REST API's in a CA Plex context

API Design and Integration into CA Plex landscape

Speaker

Software Architect and Consultant at CM First AG, Switzerland since 2008 having 30+ years of experience with the CA Plex / 2E product lines in different roles: developer, supporter, pre-sales, distributor, project manager, teacher, mentor, architect and consultant.

lorenz.alder@cmfirstgroup.com

cm FIRST
Rethink Modernization

Section 1 - API Design

cm FIRST
Rethink Modernization

API Design challenges

- Simple to use (spread)
- Self explaining / in-band out-of-band
- Support major platforms / devices / tooling
- Authentication / Authorisation / Encryption
- Versioning
- Documentation
- Appropriate architectural style
- Human / Machine
- Error reporting / Re-tries

API Design challenges

- Long running processes (asynch)
- Monitor Events / Webhooks (events)
- Data replication / offline clients
- Billing services
- Server Monitoring, Notification, Stats
- Scaleability / Caching / Performance
- Transactions, Web transactions

API consumers

- Consumers are out of your control
- They will do the unexpected
- They will not follow the application domain protocol
- Make sure the API server is robust !

Simple to consume

- Design from a consumer perspective
- Provide API Access Libraries in Java, .Net, Javascript, if necessary

API Self explaining / explorable

- Provide as much information as possible in-band
- Provide links to documentation in-band (switchable for Dev/Prod)

API documentation

- Domain Application Protocol
- State transitions
- Media types
- Link relations
- Headers
- Status Codes

API Consumer Platform support

- Create Libraries for Platforms
- Example : AWS S3

API device support

- Let consumer decide about the page size of collections
- Use the concept of «Resource expansion» to minimize data transfer for mobiles

API Security, Authentication, Authorisation

- Use TLS / HTTPs
- Basic Auth (80% case)
- Oauth 2.0
- OpenId

- Difficult to change later on

API Versioning

- Use version in the URL (provide a «latest» version)
  - Myapi.ex.com/latest/customer/123
  - Myapi.ex.com/v11/customer/123
- Use Version in HTTP Headers

