Computer Science Seminar Series

April 18, 2003

Integrating Object-Oriented Design and High-Level Petri Nets in Development of Concurrent Software Systems

Boleslaw Mikolajczak

with contributions by D. Mukhin, Ch. Sefranek, H. Hsueh, A. Cabeza, Z. Wang, B. Bauskar

Computer and Information Science Department

College of Engineering

University of Massachusetts Dartmouth

Contents

- **Motivation and Introduction**
- **Integration of Object-Orientedness and Petri Nets**
- **Applying Colored Petri Nets to Object-Oriented Design:**
  - **Abstract Node** - High Level Abstraction of an Object
  - **Interface** - First Level Refinement
  - **Method Implementation** - Second Level Refinement
  - **Inheritance versus Delegation -** for Classes and Objects
  - **Inheritance Anomaly** and Methods of Its Resolution
  - **Polymorphism and Dynamic Behavior** of Objects
- **Conclusions**
- **References**
- **Future Work**

Motivation

- **Imperative from Mellor and Shlaer (1994):**

  "The ability to execute the application analysis model is a sine qua non for any industrial-strength method because analysts need ***to verify the behavior of the model*** *with both clients and domain experts"*

- **Software development:**

  \- **orthogonality between pragmatics and theory** - pragmatics attributes and theory attributes are/should be mutually independent; i.e. it is possible to develop software systems with/without theory

  \- **pragmatics attributes:** modularity, readability, reusability

  \- **theory attributes:** formal analysis of software properties, model's executability

Introduction

- **Object-Oriented Design:**
  - well established design techniques with classes, responsibilities, and collaboration graphs being result of some OOD method
  - lacks analysis, verification and validation (V/V) methods of the designed system
  - lacks formal specification of concurrency and partial ordering of events
  - support for inheritance, polymorphism, and dynamic binding
  - UML modeling provides Message Sequence Chart (MSC) for interactions and ordering of objects; some actions may constitute co-regions, i.e. they remain unordered

Introduction, ctnd.

- **Petri Nets:**
  - well-defined formalism of parallel/distributed system modeling with graphical and algebraic representation
  - **conflict** representation and resolution (as a mechanism of choice)
  - **confusion** representation and its algorithmic detection (as co-existence of concurrency and conflict)
  - **time** representation (delay and duration) and time annotations as part of arc/transition inscriptions
  - **resource allocation** explicit representation
  - support for **abstraction and refinement** using **vicinity preserving and general Petri net morphisms** - elements of hierarchical structuring
  - weak support for composition of Petri net-based models
  - lacks clear and effective specification of system design techniques
  - strong analysis, verification and validation techniques and broadly available CASE tools

Integration of Object-Orientedness and Petri Nets

- **Three approaches of integration of Petri nets with object-oriented concepts:**

  - giving a formal basis to an object-oriented language or methodology

  - extending Petri nets by the use of complex data types for tokens

  - using object-oriented concepts directly in the Petri net formalism

- **Integration of objects with Petri nets is difficult because modeling and structuring power of objects is often in conflict with the proving facilities of Petri nets (**examples: using of complex data types for tokens, support for inheritance and polymorphism)

- **rapid prototyping of PDSS -** modeling, analysis, V/V, and performance evaluation of the designed system

Integration of Object-Orientedness and Petri Nets

- **OO and Petri nets are complementary methodologies**
- **Goal:**
  - use Object-Oriented methodology/technology on the design stage
  - use Petri Nets on the analysis and Verification/Validation stage
- **Approach #1: "Objects inside Petri Nets"**
  - increases token's intelligence
  - represents a designed system as a single, large Petri Net
  - does not contribute to abstraction of Petri Nets
- **Approach #2: "Petri Nets inside Objects"**
  - to model the inner behavior of objects (as sequential or concurrent)
  - very valuable starting point for the abstraction of Petri Nets
  - usually Object-Based rather than Object-Oriented

