ARCHITECTURAL STYLES - CONTINUED

DR. VIVEK NALLUR

VIVEK.NALLUR@SCSS.TCD.IE

https://www.scss.tcd.ie/vivek.nallur/teaching/slides/

OUTLINE OF THIS TALK

- Architectural Styles
  - Model-View-Controller
  - Blackboard
  - Service-Oriented

But first ...

ASSIGNMENT 2 - WORDCHAIN

Can be found at the usual place

https://www.scss.tcd.ie/Vivek.Nallur/teaching/cs3012/

Deadline: 19-October-2016, 10:00 a.m.

WORDCHAIN - ASSUMPTIONS YOU CAN MAKE

There will be either *one* chain or *zero*

A word may have multiple possible successors, but only one of them will result in a chain that uses all words

BACK TO STYLES - MODEL-VIEW-CONTROLLER

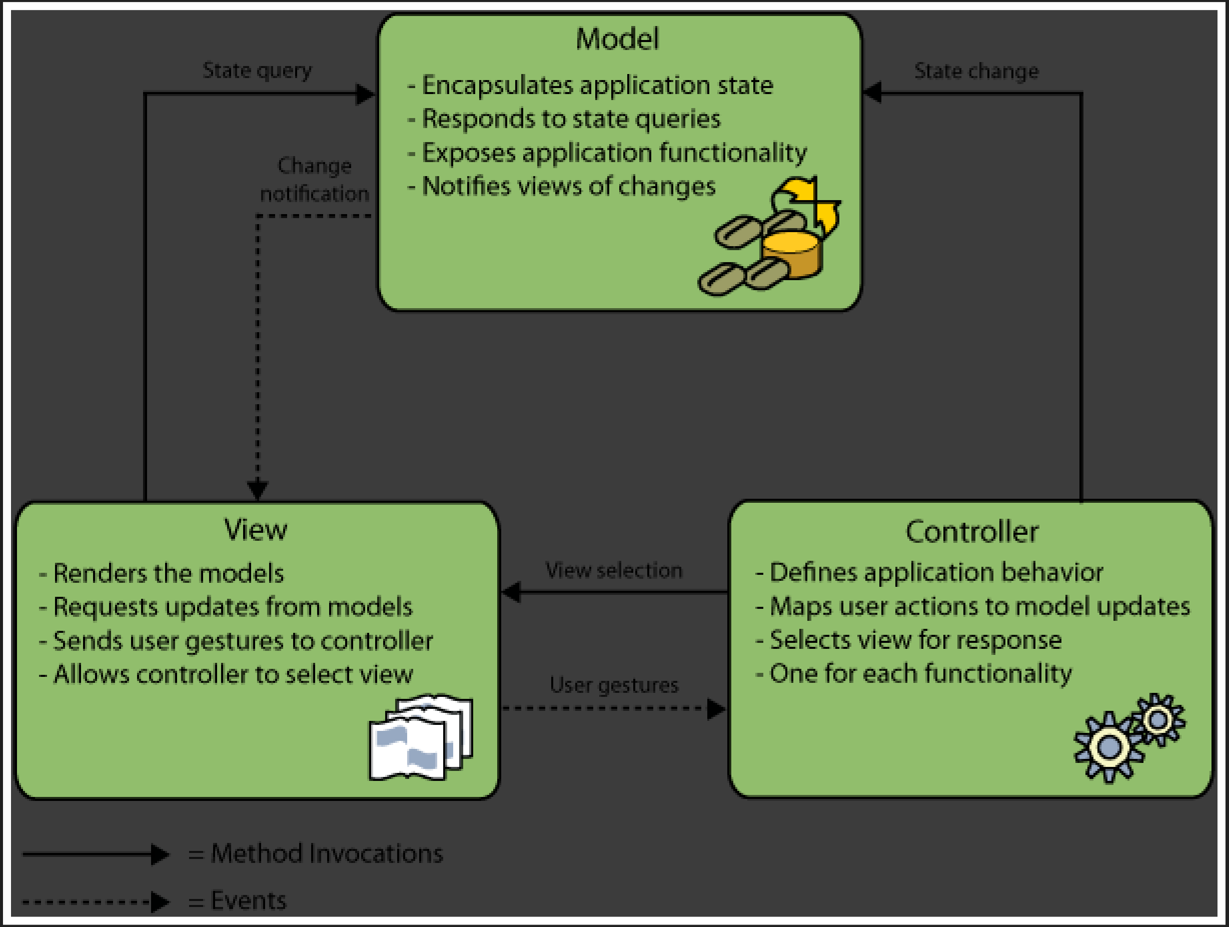
Divides an interactive application into three parts:

- Model - Responsible for data and data management
- View - Display information to the user
- Controller - Handle user-input, validate, etc.

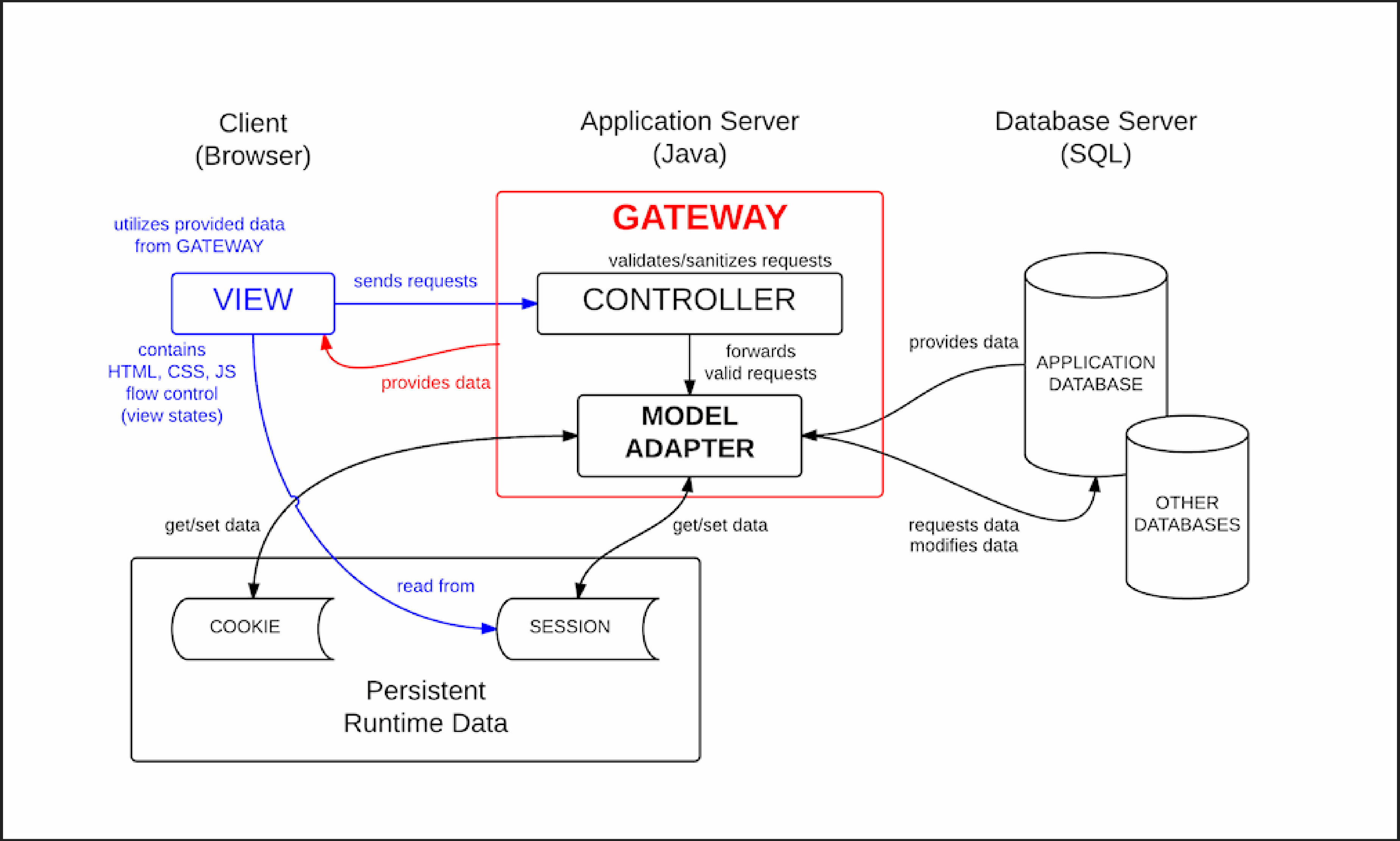
MODEL-VIEW-CONTROLLER - TYPICAL PROPERTIES OF DOMAIN