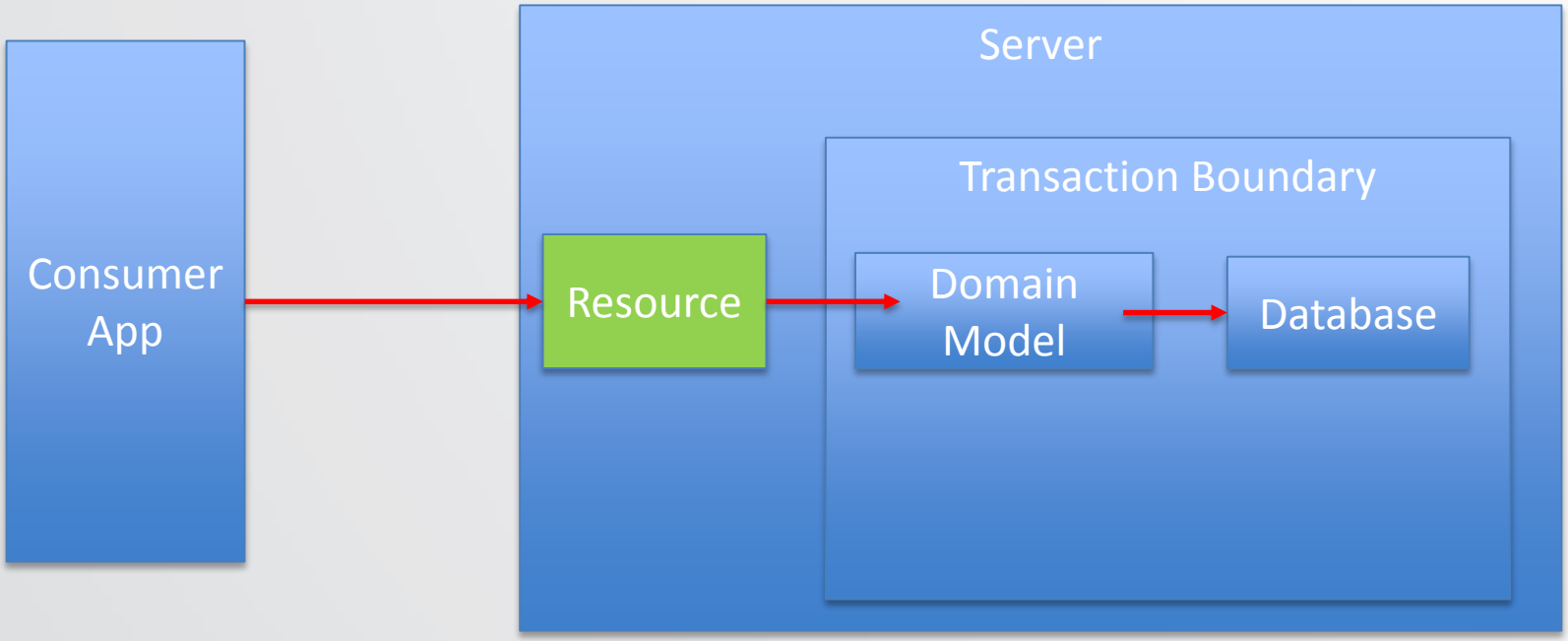
Humans vs. Machine

- If possible use one API for both human and machine web using different Representations of Resources
- Most operations on Business Systems are the the same for machines and humans

Transaction Client

- PUT or POST complete Transaction
- Example: Shopping Basket
  - Send whole order with one request
- Example: Payment
  - Send Payer and Receiver with same request

Transactions Service Boundary



Web Transactions

- Currently no standards
- Can be implemented using classic two phase commit paradigm.
- Transaction modeled as a Resource
- Transaction Coordinator is a REST service

Section 2 - API style

RPC / REST

cm FIRST
Rethink Modernization

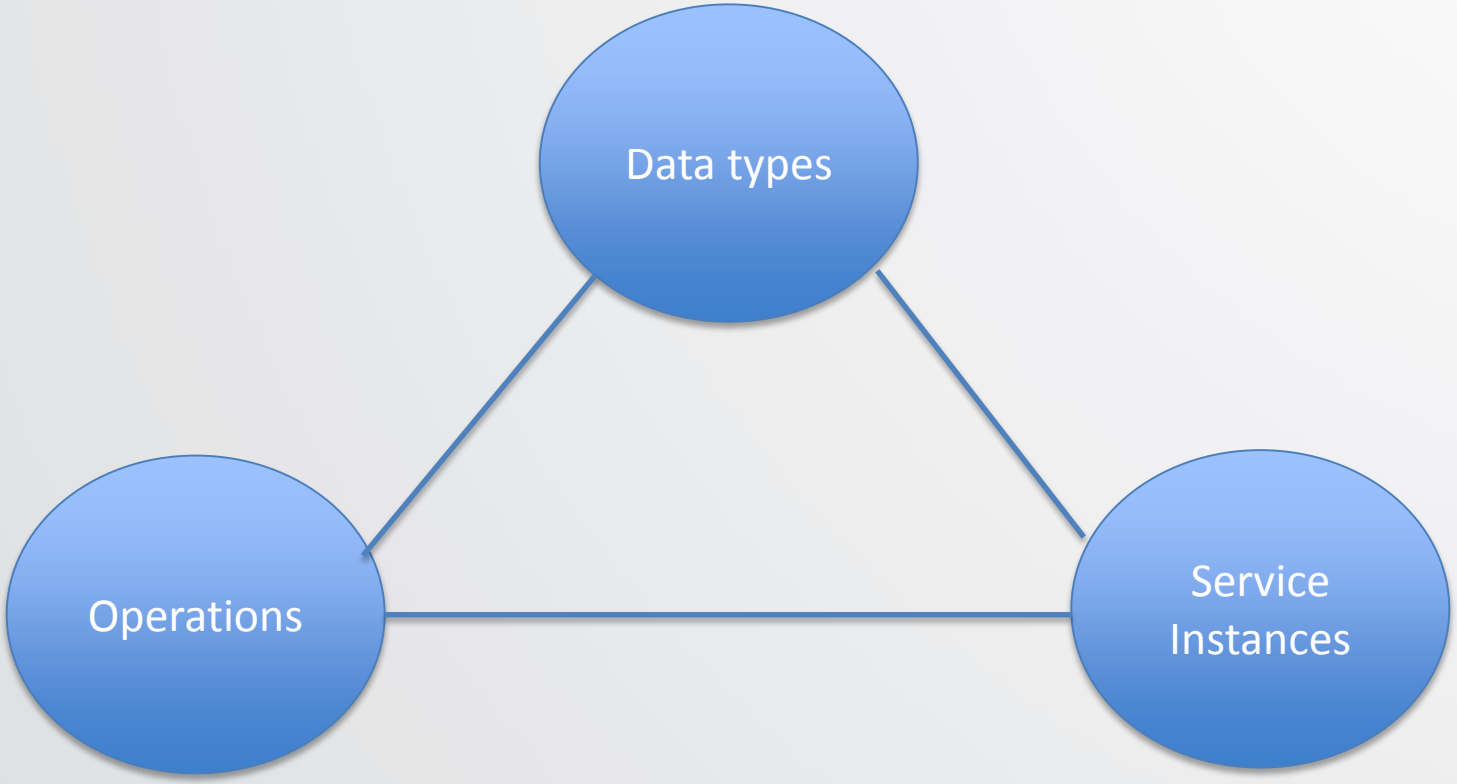
API Styles

- Tunneling (SOAP/RPC)
- URI style (REST based)
- Hypermedia style (RESTful)
- Event driven style

