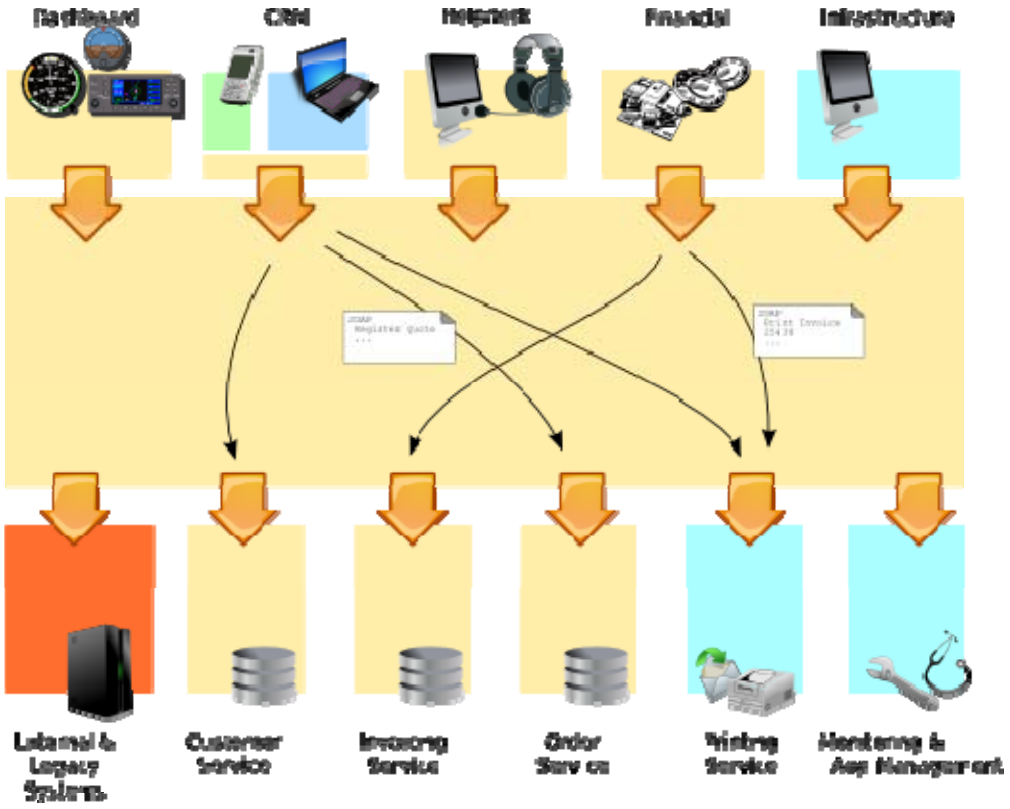
CPSC 875

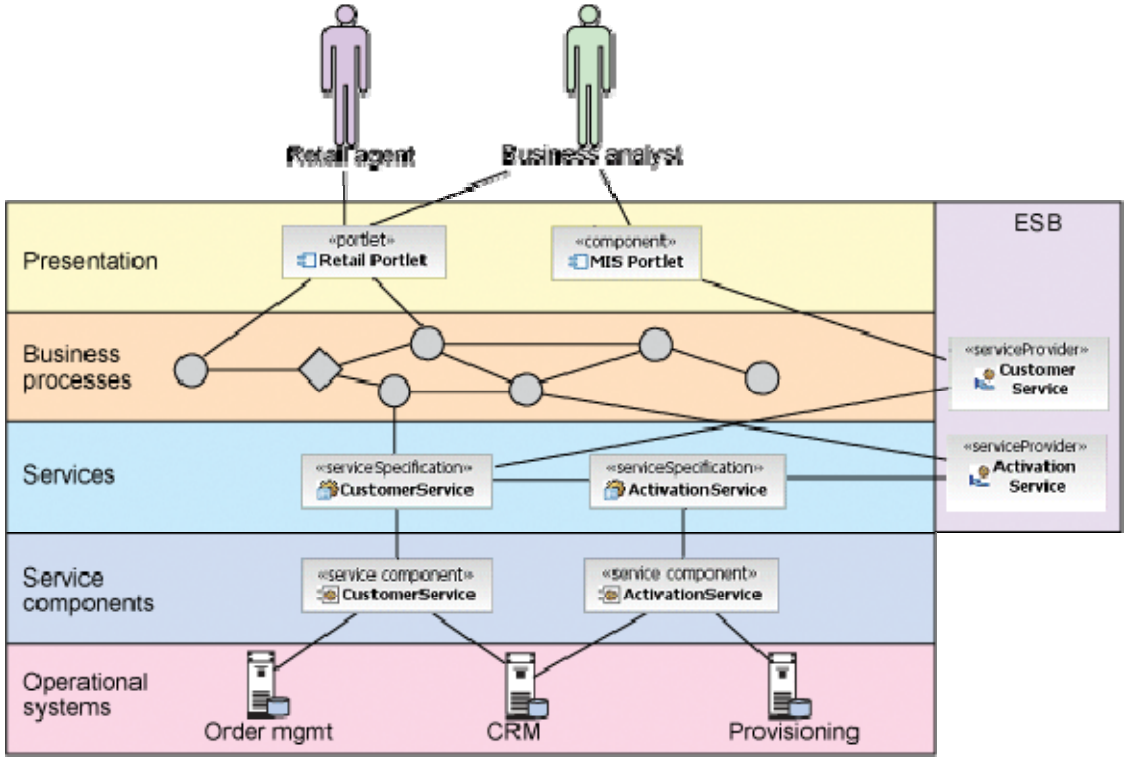
John D. McGregor

C7 - Design

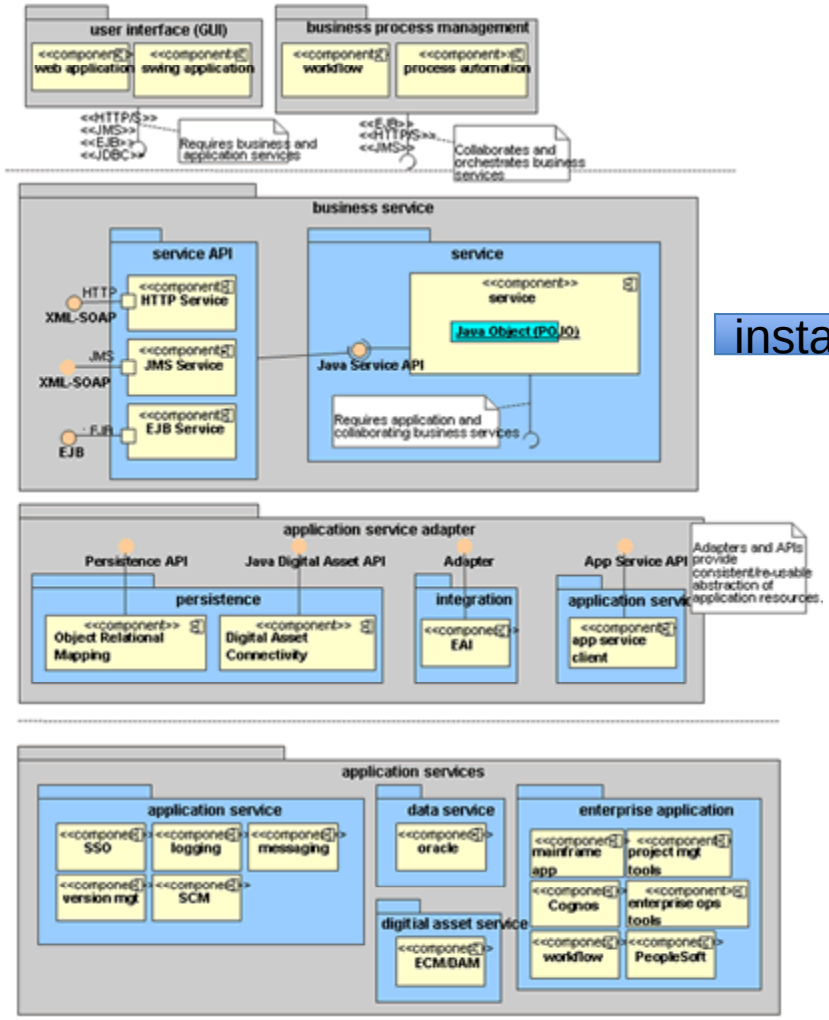
Service oriented architecture



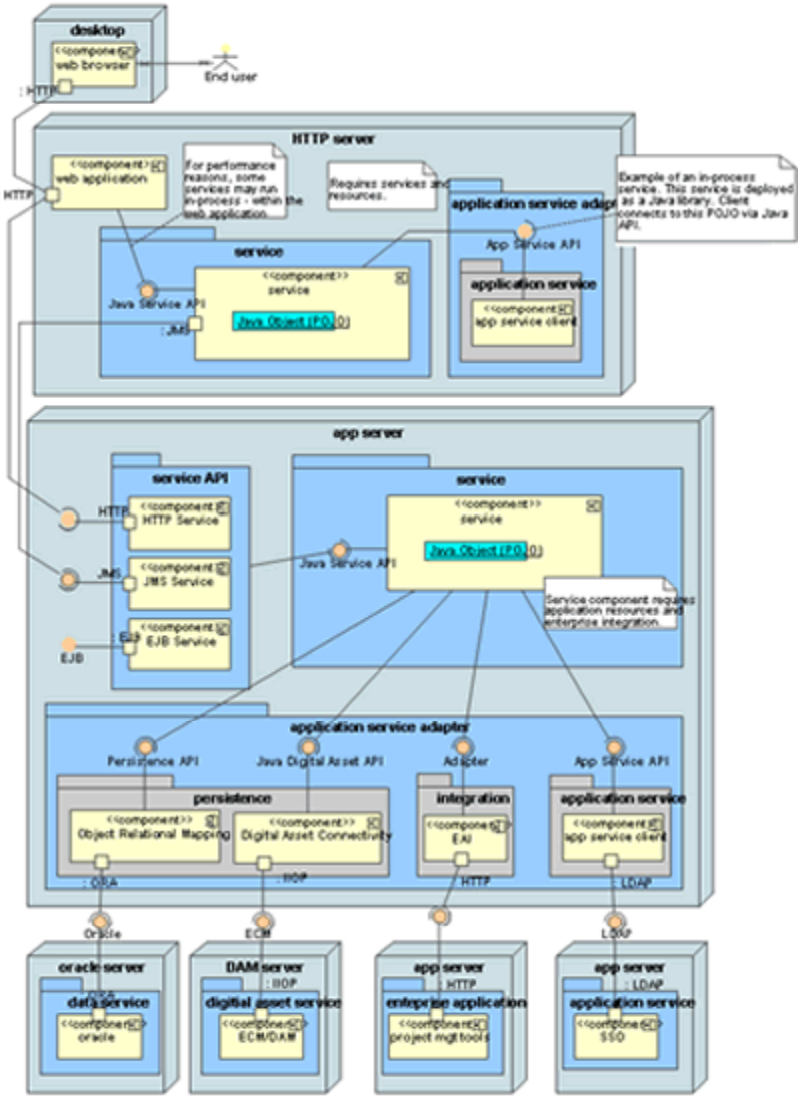
SOA


Retail agent
Business analyst
Presentation
«portlet»
Retail Portlet
«component»
MIS Portlet
ESB
Business processes
«serviceProvider»
Customer Service
Services
«serviceSpecification»
CustomerService
«serviceSpecification»
ActivationService
«serviceProvider»
Activation Service
Service components
«service component»
CustomerService
«service component»
ActivationService
Operational systems
Order mgmt
CRM
Provisioning

