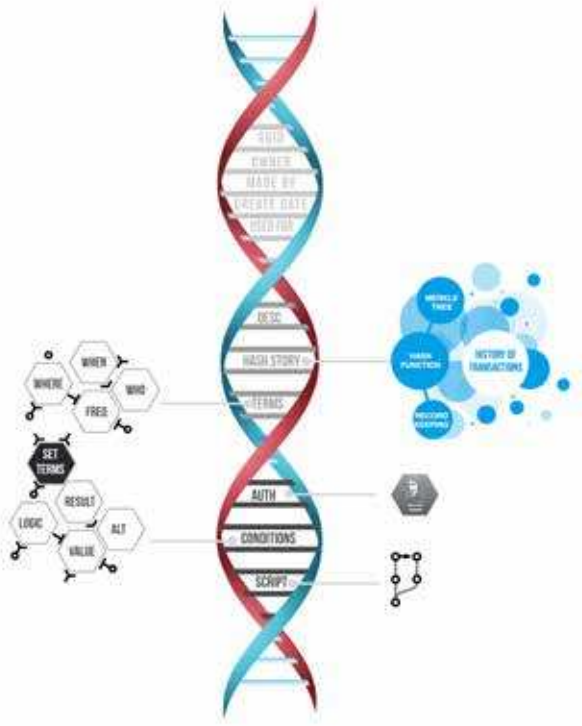
A variation of an Evolving Organism that possesses additional complexity, for the purpose of containing an explicit course of permitted actions, is something we call, a UsePolicy.

A UsePolicy is an Evolving Organism that contains an explicit course of permitted actions, that when executed, result in a transaction. We call these permitted actions for a referenced organism Terms. Such transactions are recorded in the transaction history of the participating organisms.

Use Policies operate in a context. Part of that context is a set of conditional expressions (if-then-else), called Conditions. Conditions are a binding clause and extend the functionality of the Terms, by identifying specific conditions

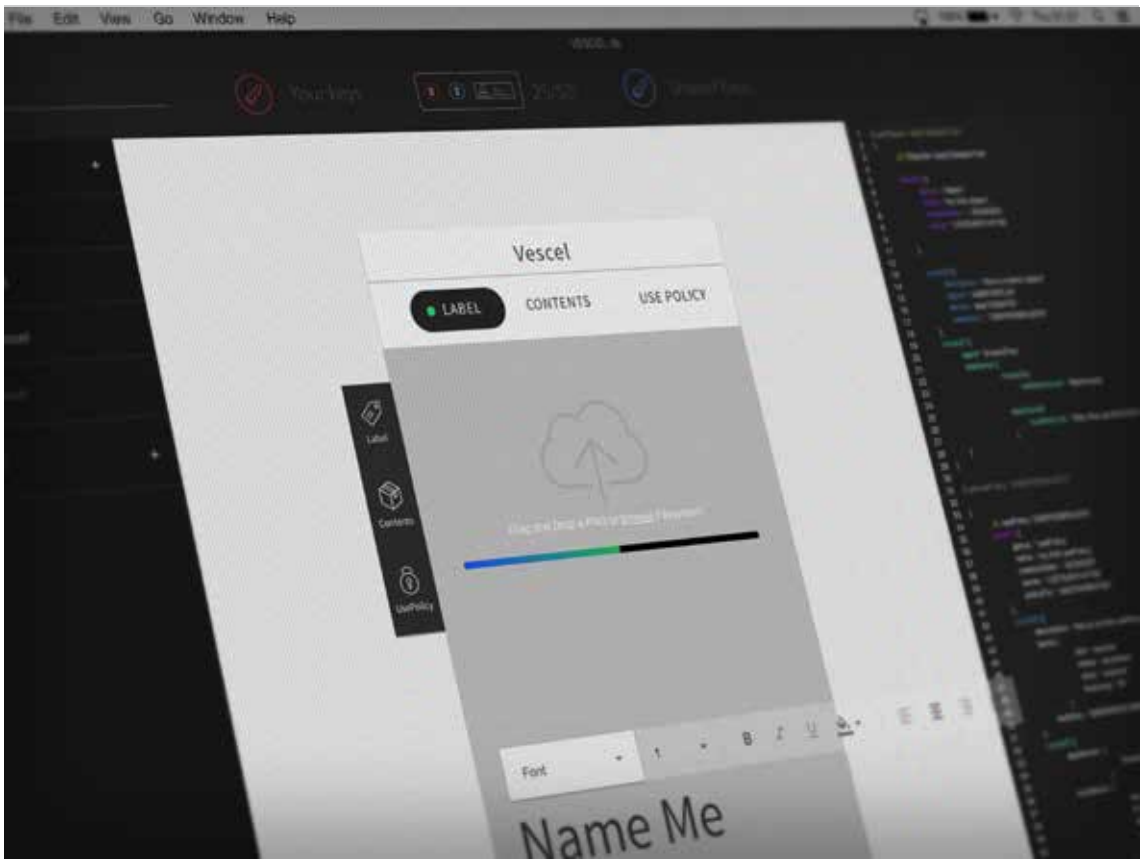
that must be met in order for the resulting transaction to be executed. Conditions are domain specific protocols that enforce the performance of logical constraints.

Terms and Conditions of a Use Policy are also used to facilitate the use or transfer of ownership of the referenced organism. The usePolicy can therefore model the natural world and effect the exchange of any classification of a communication that denotes ownership. Examples of these are commodities, currencies, digital goods, assets or it may, by extension, be mapped to represent the disposition of physical goods and/or services.



The usePolicy is the mechanism by which Onli Os (CloudMoDe) institutes a contextual control mechanism called Structural Controls. Different usePolicies can reference the same organism but authorize different sets of actions, thus producing different Terms depending on the context.

Solving the problem of uniqueness quantification allows us to build pliable computing models that move beyond access control. It permits us to define structural controls, which is a set of instructions that state “if this happens then do that” . The “this” in that statement can be any quantified query of any other system. The “that” in the statement can be any change in the terms of use.



VESCEL a datastore with Use Policies >>>

You store data in the cloud for a reason. You want to access it, move it around, share it and communicate about it. stored Data isn't just data. Data is the object of our communication.

Before we do anything with data we first make it intelligible. We put raw data into dataMoDe, which adds intelligibleness (an organizational schema). We then have put the data into objectMoDe: defining the relation (component-set of related elements) we then define relationships (how components are connected). We then move the data into cloudMoDe: adding addressability and accessibility to objects. This gives us a robust enough model to model all occurrences of data. However you store data in a computing system for a reason. Often data is the object of communication. A thing that we send a message "about". This messaging also has to be modeled, which brings us to Social MoDe.

#onlios #socliamode

COMMMUNICATION MODEL

Communication is at the center of our eco-system due to the architecture being event-based. Therefore a communication model has a profound impact on the proficiencies and efficiencies and thus performance of the system.

Schema

The Philosophy of Quantumism teaches us two things about communication: every communication has a target, and communication occurs in a context, two things to keep in mind as we develop our communication model.

Communication models, like databases (whether the programmer is aware of it or not), are based on a model or schema. Here, a schema is a cognitive framework or concept that helps organize and interpret information. A schema is a way to define the structure of something at it's most basic level. There are many communication models and each has their own schema (basic structure). The communication model used in SocialMoDe is a reduced or simplified schema because it is only concerned with computing, rather than with communication in general. The SocialMoDe foundational schema for communication consists of a sender (requestor), a message, a medium of storage, a method of communication and a recipient.
An example of instantiating the schema (creating a complete communication) is as follows:

1. the sender
2. selects a method of communication
3. stores a message using the medium of storage, which is then
4. sent via a route to the recipient.

