Chapter 21

- **Formal Modeling and Verification**

Slide Set to accompany

Software Engineering: A Practitioner's Approach, 7/e

by Roger S. Pressman

Mô hình hình thức và kiểm chứng

- *Kỹ nghệ phần mềm phòng sạch (Cleanroom software engineering)* và *phương pháp hình thức (formal methods)*
  - Cả hai yêu cầu một cách tiếp cận đặc tả chuyên dụng và mỗi chúng áp dụng cho một phương pháp kiểm chứng duy nhất.
  - Cả hai khá chính xác và đều không được sử dụng rộng rãi bởi cộng đồng công nghệ phần mềm.
- Nếu bạn phải xây dựng phần mềm "chống đạn", các phương pháp này có thể vô cùng hữu ích.

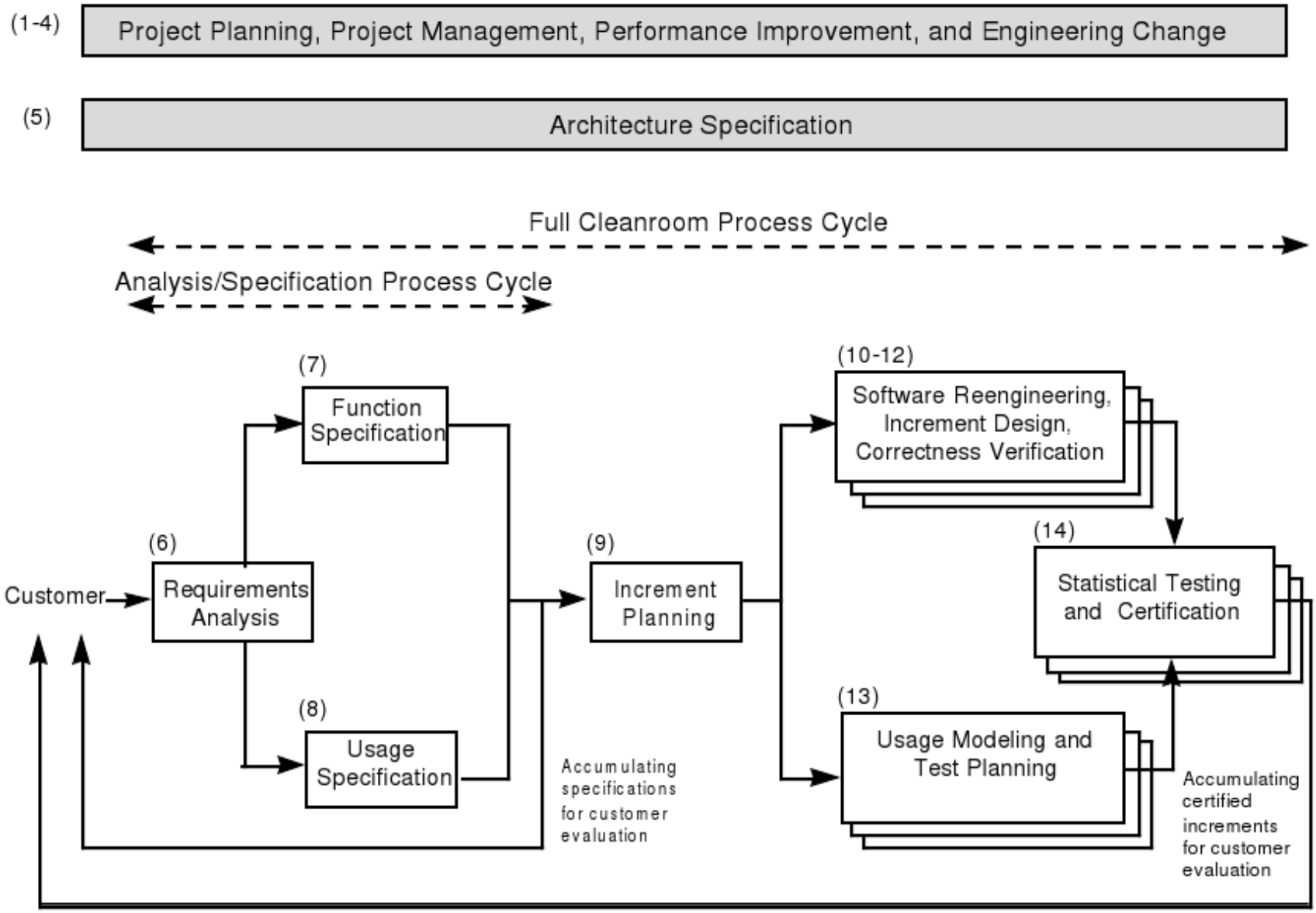
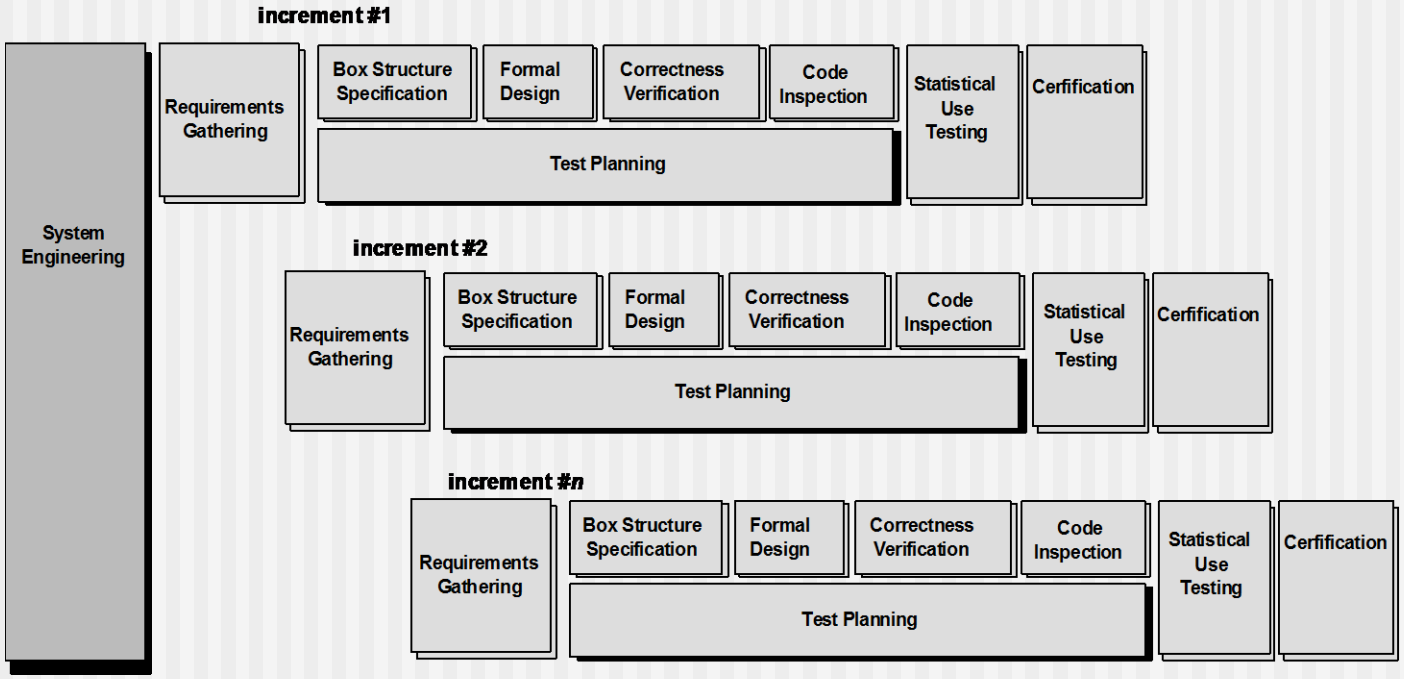


