Unified Process – The Elaboration Phase

Marek Rychlý
rychly@fit.vutbr.cz

Brno University of Technology
Faculty of Information Technology
Department of Information Systems

Information Systems Analysis and Design (AIS)
31 October 2019

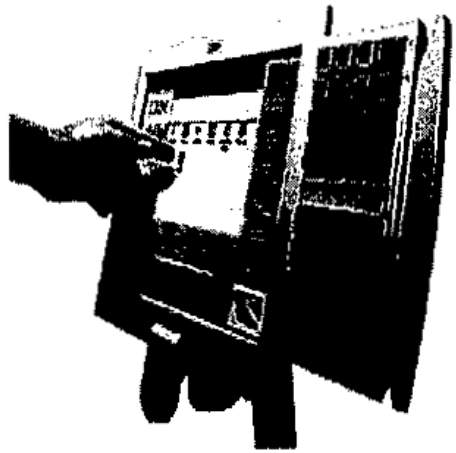
Outline

NextGen POS: Informal Specification (1)

Case One: The NextGen POS System

The first case study is the NextGen point-of-sale (POS) system. In this apparently straightforward problem domain, we shall see that there are interesting requirement and design problems to solve. In addition, it's a real problem—groups really do develop POS systems with object technologies.

A POS system is a computerized application used (in part) to record sales and handle payments; it is typically used in a retail store. It includes hardware components such as a computer and bar code scanner, and software to run the system. It interfaces to various service applications, such as a third-party tax calculator and inventory control. These systems must be relatively fault-tolerant; that is, even if remote services are temporarily unavailable (such as the inventory system), they must still be capable of capturing sales and handling at least cash payments (so that the business is not crippled).



A POS system increasingly must support multiple and varied client-side terminals and interfaces. These include a thin-client Web browser terminal, a regular

(adopted from "Applying UML and Patterns" by Craig Larman)

NextGen POS: Informal Specification (2)

personal computer with something like a Java Swing graphical user interface, touch screen input, wireless PDAs, and so forth.

Furthermore, we are creating a commercial POS system that we will sell to different clients with disparate needs in terms of business rule processing. Each client will desire a unique set of logic to execute at certain predictable points in scenarios of using the system, such as when a new sale is initiated or when a new line item is added. Therefore, we will need a mechanism to provide this flexibility and customization.

Using an iterative development strategy, we are going to proceed through requirements, object-oriented analysis, design, and implementation.

(adopted from “Applying UML and Patterns” by Craig Larman)

NextGen POS: Use Case Model (1)

Use Case UC1: Process Sale

Scope: NextGen POS application
Level: user goal
Primary Actor: Cashier
Stakeholders and Interests:
– Cashier: Wants accurate, fast entry, and no payment errors, as cash drawer shortages are deducted from his/her salary.
– Salesperson: Wants sales commissions updated.
– Customer: Wants purchase and fast service with minimal effort. Wants easily visible display of entered items and prices. Wants proof of purchase to support returns.
– Company: Wants to accurately record transactions and satisfy customer interests. Wants to ensure that Payment Authorization Service payment receivables are recorded. Wants some fault tolerance to allow sales capture even if server components (e.g., remote credit validation) are unavailable. Wants automatic and fast update of accounting and inventory.
– Manager: Wants to be able to quickly perform override operations, and easily debug Cashier problems.
– Government Tax Agencies: Want to collect tax from every sale. May be multiple agencies, such as national, state, and county.
– Payment Authorization Service: Wants to receive digital authorization requests in the correct format and protocol. Wants to accurately account for their payables to the store.
Preconditions: Cashier is identified and authenticated.
Success Guarantee (or Postconditions): Sale is saved. Tax is correctly calculated. Accounting and Inventory are updated. Commissions recorded. Receipt is generated. Payment authorization approvals are recorded.

(adopted from “Applying UML and Patterns” by Craig Larman)

NextGen POS: Use Case Model (2)

Main Success Scenario (or Basic Flow):

1. Customer arrives at POS checkout with goods and/or services to purchase.
2. Cashier starts a new sale.
3. Cashier enters item identifier.
4. System records sale line item and presents item description, price, and running total. Price calculated from a set of price rules.

Cashier repeats steps 3-4 until indicates done.

5. System presents total with taxes calculated.
6. Cashier tells Customer the total, and asks for payment.
7. Customer pays and System handles payment.
8. System logs completed sale and sends sale and payment information to the external Accounting system (for accounting and commissions) and Inventory system (to update inventory).
9. System presents receipt.
10. Customer leaves with receipt and goods (if any).

Extensions (or Alternative Flows):

*a. At any time, Manager requests an override operation:
   1. System enters Manager-authorized mode.
   2. Manager or Cashier performs one Manager-mode operation. e.g., cash balance change, resume a suspended sale on another register, void a sale, etc.
   3. System reverts to Cashier-authorized mode.

(adopted from “Applying UML and Patterns” by Craig Larman)

NextGen POS: Use Case Model (3)

*b. At any time, System fails:
To support recovery and correct accounting, ensure all transaction sensitive state and events can be recovered from any step of the scenario.
1. Cashier restarts System, logs in, and requests recovery of prior state.
2. System reconstructs prior state.
2a. System detects anomalies preventing recovery:
1. System signals error to the Cashier, records the error, and enters a clean state.
2. Cashier starts a new sale.
1a. Customer or Manager indicate to resume a suspended sale.
1. Cashier performs resume operation, and enters the ID to retrieve the sale.
2. System displays the state of the resumed sale, with subtotal.
2a. Sale not found.
1. System signals error to the Cashier.
2. Cashier probably starts new sale and re-enters all items.
3. Cashier continues with sale (probably entering more items or handling payment).
2-4a. Customer tells Cashier they have a tax-exempt status (e.g., seniors, native peoples)
1. Cashier verifies, and then enters tax-exempt status code.
2. System records status (which it will use during tax calculations)
3a. Invalid item ID (not found in system):
1. System signals error and rejects entry.
2. Cashier responds to the error:
2a. There is a human-readable item ID (e.g., a numeric UPC):
1. Cashier manually enters the item ID.
2. System displays description and price.
2a. Invalid item ID: System signals error. Cashier tries alternate method.
2b. There is no item ID, but there is a price on the tag:
1. Cashier asks Manager to perform an override operation.

(adopted from “Applying UML and Patterns” by Craig Larman)

NextGen POS: Use Case Model (4)

7a. Paying by cash:
1. Cashier enters the cash amount tendered.
2. System presents the balance due, and releases the cash drawer.
3. Cashier deposits cash tendered and returns balance in cash to Customer.
4. System records the cash payment.

7b. Paying by credit:
1. Customer enters their credit account information.
2. System displays their payment for verification.
3. Cashier confirms.
   3a. Cashier cancels payment step:
      1. System reverts to "item entry" mode.
4. System sends payment authorization request to an external Payment Authorization Service System, and requests payment approval.
   4a. System detects failure to collaborate with external system:
      1. System signals error to Cashier.
      2. Cashier asks Customer for alternate payment.
5. System receives payment approval, signals approval to Cashier, and releases cash drawer (to insert signed credit payment receipt).
   5a. System receives payment denial:
      1. System signals denial to Cashier.
      2. Cashier asks Customer for alternate payment.
   5b. Timeout waiting for response.
      1. System signals timeout to Cashier.
      2. Cashier may try again, or ask Customer for alternate payment.

(adopted from "Applying UML and Patterns" by Craig Larman)

NextGen POS: Use Case Model (5)

6. System records the credit payment, which includes the payment approval.
7. System presents credit payment signature input mechanism.
8. Cashier asks Customer for a credit payment signature. Customer enters signature.
9. If signature on paper receipt, Cashier places receipt in cash drawer and closes it.

7c. Paying by check…
7d. Paying by debit…
7e. Cashier cancels payment step:
1. System reverts to "item entry" mode.

7f. Customer presents coupons:
1. Before handling payment, Cashier records each coupon and System reduces price as appropriate. System records the used coupons for accounting reasons.
1a. Coupon entered is not for any purchased item:
1. System signals error to Cashier.

9a. There are product rebates:
1. System presents the rebate forms and rebate receipts for each item with a rebate.

9b. Customer requests gift receipt (no prices visible):
1. Cashier requests gift receipt and System presents it.

9c. Printer out of paper.
1. If System can detect the fault, will signal the problem.
2. Cashier replaces paper.
3. Cashier requests another receipt.

(adopted from "Applying UML and Patterns" by Craig Larman)

NextGen POS: Use Case Model (6)

Special Requirements:
– Touch screen UI on a large flat panel monitor. Text must be visible from 1 meter.
– Credit authorization response within 30 seconds 90% of the time.
– Somehow, we want robust recovery when access to remote services such the inventory system is failing.
– Language internationalization on the text displayed.
– Pluggable business rules to be insertable at steps 3 and 7.
– . . .

Technology and Data Variations List:
*a. Manager override entered by swiping an override card through a card reader, or entering an authorization code via the keyboard.
3a. Item identifier entered by bar code laser scanner (if bar code is present) or keyboard.
3b. Item identifier may be any UPC, EAN, JAN, or SKU coding scheme.
7a. Credit account information entered by card reader or keyboard.
7b. Credit payment signature captured on paper receipt. But within two years, we predict many customers will want digital signature capture.

Frequency of Occurrence: Could be nearly continuous.

Open Issues:
– What are the tax law variations?
– Explore the remote service recovery issue.
– What customization is needed for different businesses?
– Must a cashier take their cash drawer when they log out?
– Can the customer directly use the card reader, or does the cashier have to do it?

However, UPCs and EANs have a standards and regulatory component. See www.adams1.com/pub/russadam/upccode.html for a good overview. Also see www.uc-council.org and www.ean-int.org.

