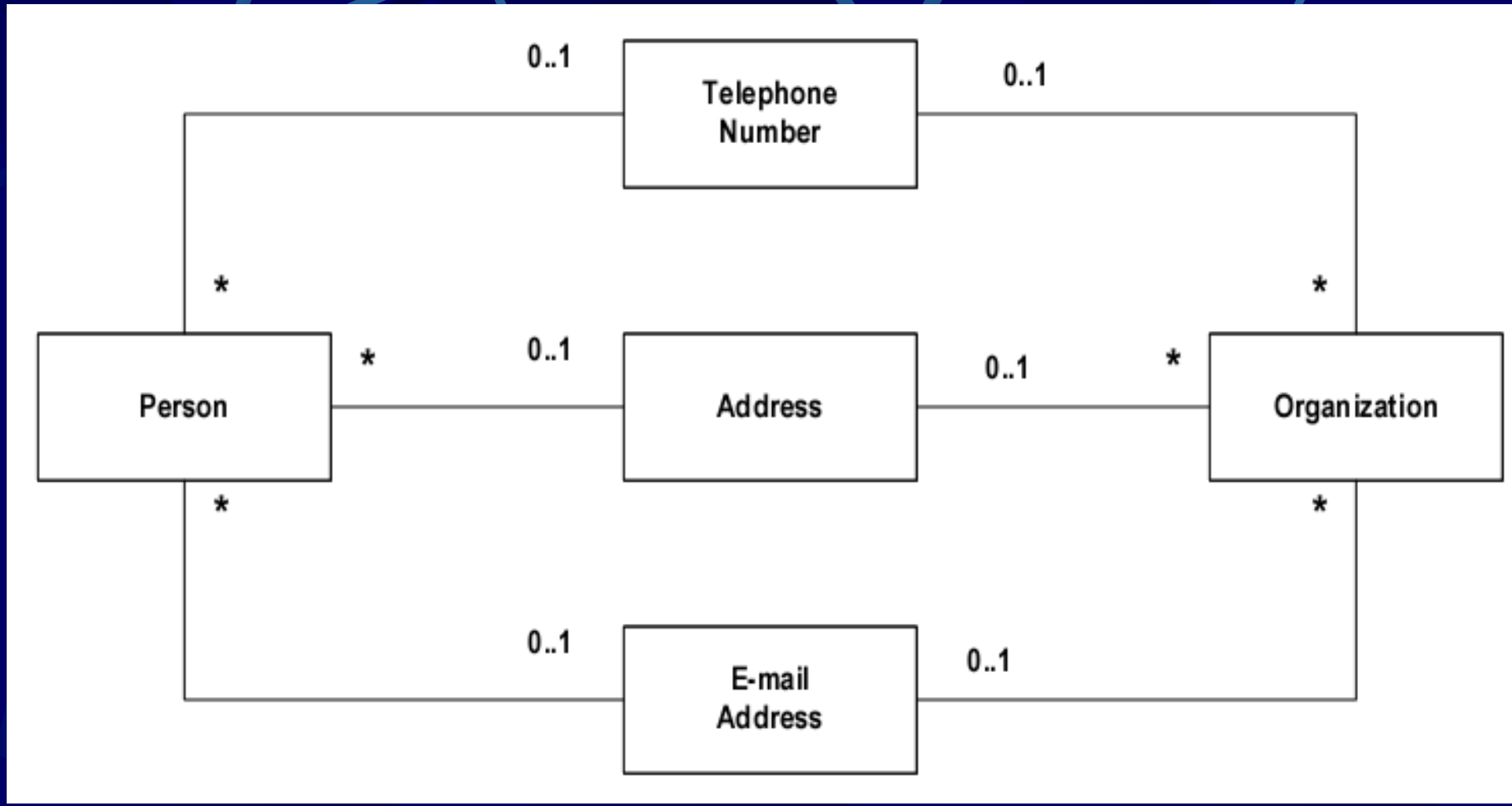
责任模式
Accountability

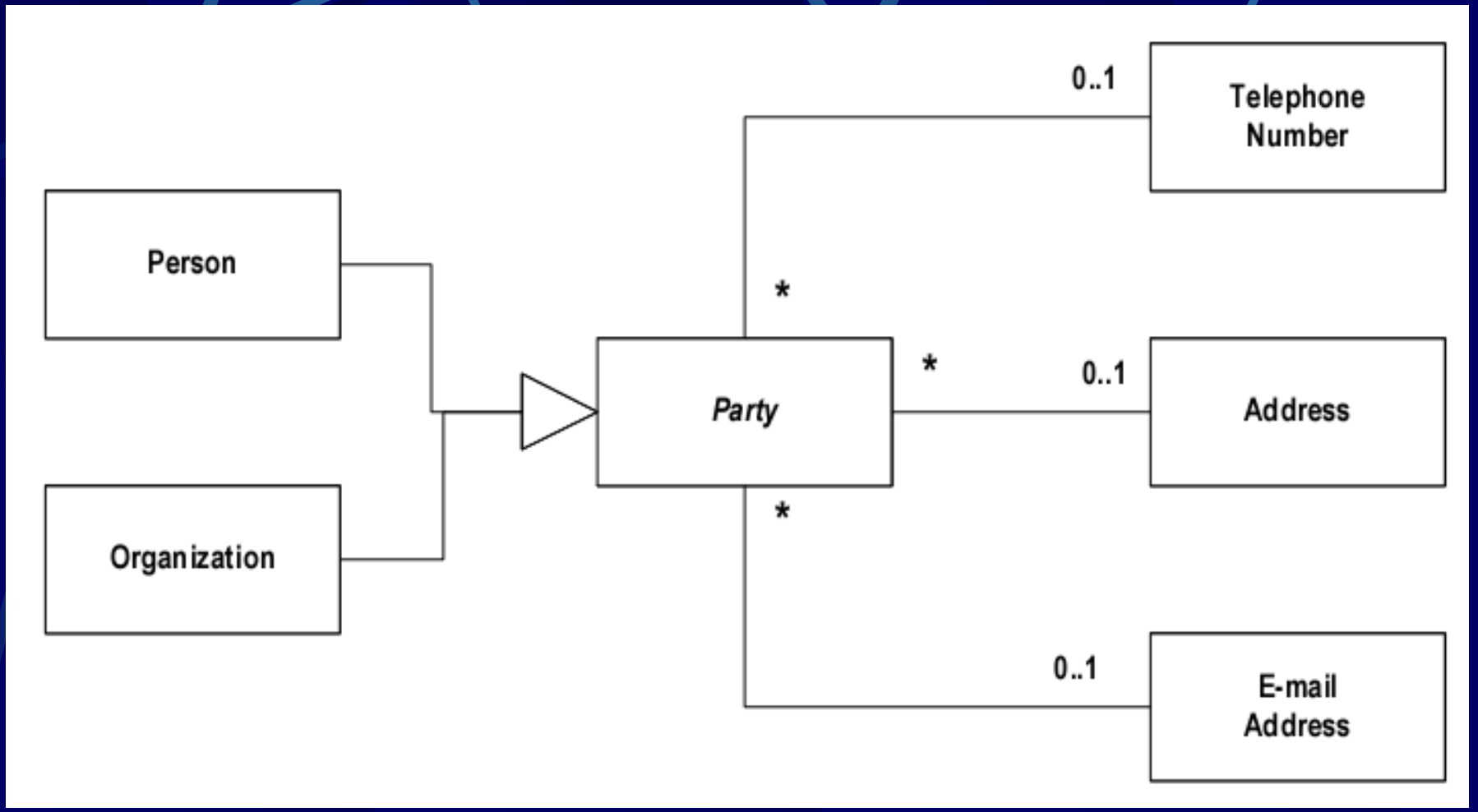
徐迎晓
复旦大学软件学院
xuyingxiao@126.com

1. Party

Example:

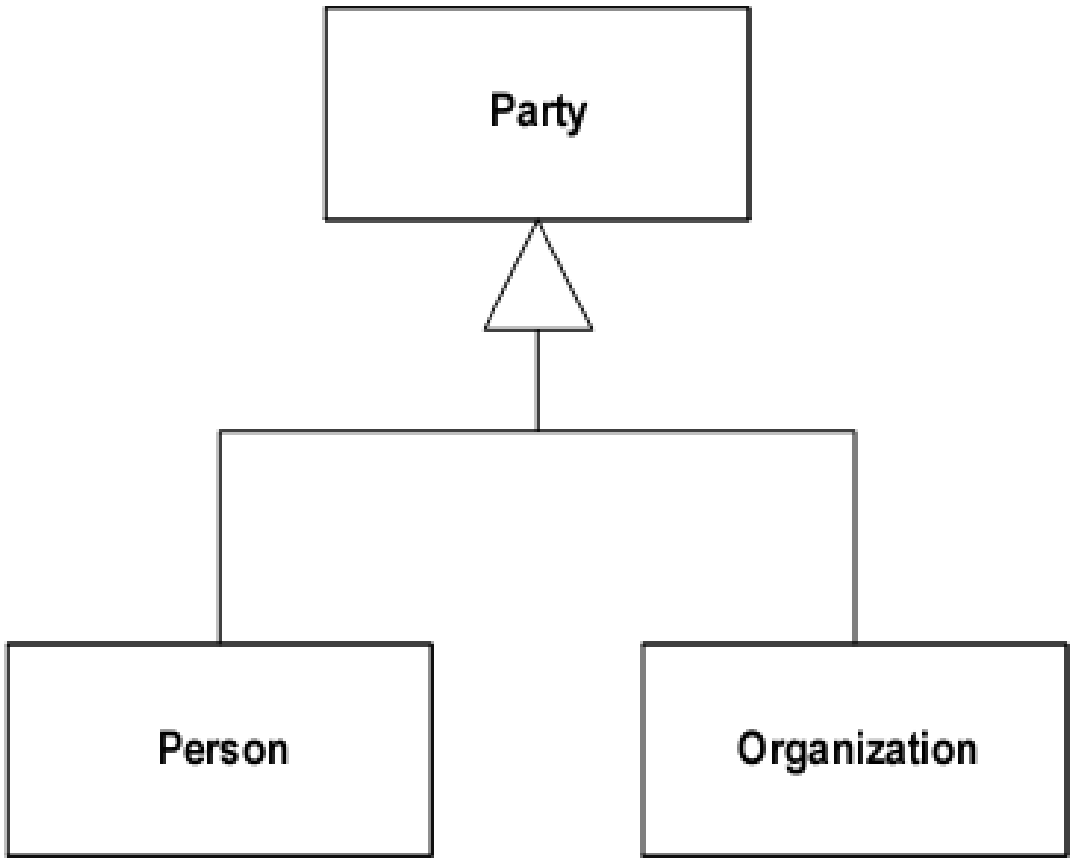
- A telephone utility's customers may be individuals or businesses. Many aspects of dealing with customers are the same, in which case they are treated as parties. Where they differ they are treated through their subtype


0..1
Telephone Number
0..1
*
*
Person
*
0..1
Address
0..1
*
Organization
*
*
0..1
E-mail Address
0..1


0..1
Telephone Number
Person
*
*
0..1
Party
Address
Organization
*
0..1
E-mail Address

Party Pattern

An abstraction of people and organizational units



- Put any behaviror that is common to people and organizational units on Party, only put things particular to one or the other on the subtype
- When you put behavior on the subtype, think about whether it makes sense on the supertype

When to use it

- when you have people and organizations in your model and you see common behavior
- when you don’t need to distinguish between people and organizations. In this case it’s useful just to define a party class and not to provide the subtypes

Main Point

- The main point of this pattern is to look for it to see if you have common behavior, and if so to use the name Party for that supertype.
- The name has become quite widely used these days, so choosing that name helps in communication

2. Organization Hierarchy

Example

- Imagine a company where people work in departments, which are organized into divisions.

- An explicit, and obvious,organizational structure.

- each part of the structure is a separate class.

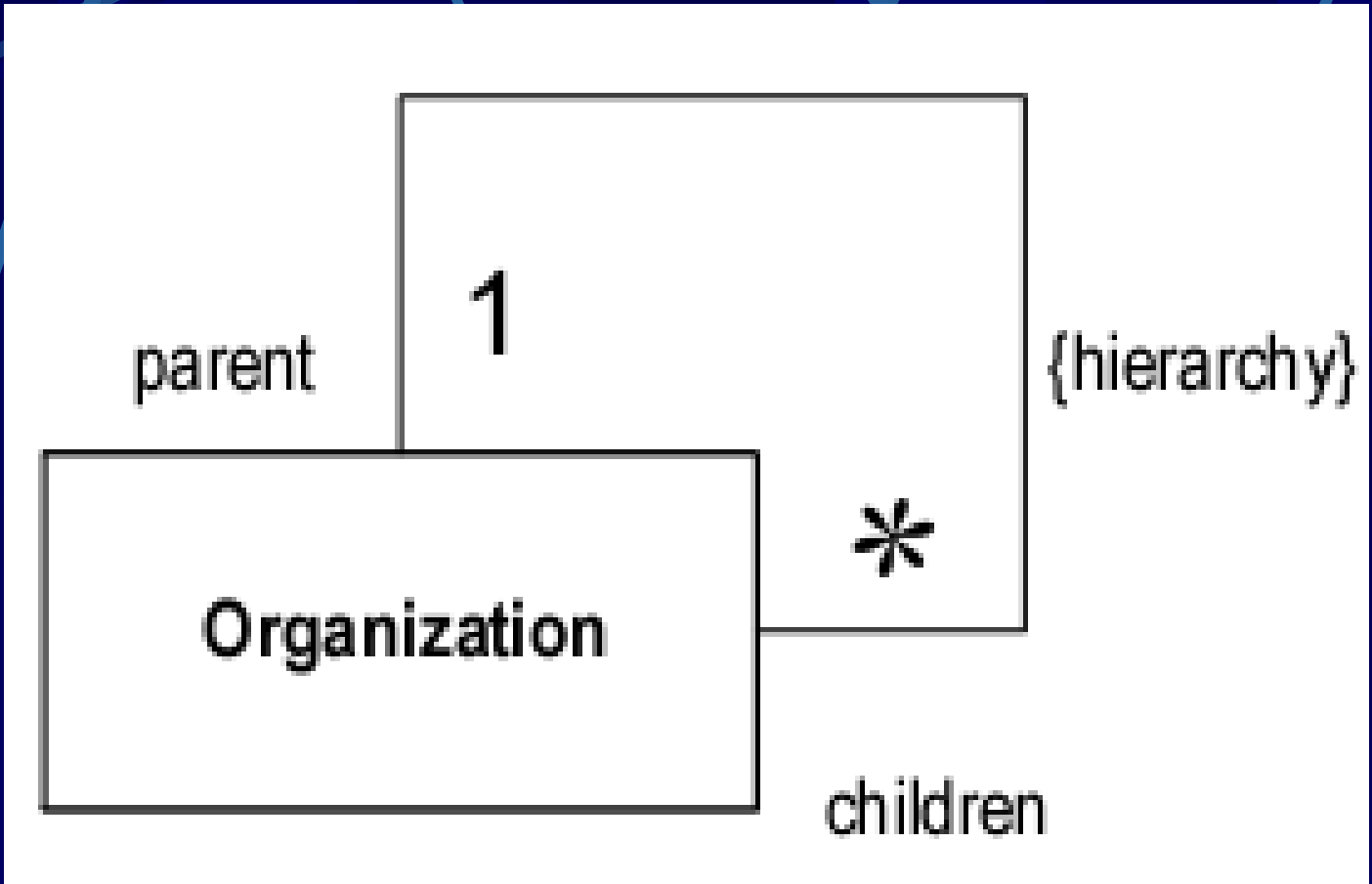
Disadvantages

- They don’t work well if there is much common behavior between the kinds of organization.
- Add regions between divisions and departments?

Solution

- Making a supertype for the organization

Organization hierarchy Pattern



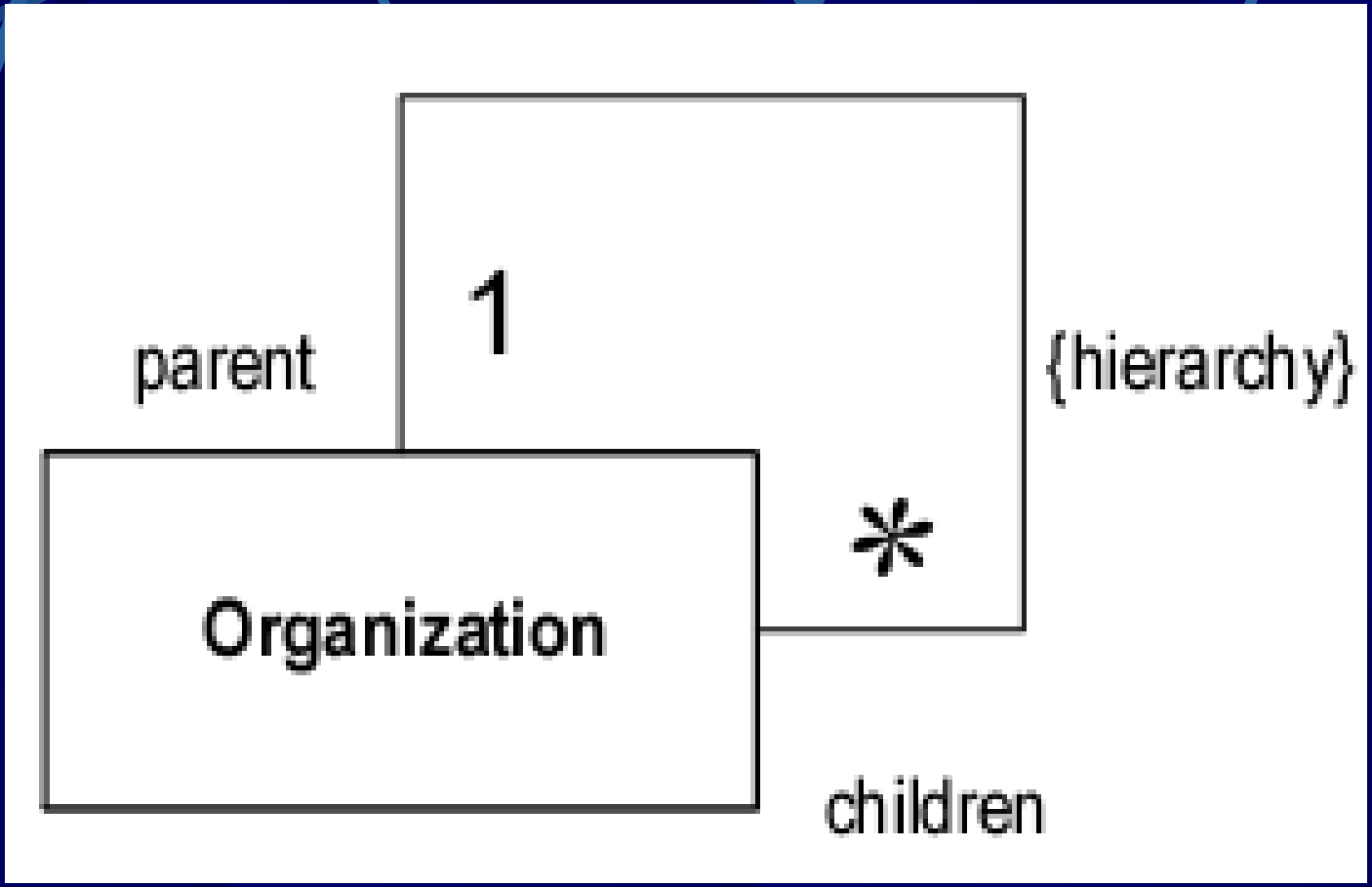

Associated Coffee Makers: Organization
sales : Organization
purchasing : Organization
Boston sales : Organization
London sales : Organization

- The organization hierarchy works best when you don’t have much different behavior between the organization sructures
- It also allows you to stay very flexible if new kinds of organizations appear.