Figure 1. Cleanroom Process Flow

The Cleanroom Process Model



Chiến lược phòng sạch I

Lập kế hoạch tăng dần— theo chiến lược tăng dần

Tập hợp yêu cầu—định nghĩa một mô tả của các yêu cầu mức khách hang (cho mỗi sự tăng lên)

Đặc tả cấu trúc hộp—mô tả đặc điểm chức năng

Thiết kế hình thức—các đặc tả (gọi là “hộp đen”) được sàng lọc một cách lặp lại (với một sự tiến dần) để trở nên tương tự với thiết kể về kiến trúc và thủ tục (gọi là “state boxes” và “clear boxes,” tương ứng).

Kiểm chứng tính đúng đắn—kiểm chứng bắt đầu với cấu trúc hộp mức cao nhất (đặc tả) và chuyển đến chi tiết thiết kế và viết mã sử dụng một tập các câu hỏi tính đúng đắn (“correctness questions.”). Nếu điều này không chứng minh được các đặc tả là đúng, nhiều phương phá hình thức (toán học) cho kiểm chứng được sử dụng.

Sinh mã, kiểm tra và xác minh—các đặc tả cấu trúc hộp thẻ, biểu diễn trong một ngôn ngữ chuyên dụng, được chuyển giao cho một ngôn ngữ lập trình dành riêng.

