Yenca (A netconf prototype)

www.madynes.org/software.html

Radu State, Oliver Festor, Benjamin Zores

MADYNES

The MADYNES Research Team
LORIA – INRIA Lorraine
615, rue du Jardin Botanique
54602 Villers-lès-Nancy
France
Radu.State@loria.fr

Outline

- Motivations
- NetConf Overview
- Yenca - Architecture
- Conclusions

Motivations

- Practical experience with a NetConf prototype
  - Usage experience
  - Development experience
- Open source distribution available for experiments to a large community
  - Interoperability studies
  - SNMP/NetConf gateways
  - Protocol optmisations

Problems (we are aware of)

- Further Evolutions of NetConf which might be incompatible with our existing implementation
- No Data Model for the management information
- No complete specification, no complete implementation either…

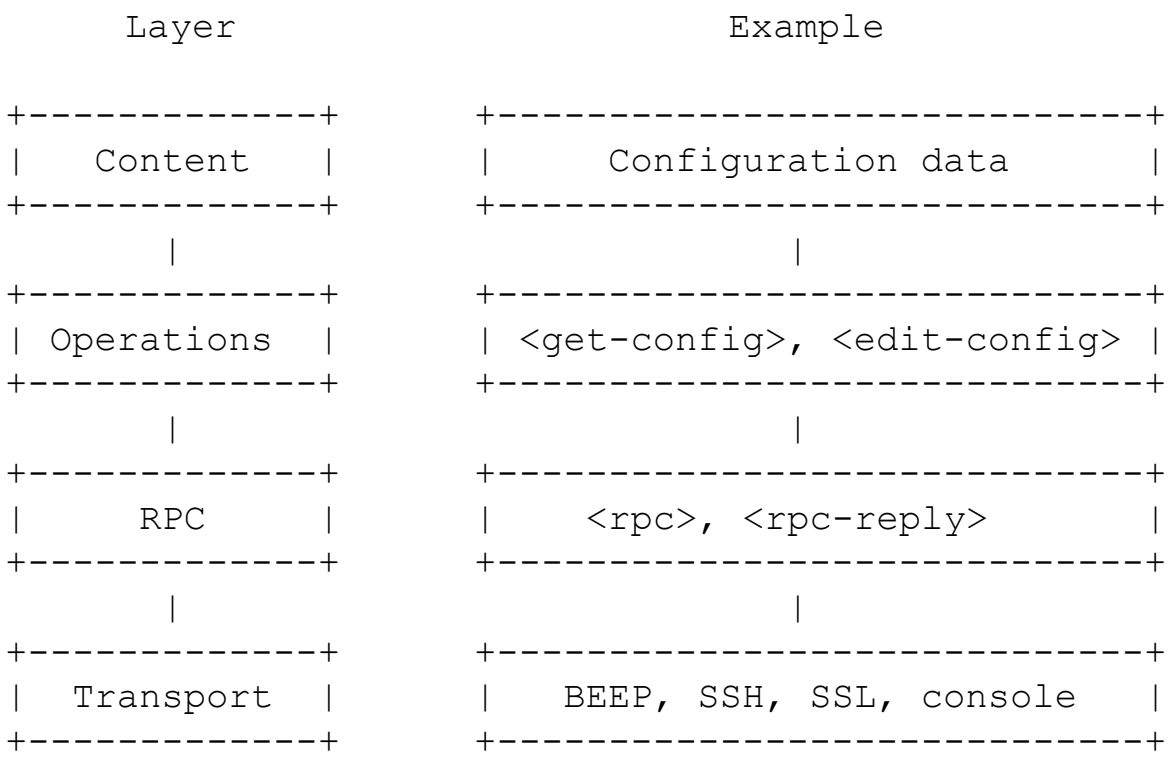
- Our dilemma : wait until everything will be complete (and maybe obsolete) or just try it out ?

NetConf Overview

- <rpc> element

```
<rpc message-id="101">
  <some-method>
   ...
   </some-method>
</rpc>
```

- <rpc-reply> element
- <rpc-error> element



- Protocol for network management.
- Remote procedure calls (RPC) to define a formal API for the network device.
- XML-based data encoding.
- Managers can discover the set of capabilities supported by the agent.

