Seventh FRAMEWORK PROGRAMME
FP7-ICT-2007-2 - ICT-2007-1.6
New Paradigms and Experimental Facilities

SPECIFIC TARGETED RESEARCH OR INNOVATION PROJECT

Deliverable D2.2
"Cognitive Engine - Experimental Low-Level Design"

Project description
Project acronym: **ECODE** Project full title: **Experimental Cognitive Distributed Engine** Grant Agreement no.: **223936**
Document Properties
Number: FP7-ICT-2007-2-1.6-223936-D2.2 Title: Cognitive Engine - Experimental Low-Level Design Responsible: M.Jakeman Editor(s): M.Jakeman Contributor(s): M.Jakeman, L.Mathy Dissemination level: Public (PU) Date of preparation: ***October 31, 2010*** Version: 1.0

D2.2 - Cognitive Engine - Experimental Low-Level Design

Executive Summary

This document is the ECODE deliverable D2.2. It provides an insight into the development plan for the Ecode Unified Architecture (EUA). The EUA is a software based router platform. As a foundation the EUA uses the eXtensible Open source Routing Platform (XORP) and builds upon the functionality it offers to enable Machine Learnine Engines (MLE's) to operate within a router. XORP provides a software platform that is used to turn regular PC's running Linux into a router. It provides mechanisms to implement extensions to the router known as XORP processes. These processes can communicate with the rest of the routing platform. The EUA will be implemented in the form of a number of XORP processes.

In this document we detail the development plan for the EUA along with specific details regarding the API that will be produced. The EUA will provide an API that exports methods which can be used to implement MLE's within a router. A new XORP process known as the Translation and Communication Interface (TCI) is to be developed. This in turn will allow the creation of modules known within the EUA as Machine Learning Processes (MLP's) and Monitoring Points (MP's), within a router. An MP is an XORP process that conforms to the EUA specification and performs some sort of monitoring within the router. An MLP is also an XORP process conforming to the EUA specification but is concerned with executing the machine learning algorithms based on the information the MP's retrieve. MLP's are able to relay messages to MP's via the TCI. This process is an important aspect of the EUA and is known as dispatching. By having the TCI as an intermediary point between MLP's and MP's it is possible to monitor what actions MLP's are attempting to take within the network and mediate these actions if necessary. The TCI will also be able to communicate with TCI's on remote routers and therefore dispatch messages between MLP's and MP's that do not reside on the same router.

Once the EUA has reached a sufficient stage of implementation it will be tested by implementing a number of the ECODE use cases within the new framework. These uses cases will be implemented by creating new MLP's and MP's that can communicate via the TCI. The implementation of these use cases will demonstrate how the EUA can be utilised to realise various Machine Learning solutions. At present the first phase of the planned development is underway with completion expected by January 2011. With the first phase complete MLP's will be able to dispatch method calls to MP's as well as the routing and forwarding elements within XORP. From here the development will focus on communication between TCI's on remote routers to enable the dispatching of calls remotely.

This document is organised as follows. Section 1 provides an introduction to some fundamental components that will be utilised as well as an abstract overview of the EUA design. Section 2 gives an overview of the TCI. Section 3 details the internal communication interfaces of the EUA. Section 4 details the external communication interfaces. Section 5 gives an overview of the development plan. Sec. 6 describes two use cases that will be implemented using the EUA. Section 7 concludes the document and Sec. 8 is an acronym reference section.

List of authors

Affiliation	**Author**
ULANC	Laurent Mathy
ULANC	Matthew Jakeman

Table of Contents

1. Introduction

The aim of the ECODE project is to develop, implement and validate experimentally a cognitive routing system. This system will use cognitive techniques to provide a routing system able to meet the future Internet challenges.

In practice, this means that routers must be enhanced to support the cognitive system by running cognitive algorithms, mainly in the form of machine learning. From a router design perspective, the traditional dual plane architecture must then be extended to incorporate a "cognitive" plane hosting the cognitive functionalities. Indeed, routers have traditionally been designed to follow the natural separation of concern that exists between functions that must be applied to the data packets and those that apply to the behaviour of the network as a whole. The resulting fast/forwarding plane and slow/control plane separation bear the wholemarks of being designed to operate at vastly different timescales: the design of the forwarding plane is driven by the overriding need for performance, often sacrificing flexibility for the increased efficiency of hardware implementations; while the control plane exhibits the characteristics of a more general computing platform, capable of supporting a wide range of complex control functions (e.g. routing, management, etc). It is therefore not surprising that, even when provided as a monolithic router platform, the forwarding and control planes are generally supported on separated sub-systems.

In ECODE, the goal is to introduce machine learning techniques to tune and enhance the control plane of routers: there is no intention to push machine learning into the fast plane, because a machine learning algorithm that would have to touch every data packet would be faced with unrealistically challenging performance requirements. Compared to the tasks generally found in the control plane of a router, machine learning-based algorithms are likely to be more complex and deal with much larger input data, resulting in more computationally demanding components. This is why we see the need to logically separate the machine learning subsystem into its own cognitive plane, as this will more readily support the likely requirement of physically distributed components in order to meet the practical computational needs of the overall router system, keeping in line with the already widespread separation of the fast and control planes in traditional routers.

Beyond the disparity of computing requirements between the traditional control components and the new cognitive components (a.k.a. Machine Learning Engines, or MLEs) of a cognitive router, another consideration must be taken into account: each cognitive component introduces a potentially independent "control loop" controlling the work, and possibly modifying the data, of the control components. One of the benefits such cognitive components afford is the great flexibility in introducing new behaviours in an otherwise difficult-to-evolve environment, but such benefit can only be achieved if as little restriction as possible is imposed on the cognitive components. This means that a same control component, and its associated data structures, could well come under the supervision of several cognitive components resulting in several, concurrent independent control loops. Such a situation could in turn result in well known system stability issue, triggered by several such control loops reacting to each other's actions in an uncontrollable way. One way to address such potentially devastating issue, is to interpose an orchestrating entity between the various control components and the cognitive components that control them, where conflict resolution can be exerted via a series of appropriate policing techniques.

Another consideration is that the need for communications between various cognitive components may arise, because these components either implement a distributed cognitive system with components on separate routers, or offer services to other (local) cognitive components. In this context, it should be noted that while two cognitive components that operate on the control plane of a same router are logically local to that router, they may well be running in physically separated locations, as each router can be implemented in a distributed fashion, with not only cognitive components, but also control and forwarding plane components residing in potentially remote hardware systems. To simplify component design and deployment, such implementation considerations must be hidden from the components themselves. The "central" entity described above for policing purposes can also be used to provide the abstraction needed to provide seamless distributed router implementation, supporting both intra-router and inter-router component communications. Hence why this entity is named "Translation and Communications Interface" (TCI), and will provide seamless communications between components in all three router planes. Examples of such communications include, but are not limited to, communications between measurement points (MPs) running on the fast plane and cognitive components using such measurement or communications between control data structures (e.g. routing information bases) located in the control plane and cognitive components.

ECODE is an experimentally-driven project. In order to achieve the evaluation and demonstration goals set in the project, a working prototype of a cognitive router must be built. It therefore makes sense to extend an existing traditional router system as opposed to build a complete cognitive router from scratch. Because of its inherent extensibility, we choose to extend the XORP router system. Indeed, XORP provides and extensible control plane system that exhibits already codified inter-component communications based on XML in the form of XRLs. Such characteristics provide an ideal base for the construction of an extended cognitive router and the following ECODE router architecture will be described in the context of the XORP router architecture.