When to use it

- Hierarchic
  - because the pattern handles a hierarchy not anything more complex.If your needs are more complicated you can tweak the pattern or use Accountability
- Affects the software
  - you only need to capture the corporate organization if it really affects what you are doing

Sample Implementation



```
class Organization ...
  private static Map instances = new HashMap();
  private String name;
  void register () {
      instances.put(name, this);
  }
  static void clearRegistry() {
      instances = new HashMap();
  }
  static Organization get(String name) {
      return (Organization) instances.get(name);
  }
```

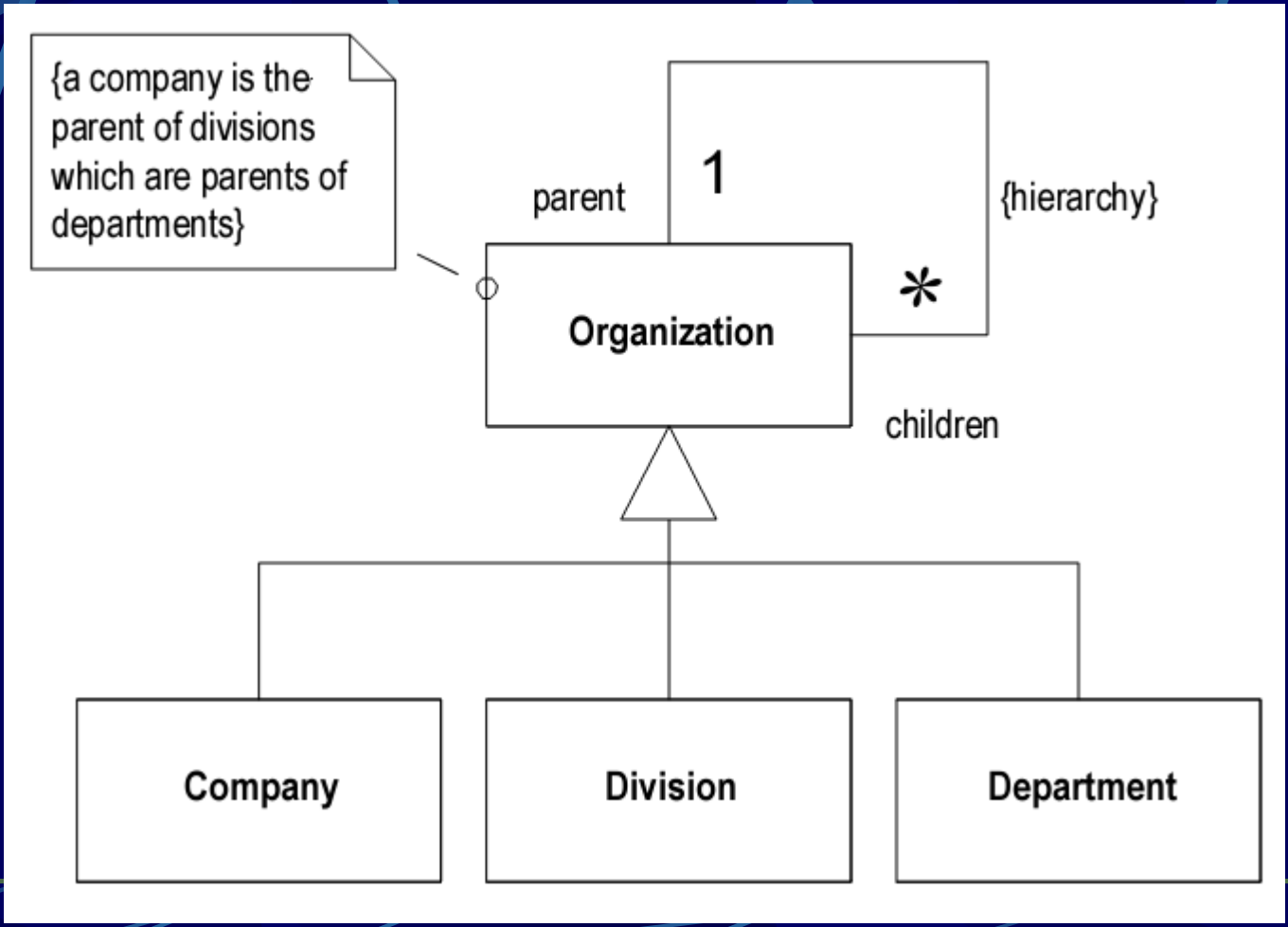
```
class Organization ...
  private Organization parent;
  Organization (String name, Organization parent) {
       this.name = name;
       this.parent = parent;
  }
  Organization getParent() {
       return parent;
  }
  Set getChildren() {
       Set result = new HashSet();
       Iterator it = instances.values().iterator();
       while (it.hasNext()) {
            Organization org = (Organization) it.next();
            if (org.getParent() != null)
               if (org.getParent().equals(this)) result.add(org);
       }
       return result;
  }
```

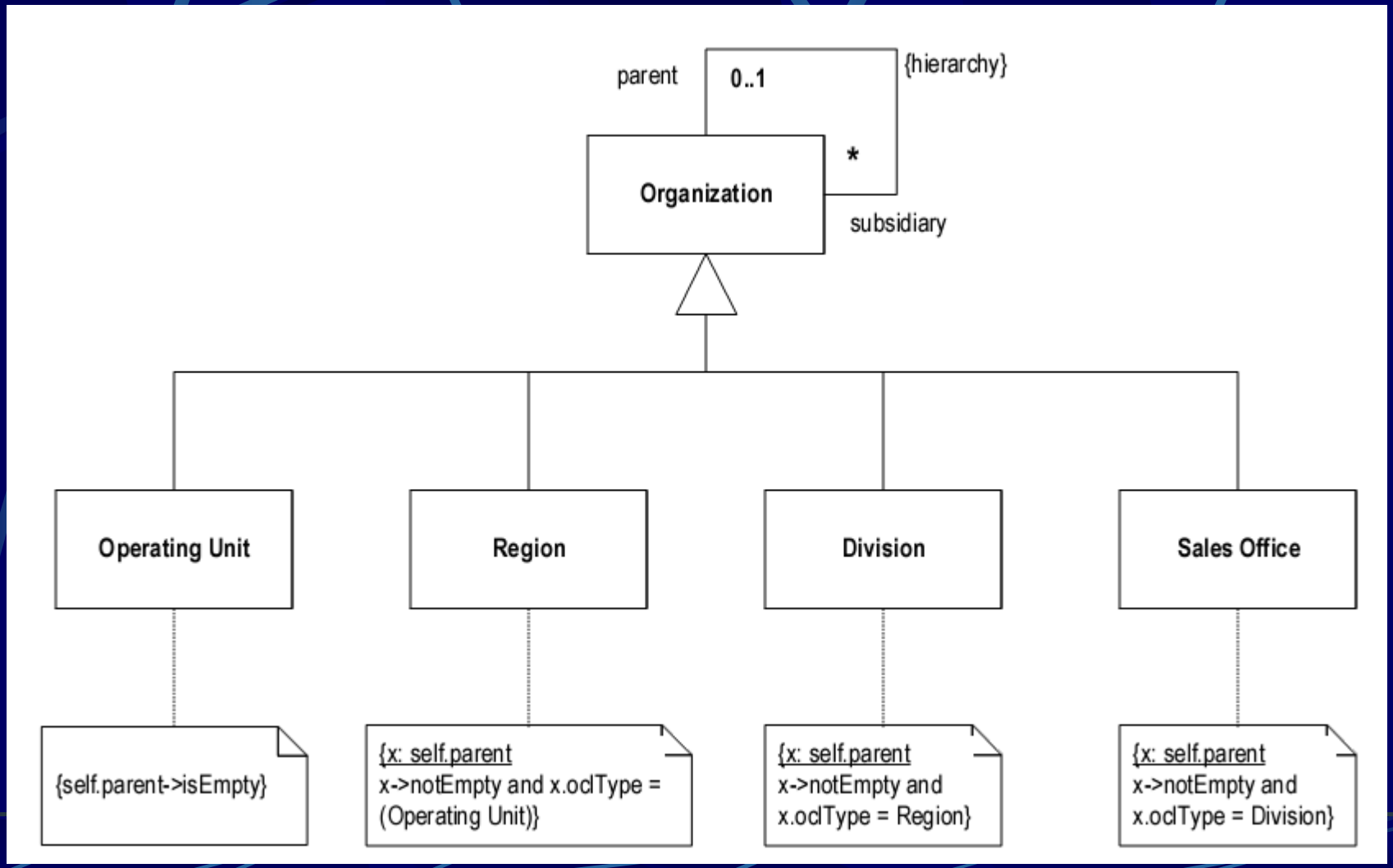
```
public Set getAncestors() {
    Set result = new HashSet();
    if (parent != null) {
        result.add(parent);
        result.addAll(parent.getAncestors());
    }
    return result;
}

public Set getDescendents() {
    Set result = new HashSet();
    result.addAll(getChildren());
    Iterator it = getChildren().iterator();
    while (it.hasNext()) {
        Organization each = (Organization) it.next();
        result.addAll(each.getDescendents());
    }
    return result;
}

public Set getSiblings() {
    Set result = new HashSet();
    result = getParent().getChildren();
    result.remove(this);
    return result;
}
```

```
void setParent(Organization arg) {
    assertValidParent(arg);
    parent = arg;
}
void assertValidParent (Organization parent) {
    if (parent != null)
        Assert.isFalse(parent.getAncestors().contains(this));
}
```

3. Variation: Subtypes for Levels

Variation: Subtypes for Levels




parent
0..1
{hierarchy}
*
Organization
subsidiary
Operating Unit
Region
Division
Sales Office
{self.parent->isEmpty}
{x: self.parent
x->notEmpty and x.oclType =
(Operating Unit)}
{x: self.parent
x->notEmpty and
x.oclType = Region}
{x: self.parent
x->notEmpty and
x.oclType = Division}

Sample Implementation

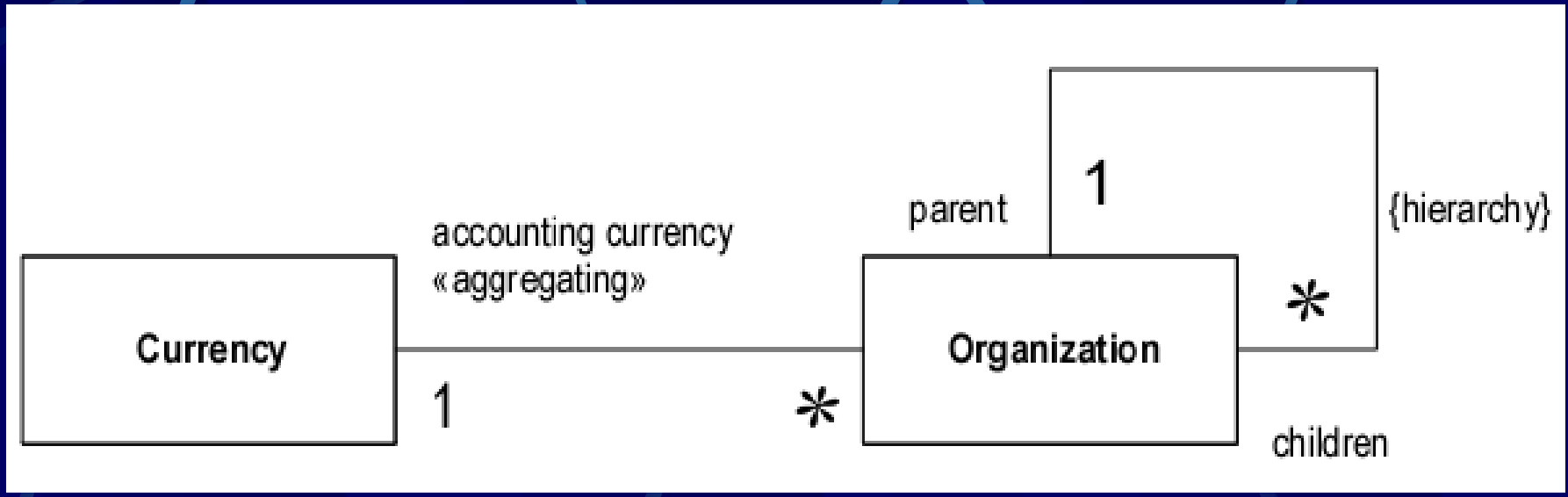
```
class Division extends Organization ...
  void assertValidParent (Organization parent) {
      Assert.isTrue(parent instanceof Company);
      super.assertValidParent(parent);
```

```
class Division...
  Division (String name, Company parent) {
      super(name, parent);
  }
```

4. Aggregating Attribute

- the head company, the sales division, and the Boston sales group all use US dollars as their accounting currency.

Aggregating Attribute



When to use it

- The key question to ask is whether a change in a value for a parent should affect all the children.
  - If the parent provides a default which is adopted by a child but subsequent changes to the parent do not change children with the same value, then that's not Aggregating Attribute

Sample Implementation

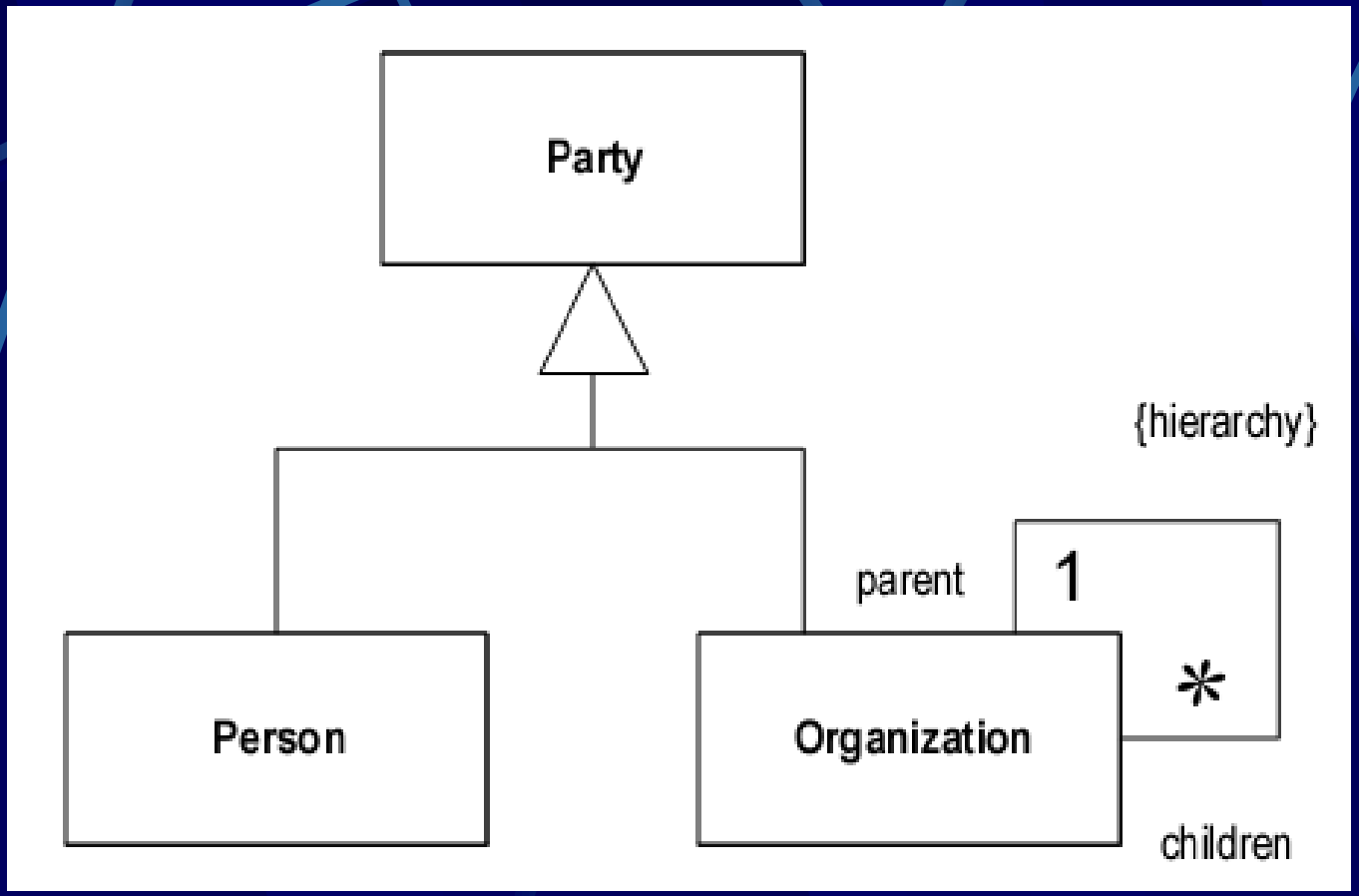
```
class Organization...
  Currency getAccountingCurrency() {
      if (accountingCurrency != null)
          return accountingCurrency;
      else {
          if (parent != null)
              return parent.getAccountingCurrency();
          else {
              Assert.unreachable();
              return null;
          }
      }
  }
```

```
void setAccountingCurrency(Currency arg) {
    assertValidAccountingCurrency(arg);
    accountingCurrency = arg;
}

void assertValidAccountingCurrency(Currency arg) {
    Assert.isFalse (arg == null && getParent() == null);
}
```