Requirements specification of Yenca

- Implement a subset of the NetConf protocol most dissimilar to existing management protocols
  - Configuration locking with respect to netconf.
  - Configuration locking with respect to other management actions (SNMP/CLI)
  - Configuration restoration
  - Dynamic discovery of agent capabilities
- Modular framework where additional components can be plugged in easily:
  - Device specific code
  - Agent extensions
- Deployment
  - Linux target environment
  - IPv6 enabled
  - OpenSSL support

Protocol Operations

Base operations

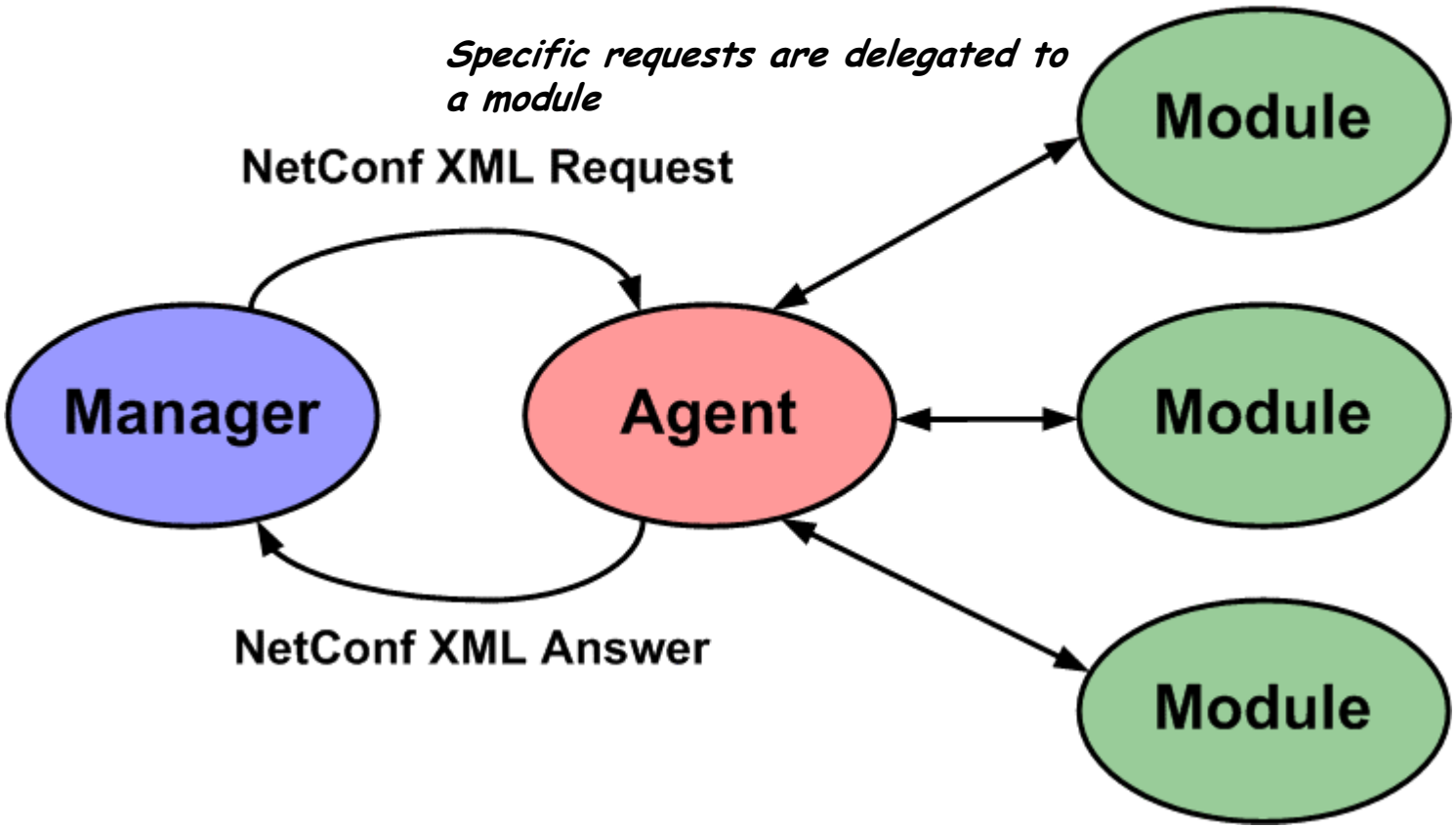
- <get-config>
- <edit-config>
- <copy-config>
- <delete-config>

Additional operations

- <get-state>
- <kill-session>
- <commit>
- <discard-changes>
- <lock>
- <unlock>

