

FUSION with USE CASES - EXTENDING FUSION FOR REQUIREMENTS MODELLING

(Draft)

Derek Coleman

HP Labs

dc@hplsrd.hpl.hp.com

H

Derek Coleman File: 011 October 20,
1995
Hewlett-Packard Laboratories

REQUIREMENTS PHASE

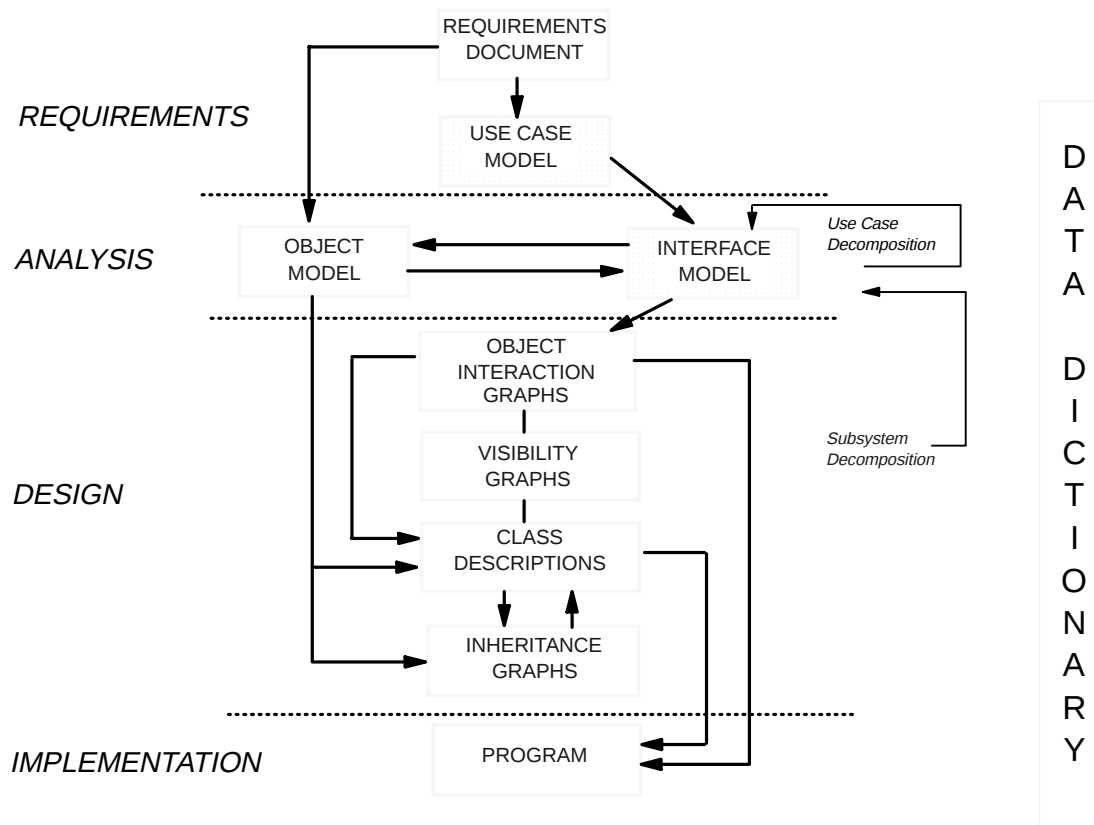
Purpose:

- Set of techniques for capturing the customer's concepts by constructing model(s) that can be
 - reviewed by the customer
 - form the basis of OOA/D
- Support traceability of requirements through to code
- Not a requirements gathering technique

STATE OF ART

- Consensus that the Objectory 'use case' approach is right.

MODIFYING THE FUSION METHOD



- extra model - use case model
- extra refinement step decomposing use cases into system operations
- some changes to interface model

Derek Coleman File: 011 October 20,
1995
Hewlett-Packard Laboratories

AGENTS AND USE-CASES

Agent (instance) : active entity that interacts
with system (aka Actor in Objectory).

Agent (class) : a set of agents with the same potential behaviour.

system : the agent of interest.

use case (instance) : a goal directed sequence of transactions with the
system whose task is to yield a result of
identifiable value to one or more agents.

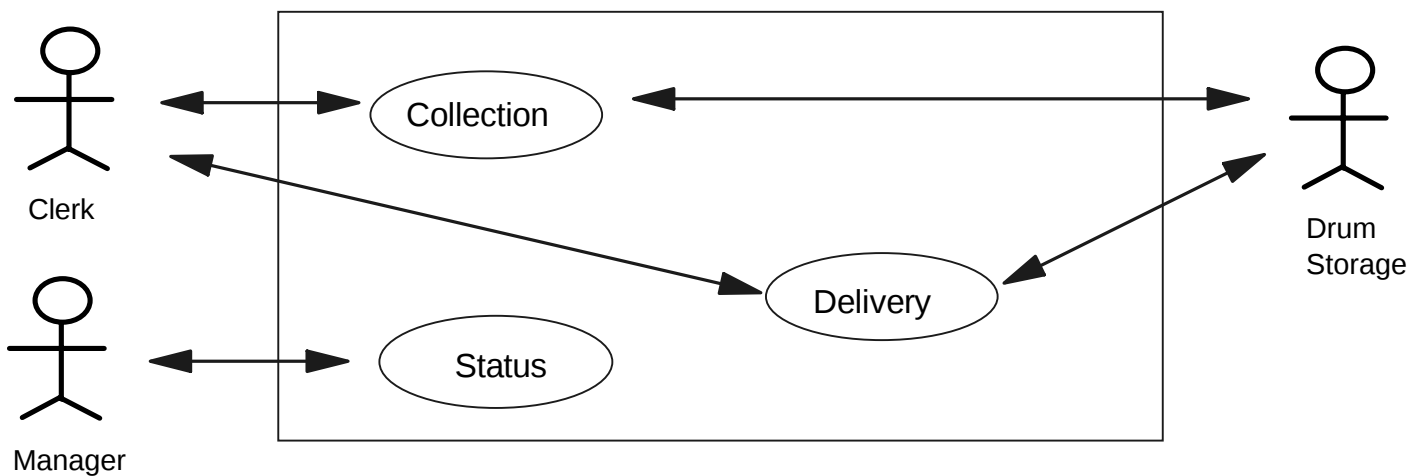
use-case (class) : a set of related use case instances.

notation note : class identifiers start with upper case
instance identifiers start with lower case

REQUIREMENTS PROCESS

- R1. Define use cases and use case model.
- R2. Structure the set of use cases.
- R3. Define an external view of the system in terms of the relationship between use cases.
- R4. Review with customer and iterate steps R1, R2 and R3.

