What Test Designers Wish from Software Models

David Gelperin
Software Quality Engineering
sqegelp@aol.com

dgelperin@sqe.com

Introduction

Testing is getting harder. To keep up with the every growing complexity of software and the testing task, newer approaches to test development must be used. One of these approaches is automated test design based on precise models of software behavior and usage.

Today, commercial support in this area is weak. There is a current and growing need for a **variety** of commercial tools supporting automated design.

The purpose of this presentation is to provide insight into the information requirements for automated test design to (current and potential) vendors of software modeling tools. It is hoped that this information will catalyze a number of commercial development efforts.

What Test Designers Wish from SW Models

David Gelperin

Software Quality Engineering

sqegelp@aol.com or
dgelperin@sqe.com

Wishes Support Automated Design

In scope:

Info reqs for automated test design

Out of scope:

* State info reqs (already adequate in UML)
* Info on automated test design process

Problem

- Today’s (and tomorrow’s) mission critical systems **require advanced test design** techniques
- These techniques entail (1) development of a behavior or usage model and then (2) automated design or implementation from that model
- Commercially available support for these techniques is weak to non-existent
- Testers of financial, insurance, industrial control and other critical systems need automated design support [The new market]

Opportunity

To provide a tester's "modeling bench" supporting **multiple forms** of:

- ultra-understandable modeling
- automatic model verification
- automatic test design
- automatic test implementation

Which Models interest **Test**?

Software aspects of **interest to test** include:

- **actions**
- **behavior** rules
- **usage** scenarios
- surface (black-box) **structure**
  - interfaces
  - input & output data

Some Choices

(1) Extend (23) UML modelers to support test design and implementation

(2) Develop independent products

SQE Goals

- To promote preventative testing
- To drive automated support for advanced test design
- To make a difference
  - We make a difference, when you make a difference

What Test Designers wish1 from SW Models

Wish 1 - Function Inventories with Semantics

- Action Lists
- Post-conditions

What does the software do?

Action Lists

For a particular system or component (being modeled), what are **all of the functions** (methods) that it performs?

Wish for a hierarchically organized index with unique identifiers **for each action**

Example of an Action List

<u>Sleep-Sound Motels Reservation System Example</u>

RS. R 001,2,3	Add, Change, Delete Reservation
	Display, Report
RS. R 004,5	Single Reservation
RS. R 006,7	Res List by Customer
RS. R 008,9	Res List by Location and Date
	Report
RS. R 010	Reservation Changes and Deletions
RS. R 011	Reservation Agent Activity
RS. R 012,13,14	Add, Change, Delete Frequent Sleeper
	Display
RS. R 015	Single Frequent Sleeper
RS. R 016	Frequent Sleeper List
RS. R 017	Report Frequent Sleeper Activity
RS. R 018,19,20	Add, Change, Delete Motel Location
	Display
RS. R 021	Single Motel Location
RS. R 022	List of Area Motel Locations
RS. R 023	Report Motel Location Activity

The “meaning” of an action (process)

Let R be a routine that inserts entries into a table of bounded capacity. Each entry has an associated key which must be a non-empty string and unique in the table.

Question: If as input, R is provided with an entry **e** having a valid key **e-key** relative to the current state of the table and R executes correctly, what are some things that must be true following execution i.e., what does it mean for R to execute correctly?

Answers: Entry (e-key) = e

& **end** count = **begin** count + 1

& e-key is not empty

& 0 <= count <= capacity

Some conditions

For the table management routine R,
a successful insertion could be modeled as:

Pre-conditions

begin count < capacity

Post-conditions

Entry (e-key) = e
& **end** count = **begin** count + 1

Invariants

e-key is not empty
& 0 <= count <= capacity

4 types of **conditions**

1) **pre-condition -** Must be true at a specified **begin** point for correct operation
2) **intermediate condition -** Always true at a specified **intermediate** point if correct operation
3) **post-condition -** Always true at a specified **end** point if correct operation
4) **invariant -** Always true **everywhere** if correct operation

Post-conditions are meaning

- The **proximate meaning** of a process **is its effect**
- Conditions specify attribute values and relationships as well as temporal constraints on inputs, results, and system states
- Modeling systems should support **easy specification** of pre & post conditions **for each** action / function / process in a model

Post-condition **References**

Meyer, Bertrand "Applying Design by Contract", IEEE Computer Vol 25 No 10 October 1992 pp 40-51

Meyer, Bertrand "Building bug-free O-O software: An introduction to Design by Contract" [Available at http://eiffel.com/doc/manuals/technology/contract/index.html]

Warmer, Jos and Kleppe, Anneke The Object Constraint Language: Precise Modeling with UML Addison-Wesley 1999

What Test Designers wish2 from SW Models

- Wish 1 - Function Inventories with Semantics
- **Wish 2 - Behavior Rules**
  - Decision Tables
  - Effect Tables
  - Extended Action Tables
  - State Models of Function (already in UML)

When does the software do?

What is a **Functional Model**?

- A consistent **set of rules** for correct behavior
- A behavior rule specifies:
  - an **input situation / condition**
  - the corresponding **required response**

  Answers: **When does the software act?**
- Ultra-understandability implies familiar, but precisely defined terminology (i.e., use of a **data dictionary** for objects, attributes, values, conditions, and actions)

This is a **Decision Table**

Rules for vending machine behavior

Selections Available	Selection	Deposit Amount	Return Lever	Return	Dispense
None	not made	> 0	---	Deposit	
Some or All	Not Made	> 0	Depressed	Deposit	
Some or All	Made	> 0 & < Price	Depressed	Deposit	
Some or All	Made	= Price	---		Selection
Some or All	Made	> Price	---	Change	Selection

Otherwise, Return and Dispense nothing

What is a **Decision Table**?

- A tabular specification of **a set of decision rules**
- Each decision rule specifies a set of one or more actions that should be performed when a specific **conjunction of simple conditions** (e.g., A & B) is True
- A **simple condition** is a logical statement (i.e., one that is either True or False) that contains neither "and" nor "or", but may contain "not". For example, temp > 98.6, payment not overdue, zip-code = 55427 are all simple conditions

Dependent Conditions

- Truth values of two conditions (simple or compound) may be **dependent** i.e., linked sometimes or always

- There are three forms of logical dependency:

  Equivalence A = B i.e., two conditions always have the same truth value e.g., (x GT y) and (y LT x)

  Opposition C = not D i.e., two conditions always have opposite truth values e.g., (x GT y) and (x LE y)

  Implication A ➔ C i.e., C must be True whenever A is True, but may be either True or False, otherwise

- **All dependencies should be documented and validated**

Examples of Implication

1) (x GT 10) ➔ (x GT 0)

2) If "order is valid" implies **order quantity > 0** and
"out of stock" implies **on-hand quantity = 0** and
"insufficient stock" implies
order quantity > **on-hand quantity**

then
order is valid AND out of stock ➔ insufficient stock

Properties of Decision Tables

- A decision table is **consistent** if and only if every situation is covered by at most one rule.
- A decision table is **relatively complete** if and only if every modeled situation is covered by at least one rule and all required actions are included. Some tables may be completed by an "otherwise" rule.
- Consistency and completeness should **always be checked**