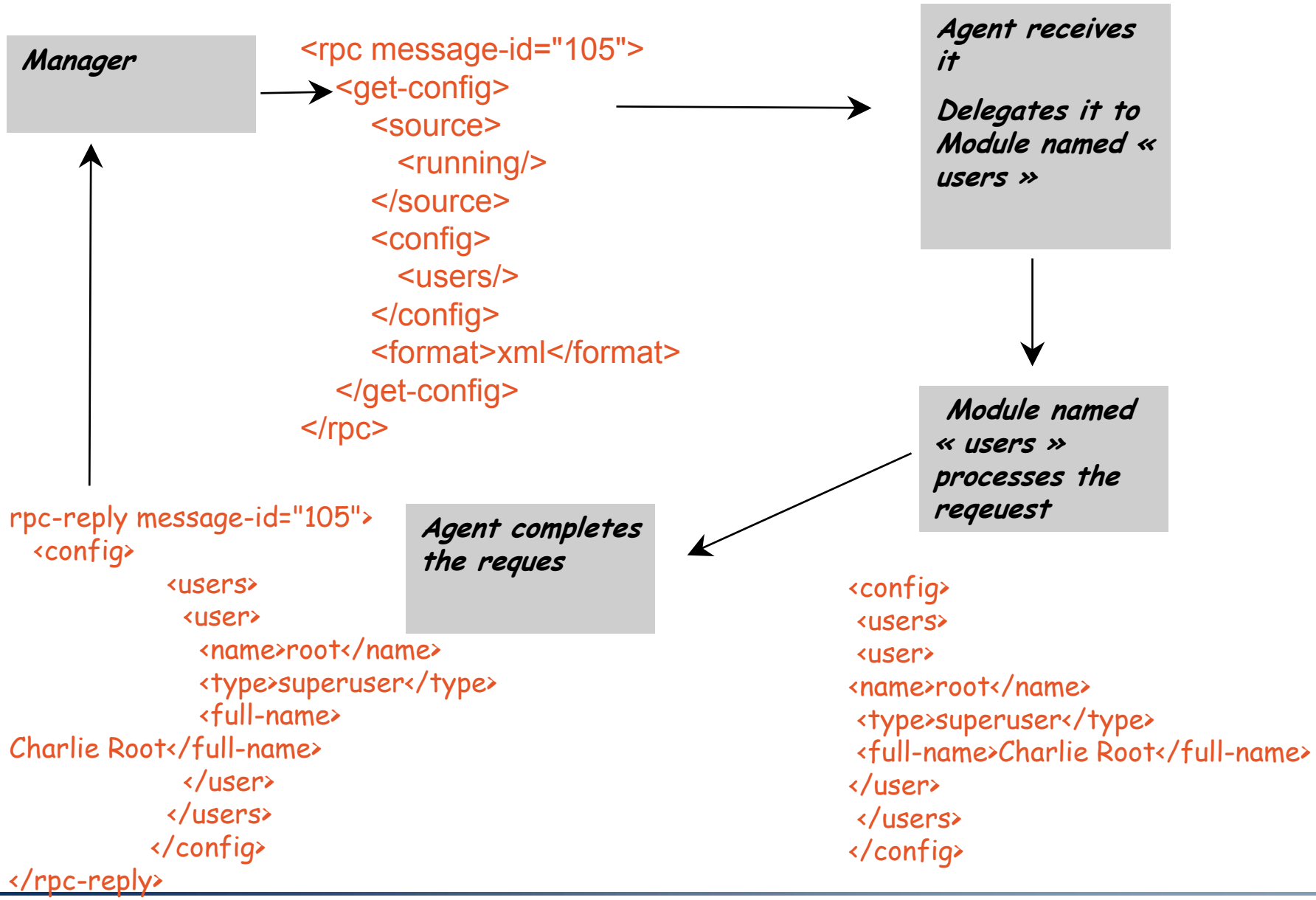
Yenca architecture

Three tiers architecture



Modules subscribe to the agent claiming responsability for a XML subtree

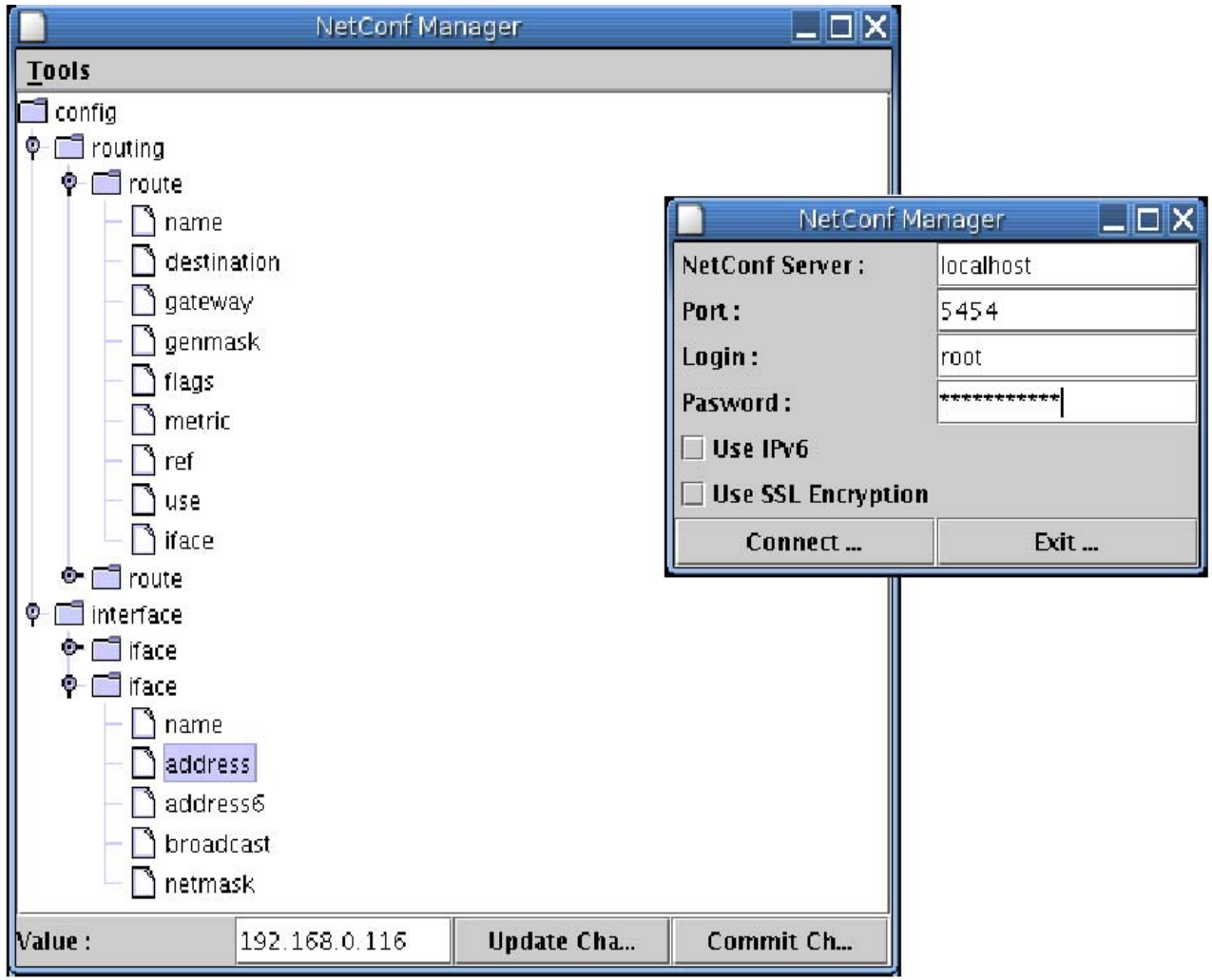
Example <get-config> operation