- Same information needs to be presented in multiple ways
- Display and behaviour of application must reflect data changes immediately
- Changes to UI should be easy (perhaps even at runtime!)

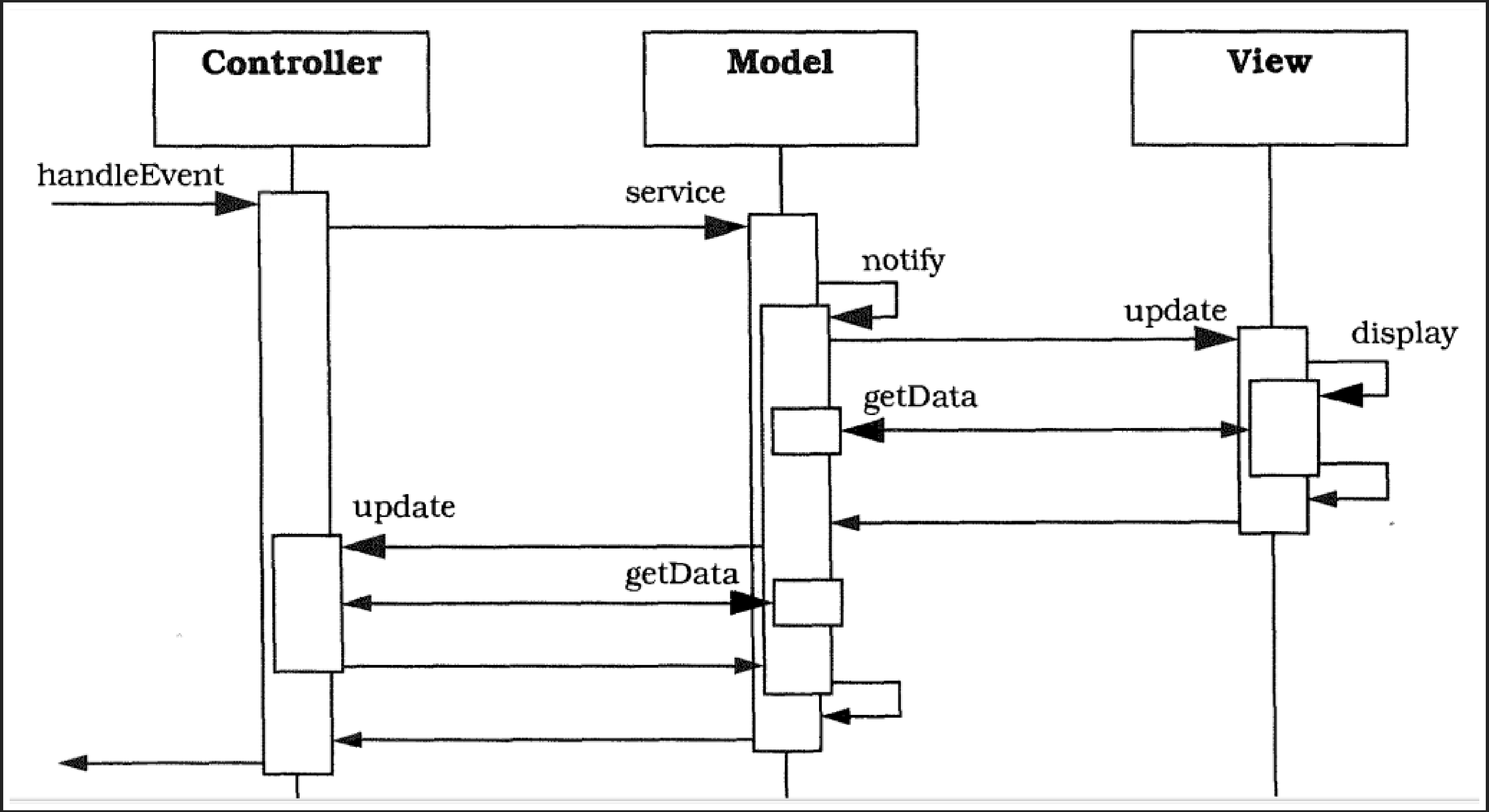
MODEL-VIEW-CONTROLLER (MVC) - SOLUTION STYLE