Unlike communication models of old (pre computing) where it was necessary for there to be a “connection” in order for communication to work, a modern computing communication model need not specify a connection but rather a medium of storage, as a connection is implied in order for the communication to occur. Just as we collapse the Shannon-Weaver concepts of encoder and decoder into the medium, the concept of connection is also unnecessary, as it is implicit in the communication itself. We also abandon the old concept of medium as all communication takes place on the same medium: that of digital data transmission. What is more pertinent to our model is the method of communication and means of storage.

Our schema reduces computing communication, in a context, to its basic components:

* requestor,
* recipient,
* message,
* means
* and method.

We use this schema to form the basis of a model. If our model is robust enough then, just as we experienced in database modeling, there are some tangible benefits that we can leverage.

We begin, much like we did with our database model, with the separation of concerns. The act of sending a message and the act of being notified are two separate concerns.

Our model therefore has three spheres

1. Mittō is the act of sending,
2. DataMode (which we already covered) is our medium of storage, and
3. Certiorari is act of being notified.

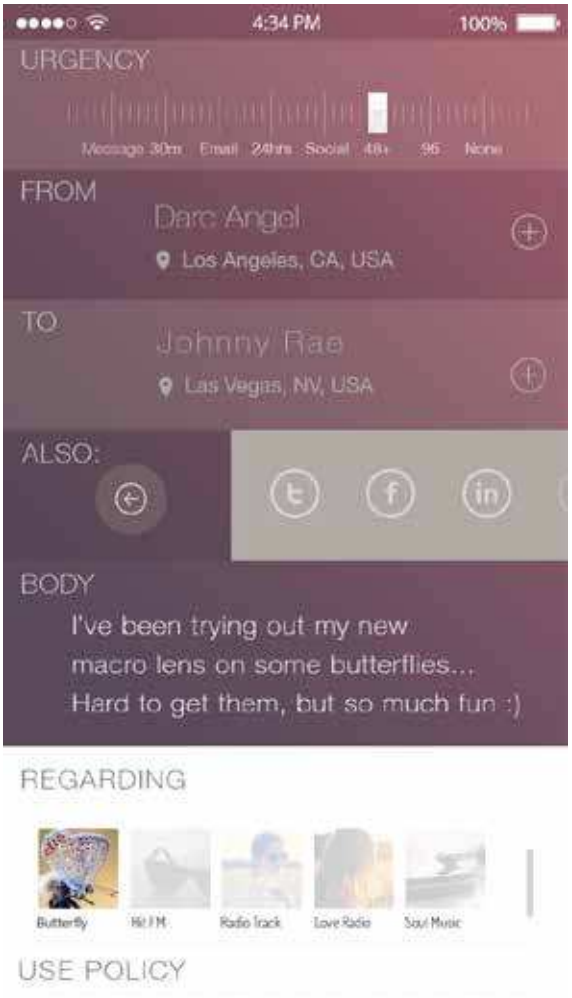
The Mittō Model

Mittō is a latin root of the word message, from the Latin mittere, missum (“to send”). Mittō, as we use it, is a term meaning the act of sending a message. The system is engineered such that an Agent (set of operations & functions) provides the capability of being able to send a communication to all Components: Subjects, Object, Stack, Collections, Catalogs and Users. All components can perform Mittō (the act of sending a message). In Onli OS any component (uSer, Subject, Object, Stack, Collection, Catalogs) can Mittō (Post) - send a message.

The core task associated with sending a message is routing, a function of addressability, answering the question of who is it going to, thus identifying the recipient.

In our communication model the method of communication is an emergent property of the organization of the message.

In other words WHO it is going to, (the identity problem), and WHEN I expect a reply, (the urgency, or Context problem) are what constitutes the method of communication. It is stored in the medium, which is DataMode and therefore every message has an identity, content and context. The underlying communication schema can have a profound impact on the UI as illustrated next.



Internet surveys have found that typically the expected response time for sending a text message to a client is about an hour, whereas when sent to a romantic partner the expectation is 5 minutes. The response time for an email is expected to be 48 hours. Whether I pick email, send a text, make a phone call, video chat or post to my social network, the most important vector in the selection of the method is the response time expected or urgency, which is a property of the communication conteXT.

For example, when you select a method (mode) of communication such as an email (response is expected with 48 hours), there is a sender (from), a recipient (To, Cc, Bcc), and a message (Subject & Body). When you send a text (response is expected within 10 minutes), the UI typically only confronts the user with an option to select a recipient but the same (sender, recipient, message) structure is there, only it is hidden. Even when the method of communication is a social app the same structure exists: there is the sender (you), a recipient (your stream) and message (text, photo, video etc). People who follow you are subscribing to your stream and receive an “update” or notification when you send a message to your stream. The distinguishing feature of the method of communication, is expressed at the UI level.

Since all Components can Mittō (send a message) this greatly simplifies our messaging system.

Post a Message Operation				
	From (**identiTY**) *Sender*	To (**conteNT**) *-Recipient*	Re (regarding) -(**conteXT**) *-Method*	About (**conteXT**) -
Text / Instant Message /Chat **1-1**	uSer A	uSer B, Array		String (140ch)
Email **1-1**	uSer A	uSer B / Array	object, stack, collection	String (140ch)
Social Message **1 - n**	uSer A	object, stack, collection	uSer A / Stack of Users (followers)	String (140ch)
Social Message **1 - n**	uSer A	object, stack, collection		String (140ch)
Stream Update **n - 1**	object, stack, collection	uSer A / Array	object, stack, collection	
Subscribe Update **n - 1**	object, stack, collection	uSer A / Array		String (140ch)
Comment **1 - Obj**	uSer A	object, stack, collection	object	String (140ch)
Mini Comment **1 - Obj**	uSer A	object, stack, collection		String (140ch)
Notification **Obj - 1**	object, stack, collection	uSer A		String (140ch)

As you can see from the Mittō chart we can now model 1-1 (one to one) methods of communication such as texting, email, instant message. We can model 1-N (one to many) methods of communication such as group messaging, multiple recipients, etc. Because Components can Mittō we can also model complex social methods such as (N -1) Many to One, which effect real-time updates (notifications) of social streams. We can model (Obj-1) Component to One messages such as Notifications and Comments.

As we can see, the elegantly simple Mittō schema, is robust enough to model most computing communications and will play a significant role in addressing the challenges of efficiently routing a message to it's intended recipient(s) in a web scale distributed environment.

Certiorari (Pub-Sub pattern)

Certiorari is Middle English, from Latin, literally, to be informed. Certiorari, as we use it, is a term meaning the act of notifying. The challenge of notification in computer science is a complex one. Who should be notified, when should they be notified, what should they be notified of. This is an operation that is separate from message sending. A necessary outcome of notification is message delivery. The act of requesting to be notified of changes is characterized as " subscribing" or listening to a channel. It follows that a process must manage or push messages into a channel. This is characterized as "publishing". One way to solve the notification problem (and subsequent delivery at the same time) is the publish-subscribe pattern or Pub/Sub/Mediate model in computer science.

Publish (push message into channel) – Subscribe (listen to a channel) is a messaging pattern where senders of messages, called publishers, do not program the messages to be sent directly to specific receivers, called subscribers. Instead, published messages are characterized into classes, without knowledge of what, if any, subscribers there may be. Similarly, subscribers express interest in one or more classes, and only receive messages that are of interest, without knowledge of what, if any, publishers there are.