Another Decision Table **Example**

Briefing on

Pass Orders System

Decision Table

Usage of Decision Tables

- **Application Domain** -- Complete analysis of smaller complex decision patterns or partial analysis of bigger ones
- **Testing Levels** -- Most appropriate for component and component integration due to large table size at higher levels
- **When to use** -- Always use, when decision tables already exist or when determining the completeness of a set of decision rules is very important. Consider using when the combinations of condition values fit on no more than two pages and can be easily read.
- **Prerequisites** -- Table development requires time, analysis skill, and availability of decision rule information

Decision Table **Automation**

- General Modeling
  + **Logic Gem** by Logic Technologies

    www.logic-gem.com/lg.htm
  + **TurboCASE/Sys** by StructSoft

    www.turbocase.com/features.html
  + Other older CASE tools (?)

Decision Table **References**

Beizer, Boris **Software Testing Techniques** Van Nostrand Reinhold 1990, Chapter 10, pp. 322-332 [Traditional description]

What is an **Effect Table**?

- An **effect condition** is a set of conjoined simple conditions sufficient **to elicit a specific effect** such as an action, output, next state, or post-condition
- An **effect rule** specifies an effect condition and the elicited effect (i.e., if effect condition, then effect)
- An **effect set** is a set of one or more effect rules
- An **effect table** is a set of effect rules for eliciting **a single effect**
- A **complete effect table** is the set of **all** effect rules for eliciting a single effect (i.e., effect if and only if one or more of the effect conditions)

4-Rule Effect Table **Example**

Selections Available	Deposit	Return Lever	EFFECT
None	Made	---	Return Deposit
Some Or All	Made	Pushed	Return Deposit

Another Effect Table **Example**

Briefing on

Pass Orders System

Effect Tables

Intersecting Effect Sets

Selections Available	Amount Deposited	Selection	EFFECT
Some	Too Much or Exact	Made	Dispense Soda
Some	Too Much	Made	Return Change

Two effect **rules co-apply** if and only if they have identical effect conditions. Two **sets of effect rules intersect** if and only if their union contains at least one pair of co-applying rules.

Unifying Effect Sets

- Two **sets of effect rules are disjoint** if they do not intersect
- **Unification** is the process of determining whether two sets of effect rules intersect and either providing all co-applying rules (the **unification set**) or reporting that the sets are disjoint
- Unification of **three or more effect sets** starts with the unification of two effect sets and continues with the unification of the resulting unification set with another effect set

Differencing Effect Sets

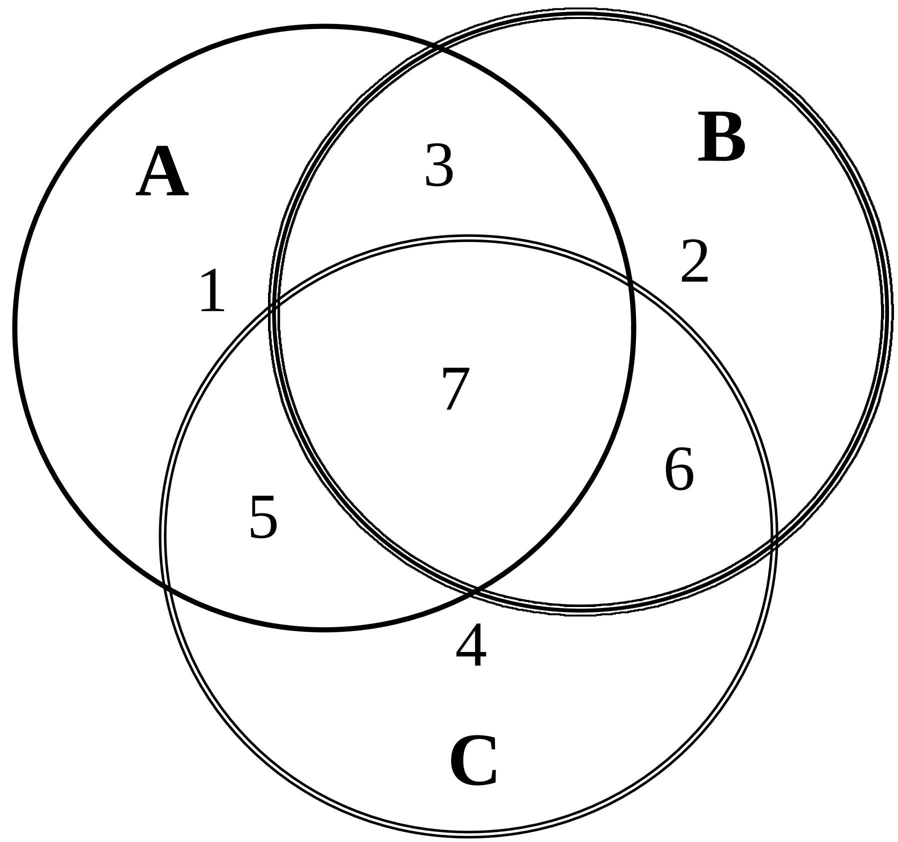
- Two effect **rules are disjoint** if they do not co-apply
- **Differencing** one set S of effect rules with respect to another set T of effect rules (S-T) is the process of determining whether S has any rules that are disjoint from those in T and either providing all such rules (the **difference set**) or reporting that all rules in S co-apply with those in T i.e., S is a unification set for S and T
- Differencing of **three or more effect sets** starts with the differencing of two effect sets and continues with the differencing of the resulting rule set with another effect set

Picture of Unifying & Differencing

If A, B, & C are effect tables, then

- Unifying A+B+C yields all the rules in 7
- Differencing A-B yields all the rules in 1+5. Differencing (A-B)-C yields all the rules in 1
- Differencing (B-C)-A yields all the rules in 2
- Differencing (C-A)-B yields all the rules in 4

Note that any of these unification or difference sets may be empty i.e., contain no pairs of co-applying rules



What is an **Extended Action Table**?

- An **action table** is an effect table where the single effect being elicited is an action
- An **extended action table** (or effect table) includes post-conditions as well as pre-conditions

Extended Action Tables **Example**

Briefing on

Pass Orders System

Extended Action Tables

Usage of Effect Tables

- **Application Domain --** Larger complex decision patterns
- **Testing Levels --** Appropriate for interoperability, system, component integration, and component testing
- **When to use --** Always use, when effect tables already exist. Consider using when the combinations of conditions (i.e., a decision table) will not fit on two pages and be readable.
- **Prerequisites --** Table development requires time, analysis skill, and availability of effect rule information

Effect Table **Automation**

- Test Generation
  + **SoftTest** by Bender & Associates

    www.softtest.com

    Input model is cause-effect graph. SoftTest translates to limited entry decision table and generates tests

Cause-effect graphs can be developed from effect tables and vise versa.

