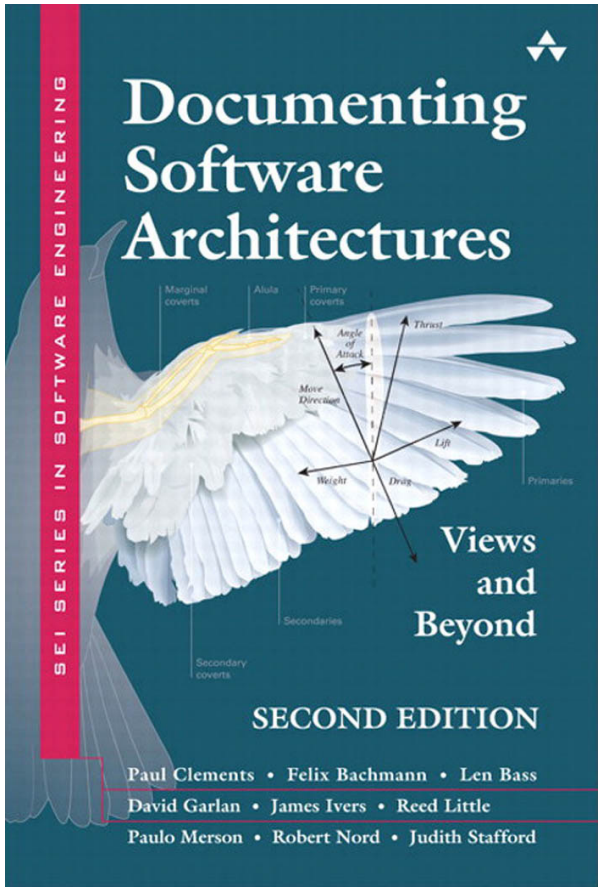
Проектирование вычислительных систем

(фрагмент курса: Архитектурное моделирование вычислительных систем)
Лекция 6: Архитектурные стили.
(Краткое изложение).

Пенской Александр Владимирович
aleksandr.penskoi@gmail.com
Университет ИТМО / 2018

Архитектурное документирование

- Задачи:
  - Architecture serves as a means of education.
  - Architecture serves as a primary vehicle for communication among stakeholders.
  - Architecture serves as the basis for system analysis and construction.
- Виды спецификаций:
  - Горизонтальная архитектура.
  - Вертикальная архитектура.

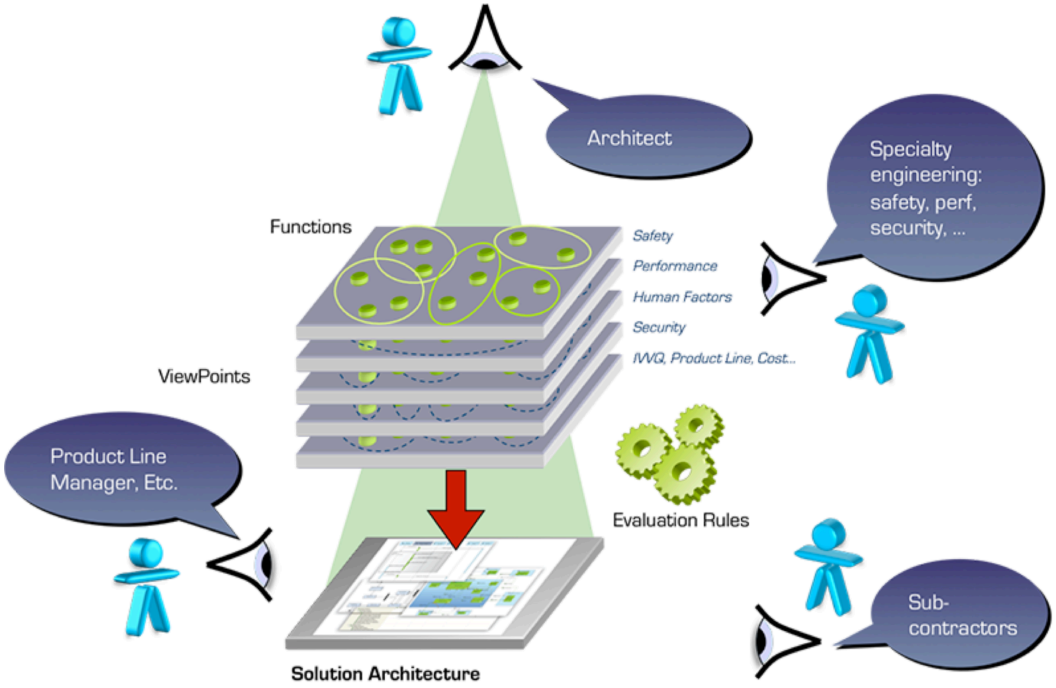


Архитектурный стиль

Основной инструмент проектирования уровневой организации — архитектурный стиль.

Архитектурный стиль (или архитектурный framework) — соглашения, принципы, практики для описания архитектуры, созданной для конкретной области применения и / или заинтересованных сторон (перевод ISO 42010).

Качество архитектурных решений выражается в совокупном снижении затрат заинтересованных сторон (stakeholders) на протяжении всего жизненного цикла системы.



Capella | PolarSys — Инструмент с открытым исподным кодом для встроенных систем.

(https://www.polarsys.org/proposals/capella)

“Вертикальная” и “горизонтальная” архитектура

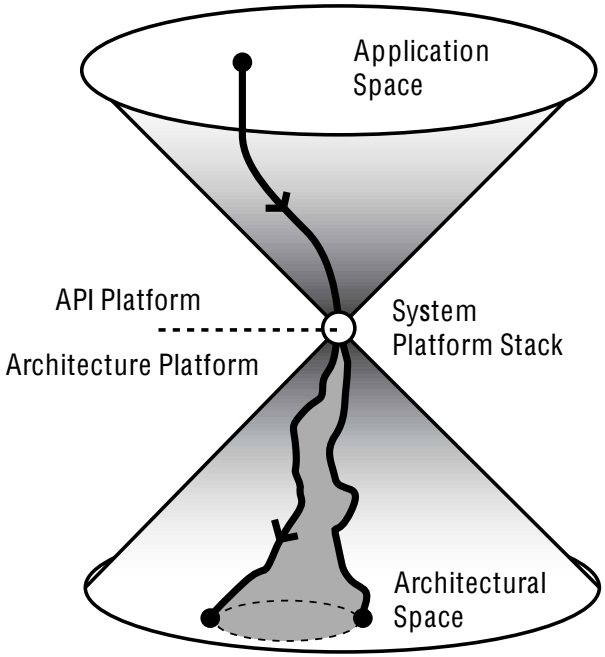
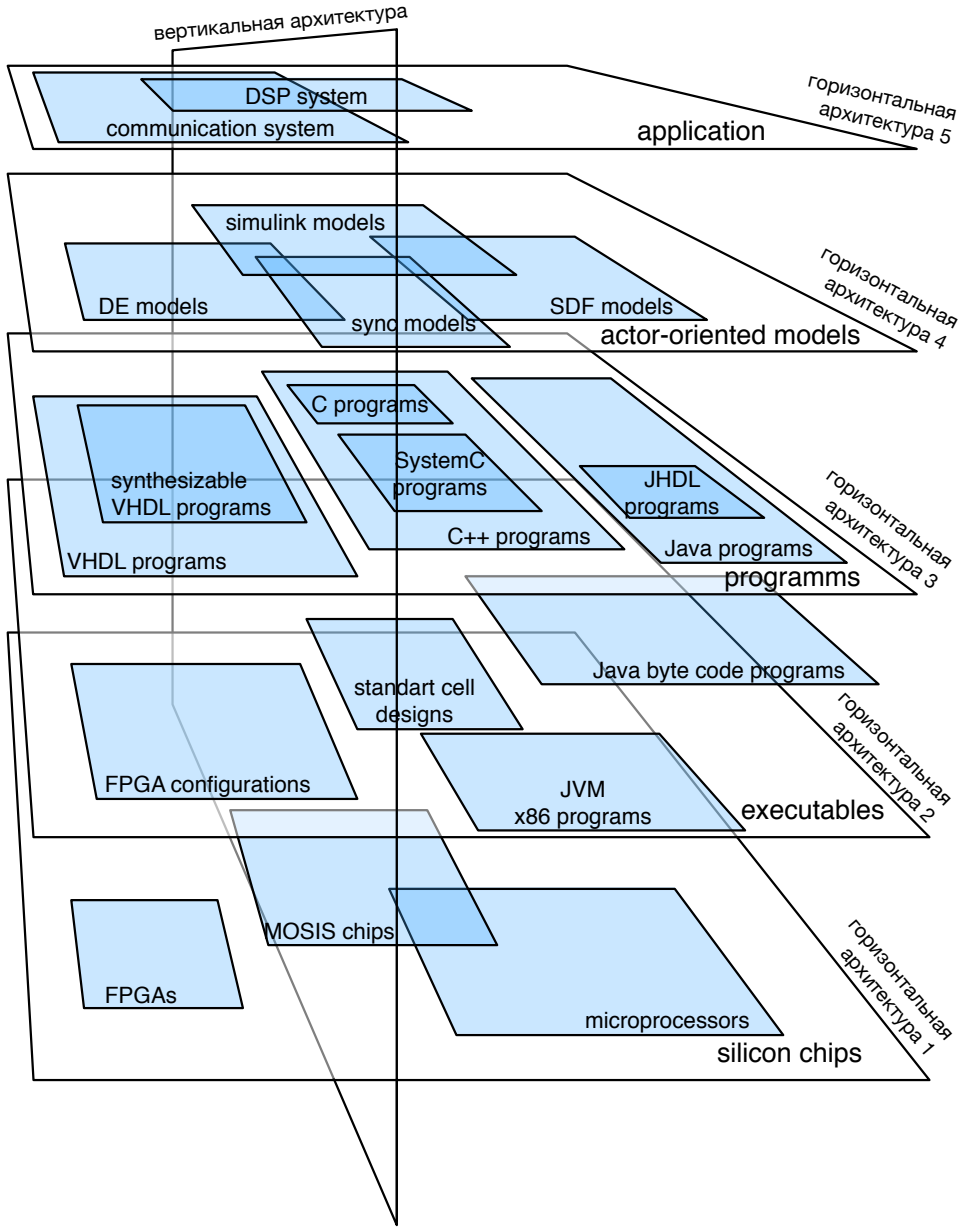


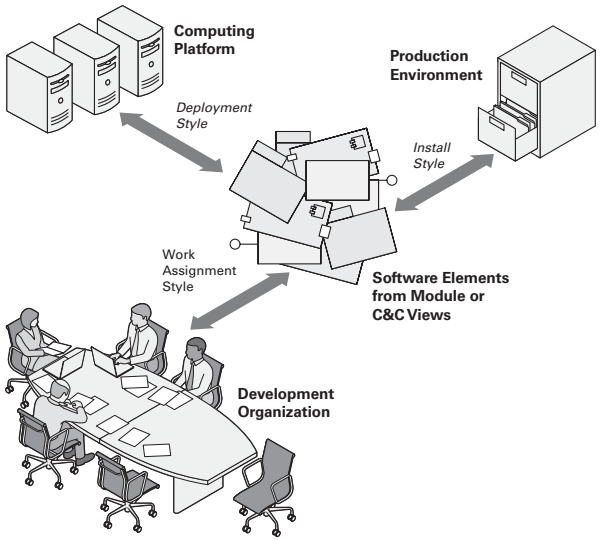
Fig. 1. The platform hourglass.

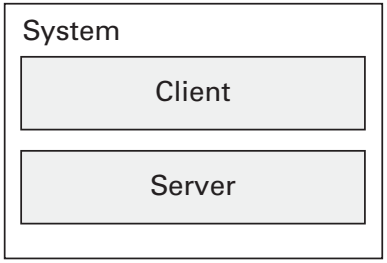
— E. A. Lee, S. Neuendorffer, and M. J. Wirthlin, “Actor-Oriented Design of Embedded Hardware and Software Systems,” J. Circuits, Syst. Comput., vol. 12, pp. 231–260, 2003.



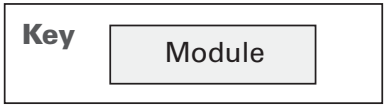
Стили архитектурного документирования

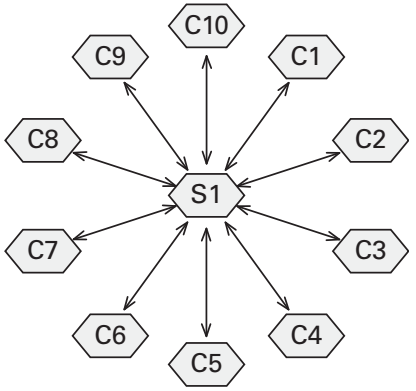
- Module Styles.
- Component-and-Connector Styles.
- Allocation Styles.



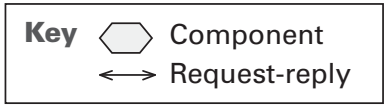


Decomposition view





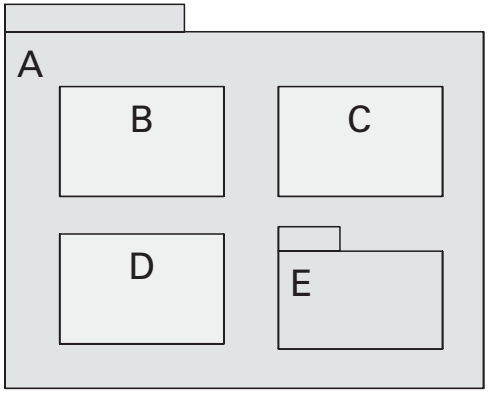
Client-server view



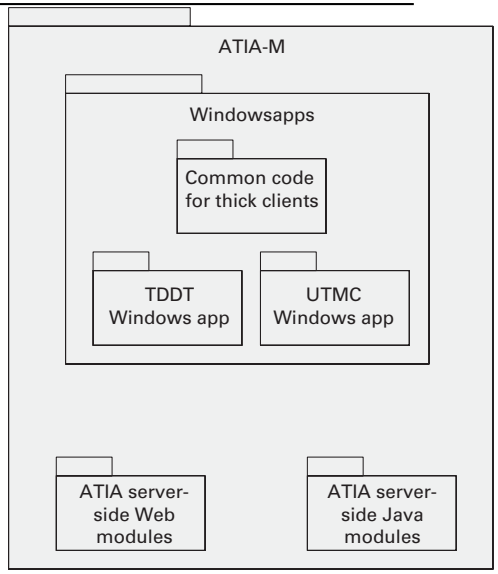
Module Styles

Elements	Modules, which are implementation units of software that provide a coherent set of responsibilities.
Relations	• *Is part of*, which defines a part/whole relationship between the submodule—the part—and the aggregate module—the whole. • *Depends on*, which defines a dependency relationship between two modules. Specific module styles elaborate what dependency is meant. • *Is a*, which defines a generalization/specialization relationship between a more specific module—the child—and a more general module—the parent.
Constraints	Different module views may impose specific topological constraints.
What It's For	• Providing a blueprint for construction of the code • Facilitating impact analysis • Planning incremental development • Supporting requirements traceability analysis • Explaining the functionality of the system and the structure of the code base • Supporting the definition of work assignments, implementation schedules, and budget information • Showing the structure of information to be persisted

Overview	The decomposition style is used for decomposing a system into units of implementation. A decomposition view describes the organization of the code as modules and submodules and shows how system responsibilities are partitioned across them.
Elements	*Module*
Relations	*Decomposition* relation, which is a form of the *is-part-of* relation. The documentation should specify the criteria used to define the decomposition.
Constraints	• No loops are allowed in the *decomposition* graph. • A module can have only one parent.
What It's For	• To reason about and communicate to newcomers the structure of software in digestible chunks • To provide input for work assignment • To reason about localization of changes



Decomposition Style



Overview The uses style shows how modules depend on each other; it is helpful for planning because it helps define subsets and increments of the system being developed.