(adopted from “Applying UML and Patterns” by Craig Larman)

NextGen POS: Use Case Model in Two Columns

Use Case UC1: Process Sale

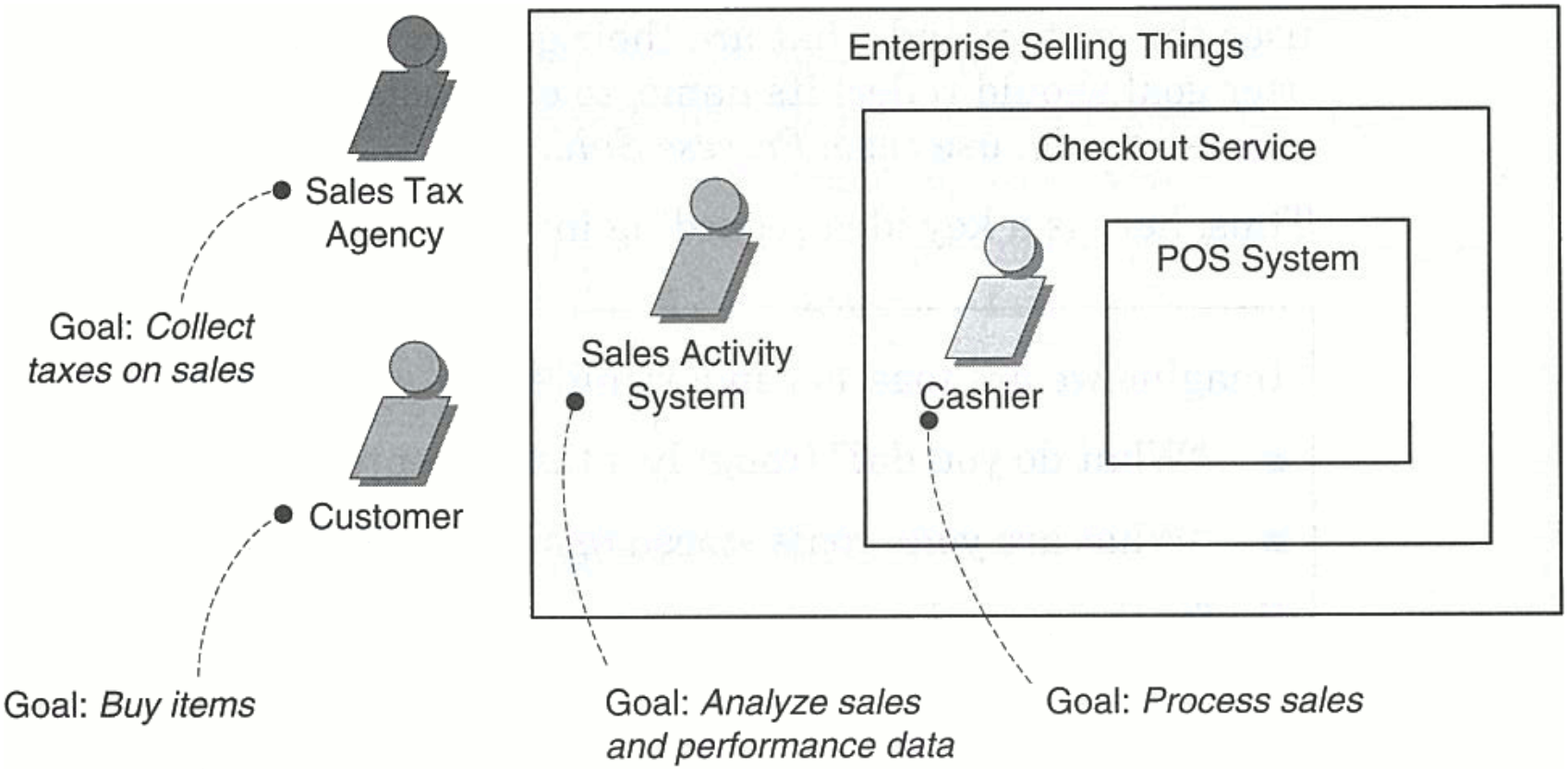
Primary Actor: ...
... as before ...

Main Success Scenario:

Actor Action (or Intention)	System Responsibility
1. Customer arrives at a POS checkout with goods and/or services to purchase.	
2. Cashier starts a new sale.	
3. Cashier enters item identifier.	
	4. Records each sale line item and presents item description and running total.
Cashier repeats steps 3-4 until indicates done.	5. Presents total with taxes calculated.
6. Cashier tells Customer the total, and asks for payment.	
7. Customer pays.	
	8. Handles payment.
	9. Logs the completed sale and sends information to the external accounting (for all accounting and commissions) and inventory systems (to update inventory). System presents receipt.
...	...

(adopted from “Applying UML and Patterns” by Craig Larman)

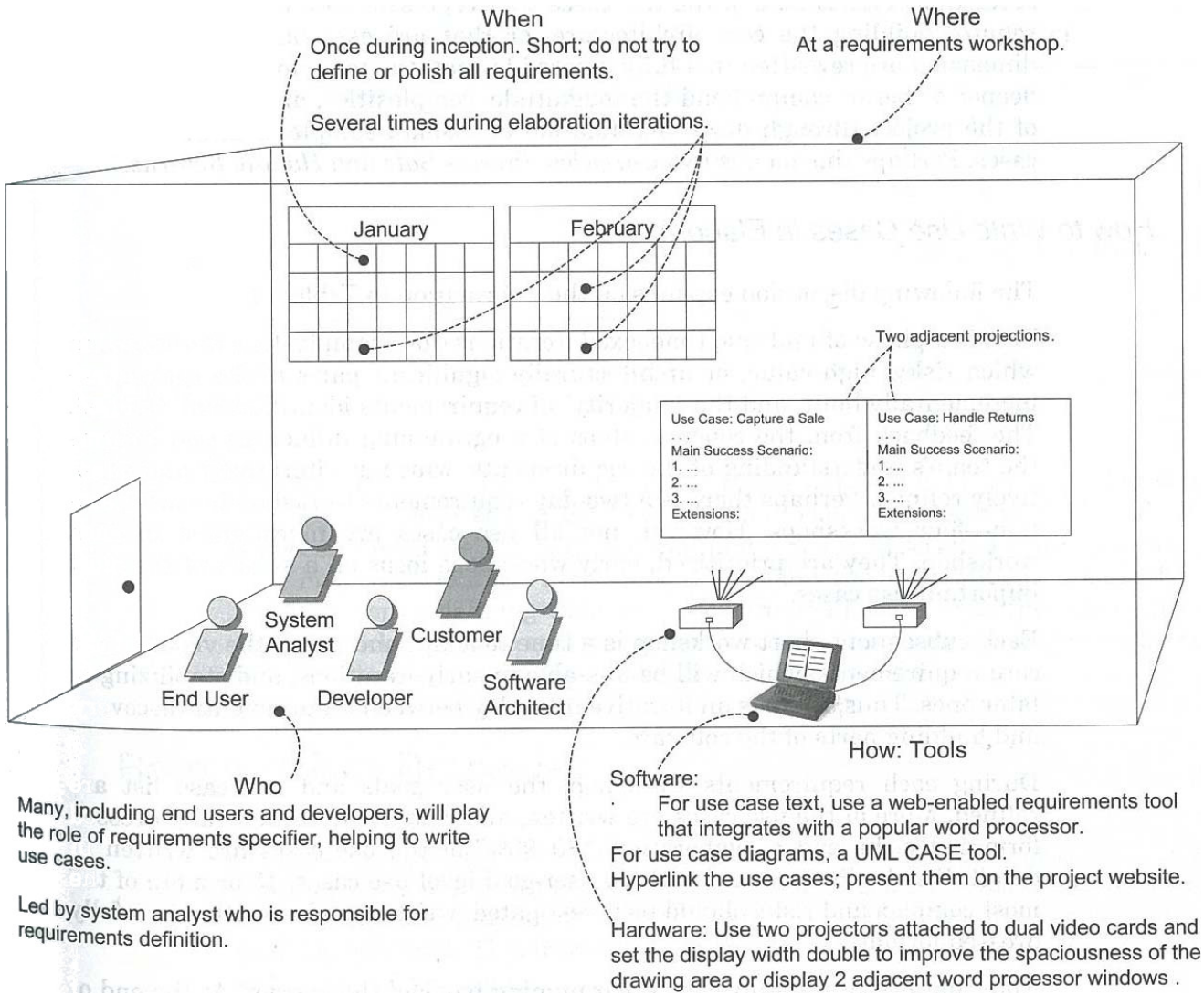
NextGen POS: The Scope of the System



(adopted from “Applying UML and Patterns” by Craig Larman)

Who is the primary actor – Customer or Cashier, and why?

NextGen POS: Requirements Workshops



(adopted from “Applying UML and Patterns” by Craig Larman)

NextGen POS: Supplementary Specification (1)

Supplementary Specification

Revision History

Version	Date	Description	Author
Inception draft	Jan 10, 2031	First draft. To be refined primarily during elaboration.	Craig Larman

Introduction

This document is the repository of all NextGen POS requirements not captured in the use cases.

Functionality

(Functionality common across many use cases)

Logging and Error Handling

Log all errors to persistent storage.

Pluggable Rules

At various scenario points of several use cases (to be defined) support the ability to customize the functionality of the system with a set of arbitrary rules that execute at that point or event.

Security

All usage requires user authentication.

(adopted from “Applying UML and Patterns” by Craig Larman)

NextGen POS: Supplementary Specification (2)

Usability

Human Factors

The customer will be able to see a large-monitor display of the POS. Therefore:

- Text should be easily visible from 1 meter.
- Avoid colors associated with common forms of color blindness.

Speed, ease, and error-free processing are paramount in sales processing, as the buyer wishes to leave quickly, or they perceive the purchasing experience (and seller) as less positive.

The cashier is often looking at the customer or items, not the computer display. Therefore, signals and warnings should be conveyed with sound rather than only via graphics.

Reliability

Recoverability

If there is failure to use external services (payment authorizer, accounting system, ...) try to solve with a local solution (e.g., store and forward) in order to still complete a sale. Much more analysis is needed here...

Performance

As mentioned under human factors, buyers want to complete sales processing *very* quickly. One bottleneck is external payment authorization. Our goal: authorization in less than1 minute, 90% of the time.

(adopted from “Applying UML and Patterns” by Craig Larman)

NextGen POS: Supplementary Specification (3)

Supportability

Adaptability

Different customers of the NextGen POS have unique business rule and processing needs while processing a sale. Therefore, at several defined points in the scenario (for example, when a new sale is initiated, when a new line item is added) pluggable business rule will be enabled.

Configurability

Different customers desire varying network configurations for their POS systems, such as thick versus thin clients, two-tier versus N-tier physical layers, and so forth. In addition, they desire the ability to modify these configurations, to reflect their changing business and performance needs. Therefore, the system will be somewhat configurable to reflect these needs. Much more analysis is needed in this area to discover the areas and degree of flexibility, and the effort to achieve it.

Implementation Constraints

NextGen leadership insists on a Java technologies solution, predicting this will improve long-term porting and supportability, in addition to ease of development.

Purchased Components

- Tax calculator. Must support pluggable calculators for different countries.

(adopted from “Applying UML and Patterns” by Craig Larman)

NextGen POS: Supplementary Specification (4)

Free Open Source Components

In general, we recommend maximizing the use of free Java technology open source components on this project.