Model
- Encapsulates application state
- Responds to state queries
- Exposes application functionality
- Notifies views of changes
State query
State change
Change notification
View
- Renders the models
- Requests updates from models
- Sends user gestures to controller
- Allows controller to select view
View selection
User gestures
Controller
- Defines application behavior
- Maps user actions to model updates
- Selects view for response
- One for each functionality
= Method Invocations
= Events

MVC - IMPLEMENTATION



MVC - SEQUENCE OF CALLS



MVC - PROS / CONS

Pros	
Multiple Views	From the same model, different views can be instantiated dynamically
Synchronized Views	Change to data is immediately reflected to all viewers
Pluggability	Views and controllers can be changed without affecting the model

MVC - PROS / CONS

Cons

Increased complexity	Simple menu items become excessively complex
Excessive updates	All changes to model may not need to be propagated
Inefficiency of data access	Levels of indirection, in the name of de-coupling

STYLE: BLACKBOARD

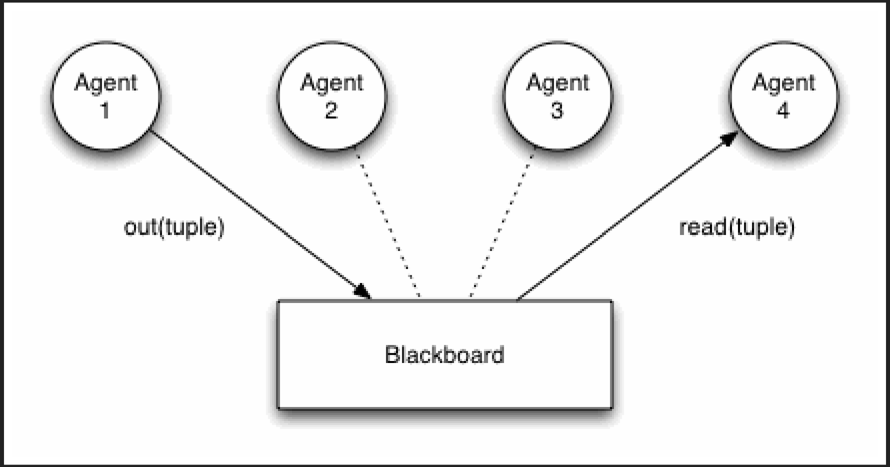
Useful for problems where no deterministic strategies are known. Several specialized subsystems pool knowledge to build an approximate/possible answer

BLACKBOARD - TYPICAL PROPERTIES OF DOMAIN

- Complete search of solution space is not feasible
- Domain is immature, so no known algorithms to solve problem
- Different algorithms solve different partial problems

