Architecture Documentation

Architecture Documentation

- Larman
  - Chapter 39
- Bass/Clements/Kazman
  - Chapter 18

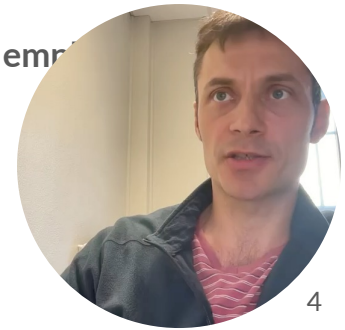
Software Architecture Document (Larman)

- Once an architecture takes shape, it may be useful to describe it, so that new developers can learn the big ideas of the system, or so that there is a common view from which to discuss changes
- When someone joins the development team, it's useful if the project coach can say, "Welcome to the NextGen project! Please go to the project website and read the ten page SAD in order to get an introduction to the big ideas."
- Quickly help someone understand the major ideas in this system
- Software Architecture Document (SAD)

Architectural Views

Having an architecture is one thing; a useful description is something else.

- A view of the system **architecture from a given perspective**; it focuses **primarily on structure, modularity, essential components, and the main control flows**. [Rational Unified Process].
- An important aspect of the view missing from this Rational Unified Process definition is the **motivation**. That is, an architectural view should explain why the architecture is the way it is.
- An architectural view is a window onto the system from a particular **perspective that emp** the **key noteworthy information or ideas**, and ignores the rest.

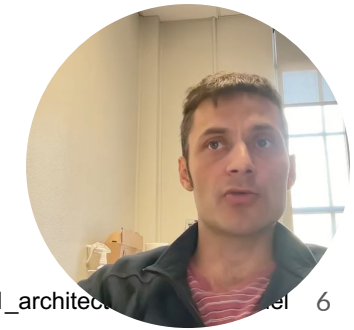


Architectural Views

- A key idea of the architectural views which concretely are text and diagrams that they do not describe all of the system from some perspective,
  - only outstanding ideas from that perspective.
- Architectural views may be created:
  - after the system is built, as a summary
  - at the end of certain iteration milestones
  - speculatively, during early iterations, as an aid in creative design work, recognizing that the original view

The N+1 / 4+1 View Model

- Core views:
  - [Larman] Logical, process, deployment, and data.
  - [Other sources] Logical, Process, Development, Physical (topology of software components)
- The ‘N+1' view
  - is the use case view, a summary of the most architecturally significant use cases or scenarios



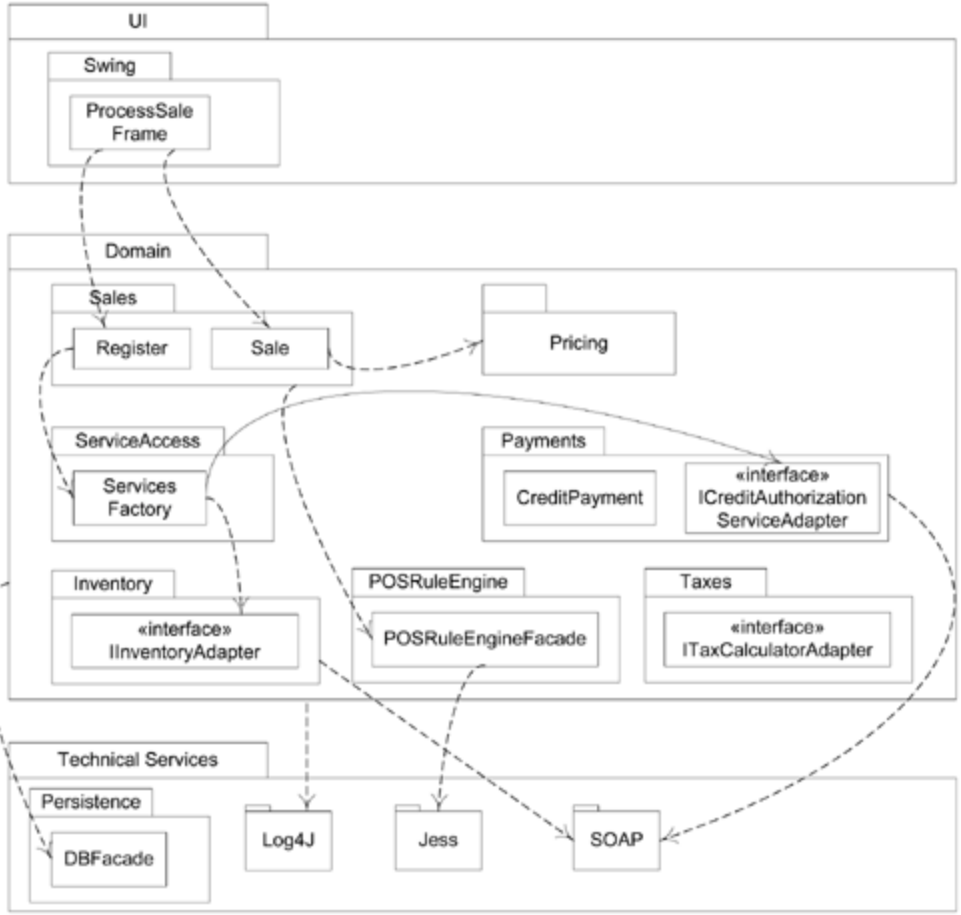
https://en.wikipedia.org/wiki/4%2B1_architect...el