Service-oriented Architecture

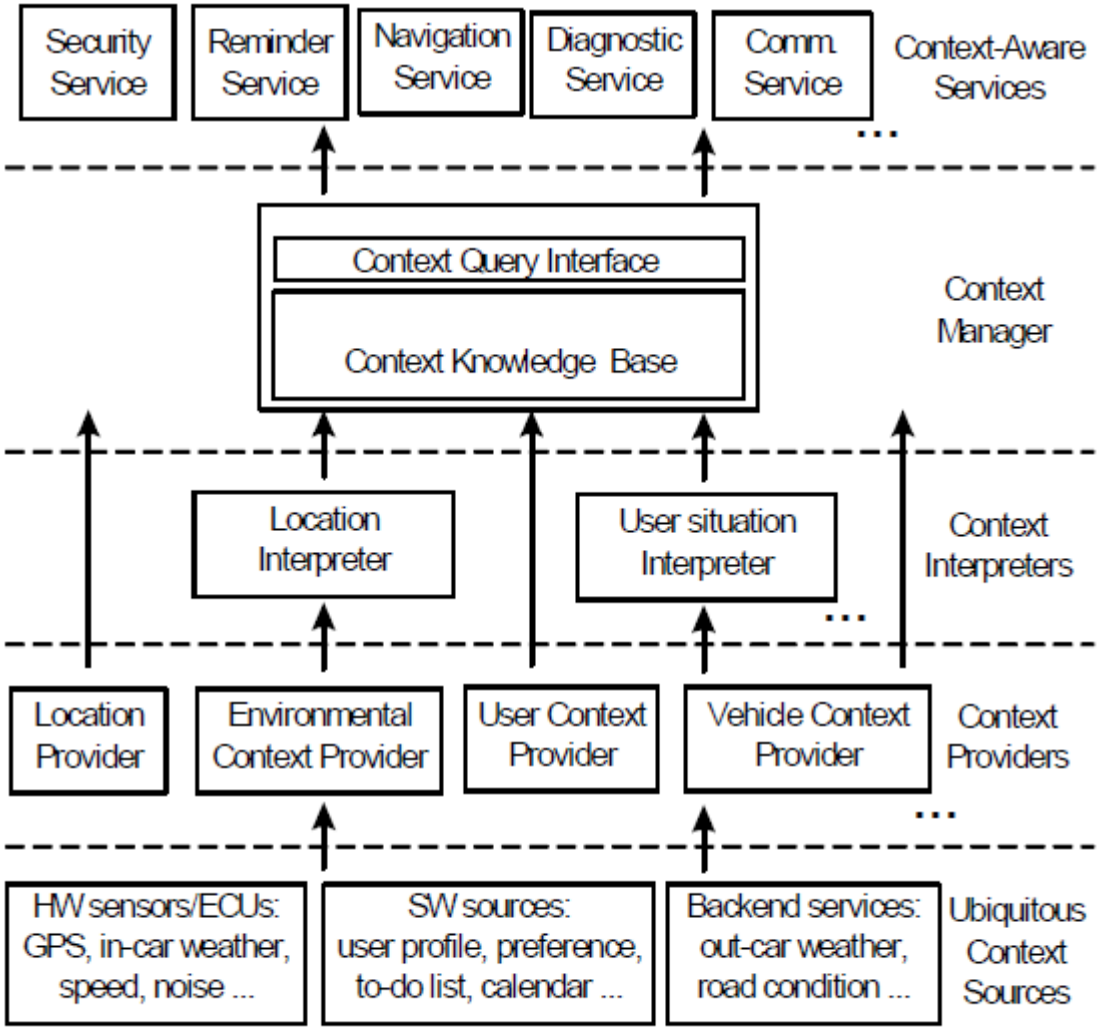


instantiate

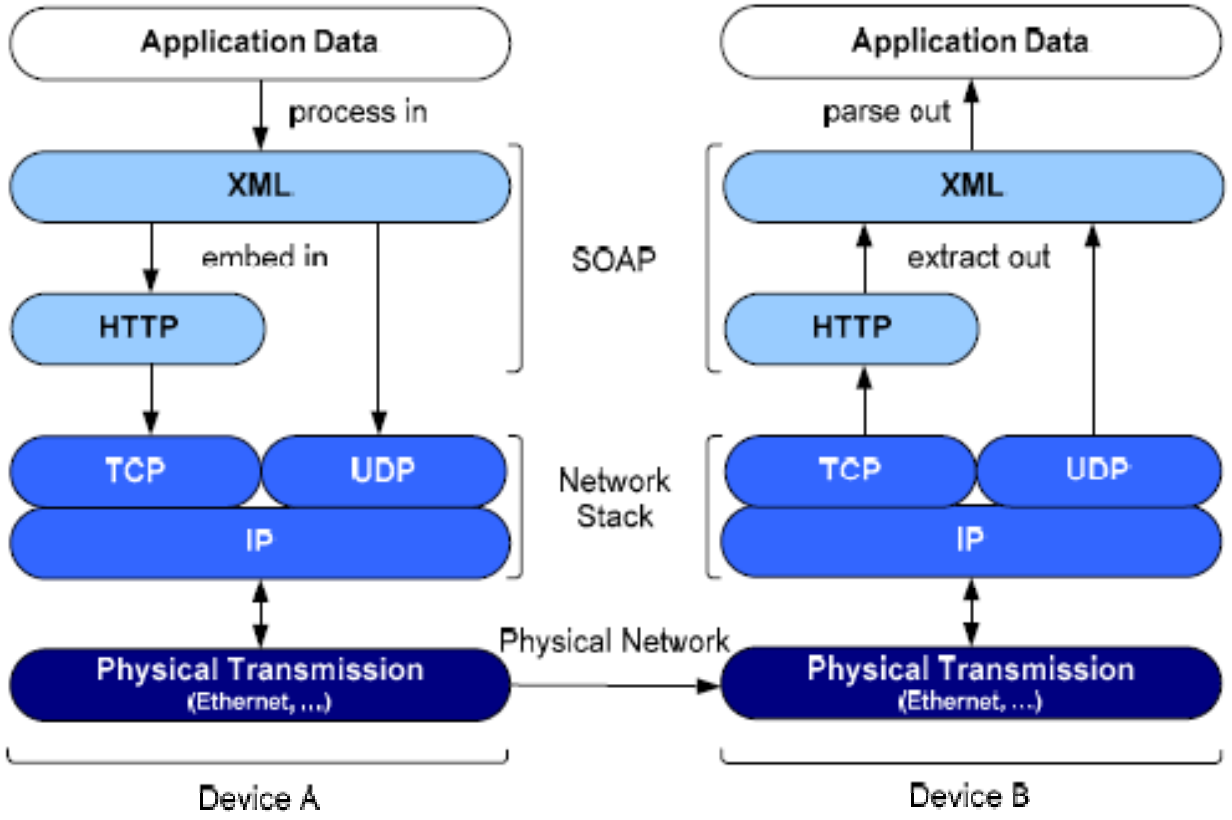


http://talkdata.com/soa_arch.html

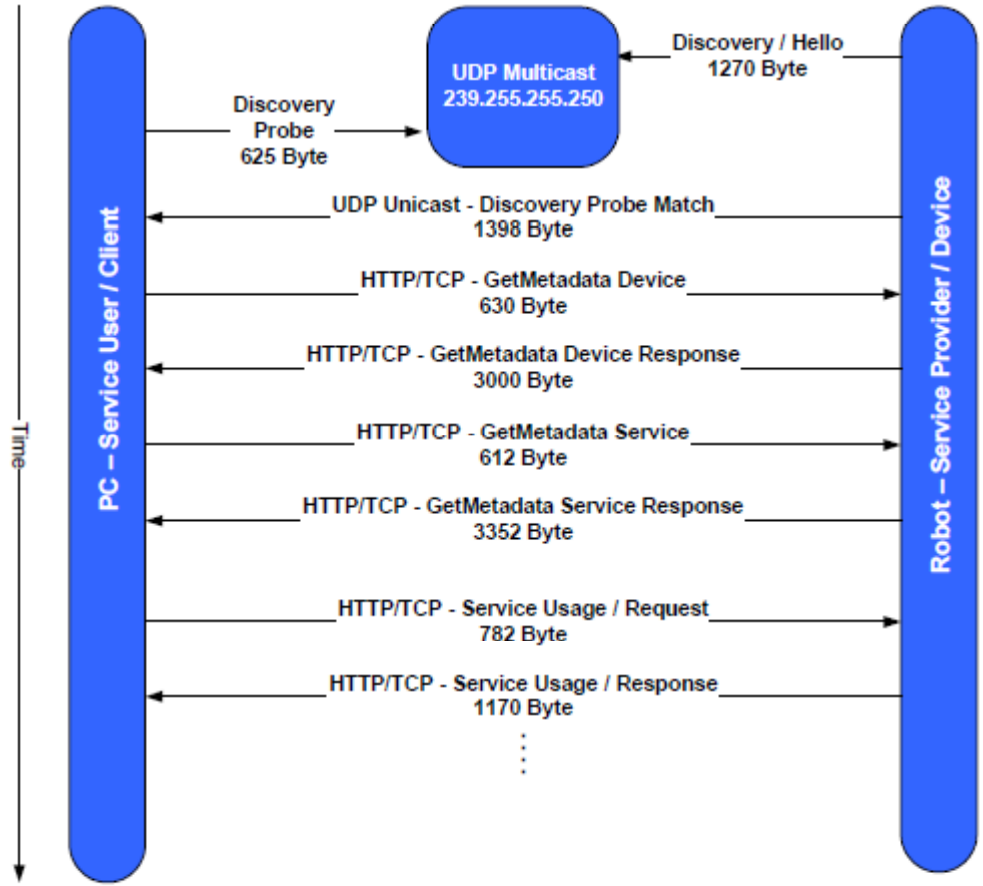
Context Aware Telematics



Real-time embedded systems



Interface messages



Service-oriented

- http://msdn.microsoft.com/en-us/library/aa480021.aspx
- **Service**
- A Component capable of performing a task. A WSDL service: A collection of end points (W3C).
- A type of capability described using WSDL (CBDI).
- **A Service Definition**
- A vehicle by which a consumer's need or want is satisfied according to a negotiated contract (implied or explicit) which includes Service Agreement, Function Offered and so on (CBDI).