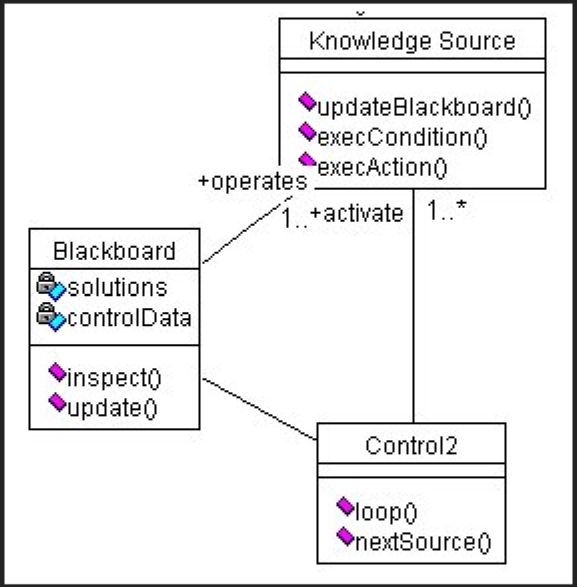
BLACKBOARD - SOLUTION STYLE

Opportunistic problem-solving using independent experts using a common data-structure

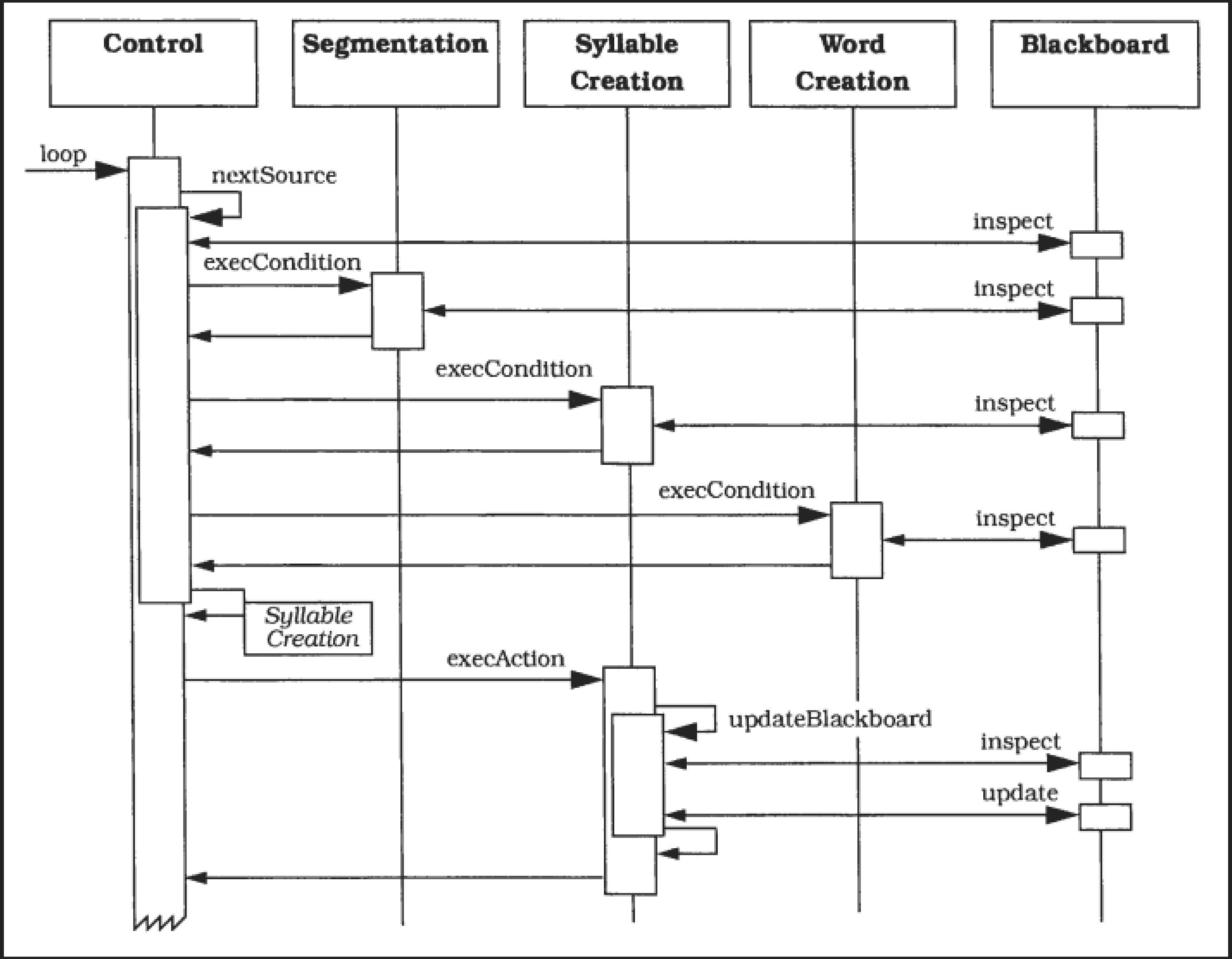


BLACKBOARD - IMPLEMENTATION

- Blackboard
- Knowledge-source(s)
- Control



BLACKBOARD - HOW IT WORKS



BLACKBOARD - PROS / CONS

Pros	
Changeability	Supports changing of knowledge sources easily
Experimentation	Strict separation of components allows easy experimentation
Fault Tolerance	All results are hypotheses, so noise in data is okay
Potential Parallelism	Disjoint algorithms can work in parallel on solution space