Logical View

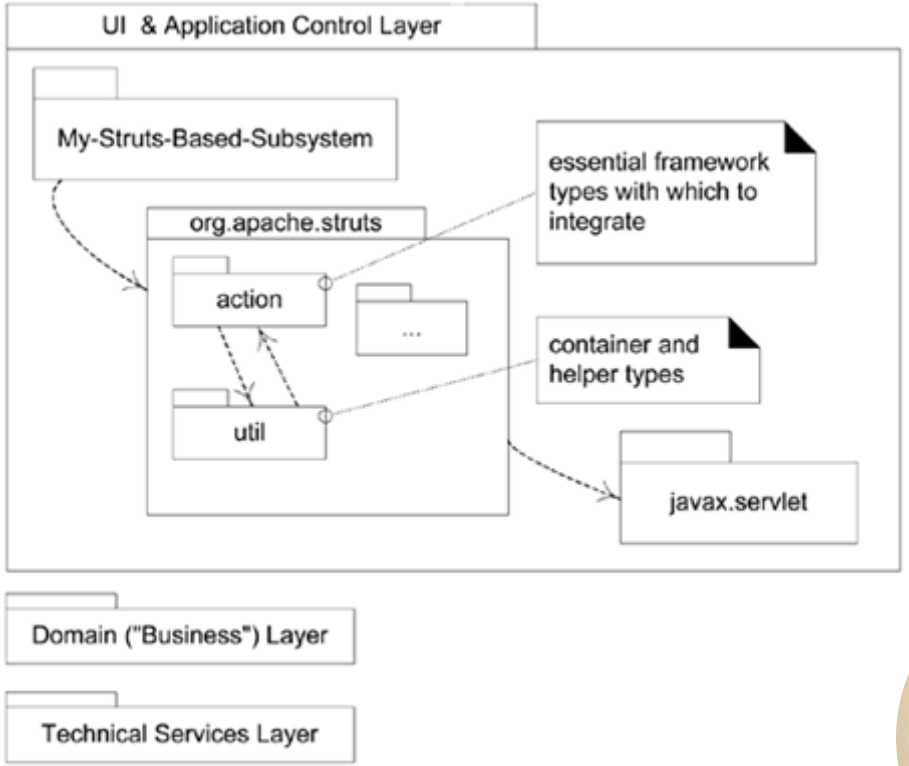
- Conceptual organization of the software in terms of the most important layers, subsystems, packages, frameworks, classes, and interfaces. Also summarizes the functionality of the major software elements, such as each subsystem.
- Shows outstanding use-case realization scenarios (as interaction diagrams) that illustrate key aspects of the system.
- A view onto the Unified Process Design Model, visualized with UML package, class, and interaction diagrams.

... other technical memos ...

Logical View



Framework


UI & Application Control Layer
My-Struts-Based-Subsystem
essential framework types with which to integrate
org.apache.struts
action
...
container and helper types
util
javax.servlet
Domain ("Business") Layer
Technical Services Layer

Process View

- Processes and threads. Their responsibilities, collaborations, and the allocation of logical elements (layers, subsystems, classes, ...) to them.

