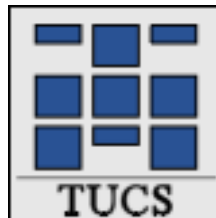


Fast Evaluation of Protocol Processor Architectures for IPv6 Routing

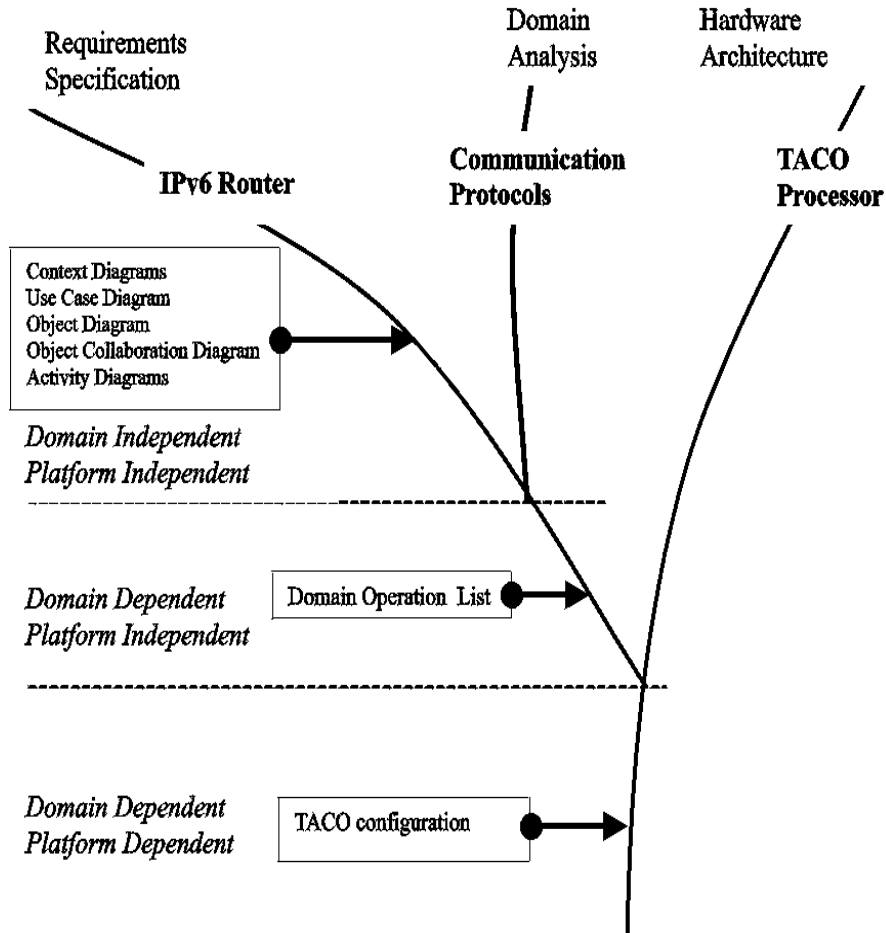
Dragos Truscan

Embedded Systems Lab

Turku Centre for Computer Science (TUCS), Finland

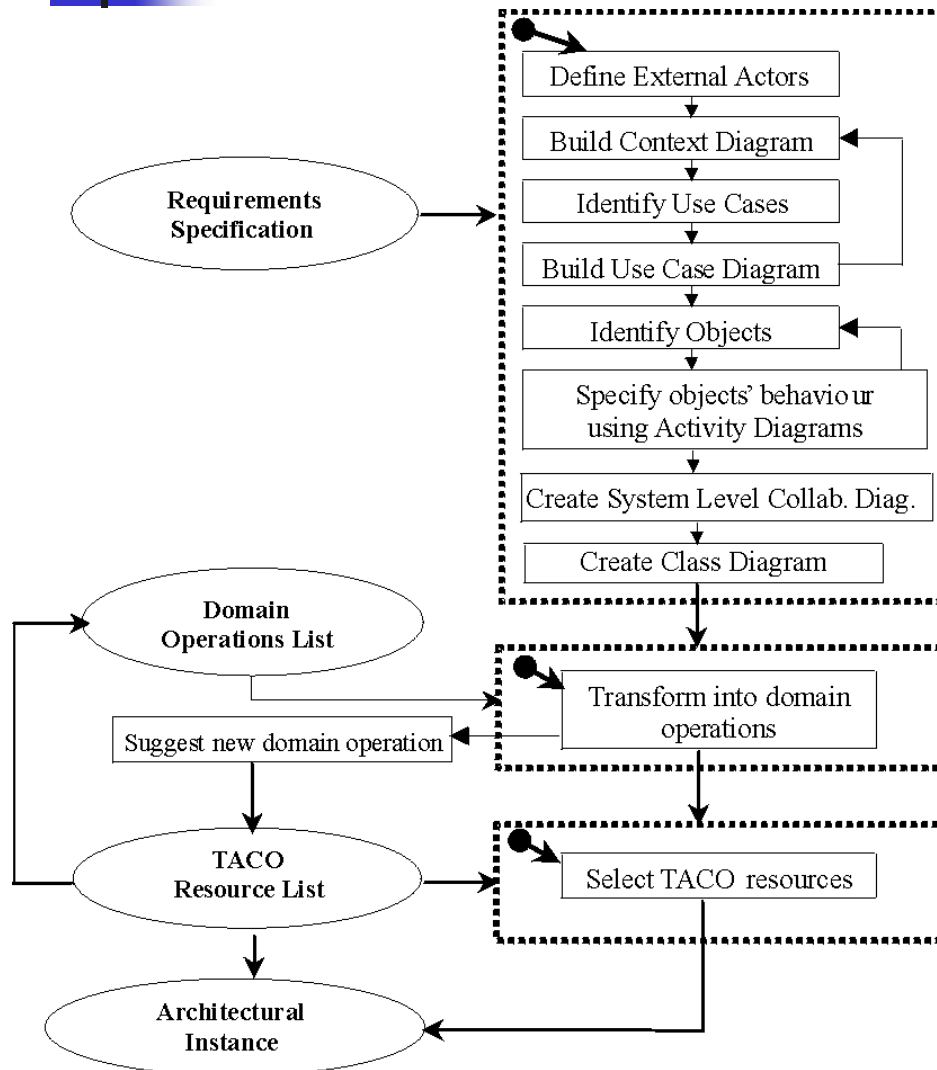


Introduction



- two approaches in designing protocol processing applications:
 - design an application for a general-purpose processor
 - create a dedicated processor that implements a given application
- we propose an incremental design flow that designs the application and configures the hardware in the same time
 - Unified Modeling Language(UML) as common notation
- configuring TACO means:
 - **qualitative configuration**
 - identify FUs needed by the application
 - **quantitative configuration**
 - decide the number of resources of each type
- sources of information:
 - application requirements (IPv6 router)
 - domain knowledge (communication protocols)
 - hardware architecture (TACO processor)