Web service

- A software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a format that machines can process (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with XML serialization in conjunction with other Web-related standards (W3C).
- A programmatic interface to a capability that is in conformance with WSnn protocols (CBDI).

Service oriented architecture

- A set of components which can be invoked, and whose interface descriptions can be published and discovered (W3C).
- *The policies, practices, frameworks that 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. (CBDI)*

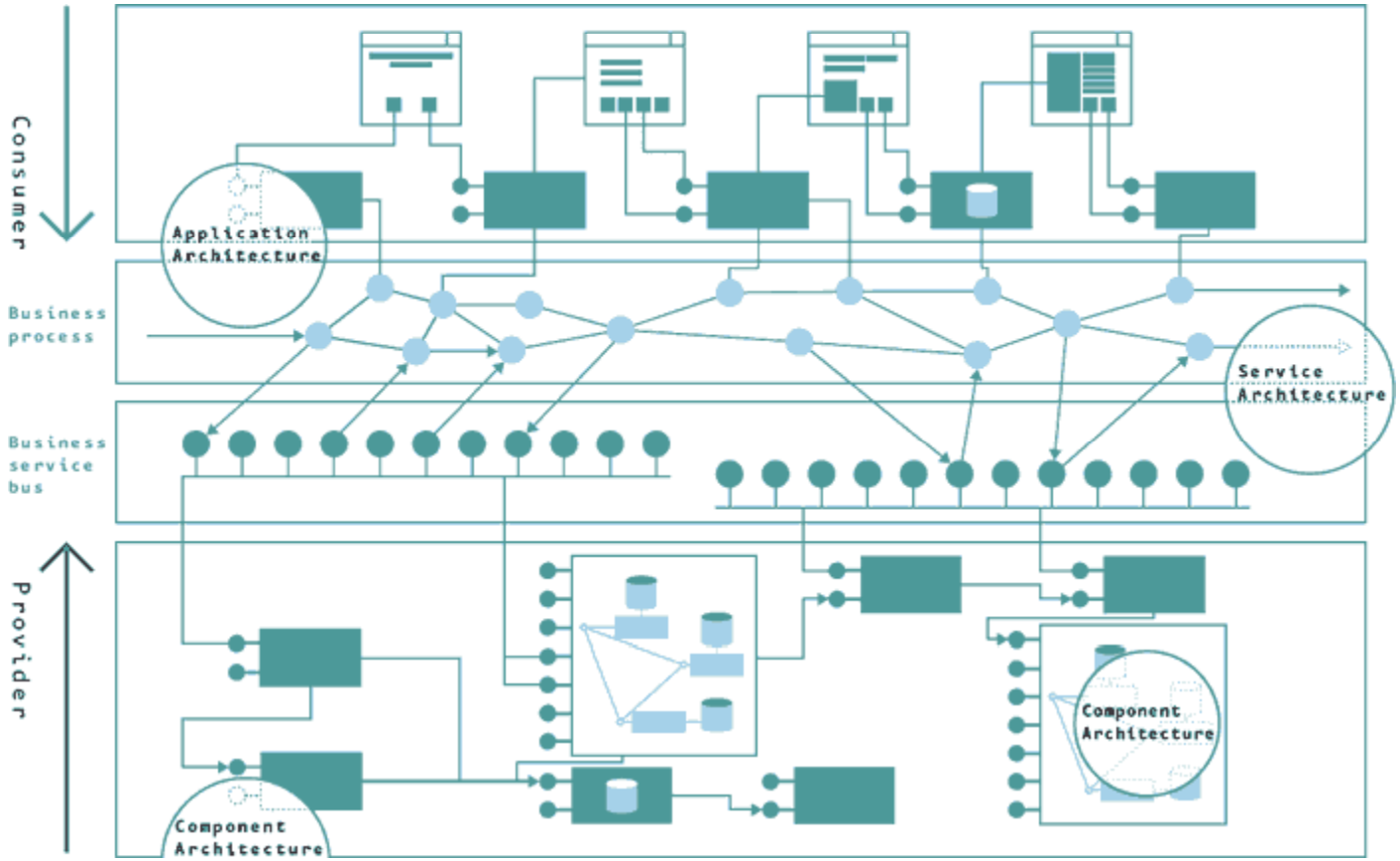
SOA Reference Architecture

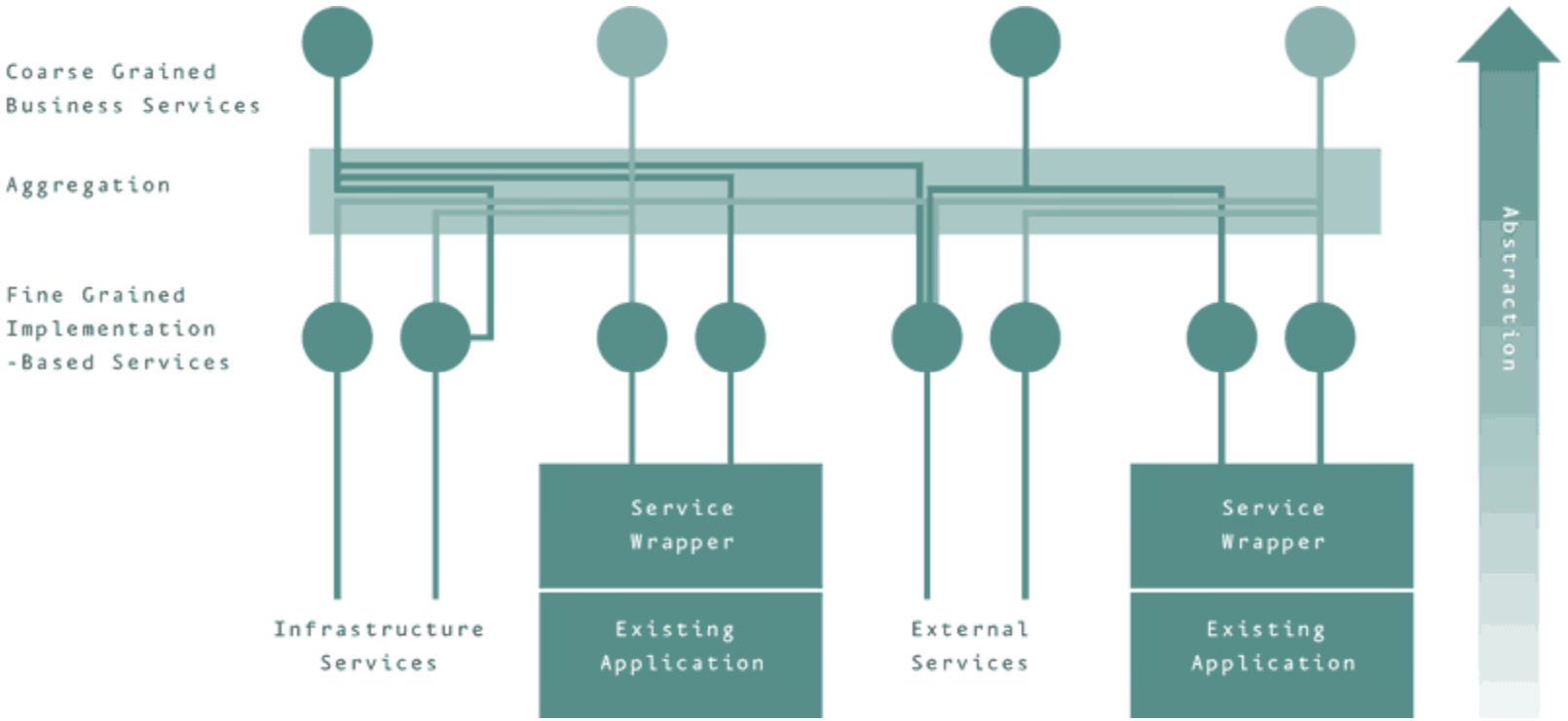
- Reference architecture – an abstraction from the software architecture
- http://docs.oasis-open.org/soa-rm/v1.0/soa-rm.pdf

Trust in software architecture

- Defense-in-depth – every component is secure and robust
- Data aggregation – minimize data that is outside of the secure system
- User-defined privacy policies – users are able to define what policies they wish to apply to their data

- http://books.google.com/books?id=OxnTlhm99A8C&pg=PA23&lpg=PA23&dq=service+oriented+architecture+in-vehicle+telematics&source=bl&ots=BRsTjItSRn&sig=BBSdixgLonKAGbZQgNNOe2WS5RU&hl=en&sa=X&ei=LXoEUfCsGI2M0QGp_4GQDg&ved=0CEkQ6AEwATgK#v=onepage&q=service%20oriented%20architecture%20in-vehicle%20telematics&f=false


Consumer
Application Architecture
Business process
Service Architecture
Business service bus
Provider
Component Architecture
Component Architecture


Coarse Grained
Business Services
Aggregation
Fine Grained
Implementation
-Based Services
Infrastructure
Services
Service
Wrapper
Existing
Application
External
Services
Service
Wrapper
Existing
Application
Abstraction

Qualities

- **Enabled by Web services**
  - *Technology neutral* Endpoint platform independence.
  - *Standardized* Standards-based protocols.
  - *Consumable* Enabling automated discovery and usage.