In this context, a couple of observations are worth making: first, a cognitive component (MLE) is simply a XORP process (known on its own as the Machine Learning Process, MP) that just happens to be running some machine-learning algorithms in conjunction with the TCI process. From the perspective of XORP, there is no "structural" difference between these processes and any other XORP process -- the MLPs will interact with the rest of the system by exchanging XRLs. Second, unless the TCI is implemented as an integral part of the XORP finder (see below), there is actually no way to force MLPs to communicate with the rest of the system via the TCI. For the sake of portability to newer versions of XORP, as well as applicability to other router systems, and because the TCI is a logical architectural component, we argue that the TCI should be realized as a stand-alone component that is optionally used by the rest of the system. In the context of XORP, this means that the TCI is implemented as a XORP process, and that MLPs must explicitly choose to use the TCI to benefit from the abstractions it provides and have their work regulated by it. This clearly puts the decision of forcing, or not, all cognitive processes through the TCI on the administrator of the router, by letting him/her choose to only deploy, or not, cognitive components that use the TCI. We believe that this provides the best level of flexibility to network administrators who will then retain complete control of their operating policies.

1.1 XORP Implementation

In order to provide the functionality required by MLP's a good underlying platform needs to be used. For this purpose we have chosen the eXtensible Open Router Platform (XORP). XORP is an open source project that implements the full functionality of a routing platform. It can be run on a number of Operating Systems and because of this it offers the perfect underlying infrastructure for the ECODE Unified Architecture (EUA).

XORP offers a stable base for the implementation of the EUA to be based upon. The word eXtensible in the title shows that the platform has been designed from the outset to be extended by developers and researchers to fulfill their own needs. XORP is a multi-process architecture. It has one process per routing protocol along with a number of process for management, configuration, and co-ordination. Communication between these processes is enabled by XORP Resource Locator's (XRL's) which are described in more detail in Section 1.1.1.

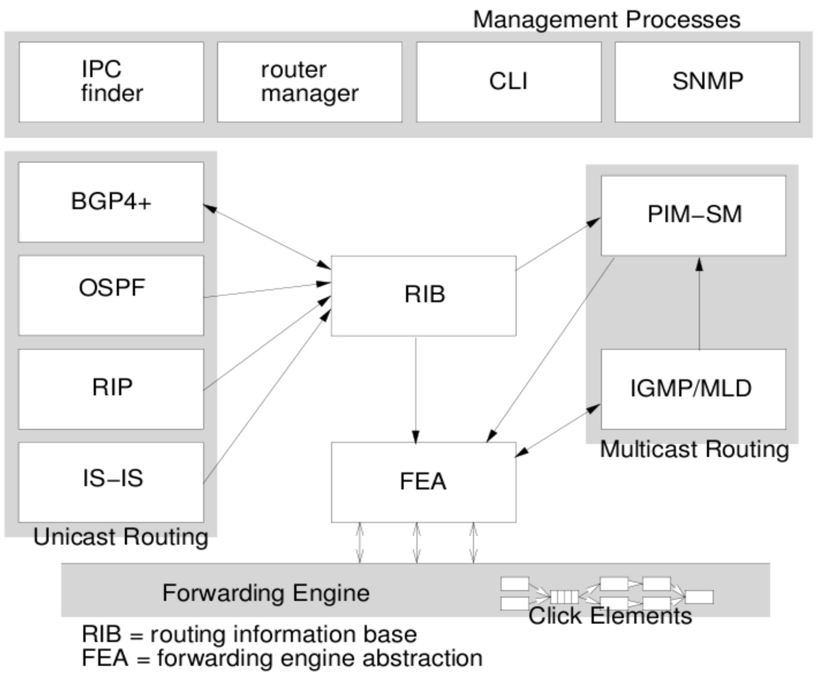

Figure 1: XORP Overview

Figure 1.1 shows an overview of the XORP architecture. Each box in the figure represents a different XORP process. A number of these processes are worthy of explanation in reference to the EUA.

The RIB is the Routing Information Base. This process is responsible for communicating with the individual routing protocol processes as well as setting information in the FEA (Forwarding Engine Abstraction). The RIB is used for setting and maintaining entries in the forwarding tables of specific protocols as well as setting entries in the forwarding table via the FEA. The FEA abstracts details of the underlying forwarding platform, whatever that may be. It could be a Click based forwarder, a conventional UNIX kernel or specialised forwarding hardware. The IPC finder is responsible for directing IPC calls to the appropriate processes. When XRL's are used to call methods on another XORP process the finder is responsible for mapping the calls to the appropriate processes for execution.

1.1.1 XORP Resource Locators

XORP uses XORP Resource Locators (XRL's) to describe inter-process calls and their arguments. They are presented in a human readable form which allows for easy manipulation for debugging purposes. XRL's are usually presented via an XRL interface. That is a set of related methods that an XRL target would implement where an XRL target is something that XRL requests can be directed to in order to be executed.

XRL interfaces make it simple to communicate between XORP processes. Each process exports its own XRL interfaces for other processes to use. This methodology also has the added advantage that XORP processes do not necessarily need to reside on the same host, another key advantage for the EUA as it means the MLE's can be on a different host to the EUA.

XRL's specify the method name along with the input and return arguments. The format of an XRL is as follows.

```
<method name> ? <input arg1>:<input arg1 type> & <input
arg2>:<input arg2 type> → <return arg1>:<return arg1 type> &
<return arg2>:<return arg2 type>
```

For example, to specify a method in the XRL interface named get_example_values that takes two input arguments input1 (a 32 bit integer) and input2 (an ipv4 address) and returns output1 (a string) we would use the following method definition.

```
get_example_values ? Input1:i32 & input2:ipv4 → output1:txt
```

As can be seen by this example the interfaces are simple to define and easily readable. This makes working with them simple for a user.

1.2 ECODE Unified Architecture Design

The EUA will be built directly on top of the XORP infrastructure. A new XORP module will be created known as the Translation and Communication Interface (TCI). The TCI exports a set of XRL interfaces that enable MLP's to communicate indirectly with the routing functionality offered by XORP. By implementing an intermediary point in the TCI there is a point of control that can be used to mediate the MLP's activities and offers a well defined set of functionality for the MLP's to take advantage of.

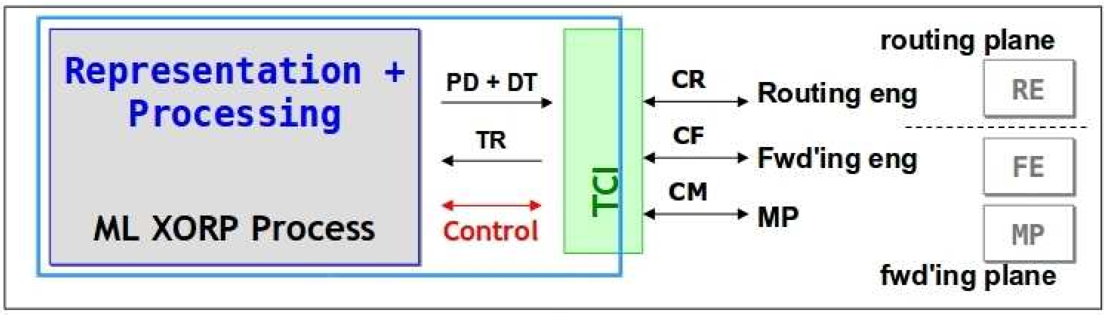

Figure 2: ECODE Unified Architecture Overview

An overview of the proposed EUA can be seen in Figure 1.2. The TCI can be seen as the intermediary communication point between the MLP and the routing infrastructure. There are a number of interfaces defined between the TCI and MLP as well as between the TCI and the routing infrastructure. The interfaces allowing the TCI to communicate with the routing infrastructure are the Cognitive Routing (CR), Cognitive Forwarding (CF) and Cognitive Monitoring (CM) interfaces, shown to the right of the TCI in Figure 1.2.