What Test Designers wish3 from SW Models

- Wish 1 - Function Inventories with Semantics
- Wish 2 - Behavior Rules
- **Wish 3 - Explicit Usage Patterns**
  - Refined Use Cases (partially in UML)
  - State Models of Usage (already in UML)
  - Grammars

Usage Classification Schemes

For most products, even those with a modest number of functions, the number of possible (effective and ineffective) usage scenarios is enormous, while the **number of scenarios actually tested can be quite large** (in the hundreds or thousands)

This means that **understanding and management** of a suite of usage test scenarios **require a clear organizational framework**

Classification Hierarchies

Usage scenarios should be hierarchically organized (i.e., tasks and subtasks) at differing levels of abstraction and be complete at each level

E.G. [Travel Information and Reservation System]

User goal: Arrange trip to grandmas

- **US 1.0 - Arrange air**
  - **US 1.1 - Determine air options and prices**
  - **US 1.2 - Make reservations for family**
  - **US 1.3 - Choose seats**
- **US 2.0 - Arrange hotel**
  - **US 2.1 - Determine hotel options and prices**
- **US 3.0 - Arrange ground transportation**

Developing a Usage Classification

Development involves the successive decomposition of a usage description resulting in a series of nested profiles (sub-models) with increasing granularity of detail.

For example:

- 1. Acquiring groups (Brokerage Back Offices, Bank Trust Depts)
- 2. User groups (Head Traders, Compliance Officers, Traders)
- 3. Usage purposes (enroll customers, process complaints)
- 4. Usage/System modes (naïve vs. experienced, normal vs. overloaded)
- 5. System functions (find customer, report activity, enter complaint)
- 6. Operations (search, setup, update)

Usage Classification **References**

Musa, John D “Operational Profiles in Software-Reliability Engineering” IEEE Software, March 1993, pp. 14-32

Musa, John Software Reliability Engineering McGraw-Hill 1999, Chapter 3

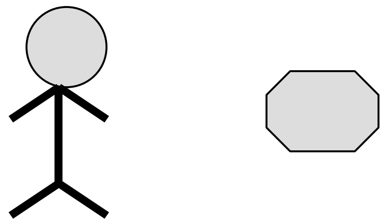
Example of Use Cases

Briefing on

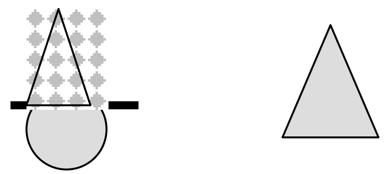
The Pass Orders System

Use Case Model

Who's Who in Use Cases



- **Complete Characters**



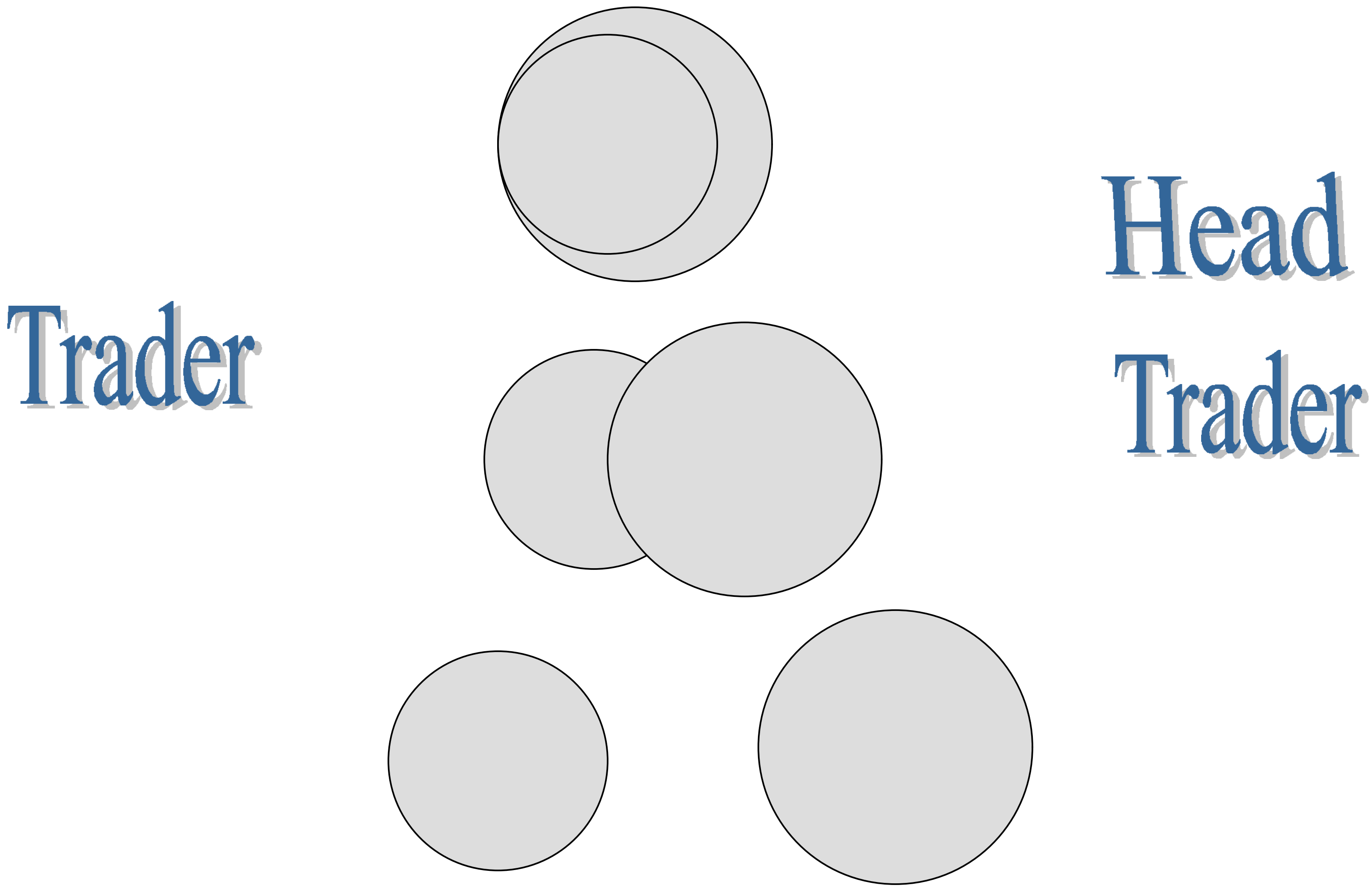
- **Partial Characters**

- **Actors**: People or specific systems that play one or more characters
- **Users**: Refers to either actors or characters

Characters

- **Complete Character**: An actual position responsible for specific actions on specific objects to meet enterprise objectives
- Two complete **characters are disjoint** if either (1) their action sets are disjoint or (2) for each common action, the associated object sets are disjoint. If two complete **characters are** not disjoint then they are **overlapping**
- **Partial Character**: An abstract position that encompasses the common part of two or more overlapping characters

Character Overlap



Character **Profile**

Specification that includes:

- Name
- Category [Person, System, or both; Complete or Partial]
- Abstract of responsibilities
- Relationships (e.g. for partial characters, the complete characters that contain them)
- Enterprise Locations (where on the org chart?)
- Systems Used and Actions & Roles per system

Components of a Use Case

- Set of one or more **users**
- Set of one or more system **activities**
- **Relationship** between the system and users during an activity
- Pre- and post**-conditions** for the use case (i.e., a post-condition behavioral model). For example, preconditions assuming successful completion of other use cases.