RPC/SOAP vs REST

- RPC is about functions
- REST is about resources
- RPC is about many interfaces
- REST is about a uniform interface

Design Options triangle



Uniform REST interface

```
Interface Resource {
    Resource(URI u);
    Response options();
    Response get();
    Response post(Request r);
    Response put(Request r);
    Response delete();
    Response patch(Request r);
}
```

cm FIRST
Rethink Modernization

RPC remote procedure call (WS*-Stack)

- Well established, Tooling available for major platforms
- WS Reliable Messaging
- WS Atomic Transactions
- WS Security
- Hides the remote aspect of operation (latency)

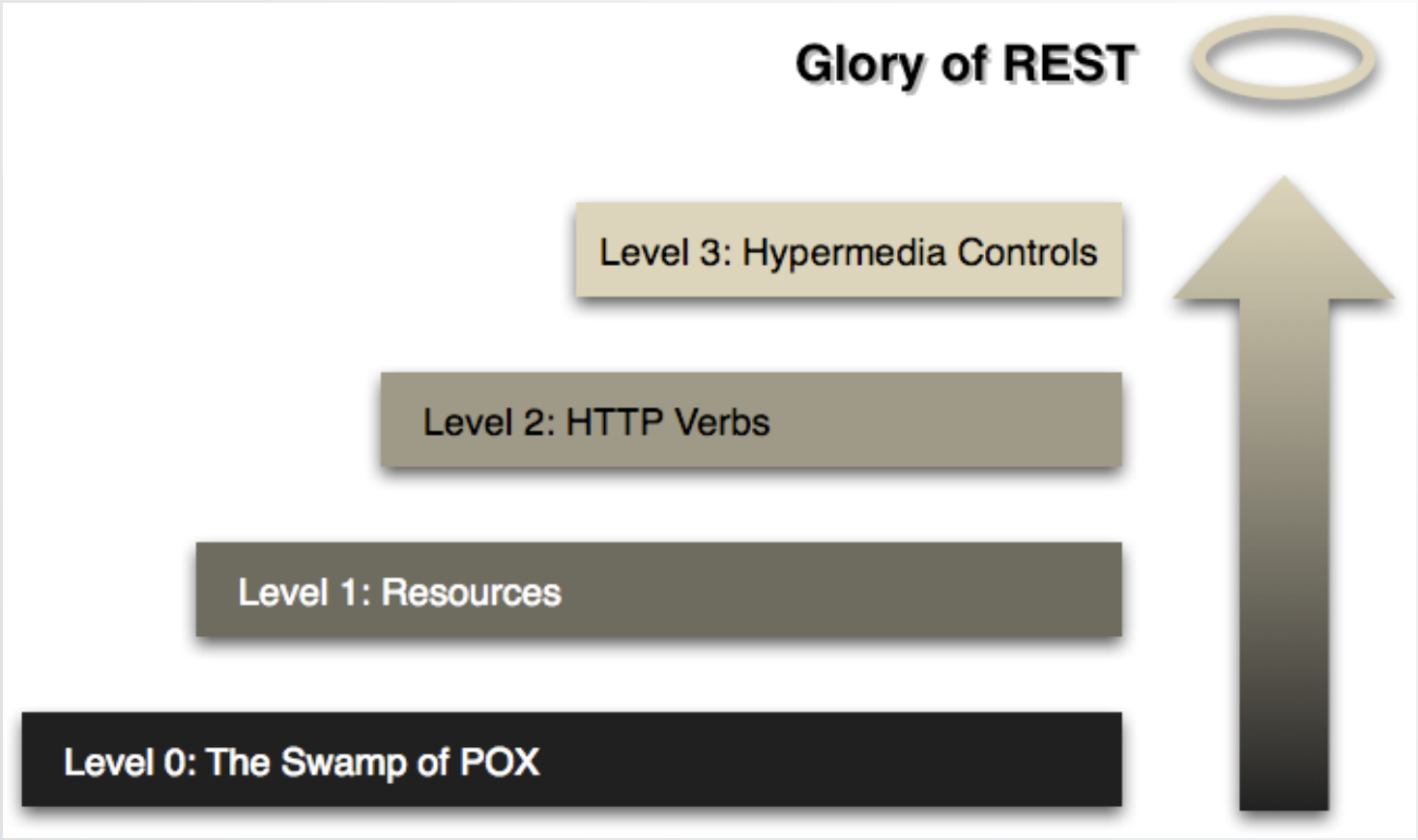
RPC remote procedure call (WS*-Stack)