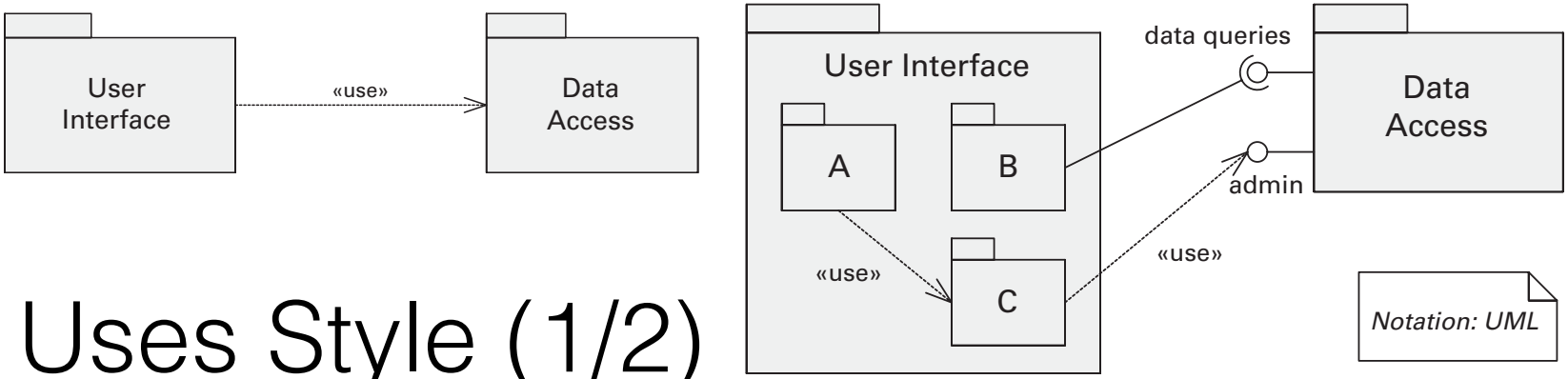
Elements *Module*

Relations The *uses* relation, which is a form of the *depends-on* relation. Module A *uses* module B if A *depends on* the presence of a correctly functioning B to satisfy its own requirements.

Constraints The uses style has no topological constraints. However, if *uses* relations present loops, broad fan-out, or long dependency chains, the ability of the architecture to be delivered in incremental subsets will be impaired.

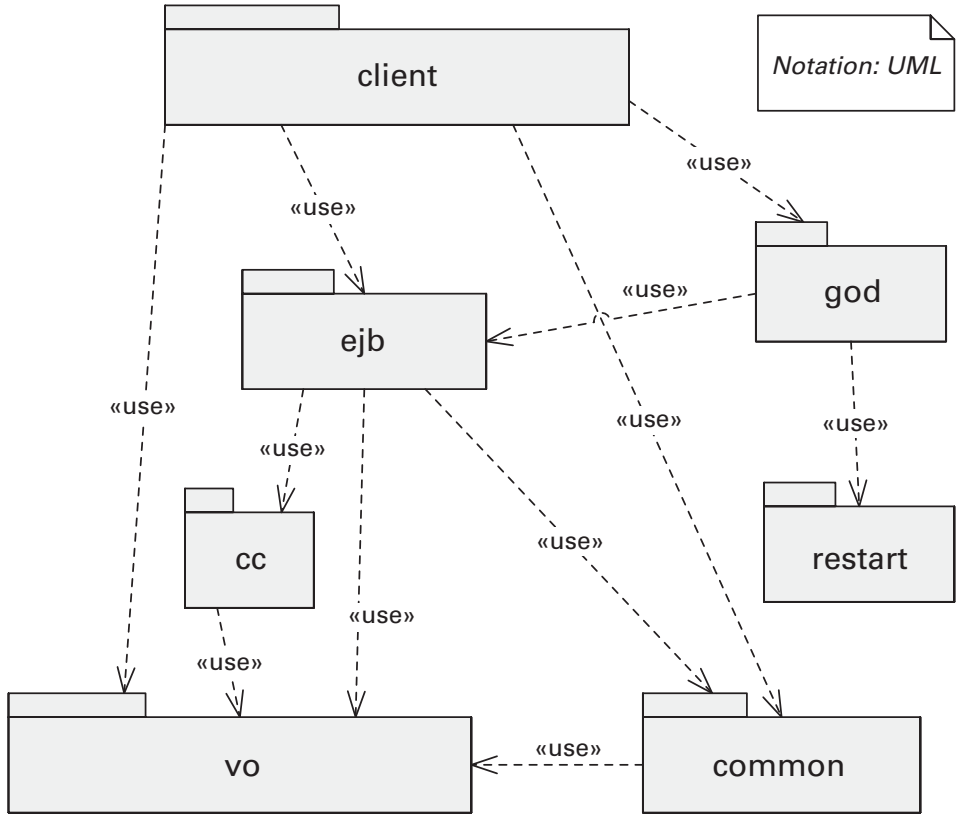
What It's For

- Planning incremental development and subsets
- Debugging and testing
- Gauging the effect of changes



Uses Style (1/2)

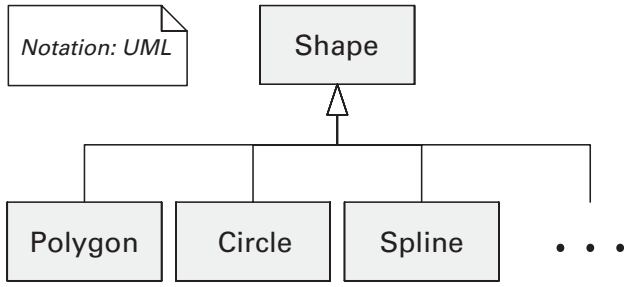
used module \ using module	client	ejb	cc	god	restart	common	vo
client	0	0	0	0	0	0	0
ejb	**1**	0	0	**1**	0	0	0
cc	0	**1**	0	0	0	0	0
god	**1**	0	0	0	0	0	0
restart	0	0	0	**1**	0	0	0
common	**1**	**1**	0	0	0	0	0
vo	**1**	**1**	**1**	0	0	**1**	0

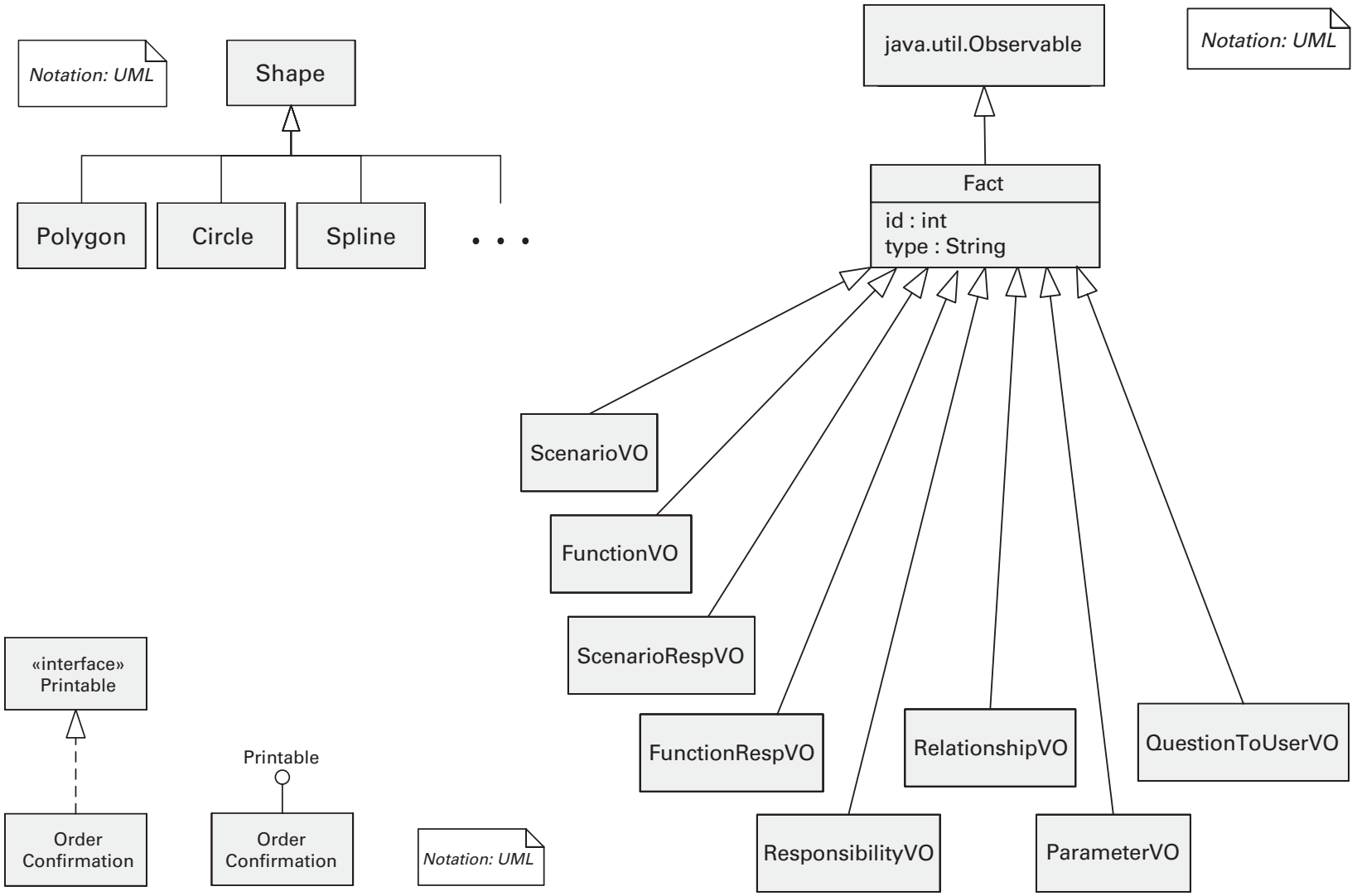


Uses Style (2/2)

Generalisation Style (1/2)

Overview	The generalization style employs the *is-a* relation to support extension and evolution of architectures and individual elements. Modules in this style are defined in such a way that they capture commonalities and variations.
Elements	*Module*. A module can have the "abstract" property to indicate it does not contain a complete implementation.
Relations	*Generalization*, which is a specialization of the *is-a* relation. The relation can be further specialized to indicate, for example, if it is class inheritance, interface inheritance, or interface realization.
Constraints	• A module can have multiple parents, although multiple inheritance is often considered a dangerous design approach. • Cycles in the *generalization* relation are not allowed; that is, a child module cannot be a generalization of one or more of its ancestor modules in a view.
What It's For	• Expressing inheritance in object-oriented designs • Incrementally describing evolution and extension • Capturing commonalities, with variations as children • Supporting reuse

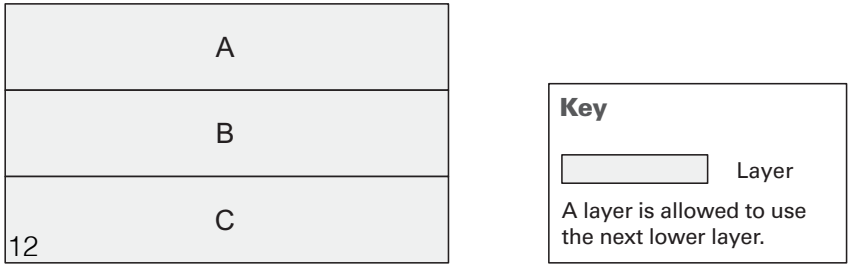




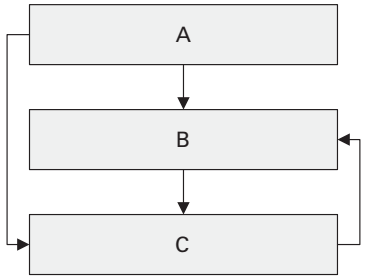
Generalisation Style (2/2)

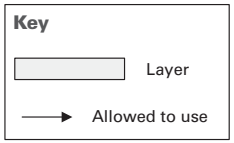
Layered Style (1/2)

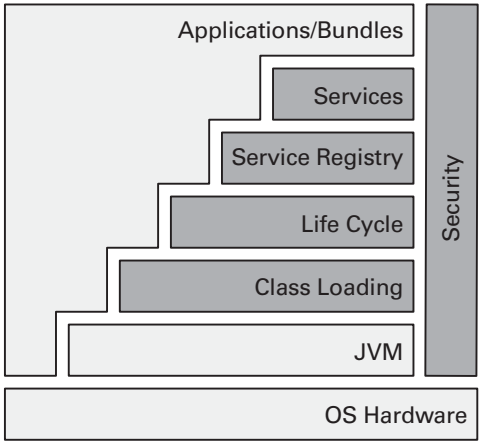
Overview	The layered style puts together layers (groupings of modules that offer a cohesive set of services) in a unidirectional *allowed-to-use* relation with each other.
Elements	*Layer*. The description of a layer should define what modules the layer contains.
Relations	*Allowed to use*, which is a specialization of the generic *depends-on* relation. The design should define the layer usage rules (for example, "A layer is allowed to use any lower layer.") and any allowable exceptions.
Constraints	• Every piece of software is allocated to exactly one layer. • There are at least two layers (typically three or more). • The *allowed-to-use* relations should not be circular (that is, a lower layer cannot use a layer above).
What It's For	• Promoting modifiability and portability • Managing complexity and facilitating the communication of the code structure to developers • Promoting reuse • Achieving separation of concerns



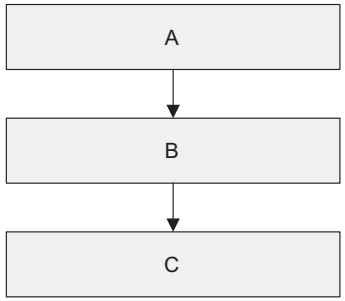
Layered Style (2/2)