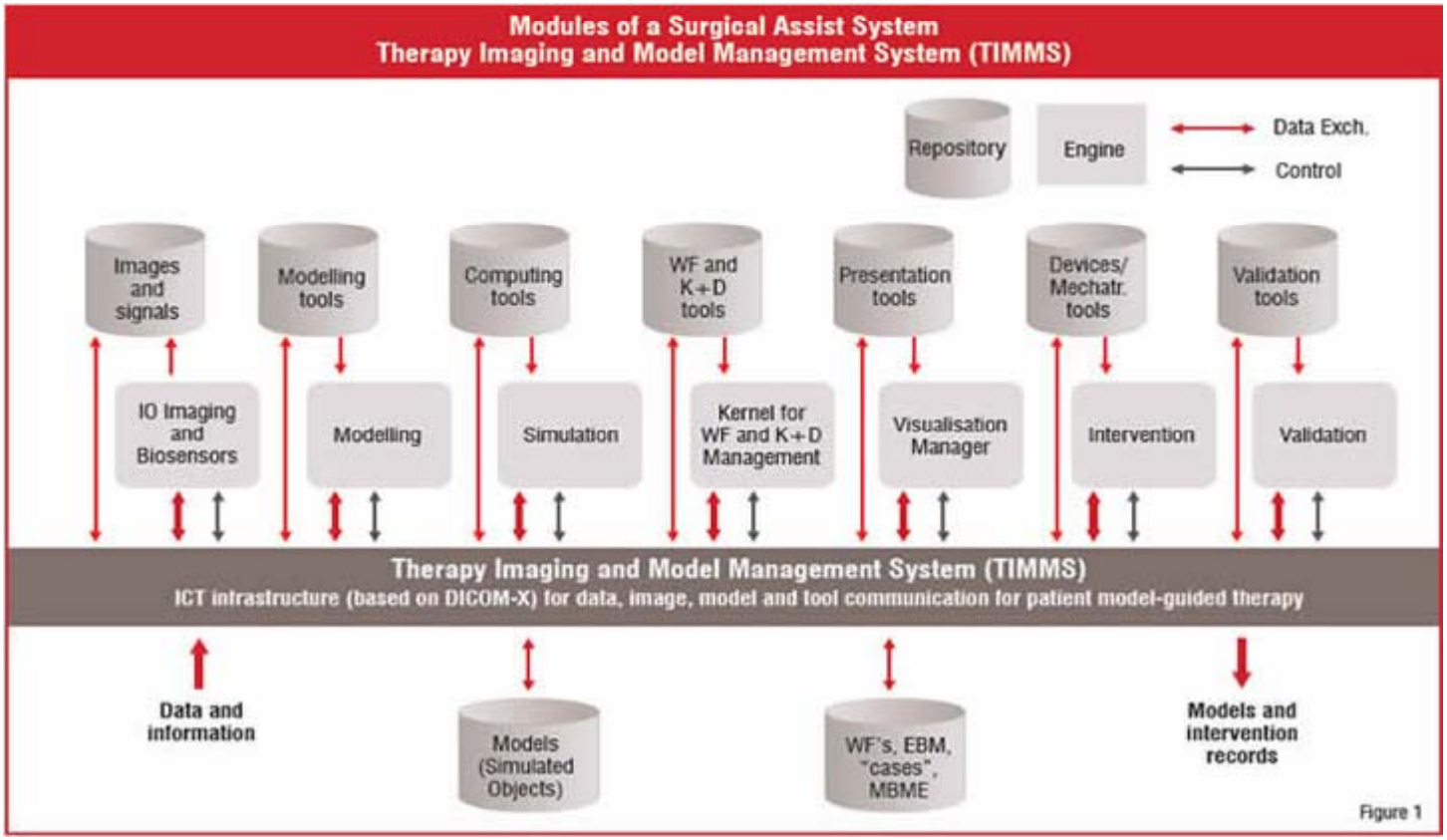
Qualities

- **Enabled by SOA**
  - *Reusable* Use of Service, not reuse by copying of code/implementation.
  - *Abstracted* Service is abstracted from the implementation.
  - *Published* Precise, published specification functionality of service interface, not implementation.
  - *Formal* Formal contract between endpoints places obligations on provider and consumer.
  - *Relevant* Functionality presented at a granularity recognized by the user as a meaningful service.

Benefits

- **There is real synchronization between the business and IT implementation perspective.**
- **A well formed service provides us with a unit of management that relates to business usage.**
- **When the service is abstracted from the implementation it is possible to consider various alternative options for delivery and collaboration models.**

Data and Events are passed

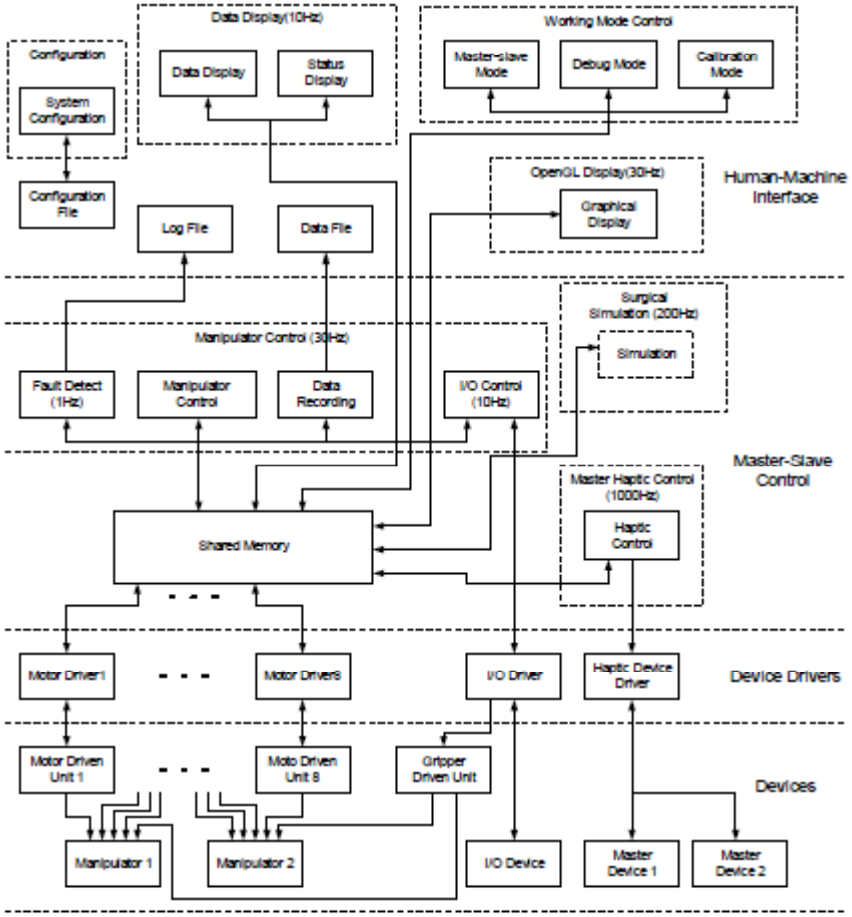


Extended examples

- http://www.health.state.mn.us/divs/istm/architecture.pdf

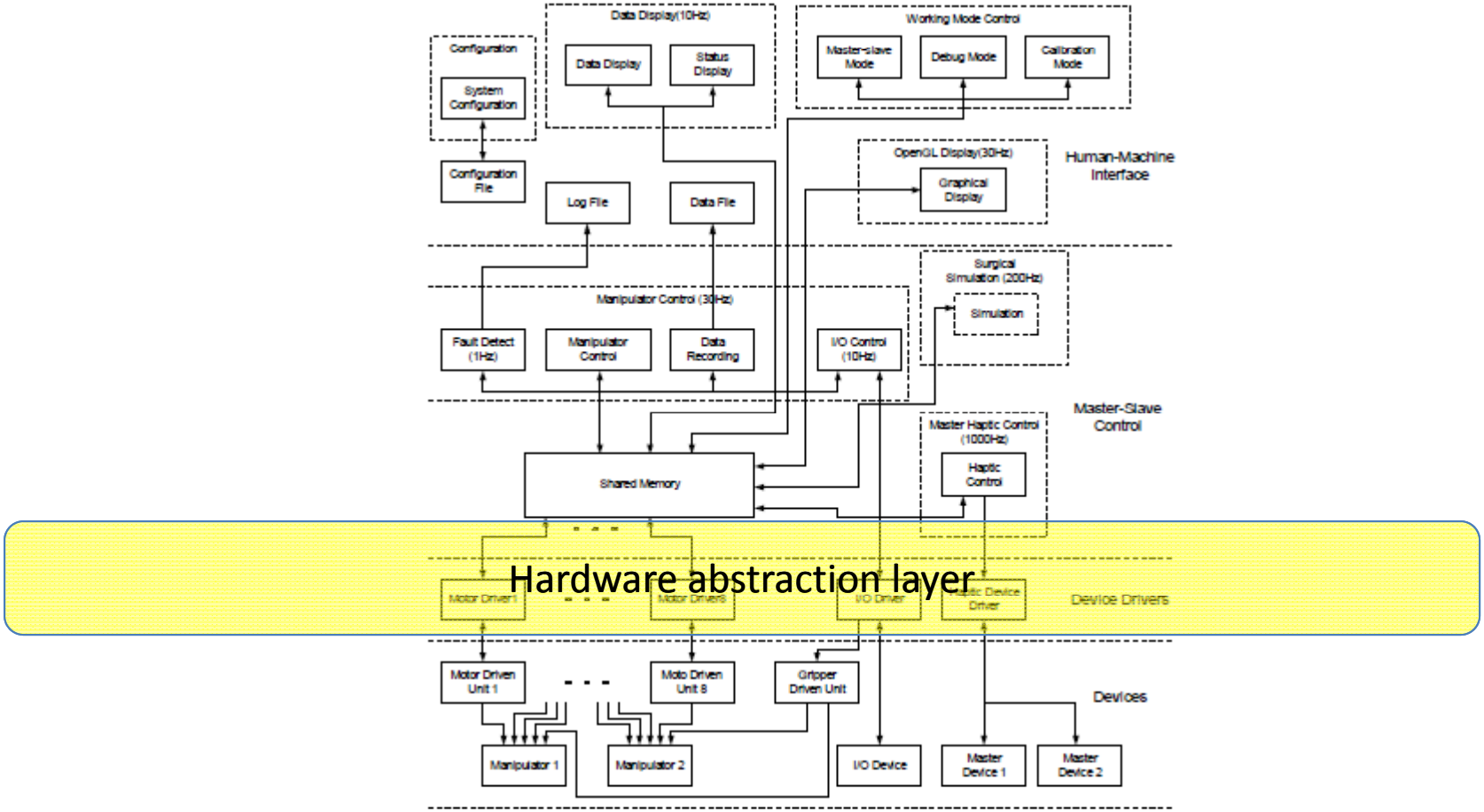
- Decompose
- Integrate
  - Individual hardware pieces are associated with drivers
  - The drivers feed applications
  - The applications are tied together by a user interface