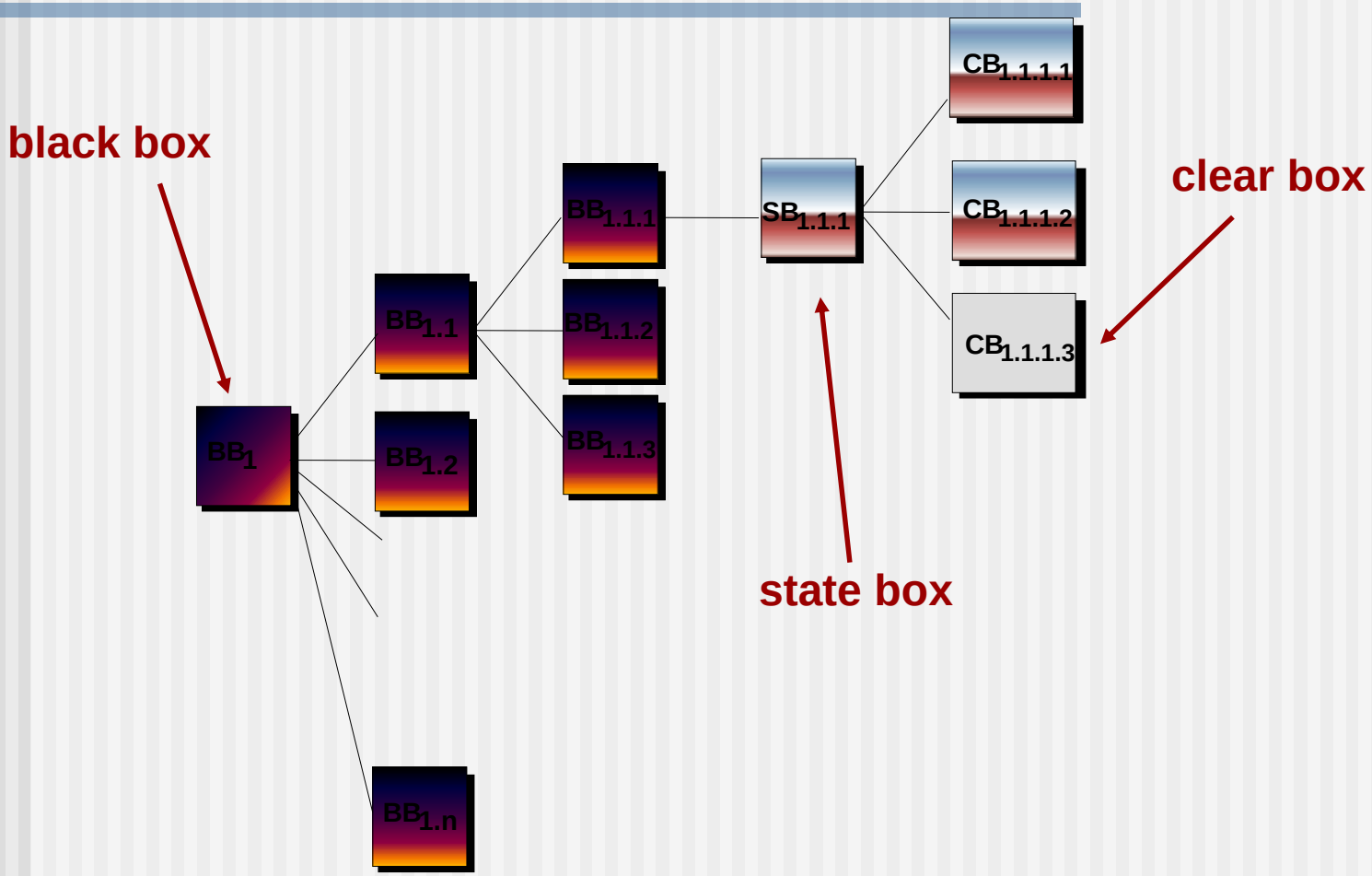
Chiến lược phòng sạch-II

Lập kế hoạch kiểm thử thống kê—một bộ các trường hợp kiểm thử mà sự thực thi của sự phân bố thống kê của cách sử dụng được lập kế hoạch và thiết kế

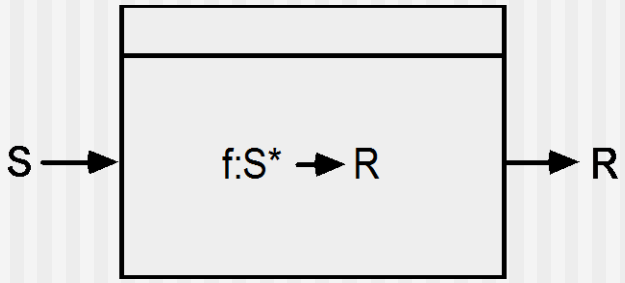
Kiểm thử cách sử dụng thống kê—thực thi một chuỗi các kiểm tra nhận được từ một mẫu thống kê (phân bố xác suất được trình bày ở trên) của tất cả các sự thực hiện có thể bởi tất cả người dung từ một tập hợp đích

Chứng nhận— một lần chứng nhận, sự xét duyệt và kiểm thử cách sử dụng được hoàn thành (và tất cả lỗi đều đúng) lượng tăng lên được chứng nhận là sẵn sàng cho sự tích hợp.

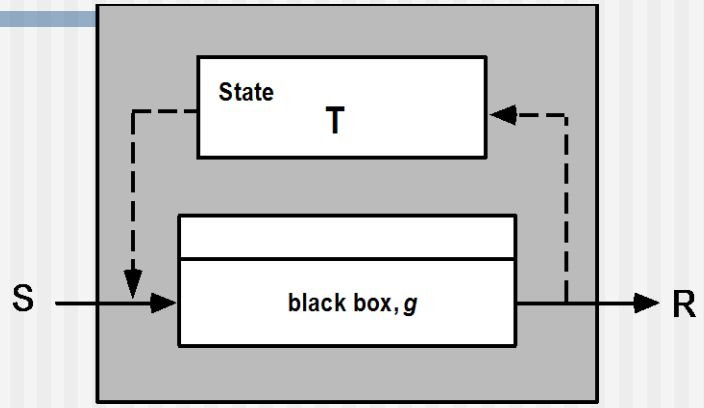
Đặc tả cấu trúc hộp



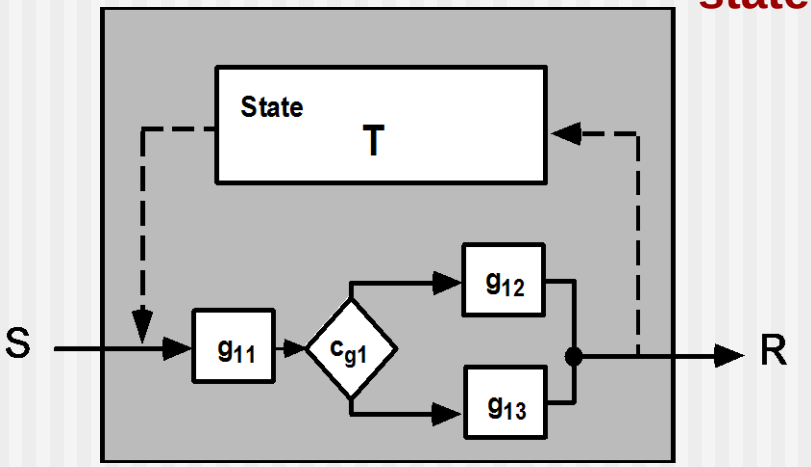
Cấu trúc hộp



black box



state box



clear box

Chi tiết hóa và kiểm chứng thiết kế

Nếu một hàm f được mở rộng thành một dãy g và h, điều kiện chính xác cho tất cả các đầu vào tới f là:

- **g có theo sau bởi h thực hiện f?**

Khi một hàm f được chi tiết hóa thành một điều kiện (if-then-else), điều kiện chính xác cho tất cả các đầu vào tới f là:

- **mỗi lần điều kiện <c> đúng thì g có thực hiện f và mỗi lần <c> sai thì h có thực hiện f?**

Khi hàm f được chi tiết hóa thành một vòng lặp, điều kiện chính xác cho mọi đầu vào tới f là:

- **Sự kết thúc có được đảm bảo?**
- **mỗi lần <c> đúng g có theo sau bởi f thực hiện f, và mỗi lần <c> sai, sự bỏ qua vòng lặp vẫn thực hiện f không?**

Lợi ích của kiểm chứng thiết kế

- Giảm sự kiểm chứng một quá trình giới hạn.
- Để cho đội phòng sạch xác minh mọi dòng của thiết kế và mã.
- Nó cho kết quả trong một mức sai sót xấp xỉ 0.
- Nó mở rộng quy mô.
- Nó sinh mã tốt hơn kiểm thử đơn vị.