STEP R1: USE CASE MODEL

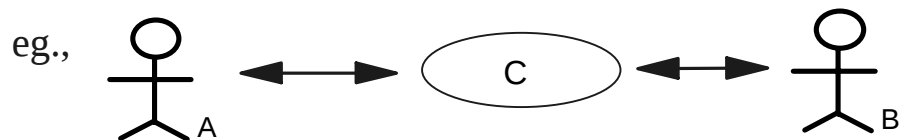


- A diagram showing agents and use cases in which they participate with the system.
- Use cases are documented by structured natural language showing sequence of transactions.

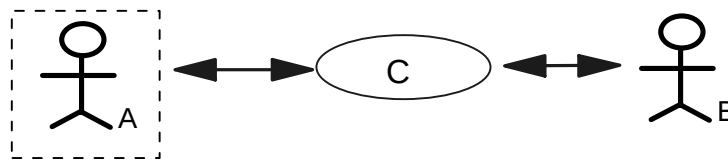
H

STEP R1: USE CASE MODEL NOTATION

- Use case model shows signature of each use case.



- Indicates each instance of c involves only interactions between a and b and system



- Indicates each instance of c involves interactions between some set of a's and b

USE CASE DOCUMENTATION

Use case: name

Description: goal statement

Transactions: steps in use case

- The description should be a clear statement of what the use case is supposed to accomplish. It defines the value of the use case.
- The transactions are numbered or lettered paragraphs. Each paragraph defines a component step in the evolution of the use case.

DELIVERY USE CASE (ECO EXAMPLE)

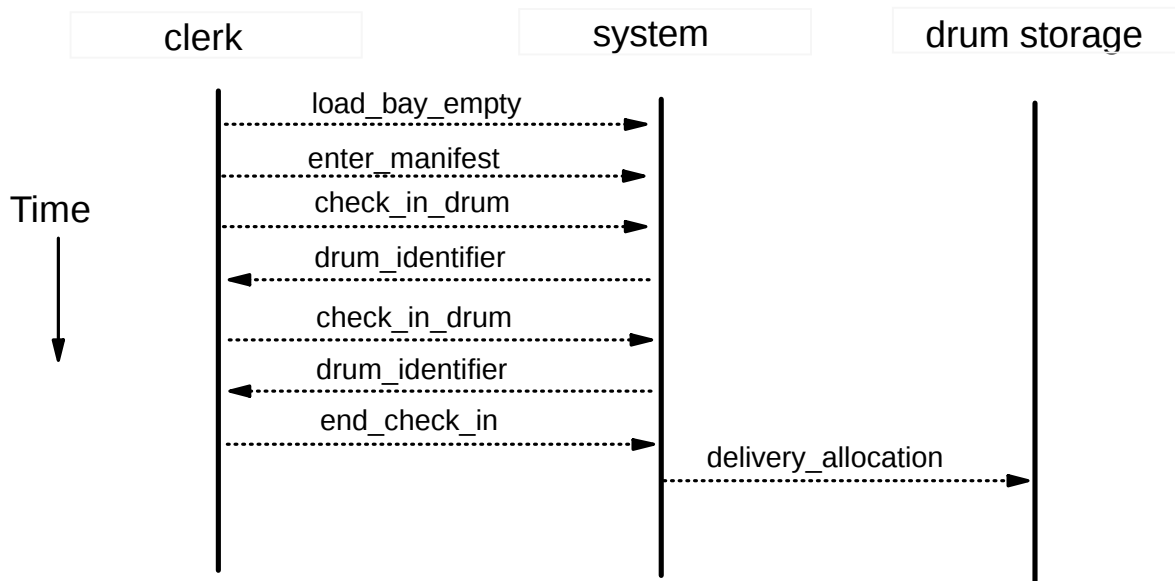
use case: Delivery

description: Delivery of a load of drums to depot. It's assumed that the manifest is always correct and there is always enough room in the depot for the drums.

transactions:

- a. The clerk tells the information system that the loading bay is now empty, so a delivery may begin.
- b. The clerk then gives the system the details from the manifest, and starts checking in the drums. The system issues identifiers for each of the checked-in drums.
- c. The system computes in which the store buildings the drums are to be stored and tells the drum storage.

SCENARIO DIAGRAMS FOR USE CASE INSTANCES



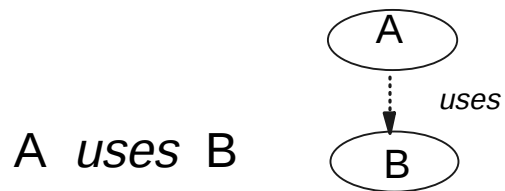
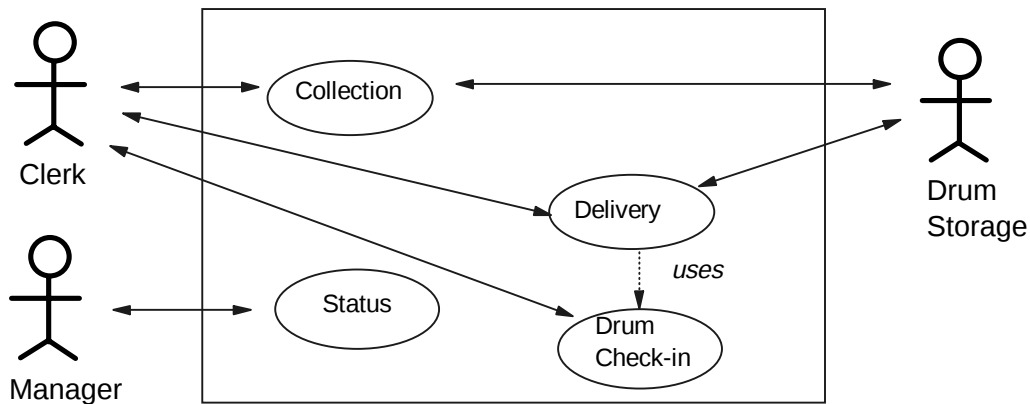
- as an aid to understanding, scenario diagrams can be used to show event flows for use case instances.
- ✓ good for showing to customers.
- ✗ may anticipate analysis by deciding protocol for interacting with system eg. every drum entered individually.