The Translation to Representation (TR), Processing to Distribution (PD), Distribution to Translation (DT) and control interfaces shown to the left of the TCI in Figure 1.2 are responsible for all communication between the MLP and the TCI. The details of all these interfaces are discussed in more detail later in this document.

The Machine Learning XORP Process (MLP) shown in Figure 1.2 is an XORP process that is implemented in such a way as to be able to communicate with the TCI over the interfaces described above. The internal specification of each MLP can be different to suit the specific purpose that it is performing. As long as it is implemented to communicate with the TCI it is able to take advantages of all the services the TCI has to offer such as communicating with MP's and the routing infrastructure in a controlled manner.

The TCI also offers Monitoring Points (MP's) a method of registering themselves so MLP's know what MP's are available at any time. MLP's can query the TCI as to what MP's are registered so they are aware of what monitoring information they are able to request.

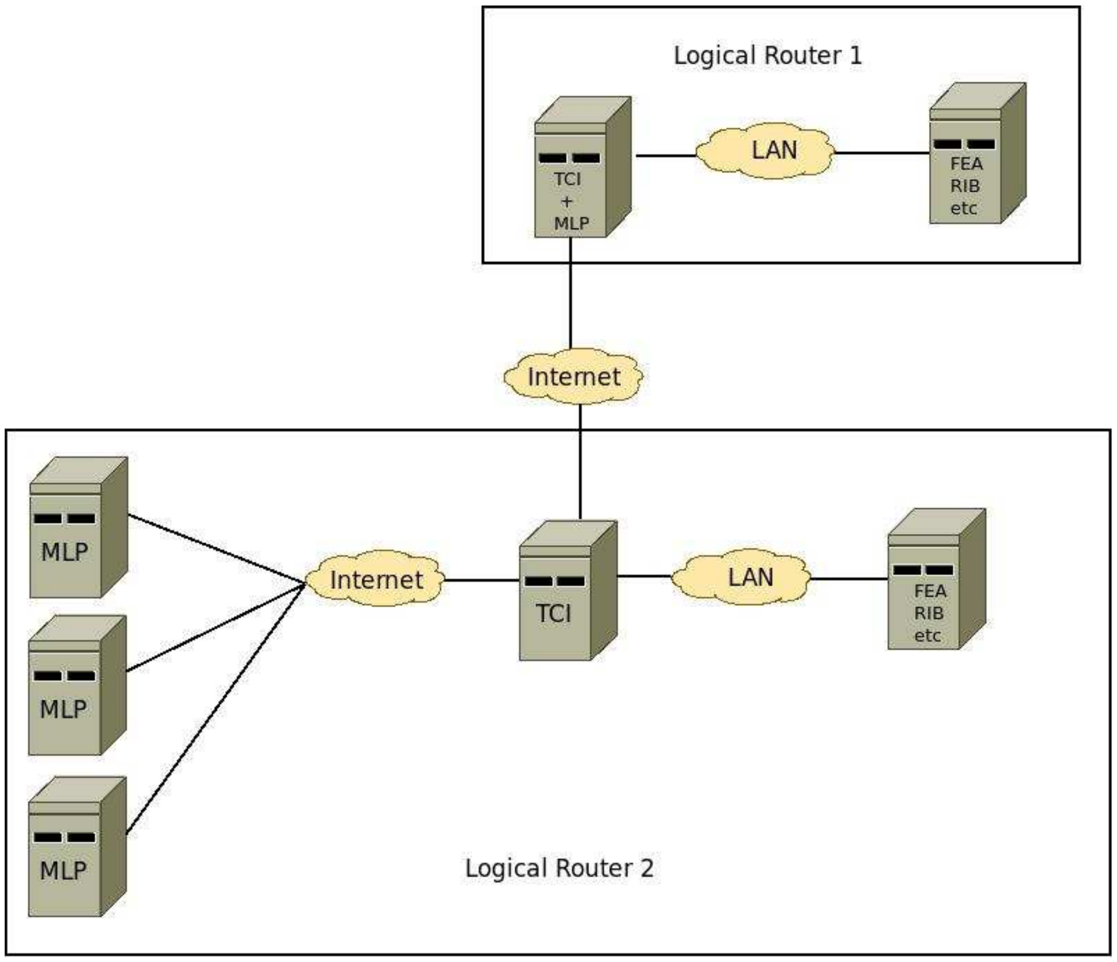

Figure 1.3: Two EUA Routers Split Across Multiple Machines

Because of the distributed nature of XORP there is no reason why all of the components of the EUA need to reside on the same machine. It is quite feasible to move elements onto remote hosts and have them communicate over the Internet or a local network.

Figure 1.3 shows an example of two complete EUA routers split across a number of different machines. As can be seen from this diagram the MLP's, TCI and XORP routing infrastructure can be separated across multiple machines and networks. An important point to note is that as shown in the diagram there is only ever one TCI per logical router. Every MLP that wishes to interact with a router can do so via this single point.

This is just one of many possible scenarios. For example, the MLP's and TCI could both be on the same machine with the XORP infrastructure on another machine, separated either by the Internet or a LAN. The XORP routing infrastructure can also be split up into multiple processes and spread across more machines if necessary. The RIB could be placed on a separate machine from the rest of the software if needed. In fact, most of the processes shown in Figure 1.1 could be placed onto separate machines if the need arose.

The fact that the processes can be placed across multiple machines on multiple networks demonstrates just how flexible the XORP platform is and this is one of the main reasons it was chosen as the underlying infrastructure for the EUA.

1.2.1 Dispatching

An important feature of the EUA is the ability to dispatch method calls to both the TCI an MLP is directly connected to and remote TCI's. This functionality is known as *dispatching*.

TCI's provide two generic methods that MLP's can utilise to execute methods. The first of these is the *dispatch* method. This allows an MLP to pass an XRL in along with the name of the TCI and the MP on which it wants the XRL to be executed. This enables an MLP to simply pass in the ID of a remote TCI as well as the name of an MP and the exact XRL to be executed and be passed back the result.

The second piece of functionality offered by EUA dispatching is the ability to pass in XRL's in the same method as for the dispatch function but extends the functionality to support more than one XRL being dispatched at a time. This is performed by the *dispatchf* XRL. An MLP can call dispatchf XRL with the specified format in which it wishes to receive the results of the XRL's (much in the same way as the C printf command). It can also pass in a number of XRL's along with the TCI and MP on which the XRL is to be executed.

2. Processes

2.1 Specification of Translation and Communication Interface (TCI)

Within the ECODE architecture the Translation and Communication Interface (TCI) is responsible for exporting the functions that MLE's can use to communicate with the routing platform.

Because the TCI is an intermediary point between the MLP's and the routing infrastructure/MP's it can perform operations to optimise requests. For example if two MLP's requests packet sampling via the TCI, one at a rate of one packet per hundred and one at a rate of one in 50, rather than setting up two separate packet sampling operations the TCI can just set up one at a rate of one in 50 and use this sample to fulfill both of the MLP's requirements. The fact that the TCI lies between the MLP'ss and the underlying infrastructure enables it to monitor the decisions that MLP's arrive at. This means that it can also act as a conflict resolution engine. If two separate MLP's attempt to keep changing a route, for example, in a way that conflicts with the other MLP's decision the TCI can mediate this behavior.

One important piece of functionality that the TCI will provide is the communication between remote MLP's. The TCI will allow MLP's to disover the addresses of remote TCI's. This in turn will enable an MLP to transmit and receive messages to remote MLP's via the TCI. This will mean that whether MLP's are local or remote they can be communicated with just as easily.