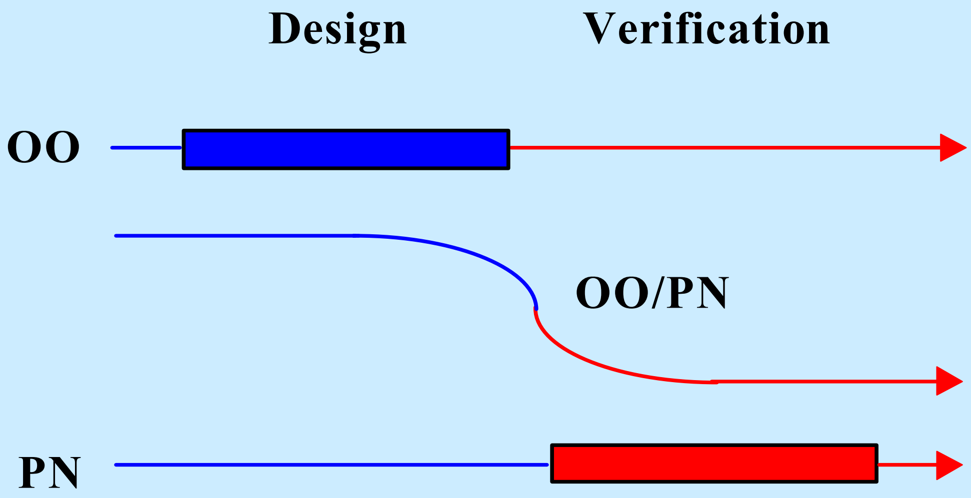
Integration of Object-Orientedness and Petri Nets

- **Petri Nets First - Top-down Approach:**
  - start with a Petri Net and represent a system by means of PN
  - objects are tokens or sub-nets
  - beneficial for verification (PNs support V/V)
  - abstraction is used to relate objects to tokens and sub-nets
- **Object Orientedness First - Bottom-up Approach:**
  - start with a result of some kind of OOD
  - Petri Nets represent classes and object interactions
  - beneficial for design (full power of design method can be utilized)
  - single, large Petri Net; no abstraction
  - **object** = (data structures, operations, object's behavior)

Integration of Object-Orientedness and Petri Nets, ctnd.

- **Sibertin-Blanc & Bastide -** Petri Nets with Objects (PNO) - tokens contain references to OO data structures and **Cooperative Objects (COO)** - objects with Object Control Structure (OBCS) - **CASE tool SYROCO**
- **Lakos -** Object Petri Nets (OPN) developed from Colored Petri nets through a serious of formal transformations that make **Object Petri Nets** behaviorally equivalent to Colored PNs - **CASE tool LOOPN**
- **Buchs & Guelfi -** Concurrent Object-Oriented Petri Nets; an object has an internal behavior defined by an algebraic net - **CASE tool CO-OPN/2**
- **Valk -** *Relating Different Semantics of Object Petri Nets*, Report, 2000, Petri Nets as Dynamic Objects; **Communicating OPNs**
- **Moldt - Object CPN** - an extension of CPNs