H

STEP R2: STRUCTURING USE CASES

- For real systems there are two problems
 - individual use cases may be *long and complex* because they are too concrete.
 - there can be a huge number of use cases because of many different variants of essentially the same use case.
- There are two structuring devices
 - uses* - which is an abstraction mechanism.
 - extends* - which allows variants to be defined.

STEP R2: "USES" - AN ABSTRACTION MECHANISM



- A incorporates B as a sub-flow of events.
- The details of use case B are hidden from A
- A must say where B is inserted. (cf subroutine)
- B is a fully fledged use-case
 - it may involve some or all of A's agents.

STRUCTURING USE CASES WITH "Uses"

use case: Delivery

description: Delivery of a load of drums to depot. It is assumed that the manifest is always correct and there is always enough room in the depot for the drums.

transactions:

- a. The clerk tells the system that the loading bay is now empty, so a delivery may begin.
- b. The clerk performs a DRUM CHECK-IN use case.
- c. The system computes in which storage buildings the drums are to be stored and sends the allocation to drum storage.

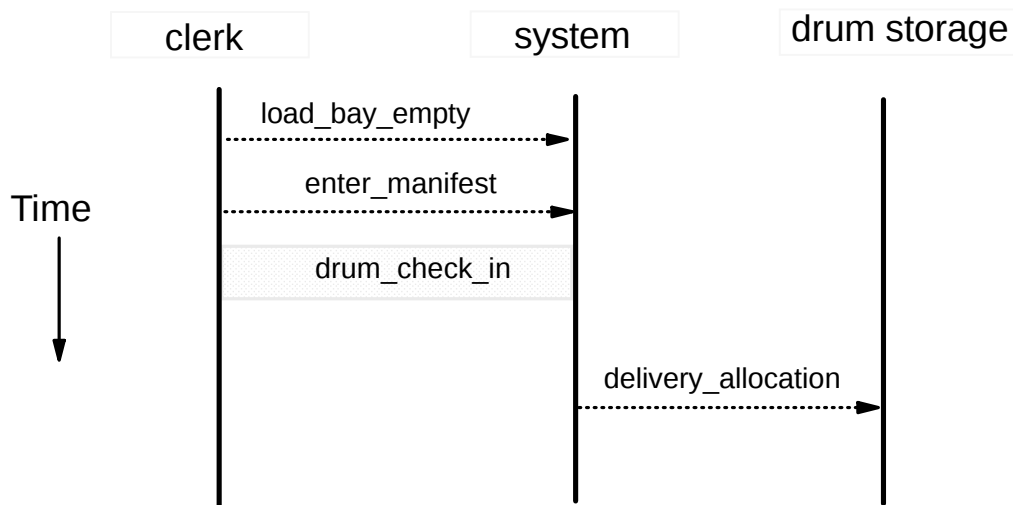
use case: Drum Check-in

description: Entry of a load manifest and all the drums in a load

transactions:

- a. The clerk gives the system the details from the manifest, and starts checking in all the drums. The system identifies for each of the checked-in drums.
- b. Where there are no more drums to be checked-in the clerk tells the system.

SHOWING SUB USE CASES ON SCENARIO DIAGRAMS



- sub use case instances can be shown by a *shaded box* joining the agents participating in the use case.
- instances of the sub use case can be shown on a separate scenario diagrams.

H

Derek Coleman File: 011 October 20,
1995
Hewlett-Packard Laboratories

USES

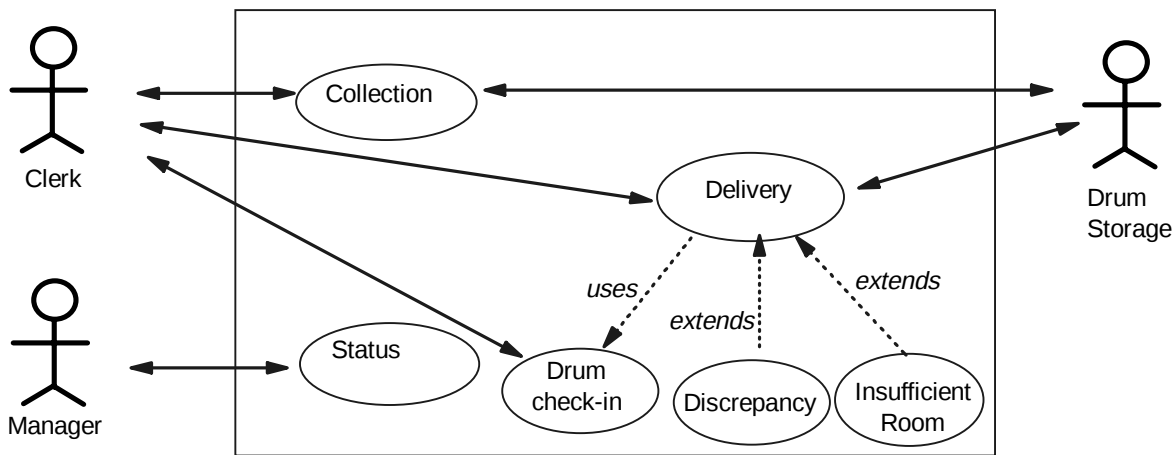
- Like a procedure call, thus a use case may be used by many different use cases
e.g., "identify customer" could be used by "place an order", "create invoice", etc.

STRUCTURING USE CASES WITH EXTENDS

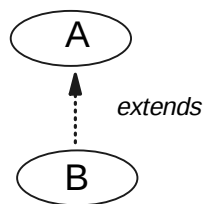
- Allows variants of a simple or ideal use case to be defined.
- Delivery makes the assumptions that
 - the manifest and load always agree.
 - there is always enough room for the load in the depot.
- A realistic Delivery use case can be defined by with two *extensions*
 - Discrepancy
 - Insufficient Room

which remove the assumptions.

STRUCTURING USE CASES: Extends



B *extends* A



- Defines two use cases, A and 'A extended by B'.
- B introduces a variation to A by inserting a conditional flow of events, for a failure or to deal with an extra complexity.
- ✗ B must say where it is inserted in A.
- ✗ B is not a fully fledged use case.