Design Flow



- Requirements specification
 - identification of system's border and its interaction with the environment
 - required functionality – Use Cases
 - objects in the system – Object Diagram
 - classes – Class Diagram
 - object behavior – Activity Diagrams

- Domain Analysis
 - create a list of domain operations
 - use domain operations to express the application specification

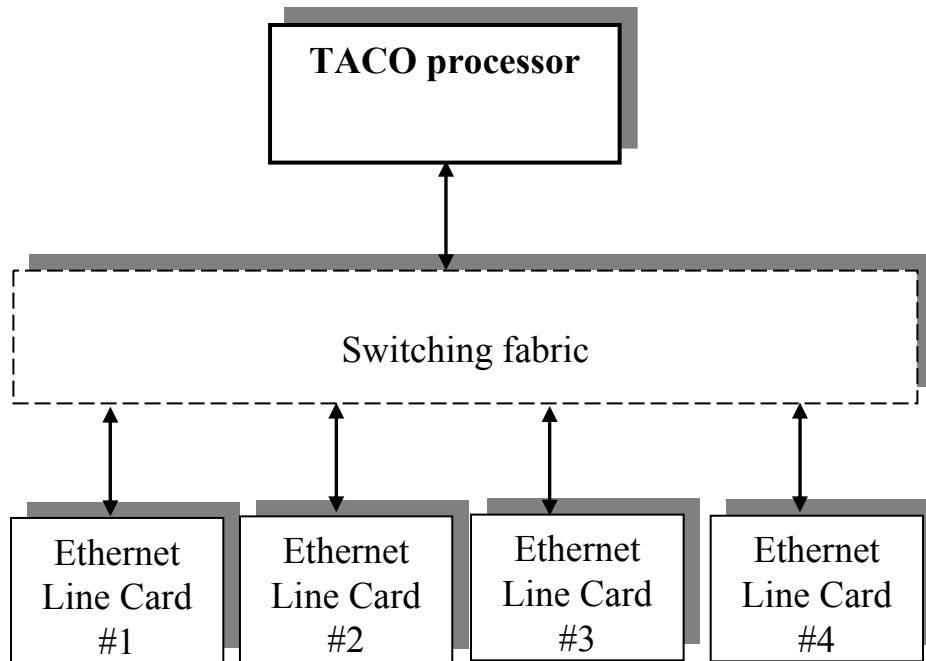
- TACO configuration
 - **qualitative configuration**
 - mapping domain operations onto physical resources
 - functional verification



IPv6 Router - case study

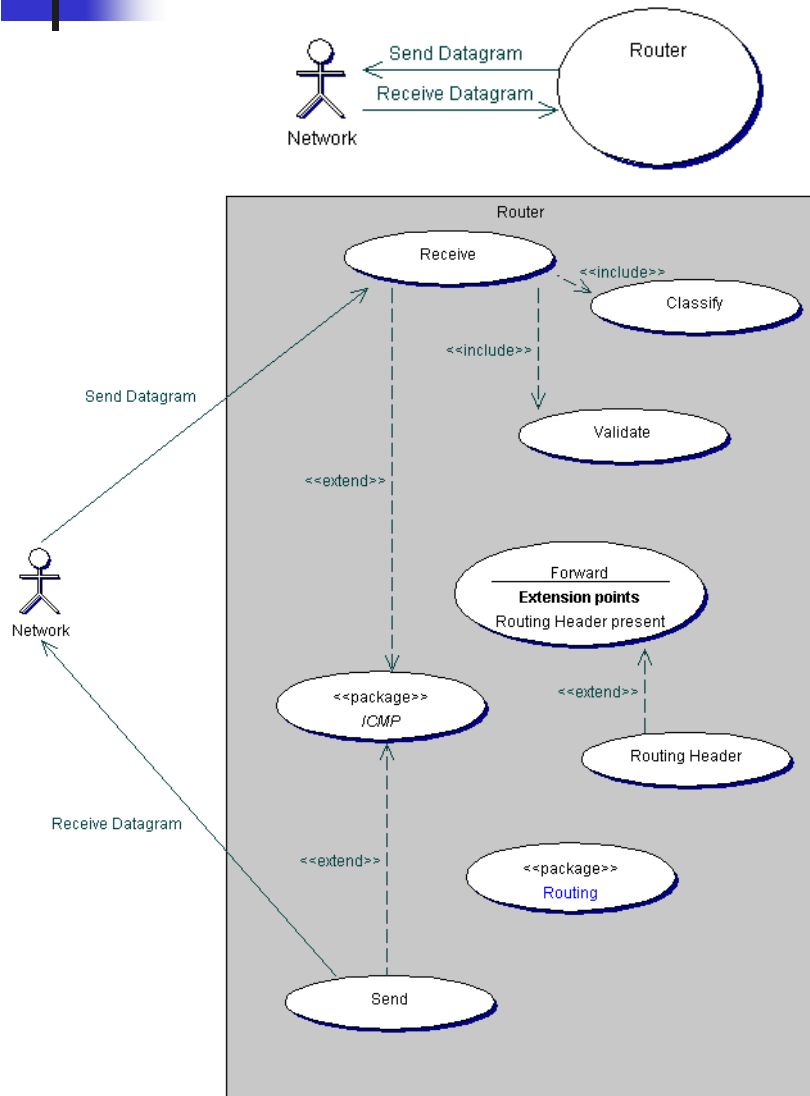
- IPv6 Router Requirements
 - **to design a 1 GB IPv6 router over Ethernet using the Routing Information Protocol (RIPng)**
- IPv6 overview
 - new generation of Internet Protocol
 - introduced to overcome addressing restrictions of IPv4
 - 128-bit address (32-bit for IPv4)
 - extensible datagram format
 - “simplified” header, new optional extension headers
 - 64-bit alignment
 - improved address hierarchy
 - security at IP level
- RIPng overview
 - interior gateway protocol
 - used in local networks
 - Distance Vector Protocol
 - metric to connected networks (1-16)
 - implemented at User Datagram Protocol (UDP) level