Decompose into modules

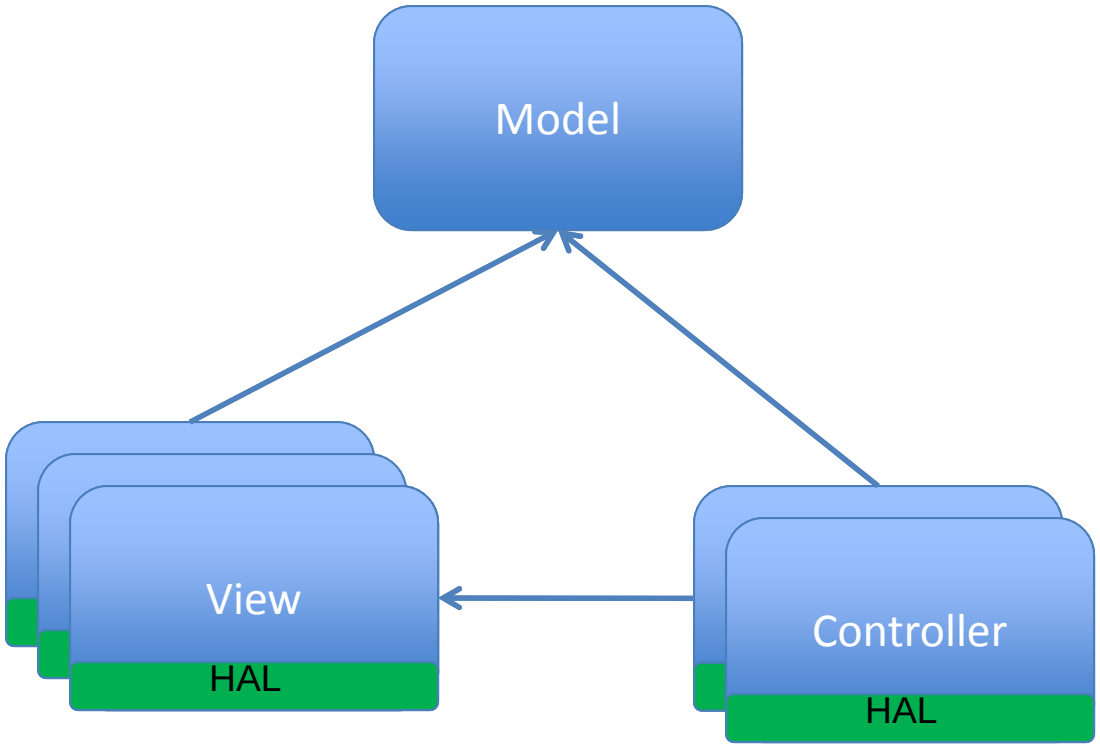


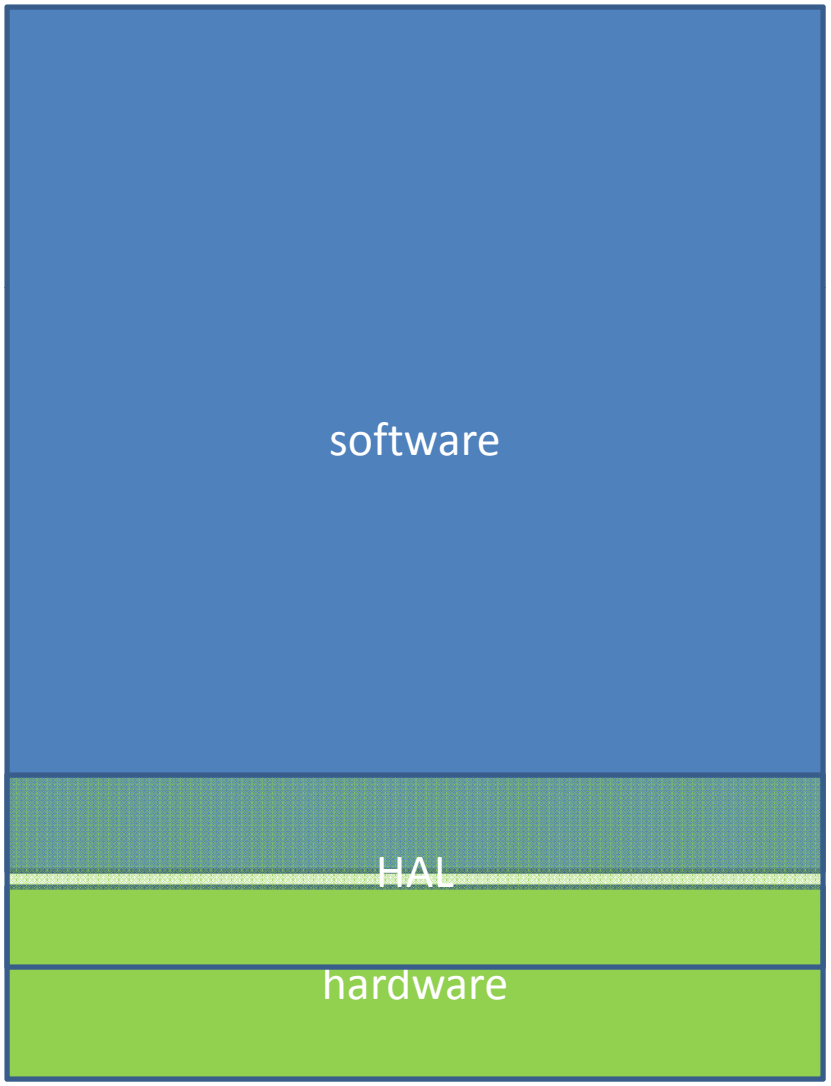
Hardware abstraction layer

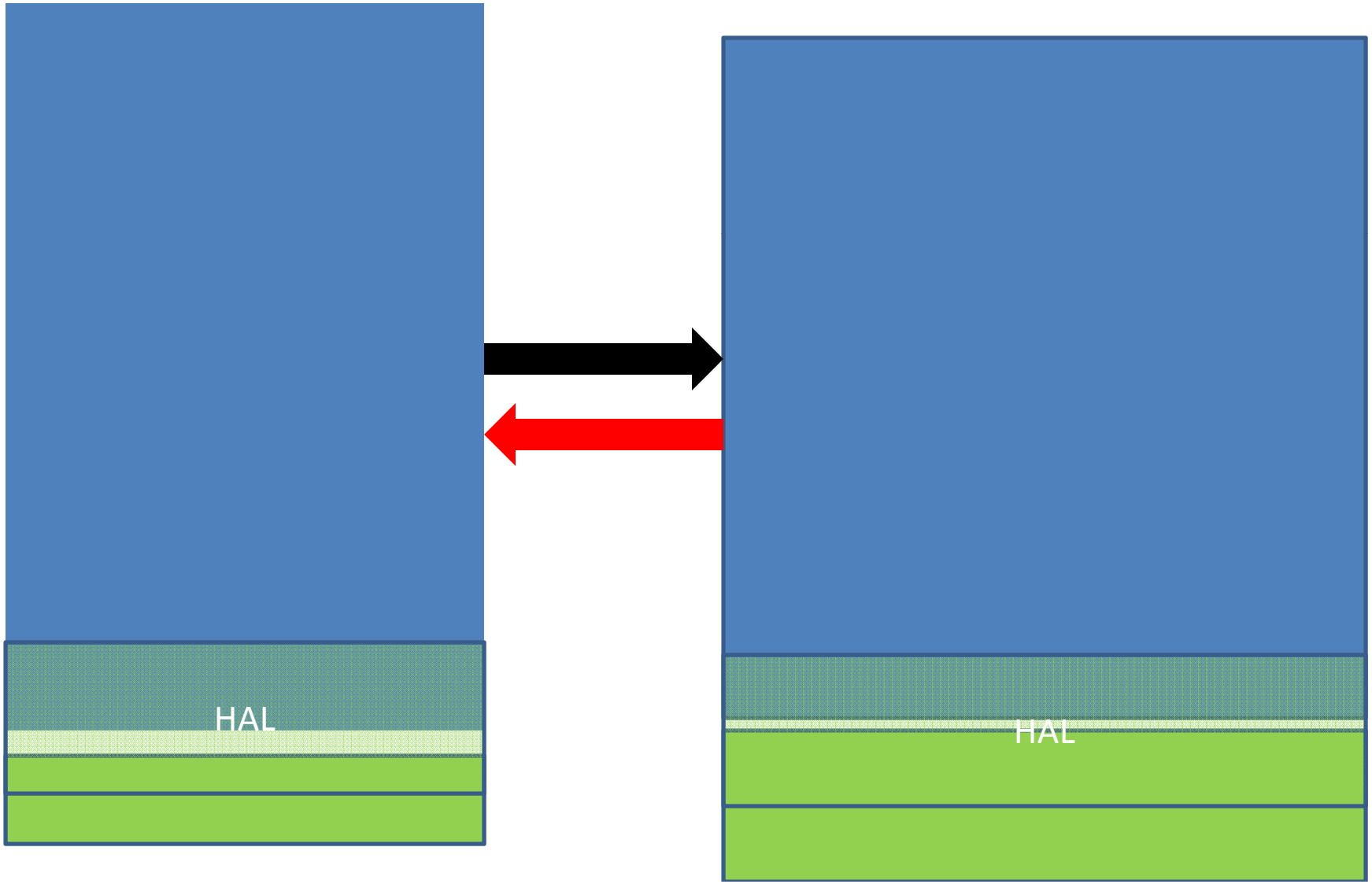
- The specific hardware is hidden from the software.
- The layer acts as an API
- An operating system usually includes a layer
- Making the API from standards allows the underlying device to be commoditized