H

EXTENSION USE CASES

use case: Delivery

description: Delivery of load of drums to depot. It is assumed that the manifest is always correct and there is always enough room in the depot for the load.

transactions:

- a. The clerk tells the system that the loading bay is now empty, so a delivery may begin.
- b. The clerk performs the DRUM CHECK-IN use case.
- c. The system computes in which storage buildings the drums are to be stored and sends the allocation to drum storage.

extension

use case: Discrepancies

description: Mismatch between manifest and drums checked-in

transactions:

- a. After step b in the Delivery use case, if any drum has been checked in which is not on the manifest, or if a drum is missing from the manifest then the system informs the clerk.

extension

use case: Insufficient Room

description: Not enough room for drums to be stored in the depot.

transactions:

- a. After step c in the Delivery use case, if any drums cannot be stored, the system tells the clerk so that they can be returned from whence they came.

H

Derek Coleman File: 011 October 20,
1995
Hewlett-Packard Laboratories

EXTENDS AND EVO-FUSION

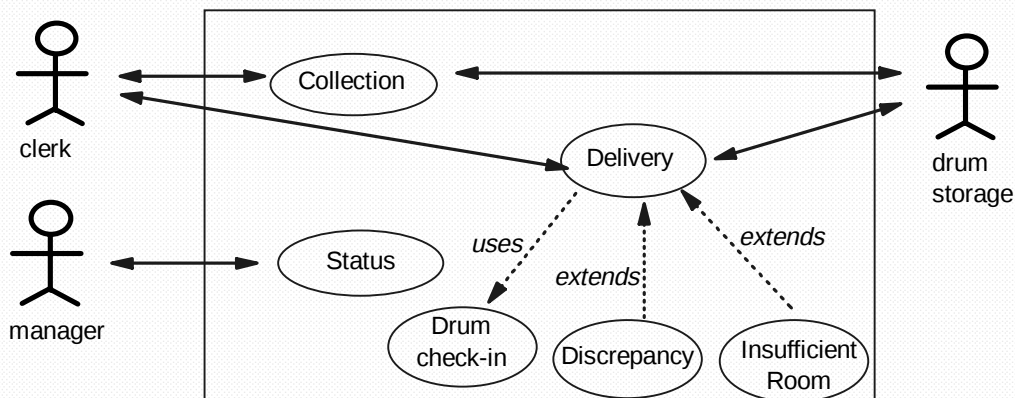
- Extends is an "editing" construct for incrementally introducing complexity into use cases.
- Useful when defining an incremental development process, e.g.
 - release #1: Delivery
 - release #2: extended by Discrepancy
 - release #3: extended by Insufficient Room

STEP R3: DEFINE EXTERNAL VIEW OF SYSTEM

- Before analysis we need to establish
 - what agent instances there are in the environment.
 - how do the use cases interact in time.
- Shown on an use case instance model which is the use case model with
 - agent classes replaced by instances.

STEP R3: DEFINE EXTERNAL VIEW OF ECO SYSTEM

ECO System



Behaviour :

The ECO system interacts with three agents: clerk, management and drum storage. There can be any number of occurrences of Delivery, Collection and Status. Instances of Delivery and Collection may occur in any order but are mutually exclusive, ie, they must not overlap. A Status use case may occur at any time and may overlap a Delivery or Collection use case.

- Overall ECO system behaviour is a composition of use cases.

AT END OF REQUIREMENTS

- Requirements defined by Agents and structured Use cases.
- Use case instances illustrated by event scenarios.
- External view of system behaviour defined by composition of use cases.
- Customer should review and agree
 - but final sign-off may only come during analysis

ANALYSIS STAGE OF FUSION

Purpose:

- What the system does
- First cut at objects

Modification to method provides

- Traceability between use cases and system operations (ie, product features)

CHANGES TO FUSION ANALYSIS PHASE

- Only Interface Model is affected
- Operation Model (schema) - generalized to apply to use cases
- Lifecycle Model replaced by interaction graphs
 - decomposition of use cases into system operations shown by a *system interaction graph*.

PROCESS FOR DEVELOPING INTERFACE MODEL

- A1. Specify functional behaviour of each complete use case using a schema. Check for overlapping use cases.
- A2. Decompose use cases into component system operations and output events. Use scenarios to determine appropriate level of granularity.
- A3. [Optional] For each use case draw a system interaction graph showing the sequencing of system operations.
- A4. Complete operation model schemata for system operations as usual.
- A5. Check consistency between use cases and their component system operations.

H

Derek Coleman File: 011 October 20,
1995
Hewlett-Packard Laboratories

STEP A1: SPECIFYING FUNCTIONAL BEHAVIOUR OF USE CASES

- A use case is like a general and non-atomic system operation.
- The definition of a use case is enriched by adding a schema.
- Schemata are written for fully fledged use cases, i.e., simple use cases or use cases together with extensions. Schemata are never written for extensions alone.
- Schema pre- and post conditions provide a more precise definition of the use case goal in terms of its effect on the system.
- Schemata provide the criteria for use case based testing and can help identify object classes.

FORMAT FOR USE-CASE SCHEMA

Use case:	name
Description:	goal statement
Reads:	object instances that may be accessed but not changed by the use case.
Changes:	object instances that may be accessed <u>or</u> changed by the use case.
In:	<agents and the information that they supply <i>to</i> the system>
Out:	<agents and the information that they receive <i>from</i> the system>
Assumes:	pre-condition for use case.
Results:	post-condition for use case.
Transactions:	steps in use case

- Information
 - a *set of values* e.g., set of drum types that are checked in. or
 - an *individual data value*
e.g., manifest or
 - a *constant or event*
e.g., end-check-in
- In and Out specify the data that must cross the user interface during the use case.