IPv6 Router Architecture



- Ethernet Line Cards
 - chosen from available products on the market
 - fully implement Ethernet protocol and other Link Layer protocols (ARP, RARP,...)
 - deal with assembly and en/decapsulation of datagrams
- Switching fabric
 - interfaces the line cards with TACO
 - dependent on the products used
- TACO processor
 - Stand-alone processor
 - implement router's functionality
 - forward messages (critical)
 - create and maintain routing table
 - handle error messages

Requirements Specification



- create Context Diagram
 - system border
 - external actors
 - communication between system and environment

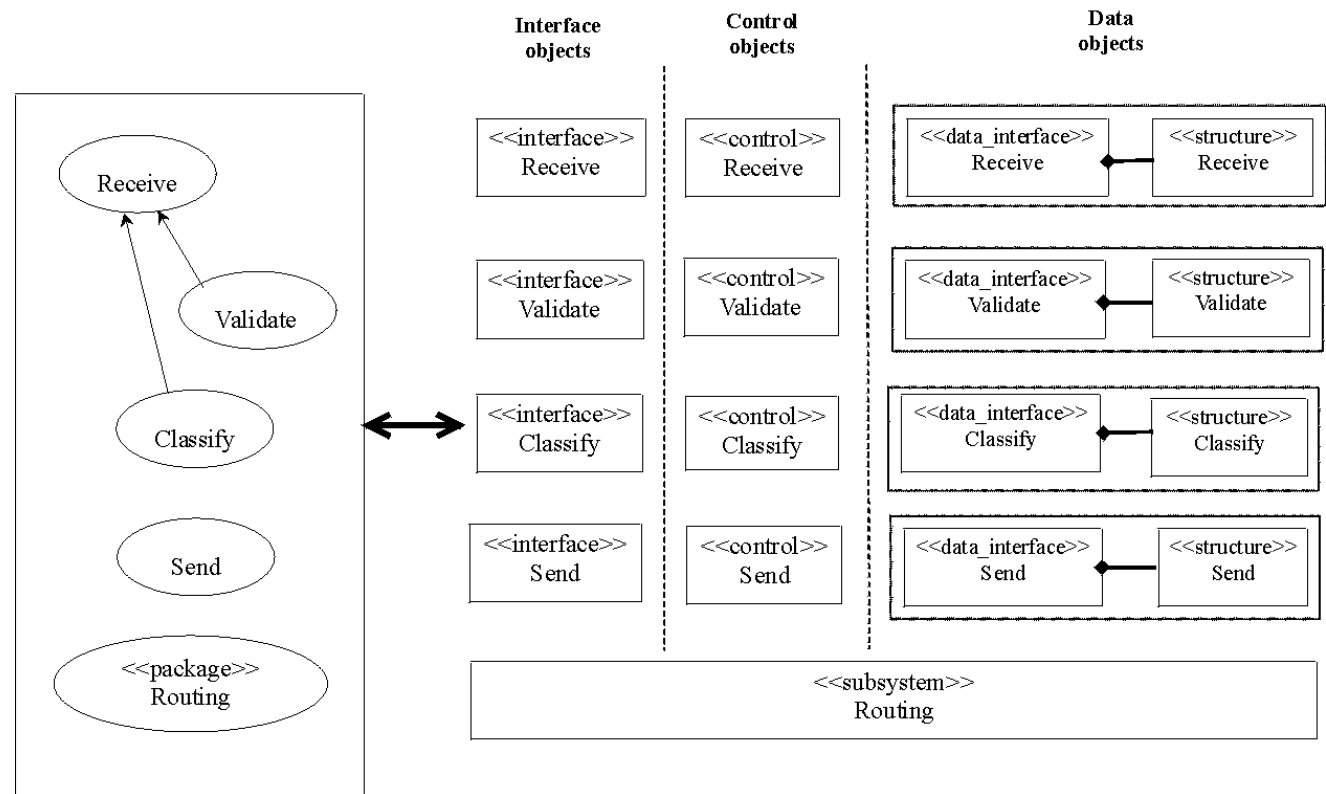
- identify pieces of functionality and create Use Cases
 - we specify both external and internal activities as use cases because both of them represent functionality the system has to implement
 - use case packages – complex functionality

- build Use Case Diagram
 - context diagram and use cases list as input

- update Context Diagram (if necessary)