Typing Users by Interactivity

Distinguish 3 types of users based on the interactivity of the system relationship

- **P -- providing** users (only provide input or cause trigger events)
- **I -- interactive** users (both provide & receive)
- **R -- receiving** users (only receive output)

Describing Activities

Use case activities can be described with increasing granularity and precision of detail using one or more of the following:

- Natural Language
- Decision/Effect Tables
- State Models
- Pseudo Code

Usage of Use Case Testing

- **Application Domain** -- Software having a variety of users, usage modes (e.g., internet usage) or usage styles (e.g., novice vs. power user)
- **Testing Levels --** Most appropriate for acceptance, interoperability, system, and component integration testing
- **When to use --** Always use, when use cases already exist. Consider developing cases as a means of identifying, specifying, and testing critical scenarios.
- **Prerequisites --** Use case development requires time, analysis and specification skill, and the availability of usage information. Automation helps.

Use Case **Automation 1**

- Test Generation
  - + **Validator** by Aonix

    www.aonix.com/Products/SQAS/sqas.html

    Input model is a use case spec from which the product generates executable scripts, data sets, requirements and test specs and suite profile reports
  - + **Test Mentor / UML Designer Connection** by Silvermark

    www.silvermark.com/STM/umlconn.htm

Use Case **Automation 2**

- Use Case Modeling
  + **HOW** by Riverton Software

    www.riverton.com/product/model.htm
  + **ObjectModeler** by Iconix Software

    www.iconixsw.com/Spec_Sheets/ObjectModeler.html

- General Modeling in UML
  – References on next 2 pages

Use Case **References**

Berard, Edward **Be Careful With "Use Cases"** (www.toa.com/pub/html/use_case.html)

Firesmith, Donald **Use Cases: the Pros and Cons** (www.ksccary.com/usecjrnl.htm)

Jacobson, Ivar et.al. **Object-Oriented Software Engineering** Addison-Wesley 1992, Chapter 7, pp. 153-174 [Original description]

Korson, Tim **Misuse of Use Cases**

(www.software-architects.com/publications/korson/Korson9803om.htm)

Schneider, Geri and Winters, Jason P. **Applying Use Cases** Addison-Wesley 1998

This is a Toy Grammar

$$
\begin{aligned}
S &\rightarrow NP + VP + . \\
NP &\rightarrow Art + Adj_0^3 + N \\
VP &\rightarrow (Vt + NP) \mid (Vi + Adv) \\
Art &\rightarrow \text{the} \mid \text{a} \\
Adj &\rightarrow \text{green} \mid \text{small} \mid \text{expensive} \\
N &\rightarrow \text{package} \mid \text{boy} \mid \text{idea} \\
Vt &\rightarrow \text{carried} \mid \text{lost} \\
Vi &\rightarrow \text{snored} \mid \text{grew} \mid \text{ran} \\
Adv &\rightarrow \text{quickly} \mid \text{furiously}
\end{aligned}
$$

What is a **Grammar**?

A set of **production rules** that describes/defines a language. Each rule has a non-terminal symbol on the left and a set of one or more alternative (|) strings on the right. A **non-terminal symbol** is an abstraction that does not appear in the language. A **string** is composed of one or more concatenated (+) symbols. Strings are either terminal (i.e., contain only terminal symbols) or non-terminal. The **language** contains all terminal strings generated by the grammar (and nothing else).

$$NP \rightarrow Art + Adj_{0}^{3} + N$$

NP	→	Art + Adj_{0}^{3} + N
VP	→	(Vt + NP) | (Vi + Adv)
Art	→	the | a
Adj	→	green | small | expensive
N	→	package | boy | idea

This is another Grammar

LocalMNPhoneNumber	$\rightarrow$	Prefix + - + Number^4
Prefix	$\rightarrow$	377 | 546 | 591
Number	$\rightarrow$	1| 2 |3 | 4 | 5 | 6 | 7 | 8 | 9

377-7777 is in the language of this grammar

Subscripts & Superscripts

- **Subscripts** on a symbol denote a minimum number of repetitions of that symbol. Zero means that it may be omitted. Without a superscript, one is the default.
- **Superscripts** denote the exact number of repetitions, when they appear alone, and the maximum number of repetitions when they appear with a subscript. One is the default.

Usage Modeling with Grammars

- A set of input event sequences can be viewed as sentences in a usage language
- This usage language can be described by a grammar
- The grammar can then be used to generate both valid and invalid input event sequences

Example of a Usage Grammar

User Session $\rightarrow$ LogOn + Applications_0^3 + LogOff

Applications $\rightarrow$ StartApp1 + App1Seqs_0^{10} + StopApp1 | …..

App1Seqs $\rightarrow$ findTask_Alt_1^5 | $\text{updateTask\_Alt}_1^3$ | …..

updateTask_Alt $\rightarrow$ updateTask&Check + Interrupt_0^1

updateTask&Check $\rightarrow$ updateTask1 + Check_updateTask1 | …..

Interrupt $\rightarrow$ App1Seqs | Applications

Check_updateTask1 is a non-terminal for immediately checking task results, i.e., an embedded oracle

String **Generation**

The process of

(1) selecting a production rule with a "start symbol" on the left (S in our example) and

(2) continuing to use other production rules to replace the non-terminals in the developing non-terminal string (i.e., contains at least one non-terminal symbol) until only terminal symbols remain

Different rule choices lead to different terminal strings. The set of all strings that can be generated by a grammar define the strings in the "language" of that grammar.