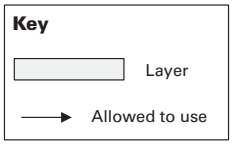


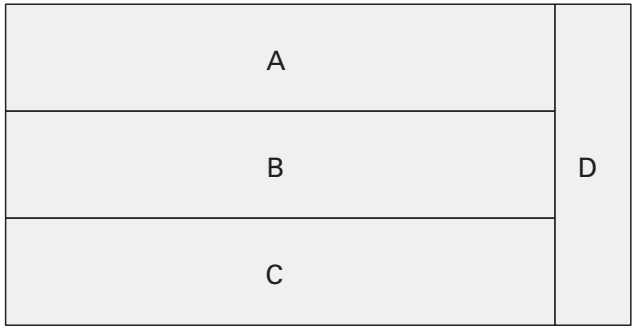


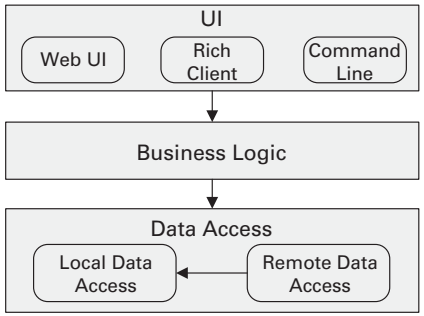


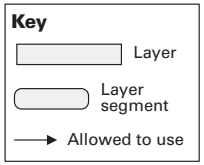
used layer \ using layer	UI presentation	controller	business logic	data access
UI presentation	0	0	0	0
controller	1	0	0	0
business logic	1	1	0	0
data access	1	1	1	0

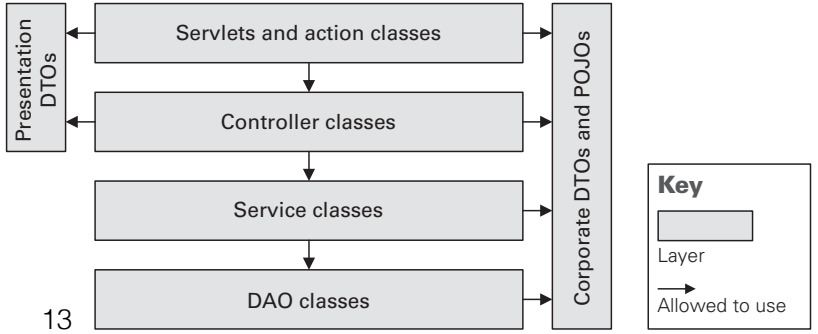




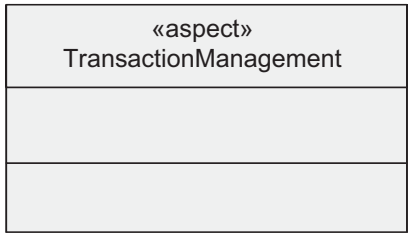


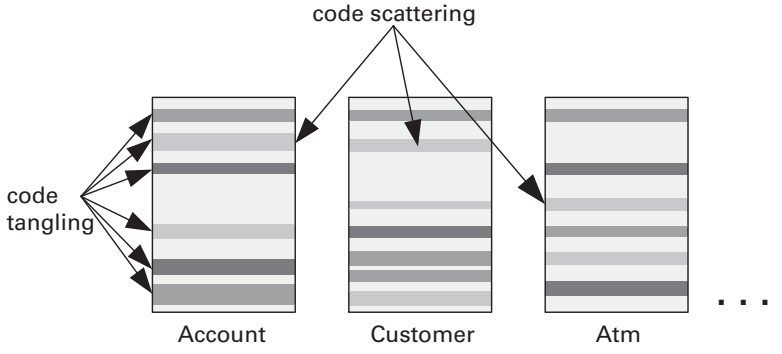






Overview	The aspects style shows aspect modules that implement crosscutting concerns and how they are bound to other modules in the system.
Elements	*Aspect*, which is a specialized module that contains the implementation of a crosscutting concern
Relations	*Crosscuts*, which binds an aspect module to a module that will be affected by the crosscutting logic of that aspect
Constraints	• An aspect can crosscut one or more regular modules as well as aspect modules. • An aspect that crosscuts itself may cause infinite recursion, depending on the implementation.
What It's For	• Modeling crosscutting concerns in object-oriented designs • Enhancing modifiability

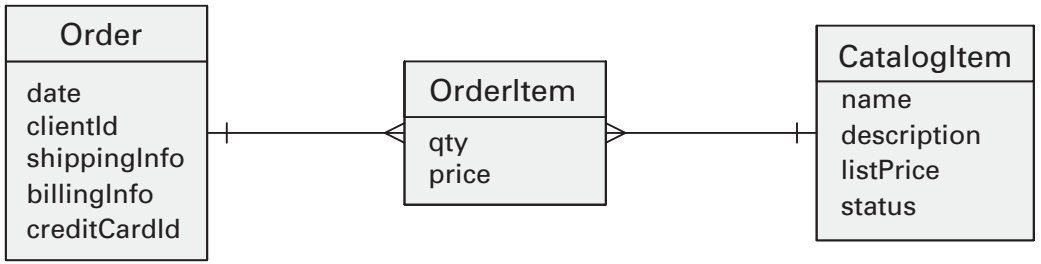


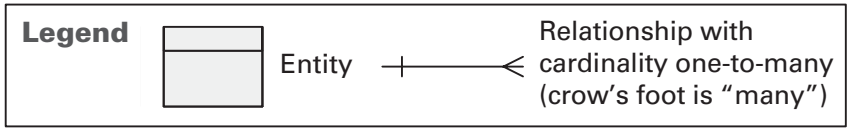


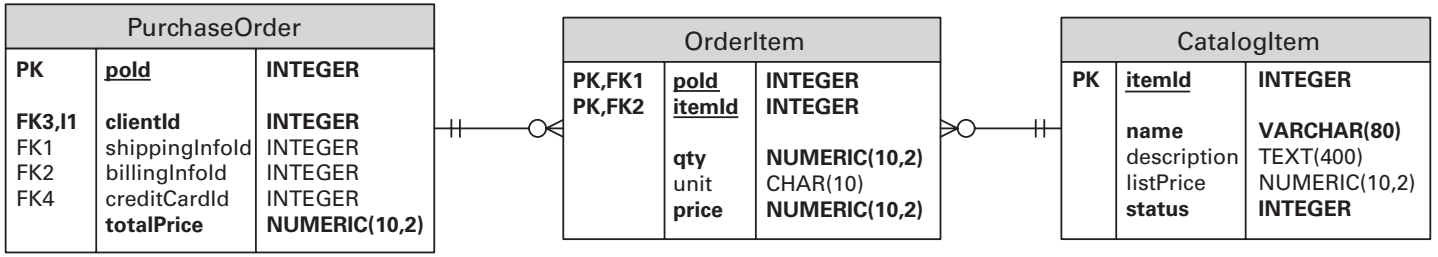
Aspect Style

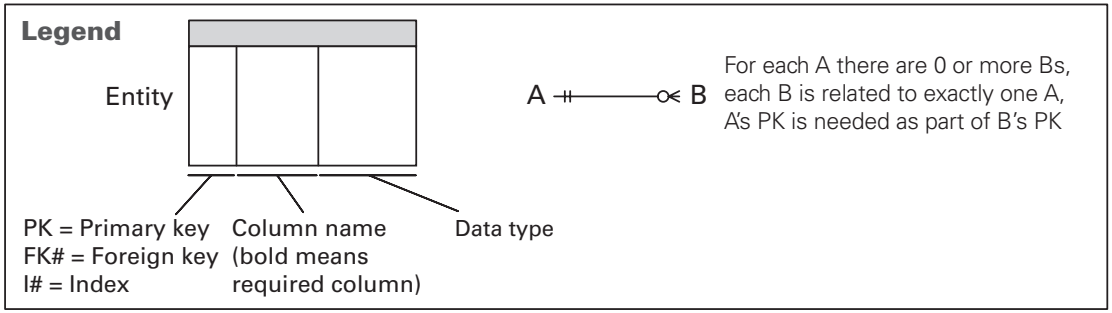
Overview	The data model describes the structure of the data entities and their relationships.
Elements	*Data entity*, which is an object that holds information that needs to be stored or somehow represented in the system. Properties include name, data attributes, primary key, and rules to grant users permission to access the entity.
Relations	• *One-to-one*, *one-to-many*, and *many-to-many* relationships, which are logical associations between data entities • *Generalization/specialization*, which indicate an *is-a* relation between entities • *Aggregation*, which turns a relationship into an aggregate entity
Constraints	Functional dependencies should be avoided.
What It's For	• Describing the structure of the data used in the system • Performing impact analysis of changes to the data model; extensibility analysis • Enforcing data quality by avoiding redundancy and inconsistency • Guiding implementation of modules that access the data

Data Model (1/2)









Data Model (2/2)

Component-and-Connector Styles

Elements

- *Components*: principal processing units and data stores. A component has a set of *ports* through which it interacts with other components (via connectors).
- *Connectors*: pathways of interaction between components. Connectors have a set of roles that indicate how components may use a connector in interactions.

Relations

- *Attachments*: component ports are associated with connector roles to yield a graph of components and connectors.
- *Interface delegation*: in some situations component ports are associated with one or more ports in an "internal" subarchitecture. Similarly for the roles of a connector.

Constraints

- Components can be attached only to connectors, not other components.
- Connectors can be attached only to components, not other connectors.
- Attachments can be made only between compatible ports and roles.
- Interface delegation can be defined only between two compatible ports (or two compatible roles).
- Connectors cannot appear in isolation; a connector must be attached to a component.

What C&C Views Are For

- Showing how the system works
- Guiding development by specifying the structure and behavior of runtime elements
- Helping architects and others to reason about runtime system quality attributes, such as performance, reliability, and availability

Data Flow Style (1/2)

Elements

- *Filter*, which is a component that transforms data read on its input ports to data written on its output ports. Filters typically execute concurrently and incrementally. Properties may specify processing rates, input/output data formats, and the transformation executed by the filter.
- *Pipe*, which is a connector that conveys data from a filter's output ports to another filter's input ports. A pipe has a single data-in and a single data-out role, preserves the sequence of data items, and does not alter the data passing through. Properties may specify buffer size, protocol of interaction, and data format that passes through a pipe.

Relations

The *attachment* relation associates filter output ports with data-in roles of a pipe, and filter input ports with data-out roles of pipes.

Computational Model

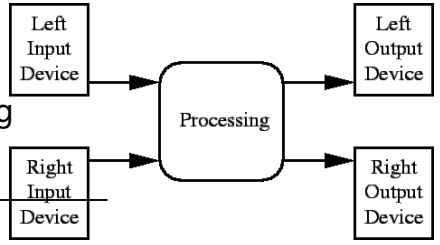
Data is transformed from a system's external inputs to its external outputs through a series of transformations performed by its filters.

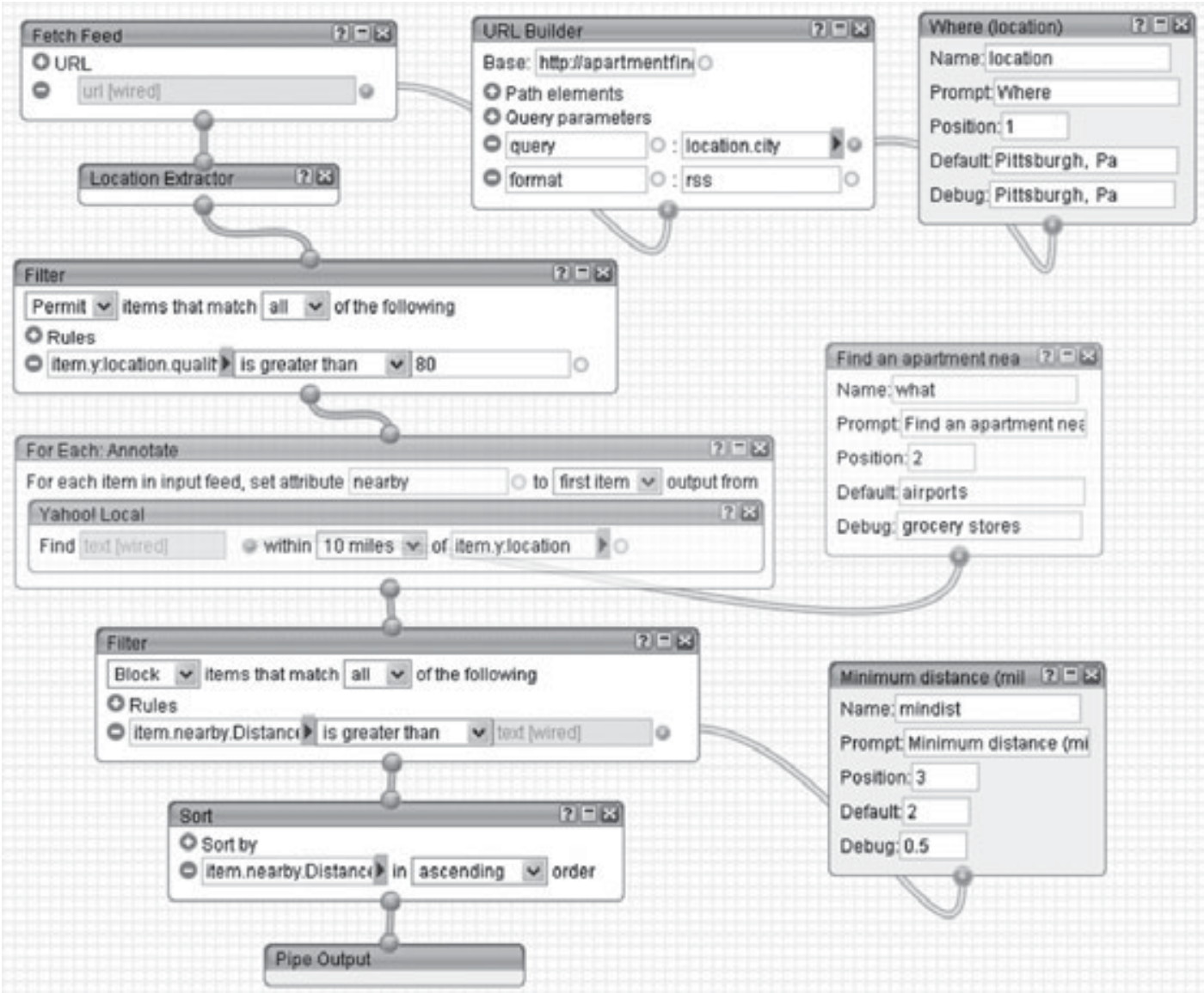
Constraints

- Pipes connect filter output ports to filter input ports.
- Connected filters must agree on the type of data being passed along the connecting pipe.
- Specializations of the style may restrict the association of components to an acyclic graph or a linear sequence—sometimes called a pipeline.
- Other specializations may prescribe that components have certain named ports, such as the *stdin*, *stdout*, and *stderr* ports of UNIX filters.

