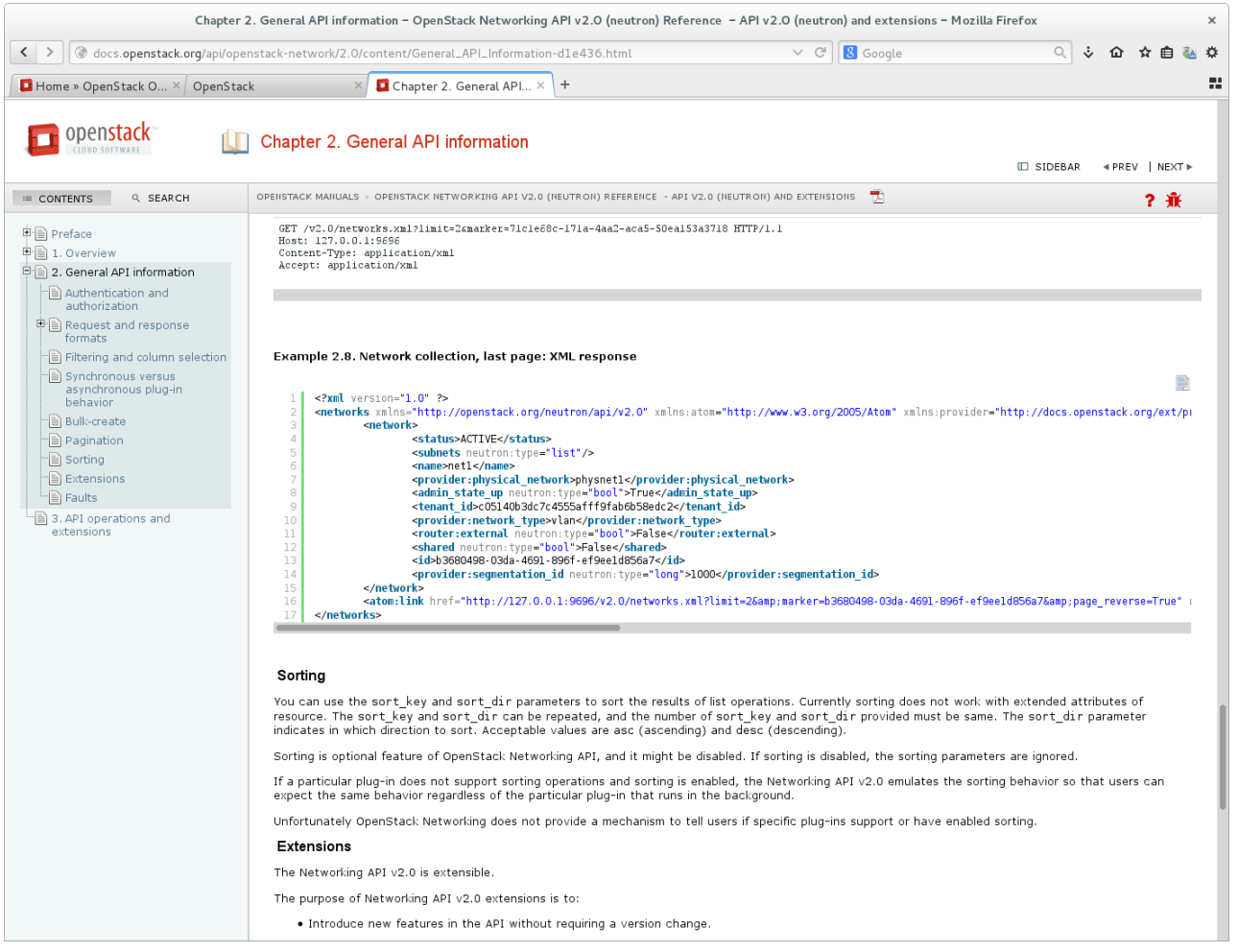
erocci

A scalable model-driven REST framework

Jean Parpaillon <jean.parpaillon@free.fr>

REST API : a user point of view

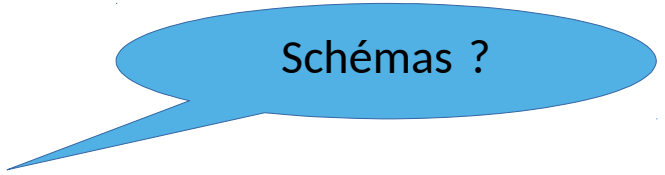


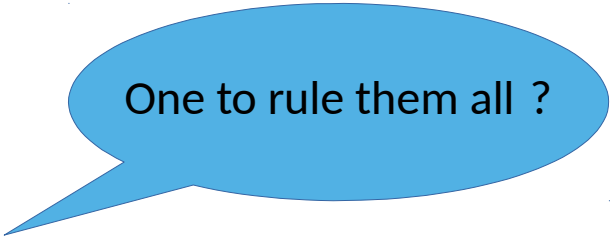
erocci REST API : a user point of view

- Data format
  - JSON format
  - XML format
  - Scalar data types (string, integer, float, etc)
- Data access protocol
  - Filtering
  - Sorting
  - Pagination
  - Authnz (tokens, etc), return codes, etc.
- Finally...
  - Application level : data types (structures, relations, etc)

erocci REST API : a user point of view

- Data format
  - JSON format
  - XML format
  - Scalar data types (string, integer, float, etc)
- Data access protocol
  - Filtering
  - Sorting
  - Pagination
  - Authnz (tokens, etc), return codes, etc.
- Finally...
  - Application level : data types (structures, relations, etc)






Let's speak about a standard ?

Let's speak about a standard for REST