Systematically Break Rules 1

- At multiple sites within multiple rules
- But just a little (e.g., only one site in one rule)

Examples using

$$NP \rightarrow Art + Adj_0^3 + N$$

1) <u>Wrong Order Bugs</u> - Exchange any two symbols and block omissions

$$NP^* \rightarrow Art + N + Adj_1^3 \qquad \text{1 of 3}$$

2) <u>Omission Bugs</u> - Delete a symbol that can not be omitted

$$NP^* \rightarrow Art + Adj_0^3 \qquad \text{1 of 2}$$

Systematically Break Rules 2

3) Extraneous Bugs

a. **More of the same --** Increase superscript by 1

NP* $\rightarrow$ Art ^{2}+ Adj $^{3}_{0}$+ N 1 of 3

b. **Outsiders --** Put any valid symbol not already in the rule at the beginning, end, or any position in the middle

NP* $\rightarrow$ Art + Adv + Adj $^{3}_{0}$+ N 1 of BigNum

b. **Foreigners --** Put any invalid symbol at the beginning, end, or any position in the middle

NP* $\rightarrow$ Art + Adj $^{3}_{0}$+ N + end 1 of HumNum

Usage Grammars Can

Generate patterns of usage event sequences

- **valid** usage **--** with the valid grammar
- **invalid** usage **--** with "broken rule" grammars
  - Choose a rule in the valid grammar and substitute one of its broken rules
  - While always using the broken rule, generate strings

Usage of Usage Grammars

- **Application Domain --** Software having a large number of usage scenarios
- **Testing Levels --** Most appropriate for acceptance, interoperability, system, and component integration testing
- **When to use --** When determining a grammar for the usage patterns is feasible
- **Prerequisites --** Grammar development requires time, analysis skill, and the availability of string generation automation

Tool for Usage Grammar Modeling

- Test Generation

  + **CleanTest** (prototype)
  by Cleanroom Software Engineering, Inc

  www.cleansoft.com/cleansoft/cleantest.html

  Input model is a usage profile specified as a tree of activity nodes from which the product generates test input in the form of activity sequences

- General Modeling with Grammars

  No known commercial products that generate “language strings” for arbitrary grammar specifications

Usage Grammar **References**

Maurer, Peter M. "Generating Test Data with Enhanced Context-Free Grammars" IEEE Software July 1990 pp. 50-55

Miller, B.A. and Pleszkoch, M.G. " A Cleanroom Test Case Generation Tool" in Poore, J.H. and Trammell, C.J. Cleanroom Software Engineering: A Reader, NCC Blackwell 1996 pp 269-286

What Test Designers wish4 from SW Models

- Wish 1 - Function Inventories with Semantics
- Wish 2 - Behavior Rules
- Wish 3 - Explicit Usage Patterns
- **Wish 4 - Surface Structure**
  - UI Maps
  - Data Dictionaries

User Interface **Maps**

- GUI maps

- Web-site maps

Graphical User Interface Maps

- **Access map** for the set of windows in a GUI showing which windows are accessible from which other windows and the set of ways to accomplish this access
- **Property lists** showing the objects in each window and their types

This is essential **navigation information**

Data Specs

- **Data grammars** - can describe any data structure, but best used for data with significant structural variation
- **Hierarchic data definitions** - used for data with few structural variations (they are really simple grammars)

Example of a Data Grammar

message	$\rightarrow$	header + body + trailer
header	$\rightarrow$	msgid + formcode
body	$\rightarrow$	packet_{1}^{7}
trailer	$\rightarrow$	packetcnt + hashtotal
msgid	$\rightarrow$	digit^{5}
digit	$\rightarrow$	1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 0
formcode	$\rightarrow$	BIN | INT | HEX
packet	$\rightarrow$	$\text{base} + \text{extension}_{0}^{3}$

Hierarchic Data **Definitions**

A description of a fixed data pattern
(i.e., a fixed parse tree)

1 Packed Message
- 2 Header
- 2 Body
  - 3 Packets (1 to 7)
- 2 Trailer

Traditional input to test data generators

What Test Designers wish5 from SW Models

- Wish 1 - Function Inventories with Semantics
- Wish 2 - Behavior Rules
- Wish 3 - Explicit Usage Patterns
- Wish 4 - UI Maps & Data Dictionaries
- **Wish 5 - Linkage between Model and Implementation**

Name Association

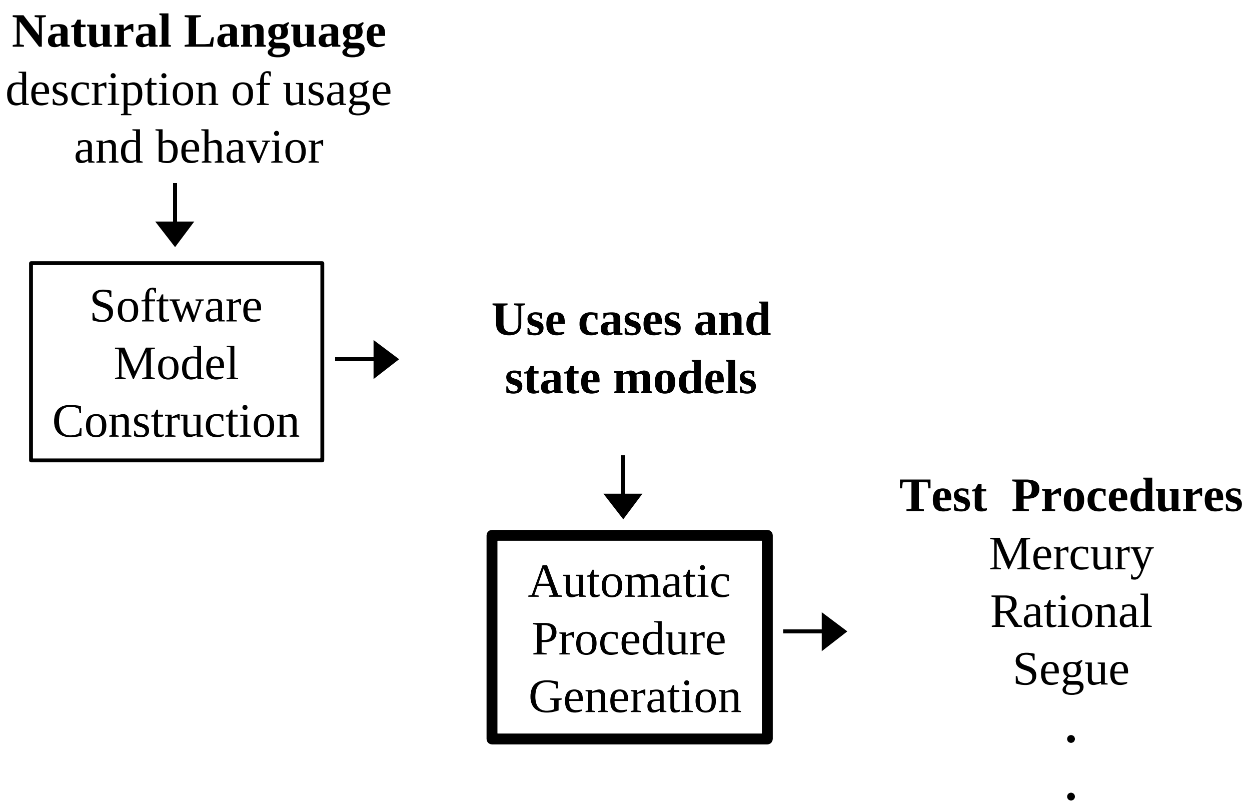
- Specification of relationship between **model** names and **implementation** names
- Reports in both directions
- Analyzers to report unrelated or ambiguous names

5 Wishes about SW Models

To Support Automatic Test Generation

- Wish 1 - Function Inventories with Semantics
- Wish 2 - Behavior Rules
- Wish 3 - Explicit Usage Patterns
- Wish 4 - UI Maps & Data Dictionaries
- Wish 5 - Linkage between Model and Implementation

Automatic Generation - Today



Automatic Generation - Tomorrow?