What It's For

- Improving reuse due to the independence of filters
- Improving throughput with parallelization of data processing
- Simplifying reasoning about overall behavior

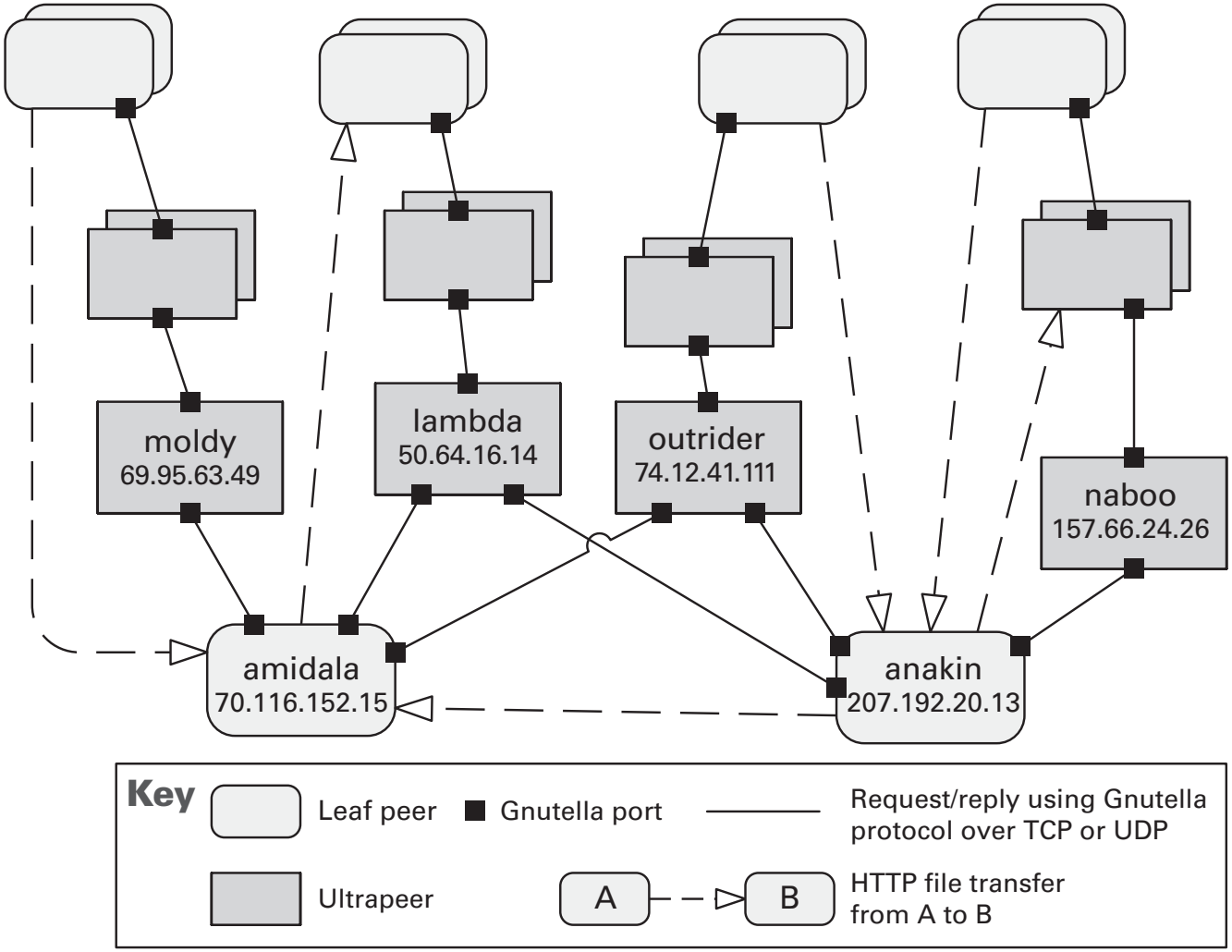




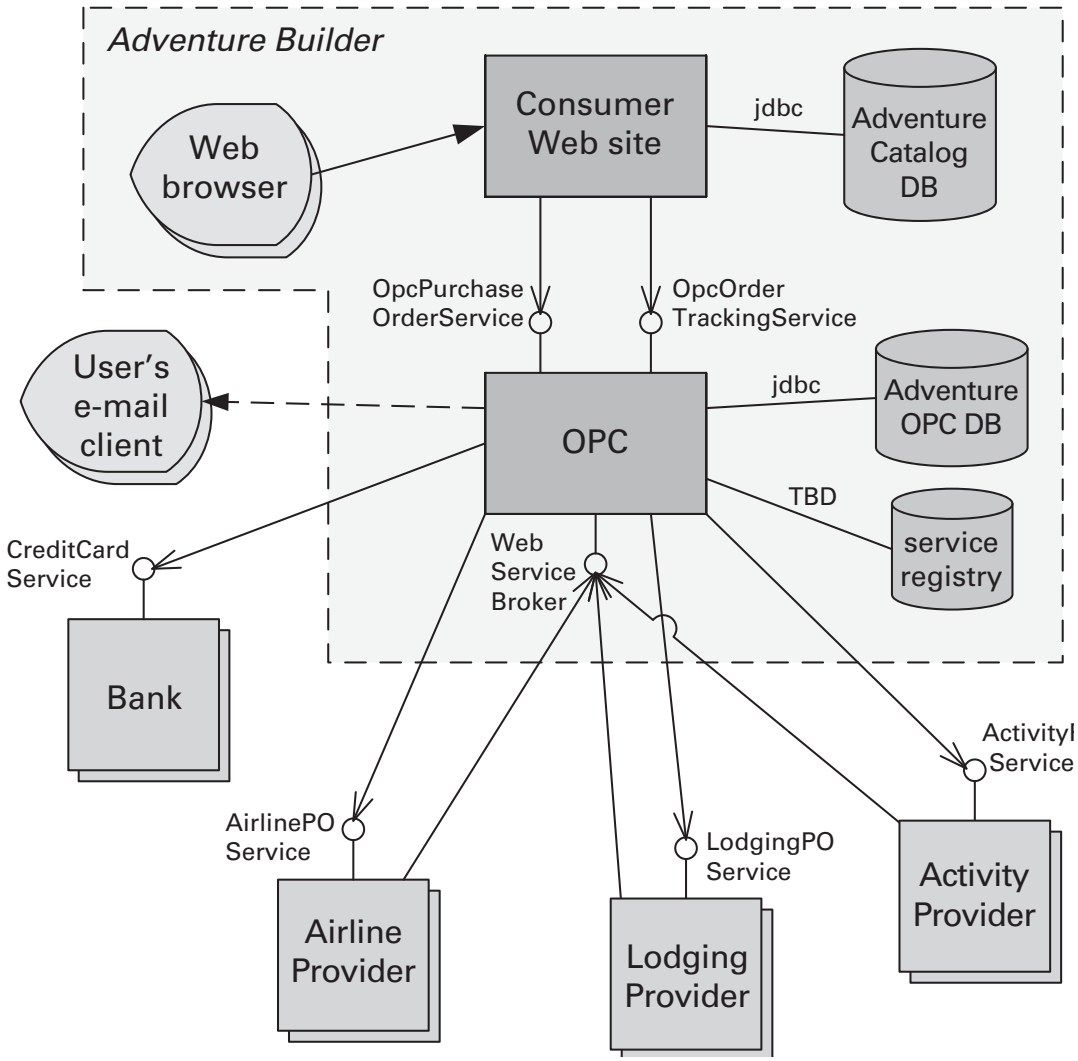
Data Flow Style (2/2)

Elements	• *Peer* component • *Call-return connector*, which is used to connect to the peer network, search for other peers, and invoke services from other peers
Relations	The *attachment* relation associates peers with call-return connectors.
Computational Model	*Computation* is achieved by cooperating peers that request services of one another.
Properties	Same as other C&C views, with an emphasis on protocols of interaction and performance-oriented properties. Attachments may change at runtime.
Constraints	• Restrictions may be placed on the number of allowable attachments to any given port, or role. • Special peer components can provide routing, indexing, and peer search capability. • Specializations may impose visibility restrictions on which components can know about other components.
What It's For	• Providing enhanced availability • Providing enhanced scalability • Enabling highly distributed systems, such as file sharing, instant messaging, and desktop grid computing

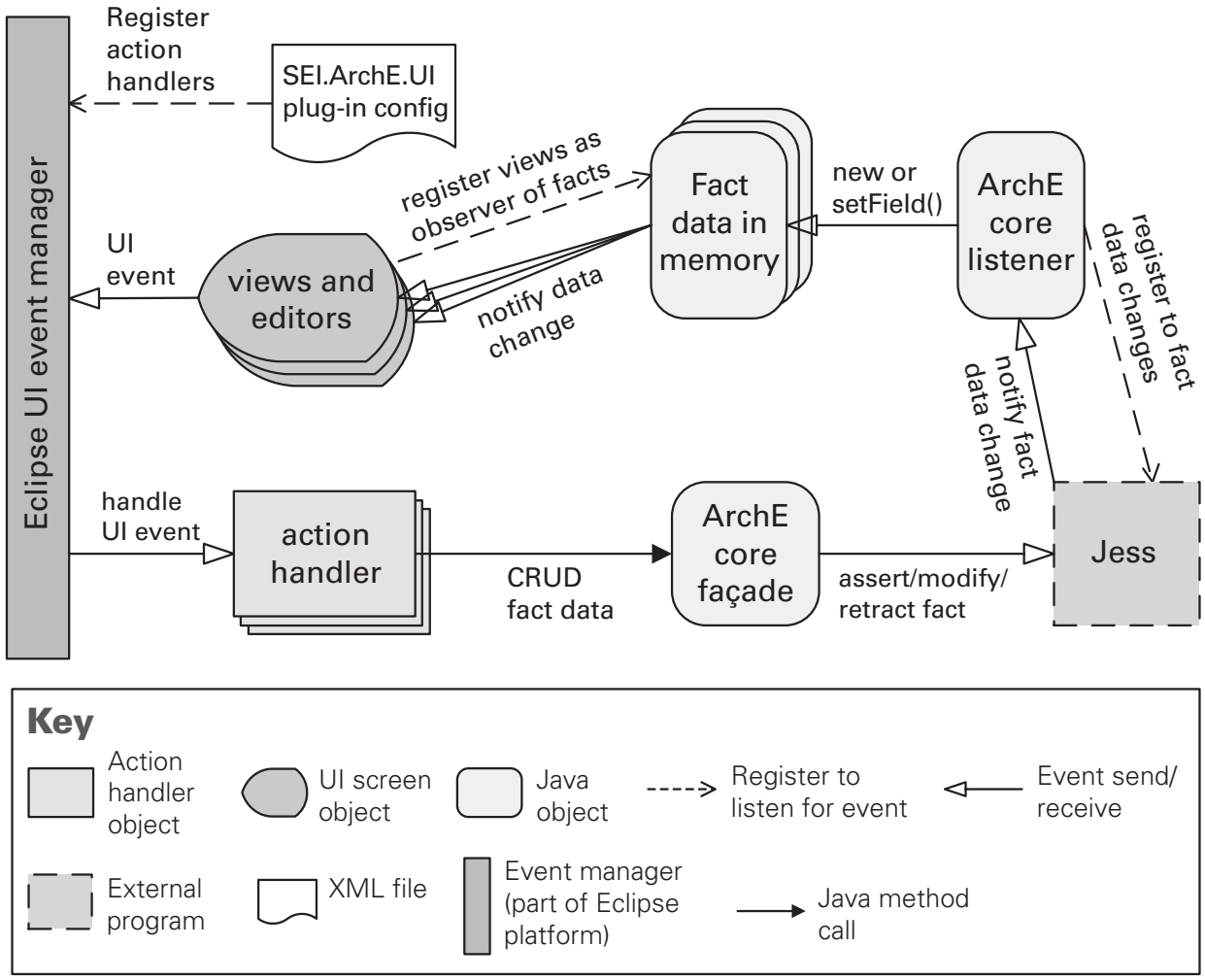
Call-Return Style
Peer-to-Peer Style (1/3)



Call-Return Style
Peer-to-Peer Style (2/3)



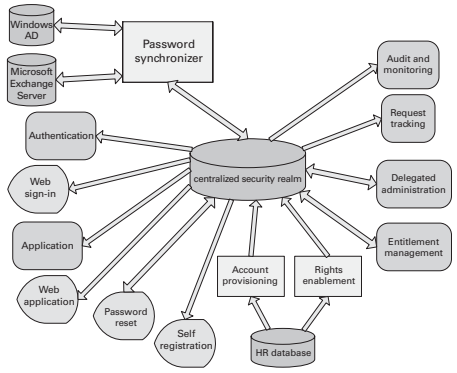
Call-Return Style
Service-Oriented Style (3/3)

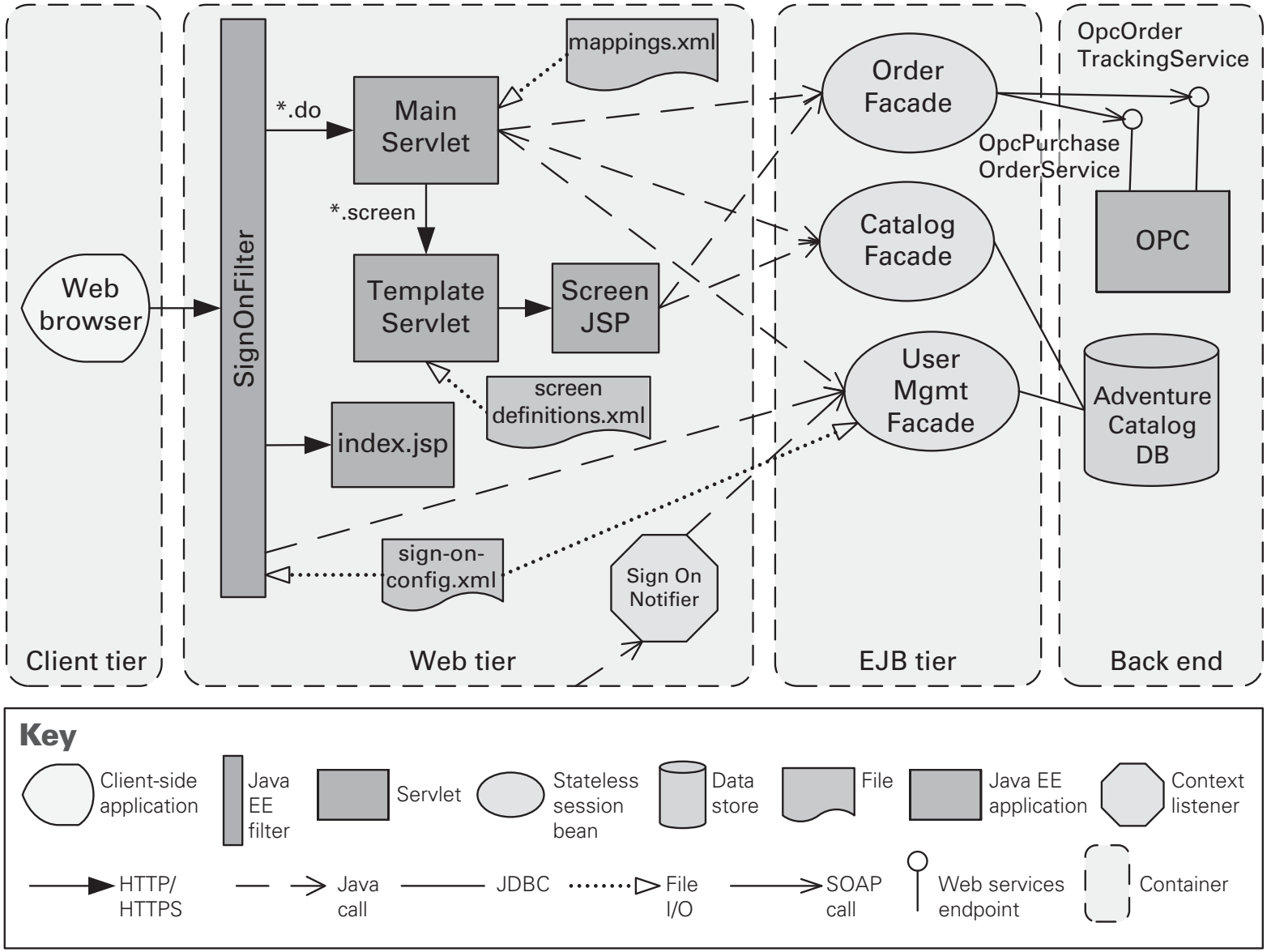


Event-Based Style
Publish-Subscribe Style

Elements	• *Repository component*. Properties include types of data stored, data performance-oriented properties, data distribution, number of accessors permitted. • *Data accessor component*. • *Data reading and writing connector*. An important property is whether the connector is transactional or not.
Relations	*Attachment* relation determines which data accessors are connected to which data repositories.
Computational Model	Communication between data accessors is mediated by a shared-data store. Control may be initiated by the data accessors or the data store. Data is made persistent by the data store.
Constraints	Data accessors interact with the data store(s).
What It's For	• Allowing multiple components to access persistent data • Providing enhanced modifiability by decoupling data producers from data consumers