- Leads to tight coupling to domain in many cases (tools)
- No notion of resource
- No hypertext
- No caching

RESTful according to Roy Fielding

- Level 3 of Richardson's Maturity Model qualifies for the attribute RESTful
- Many services are called RESTful but do not adhere to all the constraints
- There is a lot of work on standards (RFC's) to be done

REST Richardson Maturity Model



REST principles

- Unique identification of resources
- Standard methods
- Representations
- Hypermedia
- Stateless communication
- HTTP is application protocol

Resources

- Primary Resources
  - Business Concepts (Customer, Address, Order, Account, Idea, Employee)
- Sub resources (x «contains» y)
- Other Resources
  - Query, Transaction, Concepts

Representations of Resources

- Resources may have many representations
- UI client might get HTML representation
- Machine might get XML or JSON

Resources Impedance mismatch

- Not every Entity is a resource in the API
- Representation of a resource may have data from many Entities (aggregation)
- Representations may contain only a subset of the attributes of an entity (projection)

URI's for resources

- Resource Collections
  - www.ex.com/customers
  - www.ex.com/customers/12345/orders
- Resources
  - www.ex.com/customers/12345
  - www.ex.com/customers/12345/orders/89

URI naming rules

- Use nouns not verbs
- Use plural
  - www.api.ex.com/customers/1234/orders
- Use singular only for singletons
  - www.api.ex.com/configuration

URI templates

- www.api.ex.com/customers/{Id}/orders
- www.api.ex.com/customers/{Id}/orders/{Id}
- Instruction on how to build an URI
- Should not be necessary if HATEOAS is implemented

Query parameters in URI's

- https://airline.server.test/ticketing_api/{flight_id}/{passenger_id}?option=vegetarian&option=wheelchair
- URL Limit 8092 bytes

Collections / Queries

- Use Query Parameters if simple
- Use concept of Query as Resource
  - first POST to create a query (/products/queries)
  - then you GET the query (/products/queries/1)
- Never use HTTP body in GET

Collections Pages / Pagination

- Provide a page-size parameter
- Provide the total page count in the response
- Use links for navigation (first, prev, next)
  - Api.ex.com/customers/pages/3

Representations – Media Types

- text/HTML
- application/json
- application/xml
- Your own type: application/vnd.xxxx+json
- Links to IANA MIME types
  - http://www.iana.org/assignments/media-types/media-types.xhtml

Media types contd.

- ATOM ASF syndication format
- ATOM PUB publishing protocol
- ODATA
- Collection+JSON
- HAL Hypermedia Application Language
- SIREN
- RDF Resource description framework

Selecting Media type

- Depends on the requirements
- Choose formats based on JSON or XML
- Plan for Hypermedia controls
- Type should allow for a link to a Profile
- Choose formats that have some pervasiveness, tool support etc.

HTTP Verbs

- GET
- PUT
- POST
- DELETE
- PATCH
- OPTIONS
- HEAD
- TRACE

HTTP verbs – safe methods

- GET, HEAD, OPTIONS
- No changes on the server

HTTP verbs - Idempotence

- Idem = same /potence = power
- operation that will produce the same results if executed once or multiple times
- GET, HEAD, OPTIONS, PUT, DELETE must have idempotent results
- POST is not idempotent

Overriding HTTP Method

- Some proxies support only **POST** and **GET** methods. To support a RESTful API with these limitations, the API needs a way to override the HTTP method.
- Use the custom HTTP Header **X-HTTP-Method-Override** to override the POST Method.

Error handling

- Do not just send 500 with a stacktrace
- Provide meaningful payload

```
{
  "errors": [
   {
    "userMessage": "Sorry, the requested resource does not exist",
    "internalMessage": "No customer found in the database",
    "code": 12349,
    "more info": "http://smx.cmfirstgroup.com/test/api/v1/errors/12349"
   }
  ]
}
```

cm FIRST
Rethink Modernization

HTTP status codes

- Organized in groups
- 1xx Informational
- 2xx Successful
- 3xx Redirection
- 4xx Client Error
- 5xx Server Error

HTTP Headers

- Authorization: Bearer 939399399399393
- X-HTTP-Method-Override: DELETE
- Accept: audio/*;
- Location: http://api.ex.com/customer/123
- X-Callback: <http://ex.com/callback>; method="post"
- See: RFC 2616