Natural Language description of usage, behavior, & structure

↓

Software Model Construction →

Semi-formal descriptions of usage, behavior, & structure

↓

Automatic Procedure Generation →

Functional Test Case & Procedure Language

↓

FTCPL Translators →

Test Procedures
Mercury
Rational
Segue
.
.

This part is Wishware

Style Sheet for Unlimited Entry Decision Tables

Limited Entries Considered Harmful
Use Unlimited Entries

Condition Columns (header & entries)

- Name object attributes or attribute relations (e.g., stock level vs reorder point) in the header. Object attributes can be named explicitly (e.g., Customer type is) or implicitly by listing mutually exclusive values of the implied attribute with the object (e.g., Customer is: Person, Non-Person).

- Place mutually exclusive values of attributes or relationships (e.g., Below or At) as disjunctive conditions that can be TRUE in the individual entries

- When possible, combine all conditions that result in the same set of actions into a single column using disjunctive entries

- Never use [1, 0], [T, F] or [Yes, No] in an entry. When the situation is binary, use explicit condition names (e.g. Switch is: Open, Closed)

- Every possible condition should appear in some entry -- perhaps implicitly in an empty entry (see below) or an ‘otherwise’ rule

Action Columns (header & entries)

- Name individual actions or groups of actions in the header in <verb, object> format and indicate inclusion with verb or name specific action modifiers in the individual entries

	e.g.,	Reorder stock	Reorder	Reorder	
or	e.g.,	Pack order	Full		Partial

- Group actions that are mutually exclusive i.e., at most one at a time

- Group actions that are completely independent i.e., can occur in any combination

- Prefer groups of actions

Decision Rules

- Use **precise** application terminology
- When the truth values of one or more condition entries implies the truth value for another (dependent) condition, that implied value should be specified and then **marked as dependent** in some way.
- Develop a **dictionary** to provide precise definitions for application objects, attributes, values, relationships, actions, and conditions as well as dependencies between sets of conditions.
- For a decision rule, if no condition associated with a specific header is relevant to the rule, then the entry for that header should be empty. This **empty entry** is interpreted as the disjunction of all possible conditions for that header. The empty entry may be represented by a blank cell or one containing a symbol for emptiness to signal that the value has not been overlooked, but analyzed to be empty.
- Each condition should correspond to an appropriate set of actions i.e., the rules should be **correct**
- Every possible situation should have an applicable rule i.e., the set of rules should be **complete**
- No two decision rules should apply to the same situation i.e., the set of rule conditions should be **mutually exclusive**
- When constructing or modifying a decision table, these properties should be **check**ed

A Construction Process for Unlimited Entry Decision Tables

1. Identify and name every action (e.g., display record) that might be included in the behavior model. Choose liberally.

2. Identify and name every object attribute (e.g., input validity) and its associated set of values (e.g., [valid, invalid]) that might determine (condition) performance of any of the actions. Choose liberally.

3. Identify specific dependencies between groups of one or more conditions.

4. Specify a first-cut set of decision rules

5. Add actions and preconditions as necessary to make each decision rule correct and fully descriptive.

6. Discard irrelevant actions and conditions.

7. Determine if two rules with identical action sets can be combined, either (1) by combining the unmatched values of a single attribute with an "or", (2) by expanding the value sets of an attribute or (3) by introducing alternative attributes (i.e., by thinking about the situation in a different way).

8. Review the final set of dependencies and rules for completeness and correctness with domain experts and modify as required.

Abstract of Pass Orders System

This system is a partially automated front-end to a set of financial trading systems. The system handles orders for securities (i.e., stock & bonds) as well as options (e.g., puts and calls). The system is fed by electronic and manual order sources and feeds Traders, Derivative **O**ptions **T**rading **(DOT) systems, and Head Traders. It interacts with humans who provide manual orders as well as fix, authorize, or assign the electronic or manual orders provided.**

The system:

(1) accepts either manual or electronic orders,

(2) assigns an order number

(3) checks each order,

(4) supports the fixing of improper (invalid or unauthorized) orders that can be fixed,

(5) supports the assignment of proper, but unassigned, securities orders to a trader,