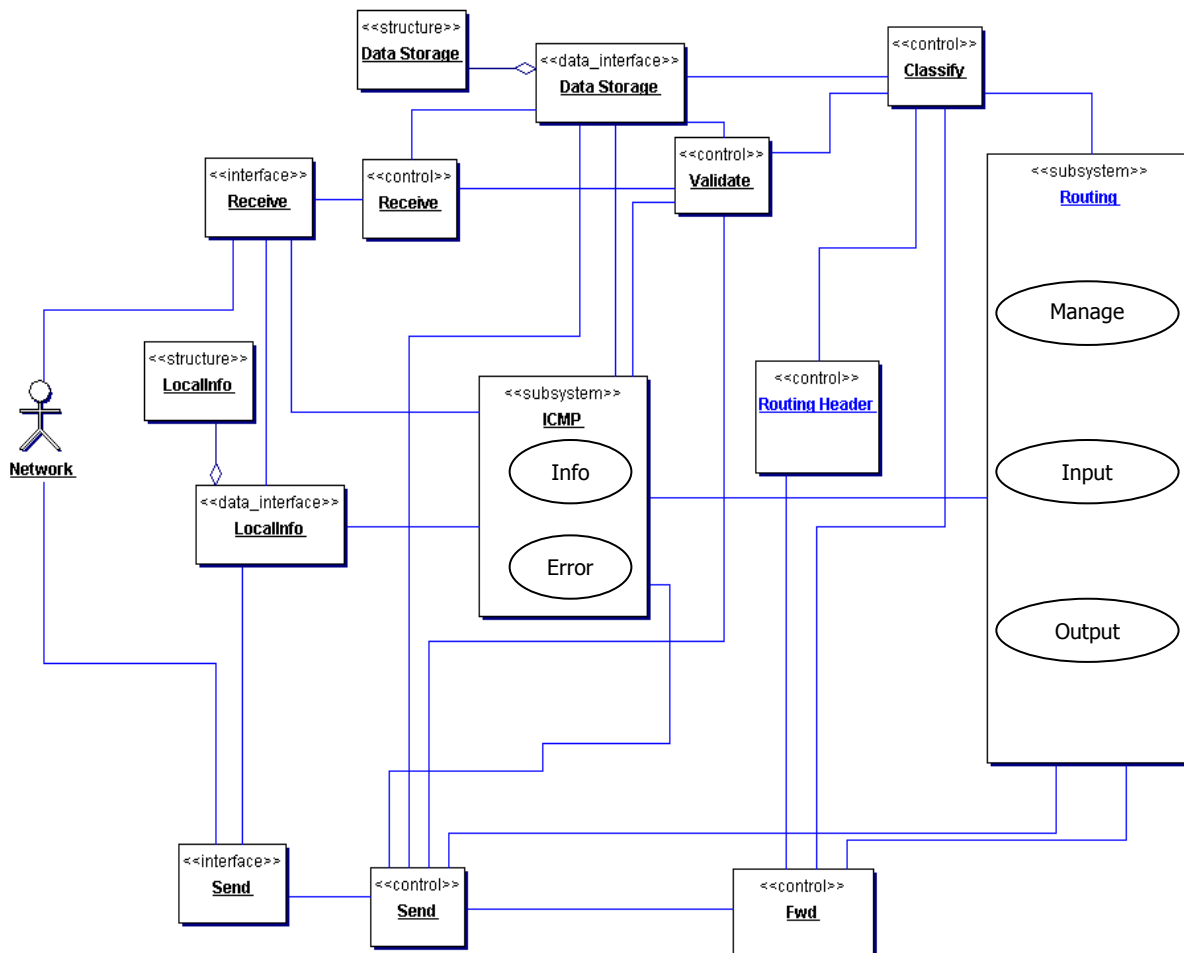
Object identification (1)

- objects are parts of the system that represent functionality
- we are more interested in identifying the objects of a system and then their classes
- each use case is split into three objects:
 - control
 - interface
 - data
 - interface (services for accessing data)
 - structure (physical implementation)



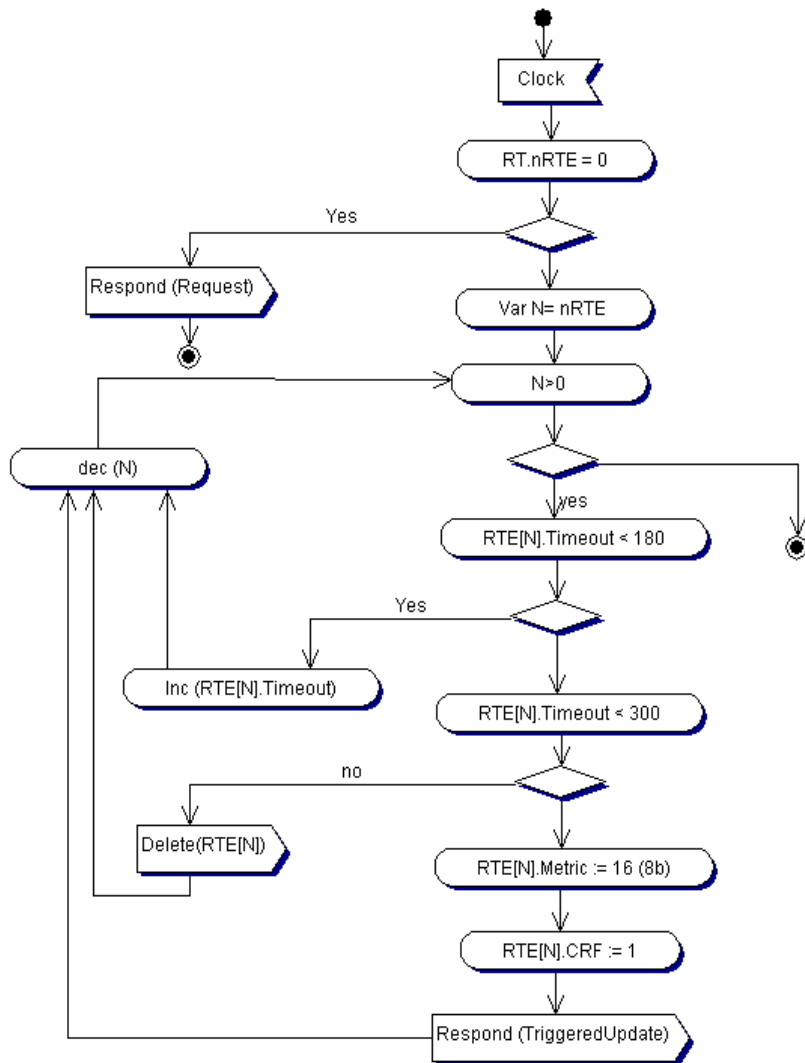
- use case packages are transformed into <<subsystem>> objects
- objects in the same category can be later refactored by splitting and joining

Object identification (2)



- apply recursively the same algorithm inside subsystems
- objects are external actors for the use cases inside subsystems
- group or split objects according to their functionality in the system
- instantiate scenarios and decide what objects are kept or disposed
- communication is depicted by association