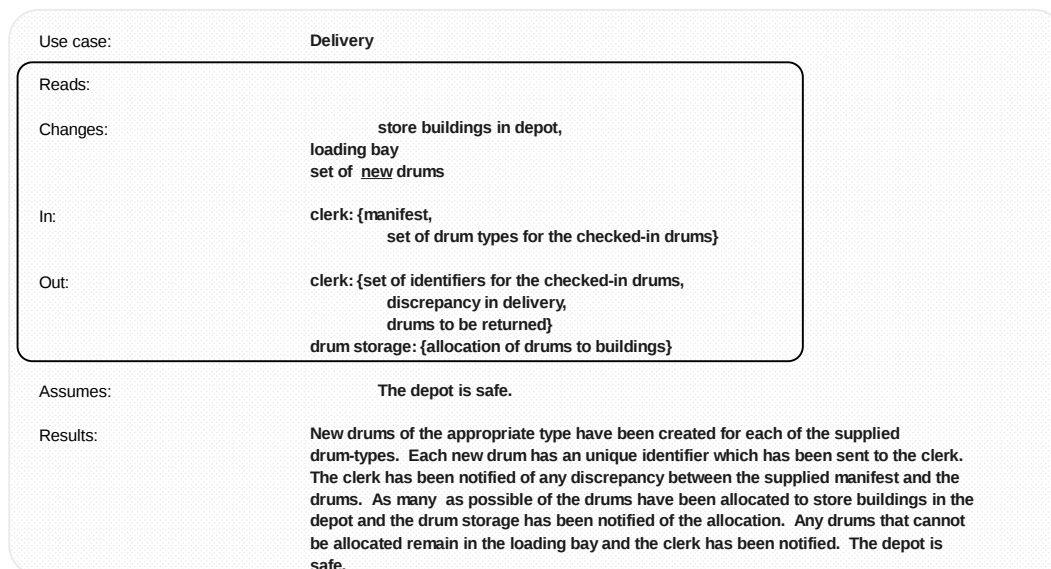
USE-CASE SCHEMA

Use case:	Delivery extended by Discrepancy and Insufficient room.
Description:	Delivery of a load of drums to depot.
Reads:	
Changes:	store buildings in depot, loading bay set of <u>new</u> drums
In:	clerk: {manifest, set of drum types for the checked-in drums}
Out:	clerk: {set of identifiers for the checked-in drums, discrepancy in delivery, drums to be returned} drum storage: {allocation of drums to buildings}
Assumes:	The depot is safe.
Results:	New drums of the appropriate type have been created for each of the supplied drum-types. Each new drum has an unique identifier which has been sent to the clerk. The clerk has been notified of any discrepancy between the supplied manifest and the drums. As many as possible of the drums have been allocated to store buildings in the depot and the drum storage has been notified of the allocation. Any drums that cannot be allocated remain in the loading bay and the clerk has been notified. The depot is safe.

USE-CASE SCHEMA
1995
Hewlett-Packard Laboratories

October 20,

STEP A1: FINDING THE OBJECTS FROM USE CASE SCHEMA



- Pre- and post conditions help find the objects - they are in the reads and changes clauses.
- The information in **In** and **Out** may also become objects or data attributes.

STEP A1: OVERLAPPING USE CASES

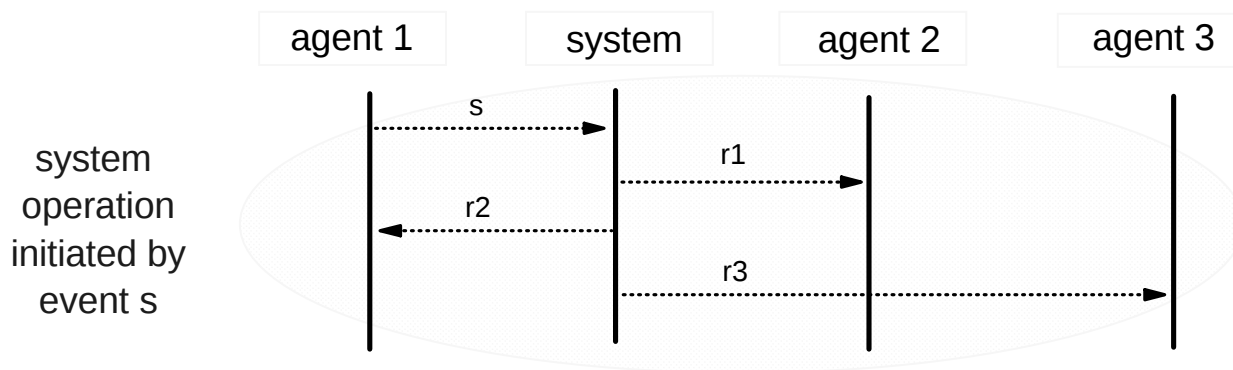
- If two use case instances can overlap in time then they can interfere with each other if they have state space in common.
- Simplest policy is to ensure that, as far as possible, the state space of potentially over-lapping use cases is disjoint and to introduce synchronization to protect shared state.
 - e.g., if Delivery and Collection could overlap then access to loading bay and buildings in depot would need to be synchronized.

STEP A2: USE CASE DECOMPOSITION

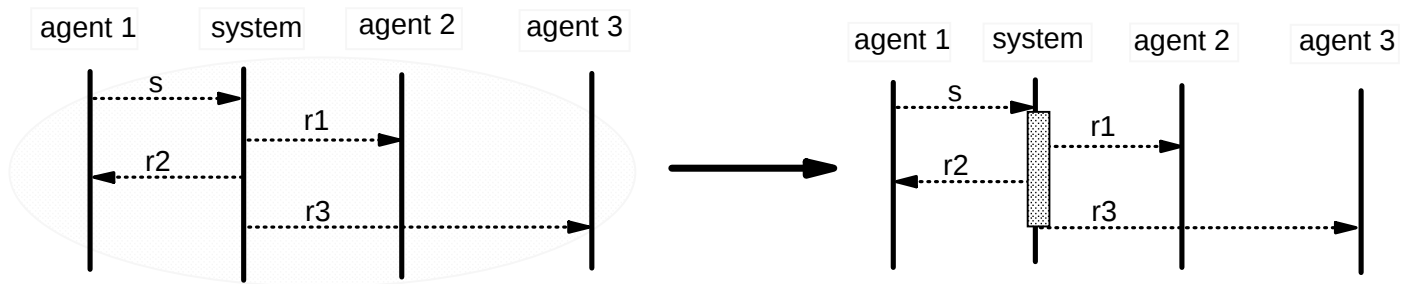
- A use case can be decomposed into sub use cases or system operations.
- The key difference is that *system operations are atomic and cannot overlap*.
- System operations correspond to the notion of uninterruptible procedure or machine instruction.

SYSTEM OPERATION AND USE CASES

- A *system operation* is a restricted pattern of event flows. One agent *initiates* the system operation by sending an event to the system. The system responds by sending zero or more events to some agents. A system operation is atomic, ie., system may not receive any other events during its execution.