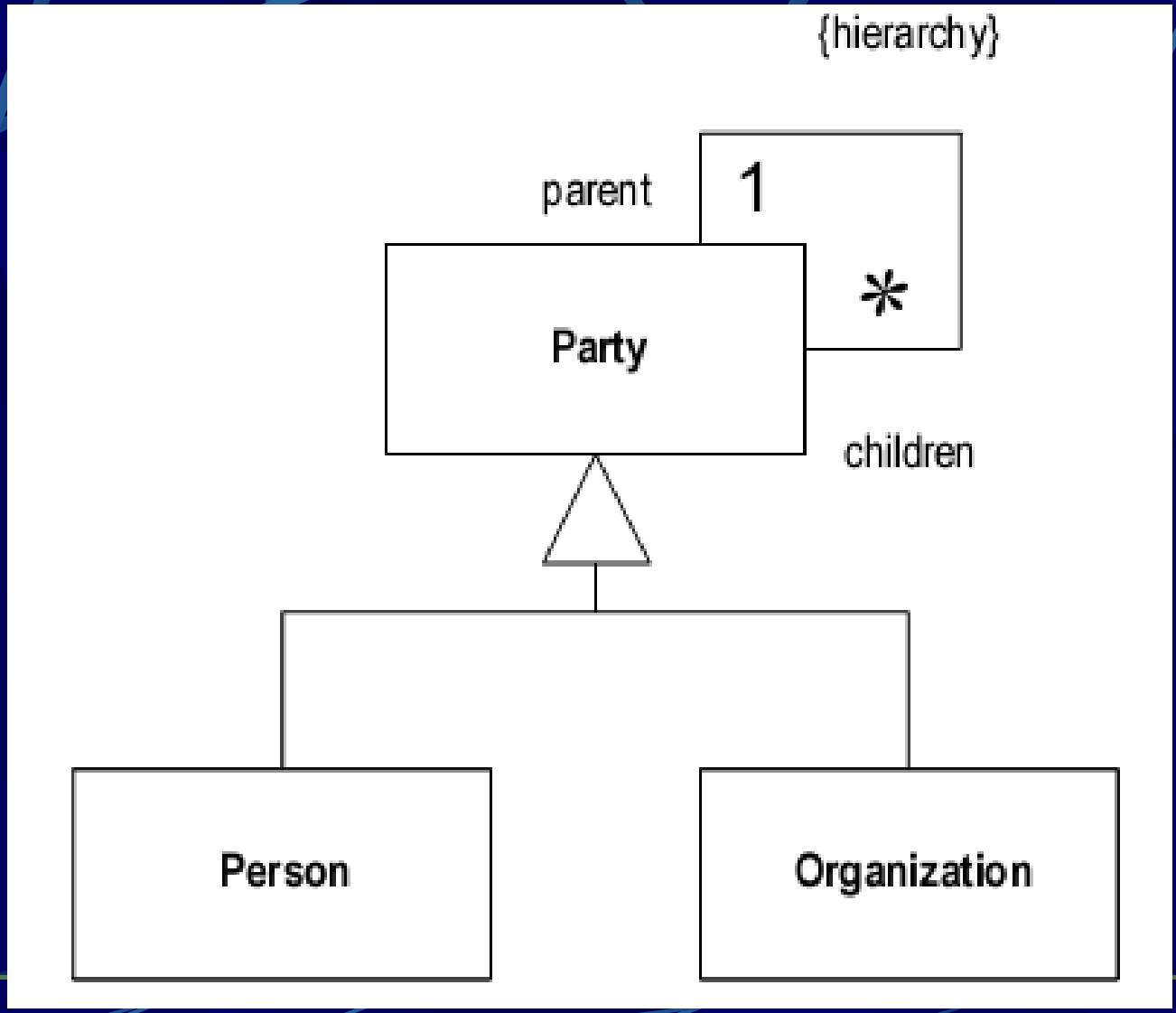
5. Accountability

.1 Party Further

- a supertype between the organization and person

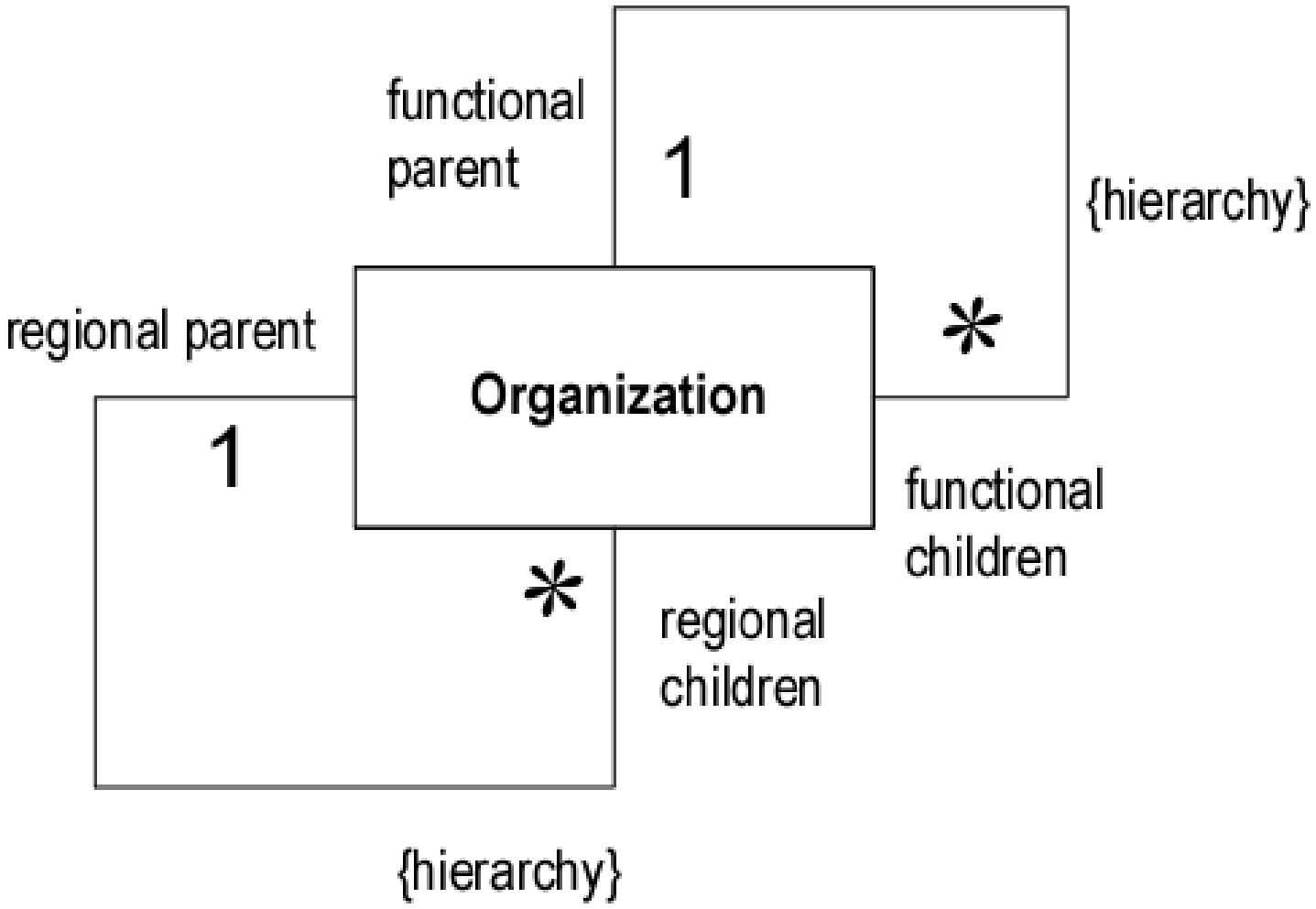

Party
{hierarchy}
parent
1
*
Person
Organization
children

A hierarchy on Party



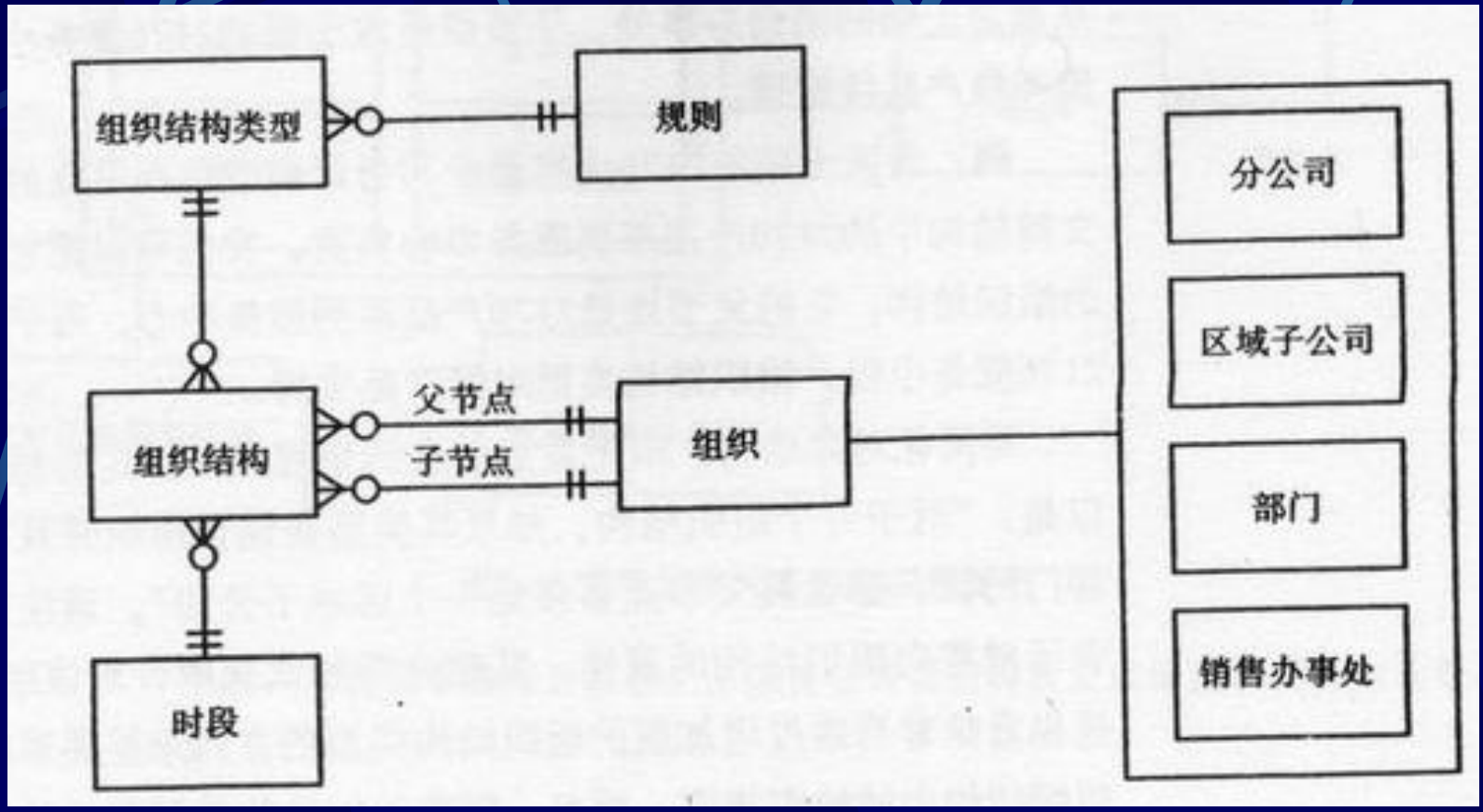
.2 different relationships between parties

- regional vs functional management
  - Sales in the London office have sales as its functional parent
  - Sales in the London office have the London office as its regional parent

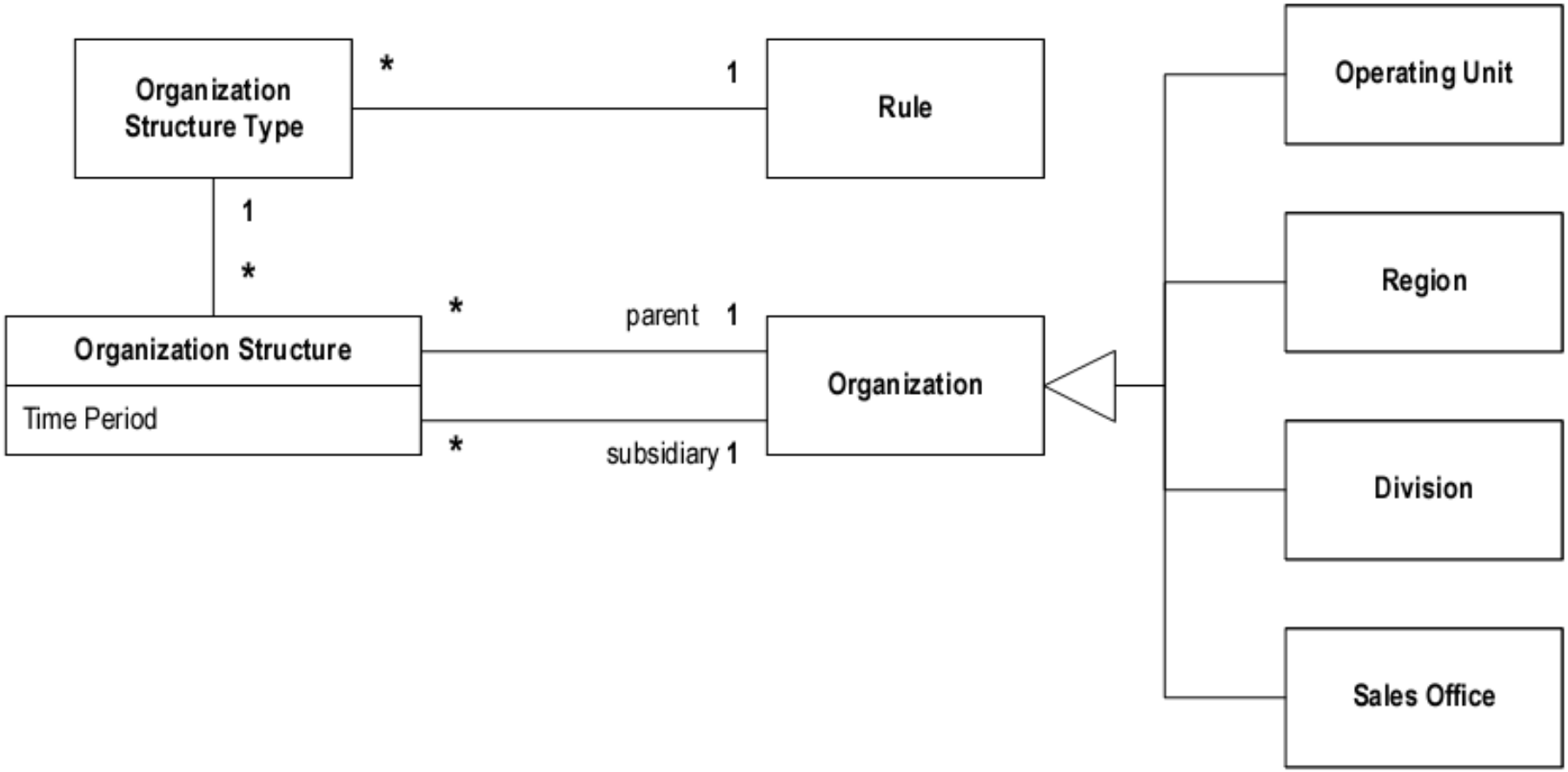

functional
parent
1
{hierarchy}
regional parent
Organization
*
1
functional
children
*
regional
children
{hierarchy}

- If you are dealing with an organization with a single hierarchy, or even a couple, then Organization Hierarchy is the simplest way to deal with things.
- However larger organizations grow beyond this.

Organization Structures



Organization Structures



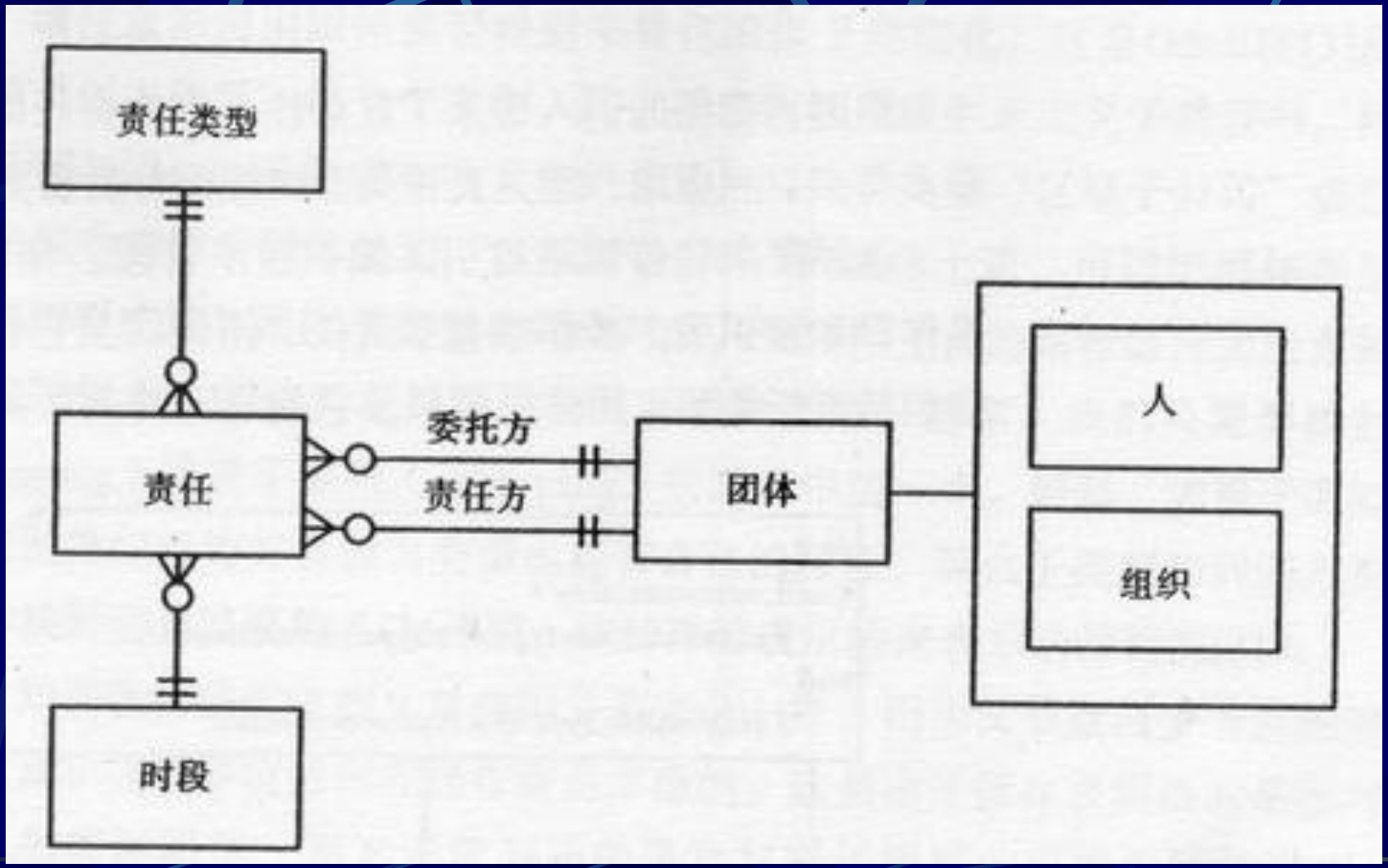
Modeling Principle *Design a model so that the most frequent modification of the model causes changes to the least number of types.*