Kiểm thử phòng sạch

- Kiểm thử cách dùng thống kê
  - Kiểm thử các các sử dụng thực tế của chương trình.
- Xác định một “phân phối xác suất sử dụng”
  - Phân tích đặc tả để nhận dạng một tập các sự kích thích.
  - Sự kích thích dẫn đến phần mềm thay đổi cách xử lý.
  - Tạo các kịch bản sử dụng.
  - Gán xác suất sử dụng với mỗi kích thích.
  - Kiểm thử trường hợp được sinh ra cho mỗi kích thích theo phần bố xác suất sử dụng.

Chứng nhận

1. Kịch bản sử dụng phải được tạo ra.
2. Một profile các sử dụng được chỉ rõ.
3. Các test case được sinh ra từ profile.
4. Kiểm thử được thực hiện và dữ liệu lỗi được ghi chép và phân tích.
5. Sự tin cậy được tính toán và chứng nhận.

Mô hình chứng nhận

Mô hình lấy mẫu. Kiểm thử phần mềm thực hiện m trường hợp ngẫu nhiên và được kiểm chúng nếu không lỗi hoặc một số lượng nhất định lỗi xảy ra. Giá trị của m thu được chính xác để chắc chắn rằng đạt được sự tin cậy yêu cầu.

Mô hình thành phần. Một hệ thống bao gồm n thành phần để kiểm chứng. Mô hình thành phần cho phép người phân tích xác định xác suất thành phần i sẽ lỗi trước khi hoàn thành.

Mô hình chứng nhận. Sự tin cậy toàn thể của hệ thống được đặt kế hoạch và chứng nhận.

Phương pháp hình thức

- *"Phương pháp hình thức sử dụng trong phát triển hệ thống máy tính dựa trên kỹ thuật mô tả thuộc tính hệ thống. Phương pháp hình thức như vậy cung cấp framework với những gì con người có thể định rõ, phát triển, và kiểm chứng hệ thống với một cách làm có hệ thống hơn là cách đặc biệt."*

  The Encyclopedia of Software Engineering **[Mar01]**

- Vấn đề với đặc tả quy ước:
  - Sự mâu thuẫn
  - Sự tối nghĩa
  - Tính chất mập mờ
  - Tình trạng chưa hoàn thành
  - Các mức hỗn hợp trừu tượng

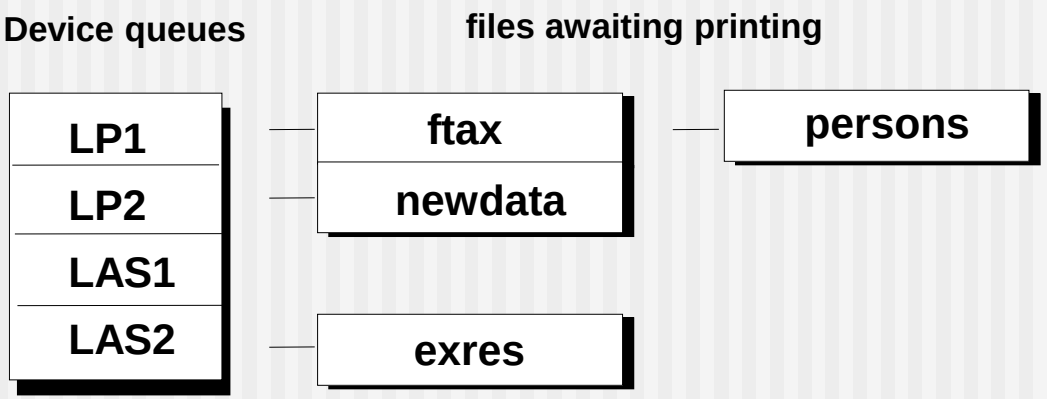
Đặc tả hình thức

- Các thuộc tính mong muốn—tính nhất quán, tính hoàn thành, và thiếu sự tối nghĩa—là các mục tiêu của phương pháp đặc tả.
- Cú pháp hình thức của ngôn ngữa đặc tả cho phép yêu cầu hoặc thiết kế được diễn dịch theo một cách duy nhất, loại trừ sự không rõ ràng thường xuyên xảy ra khi một ngôn ngữ tự nhiên (ví dụ tiếng Anh) hoặc một ký hiệu đồ họa phải được diễn dịch.
  - Phương tiện mô tả của lý thuyết tập hợp và ký hiệu logic cho phép trình bày sự việc rõ (yêu cầu).
- Tính nhất quán được đảm bảo bằng cách chứng minh toán học rằng các sự việc khởi tạo có thể được ánh xạ hình thức (sử dụng luật suy diễn) vào các câu lệnh sau này với sự đặc tả.