The TCI is implemented as an XORP process and as such can have configuration commands built directly in to the XORP command structure. This will make configuration of the TCI as simple as configuring any other part of the XORP platform.

3. Internal Communication Interfaces

The internal communication interfaces in the ECODE architecture are those used between components communicating with a single TCI. Due to the modular design of XORP these components do not necessarily have to reside on a single host.

3.1 Router to TCI Interfaces

The interfaces between the routing infrastructure and the TCI are what enables it to alter the way in which the router behaves. As XORP has been decided upon as the underlying architecture for the EUA the TCI can take advantage of the functionality that XORP offers to achieve this.

3.1.1 CF Interface

The CF interface is the means by which the TCI interacts with the forwarding infrastructure. The forwarding table is largely controlled by the routing engines inside XORP but XORP also provides various pieces of functionality that can be used by other XORP processes. The details of the methods that XORP provides are too lengthy for this document but the methods that are exported by the XORP infrastructure will be used by the TCI to manage the forwarding tables when required.

3.1.2 CR Interface

The CR interface is responsible for relaying information between the TCI and the RE. The TCI uses this interface for retrieving information regarding routes from the RE. It can also set and change routes via this interface. The methods that the RE provides are defined within XORP and can be used for a multitude of operations on the routing infrastructure. The details of these individual methods are beyond the scope of this document, however methods described later describe how MLP's can interact with the TCI and all of the information the methods regarding routing is retrieved over the CR interface. All methods provided by the RIB in XORP can be taken advantage of within the EUA.

3.1.3 CM Interface

The CM interface provides a number of methods for MP's to communicate with the TCI. It also provides the functionality for MP's to register and de-register along with propagating information to the TCI regarding the functionality an MP can provide. This section describes the methods that will be implemented on the TCI which MP's can take advantage of.

- `mp_register ? mp_id:txt`
- `mp_deregister ? mp_id:txt`

  The mp_register and mp_deregister methods allow an MP to register and deregister itself with the TCI. Once an MP has registered with the TCI it knows that the MP can be used for monitoring and if an MLE requests information for an MP that has been registered the TCI can pass on the requests and moderate the way in which the data is collected. This means that if a number of MLE's are requesting the same information from an MP the TCI can request this information once and pass it on to all interested MLES's thus reducing the load on the MP.

- `mp_advertise_methods ? mp_methods:list<txt>`

To enable MP's to inform the TCI of the methods it provides the method described above will be provided. This method can be called by an MP with the parameter mp_methods set to a list of the method names it provides separated. By providing this method the TCI can gather information about all of the MP's. It is expected that MP's will have different functionality in different versions. Therefore allowing the TCI to know exactly what methods an MP provides means it can inform an MLP that requests a non-existent function without having to try to forward the requests on to the MP each time. It also means that the TCI can relay information about available methods to an MLP without having to query the MP each time.

3.2 Outer Interfaces (TCI to MLP)

The outer interfaces define how the TCI communicates with the MLP's. There are a number of methods described in this section. All of the methods are represented in the form of an XRL interface description as explained earlier in this document.

3.2.1 Specification of Translation to Representation (TR) Outer Interface

The TR interface is one of the most important in the EUA. It is the interface that allows an MLP to request information from the TCI that it will use in its own algorithms. Because of this there are a lot of methods defined that can provide the MLP with a variety of information.

The TR interface offers a number of methods to allow MLP's to interact with the Routing Information Base (RIB). By performing these actions via the TR interface of rather than interacting directly with the RIB, the TCI can oversee any operations that are being carried out and if there are any conflicts with the routing information it can moderate them before the requests hit the RIB. The methods that will be offered by the TCI to MLE's are described below.

i) MLP Interactions

- `mlp_register ? mlp_id:txt → ret:i32`

  This method allows an MLP to register with the TCI. It is required that an MLP calls this method before it can take advantage of any of the other functionality the TCI provides. It must provide its unique mlp_id string and the return value specified in the ret parameter is 0 if the MLE was successfully registered or a positive integer on error. The currently defined error codes are:

  1. The TCI could not register the MLP as the maximum number of allowed MLE's are already registered.
  2. The TCI could not register the MLP as the load on the TCI is currently too high.

- `mlp_deregister ? mle_id:txt → ret:i32`

  The mlp_deregister method enables an MLP to remove its registration status from the TCI. This is to be used when an MLP is shutting down and no longer needs any services from the TCI. An MLP must ensure it calls this method when it is closing to free up resources within the TCI. The return value is 0 on success and a positive integer if its registration could not be successfully removed.

- `mlp_advertise_methods ? mlp_methods:list<txt>`

MLP's can inform the TCI of the methods it provides using the method described above. This method can be called by an MLP with the parameter mp_methods set to a list of the method names it provides separated. By providing this method the TCI can gather information about all of the MLP's. It is expected that MLP's will have different functionality in different versions. Therefore, allowing the TCI to know exactly what methods an MP provides means it can inform anything that requests a non-existent function without having to try to forward the requests on to the MLP each time. It also means that the TCI can relay information about available methods to other processes without having to query the MLP each time.

ii) MP Interactions

- `discover_mp ? mp_id:txt → mp_available:bool`

The discover_mp method can be used by an MLP to check if a certain MP is registered with the TCI. The MLP needs to specify the id string of the MP in the mp_id variable. The method returns a boolean value of true if the MP is registered with the TCI or false if not in the mp_available return parameter.

- `get_available_mps → available_mp_ids:list<i32>`

The get_available_mps method allows an MLP to query the TCI as to which MP's are currently registered. It returns a list of the ids of MP's the TCI is aware of.

- `get_mp_methods ? mp_id:txt → mp_methods:list<txt>`

This method allows an MLP to discover the methods that an MP registered with the TCI is advertising. By calling the method with the mp_id set to the id string of the MP to query the TCI will return a list of the MP's methods in the mp_methods parameter. This can be useful to MLP's as they can call this method and check that the MP supports any method they wish to use.

Along with a list of methods the mp_methods parameter contains metadata describing each method. Each item in the list contains the method name followed by a string that describes it. This information is separated by the “|*|” delimeter to enable the strings to be parsed. As an example take the following string that could be returned as an item in the mp_methods list:

```
ping_ip ? ip:ipv4|*|Ping an IPv4 address
```

This string can be parsed into the XRL method and the metadata by using “|*|” as the delimeter. In this case it can be seen that the XRL is called 'ping_ip' which takes an IPv4 address as a parameter. The metadata describes the method as “Ping an IPv4 address” so it is immediately obvious what functionality this XRL provides.

iii) Registering Interest

The TCI provides methods to enable MLP's to register their interest in MP's. This is desirable as if an MP is not available that the MLE wishes to use at one point in time it can register its interest so that if it becomes available the MLE can immediately begin using it. The following methods allow an MLE to register their interest in MP's.

- `register_mp_interest ? mp_id:txt`
- `deregister_mp_interest ? mp_id:txt`

The register mp_interest method allows an MLP to register its interest into the status of all MP's connected to a TCI. Whenever any MP registers or deregisters with the TCI it will inform the MLP of this using it's callback method described later. This allows an MLP to constantly know which MLP's are available to retrieve information from. The deregister_mp_interest method informs the TCI that the MLP no longer wishes to be informed of status changes of MP's.