- A standard
  - Is not a catalog of schemas
  - Is not a constraint for developers
  - Must be implemented (implementable)
- A good standard
  - Should allow to concentrate on the design, not the implementation details
  - Allow to build an ecosystem
  - Allow to accelerate the development, not the contrary

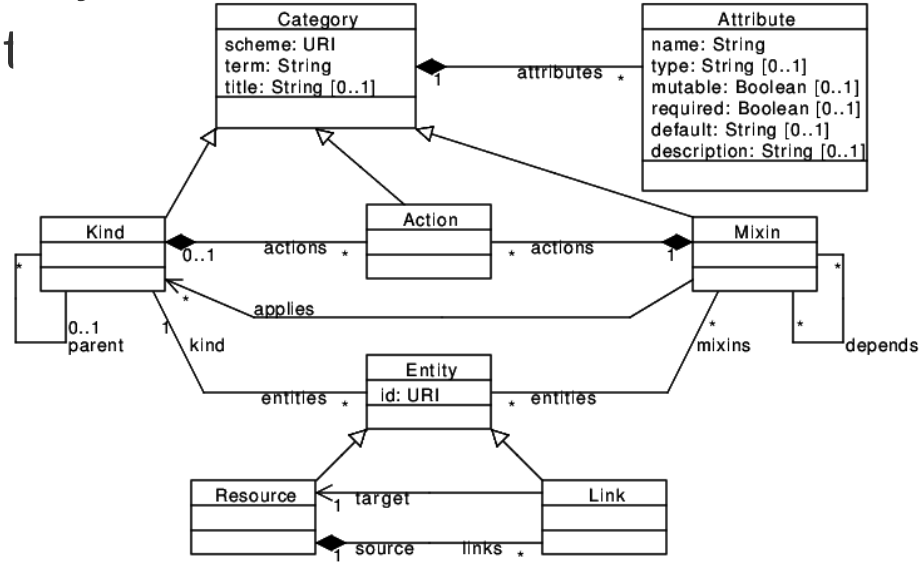
- OCCI is typed
  - A ressource is an instance of a kind
  - A kind is a named list of typed attributes and actions
    - e.g. : compute : # cores (integer), RAM (float), etc
  - Attribute : name, type, default value, mutability, etc.
  - Action : an invocable operation on a ressource
  - Kinds are inheritable

- OCCI is extensible
  - Resource can be associated with mixins
  - Mixin: a named set of additional attributes and actions
    - e.g. : IPNetworkInterface adds IP, netmask, etc to a network interface
  - User Mixin : aka « tags »
    - e.g. : http://example.com/occi/mixins#my_project1