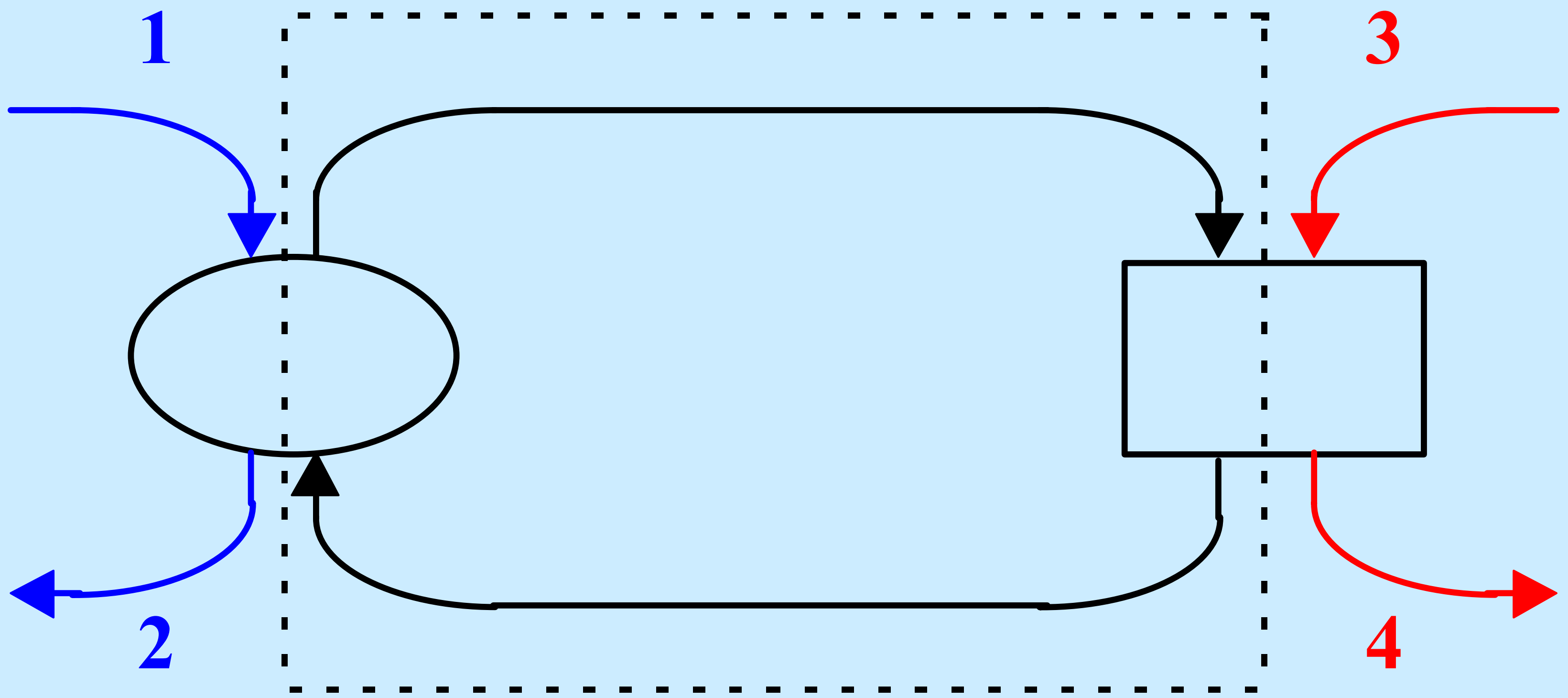
Integration of Object-Orientedness and Petri Nets - the hybrid approach



Abstract Node

- **Symmetry and Abstraction Constructs:**
  - asymmetry of existing abstraction constructs (Lakos, Petri)
  - non-unified constructs (abstract places/abstract transitions; do not emphasize the duality of places and transitions)
  - unified abstraction construct **- Abstract Node (AN)**
- **Abstract Node:**
  - AN as an abstract place and AN as an abstract transition
  - constructed by connecting a place (AN-place) and a transition (AN-transition) by two arcs with parametrizing inscriptions in a loop
  - duality between sets {1,2} and {3,4} of arcs as duality between abstract places and abstract transitions
  - if arcs {1,2} are used for embedding an abstract node into the net then it behaves like abstract place; if arcs {3,4} are used then it behaves as an abstract transition

Abstract Node



Abstract Node versus Abstract Places and Abstract Transitions

- **Natural Extension of Places and Transitions to Abstract Places and Abstract Transitions:**
  - **abstract places** may store but not modify tokens
  - **abstract transitions** may modify but not store tokens
- **Desired Solution:**
  - **abstract places** need to modify tokens (with token conservation)
  - **abstract transitions** may store tokens with some restrictions applied (atomicity of internal actions)
  - AN-place can store tokens and AN-transition can modify them
  - the modification of tokens by AN (that acts as abstract place) is done by firing AN-transition, which results in a change of color of the token of the AN-place

Abstract Node versus Abstract Places and Abstract Transitions, ctnd.

- **the internal state of AN** (that acts as abstract transition) is stored in the AN-place; AN-transition can modify an internal state and synchronizes the actions associated with its external incident arcs (arcs 3 and 4)
- **AN can be constructed using regular CPNs** without any modifications or additions; the level of abstraction can be varied by changing inscriptions of abstract node's arcs (AN-arcs) and a color set of AN-place
- **AN is the highest level of object abstraction** which is refined to interfaces and implementations