Khái niệm phương pháp hình thức

- *Lượng bất biến dữ liệu*—Một điều kiện đúng khắp quá trình thực thi của hệ thống bao gồm một tập dữ liệu
- *Trạng thái*
  - Rất nhiều ngôn ngữ hình thức, như OCL (phần 28.5), sử dụng ký hiệu trạng thái như thảo luận trong chương 7 và 8, đó là, một hệ thống có thể trong một của một ài trạng thái, mỗi chúng đại diện cho một chế độ quan sát được theo bề ngoài của xử lý.
  - Ngôn ngử Z (phần 28.6) định nghĩa một trạng thái là dữ liệu lưu trữ mà môt hệ thống truy cập và sửa đổi.
- *Vận hành*—Một hành động xảy ra trong một hệ thống và đọc ghi dữ liệu tới một trạng thái
  - *Điều kiện tiên quyết* xác định các trường hợp mà một thao tác cụ thể có hiệu lực
  - *Điều kiện sau* xác định chuyện gì xảy ra khi một thao tác hoàng thành công việc

Ví dụ—Print Spooler



Limits

LP1 -> 750
LP2 -> 500
LAS1 -> 300
LAS2 -> 200

Size

newdata -> 450
ftax -> 650
exres -> 50
persons -> 700

Trạng thái và bất biến dữ liệu

Trạng thái của spooler được biểu diễn bởi 4 thành phần: hàng đợi, thiết bị ra, giới hạn và kích.

Bất biến dữ liệu có 5 thành phần:

- Mỗi thiết bị ra được liên kết với một giới hạn trên của dòng in
- Mỗi thiết bị ra có thể được liên kết với một hàng đợi không rỗng file chờ in
- Mỗi file được liên kết với một kích thước
- Mỗi hàng đợi liên kết với một thiết bị ra bao gồm các file có kích thước nhỏ hơn giới hạn trên của thiết bị ra
- Sẽ không có nhiều hơn *MaxDevs* thiết bị ra quản trị bởi spooler

Thao tác

- Một thao tác thêm một thiết bị ra cho spooler cùng với giới hạn in liên kết của nó .
- Một thao tác xóa file từ hàng đợi liên kết với một thiết bị ra riêng biệt.
- Một thao tác thêm một file vào hàng đợi liên kết với một thiết bị ra riêng biệt.
- Một thao tác thay đổi giới hạn trên của dòng in cho một thiết bị ra riêng biệt.
- Một thao tác di chuyển file từ hàng đợi liên kết vói một thiết thiết bị ra tới một hàng đợi khác liên kết với một thiết bị ra thứ hai.

Điều kiện trước và sau

Cho thao tác đầu tiên (thêm một thiết bị mới vào spooler cùng với giới hạn in liên kết) :

Điều kiện trước: Tên thiết bị ra chưa tồn tại và hiện có ít hơn MaxDevs thiết bị ra đã biết bởi spooler

Điều kiện sau: tên của thiết bị mới đươc thêm vào tập các tên thiết bị, một một cổng vào được hình thành cho thiết bị với không file nào được liên kết với hàng đợi của nó, và thiết bị được liên kết với giới hạn in của nó.

Khái niệm toán học*

- Tập hợp và đặc tả tập hợp
- Phép toán tập hợp
- Phép toán logic
  - e.g., $\forall$ i, j: $\mathbb{N}$ • i > j i^2 => j^2
  - Với mọi cặp giá trị trong tập các số tự nhiên, nếu I lớn hon j thì i^2 lớn hơn j^2.
- Chuỗi

*A discussion of sets and constructive specification (slides 20 - 24) is no longer included within SEPA, 7/e, but is included here for those who are unfamiliar with the basic concepts.

Tập hợp và đặc tả tập xây dựng

- Tập hợp là một tập các đối tượng hoặc phần tử và được sử dụng như là viên đá đặt nền của phương pháp hình thức.
  - Liệt kê
    - {C++, Pascal, Ada, COBOL, Java}
    - #{C++, Pascal, Ada, COBOL, Java} implies *cardinality* = 5
  - Đăc tả tập xây dựng thích hợp cho kiểu liệt kê vì nó cho phép định nghĩa ngắn gọn những tập lớn.
    - $\{x, y : N \mid x + y = 10 \ \ (x, y^2)\}$

Phép toán tập hợp

- Tập chuyên dụng tượng trưng được sử dụng để biểu diễn tập hợp và phép toán tập hợp.
  - Ví dụ
    - Phép toán P được sử dụng để chỉ ra tính liên thuộc của tập hợp. Ví dụ biểu thức
      - x P X
    - Phép toán , , và # nhận tập hợp là toán hạng.
      - A , B
    - Có giá trị là *true* nếu phân tử của tập A nằm trong tập B và có giá trị false trong trường hợp khác.
  - Phép hợp, <, nhận 2 tập hợp và cho một chứa tất cả các thành phần trong tập đó loại trừ lặp lại.
    - {File1, File2, Tax, Compiler} < {NewTax, D2, D3, File2} là tập
    - {Filel, File2, Tax, Compiler, NewTax, D2, D3}

Phép toán logic

- Một thành phần quan trọng khác của phương pháp hình thức là logic: đại số boolean.
  - Ví dụ:
    - V or
    - ¬ not
    - => implies
- Định lượng toàn bộ là một cách để phát biểu về phần tử của tập hợp đúng cho mọi thành viên của tập hợp. Định lượng toàn bộ sử dụng ký tự, . Ví dụ
    - $i, j : N \; i > j => i^2 > j^2$
- Phát biểu rằng với mọi cặp giá trị trong tập hợp các số tự nhiên, nếu I lớn hơn j thì i^2 lớn hơn j^2.