NetConf Manager

Initial request :

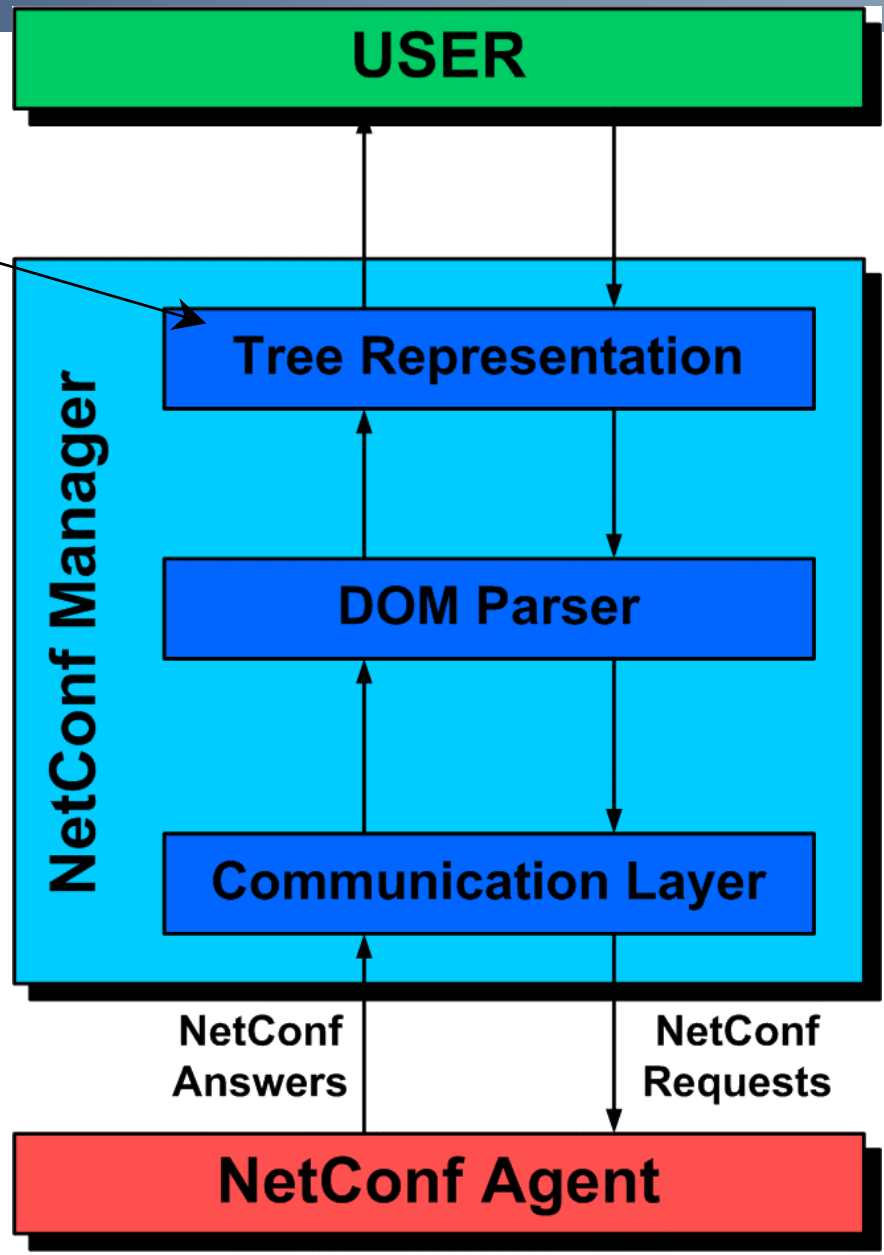
```
<rpc message-id="1">
        <get-config>
            <config>
                <all/>
            </config>
        </get-config>
    </rpc>
```