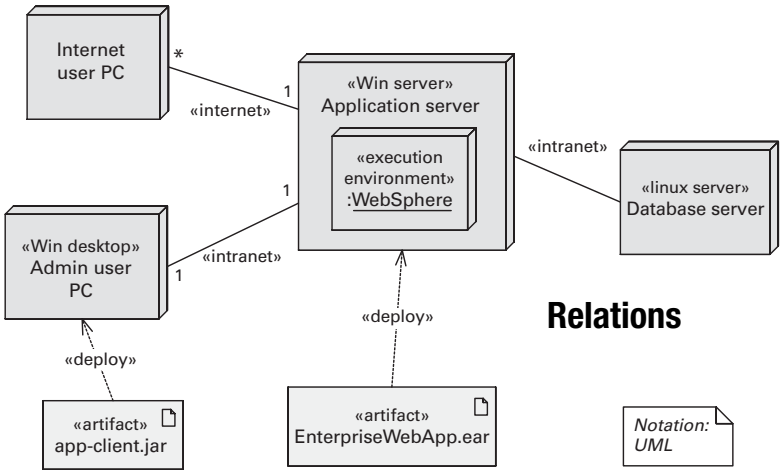
Repository Style




Client tier
Web tier
EJB tier
Back end
Web browser
SignOnFilter
*.do
Main Servlet
mappings.xml
*.screen
Template Servlet
Screen JSP
screen definitions.xml
index.jsp
sign-on-config.xml
Sign On Notifier
Order Facade
Catalog Facade
User Mgmt Facade
OpcOrder TrackingService
OpcPurchase OrderService
OPC
Adventure Catalog DB
Key
Client-side application
Java EE filter
Servlet
Stateless session bean
Data store
File
Java EE application
Context listener
HTTP/ HTTPS
Java call
JDBC
File I/O
SOAP call
Web services endpoint
Container

Allocation Styles

Overview	Allocation styles describe the mapping between the software architecture and its environment.
Elements	*Software element* and *environmental element*. A software element has properties that are *required* of the environment. An environmental element has properties that are *provided* to the software.
Relations	*Allocated-to*. A software element is mapped (allocated to) an environmental element. Properties are dependent on the particular style.
Constraints	Varies by style



Overview

The deployment style describes the mapping of components and connectors in the software architecture to the hardware of the computing platform.

Elements

- *Software element: elements from a C&C view.* Useful properties to document include the significant features required from hardware, such as processing, memory, capacity requirements, and fault tolerance.
- *Environmental elements: hardware of the computing platform—processor, memory, disk, network (such as router, bandwidth, firewall, bridge), and so on.* Useful properties of an environmental element are the significant hardware aspects that influence the allocation decision.

Relations

- *Allocated-to*. Physical units on which the software elements reside during execution. Properties include whether the allocation can change at execution time or not.
- *Migrates-to*, *copy-migrates-to*, and/or *execution-migrates-to* if the allocation is dynamic. Properties include the trigger that causes the migration.

Constraints

The allocation topology is unrestricted. However, the required properties of the software must be satisfied by the provided properties of the hardware.

Deployment Style

Install Style

Overview The install style describes the mapping of components in the software architecture to a file system in the production environment.

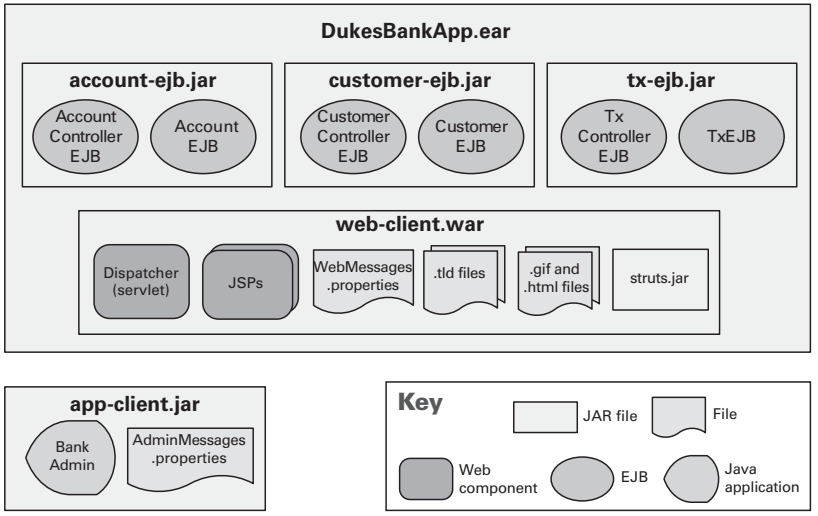
Elements

- *Software element: a C&C component*. *Required* properties of a software element, if any, usually include requirements on the production environments, such as a requirement to support Java or a database, or specific permissions on the file system.
- *Environmental element: a configuration item, such as a file or a folder. Provided* properties of an environmental element include indications of the characteristics provided by the production environments.

Relations

- *Allocated-to*. A component is allocated to a configuration item.
- *Containment*. One configuration item is contained in another.

Constraints Files and folders are organized in a tree structure, following an *is-contained-in* relation.



Work Assignment Style

Overview The work assignment style describes the mapping of the software architecture to the teams in the development organization.

Elements
- *Software element: a module.* Properties include the required skill set and available capacity (effort, time) needed.
- *Environmental element: an organizational unit, such as a person, a team, a department, a subcontractor, and so on.* Properties include the provided skill set and the capacity in terms of labor and calendar time available.

Relations *Allocated-to*. A software element is allocated to an organizational unit.

Constraints In general, the allocation is unrestricted; in practice, it is usually restricted so that one module is allocated to one organizational unit.

ECS Element (Module)		
Segment	**Subsystem**	**Organizational Unit**
Science Data Processing Segment (SDPS)	Client	Science team
	Interoperability	Prime contractor team 1
	Ingest	Prime contractor team 2
	Data Management	Data team
	Data Processing	Data team
	Data Server	Data team
	Planning	Orbital vehicle team
	uling	Orbital vehicle team
		Database team
		User interface team

Вертикальная архитектура и уровненная организация

Проблемы проектирования уровневой организации BcC

Смешение уровней приводит к взрывному росту сложности проектирования.

Основные причины смешения уровней:

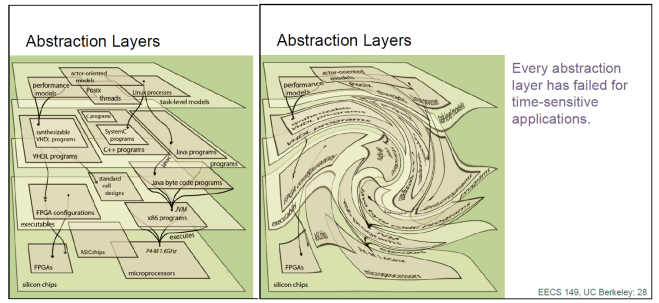
- Утилизация вычислительных ресурсов.
- Энергопотребление.
- Реальное время.

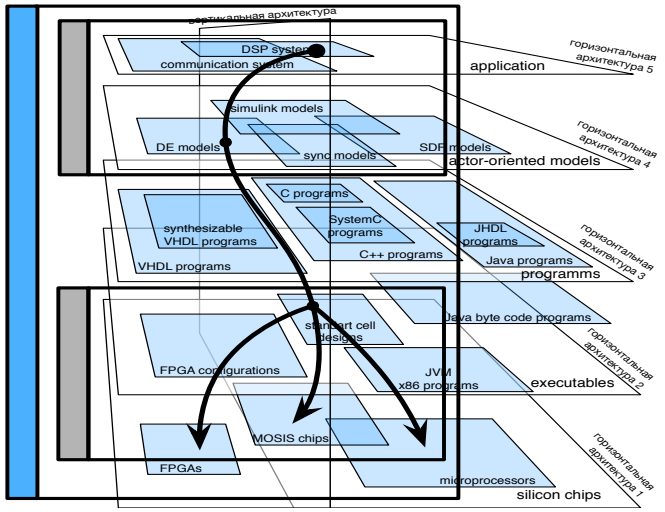
Необходимы методы и инструменты эффективного проектирования многоуровневых BcC.

САПР “общего” назначения	САПР в составе заказных элементов уровневой организации
• Шаблонная уровневая организация. • Фрагментарное покрытие маршрута проектирования многоуровневых систем. • Ориентированность на ВПл с одним уровнем организации вычислительного процесса.	• Специализированная уровневая организация. • Комплексное проектирование многоуровневых систем. • Специализированные многоуровневые вычислительные полуфабрикаты.

Примеры средств разработки многоуровневых BcC:

- Компиляция / синтез: LLVM, Xilinx ISE.
- Разработка: MPS, xText, NI LabView, Genie Lamp, Steps.
- Симуляция: GEM5,QEMU, Ptolemy.





Маршрут проектирования BcC, включающий несколько уровней организации.

Уровень и уровневая организация

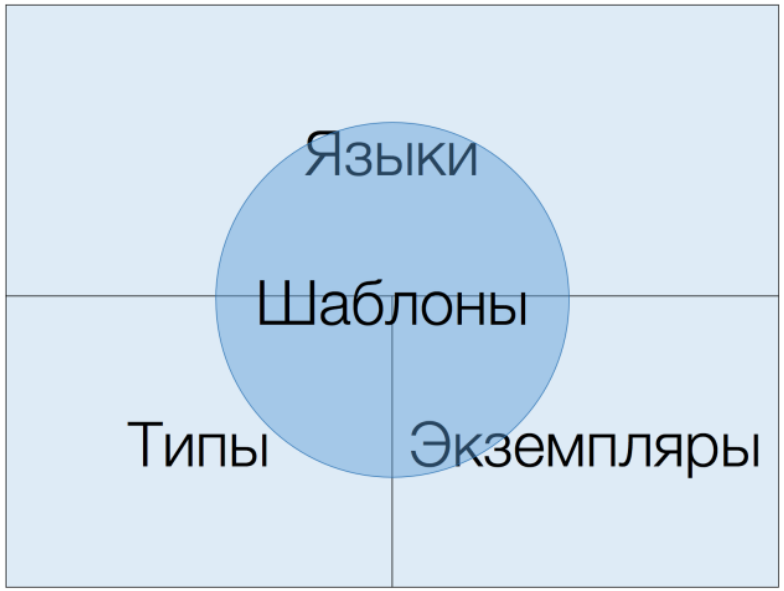
Уточнены понятия:

Уровень ВсС — один из вариантов рассмотрения вычислительного процесса целевой системы, адресованный заданной группе интересов или ориентированный на конкретную ВПл.

Уровневая организация — совокупность уровней ВсС, организованных в иерархическую структуру, определяющую методику разработки и принципы функционирования системы.

Вычислительная платформа (ВПл) — объект повторного использования для решения заданного класса вычислительных задач, включающий языковые, методологические, инструментальные и технологические средства.

Предложена классификация ВПл: по классу решаемых задач; по распространённости; по возможности к адаптации; по назначению; по способу реализации.



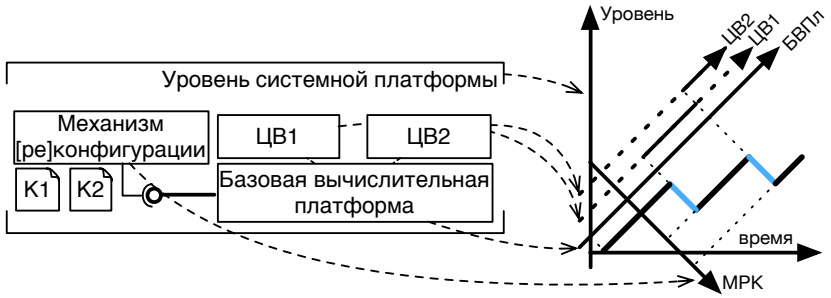
Обобщённое представление вычислительной платформы.

Онтологические модели уровневой организации

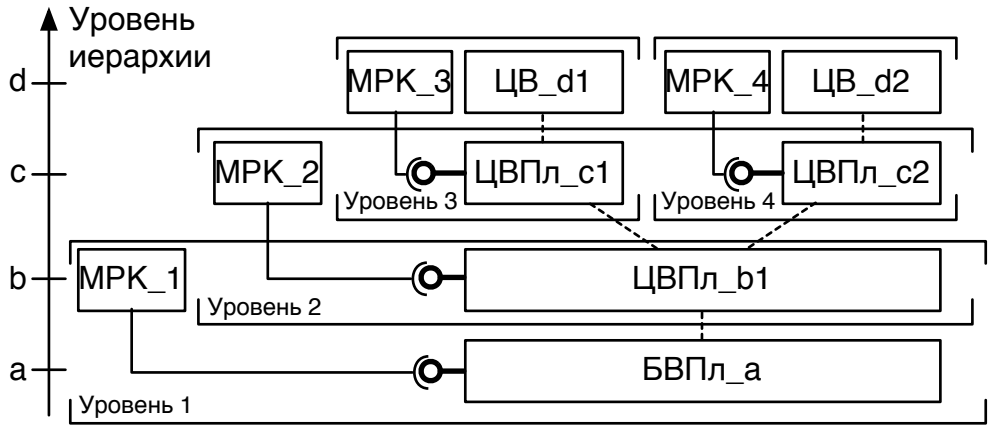
Предложены **модели:**

- **[Ре]конфигурации**, обобщающая понятия программирования, конфигурирования и реконфигурирования.
- **Иерархической организации**, описывающая уровневую организацию во времени и демонстрирующая целостность вычислительного процесса.