例：为波士顿的2176大容量卡布奇诺咖啡机而设立的服务小组向波士顿的销售办事处负责。我们可以将其刻画成这样一个组织结构模型：父节点是波士顿的销售办事处，子节点是波士顿的2176服务小组，组织结构类型叫做产品线管理。

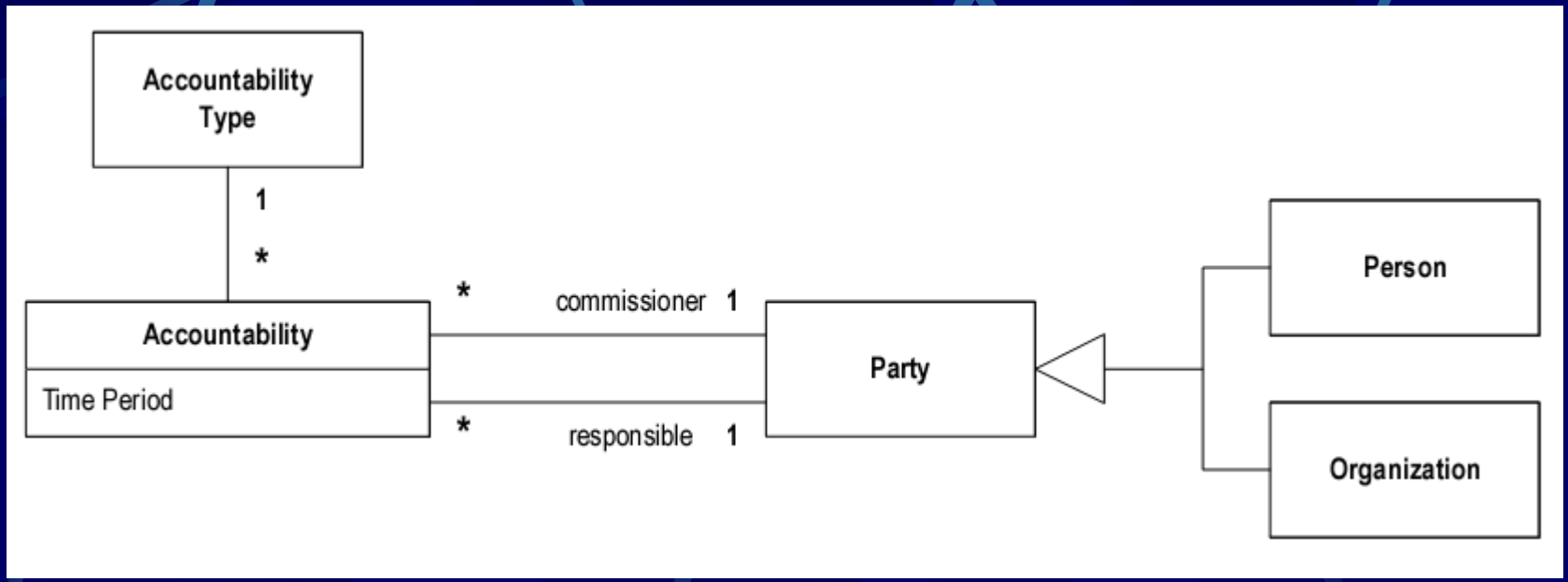
例：为波士顿的2176大容量卡布奇诺咖啡机而设立的服务小组向产品支持结构中的2170产品系列服务中心负责。我们可以把它看成是一个单独的组织结构，它的父节点是2170产品系列服务中心，而子节点是波士顿的2176服务小组，组织结构类型叫做产品支持。

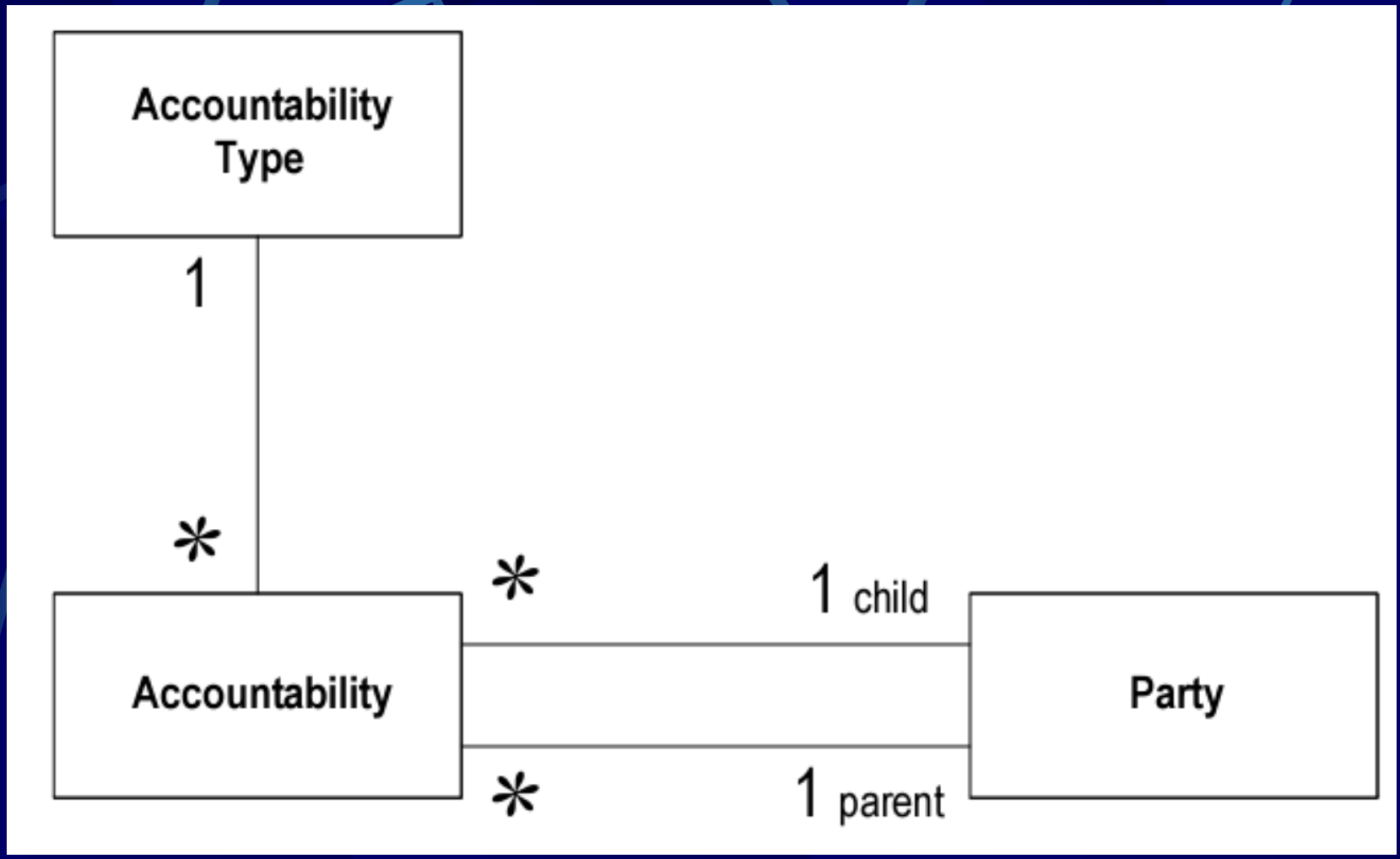
.3 Accountability

- Accountabilities represent the most powerful, and also the most complex way of dealing with organizational strucutres.

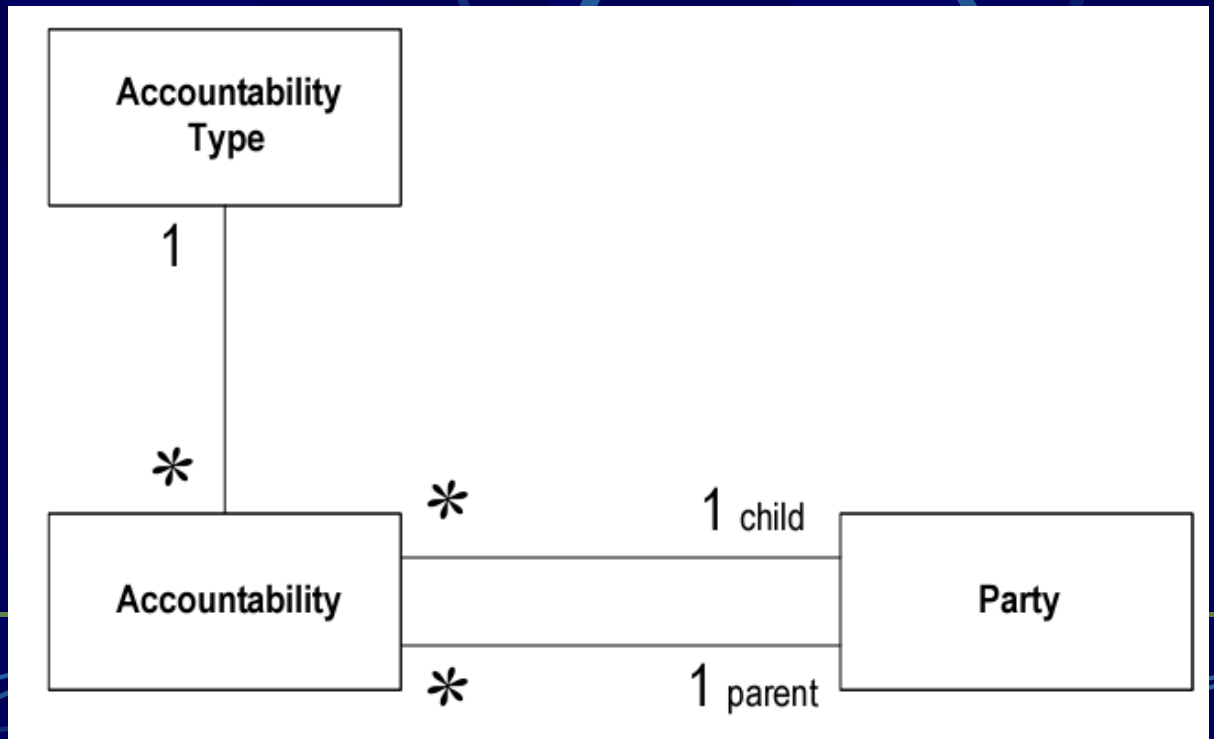

责任类型
责任
委托方
责任方
团体
人
组织
时段

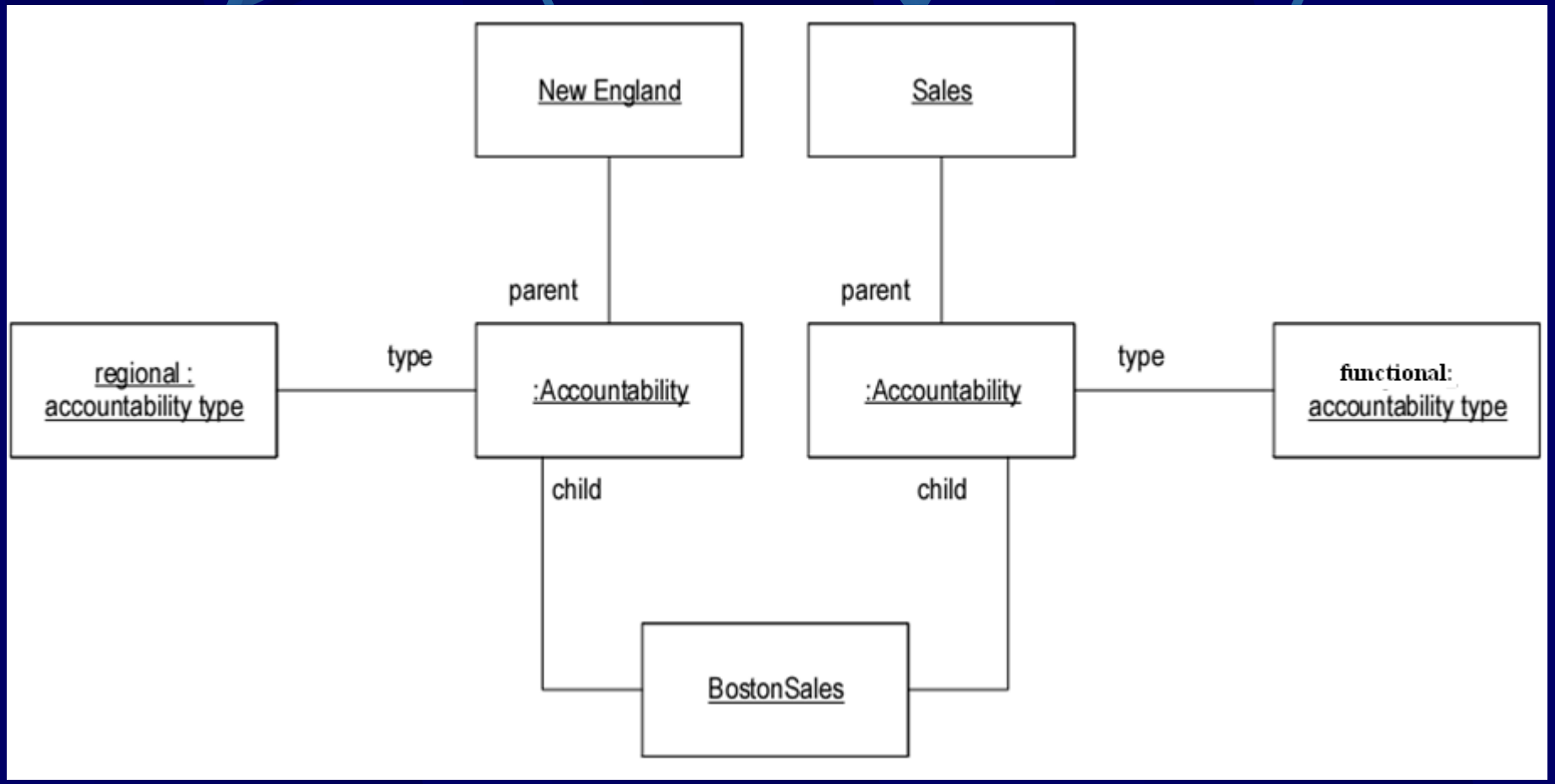
Accountability




Accountability Type
1
*
Accountability
*
*
1 child
1 parent
Party

- The previous problem
  - BostonSales as a child of NewEngland under the regional sturcture and a child of Sales under the functional structure




New England
Sales
parent
parent
regional :
accountability type
type
:Accountability
:Accountability
type
functional:
accountability type
child
child
BostonSales

例：John Smith为ACM工作，这可以被建模为一个责任模型，其中ACM是委托方，John Smith是责任方，责任类型是雇佣关系。

例：John Smith是波士顿2176服务小组的管理人员，这可以被建模为一个管理者类型的责任模型，其中John Smith对波士顿2176服务小组负责。

例：Mark Thursz是皇家医学院的成员，这可以被建模为一个职业注册类型的责任模型，其中Mark Thursz对皇家医学院负责。

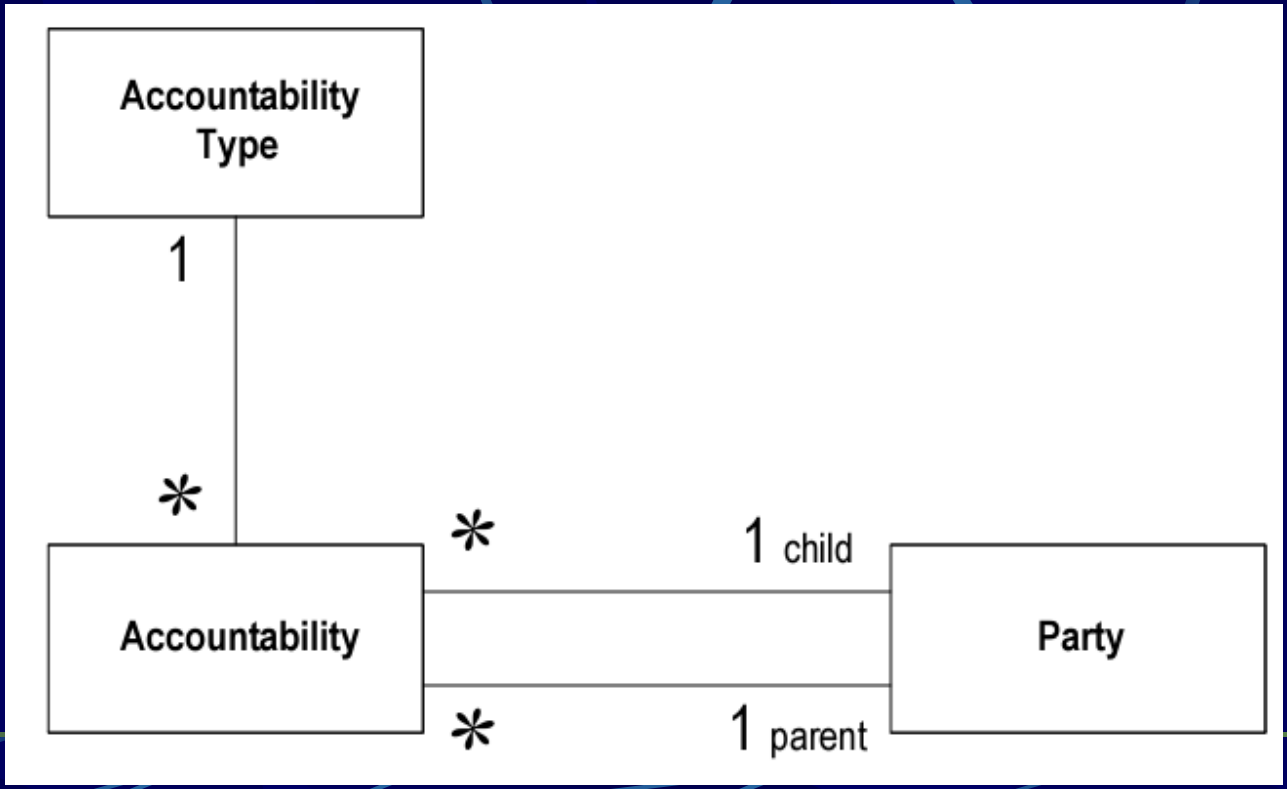
When to use it

- Use Accountability when you need to show organizational structures but Organization Hierarchy won't suffice.
- Accountability is a good bit more complicated to use than Organization Hierarchy, so don't use it until you need it.