REST API Stateless server

- There is no concept of a session in REST
- State must be kept on the client
- All information needed to perform a request on the server must be passed along with the request (Headers, representation)
- Resource Status (ordered, shipped etc) is kept on the server

Resource caching

- *The advantage of adding cache constraints is that they have the potential to partially or completely eliminate some interactions, improving efficiency, scalability, and user-perceived performance by reducing the average latency of a series of interactions.*
- **Roy Fielding**

Caching

- **improve speed**, because we want to deliver fast content to our consumer
- **fault tolerance**, because we want our service to deliver content also when it encounters internal failures
- **scalability**, because the WWW scales to bilions of consumers through hypermedia documents and we just want to do the same thing
- **reduce server load**, because we don’t want our servers to compute without the need of it

Cache types

- Local Cache
- Proxy Cache
- Reverse Proxy
- Webserver

Cache Strategies

- Expiration Model
  - Max-Age / Expires Header
- Validation Model
  - Conditional GET: If-Modified-Since
  - Conditional GET: If-None-Match: ETAG

- Model depends on business requirements

Invalidating Cache

- PUT,POST,DELETE requests will invalidate the caches that are under your control.
- Remember you cannot control intermediate caches in request path
- DB changes not driven by the API will probably need to invalidate cache entries

HATEOAS

- Hypermedia as the Engine of Application State
- Provide Links and Verbs for state transitions

HATEOAS

- HTTP as Application Protocol
- Application domain protocol
  - Describe possible next actions on resources using links.
    - Link relation
    - URI
    - HTTP verb to use
    - Expected Media type

HATEOAS single entry point

GET ordermgr.ex.com

```
{
        "links": {
                "all" : "http://ordermgr.ex.com/orders",
                "processing" : "http://ordermgr.ex.com/orders?state=processing",
                "shipped" : "http://ordermgr.ex.com/orders?state=shipped",
                "cancellations: "http://ordermgr.ex.com/cancellations",
                "reports" : "http://ordermgr.ex.com/reports"
                }
}
```

HATEOAS single entry point

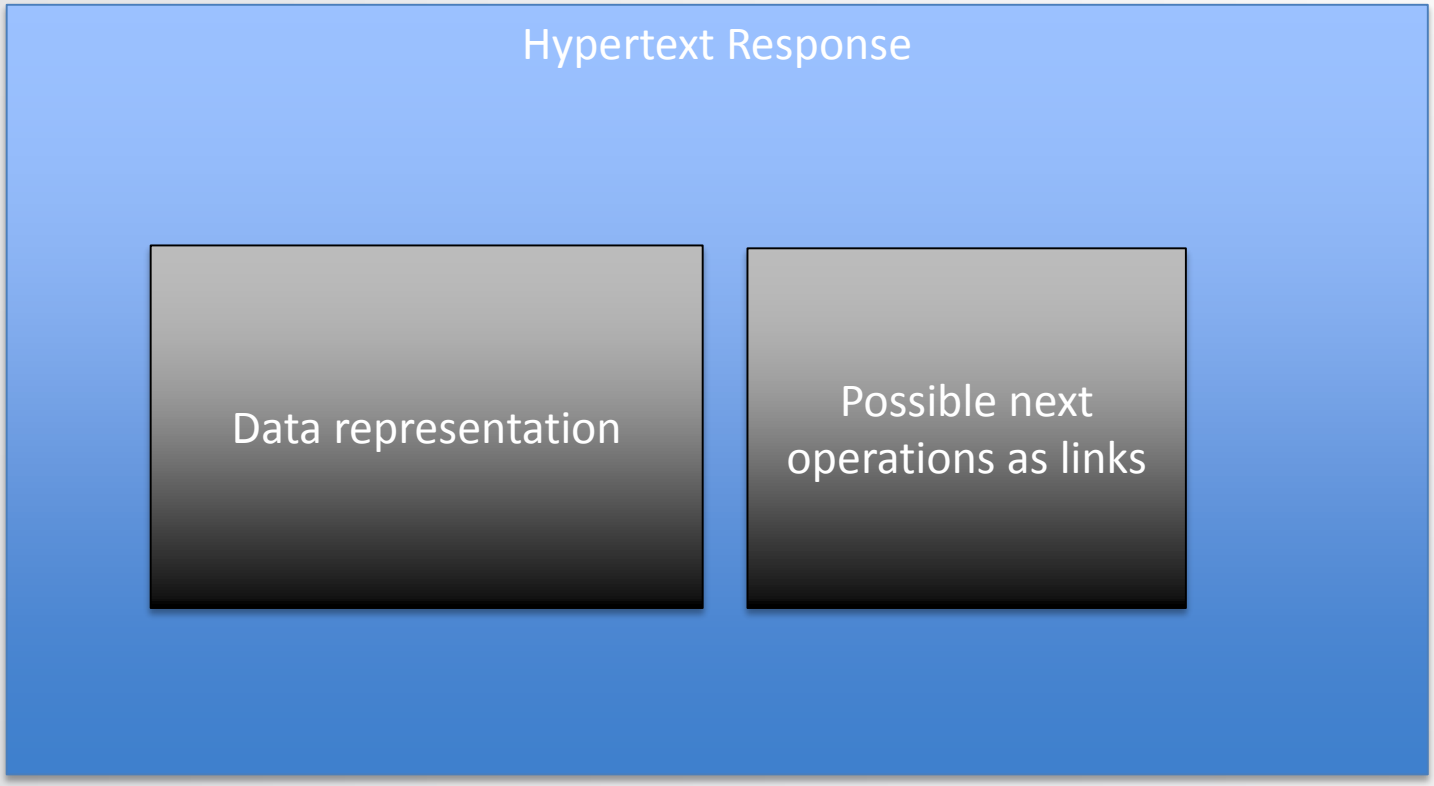
GET ordermgr.ex.com

```
{
      "links": {
            "all" : "http://ordermgr.ex.com/orders",
            "processing" : "http://ordermgr.ex.com/orders?state=processing",
            "shipped" : "http://shipmentmgr.ex.com/orders",
            "cancellations: "http://ordermgr.ex.com/cancellations",
            "reports" : "http://ordermgr.ex.com/reports"
            }
}
```