Although it is premature to definitively design and choose components, we suggest the following as likely candidates:

- JLog logging framework
- ...

Interfaces

Noteworthy Hardware and Interfaces

- Touch screen monitor (this is perceived by operating systems as a regular monitor, and the touch gestures as mouse events)
- Barcode laser scanner (these normally attach to a special keyboard, and the scanned input is perceived in software as keystrokes)
- Receipt printer
- Credit/debit card reader
- Signature reader (but not in release 1)

Software Interfaces

For most external collaborating systems (tax calculator, accounting, inventory, ...) we need to be able to plug in varying systems and thus varying interfaces.

(adopted from “Applying UML and Patterns” by Craig Larman)

NextGen POS: Supplementary Specification (5)

Application-Specific Domain (Business) Rules

(See the separate Business Rules document for general rules.)

ID	Rule	Changeability	Source
RULE1	Purchaser discount rules. Examples: Employee—20% off. Preferred Customer—10% off. Senior—15% off.	High. Each retailer uses different rules.	Retailer policy.
RULE2	Sale (transaction-level) discount rules. Applies to pre-tax total. Examples: 10% off if total greater than $100 USD. 5% off each Monday. 10% off all sales from 10am to 3pm today. Tofu 50% off from 9am-10am today.	High. Each retailer uses different rules, and they may change daily or hourly.	Retailer policy.
RULE3	Product (line item level) discount rules. Examples: 10% off tractors this week. Buy 2 veggieburgers, get 1 free.	High. Each retailer uses different rules, and they may change daily or hourly.	Retailer policy.

Legal Issues

We recommend some open source components if their licensing restrictions can be resolved to allow resale of products that include open source software.

All tax rules must, by law, be applied during sales. Note that these can change frequently.

(adopted from “Applying UML and Patterns” by Craig Larman)

NextGen POS: Supplementary Specification (6)

Information in Domains of Interest

Pricing

In addition to the pricing rules described in the domain rules section, note that products have an *original price*, and optionally a *permanent markdown price*. A product's price (before further discounts) is the permanent markdown price, if present. Organizations maintain the original price even if there is a permanent markdown price, for accounting and tax reasons.

Credit and Debit Payment Handling

When an electronic credit or debit payment is approved by a payment authorization service, they are responsible for paying the seller, not the buyer. Consequently, for each payment, the seller needs to record monies owing in their accounts receivable, from the authorization service. Usually on a nightly basis, the authorization service will perform an electronic funds transfer to the seller's account for the daily total owing, less a (small) per transaction fee that the service charges.

Sales Tax

Sales tax calculations can be very complex, and regularly change in response to legislation at all levels of government. Therefore, delegating tax calculations to third-party calculator software (of which there are several available) is advisable. Tax may be owing to city, region, state, and national bodies. Some items may be tax exempt without qualification, or exempt depending on the buyer or target recipient (for example, a farmer or a child).

Item Identifiers: UPCs, EANs, SKUs, Bar Codes, and Bar Code Readers

The NextGen POS needs to support various item identifier schemes. UPCs (Universal Product Codes), EANs (European Article Numbering) and SKUs (Stock Keeping Units) are three common identifier systems for products that are sold. Japanese Article Numbers (JANs) are a kind of EAN version.

SKUs are completely arbitrary identifiers defined by the retailer.

(adopted from "Applying UML and Patterns" by Craig Larman)

NextGen POS: Vision (1)

7.6 NextGen Example: (Partial) Vision

Vision

Revision History

Version	Date	Description	Author
inception draft	Jan 10, 2031	First draft. To be refined primarily during elaboration.	Craig Larman

The analysis in this example is illustrative, but fictitious.

Introduction

We envision a next generation fault-tolerant point-of-sale (POS) application, NextGen POS, with the flexibility to support varying customer business rules, multiple terminal and user interface mechanisms, and integration with multiple third-party supporting systems.

Positioning

Business Opportunity

Existing POS products are not adaptable to the customer's business, in terms of varying business rules and varying network designs (for example, thin client or not; 2, 3, or 4-tier architectures). In addition, they do not scale well as terminals and business increase. And, none can work in either on-line or off-line mode, dynamically adapting depending on failures. None easily integrate with many third-party systems. None allow for new terminal technologies such as mobile PDAs. There is marketplace dissatisfaction with this inflexible state of affairs, and demand for a POS that rectifies this.

Problem Statement

Traditional POS systems are inflexible, fault intolerant, and difficult to integrate with third-party systems. This leads to problems in timely sales processing, instituting improved processes that don't match the software, and accurate and timely accounting and inventory data to support measurement and planning, among other concerns. This affects cashiers, store managers, system administrators, and corporate management.

Product Position Statement

—Terse summary of who the system is for, its outstanding features, and what differentiates it from the competition.

Alternatives and Competition...

Understand who the players are, and their problems.

Stakeholder Descriptions

Market Demographics...

Stakeholder (Non-User) Summary...

User Summary...

(adopted from "Applying UML and Patterns" by Craig Larman)

NextGen POS: Vision (2)

Consolidate input from the Actor and Goals List, and the Stakeholder Interests section of the use cases.

Key High-Level Goals and Problems of the Stakeholders

A one-day requirements workshop with subject matter experts and other stakeholders, and surveys a several retail outlets led to identification of the following key goals and problems:

High-Level Goal	Priority	Problems and Concerns	Current Solutions
Fast, robust, integrated sales processing	high	Reduced speed as load increases. Loss of sales processing capability if components fail. Lack of up-to-date and accurate information from accounting and other systems due to non-integration with existing accounting, inventory, and HR systems. Leads to difficulties in measuring and planning. Inability to customize business rules to unique business requirements. Difficulty in adding new terminal or user interface types (for example, mobile PDAs).	Existing POS products provide basic sales processing, but do not address these problems.
...	...	...	...

This may be the Actor-Goal List created during use-case modeling, or a more terse summary.

User-Level Goals

The users (and external systems) need a system to fulfill these goals:

- *Cashier*: process sales, handle returns, cash in, cash out
- *System administrator*: manage users, manage security, manage system tables
- *Manager*: start up, shut down
- *Sales activity system*: analyze sales data

(adopted from “Applying UML and Patterns” by Craig Larman)

NextGen POS: Vision (3)

User Environment...

Product Overview

Product Perspective

The NextGen POS will usually reside in stores; if mobile terminals are used, they will be in close proximit to the store network, either inside or close outside. It will provide services to users, and collaborate wit other systems, as indicated in Figure Vision-1.

Summarized from the use case diagram.

Context diagrams come in different formats with varying detail, but all show the major external actors related to a system.

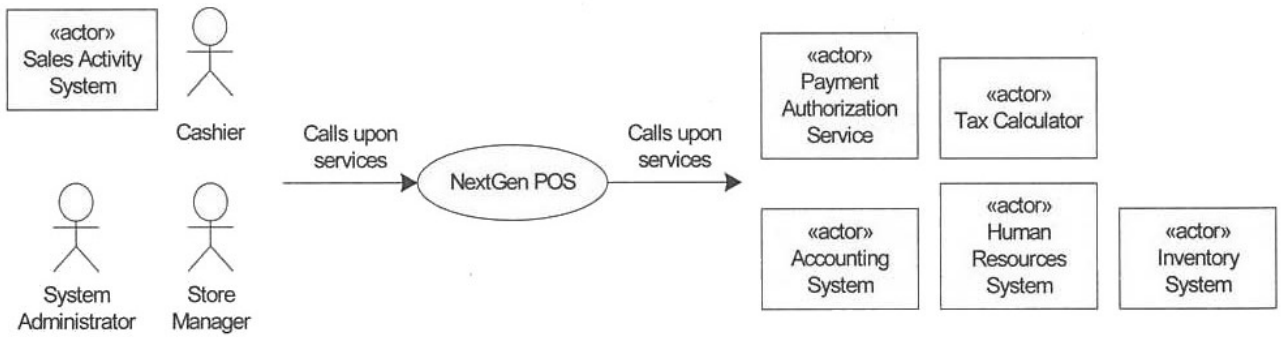


Figure Vision-1. NextGen POS system context diagram

(adopted from “Applying UML and Patterns” by Craig Larman)

NextGen POS: Vision (4)

Similar to the Actor-Goal list, this table relates goals, benefits, and solutions, but at a higher level not solely related to use cases.

It summarizes the value and differentiating qualities of the product.

Summary of Benefits

Supporting Feature	Stakeholder Benefit
Functionally, the system will provide all the common services a sales organization requires, including sales capture, payment authorization, return handling, and so forth.	Automated, fast point-of-sale services.
Automatic detection of failures, switching to local offline processing for unavailable services.	Continued sales processing when external components fail.
Pluggable business rules at various scenario points during sales processing.	Flexible business logic configuration.
Real-time transactions with third-party systems, using industry standard protocols.	Timely, accurate sales, accounting, and inventory information, to support measuring and planning.
...	...

Assumptions and Dependencies...
Cost and Pricing...
Licensing and Installation...

As discussed below, system features are a terse format to summarize functionality.

Summary of System Features

- sales capture
- payment authorization (credit, debit, check)
- system administration for users, security, code and constants tables, and so forth.
- automatic offline sales processing when external components fail
- real-time transactions, based on industry standards, with third-party systems, including inventory, accounting, human resources, tax calculators, and payment authorization services
- definition and execution of customized “pluggable” business rules at fixed, common points in the processing scenarios
- ...

Other Requirements and Constraints

Including design constraints, usability, reliability, performance, supportability, design constraints, documentation, packaging, and so forth: See the Supplementary Specification and use cases.

(adopted from “Applying UML and Patterns” by Craig Larman)

NextGen POS: Glossary

NextGen Example: A (Partial) Glossary

Glossary

Revision History

Version	Date	Description	Author
Inception draft	Jan 10, 2031	First draft. To be refined primarily during elaboration.	Craig Larman

Definitions

Term	Definition and Information	Format	Validation Rules	Aliases
item	A product or service for sale			
payment authorization	Validation by an external payment authorization service that they will make or guarantee the payment to the seller.			
payment authorization request	A composite of elements electronically sent to an authorization service, usually as a char array. Elements include: store ID, customer account number, amount, and timestamp.			
UPC	Numeric code that identifies a product. Usually symbolized with a bar code placed on products. See www.uc-council.org for details of format and validation.	12-digit code of several subparts.	Digit 12 i s a check digit.	Universal Product Code
...	...			