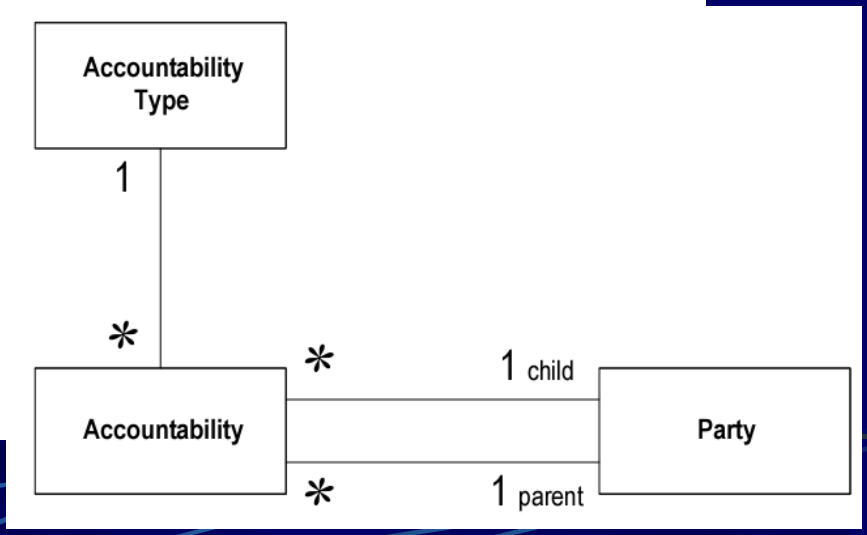
Sample Code

```
class AccountabilityType extends mf.NamedObject {
   public AccountabilityType(String name) {
        super(name);
   }
}
```

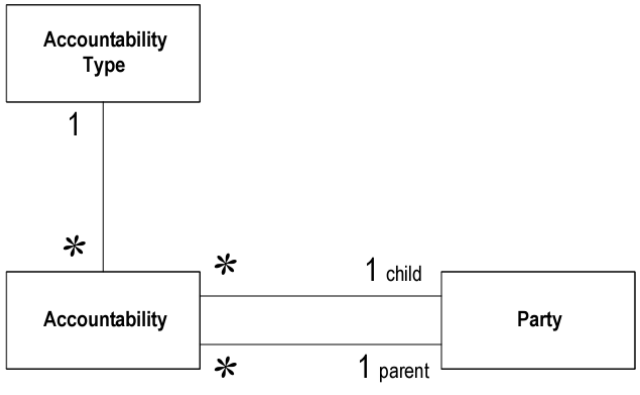
- Code: Accountability



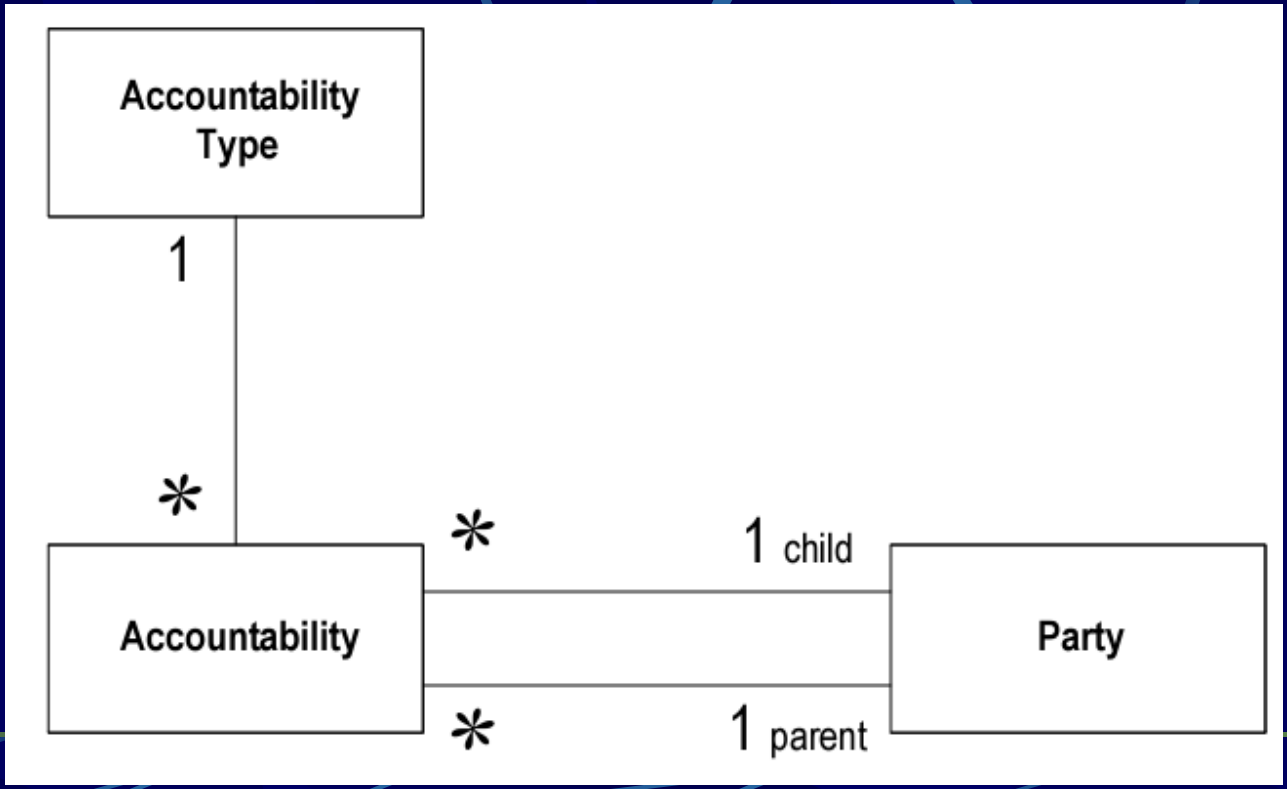
```
class Accountability {
  private Party parent;
  private Party child;
  private AccountabilityType type;

  Accountability (Party parent, Party child, AccountabilityType type) {
      this.parent = parent;
      parent.friendAddChildAccountability(this);
      this.child = child;
      child.friendAddParentAccountability(this);
      this.type = type;
  }
  Party child() {
      return child;
  }
  Party parent() {
      return parent;
  }
  AccountabilityType type() {
      return type;
  }
}
```



```
class Party extends mf.NamedObject {
  private Set parentAccountabilities = new HashSet();
  private Set childAccountabilities = new HashSet();
  public Party(String name) {
      super(name);
  }
  void friendAddChildAccountability(Accountability arg) {
      childAccountabilities.add(arg);
  }
  void friendAddParentAccountability(Accountability arg) {
      parentAccountabilities.add(arg);
  }
}
```



- Code: Party


Accountability Type
1
*
Accountability
*
1 child
Party
*
1 parent

```
class Party...
  Set parents() {
       Set result = new HashSet();
       Iterator it = parentAccountabilities.iterator();
       while (it.hasNext()) {
            Accountability each = (Accountability) it.next();
            result.add(each.parent());
       }
       return result;
  }
  Set children() {
       Set result = new HashSet();
       Iterator it = childAccountabilities.iterator();
       while (it.hasNext()) {
            Accountability each = (Accountability) it.next();
            result.add(each.child());
       }
       return result;
  }
```

```
class Tester...
      AccountabilityType supervision = new AccountabilityType("Supervises");
      Party mark = new Party("mark");
      Party tom = new Party("tom");
      Party stMarys = new Party ("St Mary's");
  public void setUp() {
      new Accountability (stMarys, mark, appointment);
      new Accountability (stMarys, tom, appointment);
  }
  public void testSimple() {
      assert(stMarys.children().contains(mark));
      assert(mark.parents().contains(stMarys));
  }
```

- you often need to carry out navigation along a single accountability type

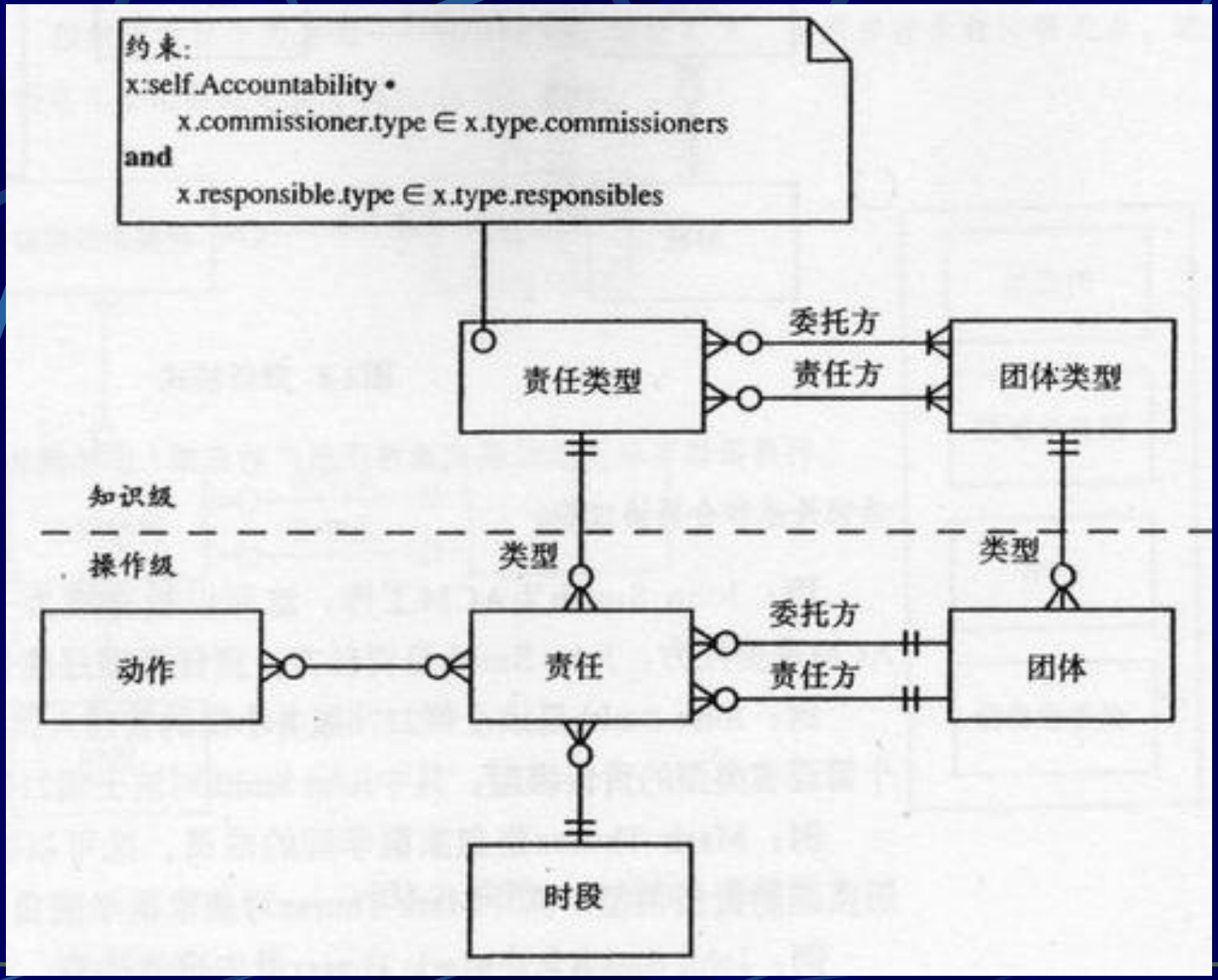
```
class Party...
  Set parents(AccountabilityType arg) {
      Set result = new HashSet();
      Iterator it = parentAccountabilities.iterator();
      while (it.hasNext()) {
          Accountability each = (Accountability) it.next();
          if (each.type().equals(arg)) result.add(each.parent());
      }
      return result;
  }
```

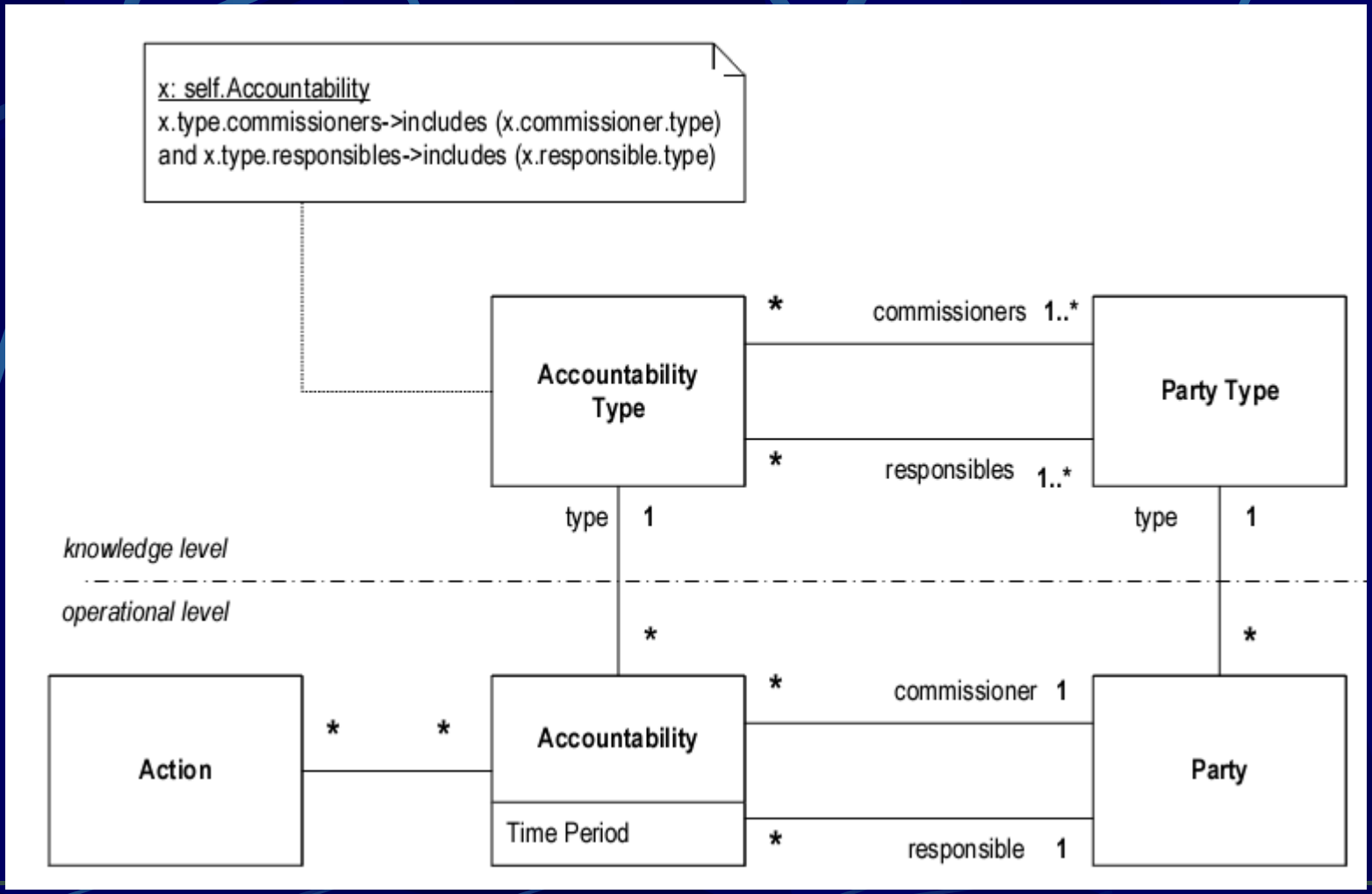
```
class Tester...
      AccountabilityType appointment = new AccountabilityType("Appointment");
  public void testParents() {
      Accountability.create(tom, mark, supervision);
      assert(mark.parents().contains(stMarys));
      assert(mark.parents(appointment).contains(stMarys));
      assertEquals(2, mark.parents().size());
      assertEquals(1, mark.parents(appointment).size());
      assertEquals(1, mark.parents(supervision).size());
      assert(mark.parents(supervision).contains(tom));
  }
```

6. Accountability Knowledge Level

- 责任类型比组织结构类型多得多，其规则更加复杂→
- 引入知识级来管理

Modeling Principle *Explicitly divide a model into operational and knowledge levels.*

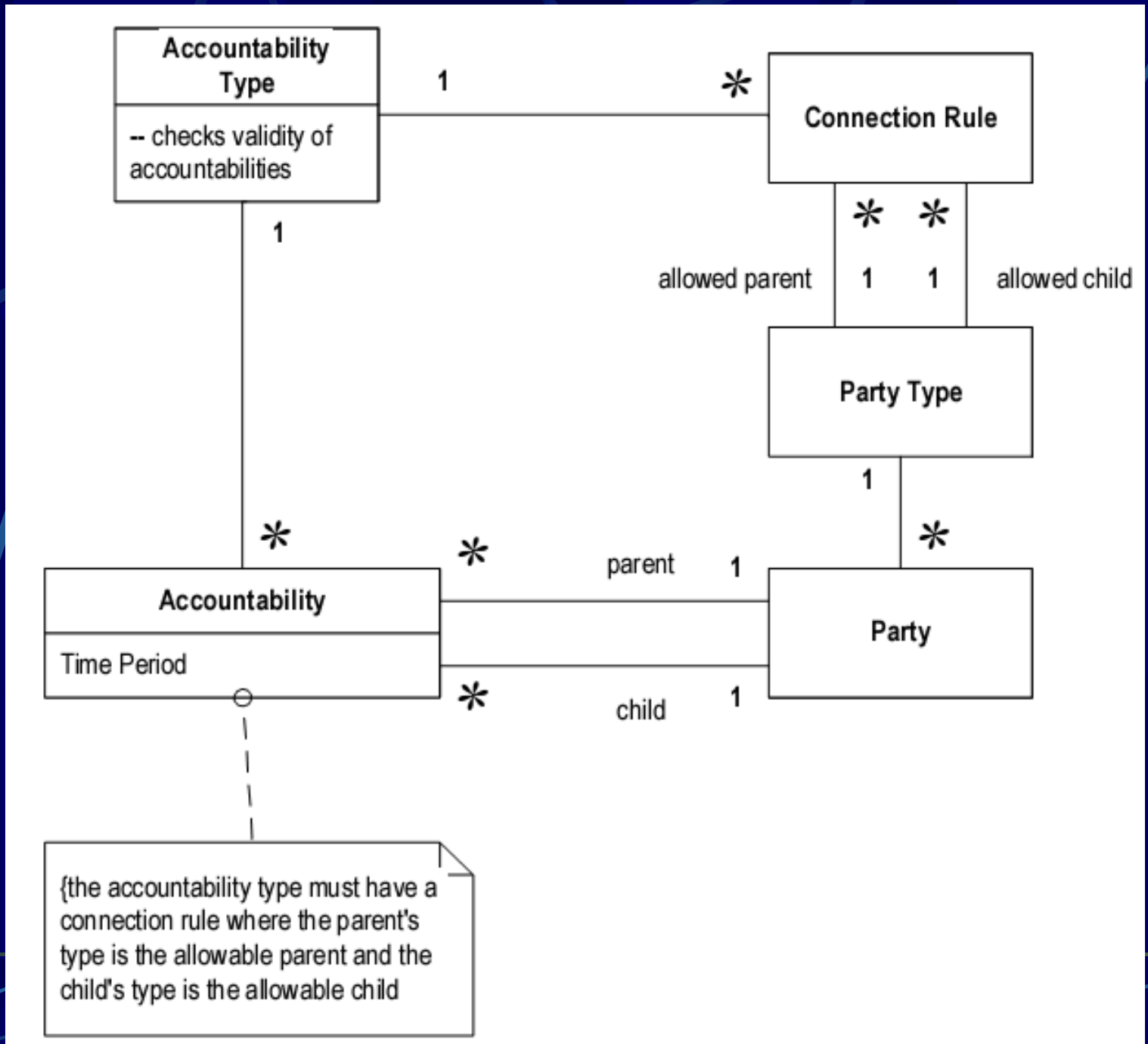

约束:
x:self.Accountability •
x.commissioner.type ∈ x.type.commissioners
and
x.responsible.type ∈ x.type.responsibles
责任类型
委托方
责任方
团体类型
知识级
操作级
类型
类型
动作
责任
委托方
责任方
团体
时段

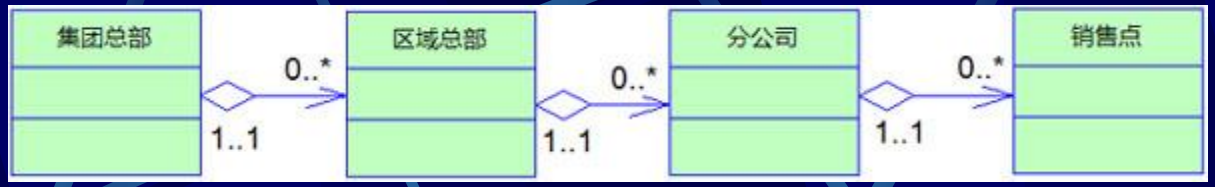

x: self.Accountability
x.type.commissioners->includes (x.commissioner.type)
and x.type.responsibles->includes (x.responsible.type)
Accountability Type
commissioners
1..*
responsibles
1..*
Party Type
type
1
type
1
knowledge level
operational level
Action
Accountability
Time Period
commissioner
1
responsible
1
Party

例：区域子公司又可细分成各个部门。这可以由一个区域子公司结构的责任类型来处理，其中委托方是区域子公司，责任方是部门。

例：患者许可可被定义成责任类型，其中委托方是患者，责任方是医生。

- Add Connection Rules
  - Connection rules provide a simple yet very flexible form of knowledge level.
  - Each accountability type contains a group of connection rules each of which defines one legal pairing of parent and child party types.