(Cite Wikipedia) In the pub/sub model, subscribers typically receive only a subset of the total messages published. The process of selecting messages for reception and matching their content is called filtering. (Cite Wikipedia) Publishers post messages to an intermediary Mediator (message broker or event bus) and subscribers register subscriptions with that Mediator, letting the Mediator perform the filtering.

The range of communication relationships from "many-to-many", to "many-to-one" requires some embodiment of a mediator concept to facilitate the sending, routing and delivery of messages. A mediator coordinates interactions (logic and behavior) thus facilitating communication back and forth. A mediator is an intervening entity such as a message broker or event bus, which performs a store and forward function to route messages.

This is in essence the Publish-Subscribe pattern or Pub/Sub/Mediate model. Pub/Sub/Mediate is something that is a capability that emerges from network topology, design, and technology choices.

If we are to solve the notification problem, which, as just illustrated is a complex one, requiring a separate schema such as the Pub/Sub/Mediate model, first, we must deconstruct the problem of notification into its constituent parts, namely Matching (filtering)

and event delivery. Matching is the problem of finding all the subscriptions that match a given notification. Event delivery is the task of delivering the notification to the set of interested subscribers selected with matching.

First let's examine how this is handled in the Pub/Sub/Mediate mode, which is the current state of the art. We begin by observing that the matching of messages assumes that the receiver of the message (the Mediator) receives messages that are not of interest, thus a filter must be implemented. Next, the Publisher in the Pub/Sub/Mediate model posts ALL messages to the same cluster or network of Mediators, because the Publisher has no knowledge of the message itself and how or where it might be routed prior to sending of the message. Finally, it is the responsibility of the Subscriber to classify messages. This classification is used by the Mediator to make a selection/filter on behalf of the Subscriber.

These assumptions do not have any significant limitations when the number of Subscribers is limited to a single Mediator cluster. Severe limitations begin to appear however, in wide-area or "Internet-wide" scale applications. (http://www0.cs.ucl.ac.uk/staff/C.Raiciu/files/craiciu-revisiting.pdf). The problem arises specifically due to compromises between the extremes of publisher-side filtering of messages (with events directly transmitted to interested subscribers) and subscriber-side filtering of messages (with events broadcast to all subscribers). Load balancing in a web-scale messaging system using this model require each Mediator to receive and therefore filter, all messages published in the system, since the Publisher has no knowledge of who is to receive a message, and therefore the message must be distributed to all possible Mediator instances, with the majority of Mediator instances discarding the message.

The Mittō schema departs from this concept in that it encapsulates both the target of the communication and the context of the communication, therefore the communication can be uniquely identified. This target and context are fully described in the Mittō subscription request. Specifically, the subscription request can uniquely identify *the identity claim (from:), the content claim (to:), and the context claim (re:) of all messages*therefore the Mediator (or Mediator cluster) responsible for routing those messages can be derived from the Mittō message schema. Thus, alleviating the burden of matching and filtering from the Mediator, which is one of the core (and resource intensive) functions of the Mediator.

This affects the act of notification or Certorari. The matching/filtering task is no longer required, as we can now calculate the server

responsible for delivering a message from the properties of the Mittō schema itself, specifically the unique identifiers stored in To (a uSer or object), From (uSer or Object) and Re (uSer and Object). The requirement of the publisher to broadcast the message to all mediators in the system in the Pub/Sub/Mediate model, is no longer a valid concern. Event or message delivery can now be accomplished using any number of technologies. These identifiers can be hashed and the result can be used to calculate the server handling the message thus: Server = Hash(To + From + Re) (mod) Number of Servers. And so on....

Mittō messages are stored persistently in the socialMode data store. This affects the act of notification or Certiorari as follows: The store is implemented such that a query for a conversation (specified by the Mittō key combination of To + From + Re), returns the last "n" messages of the conversation, ordered by time (identiTY.created). This feature allows clients to:

1. efficiently poll for conversation updates (as in email style applications),
2. initialize conversation history or streams at login,
3. or, recover conversation history or streams after system errors (server or application failure).

In the case of urgent conversations (sms or chat style), the application queries the conversation stream for the initial state, then subscribes to the conversation stream uniquely identified in the Mittō, thus receiving all subsequent conversation updates in real time.

All Mittō messages are a component and therefore have Idenity(TY)—Content-(NT) andContext (XT). have a unique identifier. This affects the act of notification or Certiorari as follows: Consistency can be maintained by applications by verifying a message has not already been received. Mittō messages also are time stamped (identiTY.created) and order can be maintained. In the case of a message arriving out of order, the application can poll the socialMoDe persistent store to insure consistency.

Finally, the Pub/Sub/Mediate model has a significant limitation in the area of control because the publisher has no "knowledge of what, if any, subscribers there may be". The Mediator cannot affect control of the message, since it's functions are limited to matching/filtering/routing. Consequently, the publisher of the message has been deprived of control of the use of the message. For example, in existing email systems, once the send key is pressed, there can be no retrieval of the message. In the Onli Os (CloudMoDe) model (on which Mitto/Certiorari is constructed) control is attached to a Use Policy applied to an instance of a component, and thus falls

under accessibility. As such, the user retains control of the communication (message or conversation), providing the user a fine level of control. For example, when a user sends an email in SocialMoDe, the email can be recalled or deleted at any time in the communication, thus controlling the ephemerality of the communication.

By separating the concerns of sending a message (Mittō), notification of subscribers and message delivery (Certiorari), we can see how the Mittō model of messages, and the Certiorari model of notification and delivery, can be used to deliver an efficient, scaleable, performant and complete communication system. Almost every form of communication we use in computing can now be modeled using our Mittō and Certiorari schema. The implementation of this is collectively called *socialMode*.

You put your data into socialMode when you communicate about it. we call this realization *the Object is the Subject and the Subject is the Object*.
Our philosophy posits that organization is a valid extent of a thing and therefore it is organization that makes a thing a thing.
In this context we discovered that it is the organization of the message determines the method selected by the sender to communicate with the recipient. We realized this insight in creating a different model or schema for communication.

There are two very practical benefits that emerge from modeling communication in this manner, both the result of the communication target and context being completely described using a common and consistent syntax and semantics: The first benefit is a significant increase in the scalability and performance over that which is possible in the current pub/sub/mediate model. The second is the ability of the user to retain control of their communication throughout the life of the conversation.

No doubt this discovery will affect our user Interface designs and make a profound impact in the domain of UI/UX concepts, but this is not the topic of this paper. What we are concerned with here is that the result of Mittō/ Ceriorari schema is a uniform communication model (socialMoDe) that, when implemented, is tightly coupled to the platform (dataMode, objectMoDe, Onli Os (CloudMoDe)), at it's most fundamental level and thus it is universally accessible..

CLOUDMODE LLC PRESENTS

VESCEL

MEDIA LAUNCH EVENT / 25 MAY

SECURE STORAGE +SMART CONTRACTS

#onlios #benefits

BENEFITS

Nov 20, 2011. Post by Dhryl Anton
Updated Nov 24 2021

The First Industrial Revolution used water and steam power to mechanize production. The second industrial revolution (1870 - 1914) was powered by oil, steel, electricity, and the internal combustion engine. The Third used information technology to automate production. The Fourth Revolution is the application of information and communication technologies to industry. Today, I believe, we are at the dawn of another fundamental shift in human society. It is powered by: the Advanced Materials Revolution, where the use of new materials have a profound impact on industry and central to this is The Data Revolution, where the intelligence you get from the data, which is generated from digital logic circuits and its derived technologies, including the computer, phone, the Cloud and the Internet, is the thing that creates new value.