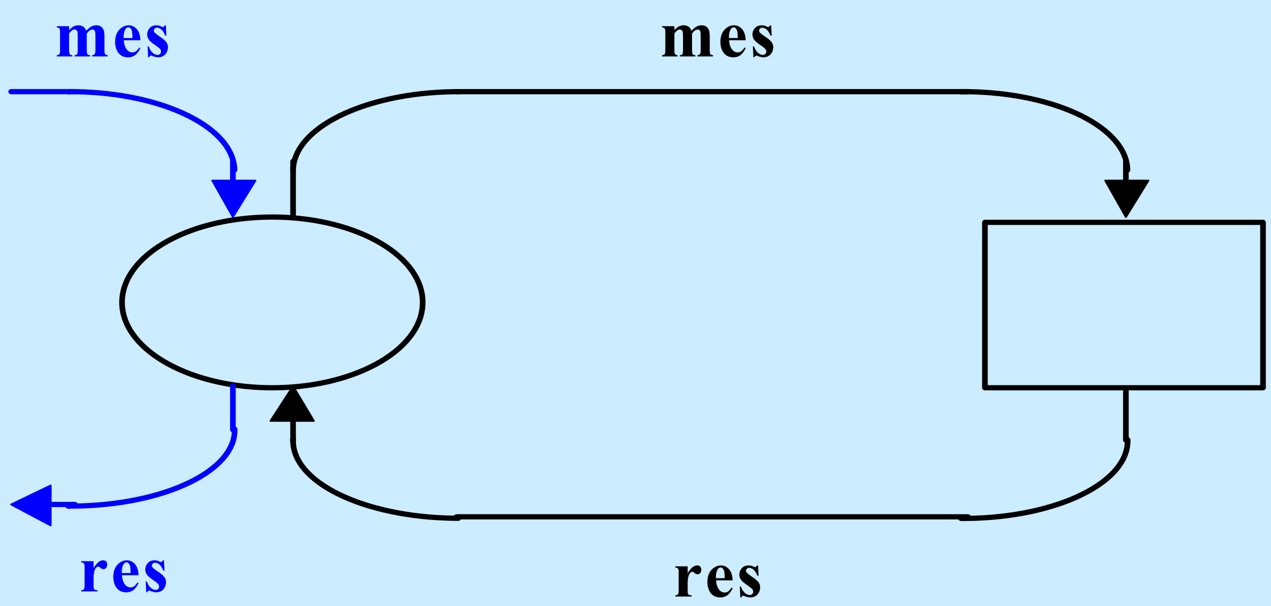
Abstract Node and Objects

- **Abstract Node as an Object:**
  - AN encapsulates both data and actions (tasks)
  - AN is the highest level abstraction of an object
  - single object level is not the highest possible level of abstraction for a system
- **Object Composition:**
  - objects can be combined to form **aggregates**
  - aggregate is an object, thus can be represented by a single AN
  - **aggregation is application specific** but it is supported by AN formalism
  - **semantics of aggregation** is supported by AN syntax

Abstract Node and Objects

- **AN are used to represent objects**
- **AN-place:**
  - serves as a message depository (incoming arcs)
  - serves as a place for retrieving results (outgoing arcs)
- **Distinguishing Between Message Tokens (Requests) and Result Tokens (Results):**
  - **message color set:** (object ID, message type, arguments)
  - **result color set:** (object ID, message type, result)
- **Object ID:**
  - unique object ID for all instances (objects) in the system
  - inter-object concurrency (instances can execute concurrently among themselves)
- **Message type used to:**
  - distinguish between those actions within a particular object
  - intra-object concurrency (concurrency between messages)