Accountability Type
-- checks validity of accountabilities
1
*
Connection Rule
1
*
*
allowed parent
1
1
allowed child
Party Type
1
*
*
*
parent
1
Accountability
Time Period
Party
*
child
1
{the accountability type must have a connection rule where the parent's type is the allowable parent and the child's type is the allowable child



Organization3Type

Org_Type_Id	Org_Type_Name
1	集团总部
2	区域总部
3	分公司
4	销售点

Organization3

Org_Id	Org_Type_Id	Org_Code	Org_Name
1	1 (集团总部)	MS	**股份有限公司
2	2 (区域总部)	RHD	华东区
3	2 (区域总部)	RHN	华南区
4	3 (分公司)	DSH	上海分公司
5	4 (销售点)	XuHui	徐家汇
6	4 (销售点)	PuDong	浦东

Organization3StructureType

Org_Struct_Type_Id	Org_Struct_Type_Name
1	生产组织结构
2	销售组织结构
3	产品线 - 打印机
4	产品线 - 扫描仪

Organization3StructureRule

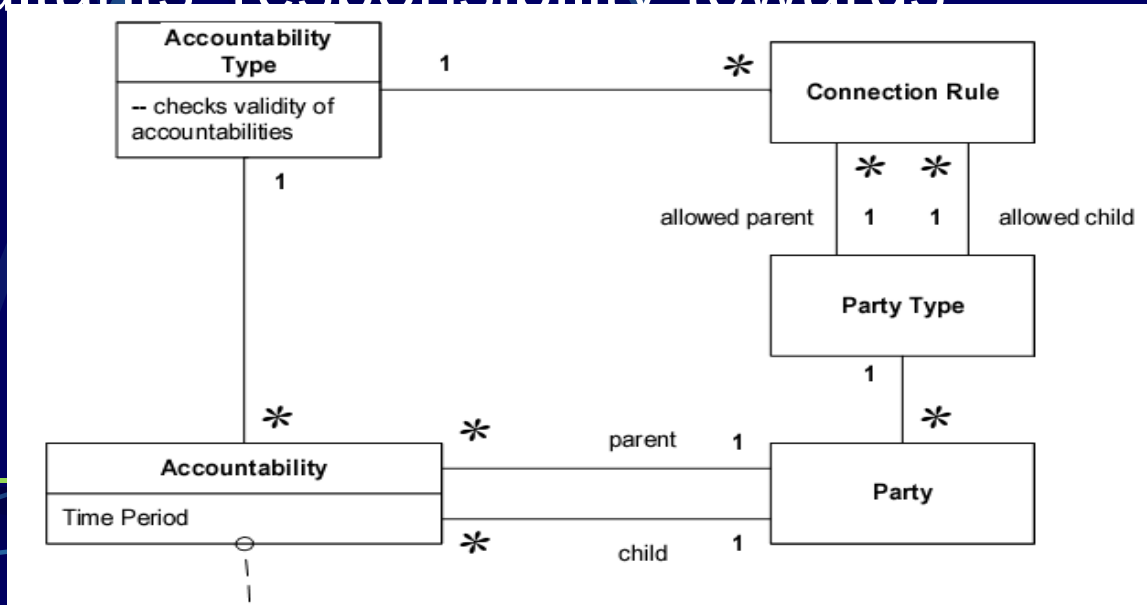
Org_Struct_Type_Id	Parent_Org_Type_Id	Child_Org_Type_Id	Child_Org_Number
2 (销售组织结构)	1 (集团总部)	2 (区域总部)	999999
2 (销售组织结构)	2 (区域总部)	3 (分公司)	999999
2 (销售组织结构)	3 (分公司)	4 (销售点)	10

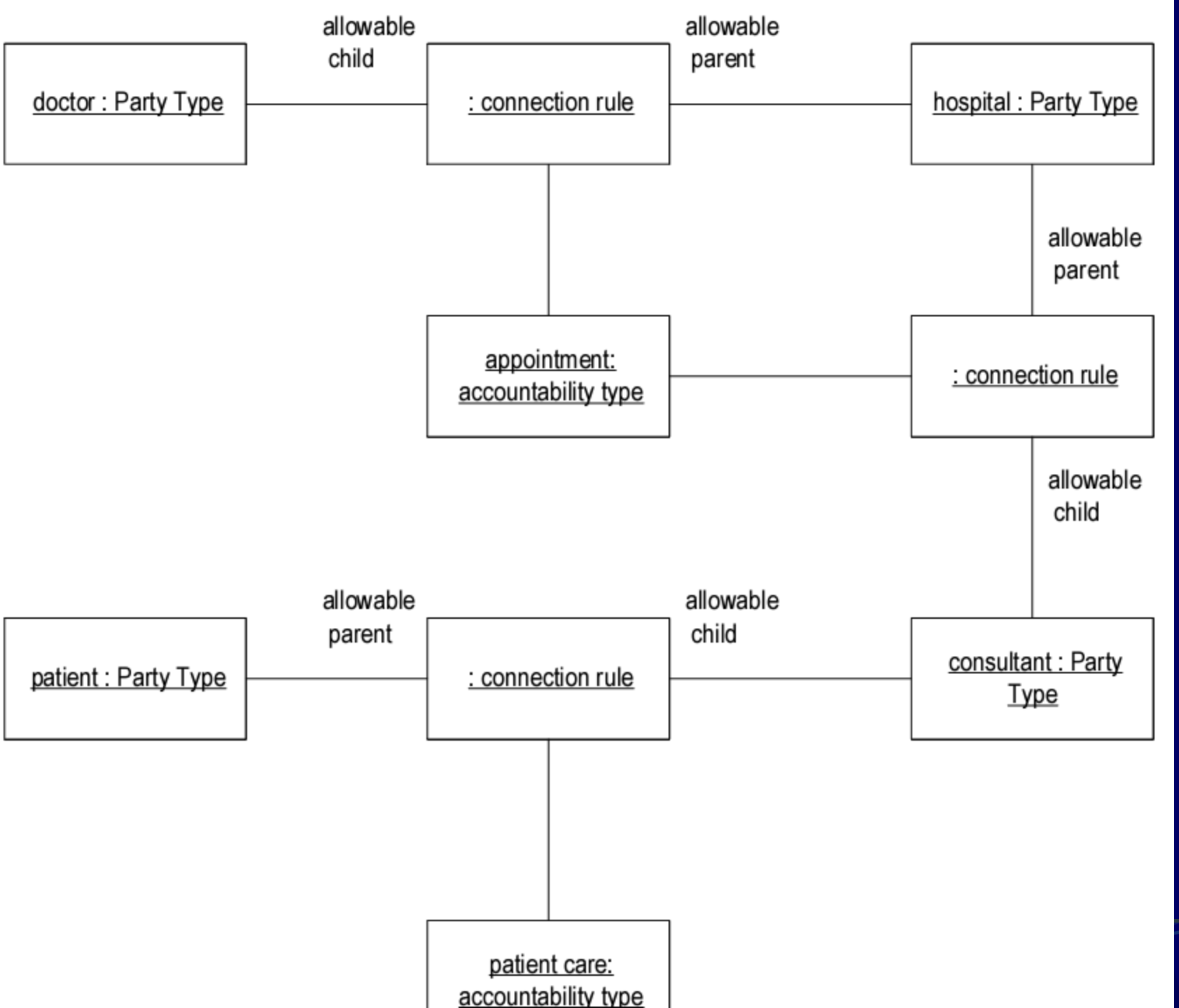
Organization3Structure

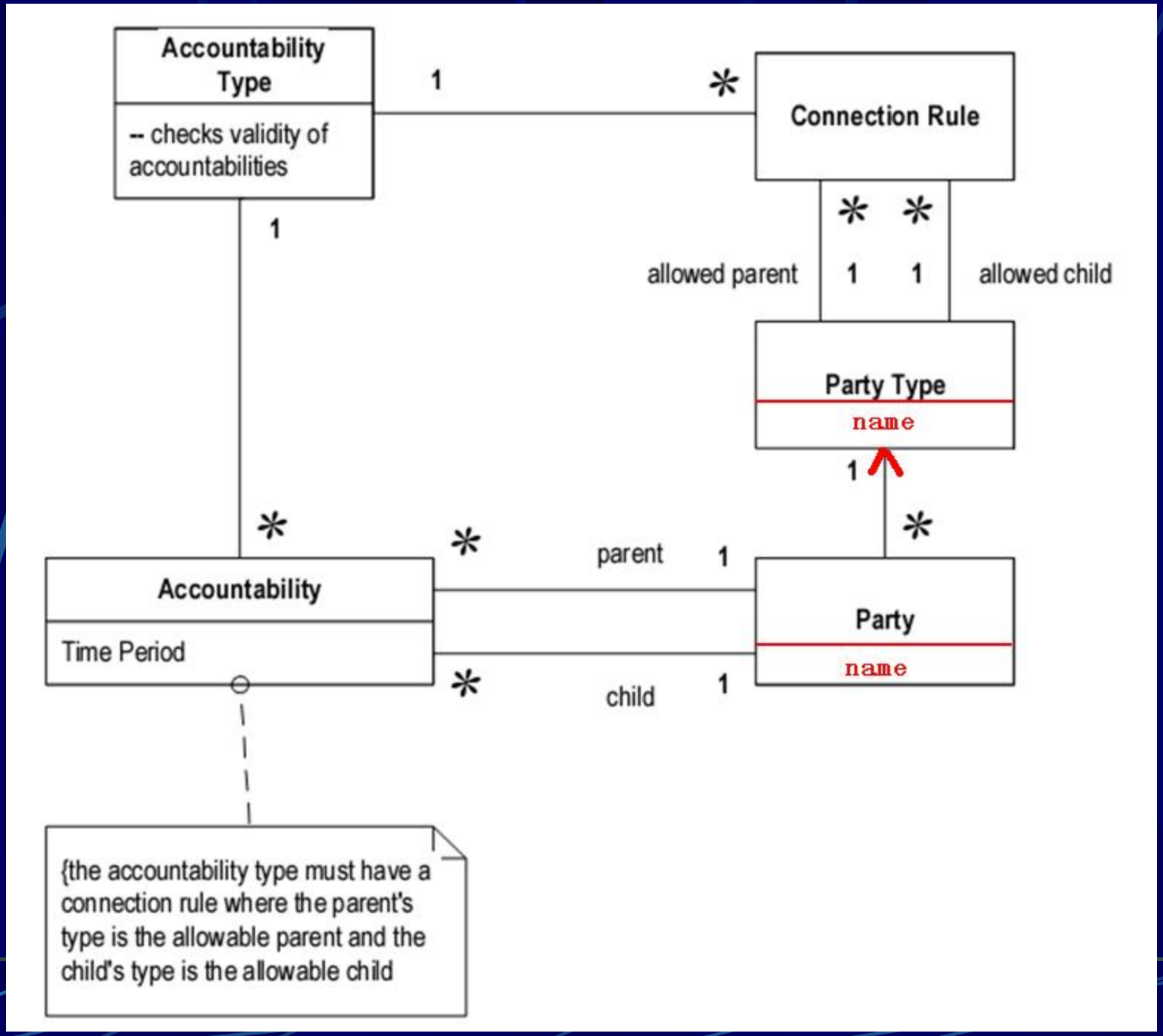
Org_Struct_Type_Id	Parent_Org_Id	Child_Org_Id
2 (销售组织结构)	1 (**股份有限公司)	2 (华东区)
2 (销售组织结构)	1 (**股份有限公司)	3 (华南区)
2 (销售组织结构)	2 (华东区)	4 (上海分公司)
2 (销售组织结构)	4 (上海分公司)	5 (徐家汇)
2 (销售组织结构)	4 (上海分公司)	6 (浦东)

Exercise

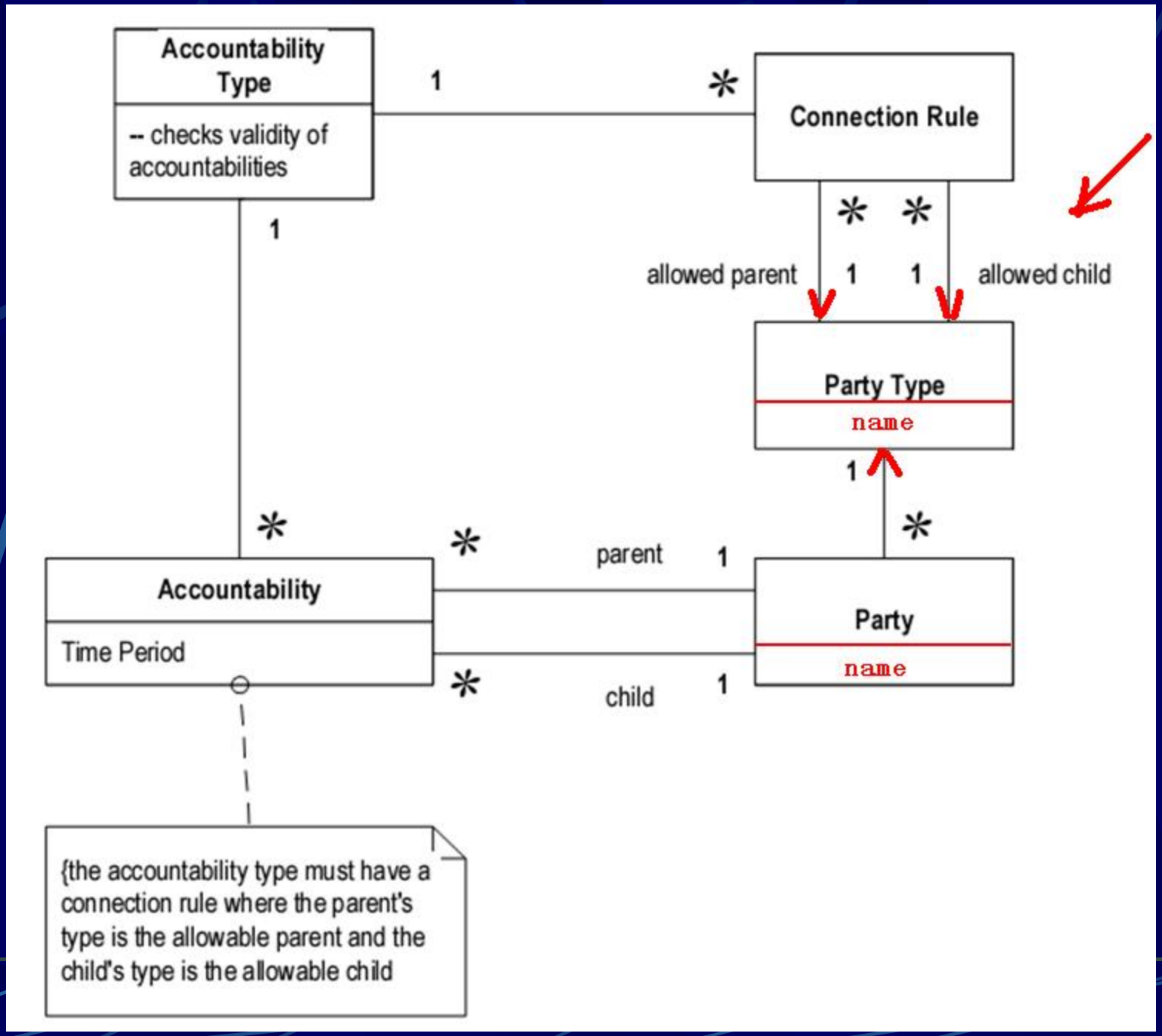
- two accountability types
  - The appointment accountability type allows consultants and doctors to be appointed to hospitals.
  - The patient care accountability type records consultants' responsibility towards patient




doctor : Party Type
allowable child
: connection rule
allowable parent
hospital : Party Type
allowable parent
appointment: accountability type
: connection rule
allowable child
patient : Party Type
allowable parent
: connection rule
allowable child
consultant : Party Type
patient care: accountability type