HATEOAS Benefits

- Decoupling of client and server
- Changes to resource distribution are simple (client just follows URI's)
- Server controllable application flow

Hypertext Request Handler



IANA Link relation values

- Self
- Next-archive
- Prev-archive
- First
- Last
- Payment
- http://www.iana.org/assignments/link-relations/link-relations.xhtml

Securing REST

- TLS / HTTPs
- Do not clutter URI's with auth information
- Use HTTP Headers

Securing REST

- Basic HTTP (only over TLS) 80 % case
- Oauth 2.0
- OpenId

Section 3 - CA Plex and RESTful API's

cm FIRST
Rethink Modernization

RESTful API and CA Plex

- No support for RESTful API generator
  - Should we generate ? Probably not.
- CA Gateway and ASO to make use of existing or new SOAP WCF services for CRUD is an option
- Third party solutions / no announcments yet

CA Plex Restful API – Resources vs. Entities

- Impedance mismatch
- Not every Entity is a resource in the API
- Representation of a resource may have data from many Entities (aggregation)
- Representations may contain only a subset of the attributes of an entity (projection)

CA Plex Restful API – Links vs. DB Relations

- We should not expose implementation details
- Some Relations will be modeled as links others will be modeled as sub-resources or embedded resources in the API

CA Plex – Restful API

- Plex architecture based on RPC style
- Major effort needed to support REST style
  - New Plex Object Types (Resource, Representation).
  - Exchange dynamic Messages/Documents instead of fixed function parameters,
  - HTTP support for Plex Runtime