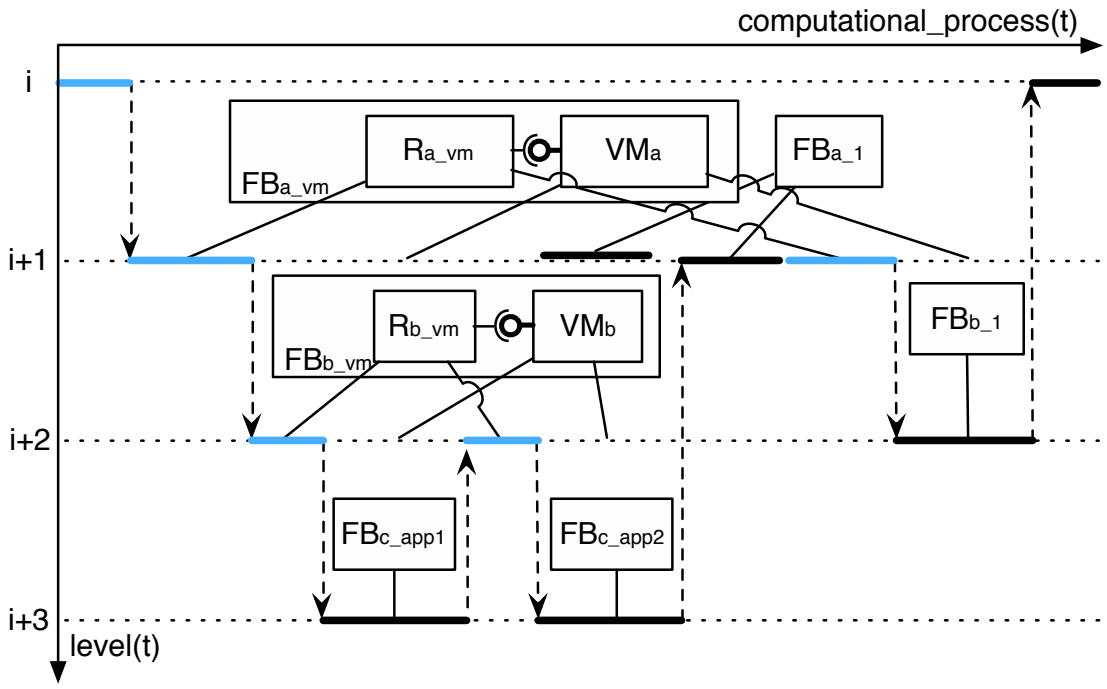
Назначение: использование в качестве шаблонов проектирования (Design pattern) при разработке заказных элементов уровневой организации.

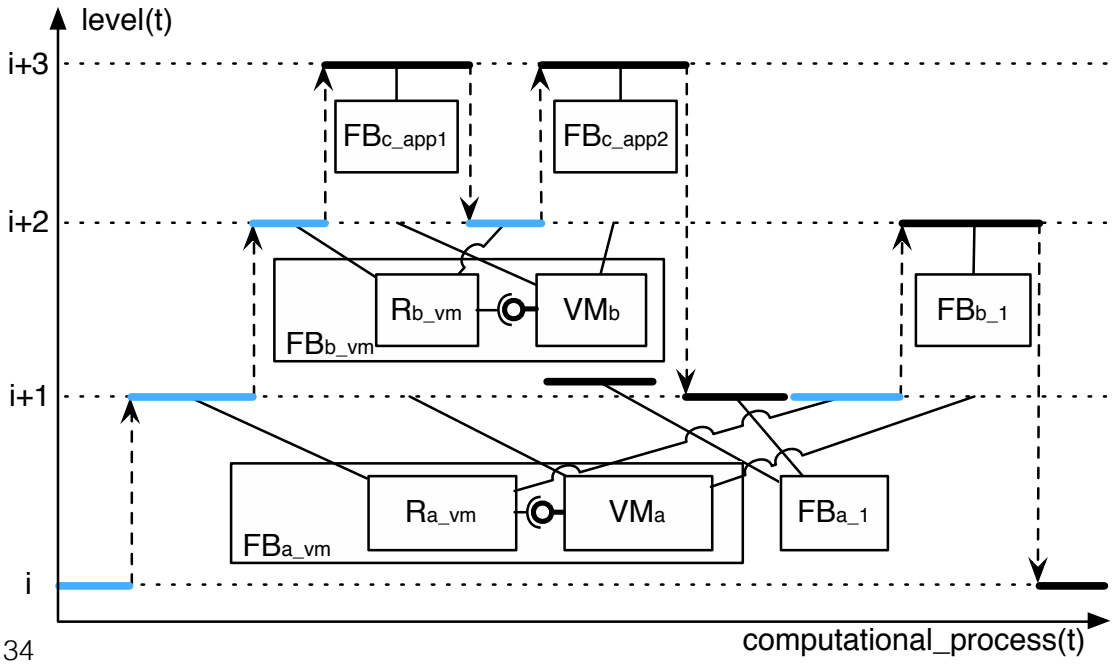


Онтологическая модель [ре]конфигурации.



Онтологическая модель иерархической организации.


computational_process(t)
i
i+1
i+2
i+3
level(t)
FBa_vm
Ra_vm
VMa
FBa_1
FBb_vm
Rb_vm
VMb
FBb_1
FBc_app1
FBc_app2


level(t)
i+3
i+2
i+1
i
computational_process(t)
FBc_app1
FBc_app2
FBb_vm
Rb_vm
VMb
FBb_1
FBa_vm
Ra_vm
VMa
FBa_1

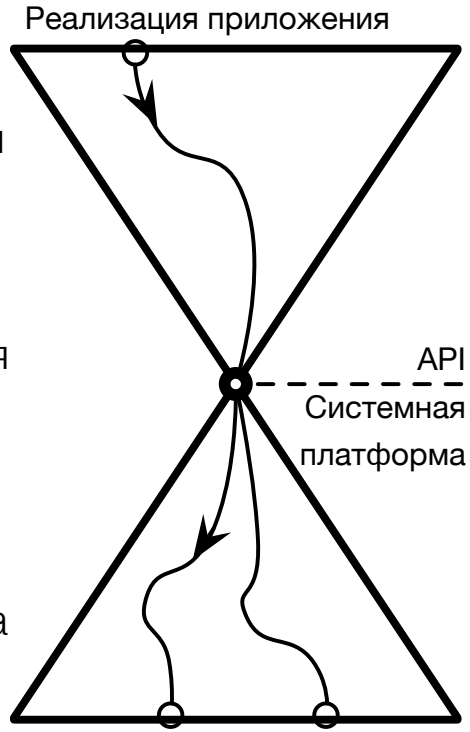
Элементы повторного использования

Уточнено понятие **системной платформы** (СПл) — совокупность ВПл, используемая при разработке и эксплуатации заданного класса ВсС, а также включающая методологические и инструментальные средства разработки.

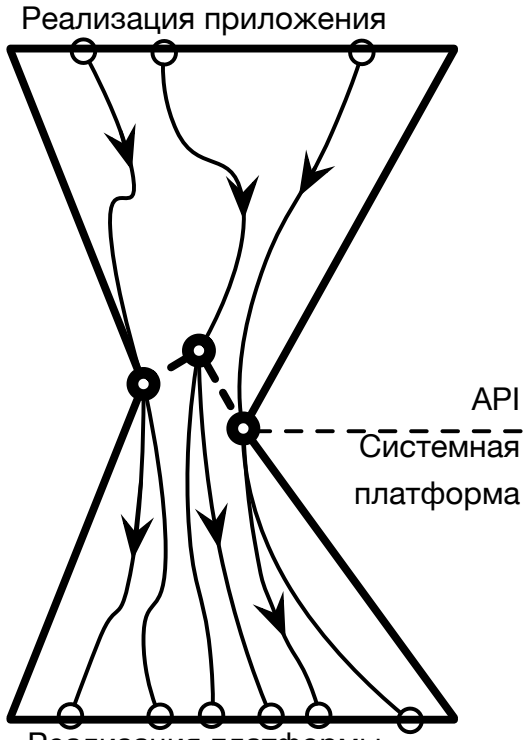
Введено понятие **многоуровневого вычислительного агрегата** (МВА) — целостного объекта повторного использования для решения фиксированного класса вычислительных задач, включаемого в состав СПл и требующего конфигурирования нескольких ВПл.

Основано на понятии архитектурного агрегата из HLD-методологии.

Определены основные элементы повторного использования уровневой организации ВсС: ВПл, МВА, СПл.



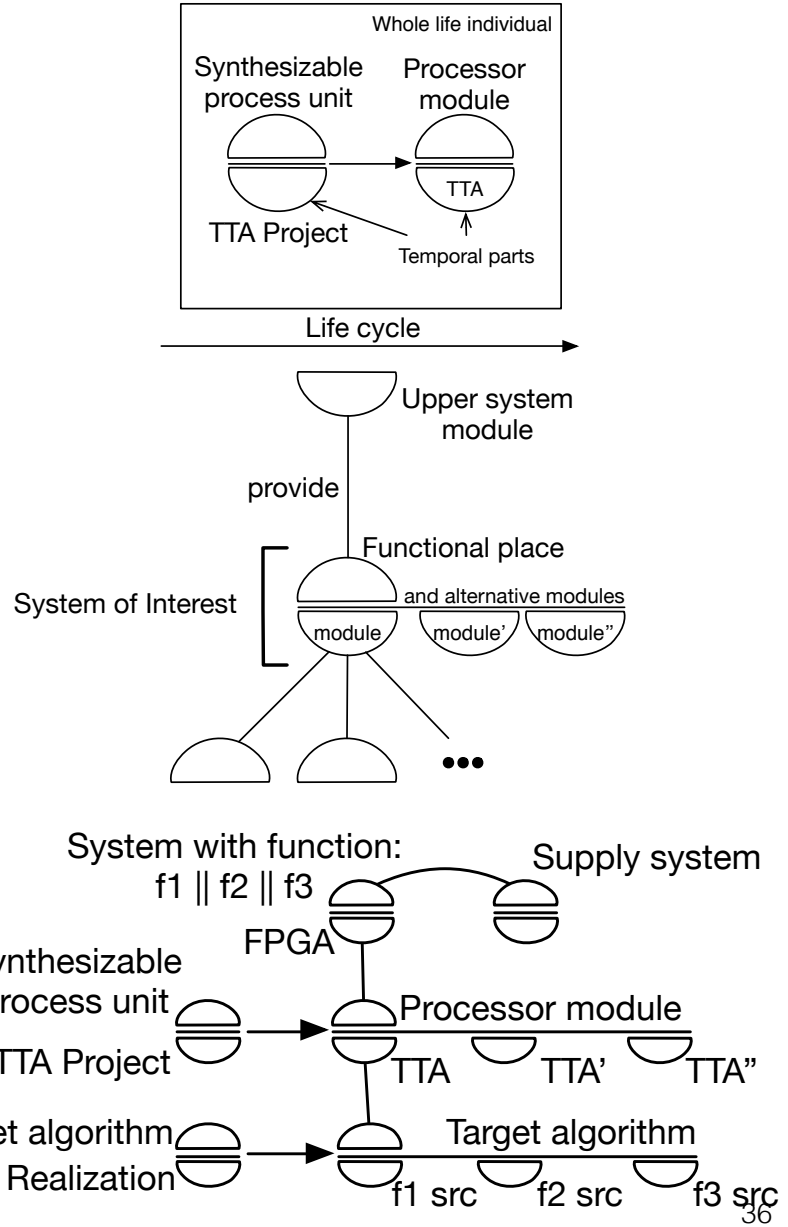
СПл — Платформно-ориентированное проектирование.



СПл — новое определение.

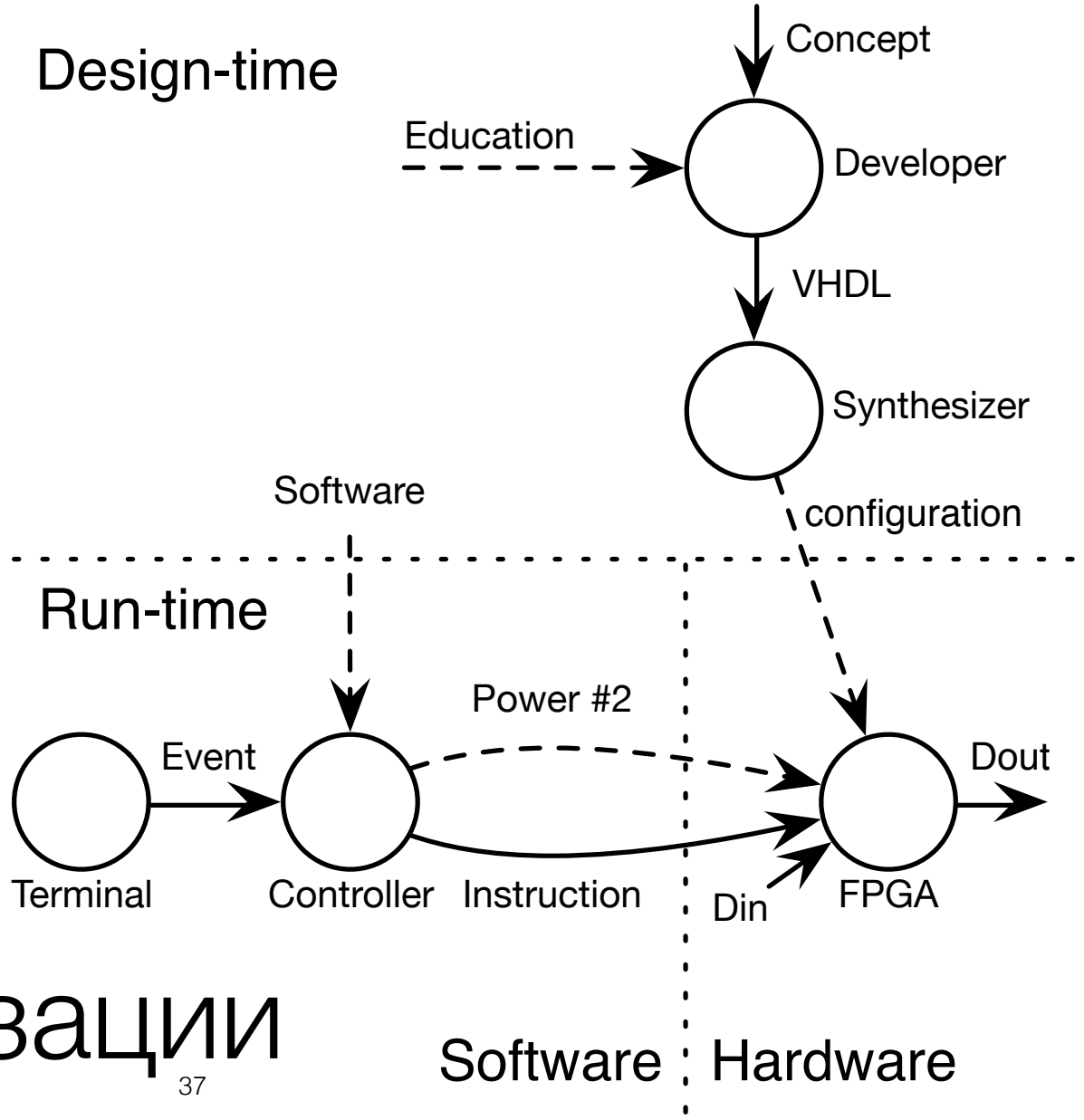
Системно-иерархический архитектурный стиль