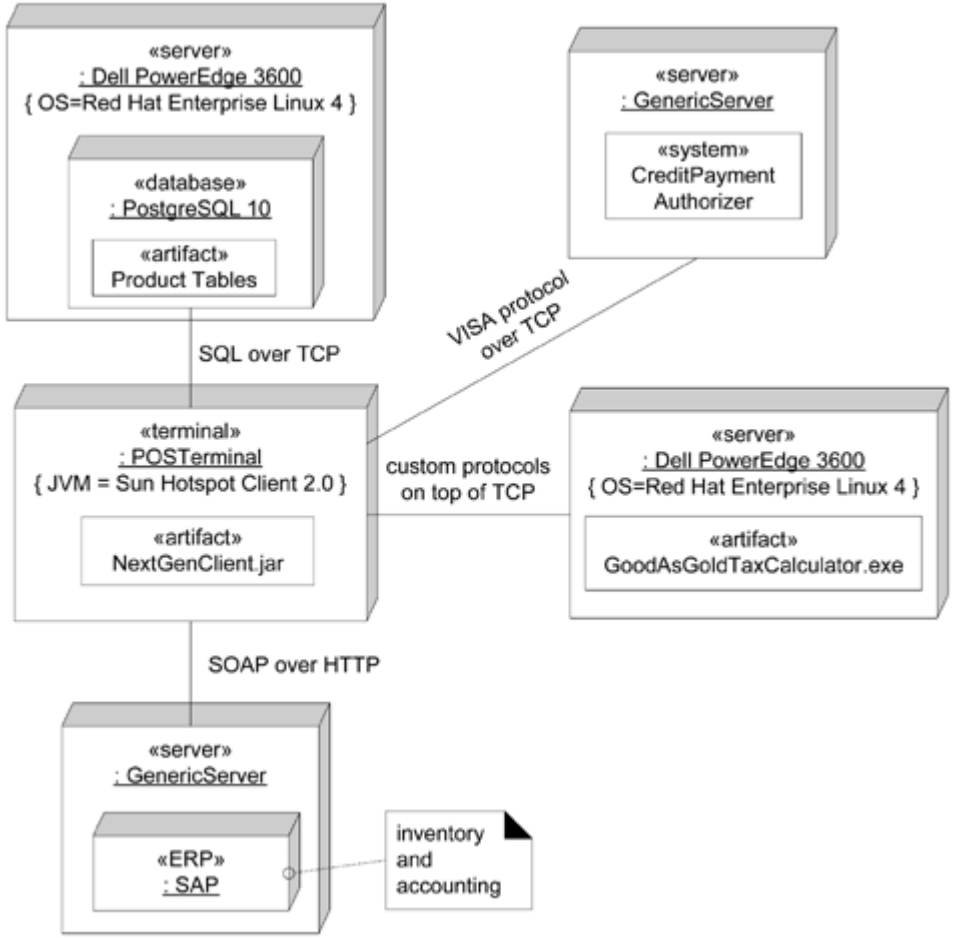
- A view onto the UP Design Model, visualized with UML class and interaction diagrams, using the UML process and thread notation.

Deployment View

- Physical deployment of processes and components to processing nodes, and the physical network configuration between nodes.

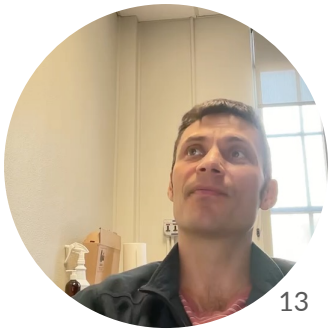
- A view onto the Deployment Model, visualized with UML deployment diagrams. Normally, the "view" is simply the entire model rather than a subset, as all of it is noteworthy.