Dynamic GUI construction/simplified XML editor

JAVA application
support for IPv6 and SSL communications

Manager architecture

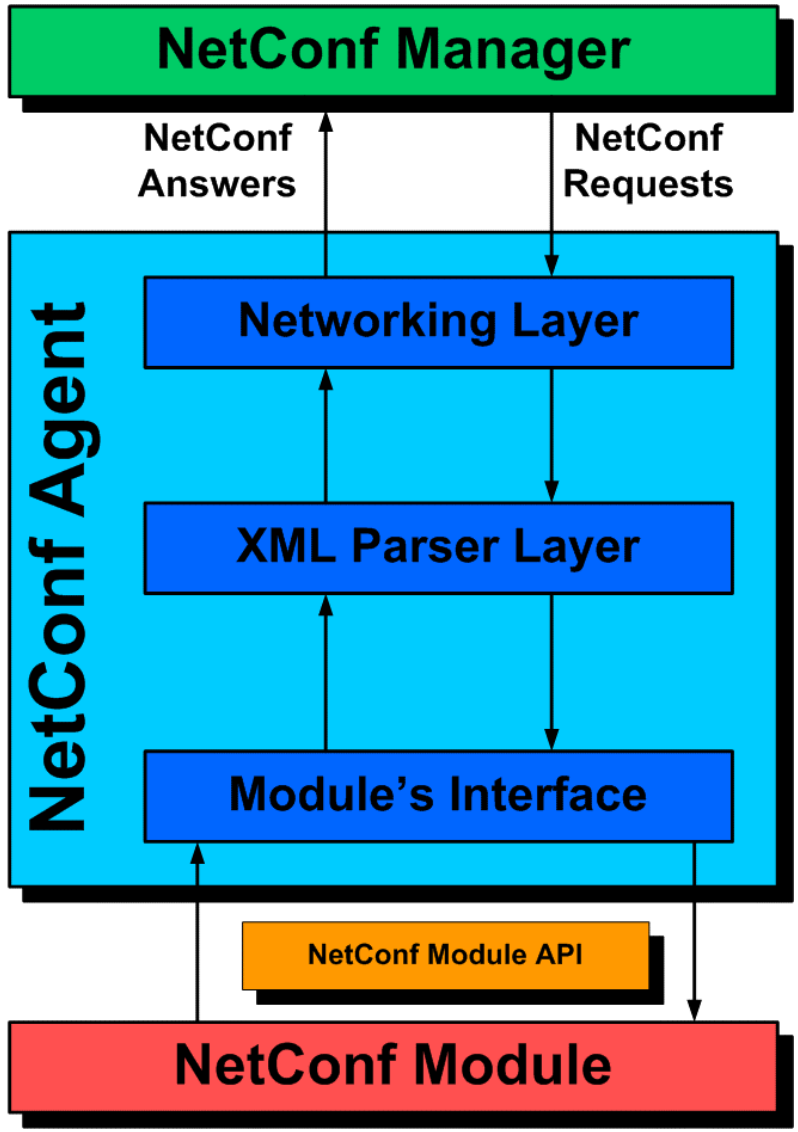


Interface between agent and end-user

No real NetConf protocol interpretation

Agent architecture

- Processes NetConf requests from the manager and associated results from modules
- Forward requests to specific modules
- Forwarding is defined by a simple mapping module-name to XML tag
- Uses pre-defined generic module API