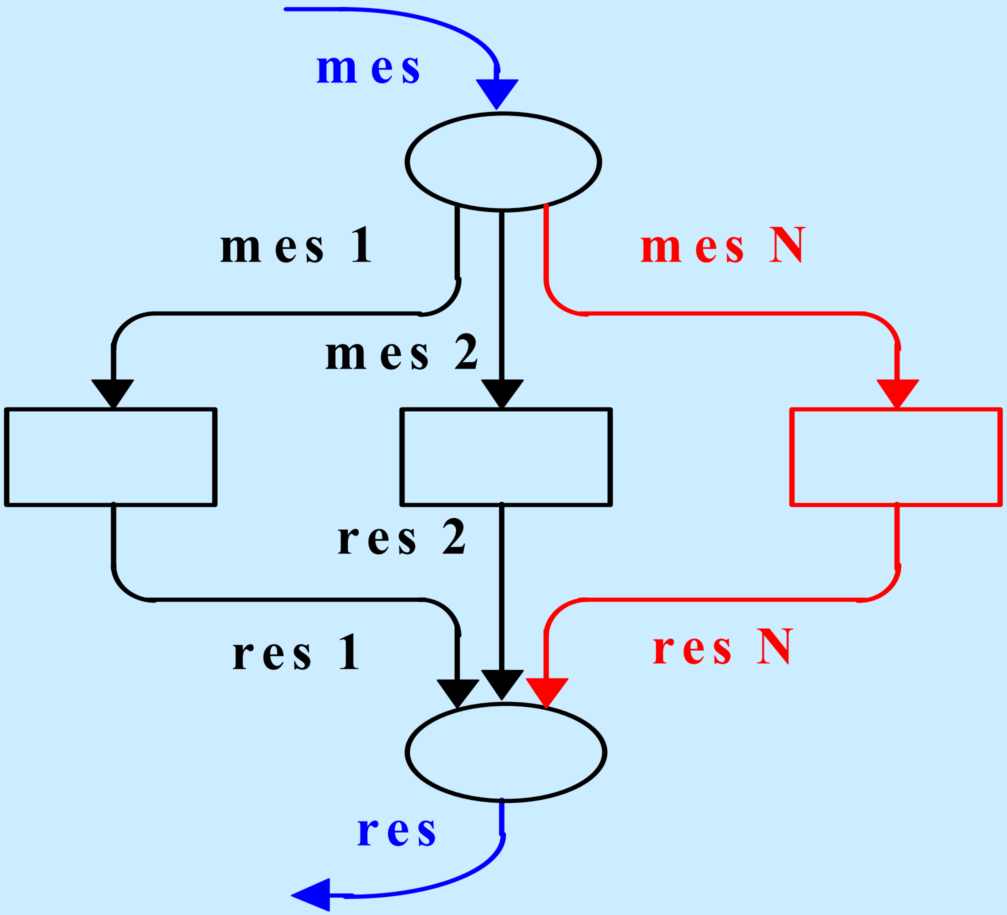
Abstract Node and Objects



Interface - First Level Refinement

- **One Object as an AN is Too Abstract**
- **Interface Refines AN in Two Ways:**
  - **it splits AN-place into two places**: the message depository place and the place for retrieving results
  - **it splits AN-transition** to as many transitions as the number of messages the object accepts
- **Each Transition (I-transition) Represents One Action of an Object**
- **Further Refinement of each I-transition is called an Implementation**