Data Display(10Hz)
Data Display
Status Display
Working Mode Control
Master-slave Mode
Debug Mode
Calibration Mode
Configuration
System Configuration
Configuration File
Log File
Data File
OpenGL Display(30Hz)
Graphical Display
Human-Machine Interface
Surgical Simulation (200Hz)
Simulation
Manipulator Control (30Hz)
Fault Detect (1Hz)
Manipulator Control
Data Recording
I/O Control (10Hz)
Master-Slave Control
Master Haptic Control (1000Hz)
Haptic Control
Shared Memory
Hardware abstraction layer
Motor Driver1
Motor Drivers
I/O Driver
Haptic Device Driver
Device Drivers
Motor Driven Unit 1
Moto Driven Unit 8
Gripper Driven Unit
Devices
Manipulator 1
Manipulator 2
I/O Device
Master Device 1
Master Device 2

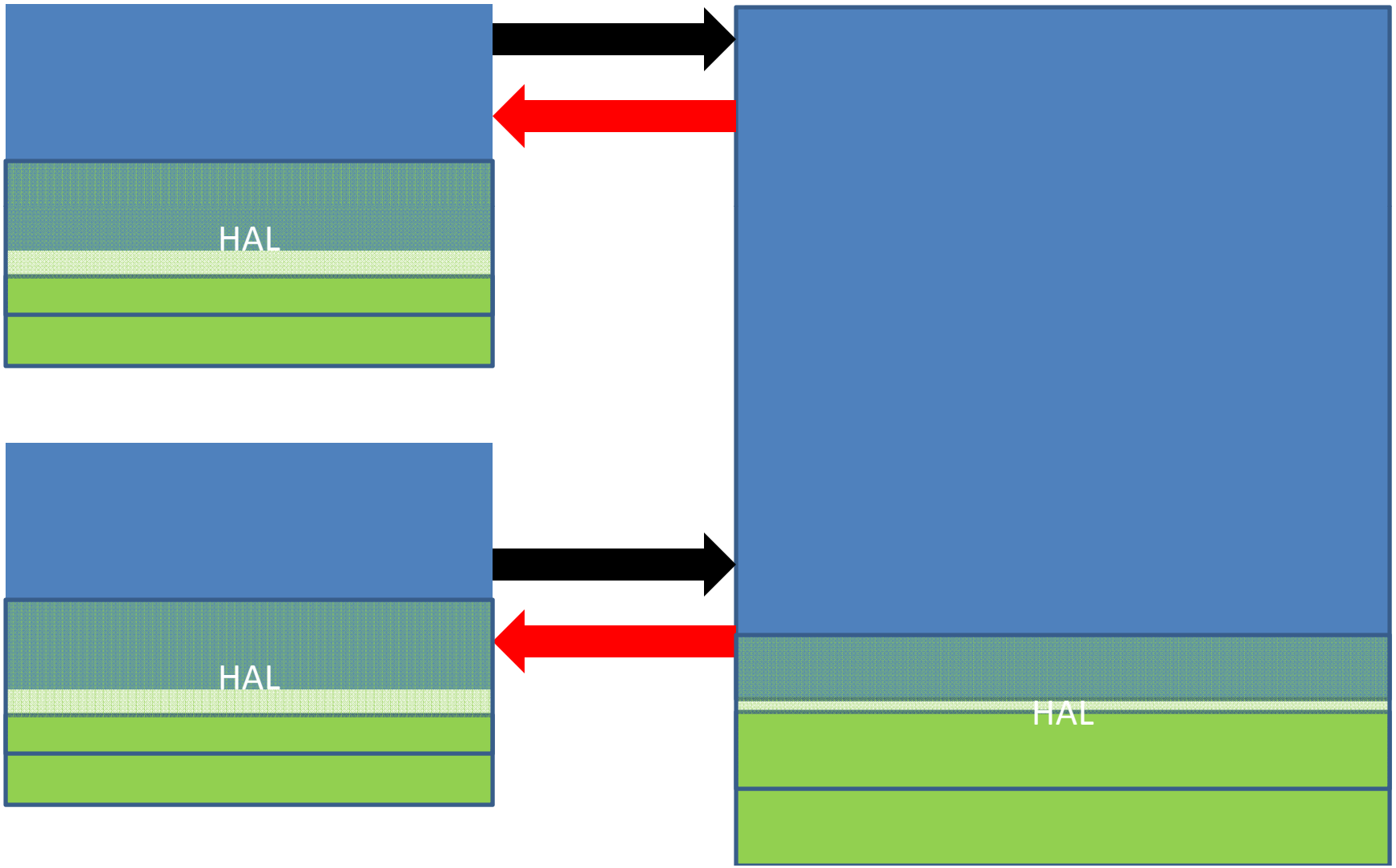
MVC



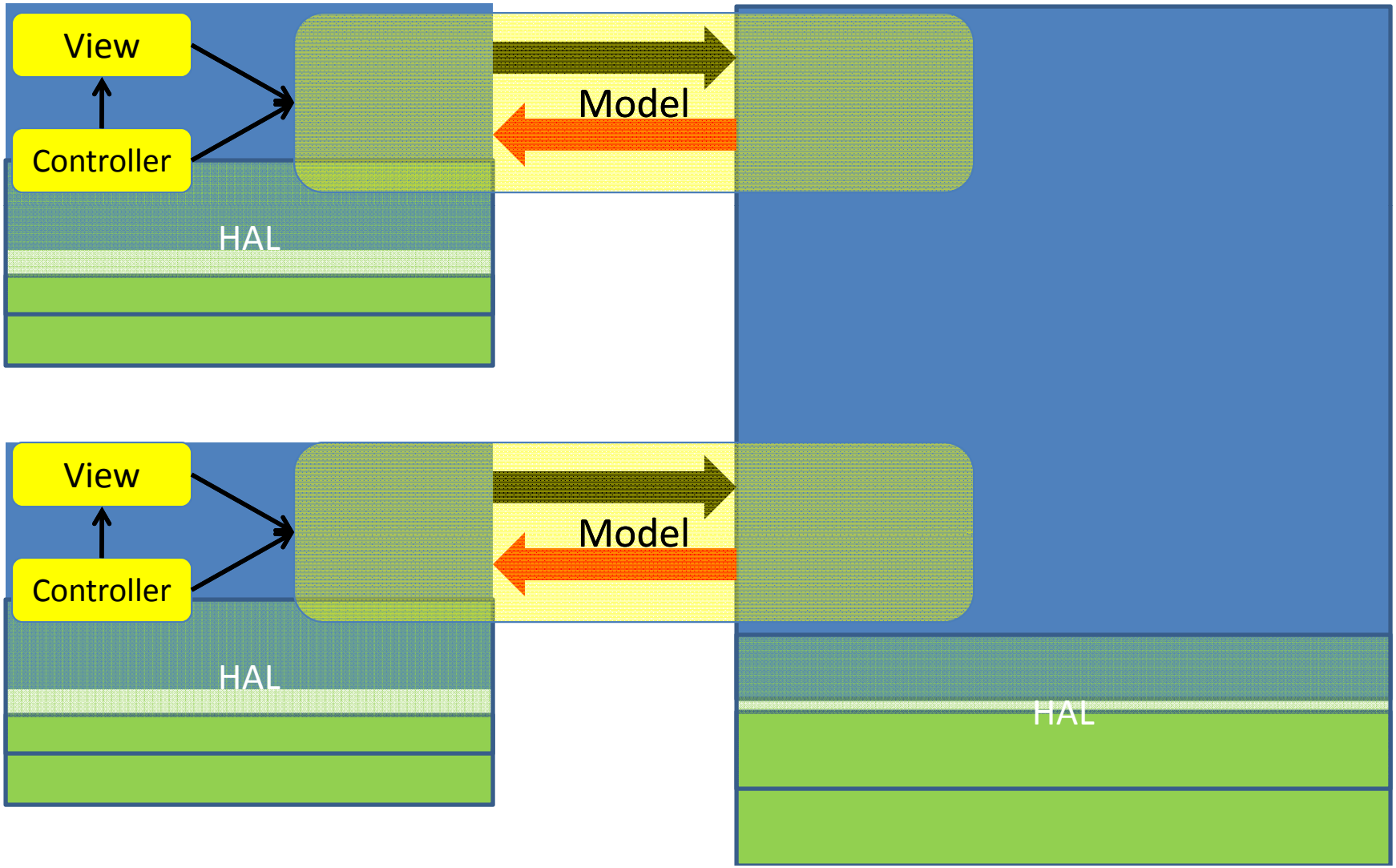

software
HAL
hardware