(6) rejects improper and unfixable orders to Head Trader

(7) passes proper options orders to an appropriate DOT system, and

(8) passes proper, assigned securities orders to a Trader

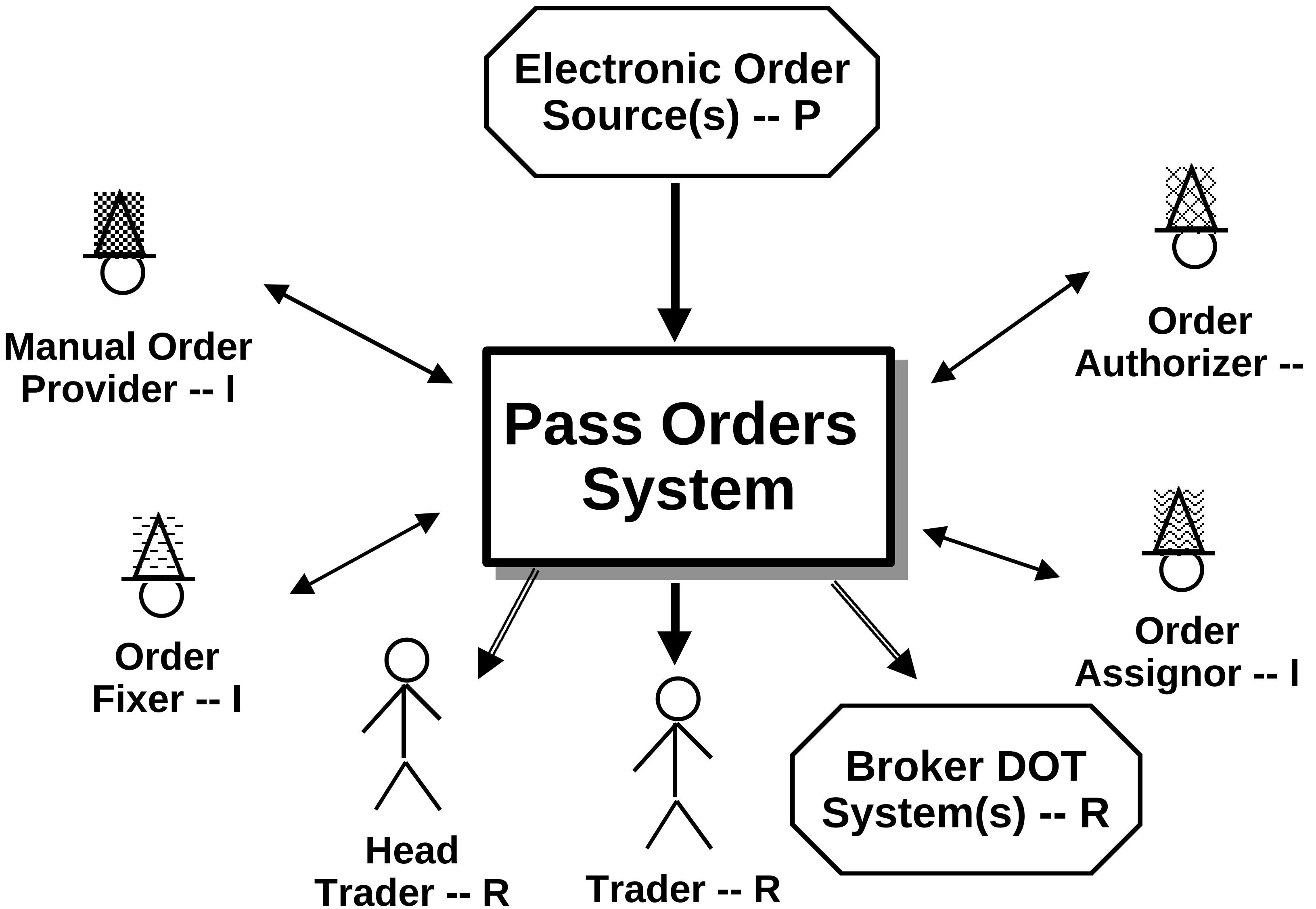
Terminology

Unassigned **order -- trader id field is empty**

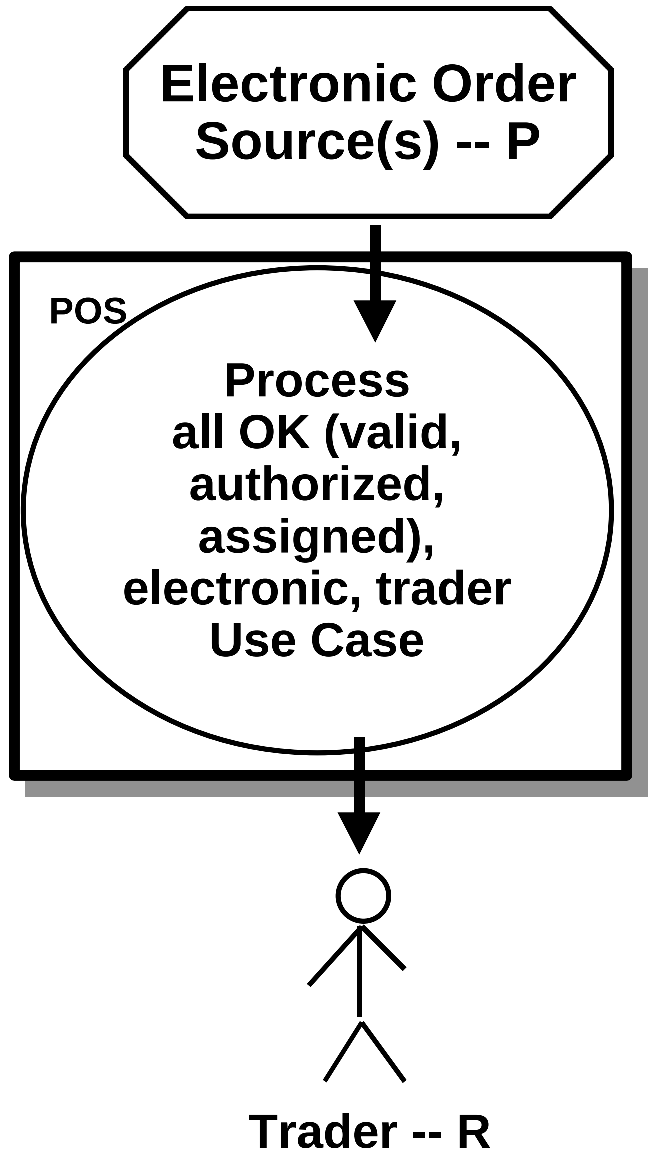
Unauthorized **order -- authorization field is empty**

Invalid **order -- order violates one or more validation criteria e.g., unrecognized security code, for data other than assignment and authorization information**

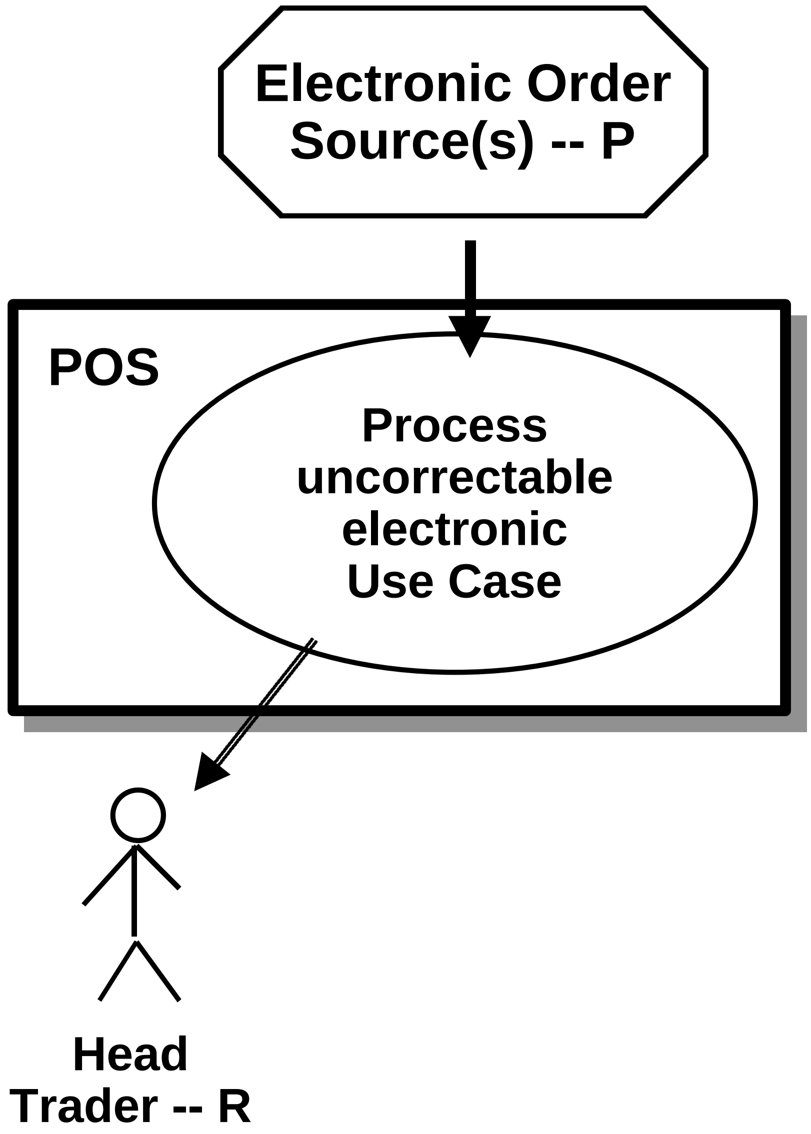
A User Population Diagram



Usage Diagram for a Simple Course



Usage Diagram for Process Uncorrectable



POS Post-conditions

Candidate Pre-conditions

0 < # of orders to be processed

0 < initial order number

POS is operational (e.g., all providing & interactive users are available)