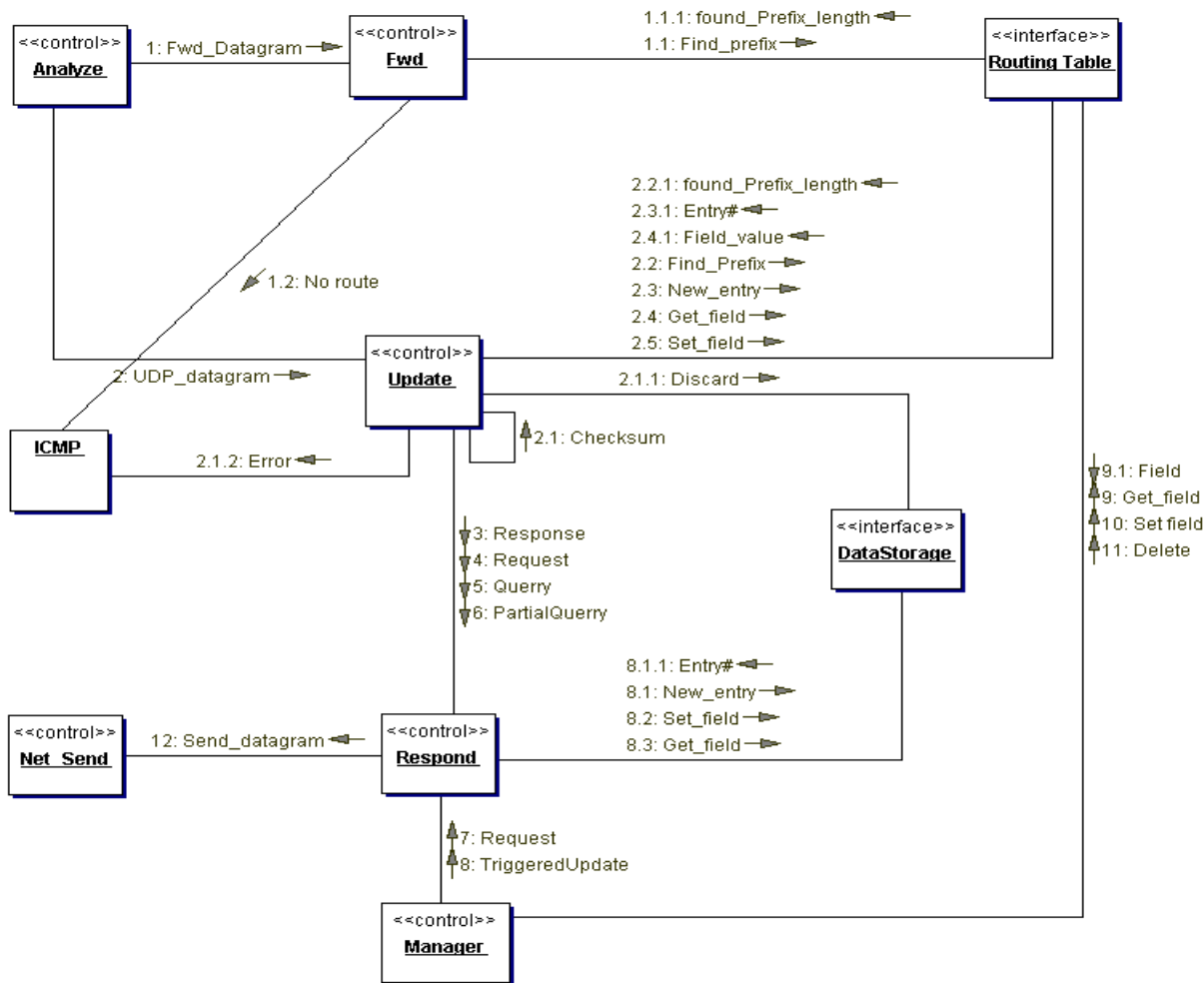
Objects' behavior – Activity diagrams



- specify flow of activities (states)
 - activity state - decomposable
 - action state – atomic
 - other artifacts: branch, receive and send events, transitions
- WHY activity diagrams
 - because we are addressing the area of protocol processing which requires data-intensive and algorithmic control-flow
 - hierarchy - activity states can be decomposed into another activity diagrams and/or action states
 - possible to navigate at different levels of detail
 - no strict language to describe activities

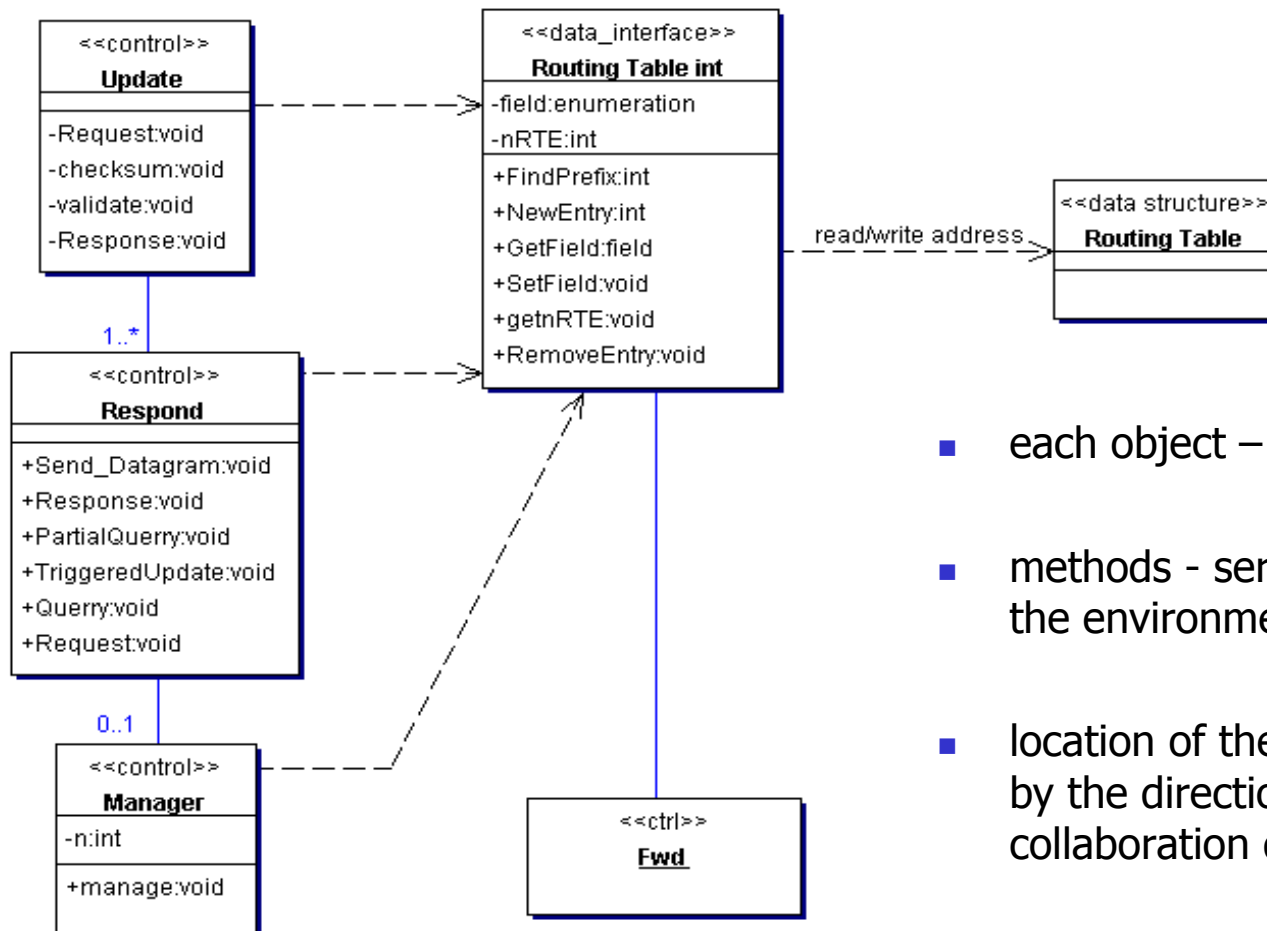
GOAL: to transform all *activity states* into *action states* directly implementable onto physical resources