- `register_single_mp_interest ? mp_id:txt & mlp_id:txt`
- `deregister_single_mp_interest ? mp_id:txt & mlp_id :txt`

The register_single_mp_interest method gives an MLP a way of registering its interest in a single MP. This is different to the register_mp_interest method in that the TCI will only notify the MLP when the status of a specific MP has changed. The deregister_single_mp_interest method removes the MLP's interest from that MP on the TCI. The mp_id parameter for both methods is the id string of the MP that the MLP is interested in. The mlp_id parameter is the ID of the MLP that is registering interest.

iv) MLP Interactions

- `discover_mlp ? mlp_id:txt → mlp_available:bool`

It is possible to query a TCI to discover if it has a specific MLP registered using the discover_mlp method. The method takes the ID of the MLP that is trying to be discovered and returns true if the MLP is registered or false otherwise.

- `get_available_mlps → available_mlp_ids:list<i32>`

This method returns a list of all MLP's registered with a TCI in the form of an XORP list.

- `get_mlp_methods ? mlp_id:txt → mlp_methods:list<txt>`

The get_mlp_methods method allows a TCI to be queried regarding what methods one of its registered MLP's supports. It's functionality is almost identical to get_mp_methods as described previously. The only difference is that it is working on MLP's rather than MP's. A list of the methods supported by the MLP ID specified is returned in the mlp_methods return argument using the same format as for get_mp_methods.

v) Dispatch Methods

The EUA provides generic methods that can be used to call methods on other XORP processes that export an XRL interface. Although it would be possible for an MLP module to call these methods directly, calling them via the TCI enables the EUA to moderate the method calls. It also allows the EUA to cache results from common calls to reduce the processing time required on the other XORP modules. This could be extremely effective when performing actions such as retrieving routes or changing routes. When an MLE reaches a decision about changing a route for example, it can call the appropriate method on the RIB to change the route. If multiple MLP's keep reaching different decisions about the same destination however, and keep changing the route to their own, the TCI can step in to moderate this.

- `dispatch ? euaxrl:txt → ret:txt`

  The dispatch method is used to dispatch an XRL method call to the TCI. The TCI can then pass the method on to the appropriate XORP process to be executed. The euaxrl parameter specifies the TCI to call the method on along with the XORP process and the actual XRL method and parameters to execute. The format of the euaxrl parameter is as follows.

  ```
  tci_id->xrl_target->xrl_method?param1&param2...
  ```

  For example, if an MLP wanted to call the method ping6 on an MP with the ID ping_mp on a TCI with the ID lancs_tci and pass it the parameter 2001:db1::1 it would use the following string for the euaxrl parameter.

  ```
  “lancs_tci->ping_mp->ping?2001:db1::1”
  ```

  When the TCI receives a method call such as this it uses the TCI resolver (explained later in this document) to resolve the IP address of lancs_tci and forwards the method call on to it where the remote TCI will call the ping method on the XORP module ping_mp. The result can be passed back and get forwarded all the way back to the MLP which requested it.

- `dispatchf ? format:txt & euaxrls:txt ->ret_str:txt`

  The dispatchf method allows an MLP to call more than one method and have the results formatted in a way that it specifies. The format string specifies how the MLP wishes the results to be displayed. The idea behind the dispatchf function comes from the standard C printf function. The format string is built up of special format arguments like %d for a decimal and %s for a string that will get replaced by the return values from the euaxrl's specified in the euaxrls parameter. The euaxrls parameter contains a list of euaxrl type method calls (as described for the dispatch method) separated by commas much in the same way that the arguments are specified for the C function printf. The format specifiers specified to date are:
  - %s - String
  - %d – Decimal
  - %ud – Unsigned Decimal
  - %i4 – IPv4 Address
  - %i4n – IPv4 Network
  - %i6 – IPv6 Address
  - %i6n – IPv6 Network
  - %b – Boolean
  - %m – MAC Address

- `dispatch_push ? euaxrl:txt & callback:txt → ret:i32`

  The dispatch_push method is for calling a method on an MP where it is known that this method will need to push information back to the MLP periodically. For network monitoring this is something that is likely to arise in a number of scenarios as reports will be generated and pushed back from an MP rather than being requested by the MLP. The euaxrl parameter specified the EUX XRL to execute on the MP. The callback parameter specifies a method that the TCI can call on the MLP to push data back whenever necessary. The method returns an integer value to indicate whether it was successful or not.

vi) Methods Regarding Remote TCI's

- `discover_tci ? closest_ip:ipv4 → tci_id:txt`

  The discover_tci method can be used to try and locate remote TCI's. The MLP must call the method and pass in an IPv4 address via the closest_ip parameter. The TCI will then query the TCI resolver to discover the TCI with the closest IP address and return the ID of it to the MLP in the tci_id return value. If the call fails then 0 is returned in the tci_id parameter.

- `tci_has_mp ? tci_id:txt & mp_id:txt → has_mp:bool`

  An MLP can call this method to discover if a remote TCI, specified in parameter, tci_id, has the MP, specified in parameter mp_id, registered with it. The local TCI can then query the remote TCI to check and return true if the remote TCI has the MP or false if not.

- `tci_has_mlp ? tci_id:txt & mp_id:txt → has_mp:bool`

  This method allows an MLP to query a remote TCI to discover if it has a specific MLP registered with it. It works in much the same way as the tci_has_mp method but takes the MLP ID as a parameter rather then the MP ID. It returns true if the remote TCI has the MLP registered or FALSE otherwise.

- `get_tci_xrls ? tci_id:txt → euaxrls:list<txt>`

  This method allows a local TCI to query a remote TCI to discover what methods it is exporting. The return parameter contains a list of EUA XRL's (as specified previously for the dispatch method). The EUA XRL list returned specifies the name of the TCI on which the methods can be called as well as the MLP and the method, including a full list of the methods parameters. The local TCI is responsible for querying the remote TCI and formatting the list of EUA XRL's before it is put into the euaxrls parameter and returned to the calling MLP.

  The advantage of having this method request the list of xrl's available from the appropriate remote TCI rather than storing the list of available functionality in the resolver is that the remote TCI could export different sets of xrl's based upon who is requesting them. For example a TCI may only wish to export a limited set of functionality to TCI's that aren't within the same AS but export more functionality to any TCI that is within its own AS. By forcing a remote TCI to request the set of xrl's directly from the local TCI it can make a decision when it receives the request as to how much functionality it exports.

- `discover_tcis_with_mp ? mp_id:txt → tci_ids:list<txt>`

  This method is used to retrieve a list from the TCI of all remote TCI's that support the MP specified in the mp_id parameter. The TCI can request the information from the TCI resolver, described later in this document. It then returns a list of TCI ID's in the tci_ids parameter to the caller.

- `discover_tcis_with_mlp ? mlp_id:txt → tci_ids:list<txt>`

The discover_tcis_with_mlp method operates in the same way as discover_tcis_with_mp except it returs a list of TCI ID's that have the MLP specified in the mlp_id parameter registered with them.

3.2.2 *Specification of DP+TD Outer Interface*

The DP+TD interface enables MLP's to provide methods that the TCI can communicate with. These methods can effectively be seen as callback functions for some of the methods the TCI provides. Many of the methods that require such functions have already been described in the previous section and this section provides an overview of the corresponding functions the MLP's must provide in order to receive the appropriate notifications.

i) MP Interest Replies

In order to receive information from the TCI regarding the status of MP's after the MP related methods described in Section 3.2.1 have been called an MLP needs to implement the following methods.

- `mp_status_changed ? mp_id:txt & mp_available:bool`

  The mp_status_changed method is called by the TCI when the availability of an MP has changed. This method is called when an MLP has called the register_mp_interest method on th TCI indicating that it wishes to be informed when the status of any MP has changed. The id string of the MP is passed in via the mp_id parameter and the MP's availability is denoted by the mp_available paramater. The mp_available paramater is set to true if the MP has recently registered with the TCI or false if it is no longer available.