- **Система** — единство функционального места и конструкции (обеспечивающей функциональные требования).
- Работа с жизненным циклом построена на концепциях “**whole-life-individual**” (индивида-всей-жизни) и “**temporal-part**” (темпоральной-части) [ISO 15926].
- Особенности:
  - Позволяет трассировать требования и свойства.
  - Явное отображение функциональных мест элементов системы на протяжение всего жизненного цикла.
  - Естественное отображение свойства модульности и многофункциональности.
  - Понятие «системы» является излишне общим, что затрудняет выделение уровней.
  - Ориентация на функциональное назначение затрудняет анализ уровневой организации.
- **Позволил обобщить понятие [ре]конфигурации.**



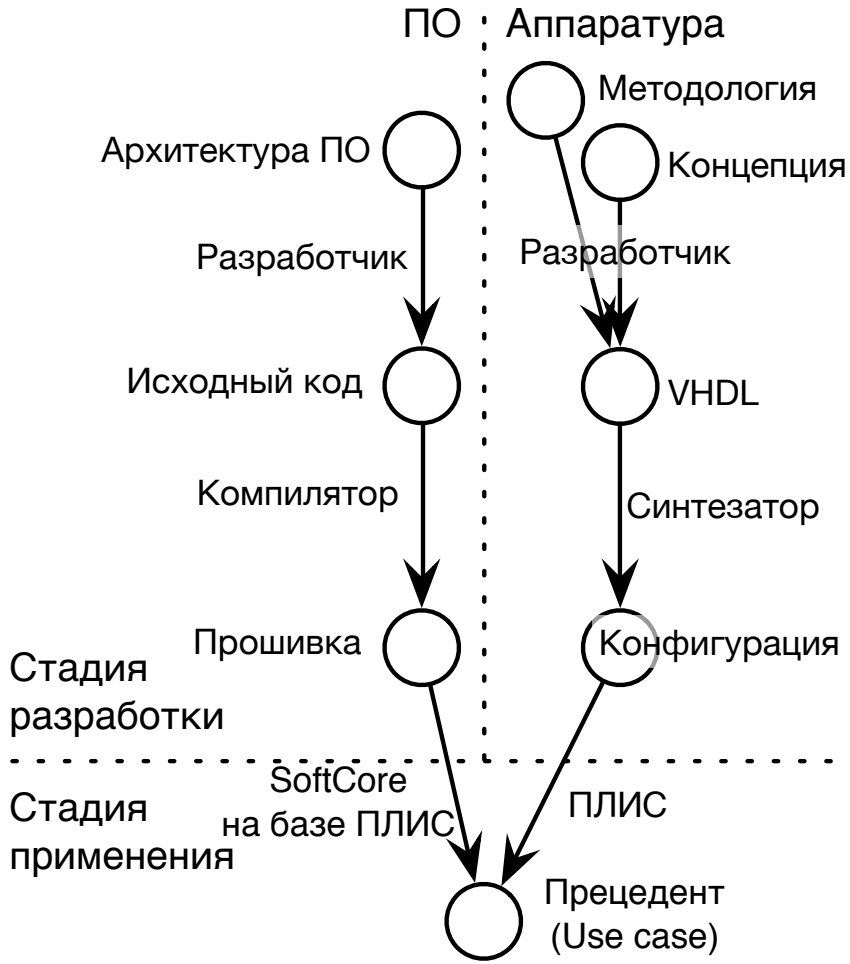
Граф актуализации

- Represent the structure of computational process actualisation.
- One of axis of design space.
- Oriented acyclic graph, where **vertex** — translator, **arcs** — any transfers between translator.
- *Unified* computation process representation for
  - Design-time and run-time phases.
  - Software and Hardware component.



Архитектурный стиль “Модифицированный граф актуализации“

- Центральное место занимают представления ВсС в рамках различных ВПл вне зависимости от стадии жизненного цикла.
- Ориентированный нециклический граф, где вершины — спецификации, используемые в ВсС, рёбра — трансляции между ними.
- Особенности:
  - Анализ инструментальных цепочек и представлений вычислительного процесса.
  - Последовательная актуализация спецификаций в физический процесс не позволяет включить в рассмотрение спецификации, используемые для верификации.
  - Унифицированное представление для различных стадий жизненного цикла и ВПл.
- **Позволил продемонстрировать необходимость целостного представления вычислительного процесса для всего жизненного цикла ВсС.**



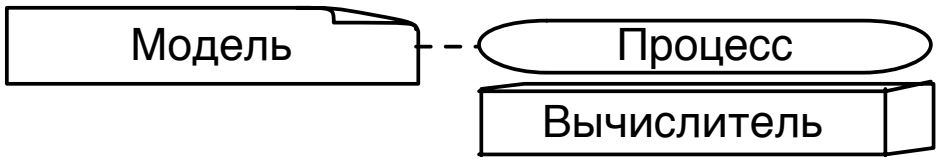
Модель-процесс-вычислитель Уровень ВсС

Представление уровня организации ВсС:

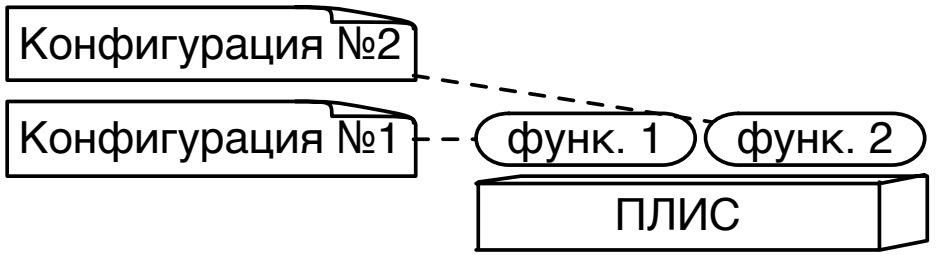
- Модель ВП (конфигурация, программа, журнал, архитектурная спецификация).
- Вычислительный процесс (ВП). Может включать частичные вычислительные процессы.
- Вычислитель.
- Тройственное отношение актуализации.

Основан на методах и принципах моделирования высших онтологий (ISO-15926, BORO-методология):

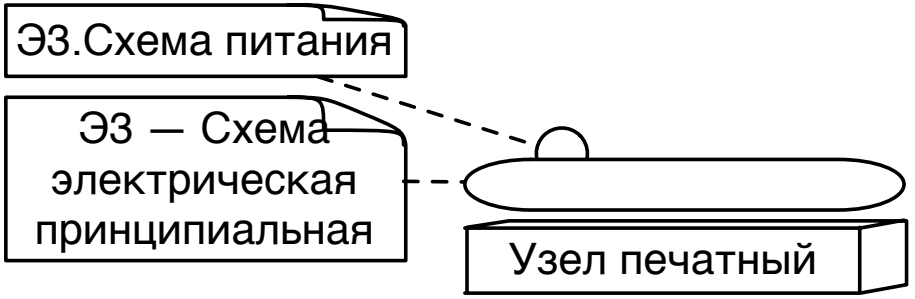
- абстрагирование от момента времени;
- абстрагирование от природы отношений.
- целостность ВП.



Пример описания уровня организации ВсС.



Уровень ПЛИС с реконфигурацией.



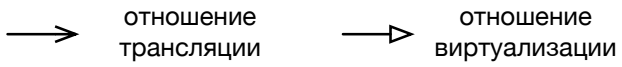
Уровень производства печатных узлов.

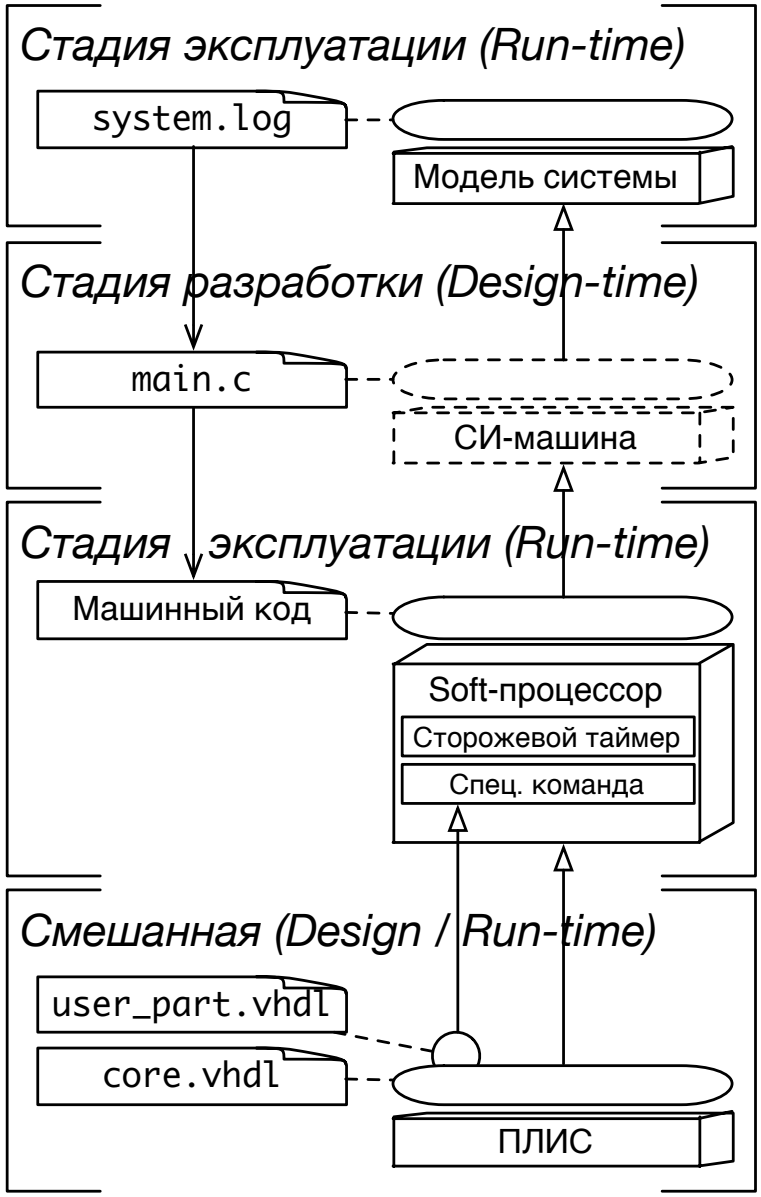
Модель-процесс-вычислитель Межуровневые взаимосвязи

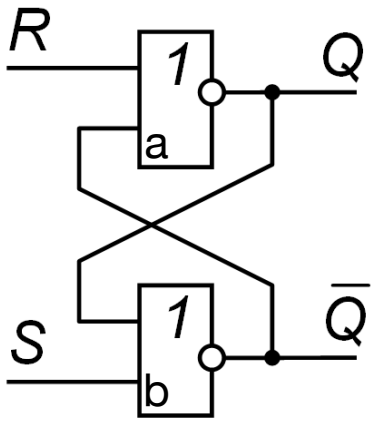
Представление уровневой иерархии:

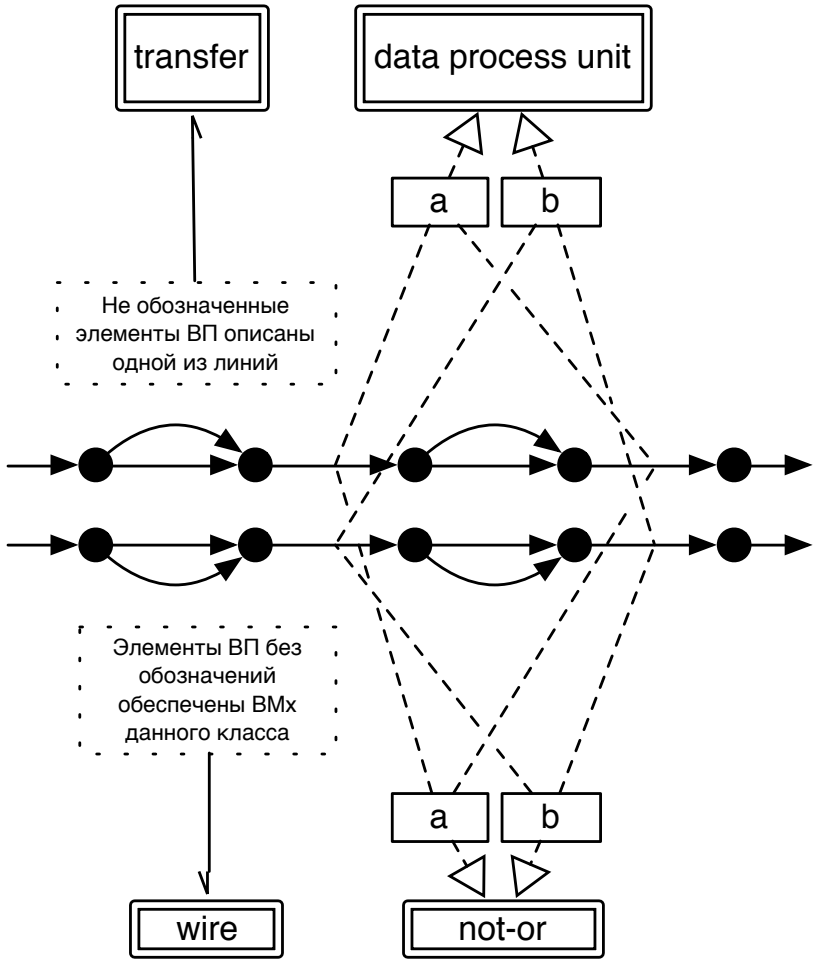
- **Отношение трансляции** — формальное соответствие моделей.
- **Отношение виртуализации** — абстракция над ВП, формирование вычислителя и модели вычислений.
  - **Вычислительный механизм (ВМх)** — абстрактный элемент вычислителя, определённый на основании структуры ВП.
- Виртуальные модели, процессы, вычислители, отношения (пунктир).

На схеме: упрощённая уровневая организация ВсС с Soft-процессором, запрограммированная на языке СИ и ведущая системный журнал.