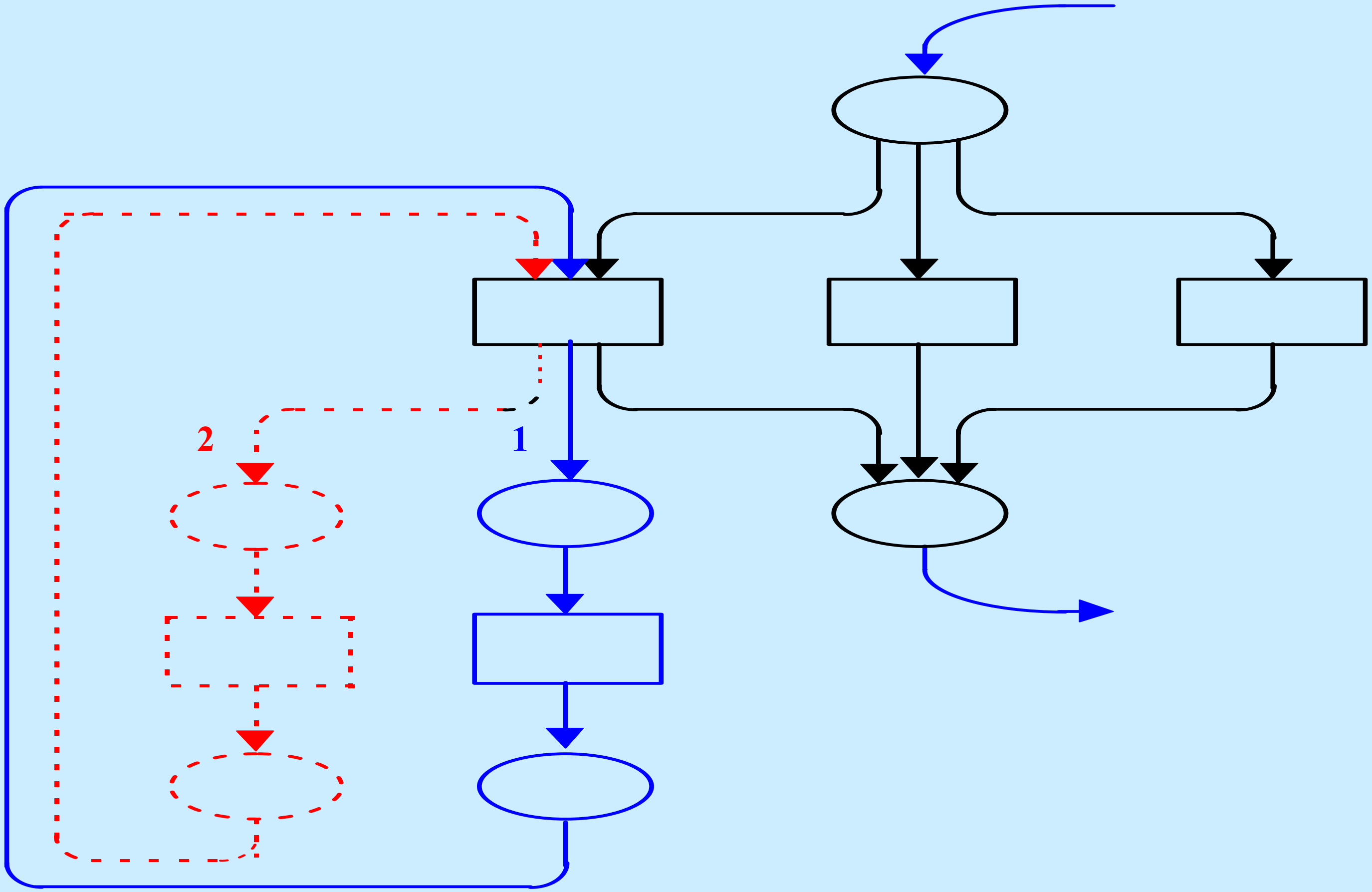
Interface - First Level Refinement



Method Implementations - Second Level Refinement

- **Implementations Provide the Most Detailed Representation of an Object**
- **Each Implementation Refines One Message Response:**
  - as sequential
  - as parallel; intra-object concurrency
  - as problem specific
- **Multiple Implementations:**
  - first implementation **solid blue**
  - second implementation **dotted red**
- **Binding and Arc Inscriptions:**
  - **by arc inscriptions:** arc 1 and arc 2 - can be static or dynamic (polymorphism and dynamic behavior of objects)

Method Implementations - Second Level Refinement



Inheritance versus Delegation

- **Inheritance. Effects on the Methods at a Class Level:**
  - **include the method** of a parent class without changes into a subclass
  - **do not include the method** of a parent class into a subclass
  - **modify the method** of a parent class in the subclass
  - **add a new method** to a subclass
- **Delegation. Object Level - Instances of Classes:**
  - **messages that are processed** without changes in a subclass are delegated to a parent class (the message is passed to a parent class)
  - **messages that are not processed in a subclass** do not have implementation
  - **modified methods** have new implementation and if needed can call a parent class
  - **new methods** have a new entry for both interface and implementation
  - **both single and multiple inheritance** can be implemented in this manner

Inheritance Anomaly

- **Single and multiple inheritance** require careful attention to avoid incorrect behavior of inherited class instances being a result of class interference on dynamics (behavior) of new inherited classes
- **three types of corrective actions to avoid inheritance anomaly:**
  - **state partitioning**
  - **state modification**
  - **history sensitiveness**
- all three types of inheritance anomaly can be cured with **modified pre-conditions and post-actions of methods** and with proper changes in dynamic behavior of objects (modified state diagrams or modified OBCS)
- **inheritance anomaly** can be assimilated into the method of OOD and PN integration by a modified implementation of methods with modified guards and modified effects of methods