- `mp_single_status_changed ? mp_id:txt & mp_available:bool`

  The mp_status_changed method is called by the TCI when the availability of a single MP has changed. This method is called when an MLP has called the register_single_mp_interest method on the TCI indicating that it only wishes to be informed when the status of a certain MP has changed. The parameters are the same as for the mp_status_changed method described above.

3.2.3 *Specification Of The Control Interface*

It is perceived that there will be a CLI that will allow the TCI to be configured. In order to enable this CLI application a number of methods will need to be implemented on the TCI for the CLI application to communicate with for setting the configuration and control parameters.

The control interfaces on the TCI allow configuration and control information to be communicated. The methods which provide this functionality are outlined below.

- `set_max_mp ? mp_limit:u32 → ret:i32`

  This method allows the configuration of the maximum number of MP's the TCI is allowed to register. This is useful to keep the TCI from becoming overloaded with MP's which could cause it to devote too much time to processing requests. The return value specified in the ret parameter is 0 if the TCI set the maximum number of connected values. On error a positive integer is returned. At present the following error codes are defined.

1. Could not set the maximum number of MP's are more than the number specified are already registered.

- `set_max_mlp ? mlp_limit:u32 → ret:i32`

  The sex_max_mlp method provides similar functionality to the set_max_mp method described above except it sets the maximum number of MLP's that are allowed to register. Again, on success 0 is returned and on error a positive integer is returned. The error codes defined at present are as follows:

1. Could not set the maximum number of MLP's as more than the number specified are already connected.

- `set_mlp_priority ? mlp_id:txt & prio:u32 → ret:i32`

  This method allows the priority level of an MLP to be set. If the TCI needs to prioritise replies or the order in which it executes requests it uses the values set for each MLP via this method. This becomes more relevant the higher the load on a TCI is as it means the it can always perform on behalf of an MLP with the highest priority first. The id of the MLP for which the priority is being set is contained in the mle_id string and the new priority is passed in via the prio parameter. The return value is 0 on success or a positive integer on error.

- `set_mlp_throttle_threshold ? mlp_id:txt & threshold:i32`

  To configure the threshold at which the TCI considers an MLP to be throttling, the set_mlp_throttle_threshold can be used. The MLP ID is passed in via the mlp_id parameter. The threshold parameter sets the number of queries an MLP is allowed to perform per second before it is considered to be throttling the TCI. If an MLP performs more queries than this it will receive an error code for any subsequent queries for a pre determined time period.

  The control interface will also need to allow policy configuration of the TCI. The policies referred to in this instance are concerned with how the TCI mediates requests from multiple MLP's that are conflict with each other. The exact specification of how this will be achieved will emerge during the course of the implementation as more is learnt about how the MLP's will interact.

4. External Communication Interfaces

4.1 TCI to TCI Communication

TCI's located on remote nodes may have reason to share information. This could be for their own purposes or because MLP's communicating with them have requested information that can only be obtained via a remote TCI such as a latency measurement between two remote TCI's.

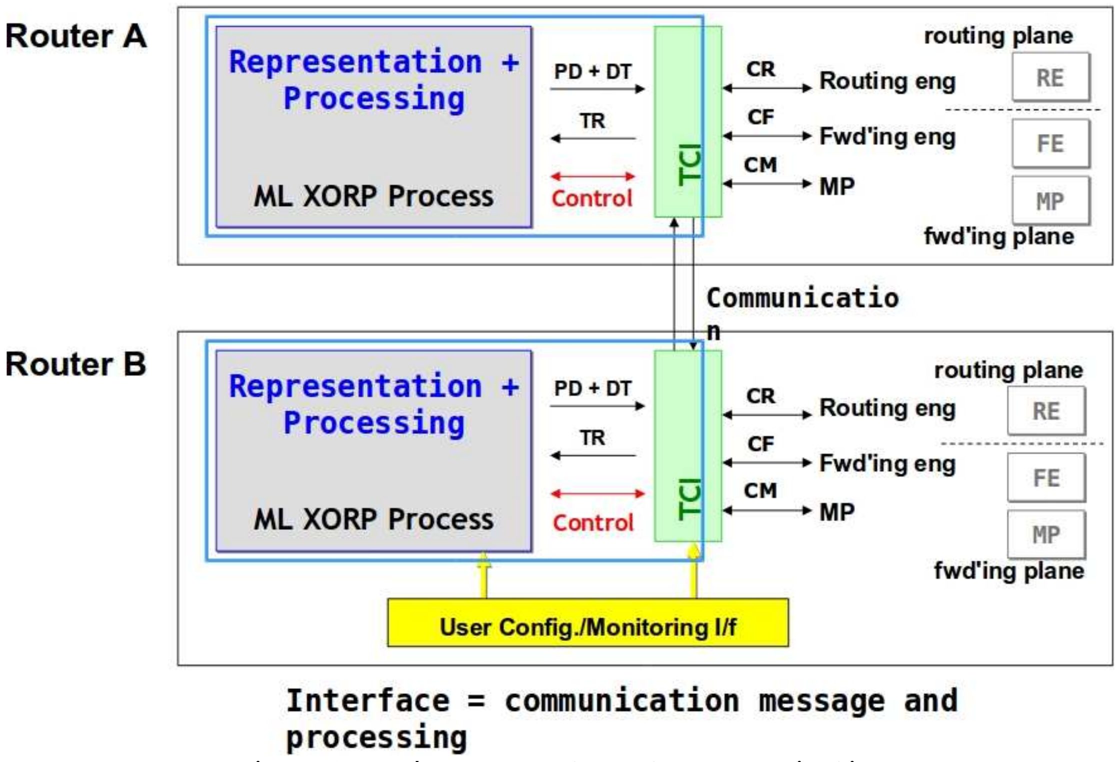

Figure 3: TCI's on separate routers communicating

The diagram in Figure 4.1 shows an overview of two remote MLP's communicating via their TCI's. In order for a TCI to communicate with a remote TCI it needs to be able to first discover the addresses of any remote TCI's. To enable this discovery the EUA specifies the TCI resolver.

4.1.1 TCI Resolver

The TCI resolver is a server with which TCI's register when they first come online. It is implemented outside of XORP as a separate entity and is comparable with DNS. When a TCI is started it must register with the TCI by sending it's own identifier string along with an IP address over which it can be contacted. If the resolver already has a TCI registered with the ID string provided it will inform the TCI in which case it will have to retry with another one. The ID string chosen by the TCI is important as it is used by other TCI's within the EUA to communicate with it.

The TCI resolver also stores a list of which MLP's and MP's each TCI has registered. This allows a TCI to request lists of which TCI's have certain MLP's and MP's registered. This can be useful for MLP's that need other MLP instances to interact with as they can simply request a list of which TCI's have an MLP with a certain ID registered. They can then use this information to communicate with the MLP's they require. Note that although the resolver has a list of which MLP's and MP's a TCI supports it does not keep track of which methods these MLP's and MP's support. This is because TCI's may wish to advertise different functionality of an MP or MLP depending on who is requesting the information. For this to function correctly lists of MP and MLP methods must be requested directly from the TCI using the get_tci_xrls method already described.

By keeping track of all registered TCI's the resolver is able to provide the IP address that maps on to the ID string. This string can be used by MLP's when calling methods on remote TCI's.

The protocol to be used between the TCI's and the resolver is yet to be decided. As this will be implemented outside of the XORP framework the messages will likely be transmitted over a traditional socket interface as opposed to an XORP like RPC communications method. As the implementation progresses it is assumed that this protocol will be specified to take into account issues that arise.