Chuỗi

- Chuỗi được chỉ định sử dụng dấu ngoặc nhọn. Ví dụ chuỗi đi trước sẽ được viết là
  - k Jones, Wilson, Shapiro, Estavezl
- Ghép nối, X, là một toán tử hai ngôi tạo một chuỗi bằng cách thêm toán hạng thứ 2 và cuối toán hạng thứ nhất. Ví dụ,
  - k 2, 3, 34, 1l X k12, 33, 34, 200 l = k 2, 3, 34, 1, 12, 33, 34, 200 l
- Toán hạng khác được áp dụng cho chuỗi là *head, tail, front,* and *last*.
  - *head* k 2, 3, 34, 1, 99, 101 l = 2
  - *tail* k 2, 3, 34, 1, 99, 101 l = 73, 34, 1,99, 1018
  - *last* k 2, 3, 34, 1, 99, 101 l = 101
  - *front* k 2, 3, 34, 1, 99, 101 l = 72, 3, 34, 1, 998

Đặc tả hình thức

- Xử lý khối
  - Xử lý khối duy trì một kho dự trữ của các khối chưa sử dụng và cũng sẽ theo dõi các khối đang sử dụng. Khi các block được giải phóng từ một file đã xóa chúng được them vào một hang đợi block chờ để được them vào kho chứa những block không được sử dụng.
    - Trạng thái

      used, free: P *BLOCKS*

      BlockQueue: *seq* P *BLOCKS*
    - Bất biến dữ liệu

      used > *free* = \

      used < *free* = *AllBlocks*

      i: dom *BlockQueue* *BlockQueue i* # *used*

      i, j : dom *BlockQueue* *i* ≠ *j* => *BlockQueue i* > *BlockQueue j* = \
    - Điều kiên trước

      #BlockQueue > 0
    - Điều kiện sau

      used' = *used* \ *head BlockQueue*

      free’ = *free* < *head BlockQueue*

      BlockQueue' = *tail BlockQueue*

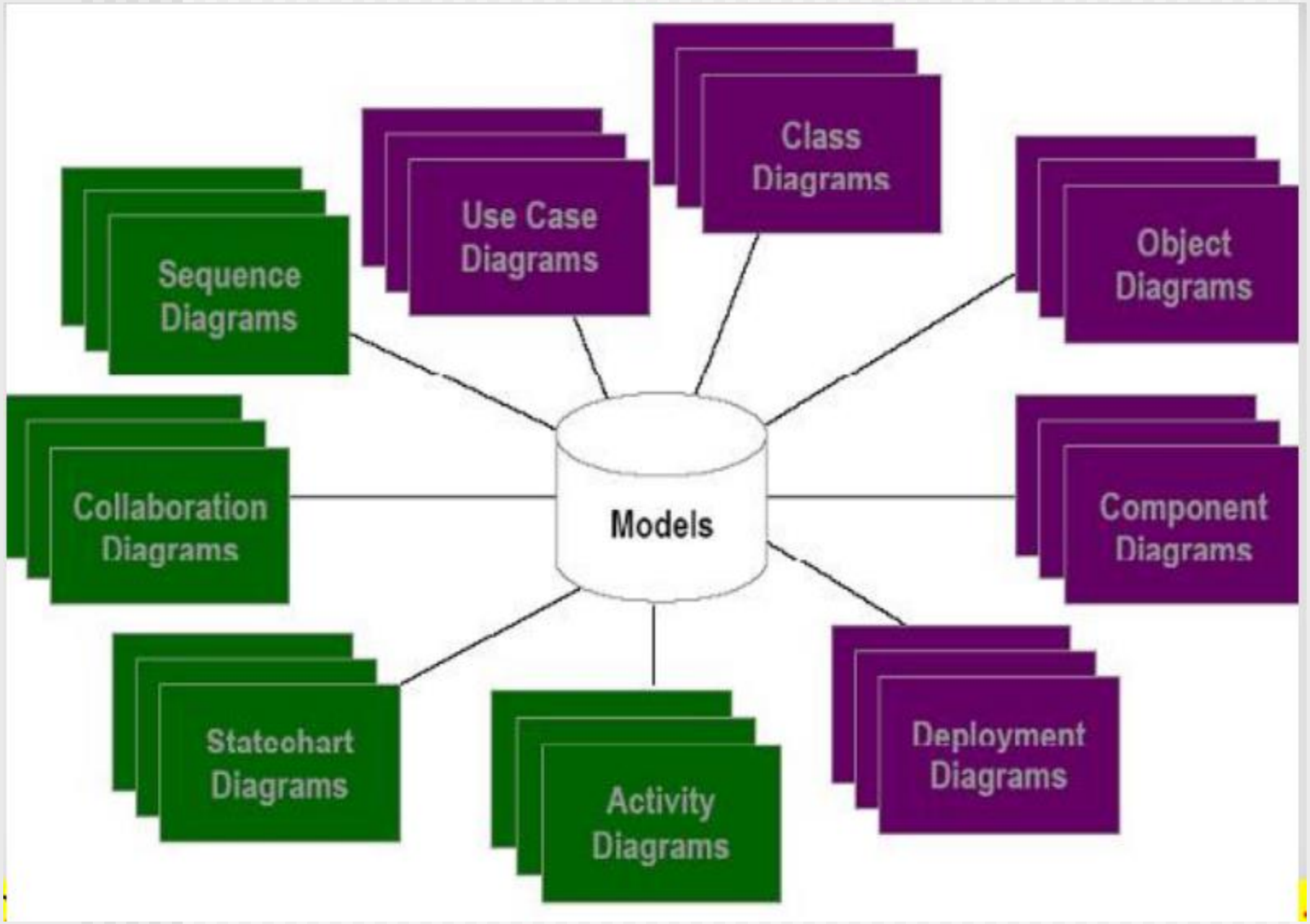
Ngôn ngữ đặc tả hình thức

- Một ngôn ngữ đặc tả hình thức được sử dụng bao gồm 3 thành phần chính:
  - Cú pháp định nghĩa ký hiệu đặc trưng mà sự đặc tả được biểu diễn bằng các ký hiệu đó
  - Ngữ nghĩa để giúp định nghĩa "tất cả các đối tượng" [WIN90] được sử dụng để mô tả hệ thống
  - Tập quan hệ định nghĩa các luật chỉ định đối tượng nào thảo mãn chính xác đặc tả
- Miền ngữ nghĩa của ngôn ngữ đặc tả hình thức thường dựa trên cú pháp nhận được từ ký hiệu lý thuyết tập hợp tiêu chuẩn và phép tính vị ngữ.
- Miền ngữ nghĩa của một ngôn ngữ đặc tả chỉ định cách mà ngôn ngữ đó biểu diễn yêu cầu của hệ thống.