(adopted from “Applying UML and Patterns” by Craig Larman)

NextGen POS: Domain (Business) Rules (1)

NextGen Example: Business Rules (Domain Rules)

Domain Rules

Revision History

Version	Date	Description	Author
inception draft	Jan 10, 2031	First draft. To be refined primarily during elaboration.	Craig Larman

(adopted from “Applying UML and Patterns” by Craig Larman)

NextGen POS: Domain (Business) Rules (2)

Rule List

(See also the separate Application-specific Rules in the Supplementary Specification.)

ID	Rule	Changeability	Source
RULE1	Signature required for credit payments.	Buyer "signature" will continue to be required, but within 2 years most of our customers want signature capture on a digital capture device, and within 5 years we expect there to be demand for support of the new unique digital code "signature" now supported by USA law.	The policy of virtually all credit authorization companies.
RULE2	Tax rules. Sales require added taxes. See government statutes for current details.	High. Tax laws change annually, at all government levels.	law
RULE3	Credit payment reversals may only be paid as a credit to the buyer's credit account, not as cash.	Low	credit authorization company policy

(adopted from "Applying UML and Patterns" by Craig Larman)

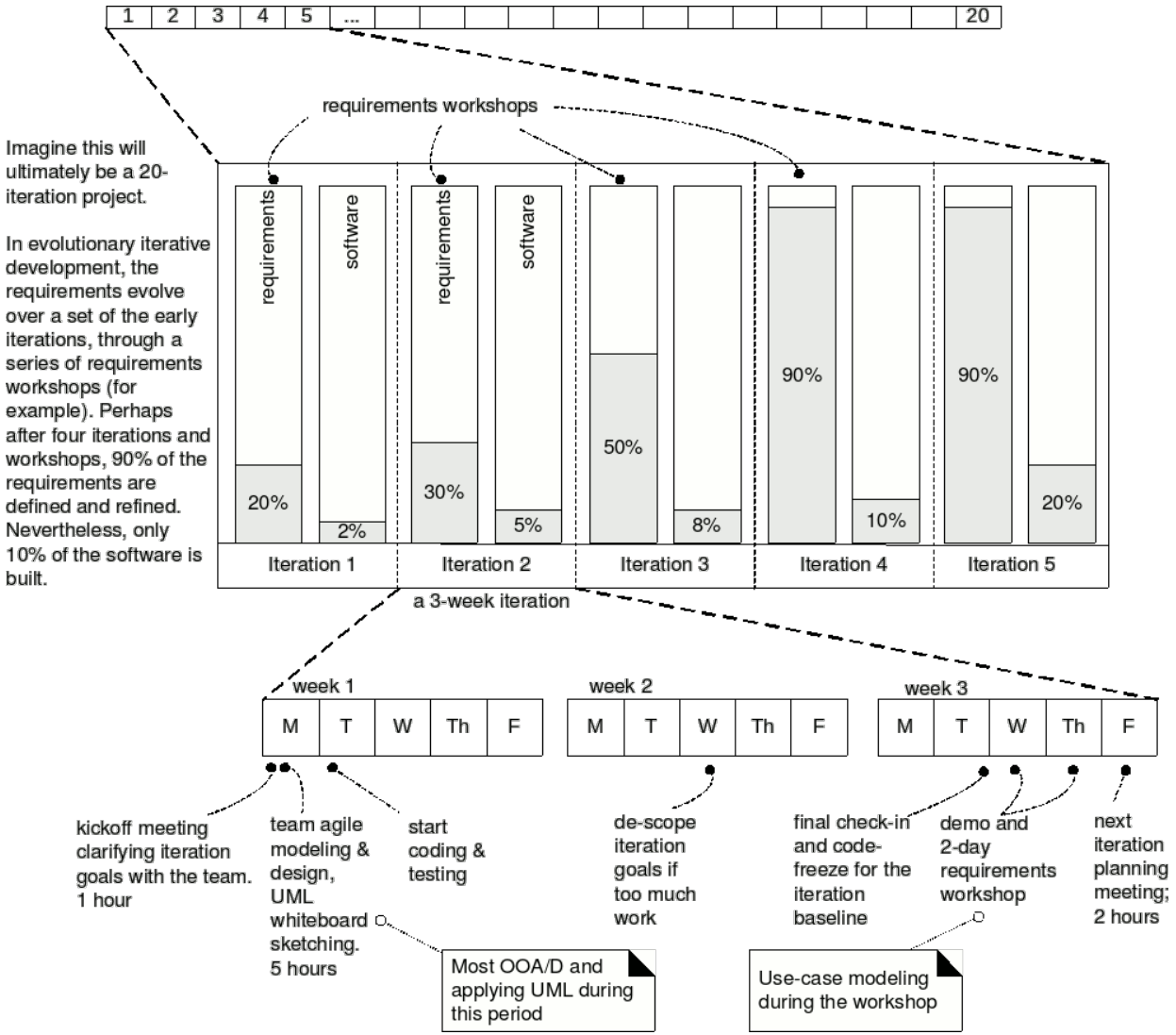
Activities and Artefacts in the Inception Phase (1)

- There was a requirements workshop (short).
  (to identify the most of actors, their goals and use cases)
- The most of use cases was briefly described.
  (critical 10 to 20% of the requirements are described in more details)
- The most of critical non-functional requirements was identified.
  (those that are risky or important for the system architecture)
- There are initial Vision and Supplement specification documents.
  (to describe a business case and information out of scope of the use case models)
- Risks were identified.

Activities and Artefacts in the Inception Phase (2)

- Thrown-away prototypes were created to support the feasibility study.
  (e.g., to demonstrate how the system will be controlled via touch-screens)
- UI prototypes were created for clarification of the requirements.
- Decisions to develop, buy, or reuse system components were made.
  (e.g., a component of tax-calculator)
- System architecture and core components were designed.
  (e.g., as two-layers client-server architecture with Oracle database server)
- There is a first-iteration plan and a set-up design/development env.
  (a list of tools for the design and development)

An Example of the Elaboration Phase



(adopted from “Applying UML and Patterns” by Craig Larman)

Goals of the Elaboration Phase

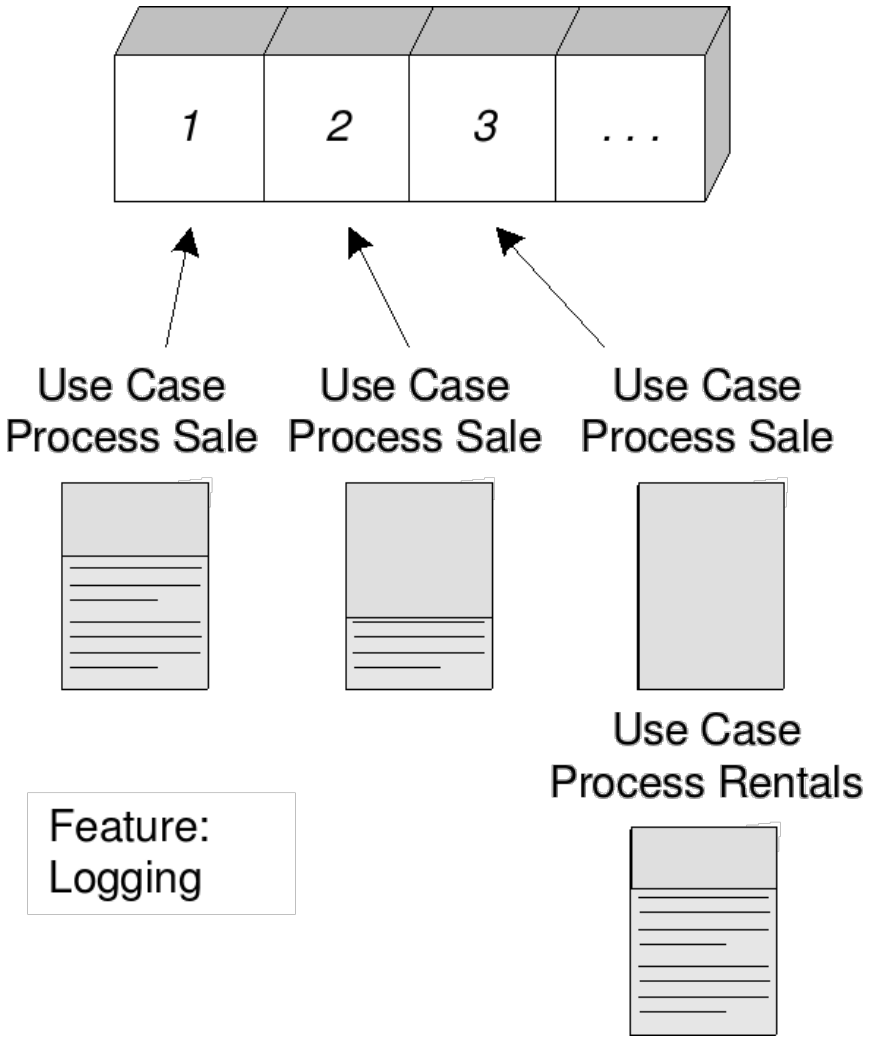
- To develop&test core and risky parts of the system architecture.
  (there will be an executable architectural baseline[1] for further development)
- To identify, describe and stabilise the most of requirements.
  (80% of the functional requirements need to be captured in this phase)
- To minimise or to eliminate the identified unacceptable risks[2].
  (i.e., to refine and to finish a risk assessment and a risk mitigation)
- To estimate resources, time, equipment, staff, and cost.
  (to redo plans and estimates created during the inception phase)

[1] An “architectural prototype”, however, not in terms of the prototyping.
[2] The risks are evaluated from the point of view of business value.

Concepts and Best-practices of the Elaboration Phase

- The iterations should be short and time-boxed.
- Development/coding should start as soon as possible.
- Core and risky architectural parts should be designed, implemented, and tested in this phase.
- Testing should be early, frequent, and realistic.
- There should be a feedback from testers, users, and developers.
  (and this feedback should result into corresponding corrections/enhancements)
- The most of requirements should be addressed during this phase, one or several per each iteration.
  (a complex use case may take several iterations, each of those iterations can address particular user stories/scenarios related to the use case)

The Progress of Implementation of Use Cases



A use case or feature is often too complex to complete in one short iteration.

Therefore, different parts or scenarios must be allocated to different iterations.

(adopted from “Applying UML and Patterns” by Craig Larman)

Artefacts of the Elaboration Phase

(without those of the inception phase)

Artifact	Comment
Domain Model	This is a visualization of the domain concepts; it is similar to a static information model of the domain entities.
Design Model	This is the set of diagrams that describes the logical design. This includes software class diagrams, object interaction diagrams, package diagrams, and so forth.
Software Architecture Document	A learning aid that summarizes the key architectural issues and their resolution in the design. It is a summary of the outstanding design ideas and their motivation in the system.
Data Model	This includes the database schemas, and the mapping strategies between object and non-object representations.
Use-Case Storyboards, UI Prototypes	A description of the user interface, paths of navigation, usability models, and so forth.