HAL
HAL

C/S



Integration styles

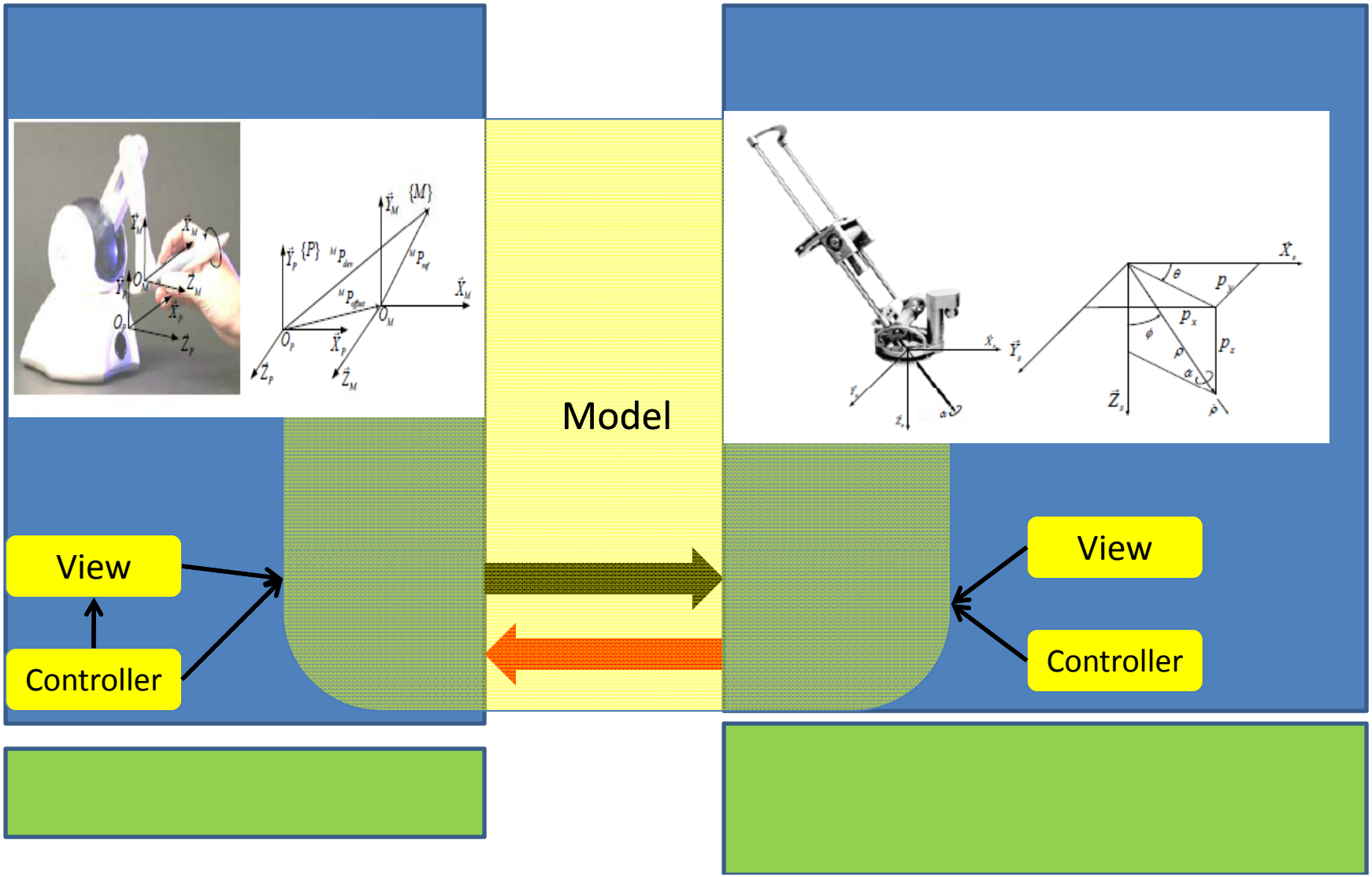


So what do we have now?

- Correct
- Reliable
- …

Any conflicts?

Master/slave



Step 4: Choose a Design Concept That Satisfies the Architectural Drivers

- Styles and patterns filtered by qualities
- When do you use …

Driver	Pattern
Efficiency	Pipe/filter
Modifiability	Layer
Flexibility	MVC
Security	Client/server

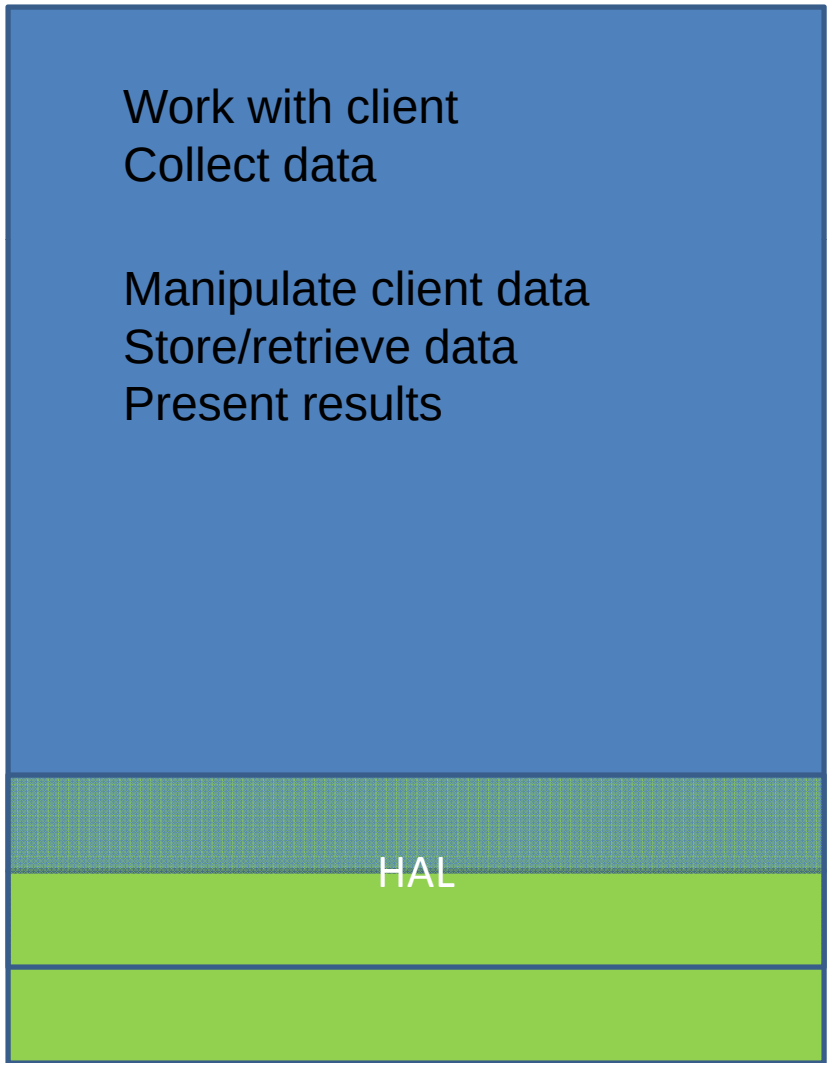
- Keep a table of these

Step 5: Instantiate Architectural Elements and Allocate Responsibilities

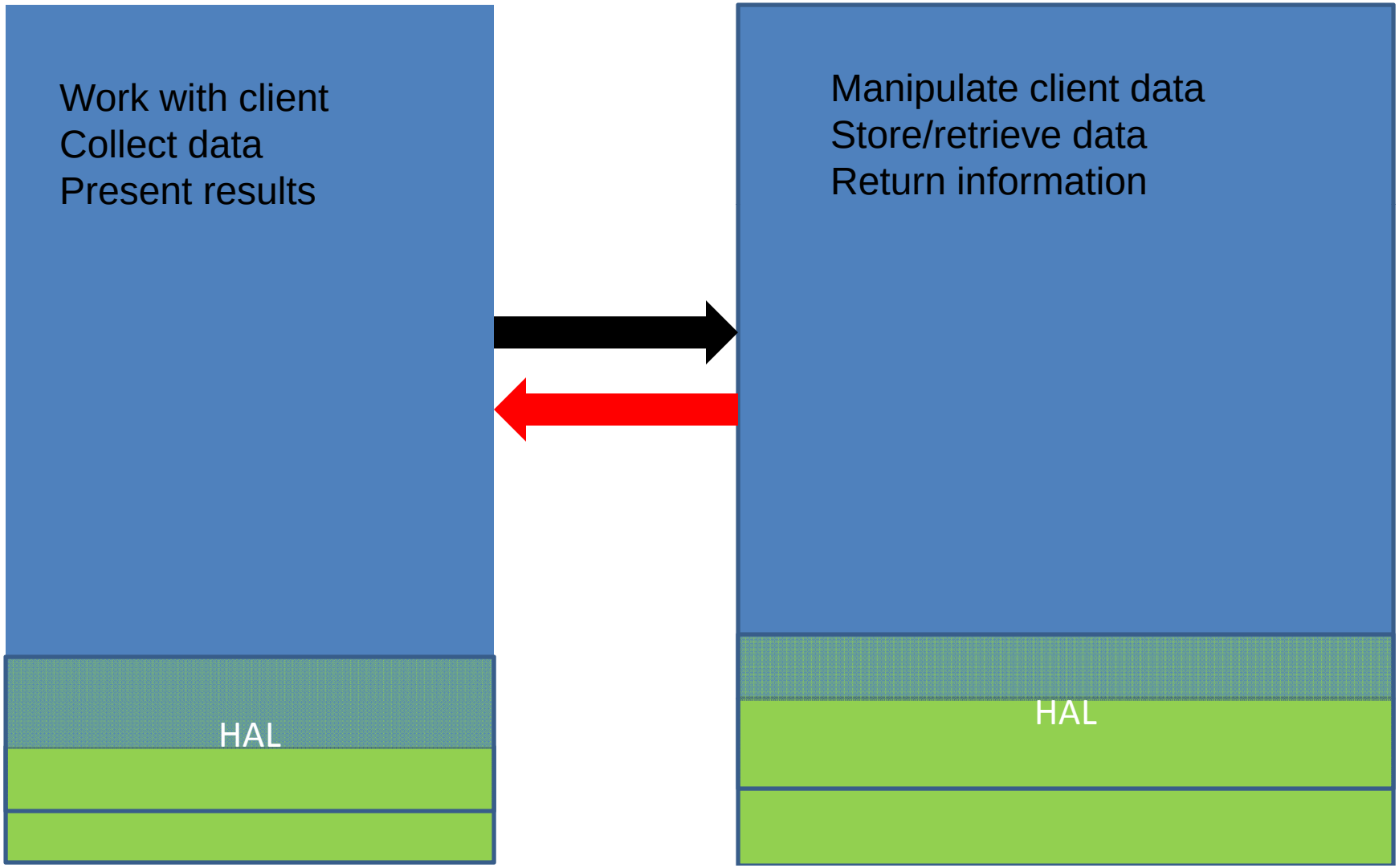
We begin with the monolith and all of the uses of the system
(Why the uses and not the requirements?)

When we decompose the monolith we also decompose the responsibilities

We also add new responsibilities from splitting some responsibilities.