CA Plex – third party web solutions

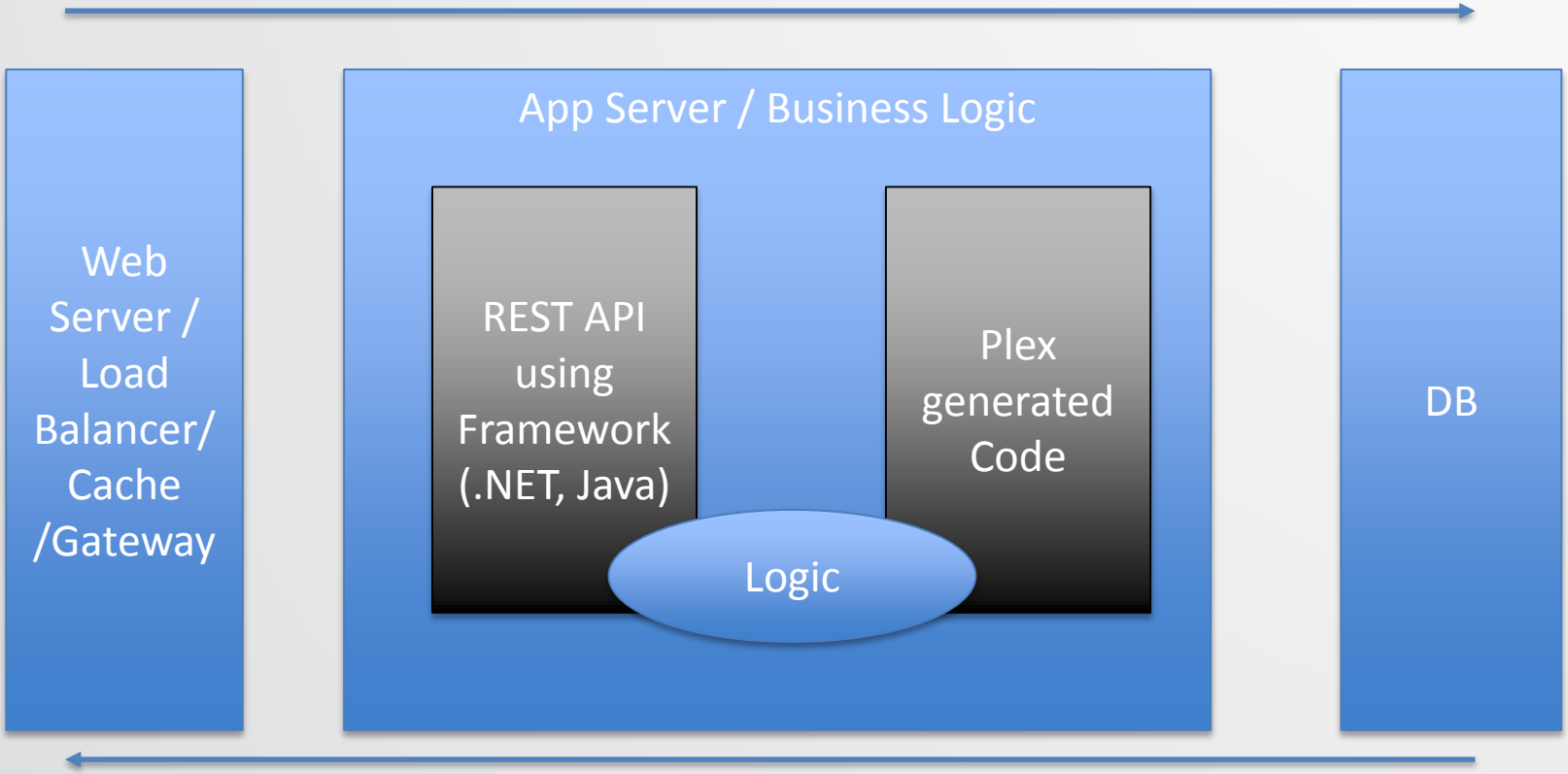
- Websydian Transact XML
- CM Webclient
- Plex XML

All of these can adopt REST style.

WCF SOAP API and CA Plex

- Create function layer for WCF Services
  Do not call domain functions directly
- Use separate fields on parameter interfaces for service functions, cast from domain fields
- WCF SOAP method generation based on Plex Function input/output parms
- MOV’s and arrays still fixed size.