(adopted from “Applying UML and Patterns” by Craig Larman)

The First Iteration of the Elaboration Phase

(an example of requirements implemented in this iteration)

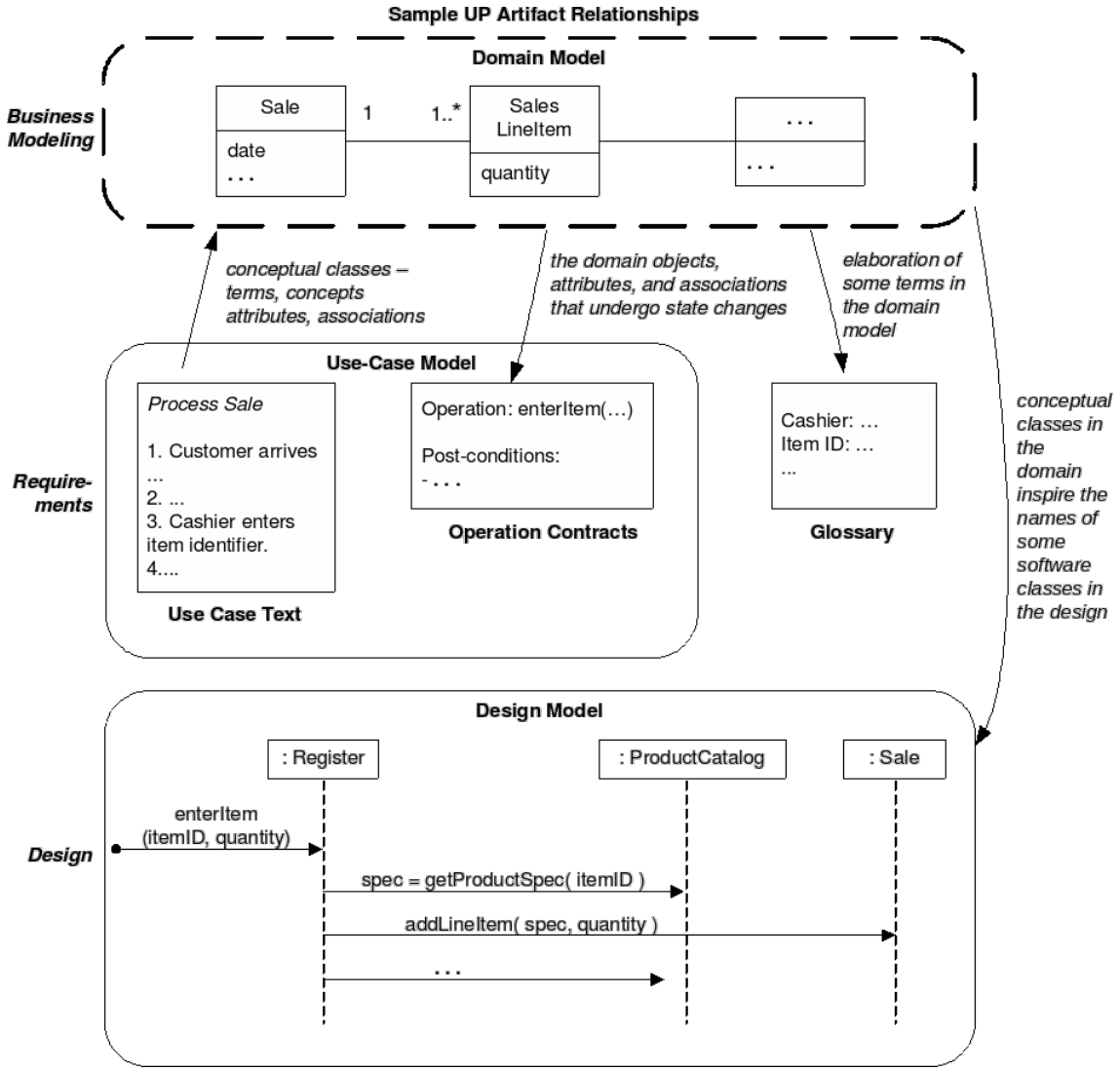
The requirements for the first iteration of the NextGen POS application follow:

- Implement a basic, key scenario of the *Process Sale* use case: entering items and receiving a cash payment.
- Implement a *Start Up* use case as necessary to support the initialization needs of the iteration.
- Nothing fancy or complex is handled, just a simple happy path scenario, and the design and implementation to support it.
- There is no collaboration with external services, such as a tax calculator or product database.
- No complex pricing rules are applied.

The design and implementation of the supporting UI, database, and so forth, would also be done, but is not covered in any detail.

(adopted from “Applying UML and Patterns” by Craig Larman)

Domain Model in the Context of Other Artefacts



(adopted from “Applying UML and Patterns” by Craig Larman)

Domain Model (DM)

- It describes concepts of the application domain, not SW objects.
  (aka “conceptual model”, however, do not confuse with CM in database modelling)
- It is not mandatory, however, it is useful and recommended.
  (the concepts of DM affect software objects in a domain layer of a design model)
- It is modelled as UML Class diagram.
- It is a specialisation of Business Object Model (BOM) in RUP.
  (BOM is an object model describing the realization of business use cases created during inception and finalized during the elaboration phase; contrary to BOM, DM focuses only on explaining "things" and products important to the business domain, e.g., excluding the responsibilities people will carry that are modelled in BOM)
- It should be done “just barely good enough”.
  (the goal is the understanding of the application domain, not a documentation)

Domain Model, not a Database Conceptual Model

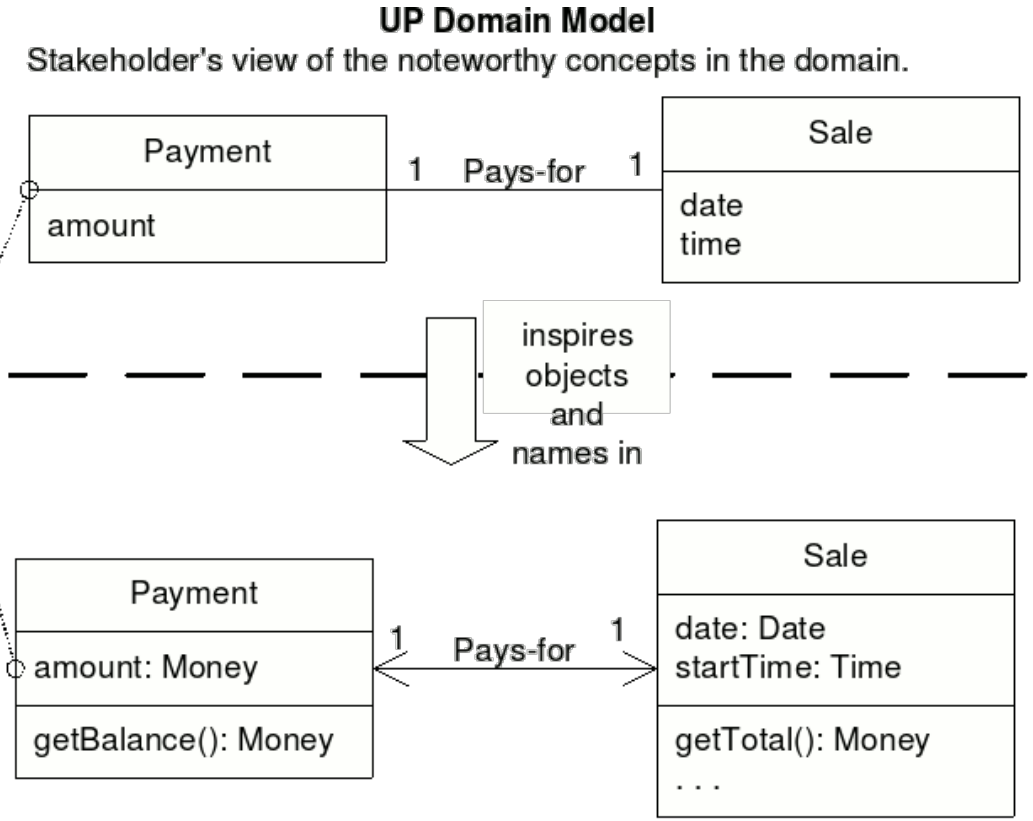
- Both DM and database CM are conceptual, however:
  - the domain model is about conceptual classes of a domain,
    (it can be regarded as a visual glossary of the application domain)
  - the DB conceptual model describes only persistent data.
    (the data stored in a storage, e.g., a database management system)
- DM is to understand the concepts of a application domain.
  (that is the reason to reuse the names of the concepts established in the DM as names of software objects in a design model during design and development)

Domain Model and Design Model

A Payment in the Domain Model is a concept, but a Payment in the Design Model is a software class. They are not the same thing, but the former *inspired* the naming and definition of the latter.

This reduces the representational gap.

This is one of the big ideas in object technology.



(adopted from “Applying UML and Patterns” by Craig Larman)

How to Create a Domain Model

1. Identify conceptual classes.
   1. Use of modify existing models of application architecture.
      (e.g., “Patterns of EA Architecture” by Martin Fowler, such as for accounting)
   2. Use a list of general categories for the application domain.
   3. Identify names of the domain concepts by linguistic analysis.
2. Model the identified conceptual classes in UML Class diagram.
3. Add associations and attributes of the classes.

An example of the list of general categories:

- category “business transaction”, e.g., for a sale, payment, etc.
- category “transaction item”, e.g., for a sale item, etc.
- category “transaction register”, e.g., for an accounting book, etc.

Another Examples of Conceptual Class Categories (1)

Conceptual Class Category	Examples
business transactions *Guideline*: These are critical (they involve money), so start with transactions.	*Sale, Payment* *Reservation*
transaction line items *Guideline*: Transactions often come with related line items, so consider these next.	*SalesLineItem*
product or service related to a transaction or transaction line item *Guideline*: Transactions are *for* something (a product or service). Consider these next.	*Item* *Flight, Seat, Meal*
where is the transaction recorded? *Guideline*: Important.	*Register, Ledger* *FlightManifest*

(adopted from “Applying UML and Patterns” by Craig Larman)

Another Examples of Conceptual Class Categories (2)

roles of people or organizations related to the transaction; actors in the use case *Guideline*: We usually need to know about the parties involved in a transaction.	*Cashier, Customer, Store* *MonopolyPlayer* *Passenger, Airline*
place of transaction; place of service	*Store* *Airport, Plane, Seat*
noteworthy events, often with a time or place we need to remember	*Sale, Payment* *MonopolyGame* *Flight*
physical objects *Guideline*: This is especially relevant when creating device-control software, or simulations.	*Item, Register* *Board, Piece, Die* *Airplane*
descriptions of things *Guideline*: See p. 147 for discussion.	*ProductDescription* *FlightDescription*

(adopted from “Applying UML and Patterns” by Craig Larman)

Getting Domain Concepts by Linguistic Analysis

Main Success Scenario (or Basic Flow):

1. **Customer** arrives at a **POS checkout** with **goods** and/or **services** to purchase.
2. **Cashier** starts a new **sale.**
3. **Cashier** enters **item identifier.**
4. System records **sale line item** and presents **item description**, **price**, and running **total**. Price calculated from a set of price rules.