Consider what you get from predictive analytics using "Big Data" techniques to predict possible outcomes. The Clean Energy Movement as the drive to move away from fossil fuels intensifies due to environmental demands, political forces and scarcity, would simply be not possible without the immediacy of the ability to measure and distribute the results. Advances in the foundational technologies that are the heart of Advanced Materials, Data and Clean Energy will play a pivotal role in the ecology of human enterprise, as did core advancements in the foundational technologies did during the Industrial Revolution. It is not the invention of something new, so much as the innovation that makes things more useful, that will pivot and give birth to entire industries. Much like the Bessemer Process did for steel, Edison and Swan's advancement of the Light Bulb, and Gottlieb Daimler's gas powered internal combustion engine. If you look at top of today's fortune 100 companies you will find many of them are left over remnants of the second industrial revolution. Core transformations and paradigm shifts in technologies represent the engines of evolution, which will shape this new form of human society.

Onli Os (CloudMoDe) begins with a revolution in the data model. In computer software a data model is a schema used to organize data. A data model includes the process of applying formal descriptions to physical data, including the specification of data tables and relationships between database tables. These elements and structures facilitate queries used to retrieve or summarize data. The main aim of a data model is to support the development of information systems by providing the definition and format of data. The design and implementation of data models can have a significant impact on getting data in, integrating data and getting data out. The benefit of a semantic approach that is implemented from the ground up makes a practical contribution in costs, performance, capability, maintenance and security.

#onlios# #benefits

BENEFITS

Onli OS, while not a physical transformation but rather an organizational paradigm shift is a core transformation of data. Put simply, our technology posits a means and method to make data intelligible. Intelligible Data is a whole new way to store data, move it around and keep it safe. It is fundamentally the use of an organization of data as the core element of an information system rather than raw, user-generated, data. Intelligible Data captures the relationship between data and treats that relationship as a single ontological unit. The implications of this in the storage, delivery and control of data is profound, when you apply it.

The concept is elegantly simple. For example, when you think of a Frank Sinatra song, you access and assemble multiple pieces of data. You bring up an image (a representation) such as an album cover, a memory of a time and place, or movie, or a photo. This is in addition to the sound recording, the audio, itself. In the context of making a selection of a Frank Sinatra song from a list of songs, in order to make a selection, you must also be confronted by the computing system, with a set of representations (several images) from which you can select. You select a representation, not the sound recording, once the selection is made then you expect the sound recording to begin playing. Therefore a Frank Sinatra sound recording is, for all practical purposes, a stack of data: a representation and a sound recording, which may or may not include other forms of data such as the lyrics and/or the liner notes. This means the data is intelligible, that is, it is understood by you and the computing system. If the computing system just confronted the user with a list of raw files with no representation no selection can be made because the data is not intelligible. You would have to listen

to each sound recording to determine, which is Frank Sinatra. This is an over simplification of the concept but it clearly illustrates that the value of one aspect of intelligibleness, representation, is not difficult to understand.

Intelligible Data is an organization of data that recognizes the reality that human beings refer to the relationship of a thing in a context as the target of their communication, rather than the thing itself. In order words the subject is an object. A manufacturing process that makes all kinds of data Intelligible has some very practical, real world benefits. By providing, at its most fundamental level, a method to store the relationships between data, we make data more useful. This permits us to create new models of computing.

The internet is all about data. A revolutionary new way of organizing that data, storing it, accessing it, searching it, presenting it, moving it around and keeping it safe, is obviously going to impact lots of domains.

Programmers use and deal with objects, and data resides in tables related by common fields (keys), there is usually a mismatch or “impedance” YouTube crash course. This increases over time. The simple fact of life is that data relationships change. Enterprises evolve and therefore code is constantly changing. This is why solving the object relational impedance problem is often called the Vietnam of Computer Science. see [Liam McLennan talk on ORMs](https://www.youtube.com/watch?v=v1N5FYspT0Y). The bottom line is that code comes and goes but data is forever.

The context of Onli OS is a model of data. The invention is a model that has a very high utility as it is robust enough to model most occurrences. Models have a concrete value. Innovative Models don’t just disrupt the existing status quo. They make the old models obsolete.

The Benefit of Application

Onli Os (CloudMoDe) is an operating stack based on a unified semantic binary model of data, where the semantics of the model are expressed as binary relationships (and collections of binary relationships) between the attributes of the model. This elegant structure forms the conceptual foundation and

determines, at the logical level, the manner in which data can be stored, organized and manipulated in a data storage system. It's infrastructure is the model by which data is implemented in a physical data model and thus defines the usability of operations that can be performed on the data. This structure offers a simple, natural, implementation-independent, flexible, and non-redundant specification of information. It makes it possible to implement a Structural Controls Mechanism, solve the problem of Uniqueness, and many more things that we cannot even imagine at this point.

Let us look at some of the practical benefits of a unified structured object model in the area of cost, maintenance, performance and security.

Costs

There are multiple components to the cost of design and implementation of a data model: requirements gathering (understanding what data is going to be modeled, the conceptual schema), requirements analysis (understanding how data will be stored and accessed, the logical schema), software implementation (implementation of the data storage mechanism, the physical schema, and design and implementation of the transformation model that translates data between physical and logical schema), and software and integration testing (testing and verification of the data model (physical and logical) before and after it is integrated into it's target application(s). Hidden costs associated with development of interfaces used to share data between systems account for 25-70% of the cost of current systems (Developing High Quality Data Models, Matthew West, Julian Fowler, Shell International Limited, 1999). Other hidden costs include 'brittleness' (where the data model breaks when attempts to extend it into other application areas are attempted), and missed opportunity costs when inadequate data models 'hide' data and relationships from users, due to the data model's inherent inability to reliably capture and store relevant data. And finally, data cannot be shared between customers and suppliers due to non-standard formats

and models.

A unified semantic model of the physical and logical schema would result in significant software development cost savings. Requirements gathering and analysis are reduced to identifying where attributes should be placed in the model, instead of creating a new structure of attributes and relationships with each application. The implementation phase (data storage, logical-physical mapping and translation) becomes a simple coding exercise, placing the attributes in the appropriate location in the model. Testing and verification of the data model prior to integration testing is greatly simplified also, requiring only the verification of data using existing (and proven) test software.

The problem of brittleness is eliminated, since the underlying model is robust, correct and easily extensible. Hidden data relationships do not exist, and data is easily shared between customers and suppliers, since the model of the data is shared by default. A new relationship can be effortless implemented and stored separately from the original data object. A unified semantic model would have tremendous impact on costs.

Maintenance

Data model designs and implementations evolve and grow as new information becomes available in a particular domain. Very few data models remain static. This evolutionary process involves the same steps as the development of the original model (requirements gathering, requirements analysis, software implementation, software test, integration test), only these must be done in the context of an existing model. Most often these tasks are undertaken by engineers that were not involved in the original model design and implementation. If the data model is not completely documented (and the documentation maintained) the personnel responsible for evolving the data model may not be aware of assumptions implicit in the design and/or implementation of the data model. Often these assumptions are uncovered only during the software and integration testing phases (or worse once the applications have been deployed), requiring the data model design/ implementation process be repeated. An expensive process.