- OCCI is relational
  - Links are ressources with additional attributes
    - occi.core.source : resource URI
    - occi.core.target : URI
  - A bounded collection is a list of ressources of the same kind/mixin :
    - e.g. : GET /collections/compute/
  - An unbounded collection is a list of ressources with same prefix :
    - e.g. : GET /myresources/

- OCCI is self-described
  - GET /-/ (capabilities)
    - List of supported kinds
    - List of supported mixins
  - Capabilities returns collection's URL
    - http://schemas.ogf.org/occi/infrastructure#compute → /collections/compute/

- OCCI is meta-model based
  - Model consistency checking
  - Automatic model implementation
  - Rendering independant
    - text/plain, application/json, ...
  - Transport independant
    - HTTP, ...



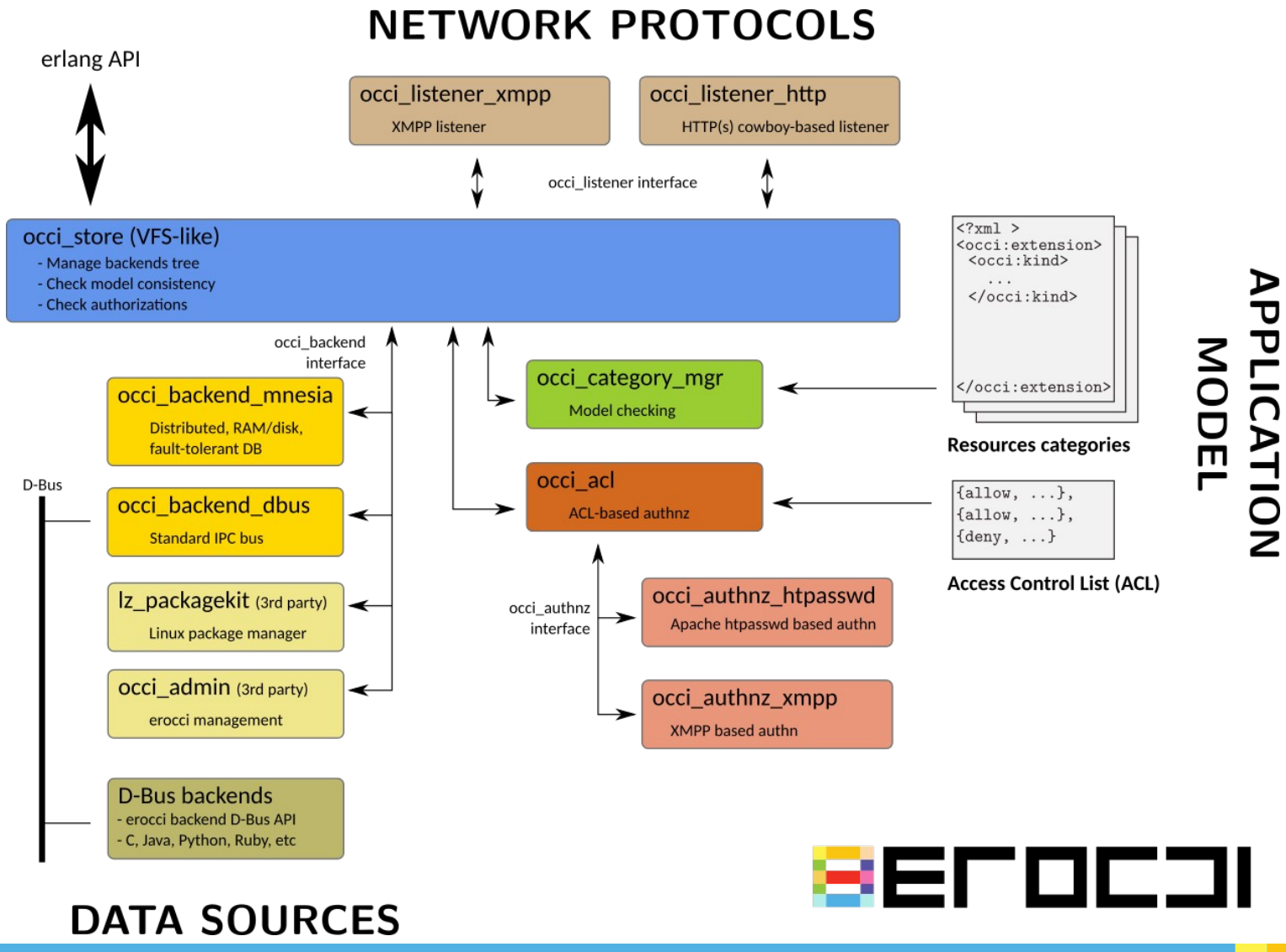
erocci OCCI Ecosystem

- Framework
  - rOCCI (ruby)
  - occi4java (not maintained)
- Specific implementations
  - CompatibleOne
  - PyOCNI
  - OpenStack, OpenNebula, etc
- Tools
  - DoYouSpeakOcci : tests
  - Monitoring (Intel)
- Limits
  - Models are language/extensions/lib dependant
  - See http://occi-wg/community/implementations/

erocci Why erocci ?

- Generic : a framework to build OCCI applications
- Less code, more design
- Small, tested, reliable
- Easy to deploy
- Scalable
  - Runs embedded in connected objects (IoT)
  - Scales out to serve millions of users
- Extensible

erocci Architecture



erocci Genericity

- Only OCCI meta-model is hard-coded
- Everything is pluggable :
  - Authentication mechanism
  - Transport protocol (HTTP, XMPP, etc)
  - Data sources (backends)
- Allow composition of services by plugging data sources (backends) at runtime

Less code, more design

- A schema for ressources' categories (kinds, mixins)
- ACL-based authorizations

erocci Small, tested, reliable

- Small
  - Around 12k lines
- Tested
  - Continuous integration
- Reliable
  - Erlang cowboy web server (LeoFS, MongooseIM, etc)
  - erlang/OTP platform provides fault tolerance