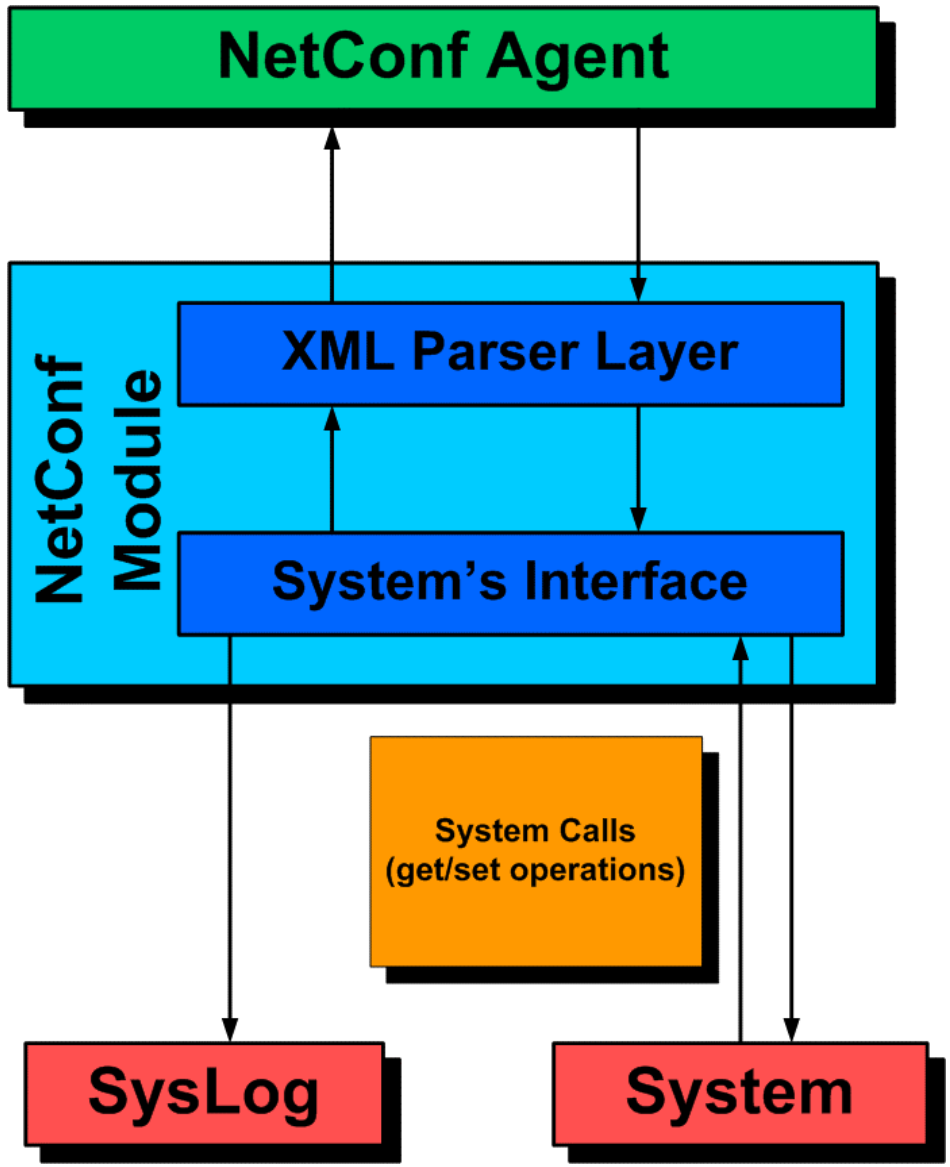
NetConf : XML parser layer

- parses XML messages.
- requests are stored in a generic structure.
- Performs module subscription and XML-tag to module name mapping
- forwards informations to dedicated modules.
- generates XML-based answers

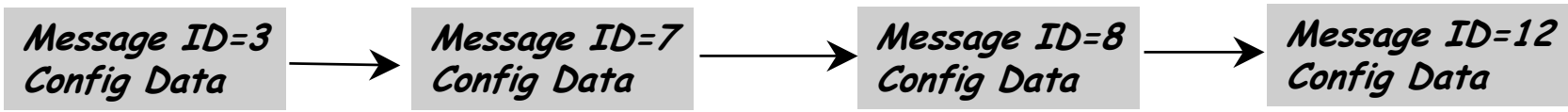
```
typedef struct {
  char *id;
  char *action;
  char *method;
  char *source;
  char *target;
  char *config;
  char *format;
} netconf_request_t;
```

Module architecture

- Interface between agent and real system
- MO specific implementation (eg. System calls..)
- SysLog mechanisms
- Implemented as shard libs in */usr/lib/netconf/*)
- Discovered dynamically by the agent



Module architecture (contd)



Restoring a particular configuration :

History of configuration requests implemented as a linked list associated to messages identifiers in order to restore a particular configuration

NetConf : Agent view on a module

- Generic way to access modules
- Agent can perform system calls without explicit declarations

```
typedef struct {
  char *name;                          /* Module's Name */
  char *desc;                          /* Description */
  void *prop;                          /* Module's properties */
  int nr;                              /* Number of module's descriptors */
  notify_f notify;                     /* Module's notify() specific function */
  list_f list;                         /* Module's list_properties() function */
  savecfg_f savecfg;                   /* Module's save_cfg() specific function */
  restcfg_f restcfg;                   /* Module's rest_cfg() specific function */
  get_request_f get_request;                    /* Module's get_request() specific
     function */
} net_module_t;
```