Polymorphism and Dynamic Behavior of Objects

- **Objects have an interface and one or more implementations of each method**
- **Interfaces, Java Interfaces, Abstract Classes:**
  - multiple implementations of methods are viewed as multiple implementations of an interface (or abstract class)
- **Polymorphism:**
  - by doing the binding by the object ID we enrich the method with polymorphism of methods where object ID is unique number
  - **Object ID** has two fields: (unique class number, unique for each class instance number)
  - **polymorphism** is a binding by class number
- **Dynamic behavior:**
  - binding based on some variable in the arc inscription that is changed during the execution of an object

Conclusions

- **OO an PN are used without major modifications:**
  - method is just a set of rules and modeling with Petri nets
  - rules are expressed by means of Colored Petri Nets
  - rules can be expressed as a set of templates (being part of CASE tool)
  - library of commonly used objects can be another objective
  - SYROCO and LOOPN CASE tools were used to test this approach
- **Objects can be Created and Verified Separately:**
  - design/verification of single object level
  - design/verification of object interaction level

References

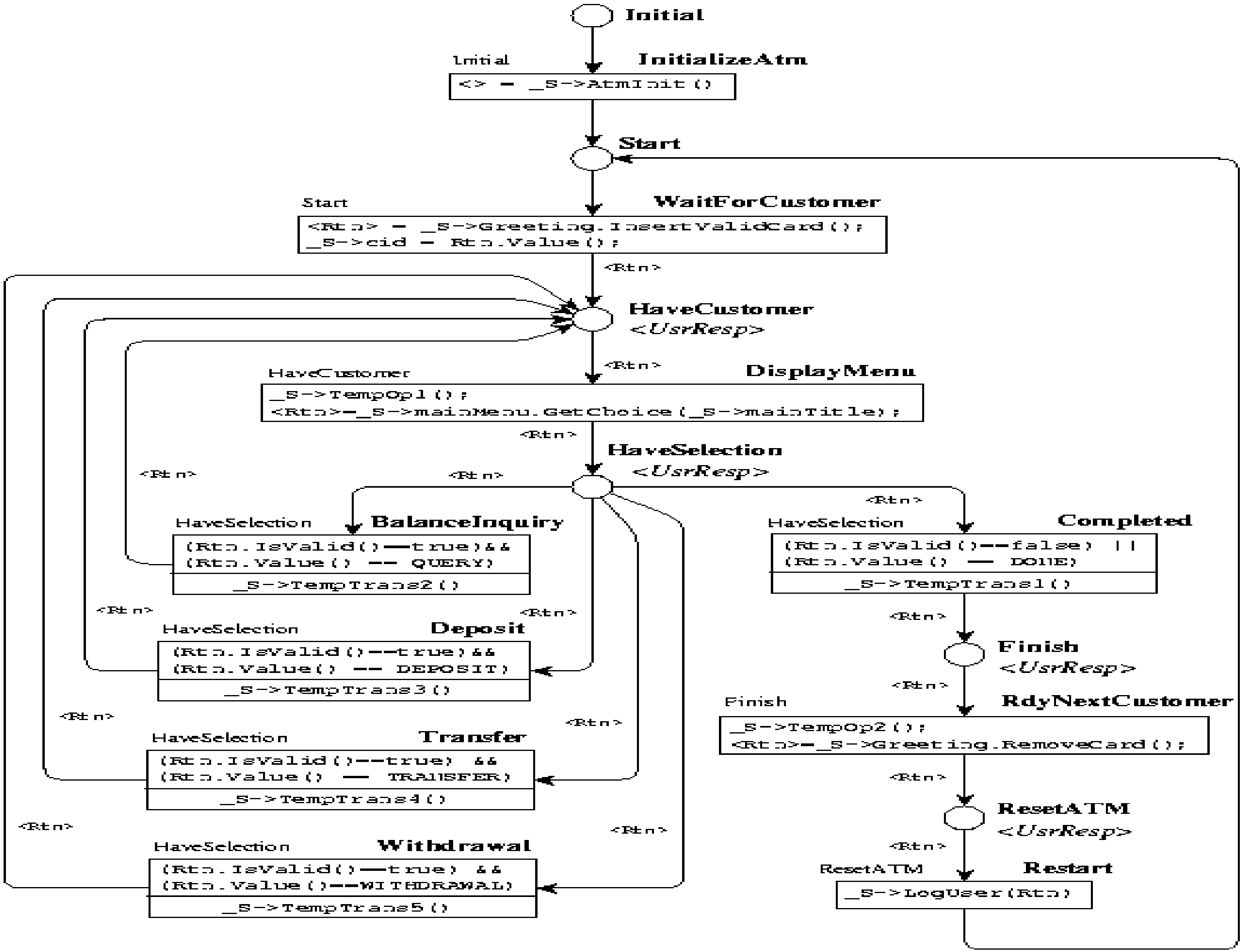
- C. Girault, R. Valk, *Petri Nets for Systems Engineering, A Guide to Modeling, Verification and Applications*, Springer, 2003.
- G. Agha, F. De Cindio, G. Rozenberg, *Concurrent Object-Oriented Programming and Petri Nets*, *State of the Art Survey*, Advances in Petri Nets,, LNCS 2001, Springer, 2001.
- D. Mukhin, B. Mikolajczak, *A Method of Concurrent Object-Oriented Design Using High-Level Petri Nets*, 1998 IEEE
- M. Ceska, V. Janousek, T. Vojnar, *PNtalk - A Computerized Tool for Object Oriented Petri Nets Modeling*, LNCS, v. 1333, Springer, 1997.
- A. Cabeza, UMD, Master Thesis, 1999
- H. Hsueh, UMD, Master Thesis and Project, 2000
- Ch. Sefranek, UMD, Master Project, 2000
- B. Mikolajczak, Z.Wang, *Structural and Behavioral Properties of Petri Net Morphisms*, Springer-Physica, Advances in Soft Computing, Springer, 2003