  - Process supervision

erocci Easy to deploy

- All included erlang/OTP platform
- Self contained : no web frontend
- Just try it !

erocci Scalable

- Thanks to erlang/OTP
  - Runs on RaspberryPI (~ 100 req/sec)
  - Scales out to a full cluster

erocci Extensible

- Pluggable mechanisms
- For those who don't like erlang
  - D-Bus backend API (experimental)

erocci (1) use case : a REST backend for blog

- No need for ORM, web server, database, templating, etc.
- Design your model
  - Blog entry kind
  - User kind
  - Contribution kind (link between a user and an entry)
- Choose a storage backend
- Choose a transport (HTTP?)
- Defines ACL
- And that's all

erocci (2) use case : API adapter

- Backend API : write a new data source for your API
  - Erlang API
  - D-Bus API (any language)
  - In any case, implements CRUD operations
- Choose a transport
- Enjoy scalability, supervision, transports, ACL, etc.

erocci (3) use case : a protocol adapter

- You have a resource oriented protocol and want to expose it through HTTP/OCCI
- Use occi_store for storing your data
- erocci deals with storage, rendering, scalability, ...

erocci Roadmap

- Authn/authz
  - x509 based authn
  - oauth
- Testing
  - Improve functional tests
  - Improve unit testing
- occi_store
  - Optimizations
  - Integrate authnz, pagination (wip)
- XMPP
  - Enable pubsub (notifications)
- New storage backends
  - Riak (wip)
  - ODBC (SQL)
  - ...

erocci Eco-system : OCCIware

- (1) consortium
  - Research, industry, standardization
- (3) pillars
  - Theory : proven meta-model (OCCI)
  - Design : Eclipse-based tools
  - Runtime : erocci, ...
- (*) use cases
  - IaaS, PaaS, SaaS, IoT, ...
  - Every RESTful services

EROCCI Resources

- Website (WIP)
  - http://erocci.ow2.org/
- Source code
  - http://github.com/erocci/erocci
- Continuous integration
  - https://travis-ci.org/jeanparpaillon/erocci
- Mailing lists
  - erocci-dev@ow2.org / erocci-info@ow2.org

erocci Thanks !

Questions ?