Modules interface

Module specific implementation of these functions are called whenever reqquests are delegated by the agent

/ Public Functions */*

- net_module_t *register_module (void);

/ Private Functions */*

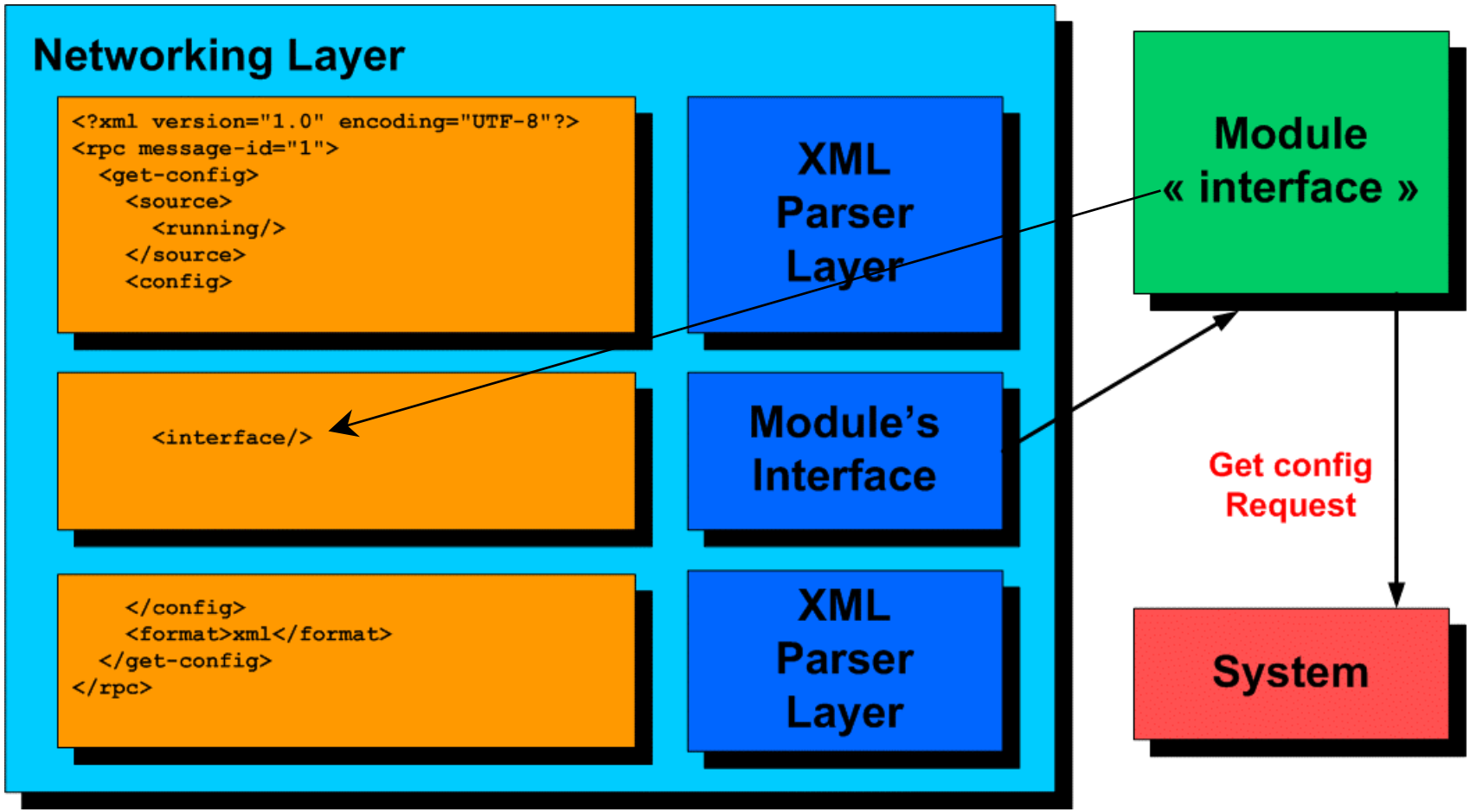
- int getcfg (void *iface);
- int setcfg (char *iface, char *prop, char *value);
- int notify (void);
- int list_properties (char *iface);
- int save_cfg (char *id);
- int rest_cfg (char *id);
- int get_request (xmlNodePtr tree, netconf_request_t *req, char *msg);