Cashier repeats steps 2-3 until indicates done.

5. System presents total with **taxes** calculated.
6. Cashier tells Customer the total, and asks for **payment.**
7. Customer pays and System handles payment.
8. System logs the completed **sale** and sends sale and payment information to the external **Accounting** (for accounting and **commissions**) and **Inventory** systems (to update inventory).
9. System presents **receipt.**
10. Customer leaves with receipt and goods (if any).

Extensions (or Alternative Flows):

...

7a. Paying by cash:

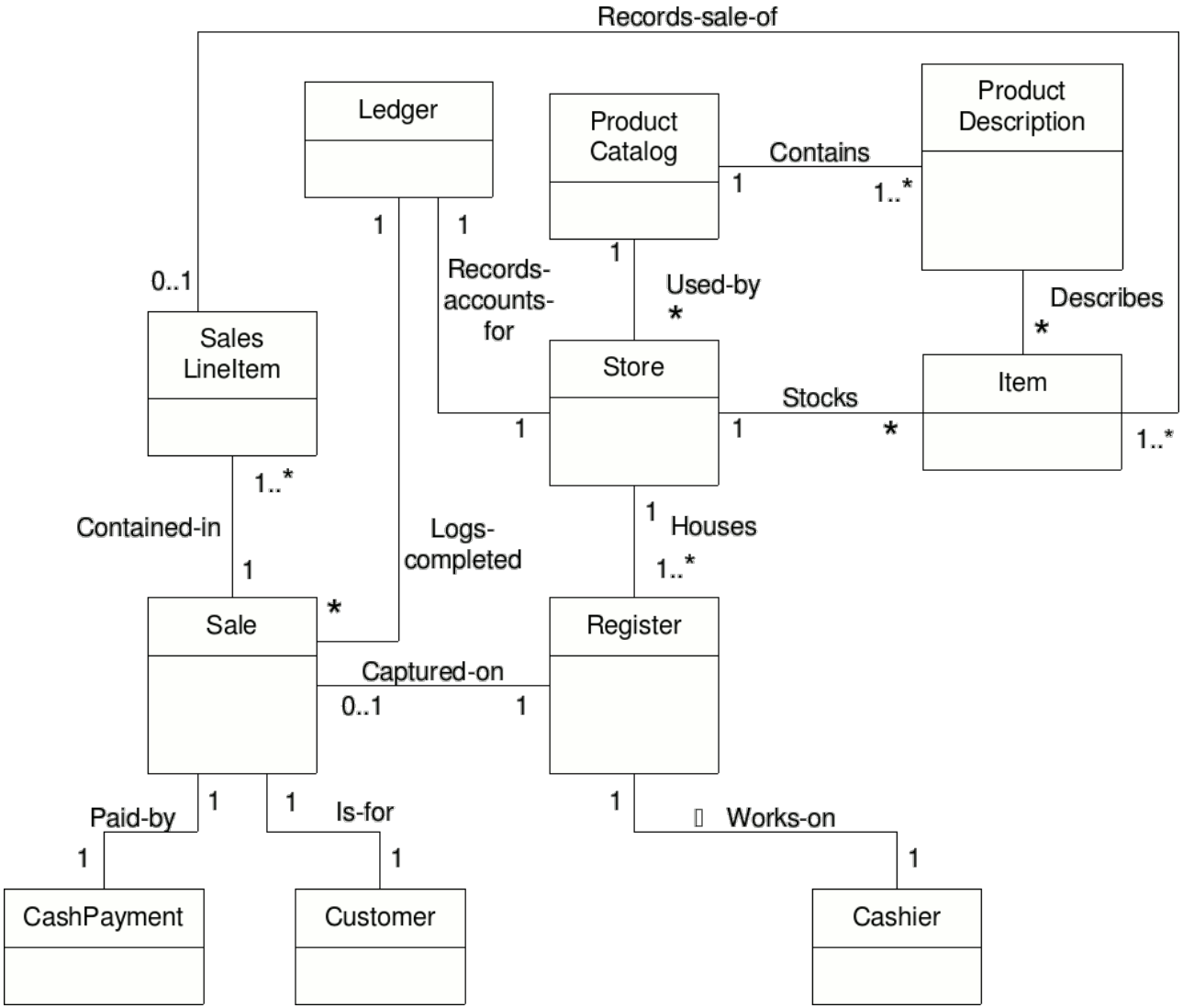
1. Cashier enters the cash **amount tendered.**
2. System presents the **balance due**, and releases the **cash drawer.**
3. Cashier deposits cash tendered and returns balance in cash to Customer.
4. System records the cash payment.

(adopted from “Applying UML and Patterns” by Craig Larman)

Best-practices in Domain Modelling

- Model conceptual class outlines, not detailed descriptions.
- Use whiteboards, flip-charts, common drawing tools, etc.
  (no special modelling tool is necessary at his level)
- Include output events of business actions if necessary.
  (e.g., a customer's bank account useful in a sale for cash-backs on discounts, etc.)
- Distinguish the conceptual classes from their attributes.
  (e.g., "fly" is a class and "destination" can be both attribute or class)
- Use "item descriptor" to extract attributes of multiple instances.
  (e.g., "sale" & "sale item", "book catalogue record" & "physical copy of a book", etc.)
- Associations of classes are their long-term relationships.
  (it is not important if/how they will be implemented later; e.g., "sale" & "sale item")
- There should be only important associations, not all possible.
  (a domain model need not to be complete to understand)

An Example: NextGen POS Domain Model (1st step)

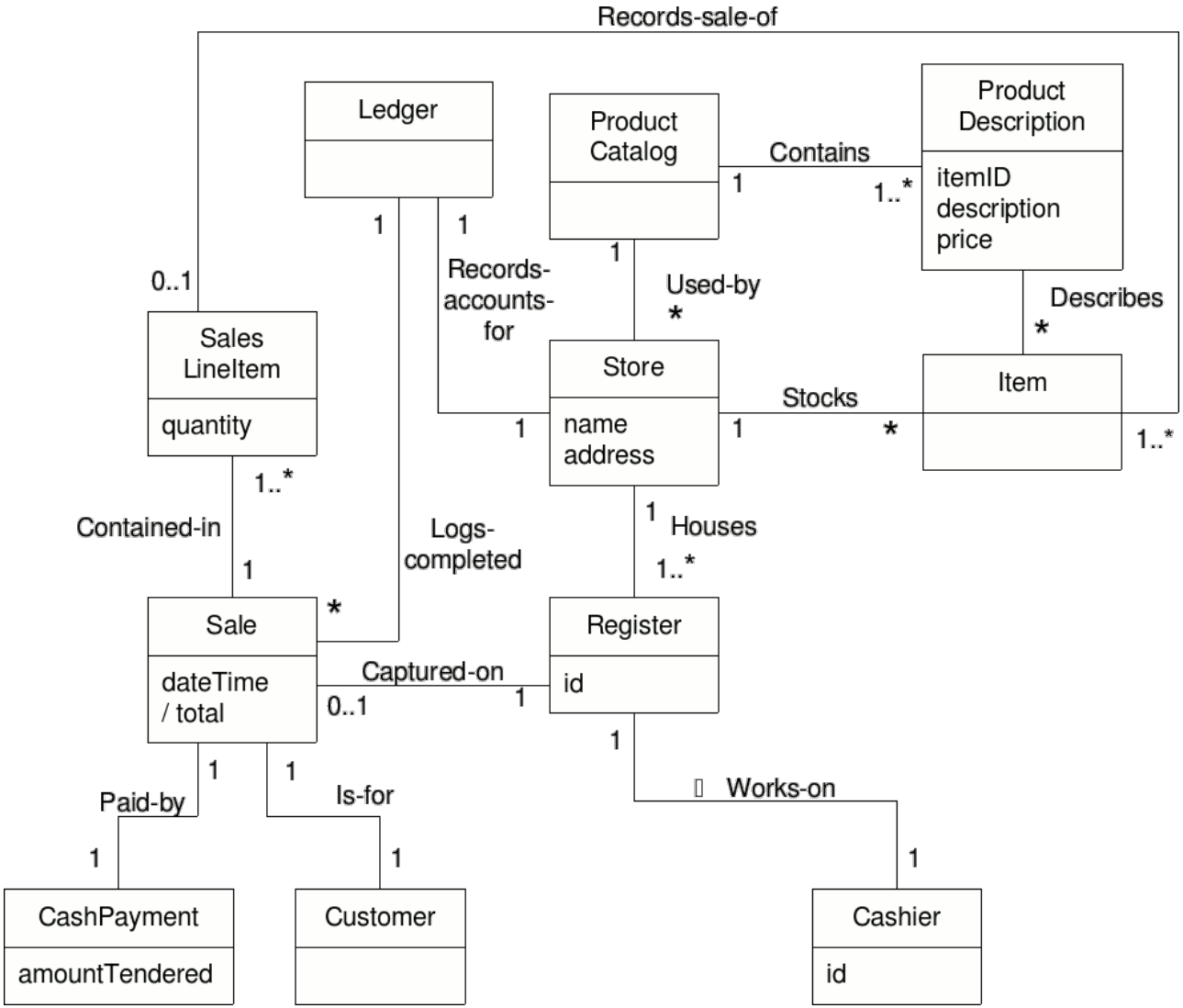


(adopted from “Applying UML and Patterns” by Craig Larman)