BLACKBOARD - PROS / CONS

Cons	
Testability	Results are not reproducible
Low Efficiency	Computationally costly to reject wrong hypotheses
High Development Effort	Since domain is ill-specified, takes years to build

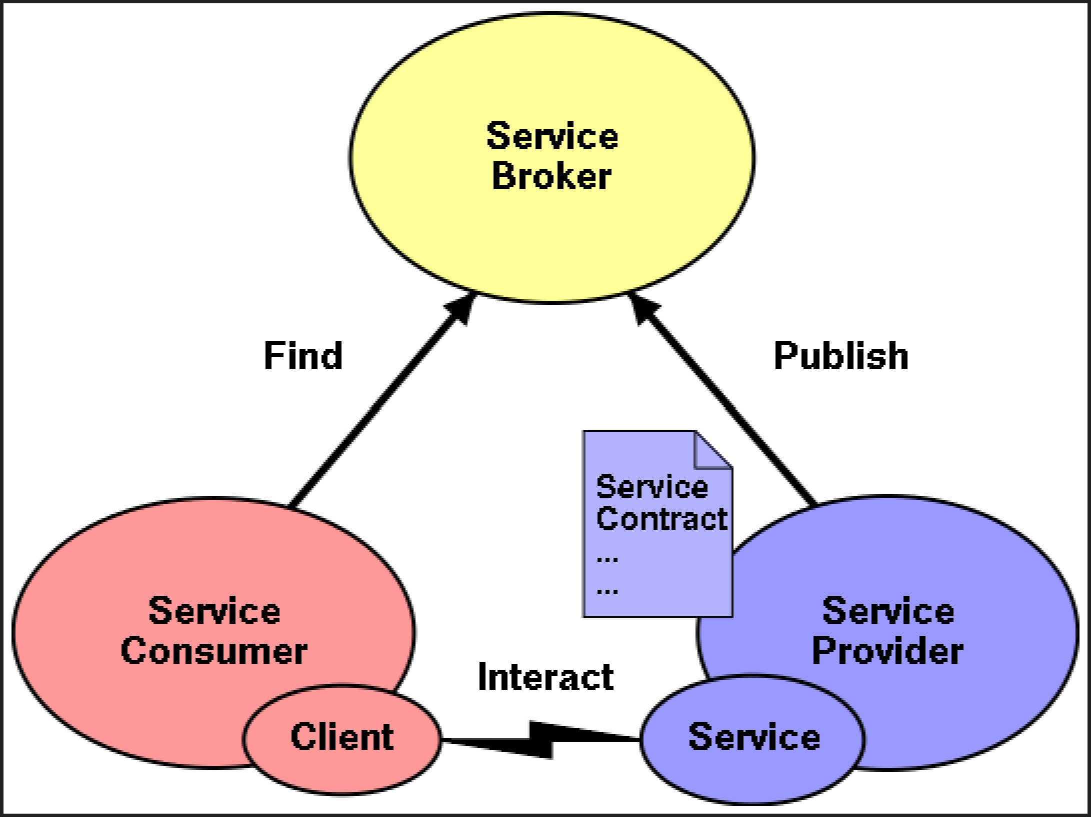
STYLE: SERVICE-ORIENTED

Enable application functionality to be provided and consumed as sets of services published at a granularity relevant to the service consumer. Services can be invoked, published and discovered, and are abstracted away from the implementation using a single, standards-based form of interface