When a data model is moved to a new technology platform (to take advantage of cost/performance advances), the model may need to be changed in order to take advantage of the new technology, perhaps even requiring a complete re-implementation of the model. These maintenance costs can easily exceed the cost of the original data model design and development by a factor of 1.5-2.0. (E. Burt Swanson, The dimensions of maintenance. Proceedings of the 2nd international conference on Software engineering, San Francisco, 1976, pp 492 — 497)

A unified semantic model of the physical and logical schema would significantly reduce maintenance costs. Maintenance could be reduced to identifying where attributes should be placed in the model. In fact all of the cost saving benefits described in the COST section are even more relevant in the domain of software maintenance. For example, since the data model is semantic in nature, documentation requirements of a specific implementation can be significantly reduced, minimizing costs up front, and costs downstream. Once developers are comfortable with the standardized semantics of the unified model, it's attributes and relationships, specific implementations are much easier to interpret, understand, maintain and extend.

Performance

A unified semantic model of the physical and logical schema would significantly affect performance of the data model in terms of speed are significantly impacted by both the data model design and implementation. Performance costs are realized first in the cost of the hardware and software systems required to provide reasonable response time when storing and accessing the data by the application. Hidden performance costs are the result of the amount of time users spend waiting for system to respond to both requests to store and retrieve data from the data model. These hidden costs can also include missed opportunity costs due to delays in either the development of maintenance of the applications dependent upon the data model.

The deployment costs associated with a particular data model are heavily influenced by the amount of translation/

transformation between the physical and logical schema. These costs are a function of the complexity of the model (number of attributes, and number of relationships between the attributes). For example, if we have a data model with 10 attributes, and 10 relationships, the addition of a single attribute and 10 relationships (a relationship between the the new attribute and each of the existing attributes) will double the amount of computing resources required to transform/translate between logical and physical models.

Traditional data storage systems optimize for storage space of and for high performance access of the physical schema. This comes at a cost when this physical schema must be translated/ transformed into the logical schema. This transformational process (from physical to logical when reading the data, and from logical to physical when writing the data) is a difficult problem, because the physical schema rarely maps to the logical schema in any straightforward manner. This transformation model has been referred to as the "Vietnam of Computer Science", requiring significant engineering resources to accomplish in an effective and efficient manner. An example is the complex task of transforming a relational table structure (the physical model) to an object model that is designed to model the real world (the logical schema).

References: (Neward, Ted (2006-06-26). "The Vietnam of Computer Science". Interoperability Happens. Retrieved 2011-01-08).

Security

Traditional security for files is based on what is called an access control list or ACL, a list of permissions attached to a file. Capability based security achieve improved file security by implementing a set of privileges associated with a file, specifying the access rights of the capability.

An ACL specifies which users are granted access to files, and what operations that user is allowed on that file. The file is accessed by a filename, which is a 'forgeable' reference, typically in plain text (for example a path name like '/Users/franksmith/Documents/mykitty.png'. This pathname does not specify any type of access rights for the file. Instead,

these access rights are maintained in a separate structure (an inherently insecure structure, which in all operating systems can be attacked or exploited in a myriad of ways), which requires the operating system to validate any attempt to access the file. If the operating system in charge of the ACL is compromised in any way, the files controlled by the OS and ACL, are inherently compromised. The semantics of ACLs have been proven to be insecure in many situations, e.g., the confused deputy problem (http://en.wikipedia.org/wiki/Confused_deputy_problem), where a computer program is innocently fooled by some other party into misusing it's authority.

Capability based security achieve improved file security by implementing a set of privileges associated with a file, specifying the access rights of the capability. A capability is much like a car key. It works in a specific car (designates a particular object or file) and anyone holding the key can perform certain actions (open the car door, start the car). Just like a valet key can specify a subset of actions from the master key (can't open trunk, console or glove box), capabilities can designate the same object/file, but authorize different sets of actions. And just like a car key, a capability can be delegated (handing the key/capability to another trusted individual) and copied. A car key is not forgeable, meaning I can't go someplace and have a key made for my neighbor's car. If I find that someone has a key to my car, I can have my car re-keyed.

Capabilities are easier to share than keys, as they are just data. They are more difficult to bypass (unlike ACL and password schemes). You have to have a valid capability to access the file. And capabilities can be revoked when they are compromised, by creating a container (a containing object) that contains the referenced object. By deleting the container, the referenced object can no longer be accessed, because the key of the container is invalid, throughout the entire system.

Advances in security is taking place all the time. Being able to implement them however is an altogether different matter. Especially when you have complex data models connected to MVC and dependent (hard wired) UI's, whose logical and physical schema are already set in stone, so to speak.

A unified semantic model of the physical and logical schema makes it possible to apply capability based models of security. New security schemes can be added retroactively without reengineering or compromising the system.

Conclusion

Onli Os (CloudMoDe) begins with a revolution in the data model. In computer software a data model is a schema used to organize data. A data model includes the process of applying formal descriptions to physical data, including the specification of data tables and relationships between database tables. These elements and structures facilitate queries used to retrieve or summarize data. The main aim of a data model is to support the development of information systems by providing the definition and format of data. The design and implementation of data models can have a significant impact on getting data in, integrating data and getting data out. The benefit of a semantic approach that is implemented from the ground up makes a practical contribution in costs, performance, capability, maintenance and security.

#onlios #implementation

IMPLEMENTATION

MICHAEL MCFALL

The desired result is a strongly consistent way to get data in and out of a datastore that enforces the organization and methods.

Data storage is the collective methods and technologies that capture and retain digital information. A database is a series of bytes that is managed by a database management system (DBMS). A file is a series of bytes that is managed by a file system. A data store is a repository for persistently storing collections of data,. It can be a database or a file system. The basic element is an EAV model. Data is recorded as three columns:

* The _entity_: the item being described.
* The _attribute_ or _parameter_: typically implemented as a foreign key into a table of attribute definitions. The attribute definitions table might contain the following columns: an attribute ID, attribute name, description, data type , and columns assisting input validation, e.g., maximum string length and regular expression, set of permissible values, etc.

*** The _value_ of the attribute.**

The core requirement of the data store/database is as an object repository, as logic is implemented outside the database. The minimum functionality of the system is defined as serializing objects in and out of the database/datastore. To do that we need a strongly consistent key value store. A key-value store is a database which uses an array of keys where each key is associated with only one value in a collection. Key-value stores usually

do not have query languages as in RDBMS to retrieve data. They only provide some simple operations such as get, put (update) and delete. The data querying (retrieving) can be handled manually at the application level. Decoupling the model from the query language is important as methods and logic can depart from conventional uses. Any EAV model can be modeled in a Key Value Store. [From RDBMS to Key-Value Store: Data Modeling Techniques | by Wishmitha S. Mendis | Medium](_https://medium.com/@wishmithasmendis/from-rdbms-to-key-value-store-data-modeling-techniques-a2874906bc46_) Example. The store module implements the hstore data type for storing sets of key/value pairs within a single PostgreSQL value.

Other Considerations

ACID (Atomicity, Consistency, Isolation, Durability) compliance is a very desirable feature when selecting a data store. As stated here (https://databricks.com/glossary/acid-transactions), ACID transactions ensure the highest possible data reliability and integrity (visit the link for a complete definition). ACID compliant transactions gurantee data consistency across widely distributed systems, even in cloud based distributed databases, where individual system nodes can be in different cabinets within a data center, or in different data centers in far flung geographic regions. A decade ago, only large relational database systems had ACID transaction compliant systems, but now multiple KV stores as well as a number of noSQL systems support ACID transactions out of the box.