It is also expected that as development continues more surrounding issues will need to be taken into account regarding the TCI resolver. Examples of such issues are security and scalability. If scalability proves to be an issue then changing the TCI resolver to take advantage of a Distributed Hash Table (DHT) may be looked into as this could drastically improve the scalability of the resolver.

5. Development Plan

In order to limit overall development risk and enable the EUA to be useful as quickly as possible, development has been split into three distinct phases. The first phase is to implement the TCI and it's communication with MLP's and MP's. The second development phase will deals with the implementation of TCI to TCI communication enabling the information retrieved by TCI's to be effectively disseminated to remote routers. The third phase is the implementation of features considered non-essential to the EUA operation but are however desirable.

One aspect of the EUA that is ongoing throughout the development process is development of the configuration and control interfaces. Most of the features will need to be configured in one way or another. For example there will need to be a maximum number of MLP's that are allowed to connect via the mlp_register() method. This will be configurable via the control interface and the implementation of features such as this will be carried out throughout the development process.

The three phases of the EUA development are as follows:

5.1 Development Phase 1 – Essential EUA Functionality

The first phase in the EUA development process is to implement the parts of the EUA that are considered essential to its operation.

Before the TCI can become useful to MLP's it needs to be able to retrieve the appropriate information from various MP's along with information from the routing infrastructure. Part of the first phase of the EUA development process involves building the interfaces between the TCI and the RE, FE and MP. By implementing the CR, CF and CM interfaces as previously described in this document the TCI will be able to effectively retrieve appropriate information from the underlying infrastructure and store it.

Registration of both MLP's and MP's is important as it allows elements of the architecture to see what is available to them at any time. Both the registering and de-registering of MP's and MLP's is to be implemented in phase 1. Along with this controlling how many MLP's and MP's are able to connect to the system will also be implemented and this will be accessible to change via the control interface.

Another important element of phase one is the implementation of the interfaces between the TCI and the MLP. To allow an MLP to interface with the TCI the PD+DT, TR and the control interfaces will need to be developed. These allow the exchange of information between an MLP and the TCI as well as enabling the TCI to be configured.

Dispatching is an important element of the EUA architecture. It is this that allows MLP's to request information from MLP's in a formatted way as well as issuing method calls on remote TCI's when the distributed EUA implementation has been completed and is therefore considered essential functionality and will implemented in phase 1.

The estimated completion date for Phase 1 of the development is January 2011.

5.2 Development Phase 2 – Make the EUA Distributed

For the EUA to operate as a distributed architecture the TCI's need to be able to communicate with each other. Task 2 in the development plan is the process of implementing this communication. In order to achieve this, the TCI resolver will need to be implemented outside of XORP to allow the TCI's to discover each other. The exact nature of the TCI resolver has yet to be established and will be an ongoing area of research throughout the development cycle.

The first iteration of the TCI resolver will use a simple client server model much like DNS does. The resolver will reside on a single server and all TCI's will communicate with this when they need to request information. Along with the TCI resolver the TCI's need to be able to relay XORP method calls between them when an MLP has dispatched a method call to a remote TCI. Phase 2 will also deal with implementing this functionality.

The estimated completion date for Phase 2 of the development plan is March 2011 at the earliest.

5.3 Development Phase 3 – Extra Features

The final development phase is the implementation of extra features that are not essential to the operation of the EUA. Although not essential, these features will enhance the EUA's capabilities and improve the services that the TCI can offer to connected MLP's.

The mediation of calls made by MLP's is a desirable property of the TCI but is not essential for it to function correctly. Therefore this functionality will be implemented within phase 3 of the development. The more time that is left at this stage will give more opportunity to research areas in which this mediation can be taken advantage of. This will allow the TCI to perform more efficiently and offer the best possible service to MLP's with minimal operational overhead.

Changing the TCI resolver to be more scalable is another desirable property. As the number of TCI's grows so does the strain put on the TCI resolver. If time permits phase 3 of the development will look at how the scalability of the resolver could be improved. This would most likely come from implementing the resolver as a DHT.

These are only a small number of possible ways the EUA could be improved. Any other promising possibilities that arise during the development of the previous phases will be researched during the third phase, as time permits.

Phase 3 of the development plan will run from the completion of Phase 2 until the completion of the ECODE project.

6. Use Case Scenarios

6.1 Use Case: Adaptive Monitoring

In the following paragraphs we run the different methods needed to accomplish a given adaptive monitoring task using the API(s) provided in D2.2.

6.1.1 Registering an adaptive monitoring point (MP) within the TCI

First of all any installed adaptive monitoring point should register using the following method to register itself with the TCI.

```
mp_register?mp_id:txt
```

Then, advertises the list of methods it supports:

```
mp_advertise_methods?mp_methods:txt
```

6.1.2 Registering the centralized MLE

```
mlp_register?mle_id:txt->ret:i32
```

6.1.3 Adapting the Sampling Rate of a given router: MLP -TCI-MP

Here we suppose that the MLP is receiving the NetFlow reports from the MP, and then once it decides to update the sampling rate of a given monitoring point, it should run the following algorithm:

First it gets the list of registered monitoring points from the TCI:

```
get_available_mps->available_mps_ids:txt
```

Once received, it looks for the list of monitoring points capable of adaptive passive monitoring (those related to INRIA use case). Then, it sends a request through the TCI in order to change the sampling rate of the selected MP.

```
For i in (get_available_mps->available_mps_ids:txt)
     For j in (get_mp_methods?i:txt->mp_methods:txt)
          If (j == update_sampling_rate(INRIA))
          {
               Inria_tci->i-> update_sampling_rate?a&b
               // (The sampling rate is a/b, b<>0, a<=b)
          }
          End
     End
End
```

6.1.4 Receiving the list of NetFlow reports

Our passive monitoring engine relies on a set of text configuration files, which should be setup before starting the daemon. Once the daemon is started it registers within the TCI according to the methods we describe in the first paragraph. Then it starts collecting traffics according to the initial sampling rate specified in the configuration file, construction NetFlow reports and sending them back to the MLP via the TCI once the MLP is registered within the TCI.

6.2 Use Case: Network Coordinate System

6.2.1 Introduction

In a Network Coordinate System (NCS), we associate mathematical attributes —the network coordinates— to the nodes[1] part of the system, in such a way that knowing the coordinates of two nodes we can compute some kind of distance(s) between them. A simple example is the association of Euclidian coordinates to routers of a network, with Euclidian distance between them representing estimations of Round Trip Times. A more complex example would be the association of a pair of vectors to each router. With the knowledge of the pairs of two distant routers, we are then able to compute estimations of one-way delays in both directions between them.

Originally, such systems were deployed most of the time in peripheral devices (or end hosts) of network, often associated with a Peer-to-Peer (P2P) network organization. In such context, a node belonging to a Network Coordinate System essentially needs three inputs to process the algorithm and update its local coordinates[2]. These are the *local coordinates*, the *remote coordinates* and the *measurement*[3] between the nodes.

However, for deployment at the core components of the network, it seems interesting to adapt the architecture. For easy operation, having some kind of “Coordinates Server” (CS) rather than having to poll each router to know their coordinates is something desirable. With this in mind, using this server to compute the coordinates is a logical step further. It would be responsible for computation of coordinates of nodes belonging to the same administrative domain (or Autonomous System in Internet terminology). To do so, it will have direct access to the coordinates of router within the AS and can ask coordinates of routers from different AS's to similar Coordinate Servers. Obviously, it will still need the measurement of RTT between arbitrary nodes. The TCI of the ECODE Unified Architecture will be of great help here since it permits to send commands (XRL calls) to distant routers which implement it.