Ngôn ngữ ràng buộc đối tượng (OCL)

- Ký hiệu hình thức được phát triên nên người dùng UML có thể thêm nhiều hơn tính chính xác vào đặc tả của họ
- Tất cả khả năng của toán học logic và rời rạc đều khả dụng trong ngôn ngữ này
- Tuy nhiên người thiết kế OCL quyết định chỉ ký từ ASCII (hơn là ký hiệu toán học tiêu chuẩn) nên sử dụng trong phát biểu OCL.

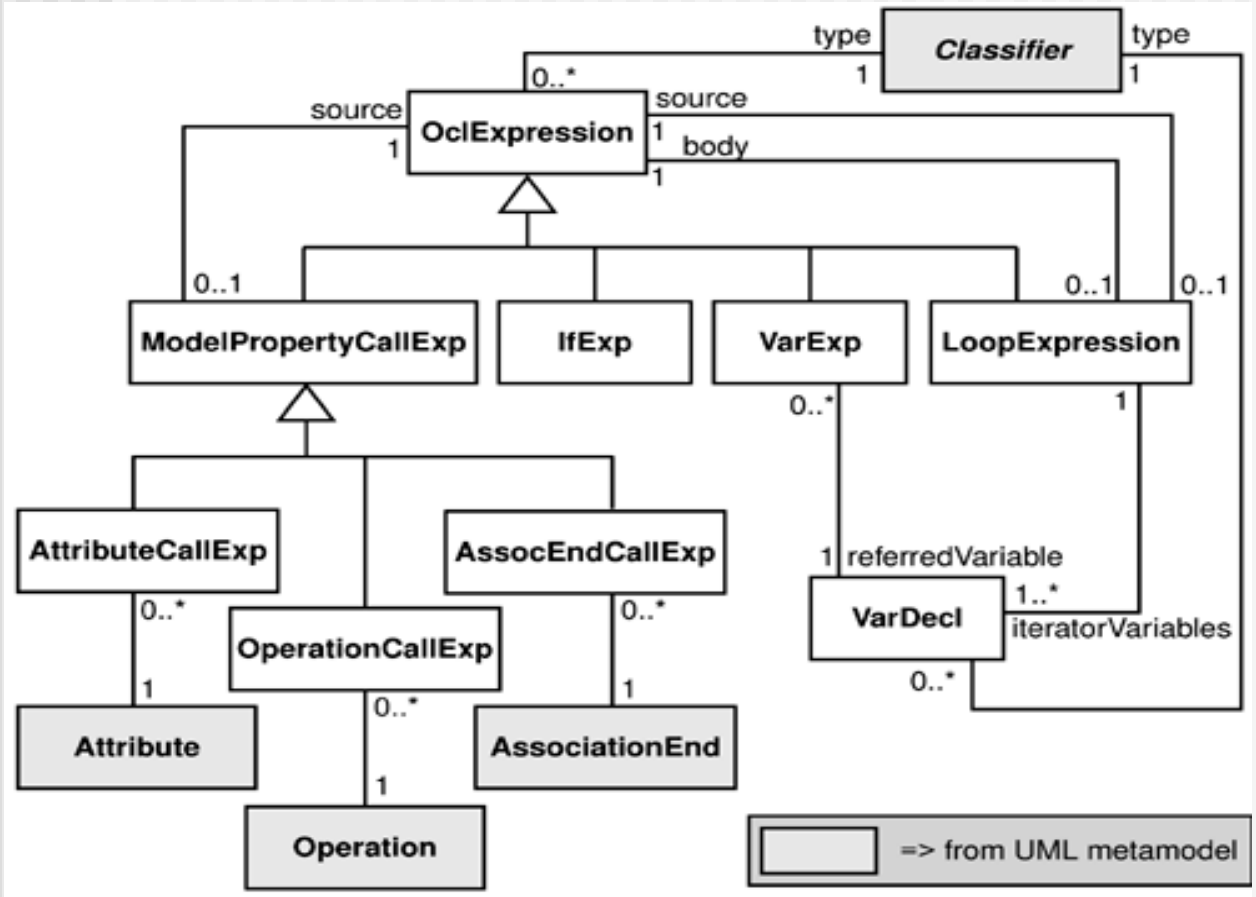
LƯỢC ĐỒ TRONG UML



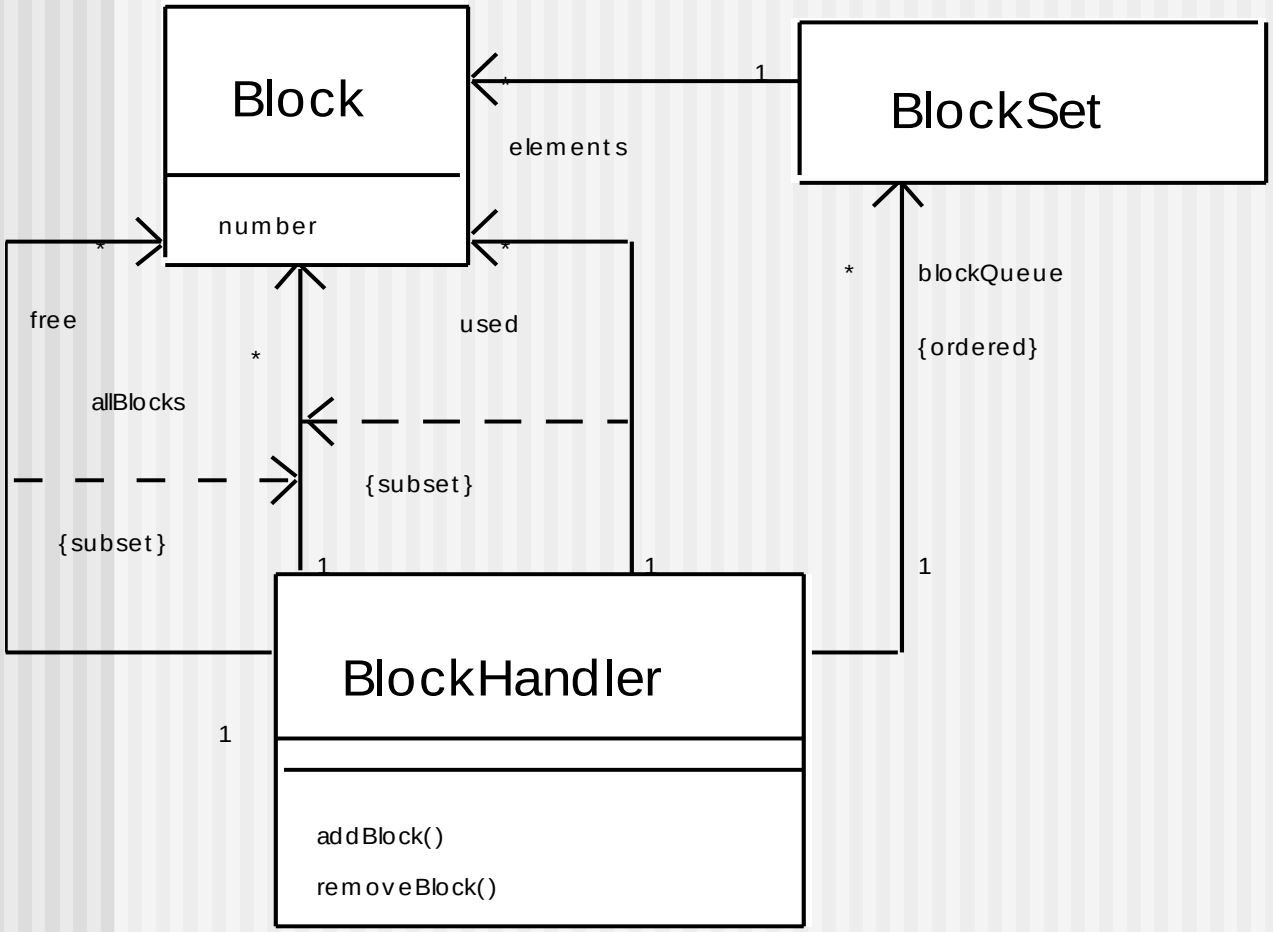
Tổng quan OCL

- Giống như ngôn ngữ lập trình hướng đối tượng, một biểu thức OCL bao gồm toán tử thực hiện trên đối tượng.
- Tuy nhiên kết quả của một biểu thức đầy đủ phải luôn là giá trị Boolean đúng hoặc sai.
- Các đối tượng có thể là trường hợp của lớp **Collection** OCL, lớp **Set** và **Sequence** là hai lớp con.
- Xem bảng 28.1 cho tóm lược ký hiệu OCL

The OCL and UML Metamodels



BlockHandler sử dụng UML



BlockHandler trong OCL

- Không khối nào được đánh dấu cả không sử dụng và sử dụng.
  - ***context*** *BlockHandler inv:*
  - *(self.used->intersection(self.free)) ->isEmpty()*
- Tất cả tập blocks giữ trong hang đợi sẽ là tập con của tập hợp các blocks đang được sử dụng.
  - ***context*** *BlockHandler inv:*
  - *blockQueue->forAll(aBlockSet | used->includesAll(aBlockSet))*
- Không phần tử nào của hang đợi sẽ chứa cùng một số block.
  - ***context*** *BlockHandler* ***inv****:*
  - *blockQueue->forAll(blockSet1, blockSet2 |*