Future Work

- **testing the method** using average size example such as ATM System from Wirfs-Brock, Wilkerson, Wiener, *Designing Object-Oriented Software*, 1990
- using Petri net **vicinity preserving or general morphisms** that preserve certain structural and behavioral properties to provide abstraction and refinement building mechanisms during system specification by PNs (as a set of template transformations)

The ATM Machine

- The **ATM Class** OBCS Diagrams:
- - The **Root ATM** OBCS
- - The **ATMInit** Service OBCS
- The **BankCardReader** Class OBCS Diagrams:
- - The **Input()** Service
- - The **Eject()** Service
- The **Form** Class OBCS Diagram
- The **Menu** Class OBCS
- The **User Message** Class OBCS:
- - The **InsertValidCard** Service
- - The **RemoveCard** Service



Initial
Initial
InitializeAtm
<> = _S->AtmInit()
Start
Start
WaitForCustomer
<Rtn> = _S->Greeting.InsertValidCard();
_S->cid = Rtn.Value();
<Rtn>
HaveCustomer
<UsrResp>
<Rtn>
HaveCustomer
DisplayMenu
_S->TempOp1();
<Rtn>=_S->mainMenu.GetChoice(_S->mainTitle);
<Rtn>
HaveSelection
<UsrResp>
<Rtn>
<Rtn>
<Rtn>
HaveSelection
BalanceInquiry
(Rtn.IsValid()==true)&&
(Rtn.Value() == QUERY)
_S->TempTrans2()
HaveSelection
Completed
(Rtn.IsValid()==false) ||
(Rtn.Value() == DONE)
_S->TempTrans1()
<Rtn>
<Rtn>
HaveSelection
Deposit
(Rtn.IsValid()==true)&&
(Rtn.Value() == DEPOSIT)
_S->TempTrans3()
<Rtn>
Finish
<UsrResp>
<Rtn>
Finish
RdyNextCustomer
_S->TempOp2();
<Rtn>=_S->Greeting.RemoveCard();
<Rtn>
<Rtn>
HaveSelection
Transfer
(Rtn.IsValid()==true) &&
(Rtn.Value() == TRANSFER)
_S->TempTrans4()
<Rtn>
ResetATM
<UsrResp>
<Rtn>
<Rtn>
<Rtn>
HaveSelection
Withdrawal
(Rtn.IsValid()==true) &&
(Rtn.Value()==WITHDRAWAL)
_S->TempTrans5()
ResetATM
Restart
_S->LogUser(Rtn)