«typedef» AccessPattern: vector<tuple<AccessType, unsigned int»

> «typedef» DevID: unsigned int

> > «typedef» Cost: unsigned int

«typedef» LinkID: unsigned int

«typedef» DevInfo: tuple<std::string, Cost, Cost, double, unsigned int>

«enumeration» AccessType FREE **BASIC EXPENSIVE**

next_id: DevID

ID: const DevID

BAC: const Cost

EAC: const Cost

veclen: unsigned int)

+ isNull(): bool

+ getID(): DevID

+ getName(): string

+ getCapacity(): double

+ getVectorLength(): unsigned int

NAME: const string

CAPACITY: const double

VECTOR LENGTH: const unsigned int

+ «constructor» Device(source: const Device&)

+ getBasicAccessCost(N: const unsigned int): Cost

+ getExpensiveAccessCost(N: const unsigned int): Cost

+ «constructor» Device(source: Device&&)

+ «constructor» Device(NULL: void*)

«enumeration» NetworkType

PART CONN GRAPH FULL CONN GRAPH STAR RING CART

Device

BasicCostModel Note # hardware: Hardware BasicCostModel defines trivial responses to these # known_data_layouts: map<string, DataLayout> queries. # defaultLayouts(): void It can be inherited from and query functions - «constructor» BasicCostModel(hw_info: const vector<DevInfo>&) overridden as we see fit. - «constructor» BasicCostModel(Hardware& hw) + getHardware(): Hardware& + addDataLayout(name: string, extent: unsigned int, ap: AccessPattern&): void + rmDataLayout(name: string): void + getDataLayout(NAME: const string): const DataLayout& + accessCost(DEV_ID: const DevID, LAYOUT: const DataLayout&, AP: const AccessPattern&, COUNT: const unsigned int): Cost + accessCost(DEV_ID: const DevID, LAYOUT: const DataLayout&, AP: const AccessPattern&, COUNT: const unsigned int, HARDWARE: const Hardware&): Cost + movementCost(DEV_SRC: const DevID, LAYOUT_SRC: const DataLayout&, DEV_DEST: const DevID, LAYOUT_DEST: const DataLayout&): Cost + movementCost(DEV_SRC: const DevID, LAYOUT_SRC: const DataLayout&, DEV_DEST: const DevID, LAYOUT_DEST: const DataLayout&, hardware: Hardware&): Cost + movementDecision(DEV_SRC: const DevID, LAYOUT_SRC: const DataLayout&, DEV_DEST: const DataLayout&, AP: const AccessPattern&, COUNT: const unsigned int): bool - movementDecision(DEV SRC: const DevID, LAYOUT SRC: const DataLayout&, DEV DEST: const DataLayout&, AP: const AccessPattern&, COUNT: const unsigned int, hardware: Hardware&): bool + recommendDevice(LAYOUT: const DataLayout&, AP: const AccessPattern&, COUNT: const unsigned int): DevID - recommendDevice(LAYOUT: const DataLayout&, AP: const AccessPattern&, COUNT: const unsigned int, HARDWARE: const Hardware&): DevID Ý 0..1 uses» 0..1 Hardware Access num_devices: unsigned int PATTERN: const AccessPattern - DATA LAYOUT: const DataLayout devices: vector<Device> - COUNT: const unsigned int topo: Topology - NULLDEV: const Device unrollAccessPattern(IN_PATTERN: const AccessPattern&, LAYOUT: const DataLayout&): const AccessPattern + «constructor» Hardware(device_info: const vector<DevInfo>&) + «constructor» Access(PATT: AccessPattern&, TYPE: const DataLayout&, count = 1: unsigned int) + «constructor» Hardware(device_info: const vector<DevInfo>&, net_type: NetworkType) + «constructor» Hardware(device_info: const vector<DevInfo>&, old_hw: Hardware&) + «constructor» Hardware(device_info: const vector<DevInfo>&, old_hw: Hardware&, + getReps(): unsigned int + begin(): AccessPattern::const_iterator net_type: NetworkType) + end(): AccessPattern::const_iterator + getDeviceName(DEV_ID: const DevID): string + getNumDevices(): unsigned int + getDevice(DEV_ID: const DevID): const Device& + getDevices(): const vector<Device>& + getTopology(): const Topology& DataLayout NAME: const string EXTENT: const unsigned int PATTERN: const AccessPattern **Topology** + «constructor» DataLayout(name: const string, extent: const unsigned int, ap: const AccessPattern&) NETWORK_TYPE: const NetworkType - topo_graph: lemon::ListGraph getName(): string - topo_devs: lemon::ListGraph::NodeMap<DevID> + getExtent(): unsigned int - topo nodes: unordered map<DevID, lemon::ListGraph::Node> + getPattern(): const AccessPattern& - topo links: lemon::ListGraph::EdgeMap<Link> - topo_edges: unordered_map<LinkID, lemon::ListGraph::Edge> reserveEdge(num_devices: unsigned int. type: NetworkType); void + «constructor» Device(name: string, bac: Cost, eac: Cost, cap: double, + «constructor» Topology(num_devices: const unsigned int, type = PART CONN GRAPH: const NetworkType) + «constructor» Topology(num_devices: const unsigned int, dev_vec: const vector<DevID>&, Link type = PART CONN GRAPH: const NetworkType) + «constructor» Topology(num_devices: const unsigned int, old_topo: const Topology&) - link id: LinkID + «constructor» Topology(num_devices: const unsigned int, old_topo: const Topology&, latency: Cost type: const NetworkType) - inverse bw: Cost + «constructor» Link() + getNetworkType(): NetworkType + «constructor» Link(lat: Cost, inverse_bw: Cost) + getNumDevices(): unsigned int + getNumLinks(): unsigned int 1..* + operator+=(RHS: const Link&): Link& + addDevice(DEV ID: const DevID): void + «friend» operator+(lhs: Link, RHS: const Link&): Link + addDevice(DEV_VEC: const vector<DevID>&): void + setLinkID(A: const DevID, B: const DevID): void + removeDevice(DEV_ID: const DevID): void + getLinkID(): LinkID + removeDevice(DEV_VEC: const vector<DevID>&): void + getLatency(): Cost + setLink(IDA: const DevID, IDB: const DevID, link: Link): void + getInverseBW(): Cost + unsetLink(IDA: const DevID, IDB: const DevID): void + linkExists(IDA: const DevID, IDB: const DevID): bool + routeExists(IDA: const DevID, IDB: const DevID): bool

+ getMostDirectRoute(IDA: const DevID, IDB: const DevID): vector<DevID>

+ getHighestBWRoute(IDA: const DevID, IDB: const DevID): vector<DevID>

+ getLowestLatencyRoute(IDA: const DevID, IDB: const DevID): vector<DevID>