6.2.2 Use Case

From the network point of view, we will have three kinds of routers:

- “ECODE routers” which run a TCI instance. We expect them to have Monitoring Points able to measure RTT’s (or something else) between them. We do not expect them to have a NCS module running. These are the routers for which we will compute coordinates.
- “NCS router”. It is running a TCI but it also operates an NCS module. For the sake of simplicity, we will hereafter consider that it is alone in its administrative domain but it should obviously be part of a redundant architecture to avoid a single point of failure.
- The last ones are “standard routers”, they do not run a TCI instance or do not have the appropriate feature to measure delays. Thus they are not port of ECODE or our NCS. We will also use the TCI resolver.

1 And by node in this context, we mean some logical entity for which we can compute and consequently attribute coordinates. It refers most of the time to a router as a whole.

2 Unless said otherwise, we consider the relative error as part of coordinates.

3 This often refers to a delay measurement, such as RTT. However, it could possibly something different, such as, for example, measured available bandwidth.

From software point of view, we will essentially speak about the TCI, the NCS module and Monitoring Points.

6.2.3 Local TCI IDs

First the NCS router will have the responsibility to compute the coordinates for the ECODE routers in its administrative domain. Since they are under same administration, we expect that their TCI IDs are known and that the NCS module will be manually configured to compute coordinates of these relevant ECODE routers.

6.2.4 Discovering Coordinate Servers

Each NCS router (or Coordinate Server) will have to discover other CS's. Without any neighbor, it would be only able to compute coordinates of its local ECODE routers within a partitioned space (obviously independent from the rest of the Internet). Discovering these CS's is as simple as asking the TCI resolver for them using the method:

```
discover_tcis_with_mlp ? mlp_id:txt → tci_ids:list<txt>
```

In other words, the NCS router will request to the TCI resolver a list of routers with the same advertised MLP.

6.2.5 Build and maintain Coordinate Databases

After the reception of the CS list, our particular CS will build neighbor relationships with some of them. Even in a context as wide as Internet, their amount can be limited if each CS share all its information with the others (that is, each CS should eventually receive coordinates from its *n* neighbors, and from their n^2 neighbors —minus redundant ones). So, in function of the success of the system and the expected coverage, we could choose a reasonable number of neighbor CS (32, 64 or 128 will obviously give coordinates of routers in resp. 1024, 4096 or 16384 other AS's).[4]

So, the Coordinate Server will subscribe (through a method defined within the NCS module) to each of its neighbors. Each CS will maintain a list of subscribers and will share its information with each one of them (sending all its Coordinate Database at first contact and updates afterwards).

6.2.6 Keep the local coordinates up-to-date

The CS is responsible for maintaining the network coordinates of routers within its AS. It will regularly ask to each ECODE router under its administration to perform a measurement towards some node whose coordinates are presents in its Coordinate Database. It will do so using the following method:

```
dispatch ? euaxrls:txt → ret:str
```

with the euaxrl looking like :

```
origin_tci_id->ping_mp->ping4?distant_tci_id
```

4 In simulated or emulated environments, the amount of different ASes will certainly be limited and a full peering between the CSes could be sufficient and is likely to happen.

Notice that we expect these nodes to have the right MP available since they are part of the system. However, we can still ensure that fact by querying the ECODE router through :

```
tci_has_mp ? tci_id:txt & mp_id:txt → has_mp:bool
```

with mp_id replaced by `ping` if we want to perform a RTT measurement.

7. Concluding remarks

7.1 Risk Assessment

In order to reduce any risk associated with the development and to ensure that a working implementation can be produced it is important to limit the risk involved. This has been accomplished by the specification of the development plan described in Section 5 of this report. By dividing the development process into several phases it allows development to continue whilst evaluation and debugging of the earlier phases is performed simultaneously.

By splitting the development into 3 phases (ordered by importance of their resulting functionality) the risk of ending up with an architecture that cannot fulfill its basic functionality is mitigated. Phase 1 of the development will result in a minimal system that can serve the needs of MP's and MLP's albeit without all of the desirable properties. Phase 2 will see the addition of a distributed aspect to the architecture allowing remote TCI's to communicate with each other, exchange information and call XRL methods on remote TCI's with the aid of the local dispatch method. Phase 3 sees the addition of extra functionality that whilst not essential will further the research of the ECODE project. As long as the phases are completed in the order in which they have been presented in this document, any risk associated with the development cycle is minimised.

7.2 Risk Analysis

Risk	Probability	Effects
1. Development of dispatching exceeds estimated time	Low	Development of phase 2 and 3 of the development plan are moved further back in the time scale and development time for phase 3 is reduced.
2. Development of the distributed EUA exceeds estimated time	Medium	Time allocated to phase 3 of the project will be reduced.
3. Client-Server based TCI resolver is not sufficiently scalable	Very Low	Research into more scalable resolver will need to be carried out.
4. Github Failure	Very Low	Delay in development time whilst various repositories are updated with new information.
5. EUA Use Case Integration Not Fully Completed	Medium	No case studies for the EUA software resulting in poor evaluation
6. Formatted dispatching strategy is not realisable in the project time scale	Low	Dispatchf API method will not be available.
7. Inter process delay	Medium	If the inter process delay of XORP processes proves to be high some use cases could suffer
8. Reliability	Low	If system is not reliable results from evaluation of use cases could be inconsistent

Mitigation Strategies

- Risk_1: The third phase of the development plan is the development of features that whilst it would be nice to have, are not essential. If the development of dispatching in phase 1 exceeds expectations then the number of features developed during phase 3 will have to be reduced to provide the extra time needed.

- Risk_2: If the development of the distributed EUA in phase 2 of development exceeds the estimated time frame then fewer extra features will be implemented in phase 3.

- Risk_3: If a client – server based TCI resolver doesn't scale sufficiently for the purpose of the ECODE evaluation, a different approach will have to be considered. If this occurs then another possibility such as using a DHT for the resolver will have to be considered. Due to the relatively small number of experiments to be carried out however, we see the probability of this risk happening as being very low.

- Risk_4: If github fails or if they go out of business this will have a small impact on the development process for the EUA. Luckily this will not pose a large problem as with the git version control system each local copy of data holds a full copy of all revisions. The developers will need to agree upon where to host the main copy that everyone syncs with and update their local repositories accordingly.

- Risk_5: If ECODE use cases are not integrated into the EUA it could pose a problem with respect to the evaluation of the software framework. To limit the risk of this all functionality will be made available as soon as it has been developed to enable the use case developers to integrate as quickly as possible.

- Risk_6: It is possible that formatted dispatching will not be realisable in the project time scale. If this is the case the EUA will still be functional. The downside will be that each dispatch will only be able to call one EUAXRL at a time. Whilst this will limit functionality of the system, it will not cause the system to function incorrectly.

- Risk_7: It is possible that the delay between XORP processes will be of a level that could prove problematic for some use cases. In this case we will first have to discover the extent of the problems caused by the delay. We will also need to look at the XORP code base to see if we can reduce it.

- Risk_8: Making the system as reliable as possible will be a strong concern throughout the development process. To enable the results of experiments to be consistent the system will need to produce reliable result sets over a number of runs. This will be tested to ensure consistency and reliability by testing use cases repeatedly.

8. Acronyms

CF – Cognitive Forwarding
CM – Cognitive Monitoring
CR – Cognitive Routing
DHT – Distributed Hash Table
DT – Distribution to Translation
EUA – ECODE Unified Architecture
FE – Forwarding Engine
FEA – Forwarding Engine Abstraction
MLE – Machine Learning Engine
MLP – Machine Learning Process
MP – Monitoring Point
PD – Processing To Distribution
RIB – Routing Information Base
RE – Routing Engine
TCI – Translation And Communications Interface
TR – Translation to Representation