Scenario – collaboration diagrams



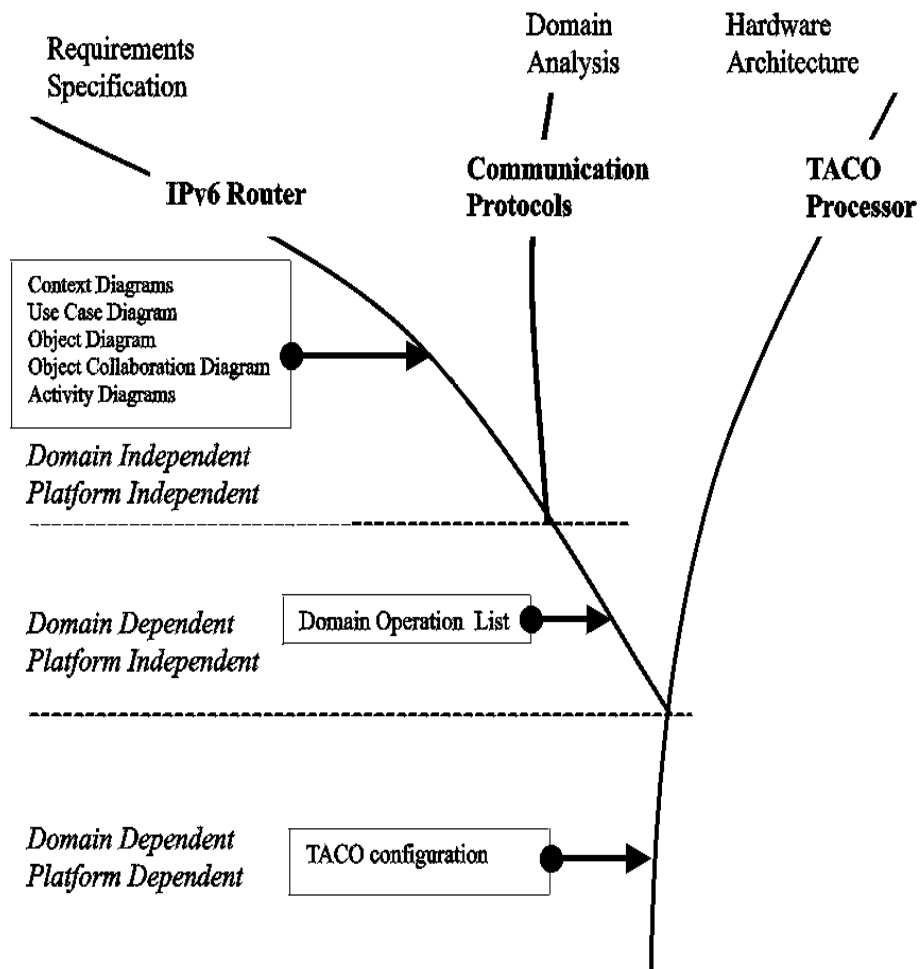
- apply existing scenarios for each use case
- an use case can have one or many scenarios
- communication is expressed as service requests
- Dewey decimal numbering scheme to show precedence of messages in time and their thread of execution (scenario)
- System-level Collaboration Diagram by overlapping all scenarios

Class Diagram



- each object – a class
- methods - services provided by a class to the environment
- location of the class methods determined by the direction of the messages in the collaboration diagram

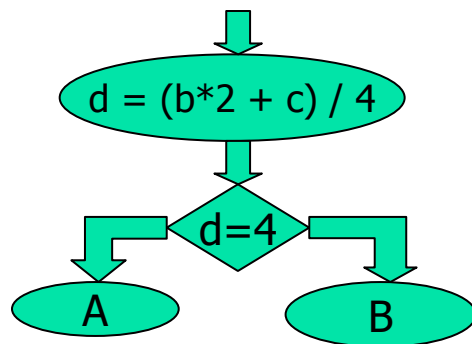
Domain Analysis – Operation List



- until now we have specified the application without any concerns for the physical platform architecture
- use a list of operations that is specific for a given domain of applications (protocol processing)
- Domain operations list
 - analyzing previous implementations in the same domain of application
 - independent of the hardware architecture
- operations in the list to be mapped onto operations provided by physical resources

Mapping Functional Spec. on Dom. Ops.