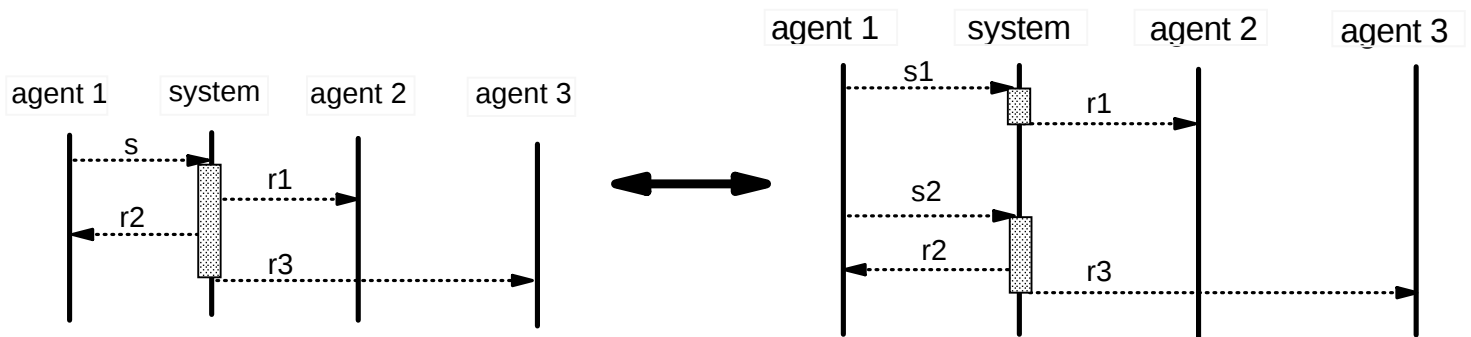
SHOWING SYSTEM OPERATIONS ON SCENARIO DIAGRAMS



- A system operation is shown as a box on the system timeline which
 - starts with the initiating event
 - encompasses all the output events of the system operation
- "System operation s" is shorthand for "system operation initiated by event s"

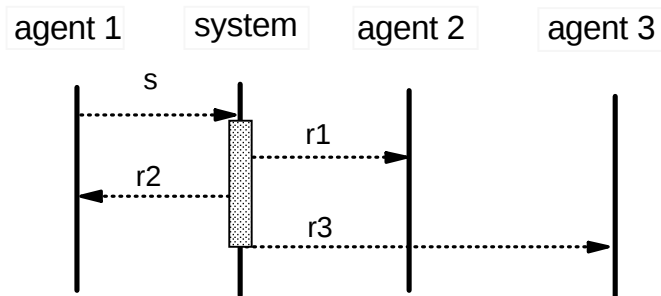
GRANULARITY OF SYSTEM OPERATIONS

- A system operation can be decomposed into a sequence of lower granularity system operations by introducing new initiating events.



- The effect of s can be achieved by s1 followed by s2.
- Decomposition makes the use case more interactive.
- Similarly a sequence of system operations can be composed to form a larger grain system operation.

SCENARIOS AND SCHEMATA FOR SYSTEM OPERATIONS



Operation: s

Reads:

Changes:

Sends: agent 1 {r2}
agent 2 {r1}
agent 3 {r3}

Assumes:

Result:

- Scenarios drive definition of operation schemata.

H

STEP A2: DECOMPOSE USE CASES INTO SYSTEM OPERATIONS

A2.1: Draw a representative set of event scenarios for each use case. Ensure all 'corner' cases are covered.

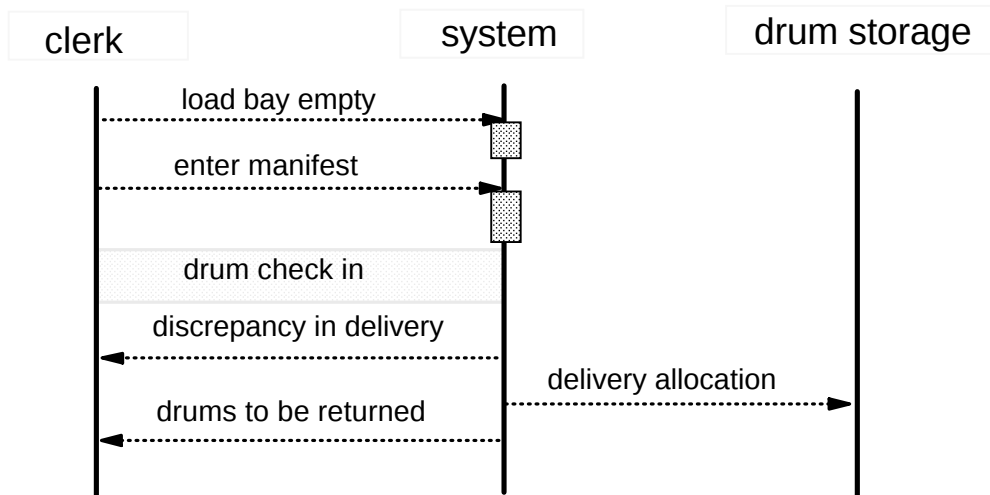
Each event sent to the system initiates a system operation. Where necessary

- add new initiating events
- adjust definition of sub use cases to ensure atomicity of system operation.

A2.2: Define first cut at the schema (description and sends clause) for each system operation that is introduced.

A2.1: PARTITION SCENARIOS INTO SYSTEM OPERATION

Delivery:



- load bay empty and enter manifest initiate system operation with no outputs.

Problem

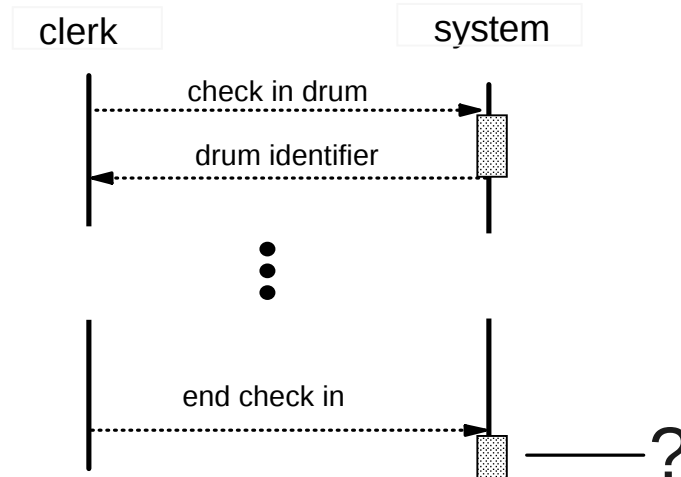
- there is no initiating event to cause the output of discrepancy in delivery, delivery allocation and drums to be returned.