Best-practices for Attributes in Domain Modelling

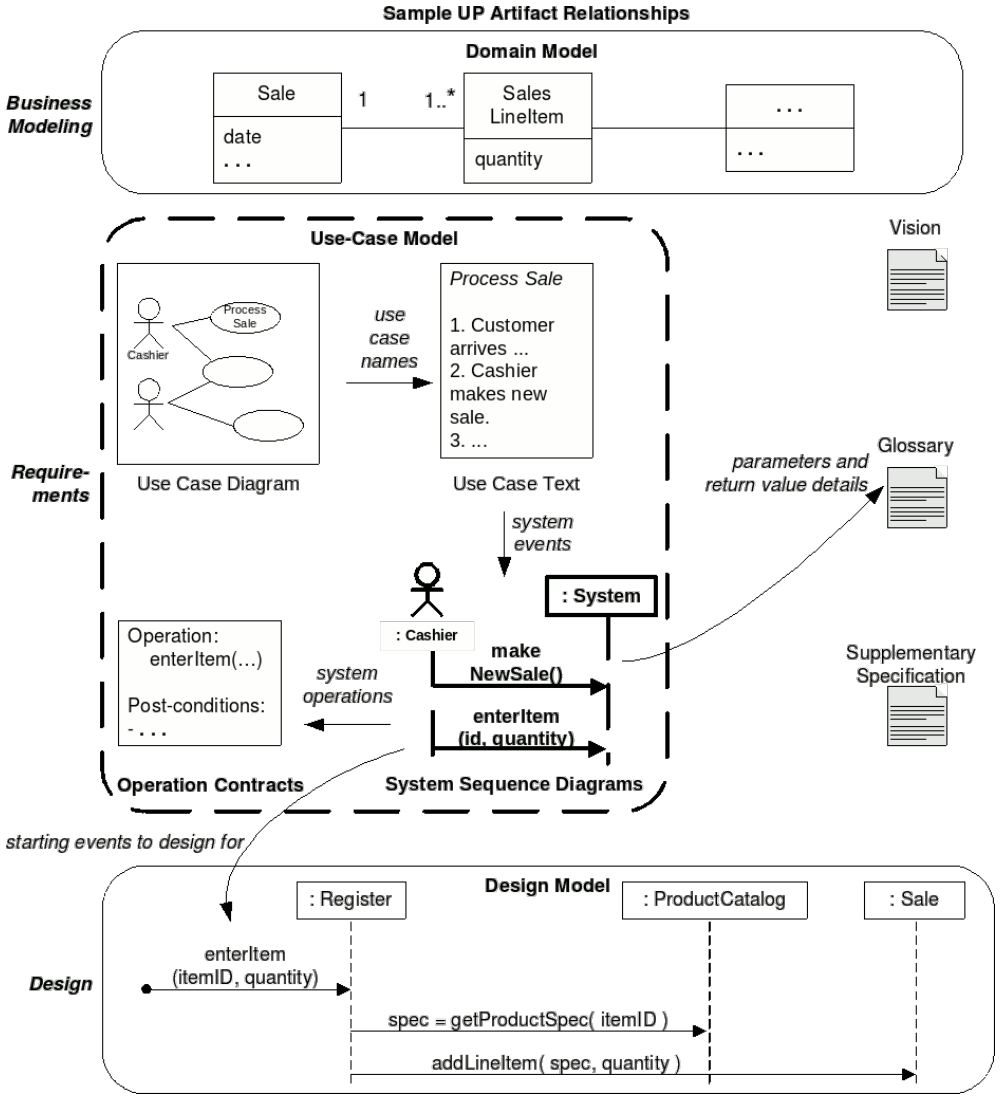
- Attributes of the classes are important to understand information requirements of user scenarios.
  (they represent knowledge on particular concepts of the application domain)
- Attributes contain values of the class instances to remember.
  (to remember during the life-time of the instances)
- It is useful to model also “derived attributes”.
  (their names should be prefixed by “/” according to the UML notation)
- The attributes should be of primitive data types, not structured.
  (the structured attributes may be instances of domain model classes)
- Model associations of classes rather than attributes describing that one class knows another.

An Example: NextGen POS Domain Model (2nd step)



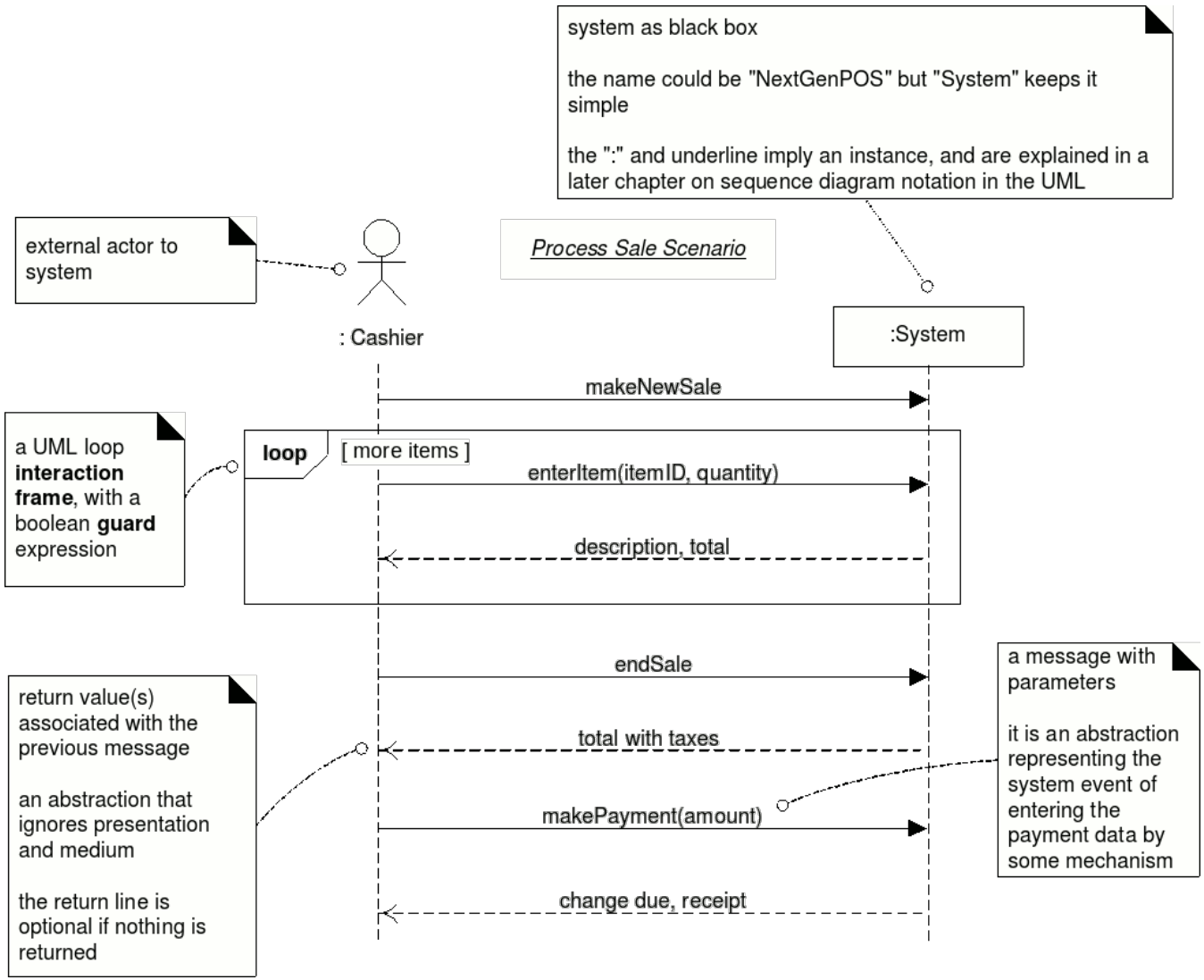
(adopted from “Applying UML and Patterns” by Craig Larman)

System Sequence Diagram (SSD) and Other Artefacts



(adopted from “Applying UML and Patterns” by Craig Larman)

An Example: SSD of Sale Process in NextGen POS



(adopted from “Applying UML and Patterns” by Craig Larman)

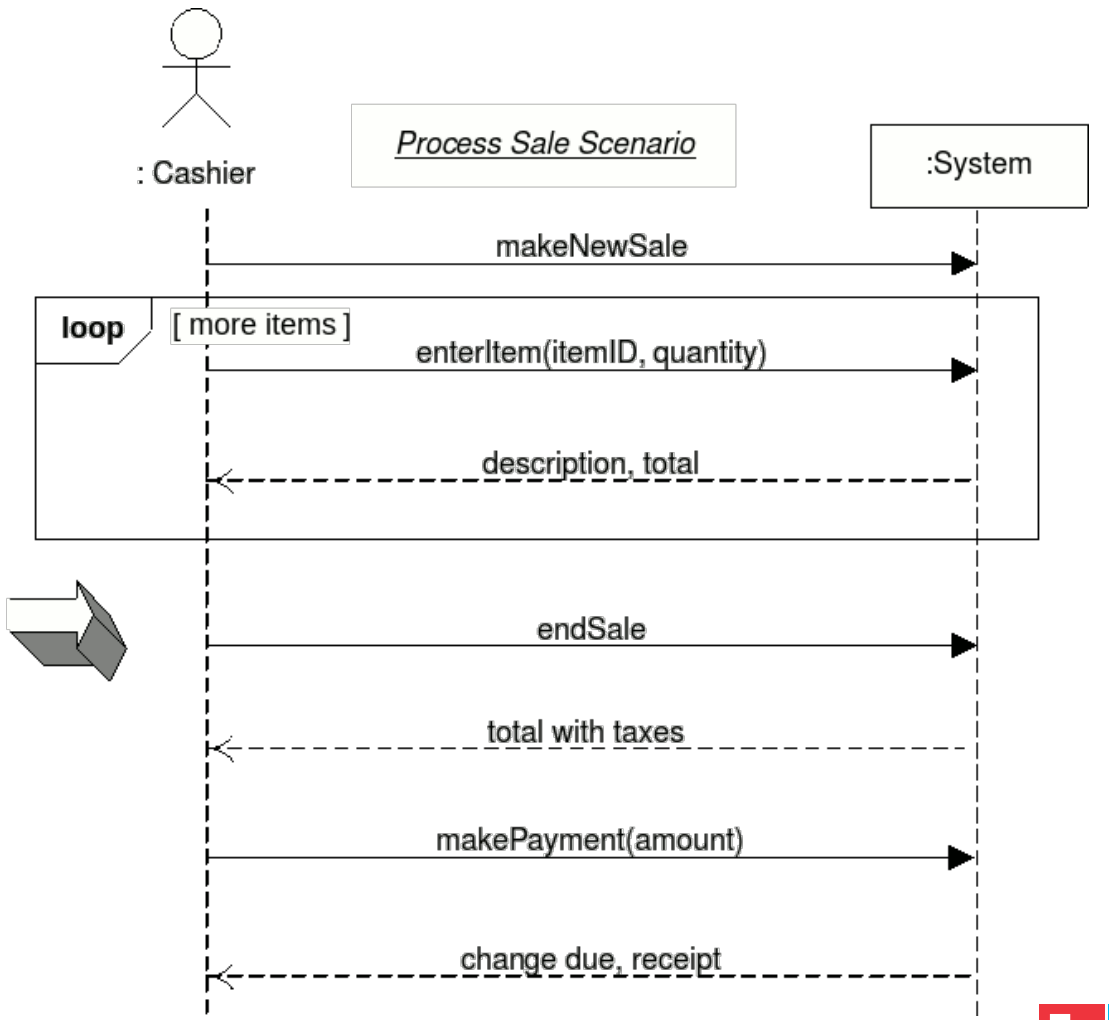
System Sequence Diagram (SSD)