Example : interface module

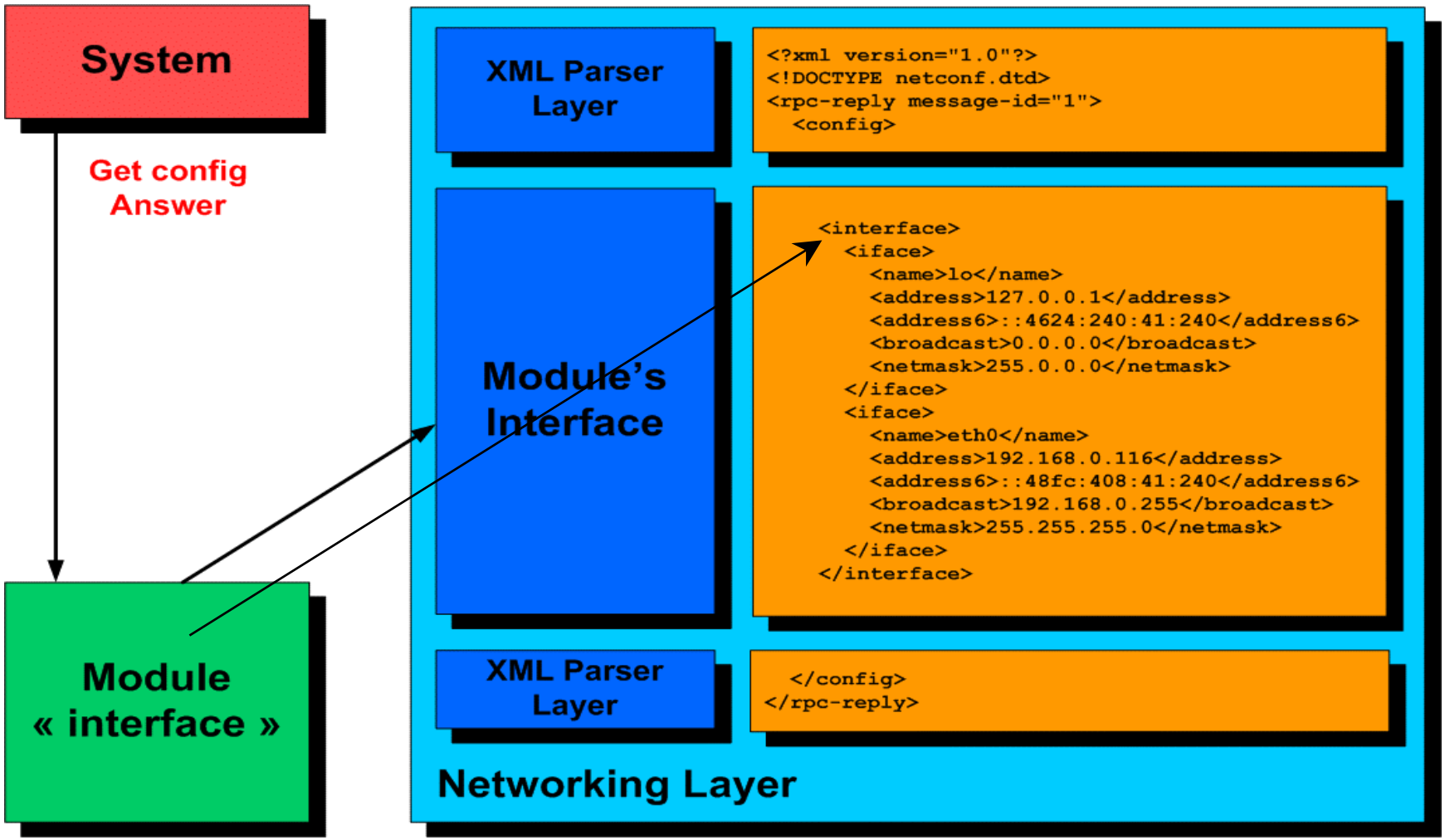
- Each module has a structure defining all available properties (Ex: the routing properties table) :

```
typedef struct {
  char name[RTG_INFO_LEN];                /* Name (int) */
  char dest[RTG_INFO_LEN];                /* Destination */
  char gw[RTG_INFO_LEN];                  /* Gateway */
  char mask[RTG_INFO_LEN];                /* Genmask */
  char flag[RTG_INFO_LEN];                /* Flags */
  int metric;                             /* Metric */
  int ref;                                /* Ref */
  int use;                                /* Use */
  char iface[RTG_INFO_LEN];               /* Interface */
} route_prop_t;
```

Receiving an XML request



Associated XML answer



Current Limitations and future work

- Early prototype with inherent shortcomings (only running config is supported)
- Mapping from XML tag to a module name is ad-hoc based on string comparison (no data-model ..)
  - Example <interfaces> tag in XML is delegated to a module called « interfaces »
- A module can only require responsability for one global subtree (no additional sub-delegation is possible)
  - For instance if you have 2 interfaces (Cisco+Intel) you must have one module responsible for both of them
  - This will be addressed in a future release (XPATH usage)
- Each module is responsible to parse XML…
- Locking with respect to other management frameworks is not yet bug-free (thus not included in the current release)
  - Bug in a kernel module..
  - We are working on it.