H

Derek Coleman File: 011 October 20,
1995
Hewlett-Packard Laboratories

A2.1: ADJUSTING SUB USE CASES

Drum Check-in:



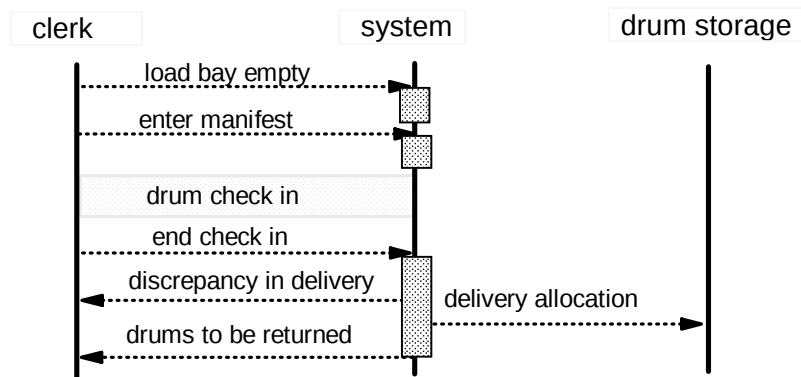
- `check_in_drum` can initiate a system operation that outputs `drum_identifier`
- `end_check_in` can initiate a system operation that outputs `discrepancy_in_delivery`, `delivery_allocation` and `drums_to_be_returned`.

Solution

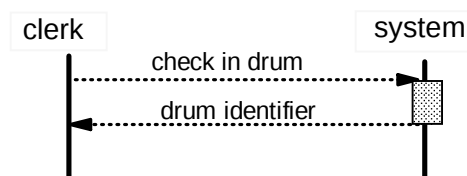
H

A2:1 SCENARIO SHOWING FINAL PARTITION OF DELIVERY

Delivery:



Drum Check-in:



H

A2.2: FIRST CUT AT SYSTEM OPERATION SCHEMATA

Operation: load bay empty

Description:

Reads:

Changes:

Sends: -----

Assumes:

Result:

Operation: enter manifest

Description:

Reads:

Changes:

Sends: -----

Assumes:

Result:

Operation: check in drum

Description:

Reads:

Changes:

Sends: clerk: {drum-identifies}

Assumes:

Result:

Operation: end check in

Description:

Reads:

Changes:

Sends: clerk: {discrepancy in delivery,
drums to be returned}
drum storage: {delivery allocation}

Assumes:

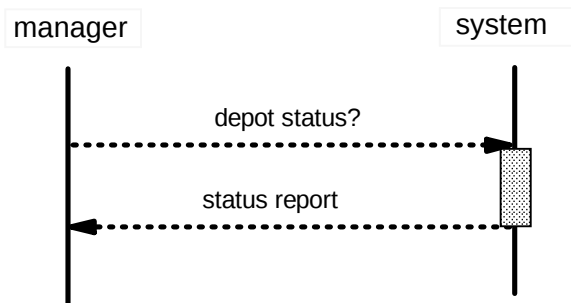
Result:

H

Derek Coleman File: 011 October 20,
1995
Hewlett-Packard Laboratories

STEP A2.2: FIRST CUT AT SCHEMATA FOR OTHER USE CASES

Status:



Operation: depot status?
Description:
Reads:
Changes:
Sends: manager: {status report}
Assumes:
Result:

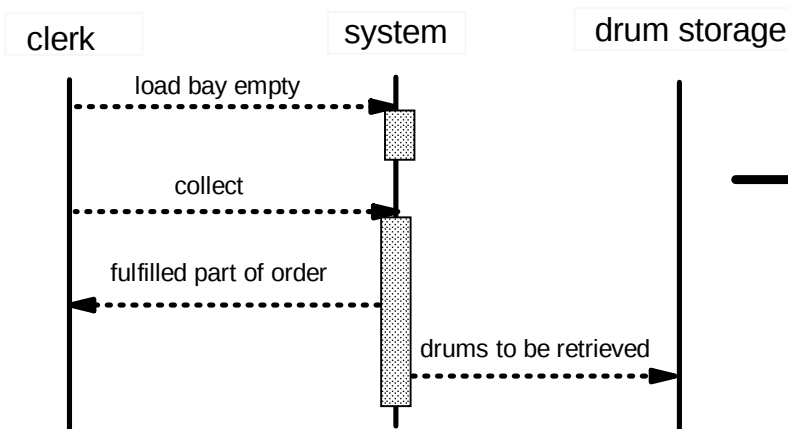


Operation: is vulnerable
Description:
Reads:
Changes:
Sends: manager: {vulnerability report}
Assumes:
Result:

H

STEP A2.2: DEFINE FIRST CUT SCHEMATA

Collection:



Operation: Collect

Description:

Reads:

Changes:

Sends: clerk: {fulfilled part of order}
drum storage: {drums to be retrieved}

Assumes:

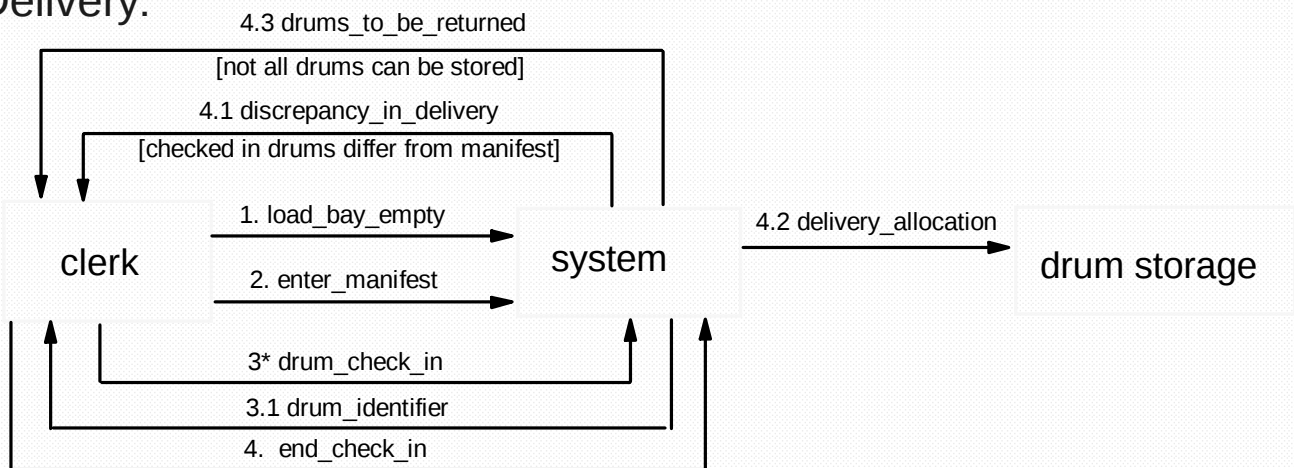
Result:

STEP A3: SYSTEM INTERACTION GRAPHS [OPTIONAL]

- The Transaction steps defined for a use case class are formalized as a System Interaction Graph (SIG)
- Each SIG defines the flow of system operations and output events for a use case class.
- The diagrammatic notation is almost identical to object interaction graphs. The difference with OIGs are:
 1. out arrows from the system are events not messages
 2. there is no unlabeled initiating arrow

STEP A3: SYSTEM INTERACTION GRAPHS

Delivery:

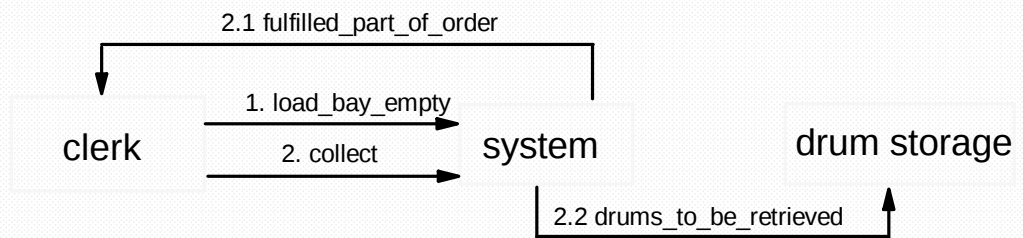


Description:

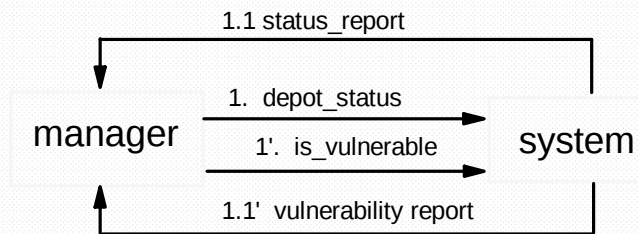
The clerk issues `load_bay_empty` and `enter_manifest` system operations. The clerk then issues a sequence of `drum_check_in` operations, and corresponding to each system operation receives a `drum_identifier`. Finally the clerk issues an `end_check_in` system operation which results in a `delivery_allocation` being sent to `drum_storage`. If there was a discrepancy or if not all drums can be stored the clerk is notified.

STEP A3: OTHER USE CASES FOR ECO SYSTEM

Collection



Status



SYSTEM INTERACTION GRAPHS

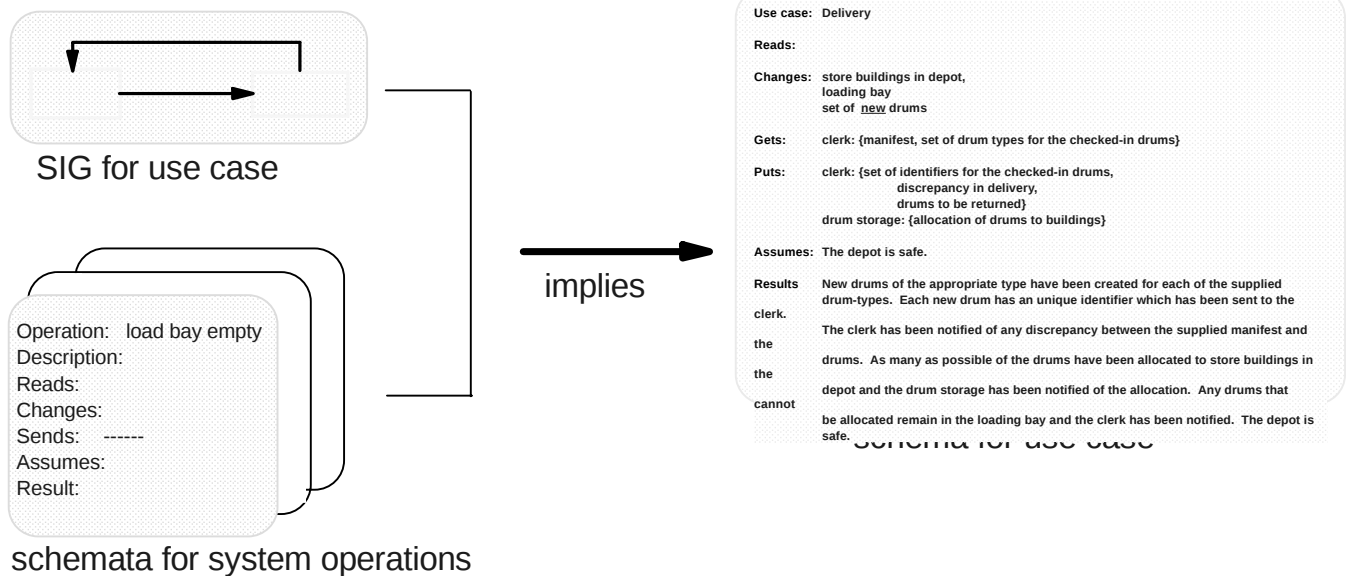
- SIGs define how use case classes are mapped to system operations - scenarios just show instances
 - provides traceability from informal transactions to system operations and events.
 - necessary for showing that a use case goal is satisfied by the chosen system operations.
 - provides a notation for generating test-cases.
- Used instead of lifecycles
- Optional - currently being evaluated for their usefulness.

STEP A4: SCHEMATA FOR SYSTEM OPERATIONS

- The schema for each system operation is completed as in standard Fusion

STEP A5: CHECKING CONSISTENCY BETWEEN USE CASES AND SYSTEM OPERATIONS

- For each use case



- Establish the validity by inspections

STEP A5: OVERLAPPING USE CASES

- If two use cases can overlap then it is also necessary to check for interference.
 - i.e., check that the execution of the system operations belonging to one use case cannot violate the **assumes** clause (pre-condition) of any system operation in any other use case.

Reference: Specification and Design of (Parallel) Programs
C.B. Jones, Proceedings of IFIP, 1983
pages 321-332

REST OF ANALYSIS PHASE

- Use schemata for use cases and system operations to
 - construct object model and system object model.
 - check consistency and completeness of analysis models as usual.

CONCLUSION

- Fusion and Requirements:
 - based on Objectory use cases
 - defines an external view of the system
- Provides an external view of the system as a composition of use cases.
- Provides a seamless front-end for Fusion analysis phase
- Traceable from requirements through to code