As an application grows in both functionality and applicability, it can (and will) take advantage of fully distributed systems, as described above. This means that application technologies must be selected that meet CAP requirements for our system. the CAP (Consistent, Available, Partition Tolerant) theorem,(-described in) https://towardsdatascience.com/cap-theorem-and-distributed-database-management-systems-5c2be977950e). Essentially, the CAP theorem states that in the presence of partitions (such as network failures or reconfigurations), a system cannot be both consistent AND available, and a choice must be made between the two. In general, OnliOS chooses consistency over availability, and is therefore what is known as a CP system in CAP terminology. In business (particularly financial systems) both consistency and availability are crucial. Any Distributed Database system that we select for implementation requires some tradeoffs. Further discussion of the CAP theorem and it's extension PACELC, in combination with ACID and BASE at https://www.itechart.com/blog/all-you-didnt-know-about-cap-theorem/ illustrate that the tradeoffs of a particular implementation when placed in a real world scenario.

High Availability as mentioned above is still a business requirement, with the goal of minimizing service outages for customers. High Availability is typically accomplished in a distributed database with a combination of replication (of data across multiple nodes) accompanied by a smart failover system, where clients are always able to read from an up to date copy of the data. Such a system would be considered to be CA in CAP terminology. However, transactions, and specifically transactions in the context of two phase commits (https://en.wikipedia.org/wiki/Two-phase_commit_protocol), a type of distributed, atomic algorithm that insures consistency of data, throw an interesting challenge into this formula.

Transactions in non-distributed datastores are well understood and have been implemented in many types of databases. However, distributed transactions for use in large-scale distributed systems have only become broadly available in the open source community in the last decade. Suffice it to say that the state of the art when we first set out on implementing OnliOS was not even an art form. As such we developed our own distributed transaction manager that leveraged a microservice architecture.

Microservices (or microservices architecture, https://microservices.io/patterns/microservices.html) are an architectural approach to building computing applications in which a single application is composed of many loosely coupled and independently deployable smaller components, or services. An event-driven architecture uses a common event bus and events to trigger and communicate between decoupled services. An event is a message and/or a change in state. Event bus stores events and provide a loosely coupled communication system between objects, services and other consumers of the events.

The loose coupling of a microservices architecture presents some interesting challenges. Specifically when a service creates a transaction, commits it and a subsequent downstream service is unable to complete the distributed (cross-service) portion of the transaction. In this case a distributed transaction manager must be able to control/commit and rollback different portions of the transaction. These types of distributed transactions have been around since long before cloud based distributed databases were created. Take an ATM transaction for example, where the client system is disconnected from an intermediate server (a VISA rail for example). In such cases, failures result in compensating transactions across the entire network. OnliOS includes such a distributed transaction manager, that manages sequences of atomic transactions distributed across the network.

Lessons

The current release of OnliOS is the result of over a decades work addressing these issues, across multiple generations of distributed systems and database technologies.

Consistency

Data consistency is the highest priority in OnliOS. Data consistency must be designed into the architecture and implemntation of appliances. First a significant investment in the communiation layers between agents (services), appliances (applications) and trays (clients) is required. The Ghost protocol as implemented in OnliOS has over a decade of development work, and provides a robust and rich feature set for distributed RPC's over international communication networks.

Second reliable and performant datastores need to be implemented for each level of service. Various versions of the OnliOS have used raw KV stores, NoSQL datastores, and relational datastores.As such, OnliOS appliances can be implemented using the best tools for the job.

Third, we realize that every system will have failure scenarios that are outside of any business service guarantees. Our experience with building a full-fledged, distributed transaction manager has given us a powerful tool that insures that independent of the underlying data storage technology, any particular appliance can implement compensating transaction systems that meet busiess requirements and service level requirements.

Replication

OnliOS systems are implemented with a replication architecture, where at least 3 copies of every persistent store is maintained; 3 instances of every critical service is always available; and when required multiple appliance instances are maintained in disparate data centers. In addition, disaster recovery scenarios are implemented, such that in the event of a natural disaster, resulting in long term power outage, or system destruction, the customers OnliOS system can be rapidly re-constituted. In some cases, a hot-backup replica can also be maintained, available for instant failover.

Availability and Reliability

Mission critical applications require dedicated hardware solutions in order to meet the needs of our customers. This includes, compute, storage, networking and security. As such, we maintain dedicated systems, in multiple data centers, all connected using our own private networking systems. All customer data is stored behind secure hardware

firewalls, that is not addressable/reachable by the public internet. With this architecture, and the inherent replicated architecture of OnliOS, we can provide businesses with the availability they need to deliver to their customers.

Natural DSL

Domain Specific Languages (DSL) are computing languages designed to do one thing, and to do it well (https://en.wikipedia.org/wiki/Domain-specific_language). With OnliOS, we have implemented three different DSLs (with a fourth coming soon), that share a common yaml syntac, that provide customers with the ability to build applications in an incredibly short period of Tim. By implementing these Domain Specific Languages, customers can define their systems at a very high level, without having to dive into the details of a general purpose programming language like go or C#.

DSL's as used in the OnliOS provide a number of important functions within the overall design. The first is communication: each DSL can be quickly communicated (and therefore learned) in a very short period of time. The DSL is implemented in such a way to provide immediate feedback to the user when there is basic problem with the definition (error checking). The DSL itself can be examined to identify errors much more quickly than with a general purpose language. Finally, the output of the DSL is guaranteed to operate within the OnliOS in regards to data consistency, replication, availability and reliability, as the output is literally built on the OnliOS libraries.

These DSL's provide a high level, Natural interface for:

Objects:

The OnliOS Object Natural DSL provides a language to define Objects, in such a way that all basic persistence functions, all wire protocol functions are generated and compiled in advance. The output of this Natural DSL is guaranted to install, run and perform as fast as any hand tuned code.

Use Policies:

The OnliOS Use Policy Natural DSL provides a complete language for defining terms and conditions of how an Object is used in the system. This Use Policy is dynamic, in that it is not static and can be changed on the fly. In fact, Use Policies can be programmed in such a way that they can modify persistent data, and therfore become self modifying.

Agent Policies:

The OnliOS Agent Policy Natural DSL provides a complete language for defining

the actions of an Agent. Agents coordinate and operate on Objects (including Events), which have their own Use Policies. Any Agent that operates on multiple objects simultaneously (or iteratively) is implemented using this Agent Policy Natural DSL

Specifics

OnliOS is layered on top of best of class, industry standard software as follows:

- Orchestration: Kubernetes, Rancher
- Networks: etcd, Consul
- Messaging: Kafka, Zookeeper
- Search and Reporting: ElasticSearch and Kibana
- Datastore: CockroachDB, Mongo v4, Badger

Nomenclature

* Organism- a organization of intelligible semantic media.
* Event - a change in state of a message
* Event bus is a mediator that transfers a message from a sender to a receiver.
* Event Definition -Datamode Model
* Event Continuum (a place where a series of events occur and are stored)
* Agents - set of processes that act on and interact with events.
* AI Agents - recognize patterns in the continuum and draws inferences, the results of which become new events
* Appliance (client side)- present Events to User in a View
* Authentication service - authenticate user and authorize connections
* Assemblers - integrate and organize resources in an appliance view
* Appliance use Agents to accomplish tasks (eg: usermode)
* Request - communication instruction
* RequestService —stores and manages requests —will need to Access Catalog (authentication)
* RequestManager -process for managing requests
* Resources -assets necessary for performing a task
* ResourceTask -assigned computing process
* ResourceTaskManagement - a set of processes to manage processes
* Catalog (the resources of the agent)

OPPORTUNITY

Onli, collectively speaking, is a foundation to create an entirely new classes of computing applications we call appliances. Appliances are distinct from apps (applications that run on mobile devices) and applications (applications that run on desktop devices) , in a client/server architecture, in that Appliances are computing applications that move intelligible data around.