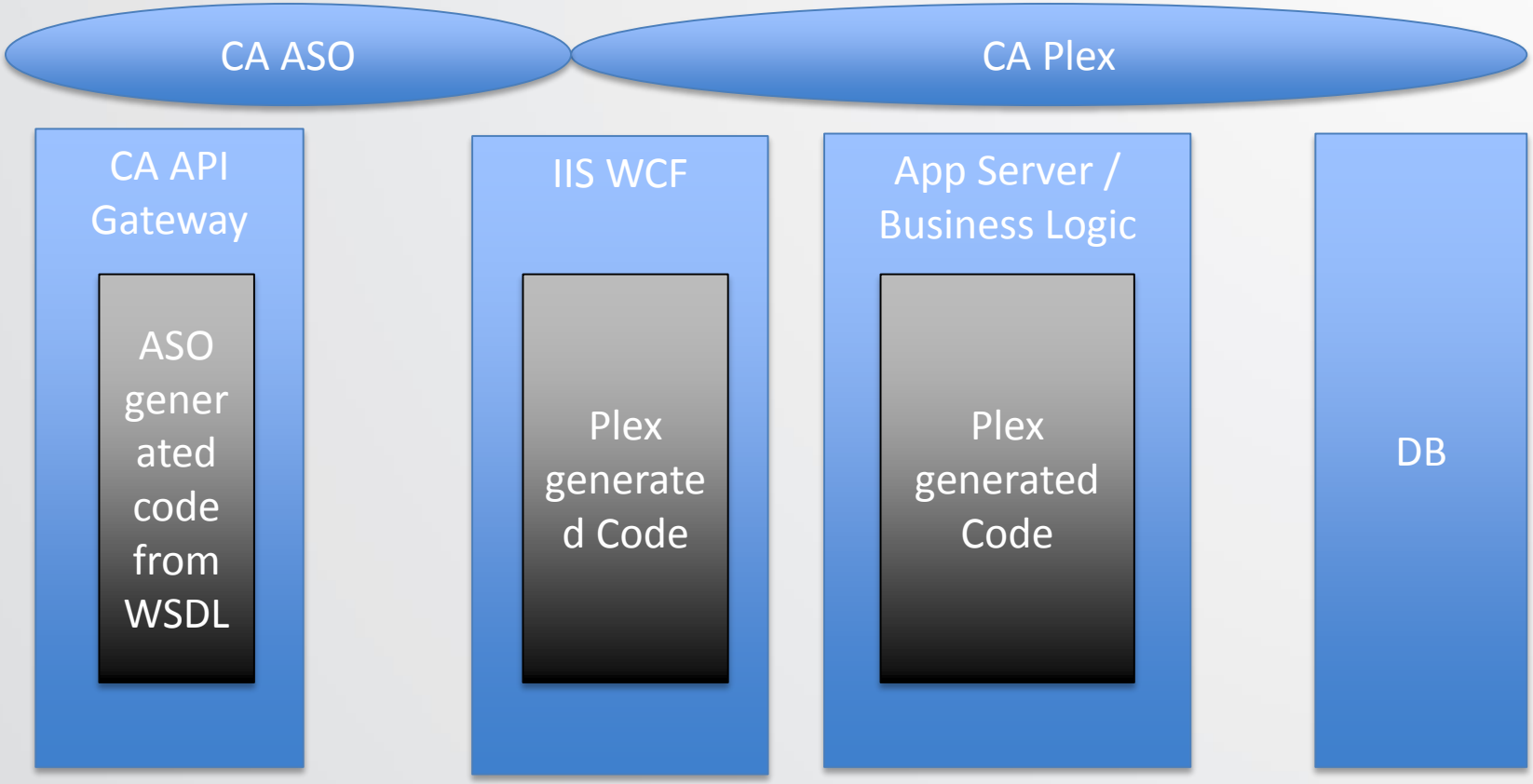
RESTful API and CA Plex



Write API Services in Java/.Net

- .NET WCF Restful Services
- Java JAX-RS
- Use PlexRuntime to call domain functions
- Direct Object to XML or JSON mapping should be avoided
- Resource Handlers, State Machines
- Decouple API model and Domain model

CA Plex and App Services Orchestrator



CM FIRST
Rethink Modernization

Application Logic

- Logic should be on the server
- CRUD only API will lead to «fat clients»
  - Validation / code duplication
  - Status Transitions

Datamodel supporting REST

- Use GUIDS on Tables
- Add a mapping table to the picture
  - GUID
  - Tablename
  - Key Values

Key mapping table

Bericht anzeigen

Anfang auf Zeile . . . . . +99999

Zeile ....+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8....+.

	ZGT_GUID	TBL_TabellenName	ZGT_PrimKeyAggregat
410453	0004ac182f7c-911d-19a7-053d-26781403	RIPOPP	<11><1018484><13><1>
410454	0004ac182f7c-911d-19a7-053d-267cf403	RIPOPP	<11><1018484><10><1>
410455	0004ac182f7c-911d-19a7-053d-26807803	RIPOPP	<11><1018484><20><1>
410456	0004ac182f7c-911d-19a7-053d-26881403	RIPPER	<11><3016388>
410457	0004ac182f7c-911d-19a7-053d-268a9403	RIPPEG	<11><3016388><1>
410458	0004ac182f7c-911d-19a7-053d-268e3403	RIPPRV	<11><1018484><3016388><1><1>
410459	0004ac182f7c-911d-19a7-053d-2696a803	RIPPER	<11><3016389>
410460	0004ac182f7c-911d-19a7-053d-26994403	RIPPEG	<11><3016389><1>

cm FIRST
Rethink Modernization

URI composition from ent and keys

- Mandate, Customer, Order, Orderline, Shipmentline
- Api.ex.com/mandates/{key}/customers/{key}/orders/{key}/orderlines/{key}/shipmentlines/{key}
- Api.ex.com/shipmentlines/{key}

Questions ?

cm FIRST
Rethink Modernization

Recommended REST reads

- RESTful Web API (Leonard Richardson, Mike Amundsen, Sam Ruby) ENG
- REST in Practice (Jim Webber,Savas Parastatidis,Ian Robinson) ENG
- REST und HTTP (Stefan Tilkov, Martin eigenbrodt, Silvia Schreier) GER

Thank you !

lorenz.alder@cmfirstgroup.com

cm FIRST
Rethink Modernization