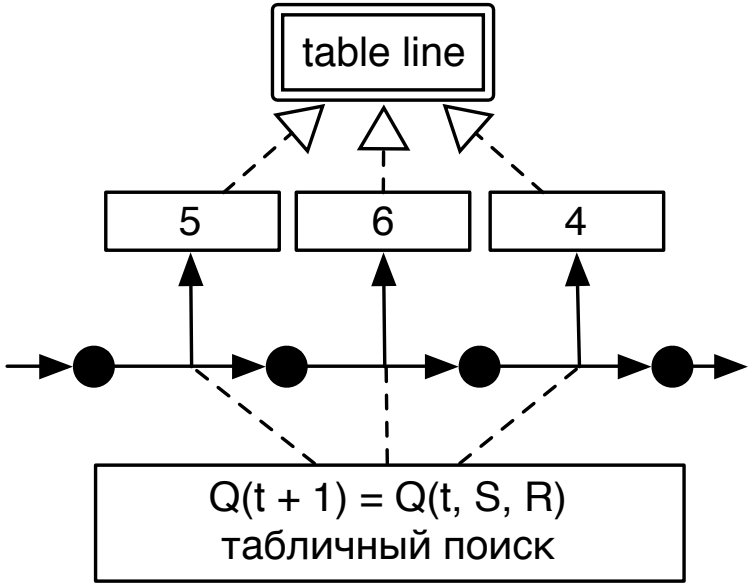




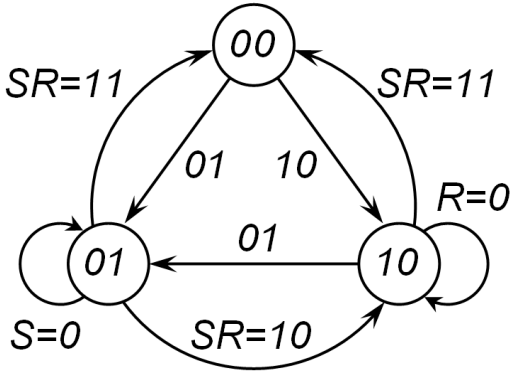


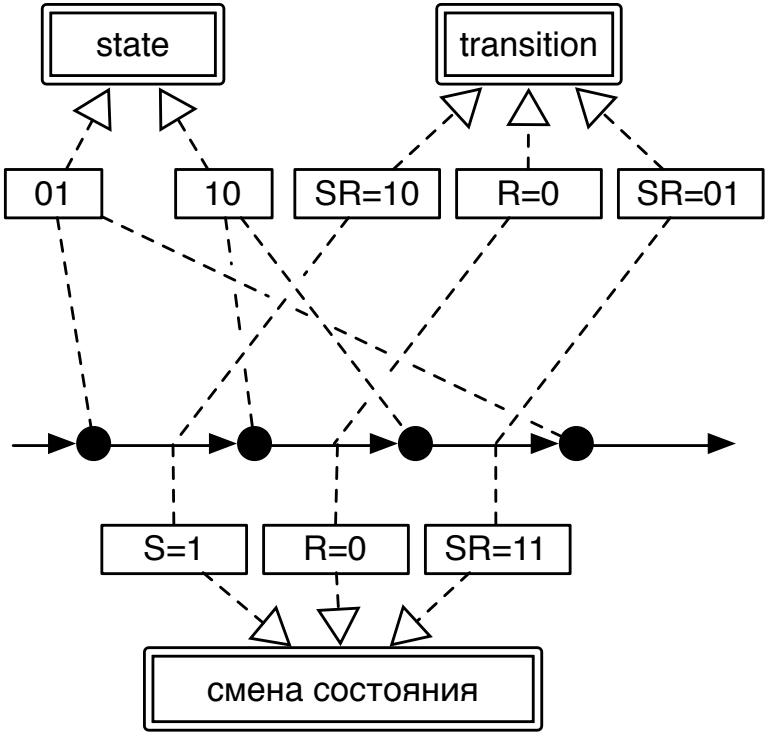


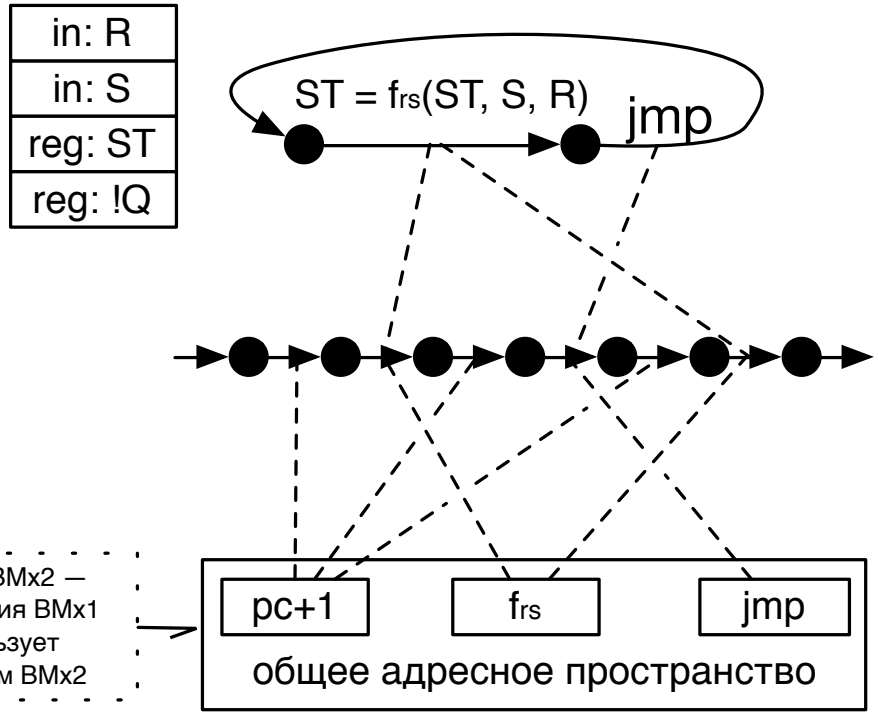
S	R	Q(t)	$\overline{Q}$(t)	Q(t+1)	$\overline{Q}$(t+1)
0	0	0	1	0	1
0	0	1	0	1	0
0	1	0	1	0	1
0	1	1	0	0	1
1	0	0	1	1	0
1	0	1	0	1	0
1	1	0	1	не определено	не определено
1	1	1	0	не определено	не определено



RS-триггер

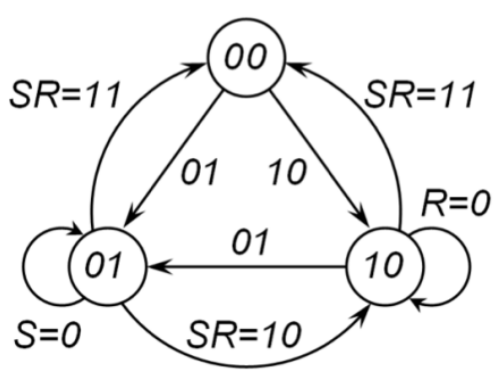




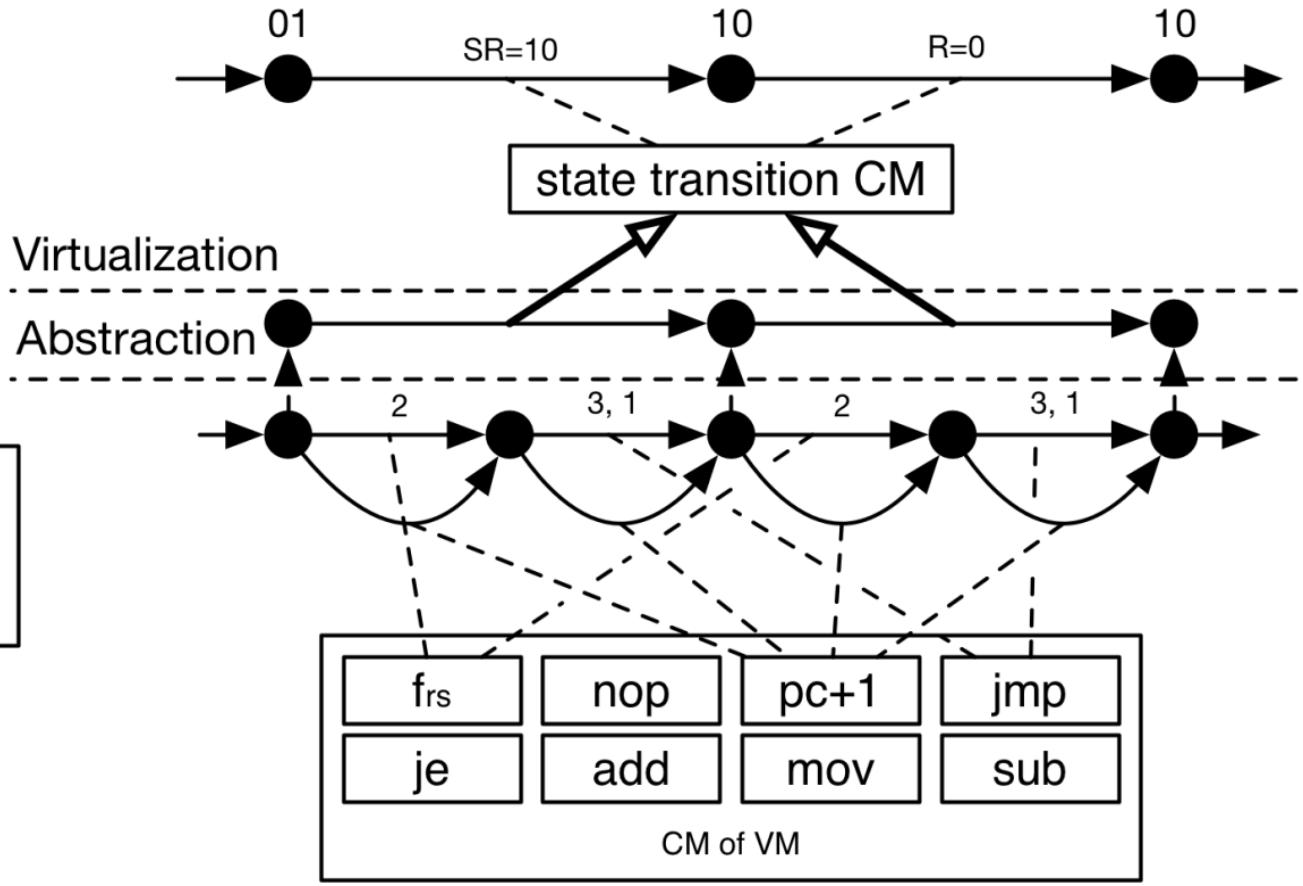


BMx1 в BMx2 — реализация BMx1 использует механизм BMx2

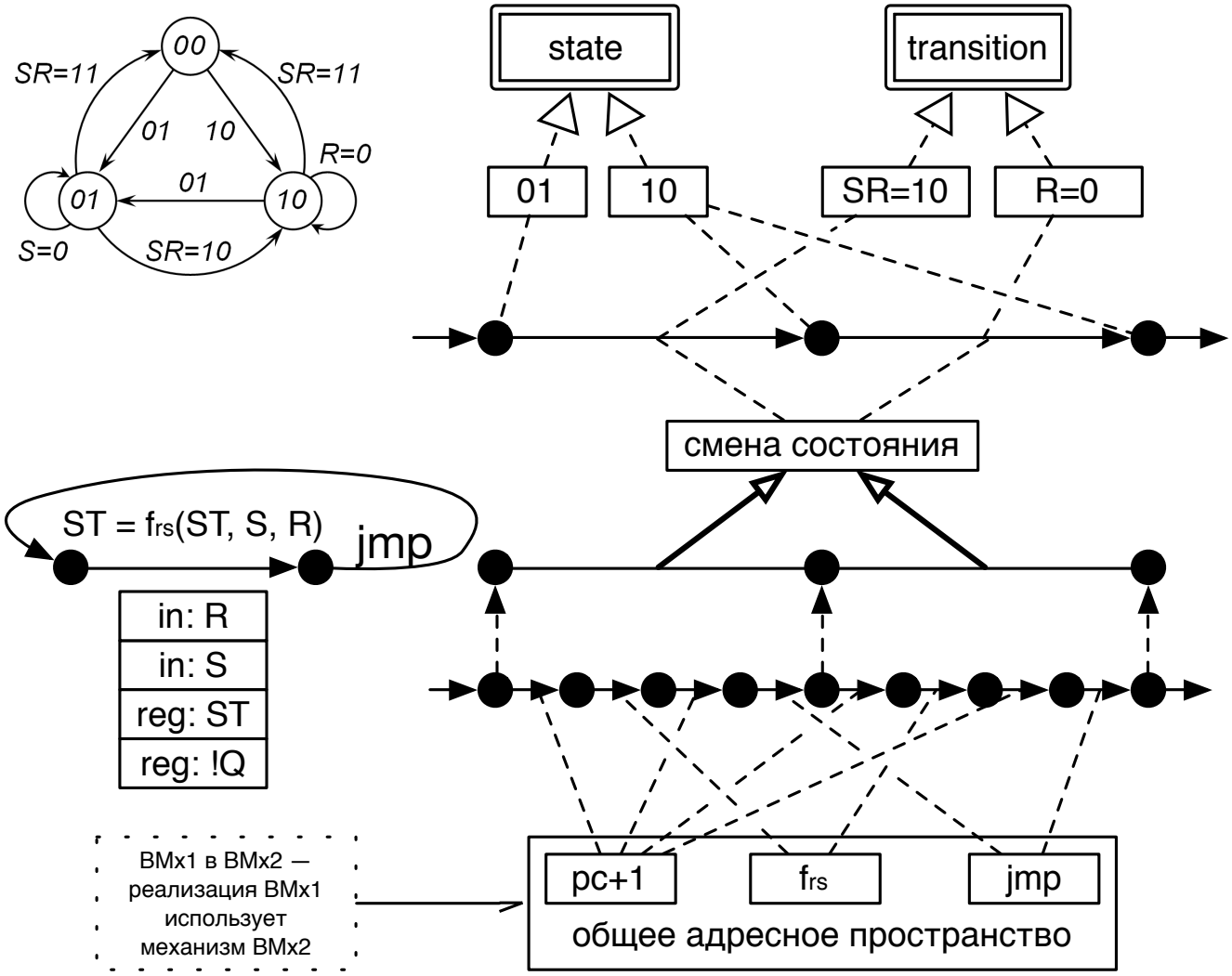
RS-триггер


00
SR=11
SR=11
01
10
R=0
01
01
10
S=0
SR=10


LABEL START
ST = frs(ST, S, R)
JMP START
1
2
3


01
SR=10
10
R=0
10
state transition CM
Virtualization
Abstraction
2
3, 1
2
3, 1
frs
nop
pc+1
jmp
je
add
mov
sub
CM of VM

RS-триггер



RS триггер (3/3)

1. Decomposition style
2. Uses style
3. Generalisation style
4. Layered style
5. Aspect style
6. Data model
7. Data flow style
8. Call-return style Peer-to-peer
9. Call-return style Service-Oriented style
10. Event-Based Style
11. Repository Style
12. Deployment style
13. Install style
14. Work assignment style
15. Системно-иерархический архитектурный стиль
16. Граф актуализации
17. Модифицированный граф актуализации
18. Модель-процесс-вычислитель
19. ———————————————————
20. IEC 61131
21. IEC 61499
22. AADL
23. Domain Diagrams (xtUML)
24. Entity Diagrams
25. ISO 15926
26. ….