Domain & Platform Independent



Domain Operations List

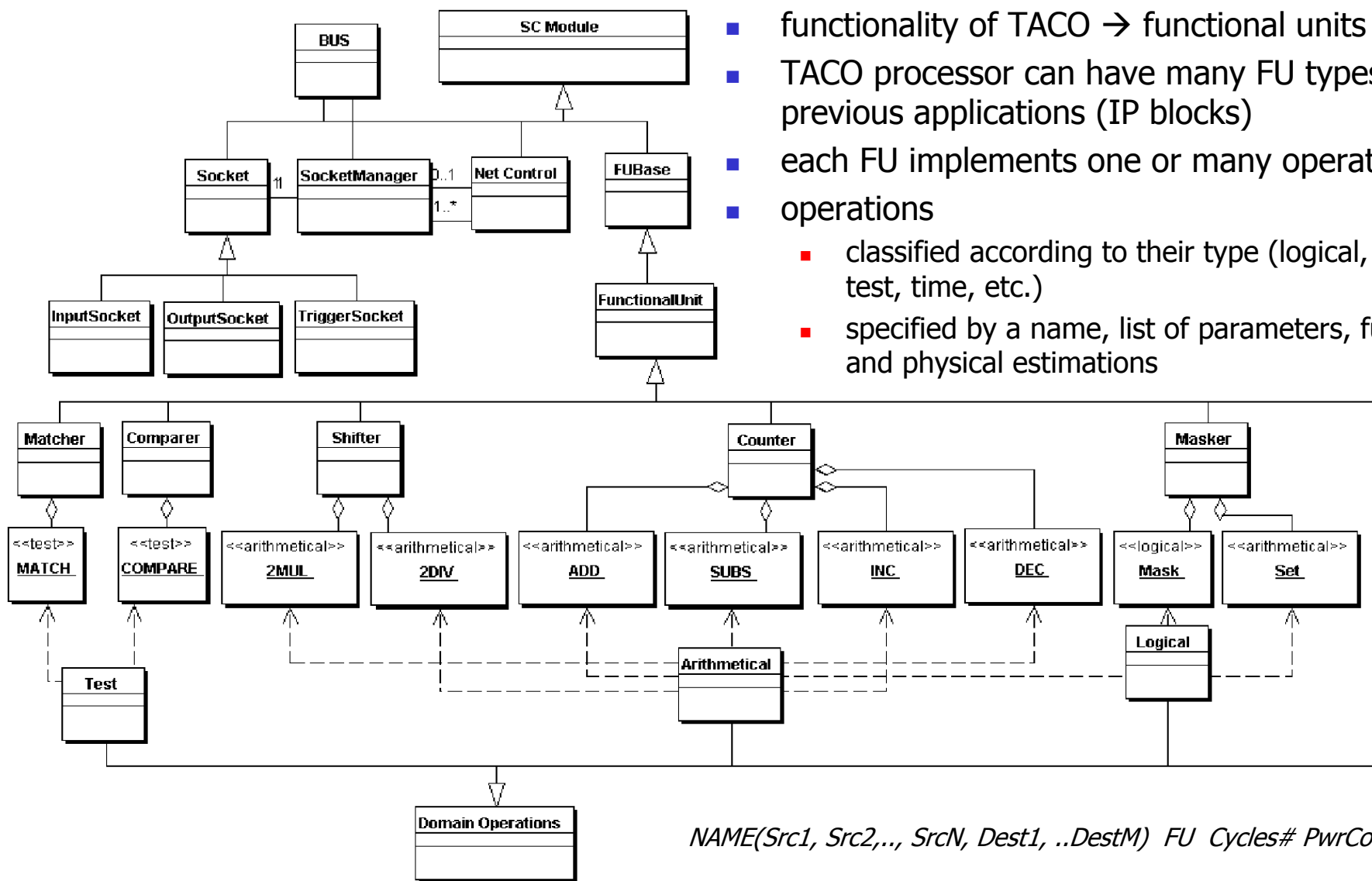
...
ADD(src1, src2, res)
MUL2(src, res)
DIV2(src, res)
SHL(src1, src2, res)
SHR(src1, src2, res)
SUB(src1, src2, res)
CMP(src1, src2, op, res)
JUMP dst;
SET (src1, src2, src3, res)
...

*Domain Dependent
Platform Independent*

```
...  
MUL2(b, R1)  
ADD(R1, c, R2)  
SHR(R2, 2, R3)  
CMP(R3, 4, "=", R4)  
R4 : JUMP A;  
!R4: JUMP B;  
...
```

- there can be several choices in selecting the domain operations
- based on designer's experience

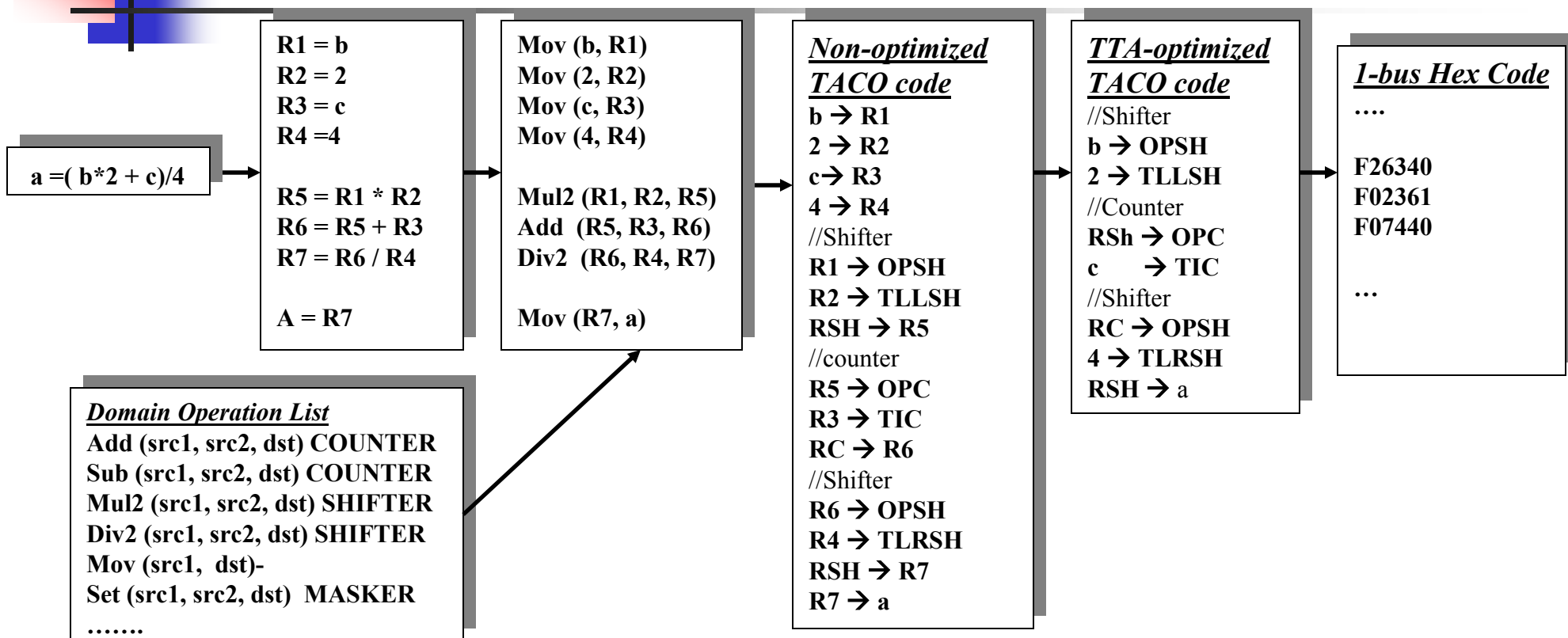
Mapping Domain Operations on TACO



- functionality of TACO → functional units
- TACO processor can have many FU types created in previous applications (IP blocks)
- each FU implements one or many operations
- operations
 - classified according to their type (logical, arithmetical, test, time, etc.)
 - specified by a name, list of parameters, functional unit and physical estimations