One lucrative market is the use of intelligible data to create micro-commodities (digital assets) and Appliances that enabling the trading of said class of asset

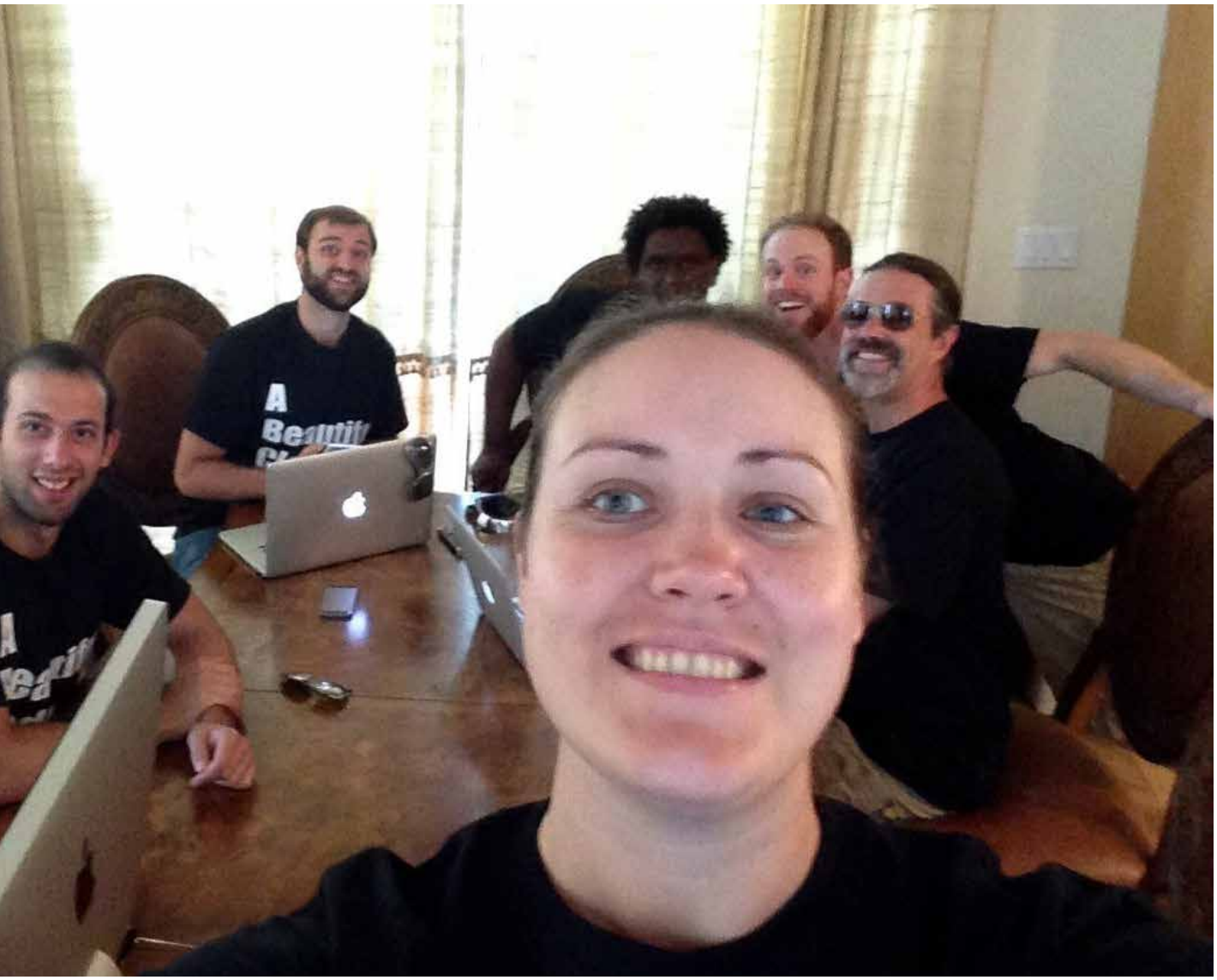
The ONLI Corporation builds and maintains the ecosystem, builds tools and technologies that will help developers build appliances. .

Onli is a light bulb to the candle of DLT technology. A lightbulb is not a new kind of candle. It is a different way of generating light.

#onlios #application

APPLICATION

You can only truly understand a technology when you have spent some time applying it. Much more is realized in the application that in the invention. Over the past 10 years we have applied the technology on a broad spectrum of computing. In 2021 we believe the technology is mature enough and tested enough to be able to move to the productization phase.



We have taken our database technology, combined with everything that we have learned in the past 12 years to build a single vision we call Onli.

Onli has all the benefits with none of the problems of DLT's. The company intends to leverage the Onli Technology to give its products a competitive advantage.

ONLI

An Onli is a dataset generated by a uniqueness quantification algorithm, where operations (copy, read, write or update) evolve the dataset by hashing the previous state and storing it in the structure, as a chain of blocks, thus advancing the state during each transaction. This enables the dataset to maintain a real time single global state across a network of connected devices. Each Onli dataset is stored as a single capsule, that is designed to record changes in ownership in an immutable blockchain, guaranteeing the uniqueness of the Onli across multiple changes in ownership.

Onli's are minted not mined. Onli's are issued by a centralized dedicated process called a Treasury Agent. Ownership transfers are executed in the central Settlement Service. Since each Onli is stored as a single capsule of data, transactions are settled (ownership transfer finalized, and recorded) nearly instantaneously, achieving rapid payment finality.

Every Onli has an owner. It is not anonymous. All owners of Onlis are strongly authenticated, using the same personal blockchain authentication and identity management system that incorporates multi-factor authentication (MFA), out of band communication with the authentication system, one-time single use private keys (OTP) and bio-metric systems. Any use of the Onli (copy, read, write or update) requires the owner's personal blockchain to provide a transaction authentication key, evolving both the Onli and the owner's personal blockchain.

Onli transactions are private but not anonymous. An Onli is designed to record changes in ownership on its blockchain, therefore there must be an explicitly identified owner. A record of ownership of each Onli is maintained by the central treasury as well as the independent ownership-possession verification that is established through a distributed validation network (DVN) maintained by authorized users and members of the Onli Exchange. By default the system does not record ownership history, only the current owner and owner from which ownership of the Onli was transferred. This feature can be tailored to comply with all financial regulations privacy rules, and audit and accountability reporting.

Onli are based on the concept of Actual Possession, therefore there is no ledger that is being copied into a distributed manner. There is no mining as there is no double spend problem (you can't spend what you don't have) to be protected against, nor is there the possibility of a 51% attack (which is a risk of a distributed ledger systems) changing transactions. Security of the transaction is enforced by the strong authentication system combined with a unique, symmetrical and bi-directional protocol using certificate-pinned TLS sockets, eliminating man-in-the-middle attacks. This symmetrical transport system guarantees that no authentication or identity information is ever transported over the same connection as the Onli dataset, securing the Onli dataset from theft during transmission. Since there are no ledgers and no mining, the fee structure can be fixed and stable.

Each Onli is a unitary dataset, meaning that it is a single, unique evolving entity. It is quantified and whole meaning it is neither divisible or fungible.Much like a real coin you can use denominations for subdivisions. Onli cannot be replaced by another identical item, because of its uniqueness. It remains consistently unique across a network of connected devices. This makes an Onli perfectly suited to represent financial assets.