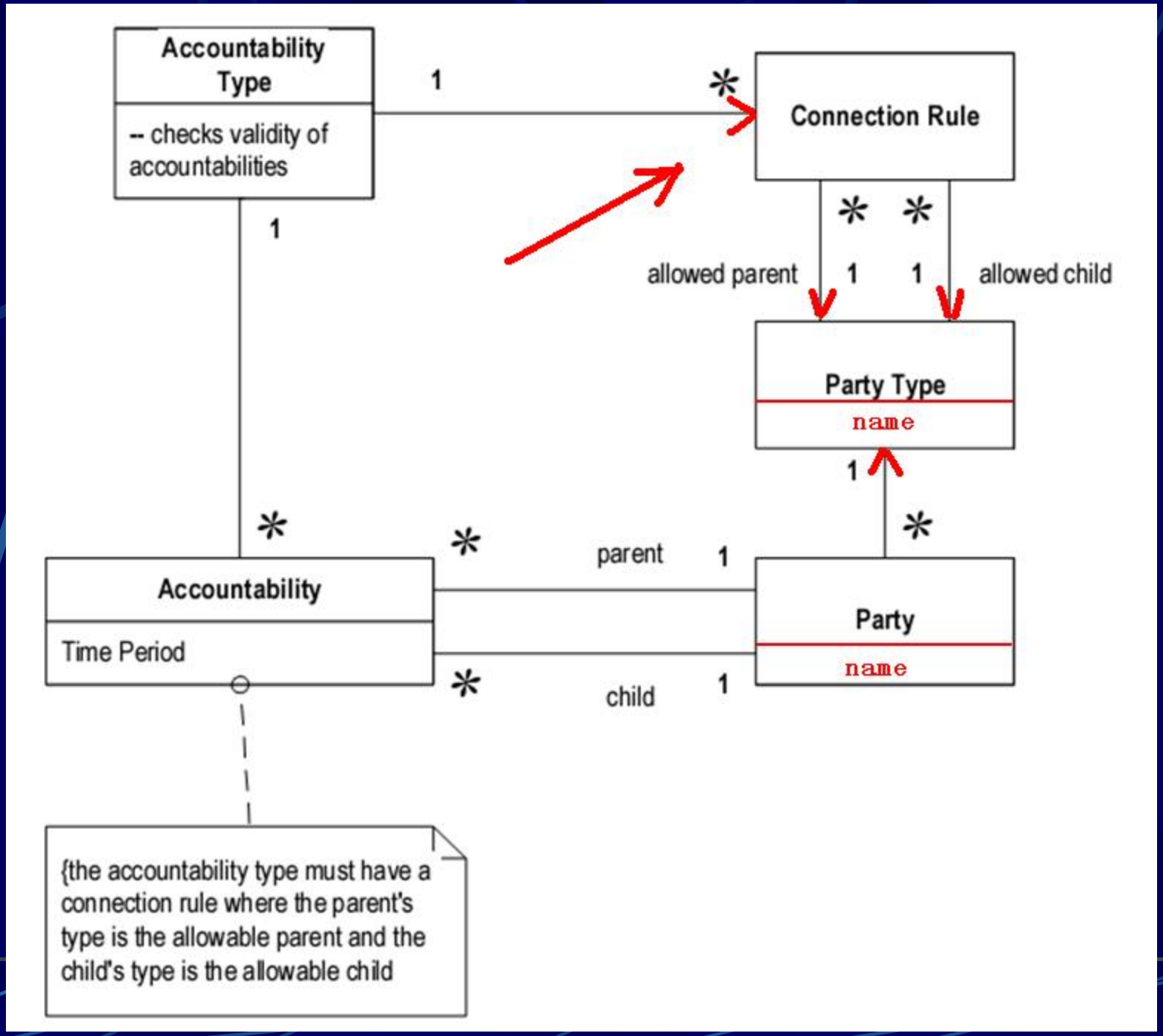

Accountability Type
-- checks validity of accountabilities
1
*
Connection Rule
1
*
*
allowed parent
1
1
allowed child
Party Type
name
1
*
*
*
parent
1
Accountability
Time Period
Party
name
*
child
1
{the accountability type must have a connection rule where the parent's type is the allowable parent and the child's type is the allowable child

```
class PartyType extends mf.NamedObject {
  public PartyType(String name) {
      super(name);
  }
}
class Party ...
  private PartyType type;
  public Party(String name, PartyType type) {
      super(name);
      this.type = type;
  }
  PartyType type() {
      return type;
  }
}
```


Accountability Type
-- checks validity of accountabilities
1
*
Connection Rule
1
*
*
allowed parent
1
1
allowed child
Party Type
name
1
*
*
*
parent
1
Accountability
Time Period
Party
name
*
child
1
{the accountability type must have a connection rule where the parent's type is the allowable parent and the child's type is the allowable child

- Rules

```
class ConnectionRule {
  PartyType allowedParent;
  PartyType allowedChild;
  public ConnectionRule(PartyType parent, PartyType child) {
      this.allowedChild = child;
      this.allowedParent = parent;
  }
}
```


Accountability Type
-- checks validity of accountabilities
1
*
Connection Rule
1
*
*
allowed parent
1
1
allowed child
Party Type
name
1
*
*
*
parent
1
Accountability
Time Period
Party
name
*
child
1
{the accountability type must have a connection rule where the parent's type is the allowable parent and the child's type is the allowable child

- Add rules to the accountability type

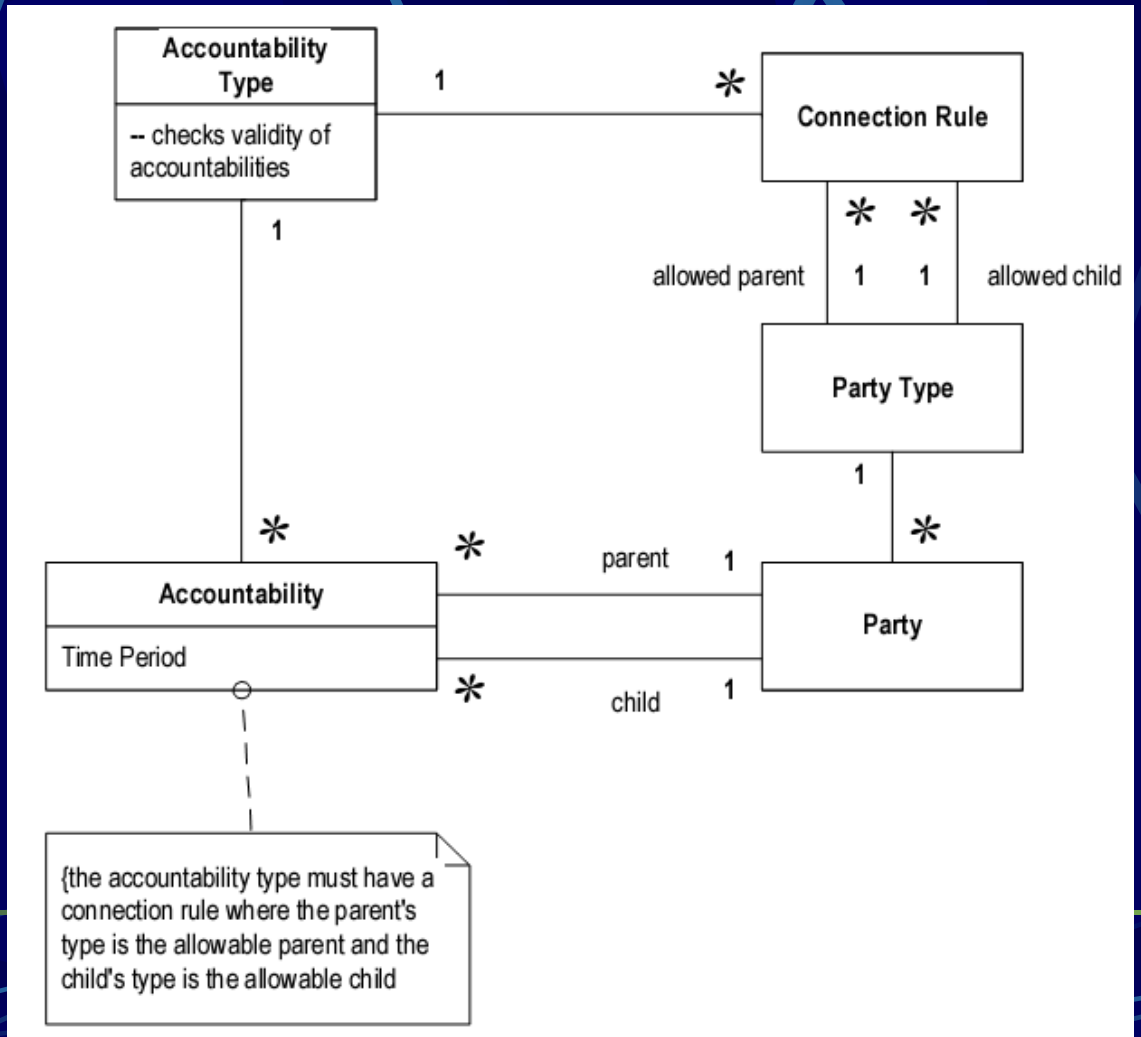
```
class ConnectionAccountabilityType extends AccountabilityType ...
  Set connectionRules = new HashSet();
  public ConnectionAccountabilityType(String name) {
      super(name);
  }
  void addConnectionRule (PartyType parent, PartyType child) {
      connectionRules.add(new ConnectionRule(parent, child));
  }
```

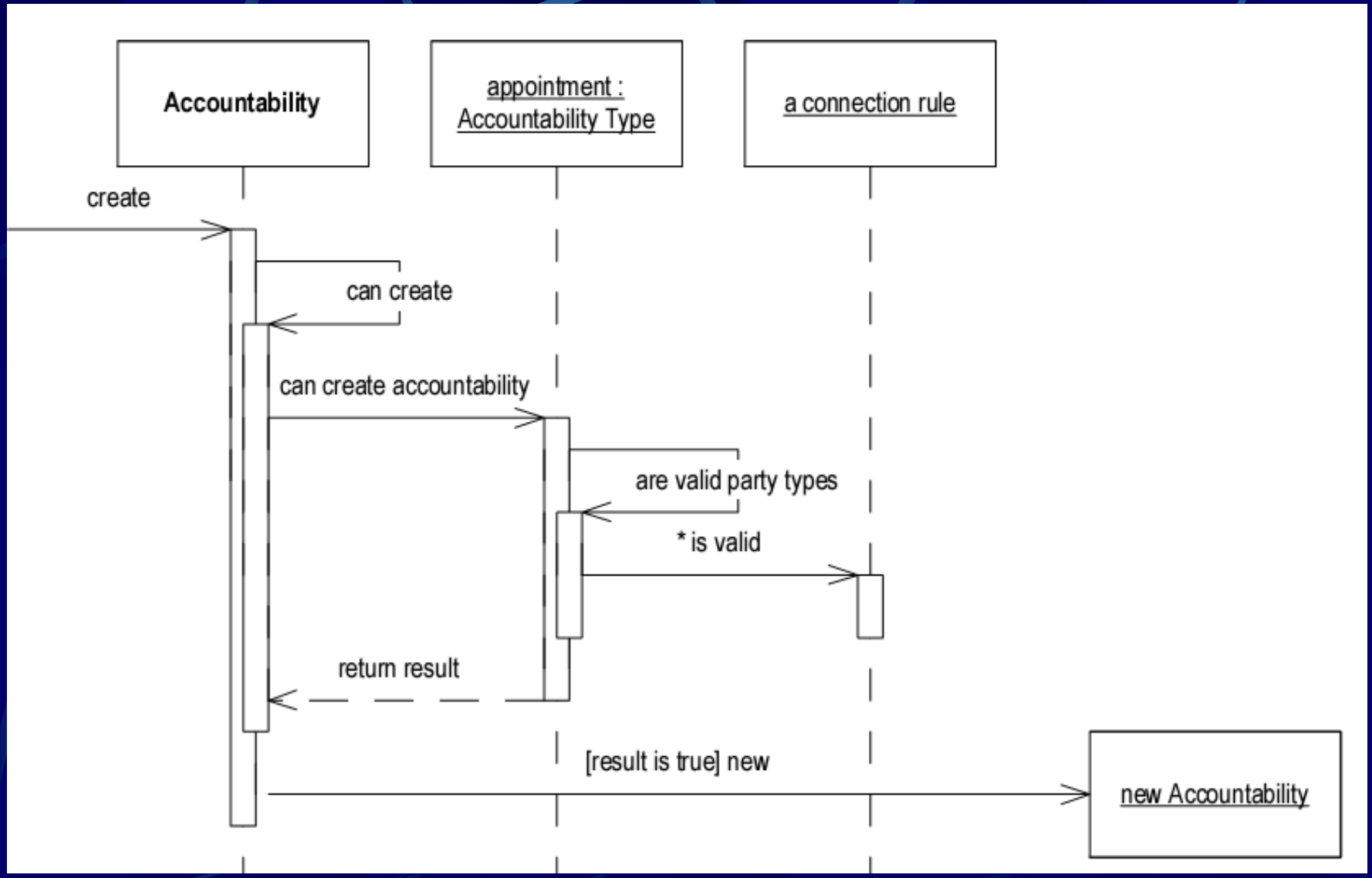
```
class Tester...
  private PartyType hospital = new PartyType("Hospital");
  private PartyType doctor = new PartyType("Doctor");
  private PartyType patient = new PartyType("Patient");
  private PartyType consultant = new PartyType("Consultant");
  private ConnectionAccountabilityType appointment
      = new ConnectionAccountabilityType("Appointment");
  private ConnectionAccountabilityType supervision
      = new ConnectionAccountabilityType("Supervises");

  public void setUp()...
      appointment.addConnectionRule(hospital, doctor);
      appointment.addConnectionRule(hospital, consultant);
      supervision.addConnectionRule(doctor, doctor);
      supervision.addConnectionRule(consultant, doctor);
      supervision.addConnectionRule(consultant, consultant);

      mark = new Party("mark", consultant);
      tom = new Party("tom", consultant);
      stMarys = new Party ("St Mary's", hospital);
```

Object Diagram？

Interaction-- validation

- static




Accountability
appointment :
Accountability Type
a connection rule
create
can create
can create accountability
are valid party types
* is valid
return result
[result is true] new
new Accountability

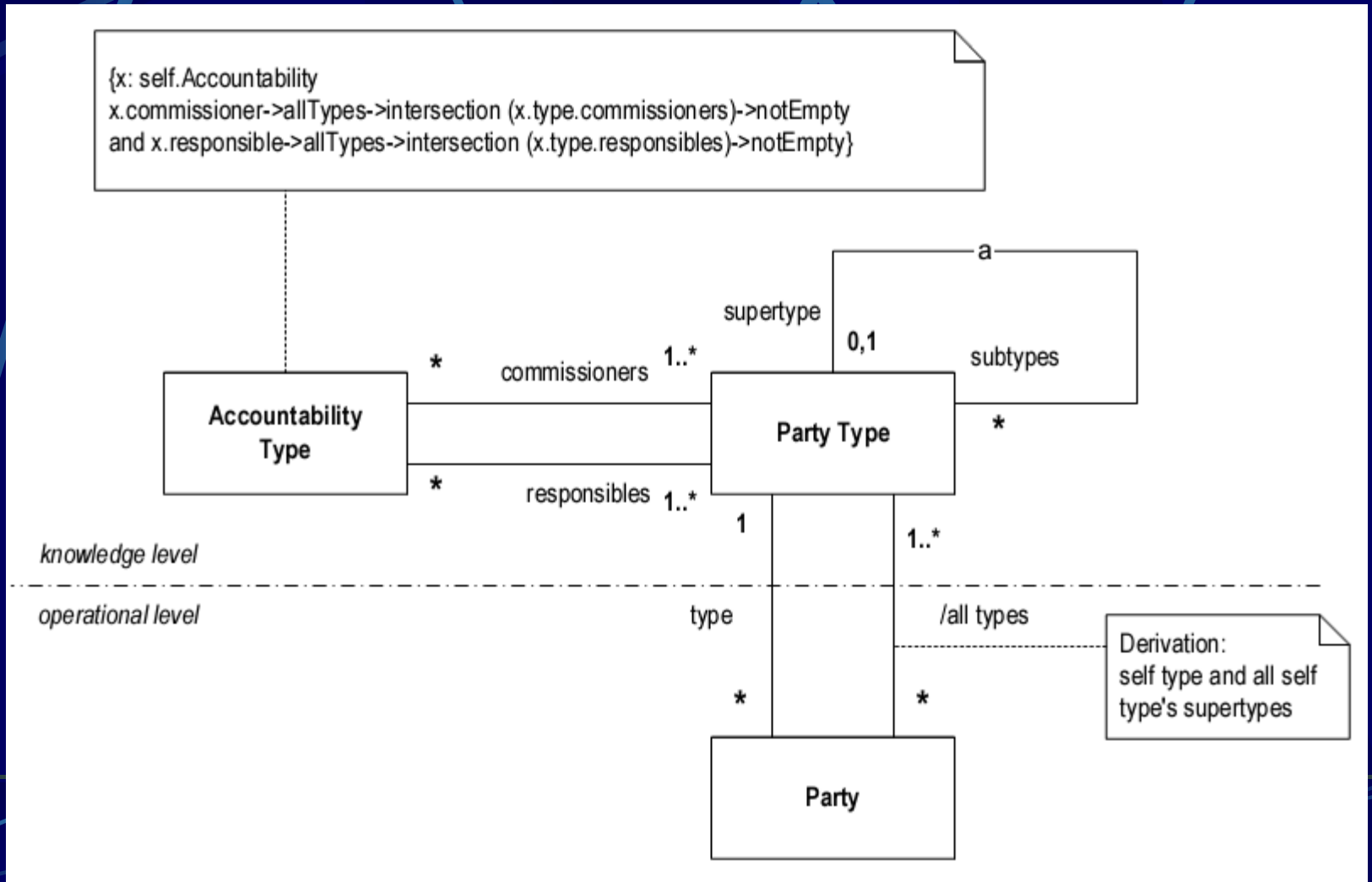
```
class Accountability...
  static boolean canCreate (Party parent, Party child, AccountabilityType type) {
      if (parent.equals(child)) return false;
      if (parent.ancestorsInclude(child, type)) return false;
      return type.canCreateAccountability(parent, child);
  }
```

- The accountability type can now check the parent and the child through the connection rules.