NAME(Src1, Src2,.., SrcN, Dest1, ..DestM) FU Cycles# PwrCons Area

Functional verification



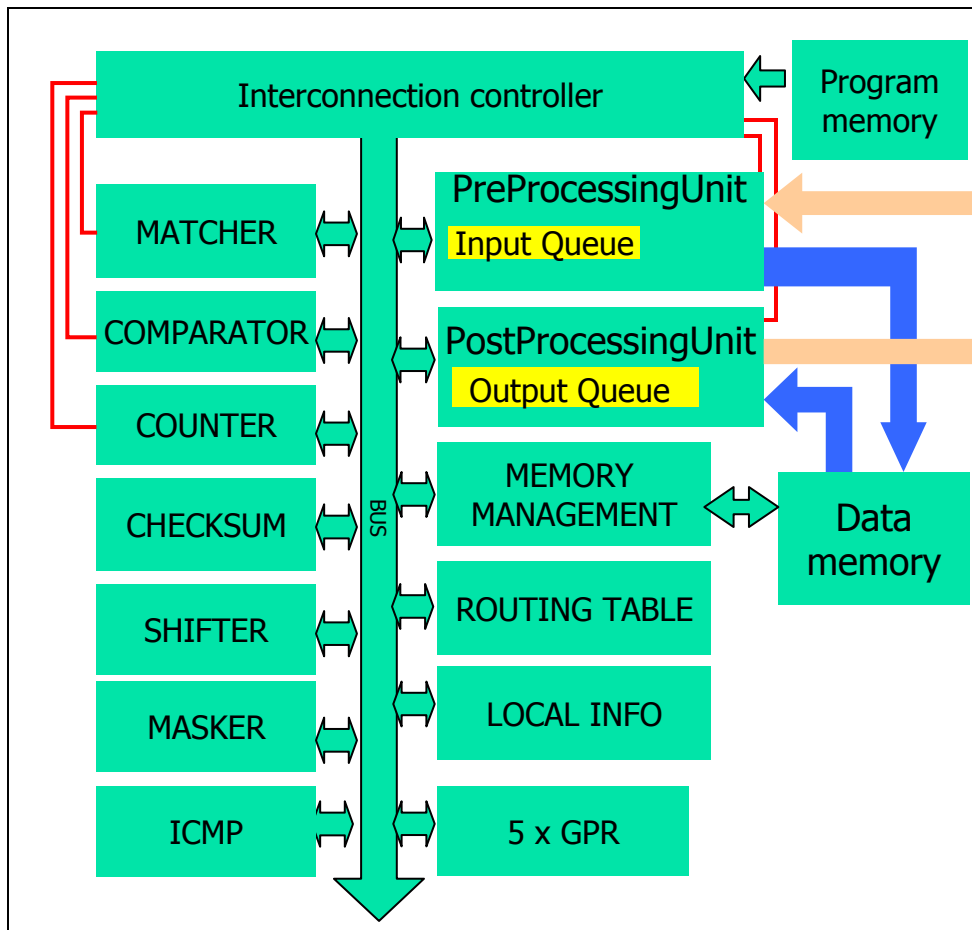
- create TACO instance with one resource of each type (Virtual Processor)
- simulate the architecture to verify its correct functionality
 - domain operations are transformed into TACO move operations
 - each domain operation → macro of TACO moves
 - generic registers are eliminated by applying TTA specific optimizations



IPv6 Router – Required Resources

- identified FU's (one clock cycle/operation)
 - Memory Management (read, write)
 - Checksum (32-bit computation)
 - Matcher (bitwise comparison of strings)
 - Comparator(equality, less then, greater then, ...)
 - Counter (addition, subtraction, counting up/down)
 - Shifter (logical and arithmetical, multiplication and division by power of 2)
 - Masker (sets a number of bits according to a given mask)
 - General Register Unit (GRU) - 5 registers
 - Routing Table Unit (provides access to the routing information)
 - Local Info Unit (fast access to local Information)
 - Input Preprocessing Unit (handles input traffic from the network)
 - Output Preprocessing Unit (handles output traffic to the network)

Proposed router architecture



TACO Processor (32-bit data bus):

- scans input buffers for incoming datagrams
- saves them in the main memory
- a datagram is uniquely identified by (MemAdd, RecInt, Length)
- datagrams stored in the memory are processed one at the time
- decides how to route datagrams between line cards
- headers are modified and datagram saved in the output buffer of the corresponding interface



Architectural limitations

- immediate integer generation depends on the number of buses
 - 1 bus – 8 bit immediate integer (picture)
 - 2 buses – 32+8 bit immediate integer (picture)
 - solution: immediate integers stored in the main memory
- loading of the program counter also limited by size of immediate integer size:
 - difficult to increment the Program Counter with large values
 - solution: large *absolute values* computed as repeated *relative values*



Simulation and Estimation

- Simulation
 - Early simulation of functionality using SystemC

- Estimation
 - each FU has clock cycle, area and power consumption estimations
 - by varying the number of FUs and the number of buses we can evaluate different processor configurations, with respect to:
 - required throughput
 - power consumption and area constraints
 - early identification of bottlenecks
 - by analyzing the program code we can identify sequences of TACO moves that require long computational time
 - we propose multiplication of resources accordingly
 - optimization/creation of new dedicated FU



Conclusions

- proposed an incremental UML design flow for TTA-based network processors
- suggested the configuration of the hardware platform by identifying required functionality of the application
- reuse of hardware components by applying OO-techniques and domain knowledge
- early system-level functional testing and estimation of the physical performance



Future work

- formalized description of the design flow
- adding parameterized real-time information during specification and analysis phases
- adding physical constraints during domain mapping phase
- improve domain analysis phase
- automate as many design steps as possible
- heuristics for suggesting a “close to optimal” quantitative configuration
- integrated tool for top-down design flow