Onli is the only virtual currency that complies with the EU GDPR (General Data Protection Regulation (https://en.wikipedia.org/wiki/General_Data_Protection_Regulation)) privacy laws, and California Consumer Privacy Act of 2018 (https://en.wikipedia.org/wiki/California_Consumer_Privacy_Act).

The Onli Platform

The Onli Platform a set of technologies that make it easy to build and deploy appliances. The platform has a few different components.

* The Onli protocol,
* The Onli one infrastructure,
* The Onli cloud platform and our api,
* Convey, which is our open source middleware,
* Onli ID.
* Vescel - a capability-based container

Together these technologies makes it easy to build applications that move value around.

*Onli*is a protocol. A protocol, in computer science, is a set of rules or procedures that governs the transfer of data between two or more electronic devices. A protocol establishes how the information must be structured and how each party is going to store it, send it and receive it. Onli is a patented protocol that governs how data is going to be stored, how it is going to be transferred, and how it is going to change or evolve when it moves.

The infrastructure on which the protocol runs is called the *Onli One.* It's a private part of the internet. You access the computing resources of the Onli One, like compute, storage, network, statistics, via the Onli Cloud.

The *Onli Cloud* delivers, high performance, cloud computing services for running onli powered value appliances. Among these services are, the all new, Settlement Services, for changing the ownership of onli micro commodities, and Market Services, for executing and validating transactions. Onli Cloud significantly outperforms everyone else, in every category that matters, because, we built our environment, from the ground up, for applications that move value around, for value appliances. Onli Cloud is the cloth. The customer is the tailor.

You build things with Onli. The most common use of Onli is to build micro-commodities. Of course, what you want to do, is to create a market to buy, sell and transfer these micro commodities. To do that you use the Onli Cloud API.

The *Onli Cloud API*is elegantly simple. You place an order through the Marketplace api. and that gives you a receipt. If the order can't be executed then you get an error. With three API calls you can build robust experiences for your customers.

Since this is a backend for an enterprise applications. You will need to convey information from your appliances to the Onli.Cloud. This will require middleware. Middleware is software that provides common services. It lives between your appliance and the Onli Cloud. Our Middleware stack is called, *Convey*. It is open-source. It is built with standard components and run on the ELK stack.

Onli ID, is our Value Storage solution. You are going to need a place to store value. On most protocols, this is called a wallet and you have to build this. On the Onli platform, value is stored in a vault not a wallet. Onli ID gives you authentication, identity management, and value storage. All at no extra cost. Users download ONLI ID, which is a free app on most app stores. The Phone is used by Onli ID as an authentication device and it also serves as their wallet or vault. The phone is the vault. The phone is also the password. Onli ID is an important security, time and cost saving feature for our developers.

Vescel

A Vescel is a programmatic cryptographic container, where a container is an organization of data, stored in a structured computer file, which is cryptographically secure and programmatically accessible.

Vescel is a container. Control over how the

container is used is done with something called a UsePolicy. A UsePolicy is an elegant way of allowing the user to describe the context of use without writing code. Vescel has a powerful programatic feature that allows the user to specify if “this “happens then change the UsePolicy. The “this” in that statement can be any quantified query of any other system. A use policy is a set of controls delineated by terms and conditions. Terms control the disposition of the container such as who, where, when, and frequency (how many times). Conditions are logic that operate on data from Oracles. Oracles are a trusted data source. You can get a value from an external source such as the results from an api call. You then use logic to operate on that value, which results in a change in the terms of the use of the container. All without a single line of code

A Vescel can be encrypted using asymmetric encryption, where the public key is negotiated using a Diffie-Hellman key exchange. This public key can be held in private by the owner of the Vescel, and used to either destructively decrypt the vescel contents or to transfer ownership to another strongly authenticated user of the system.

A Vescel is tightly coupled to the database management system that creates and manages the stored structured computer file. Any event (copy, read, write or update) generates an entry that is stored in the container, resulting in the data stored in the container to be consistently immutable.

Owners of a Vescel are strongly authenticated, using a personal blockchain based identity management system that incorporates multi-factor authentication (MFA), using factors including one-time single use private keys (OTP), unique identifiers such as email and bio-metric systems. The authentication system uses out-of-band communication with servers and end-user applications, whereby no user identification is ever transmitted over the connection secured by the authentication system.

Any use of the Vescel (copy, read, write or update) requires that the owner’s personal blockchain be authenticated to provide a transaction authentication key, evolving both the Vescel and the owner’s personal blockchain.

A Vescel incorporates three additional data structures that
1) control access to the contents of the Vescel,
2) provide a publishable representation of the contents that does not reveal the actual contents
and 3) a set of rules describing how the contents of the Vescel are to be encrypted/decrypted during changes of ownership and or repossession of the contents taken by the

owner.

Vescel makes it possible to realize the promise of smart contracts in an elegant, reliable, scalable and more proficient manner.

Marketability

Onli brings scalability and speed. Stocks and bonds actually settle within two business days after the transaction date. This two-day window is called the T+2. Government bills, bonds, and options settle the next business day. Spot foreign exchange transactions usually settle two business days after the execution date. ACH transfers and other types of money transfers take 2-3 days. Bitcoin takes 24 hours . Visa takes from 24 hours (for debit) to 6 months (for credit) to settle. *Onli settles in 0.20 seconds.* This is competitive advantage.

Onli makes it possible to build new classes of assets that fully realize the benefits of a digital financial instrument that has the properties of provable title, secured ownership as well as secure possession of the asset itself. It is also possible to add these properties to the existing digital assets. This enables new digital assets to be able to fit within existing asset custody regulations and infrastructure/ systems. These features solve many of the challenges of custody in digital assets, including delivery versus payment, auditing, direct asset holding, transit of assets. It makes it possible to create GAAP compatible digital assets.

Finance is founded on the principle of title and accounting. Unique Ownership is the critical component of an asset. Ownership is a bundle of rights: established by legislation (law) and enforced through regulation. Accounting & Auditing are necessary to the operation of a financial asset. Accounting is fundamentally three assertions: The assertion of rights and obligations, the assertion of existence and the assertion of valuation. The root of all these assertions and coincidently of law is property rights, which comes down to one thing, proof of unique ownership: title. Anonymity, consensus of copies, no authentication, these are all the foundations that make crypto work, which are the anthesis of the core fundamentals, the physics as it were, of financial assets. Crypto is not a financial asset. Although there is clearly a market for it. Wrapping crypto in an Onli adds the features of title, strong authentication, ownership, which are needed for financial assets.

Creating a new asset with the features of title and ownership, stability and speed will offer tremendous advantage as it fits within regulatory frameworks. It doesn't attempt to defy the physics of finance.

BDO "/Cryptocurrencies are not financial assets. They also lack physical substance. Therefore, they meet the definition of an in-

tangible asset and would be recorded at acquisition cost (i.e. price paid or consideration given). Intangible assets are subject to an impairment test. Any recognized impairment losses cannot be subsequently reversed. "/

https://www.bdo.com/insights/assurance/financial-reporting/cryptocurrency-the-top-things-you-need-to-know

https://www.forbes.com/sites/shehanchandrasekera/2020/05/21/how-are-cryptocurrencies-classified-in-gaap-financials/?

https://www.accountingtoday.com/news/companies-need-to-be-careful-with-accounting-for-crypto

Onli is a new way to store value, move it around and keep it safe. This is a record of the concepts and ideas behind this unique approach to storing data, moving it around and keep it safe.

This is the property of the Onli Corporation. The purpose of this book is to inform, instruct and teach by presenting a concise statement of national principles. Nothing heren shall be construed as or constitute anything another than this purpose.