  - *blockSet1 <> blockSet2 implies*
  - *blockSet1.elements.number->excludesAll(blockSet2.elements.number))*
  - The expression before *implies* is needed to ensure we ignore pairs where both elements are the same Block.
- Tập các block được sử dụng và block không được sử dụng sẽ là tập tổng cộng các block bổ sung file.
  - ***context*** *BlockHandler* ***inv****:*
  - *allBlocks = used->union(free)*
- Tập các block không sử dụng sẽ không có số block lặp lại.
  - ***context*** *BlockHandler* ***inv****:*
  - *free->isUnique(aBlock | aBlock.number)*
- Tập các block được sử dụng sẽ không có số block lặp lại.
  - ***context*** *BlockHandler* ***inv****:*
  - *used->isUnique(aBlock | aBlock.number)*

Ngôn ngữ Z

- Tổ chức vào trong giản đồ
  - Định nghĩa các biến
  - Thành lập mối quan hệ giữa các biến
  - Tương tự cho “module” trong ngôn ngữ tiêu chuẩn
- Ký hiệu được mô tả trong bảng 21.2

BlockHandler trong Z

Các ví dụ giản đồ sau mô tả trạng thái của xử lý khối và bất biến dữ liệu:

———BlockHandler———————————————

used, free : P *BLOCKS*

BlockQueue : seq P *BLOCKS*

———————————————————————————

used > *free* = \

used < *free* = *AllBlocks*

i: dom *BlockQueue* *BlockQueue i* # *used*

i, j : dom *BlockQueue* *i* ≠ *j* => *BlockQueue i* > *BlockQueue j* = \

———————————————————————————

See Section 21.7.2 for further expansion of the specification