Candidate Post-conditions

[# orders passed to Traders + # orders passed to DOT systems
+ # rejected orders] = [# automated orders + # manual orders]

final order number – initial order number + 1
= [# automated orders + # manual orders]

all passed orders are valid, authorized, and assigned

all rejected orders are unfixable within POS

all unfixable orders are rejected to a Head Trader

all orders passed to a DOT system are option orders

all orders passed to a Trader are security orders assigned to that Trader

structure of every passed order is valid

contents of every passed or rejected order
= [contents of incoming order + order number]

[(time stamp on last message processed – time stamp on first message processed)
/ # of messages processed] < acceptable throughput threshold

Candidate Invariants

all orders are either manual or electronic

Pass Orders System

Decision Table, Unlimited Style

Rule #	Order is: Valid Correctable Uncorrectable	Order is: Authorized OVerrideable Unoverrideable OTher	Order is: Assigned Unassigned	Order is: **DOT** **(D**erivative **O**ptions **T**rading) Trader	Receive order & assign order number	Pass OK orders to DOT, Trader	Support	Reject
1	Uncorrectable	---	---	---	**X**			**uncorrectable**
2	Valid or Correctable	Unoverrideable	---	---	**X**			**unoverrideable**
3	Correctable	OVerrideable	Unassigned	DOT	**X**	**DOT**	**correction overriding assignment**	
4	Correctable	OVerrideable	Unassigned	Trader	**X**	**Trader**	**correction overriding assignment**	
5	Correctable	OVerrideable	Assigned	DOT	**X**	**DOT**	**correction overriding**	
6	Correctable	Authorized or OTher	Unassigned	DOT	**X**	**DOT**	**correction assignment**	
7	Valid	OVerrideable	Unassigned	DOT	**X**	**DOT**	**overriding assignment**	
8	Correctable	OVerrideable	Assigned	Trader	**X**	**Trader**	**correction overriding**	

9	Correctable	Authorized or OTher	Unassigned	Trader		**X**	**Trader**	**correction assignment**	
10	Valid	OVerrideable	Unassigned	Trader		**X**	**Trader**	**overriding assignment**	
11	Correctable	Authorized or OTher	Assigned	DOT		**X**	**DOT**	**correction**	
12	Valid	OVerrideable	Assigned	DOT		**X**	**DOT**	**overriding**	
13	Valid	Authorized or OTher	Unassigned	DOT		**X**	**DOT**	**assignment**	
14	Correctable	Authorized or OTher	Assigned	Trader		**X**	**Trader**	**correction**	
15	Valid	OVerrideable	Assigned	Trader		**X**	**Trader**	**overriding**	
16	Valid	Authorized or OTher	Unassigned	Trader		**X**	**Trader**	**assignment**	
17	Valid	Authorized or OTher	Assigned	DOT		**X**	**DOT**		
18	Valid	Authorized or OTher	Assigned	Trader		**X**	**Trader**		

Dictionary of Objects, Conditions, & Actions

Pass Order System

<u>Objects & Attributes</u>

Order
- Order Number

<u>Objects & Conditions</u>

Orders are:

- **Valid –** *security code* and *order quantity* are both valid
- Invalid orders (not Valid) are:
  - **Correctable** – ???
  - **Uncorrectable –** not Correctable

Orders are:

- **Authorized** – *authorization code* is valid
- Unauthorized (not Authorized) orders are:
  - **Overrideable** – ???
  - **Unoverrideable** – not Overrideable
- **Other** – ???

Orders are:

- **Assigned –** *trader id* is valid
- **Unassigned –** not Assigned

Orders are:

- **DOT** – *security type* = derivative
- **Trader** – not DOT

<u>Actions</u>

Receive electronic or manual orders
Assign order number
Support order correction
Support authorization overriding
Support order assignment
Pass OK orders to traders or DOT system
Pass uncorrectable or unoverrideable orders to head trader

Pass Orders System

Effect Tables, Unlimited Style

Receive manual or electronic order	Assign unique order number
X	**X**

Rule ID	Order is: Valid Correctable Uncorrectable	Order is: Authorized OVerrideable Unoverrideable OTher	Reject
R1	Uncorrectable	-----	**uncorrectable**
R2	Valid or Correctable	Unoverrideable	**unoverrideable**

Rule ID	Order is: Valid Correctable Uncorrectable	Order is: Authorized OVerrideable Unoverrideable OTher	Order is: DOT Trader	Pass OK orders to DOT, Trader
P1	Valid or Correctable	Authorized Overrideable or OTher	DOT	**DOT**
P2	Valid or Correctable	Authorized Overrideable or OTher	Trader	**Trader**

Rule ID	Order is: Valid Correctable Uncorrectable	Order is: Authorized OVerrideable Unoverrideable OTher	Order is: Assigned Unassigned	Support
S1	Correctable	Authorized Overrideable or OTher	-----	**correction**
S2	Valid or Correctable	Overrideable	-----	**overriding**
S3	Valid or Correctable	Authorized Overrideable or OTher	Unassigned	**assignment**

Pass Orders System

Extended Action Tables, Unlimited Style

Receive manual or electronic order	Assign unique order number	Post-Conditions
X	**X**	**Order received & Order number assigned**

Rule ID	Order is: Valid Correctable Uncorrectable	Order is: Authorized OVerrideable Unoverrideable OTher	Reject	Post-Conditions
R1	Uncorrectable	-----	**uncorrectable**	**Order rejected as uncorrectable**
R2	Valid or Correctable	Unoverrideable	**unoverrideable**	**Order rejected as unoverrideable**

Rule ID	Order is: Valid Correctable Uncorrectable	Order is: Authorized OVerrideable Unoverrideable OTher	Order is: DOT Trader	Pass OK orders to DOT, Trader	Post-Conditions
P1	Valid or Correctable	Authorized Overrideable or OTher	DOT	**DOT**	**Order is OK & Order sent to DOT**
P2	Valid or Correctable	Authorized Overrideable or OTher	Trader	**Trader**	**Order is OK & Order sent to Trader**

Rule ID	Order is: Valid Correctable Uncorrectable	Order is: Authorized OVerrideable Unoverrideable OTher	Order is: Assigned Unassigned		Support	Post-Conditions
S1	Correctable	Authorized Overrideable or OTher	-----		**correction**	**Order is Valid**
S2	Valid or Correctable	Overrideable	-----		**overriding**	**Order is Authorized**
S3	Valid or Correctable	Authorized Overrideable or OTher	Unassigned		**assignment**	**Order is Assigned**