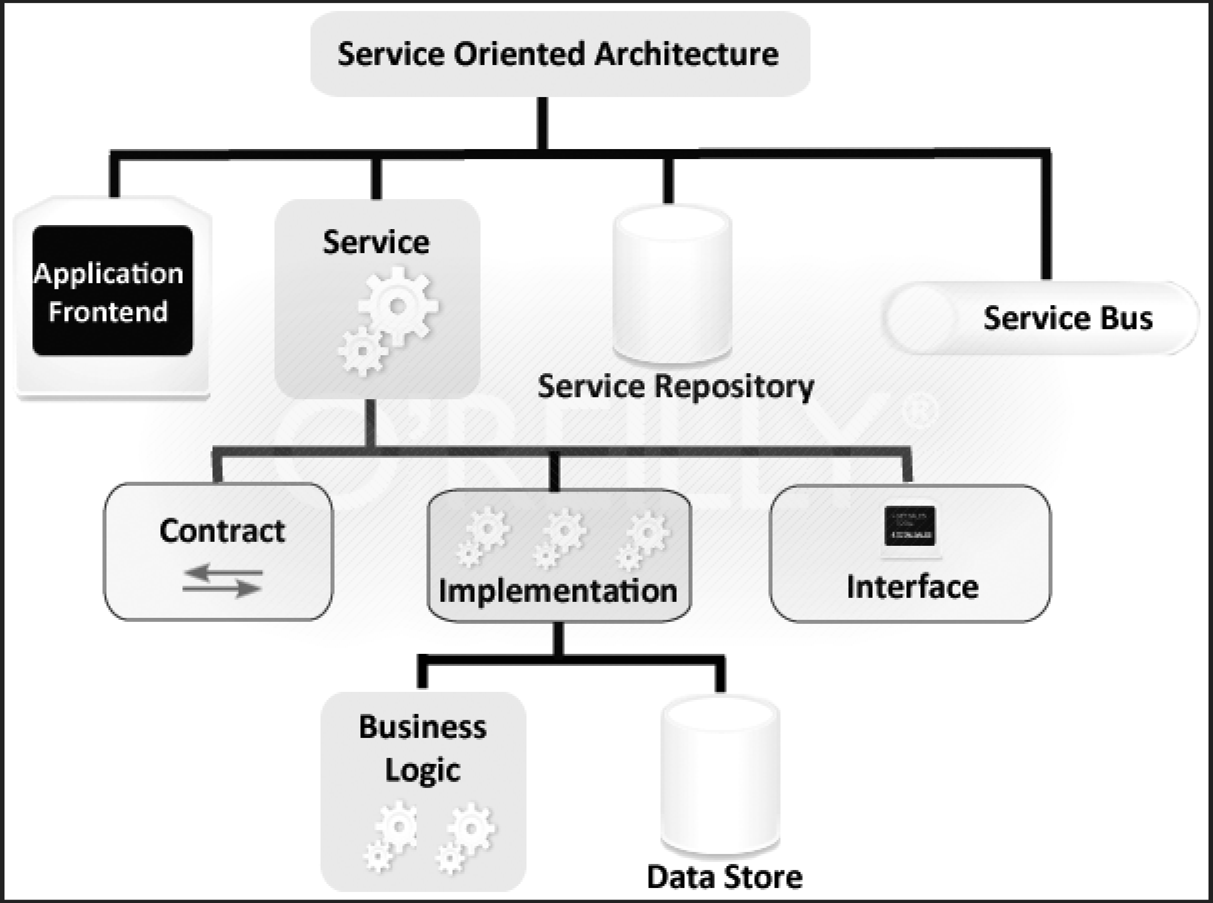
SERVICE-ORIENTED - TYPICAL PROPERTIES OF DOMAIN