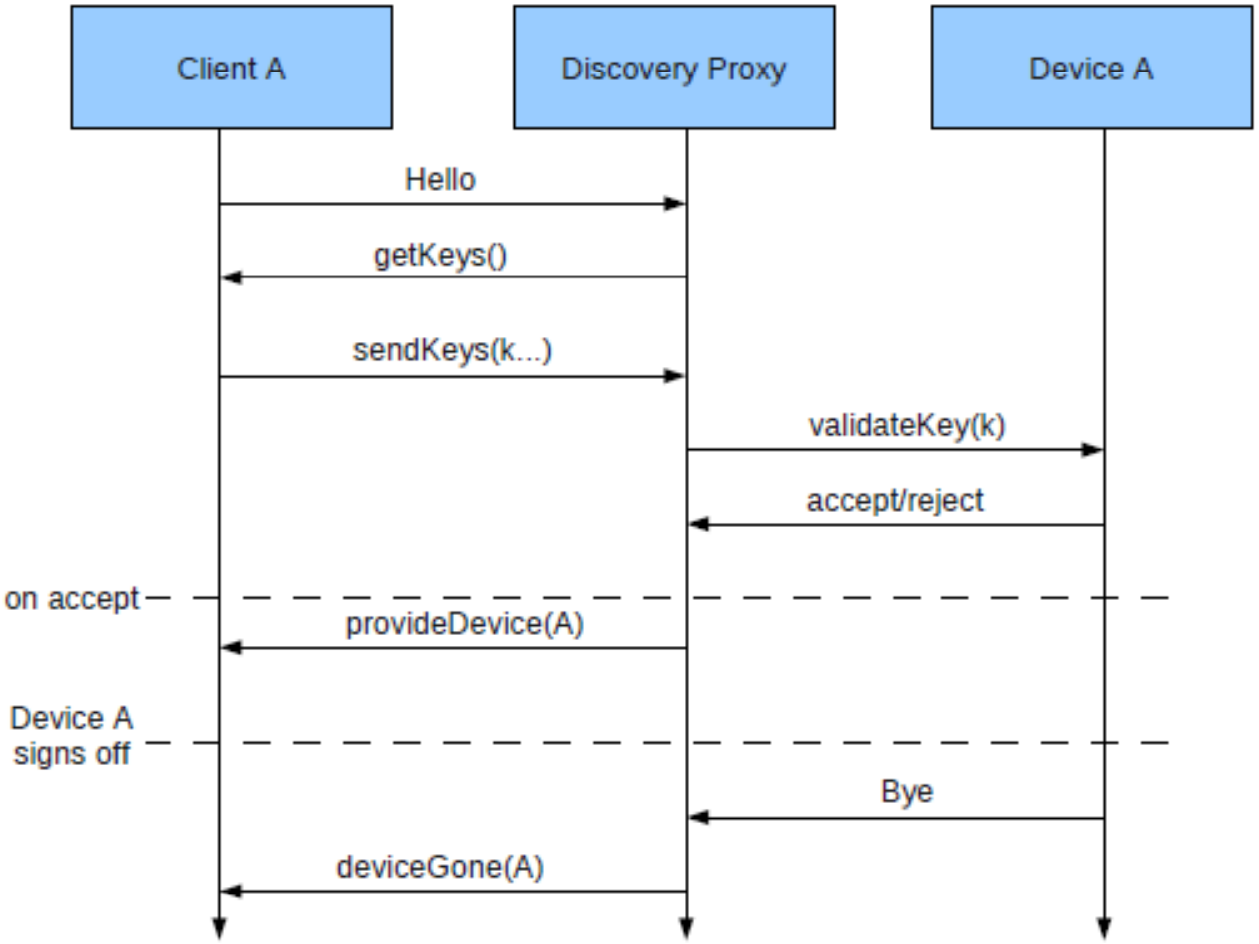
Step 5: Instantiate Architectural Elements and Allocate Responsibilities



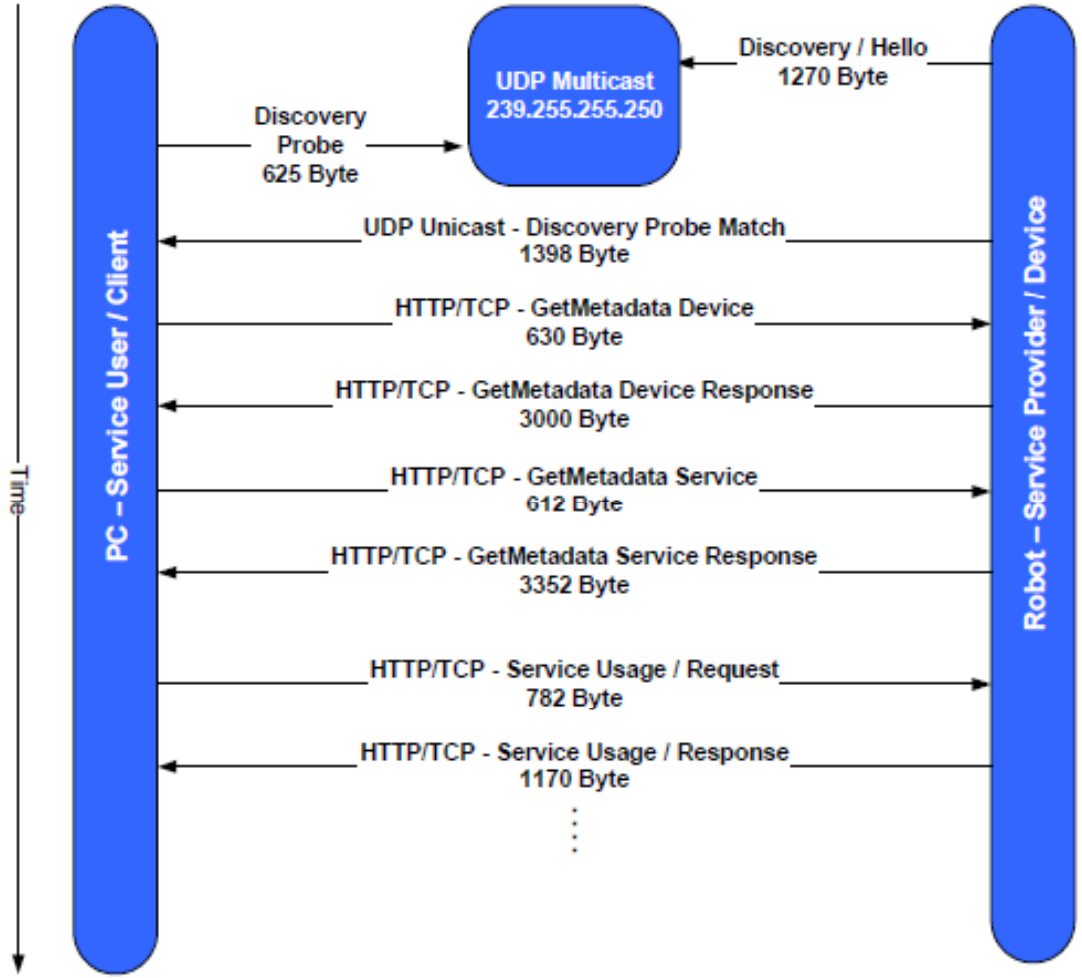
Step 6: Define Interfaces for Instantiated Elements

- Start with all the requirements
- What does each module need from others and what does it produce?
- Requires:
- Provides:

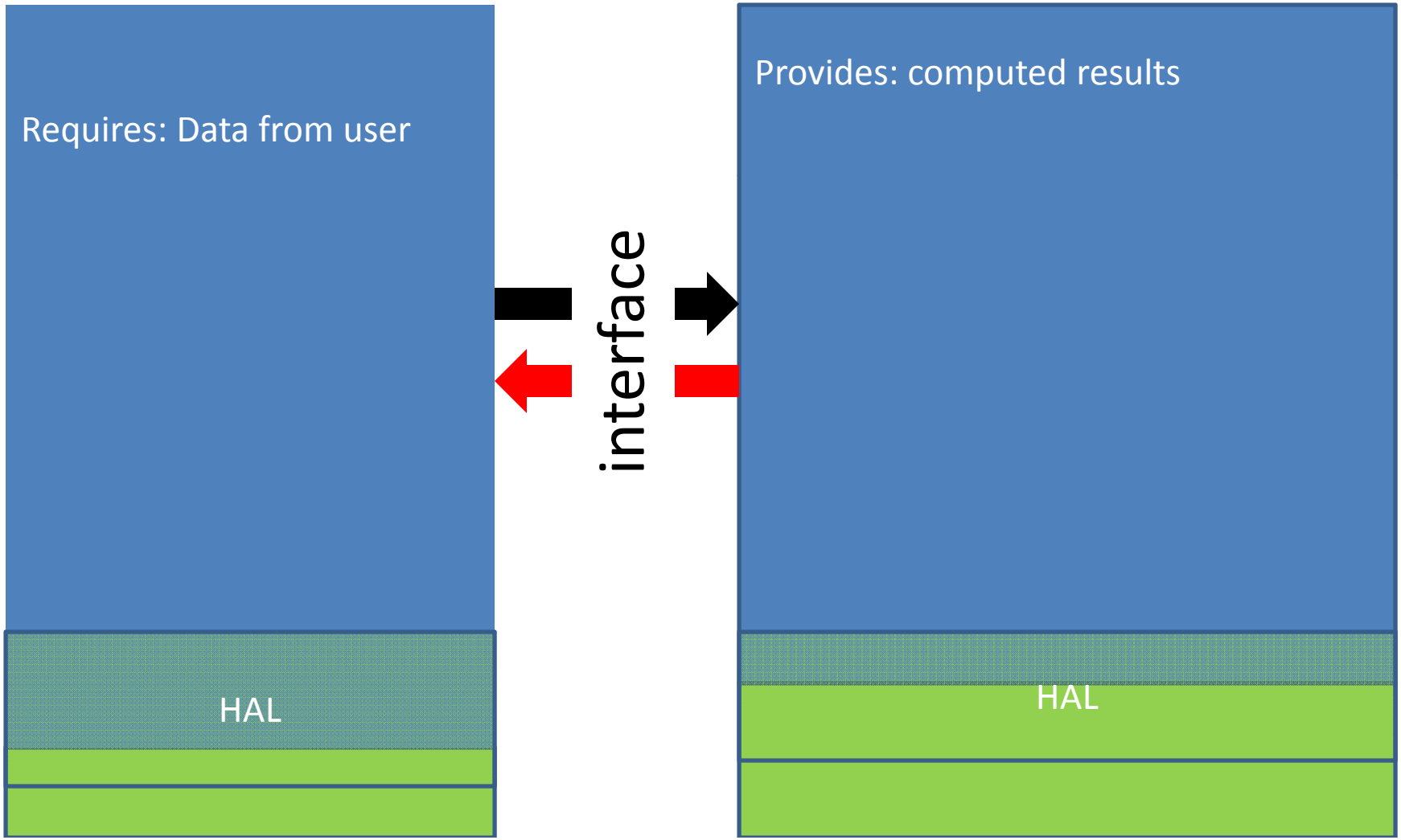
Registration



Specification



Step 6: Define Interfaces for Instantiated Elements



What do we need?

- Languages for specifying the interfaces

Here's what you are going to do…

- Do a first level decomposition of your architecture
- Capture that decomp in AADL
- Submit the *.aadl files by Monday Feb 4th by 11:59pm.