- To model input and output events to/from a system.
  (the event-flow is initiated by an actor and a system operation is required)
- It is modelled as UML Sequence diagram.
- It describes actors, generated events and their order.
  (the events generated by the actors and a system)
- To describe main scenarios in each important use case.
  (the main scenarios and their frequent and complex alternatives)

An Example: NextGen POS Sale in Scenario and SSD

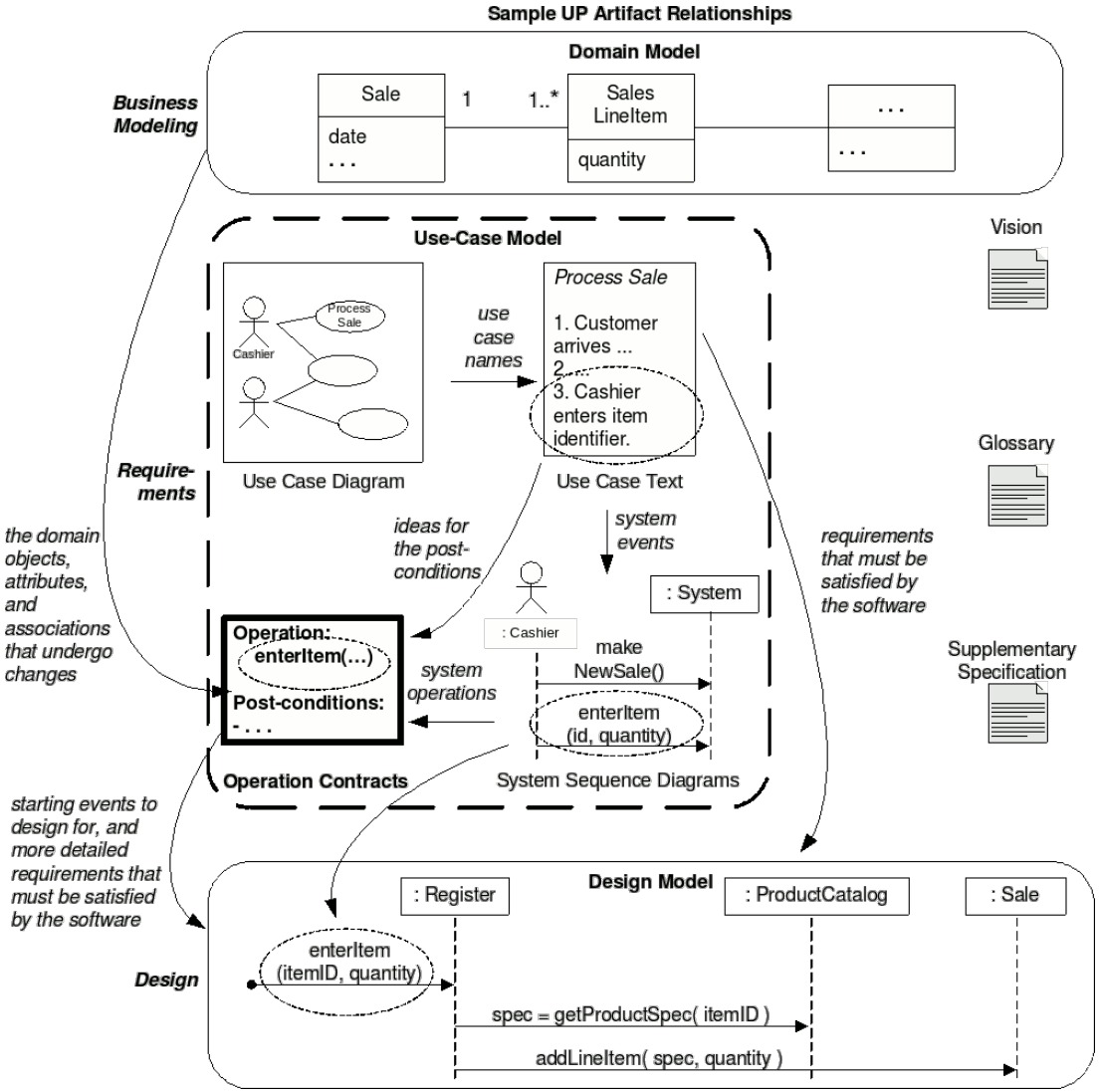
Simple cash-only *Process Sale* scenario:

1. Customer arrives at a POS checkout with goods and/or services to purchase.
2. Cashier starts a new sale.
3. Cashier enters item identifier.
4. System records sale line item and presents item description, price, and running total.
Cashier repeats steps 3-4 until indicates done.
5. System presents total with taxes calculated.
6. Cashier tells Customer the total, and asks for payment.
7. Customer pays and System handles payment.
...



(adopted from “Applying UML and Patterns” by Craig Larman)

Operation Contracts and Other Artefacts

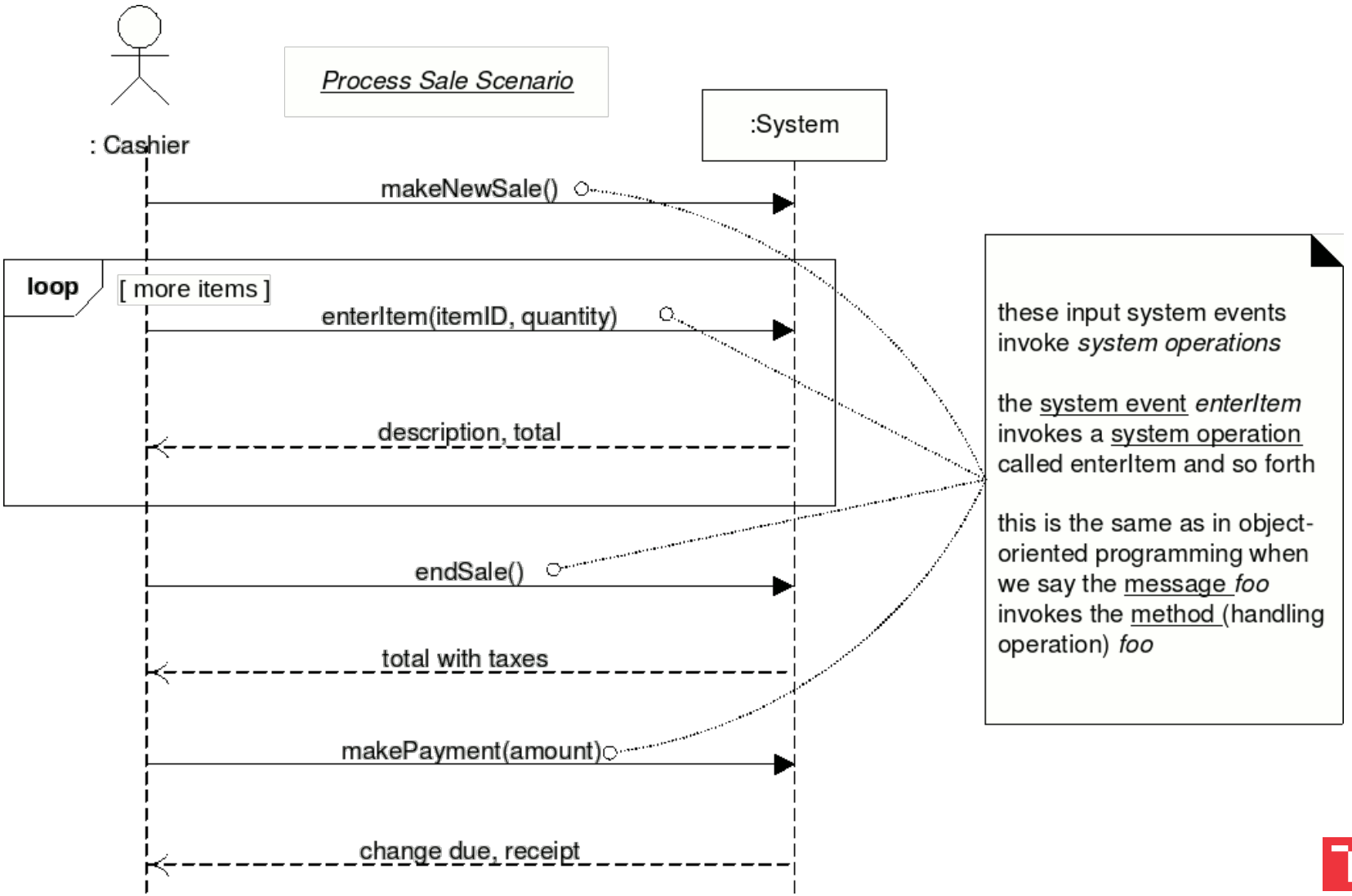


(adopted from “Applying UML and Patterns” by Craig Larman)

Operation Contracts

- To model operations provided to actors by a modelled system.
  (operations on the system's user interface or application interface, UI/API)
- The operations are invoked by incoming events (see SSD).
- A contract describes behaviour of a system during the operation.
  (especially, the state of domain model objects after the contracted operation, i.e., a post-condition of the operation)
- It can be considered as an extension of the use case model.
  (it makes the use cases more detailed in terms of the interaction with a system)

An Example: Input Events and Operation Contracts



(adopted from “Applying UML and Patterns” by Craig Larman)

An Example: Operation “enterItem” Contract

Contract CO2: enterItem

Operation:	enterItem(itemID: ItemID, quantity: integer)
Cross References:	Use Cases: Process Sale
Preconditions:	There is a sale underway.
Postconditions:	– A SalesLineItem instance sli was created *(instance creation)*. – sli was associated with the current Sale *(association formed)*. – sli.quantity became quantity *(attribute modification)*. – sli was associated with a ProductDescription, based on itemID match *(association formed)*.

The categorizations such as “*(instance creation)*” are a learning aid, not properly part of the contract.

(adopted from “Applying UML and Patterns” by Craig Larman)

Thank you for your attention!

Marek Rychlý
<rychly@fit.vutbr.cz>