- Automated discovery and usage are essential
- Platform independence of service endpoint
- Formal contract places obligations on both consumer and provider

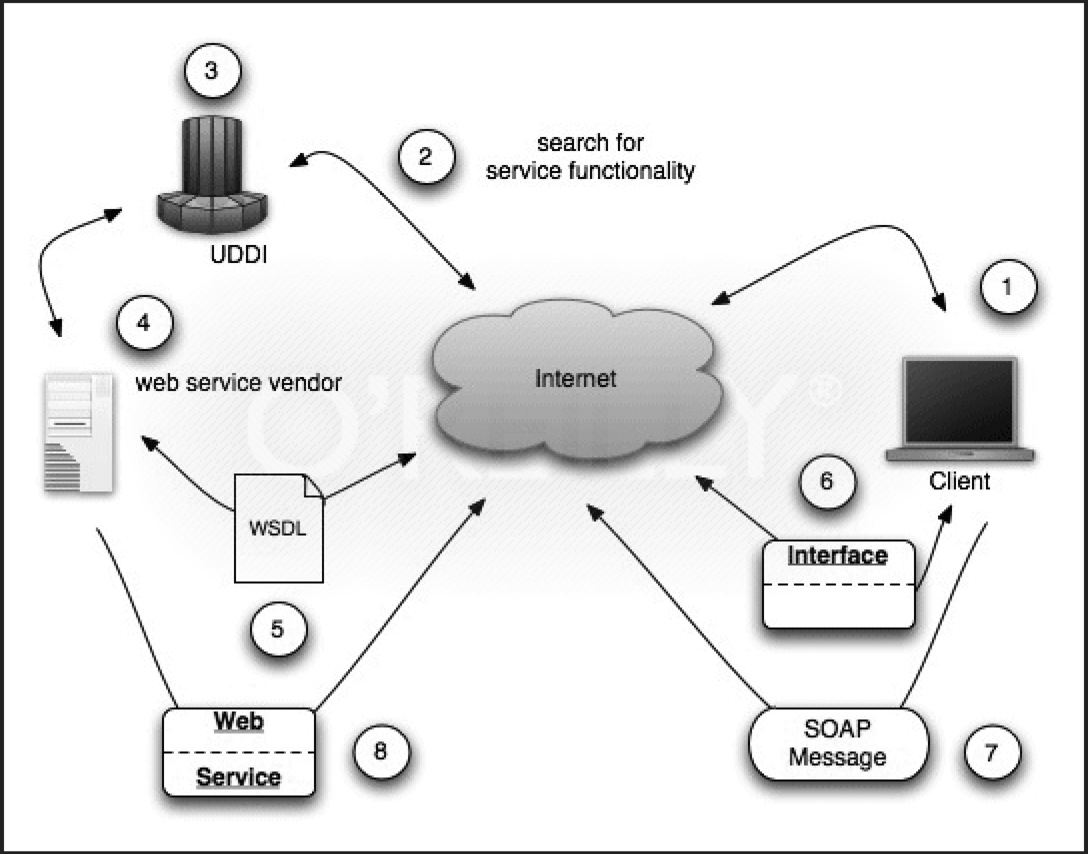
SERVICE-ORIENTED - SOLUTION STYLE



SERVICE-ORIENTED - IMPLEMENTATION



SERVICE-ORIENTED - LIFECYCLE



SERVICE-ORIENTED - PROS / CONS

Pros	
Reusability	Small, self-contained, loosely coupled functionality
Maintainability	Can change between versions, as long as contract is not violated
Scalability	Multiple instances can run on the same server

SERVICE-ORIENTED - PROS / CONS

Cons	
Service Management	Orchestration or Choreography is complex
Overhead	Computationally costly to constantly validate parameters and use HTTP

NEXT CLASS: OO DESIGN AND CLASS DIAGRAMS

Guest lecture by: Dr. Ivana Dusparic

THAT'S ALL, FOLKS!

Questions? Comments?