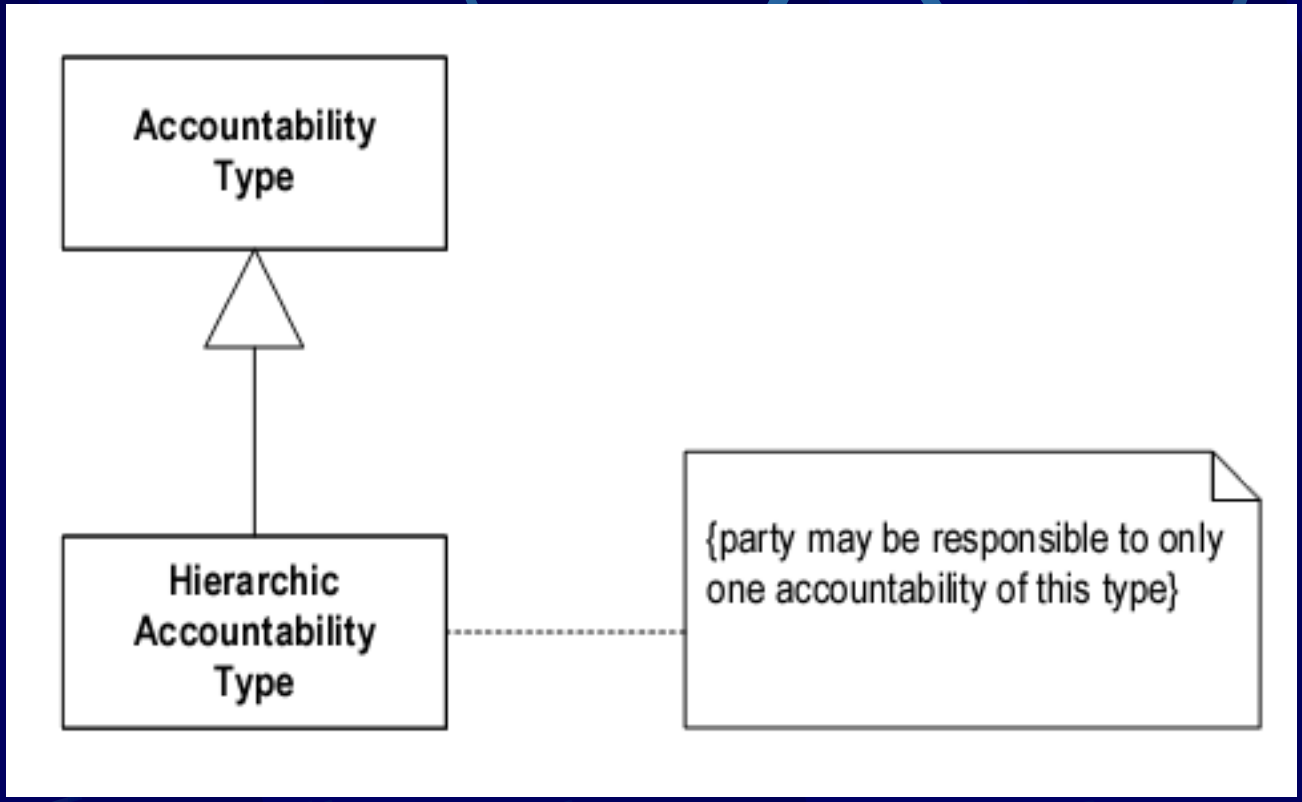
```
class AccountabilityType...
  boolean canCreateAccountability(Party parent, Party child) {
      return areValidPartyTypes(parent, child);
  }

class ConnectionRuleAccountabilityType...
  protected boolean areValidPartyTypes(Party parent, Party child) {
      Iterator it = connectionRules.iterator();
      while (it.hasNext()) {
          ConnectionRule rule = (ConnectionRule) it.next();
          if (rule.isValid(parent, child)) return true;
      }
      return false;
  }

class ConnectionRule...
  boolean isValid (Party parent, Party child) {
      return (parent.type().equals(allowedParent) &&
          child.type().equals(allowedChild));
  }
```

```
class Tester...
  public void testNoConnectionRule() {
      try {
          Accountability.create(mark, stMarys, appointment);
          fail("created accountability without connection rule");
      } catch (Exception ignore) {}
      assert(!stMarys.parents().contains(mark)); // am I paranoid?
  }
```

Party Type Generalizations



Hierarchic Accountability Type



```
class AccountabilityType...
  boolean canCreateAccountability(Party parent, Party child) {
      return areValidPartyTypes(parent, child);
  }

class ConnectionRuleAccountabilityType...
  protected boolean areValidPartyTypes(Party parent, Party child) {
      Iterator it = connectionRules.iterator();
      while (it.hasNext()) {
          ConnectionRule rule = (ConnectionRule) it.next();
          if (rule.isValid(parent, child)) return true;
      }
      return false;
  }
```

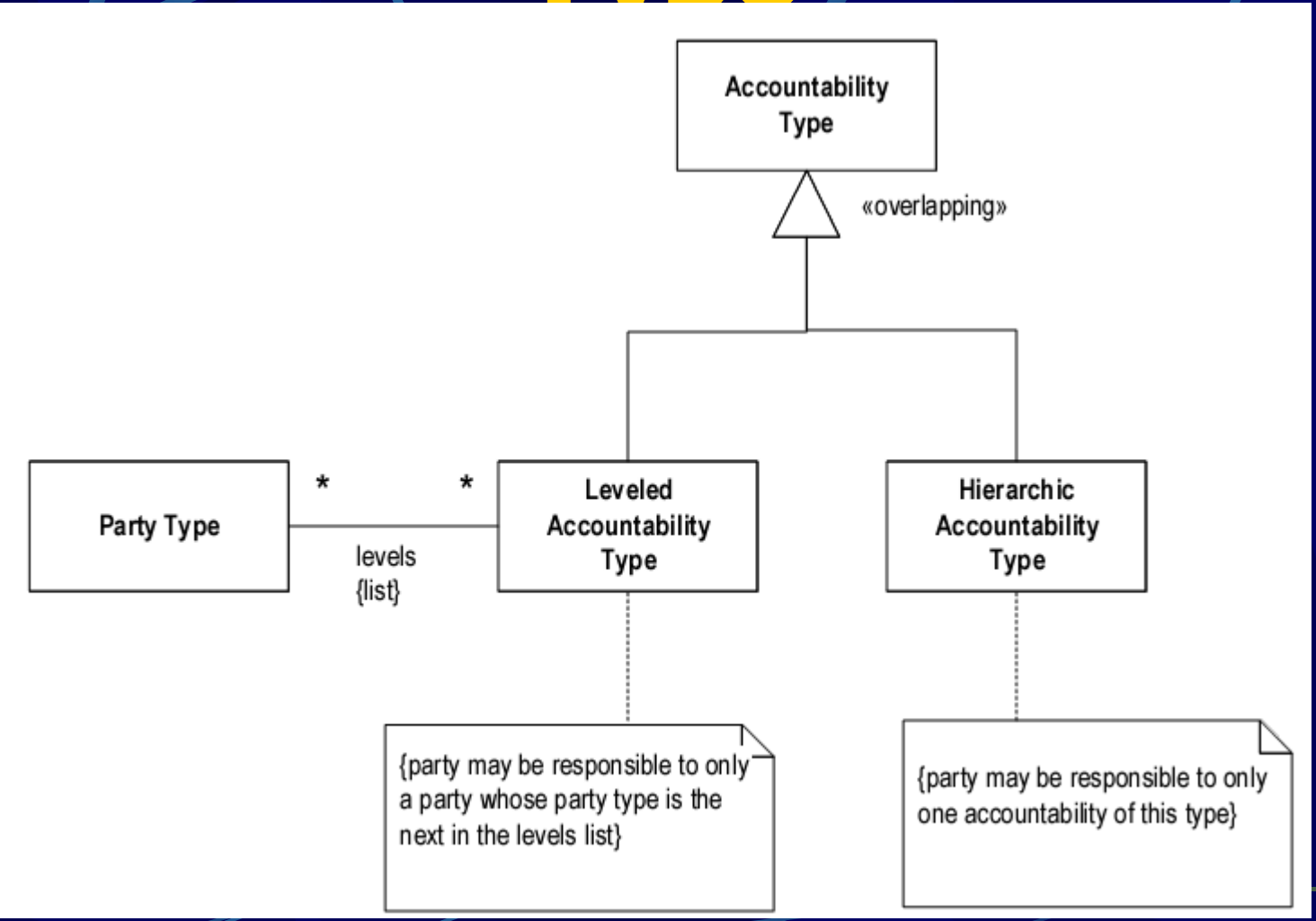
```
class AccountabilityType ...
  private boolean isHierarchic = false;
  void beHierarchic() {
      isHierarchic = true;
  }
  boolean canCreateAccountability(Party parent, Party child) {
      if (isHierarchic && child.parents(this).size() != 0) return false;
      return areValidPartyTypes(parent, child);
  }
}
```

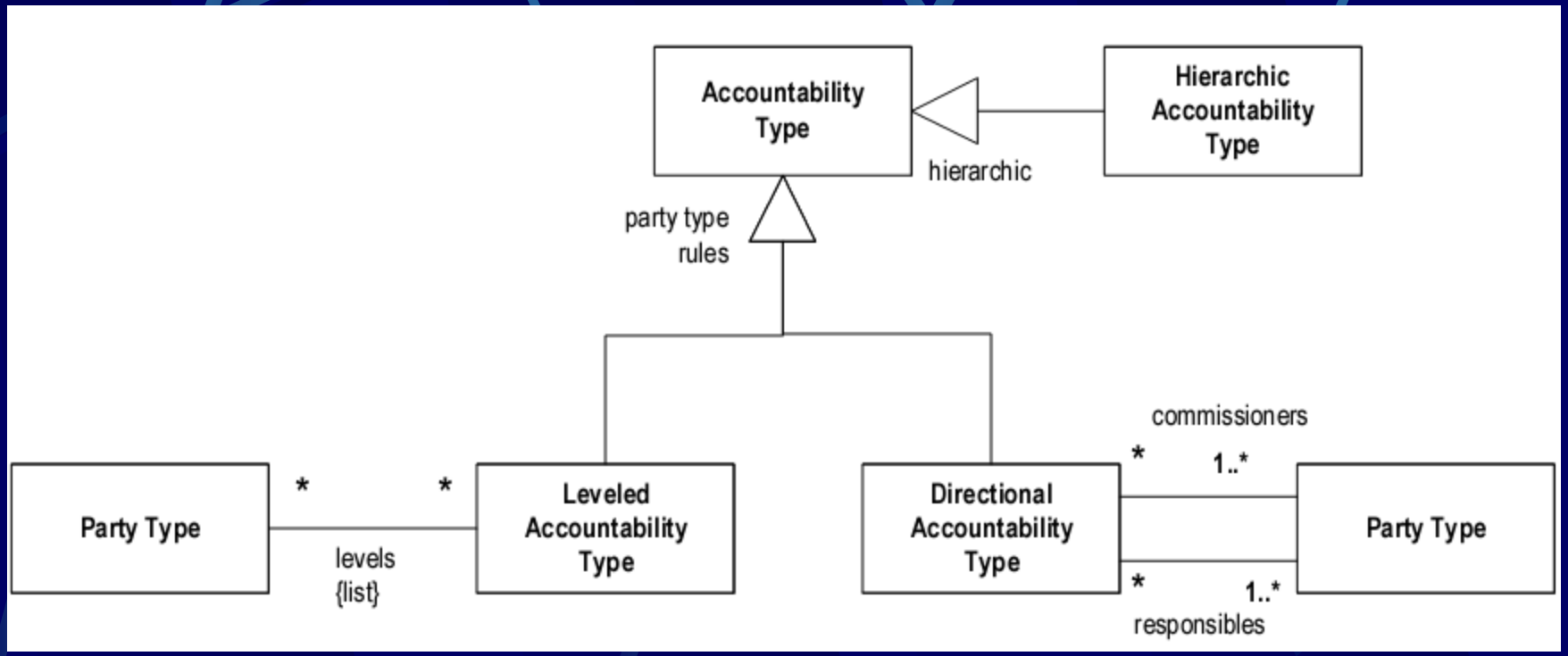
Levelled Accountability Type

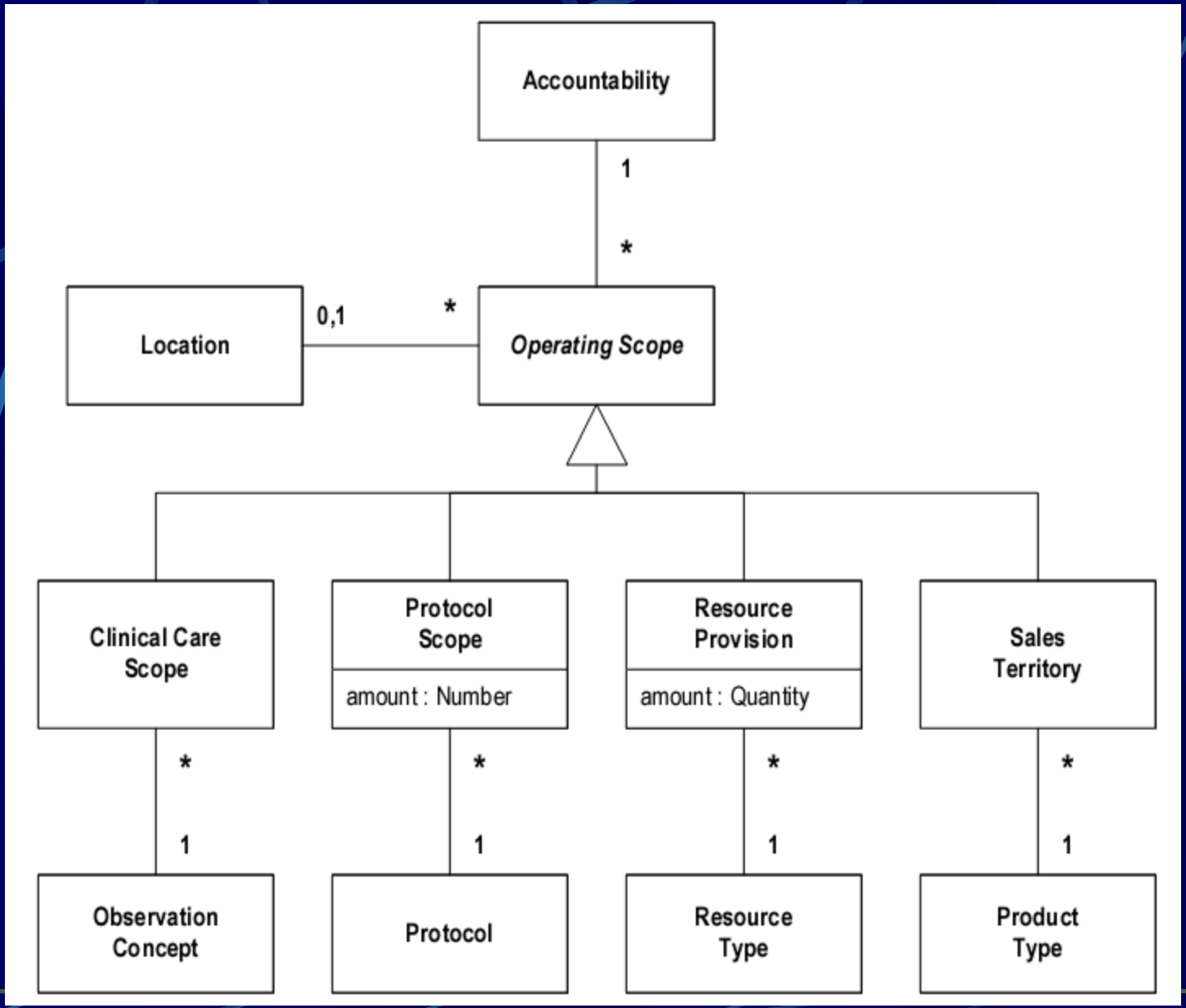
- An example of this is a regional breakdown where national parties have children that are states, which have children that are counties, that have children that have cities.

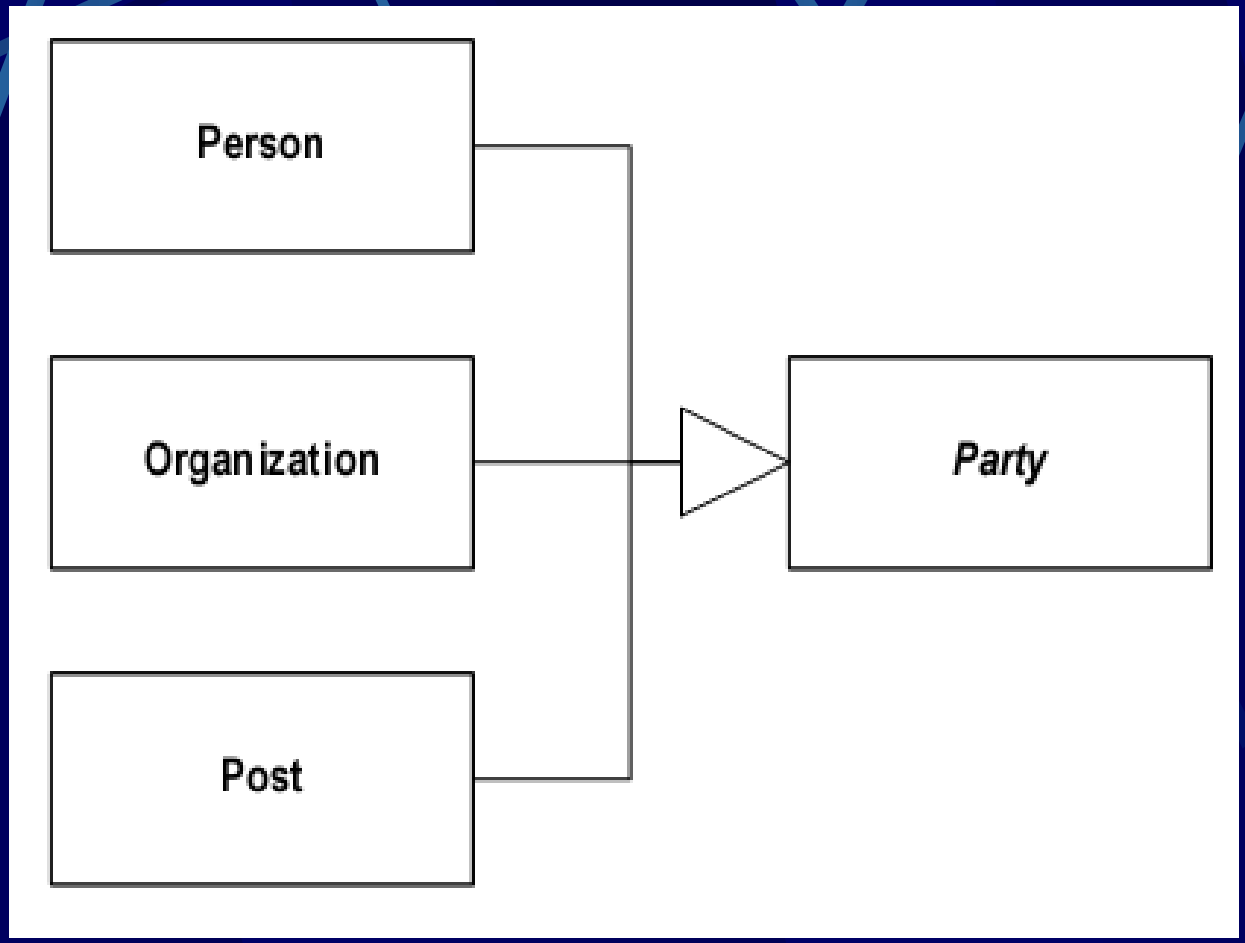
```
private PartyType nation = new PartyType("nation");
private PartyType state = new PartyType("state");
private PartyType county = new PartyType("county");
private PartyType city = new PartyType("city");
```

Levelled Accountability Type




Accountability Type
Hierarchic Accountability Type
hierarchic
party type rules
commissioners
Party Type
*
*
levels {list}
Leveled Accountability Type
Directional Accountability Type
*
1..*
*
1..*
responsibles
Party Type


Accountability
1
*
Location
0,1
*
Operating Scope
Clinical Care Scope
Protocol Scope
amount : Number
Resource Provision
amount : Quantity
Sales Territory
*
1
*
1
*
1
*
1
Observation Concept
Protocol
Resource Type
Product Type


Person
Organization
Post
Party