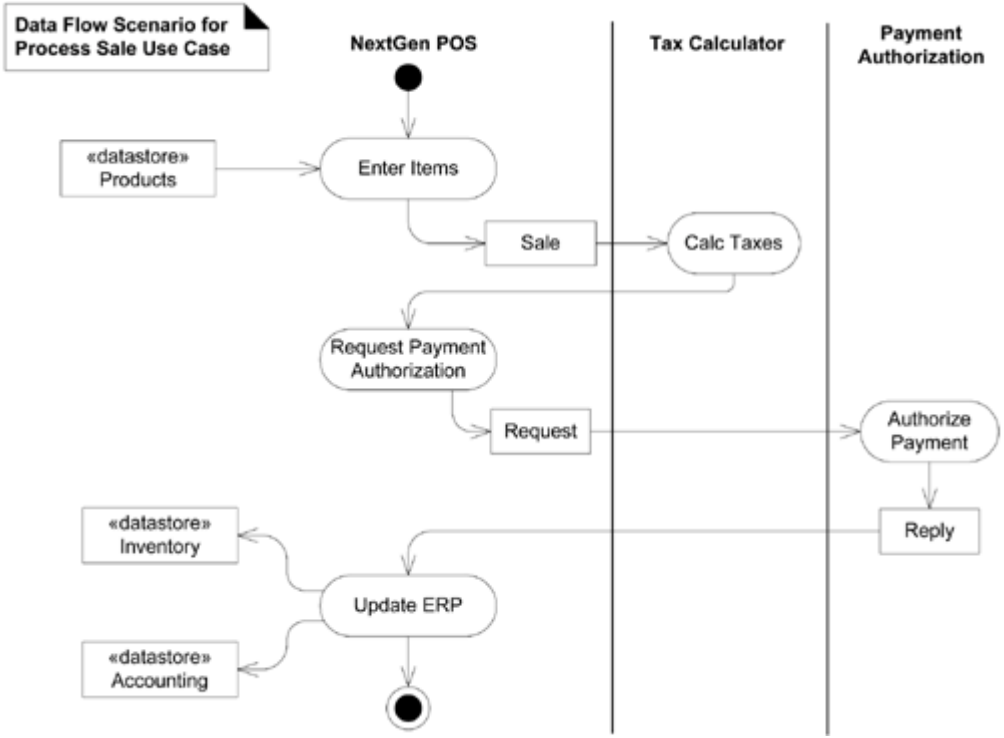
Deployment View



Data View

- Overview of the data flows, persistent data schema, the schema mapping from objects to persistent data (usually in a relational database), the mechanism of mapping from objects to a database, database stored procedures and triggers.
- In part, a view onto the Data Model, visualized with UML class diagrams used to describe a data model.
- Data flows can be shown with UML activity diagrams.

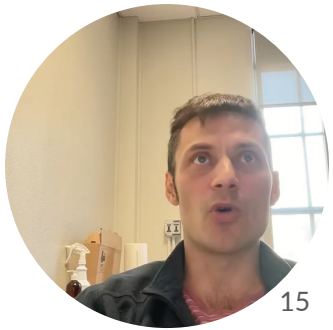



Data Flow Scenario for Process Sale Use Case
NextGen POS
Tax Calculator
Payment Authorization
«datastore» Products
Enter Items
Sale
Calc Taxes
Request Payment Authorization
Request
Authorize Payment
Reply
«datastore» Inventory
Update ERP
«datastore» Accounting

Security View

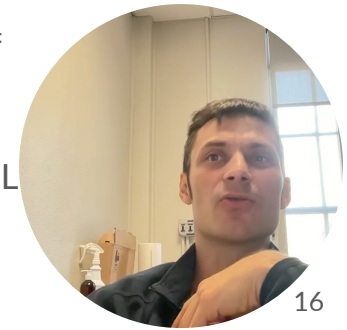
- Overview of the security schemes, and points within the architecture that security is applied, such as HTTP authentication, database authentication, and so forth.

- Could be a view onto the UP Deployment Model, visualized with UML deployment diagrams that highlight the key points of security, and related files.



Implementation View

- The actual source code, executables, and so forth.
  - Parts:
    - 1) deliverables,
    - 2) things that create deliverables (such as source code and graphics)
    - including Web pages, DLLs, executables, source code, and so forth, and their organization such as source code in Java packages, and bytecode organized into JAR files.
- The implementation view is a summary description of the noteworthy organization of deliverables and the things that create deliverables (such as the source code).
- A view onto the UP Implementation Model, expressed in text and visualized with UML and component diagrams.



Development View

- Summarizes information developers need to know about the setup of the development environment.

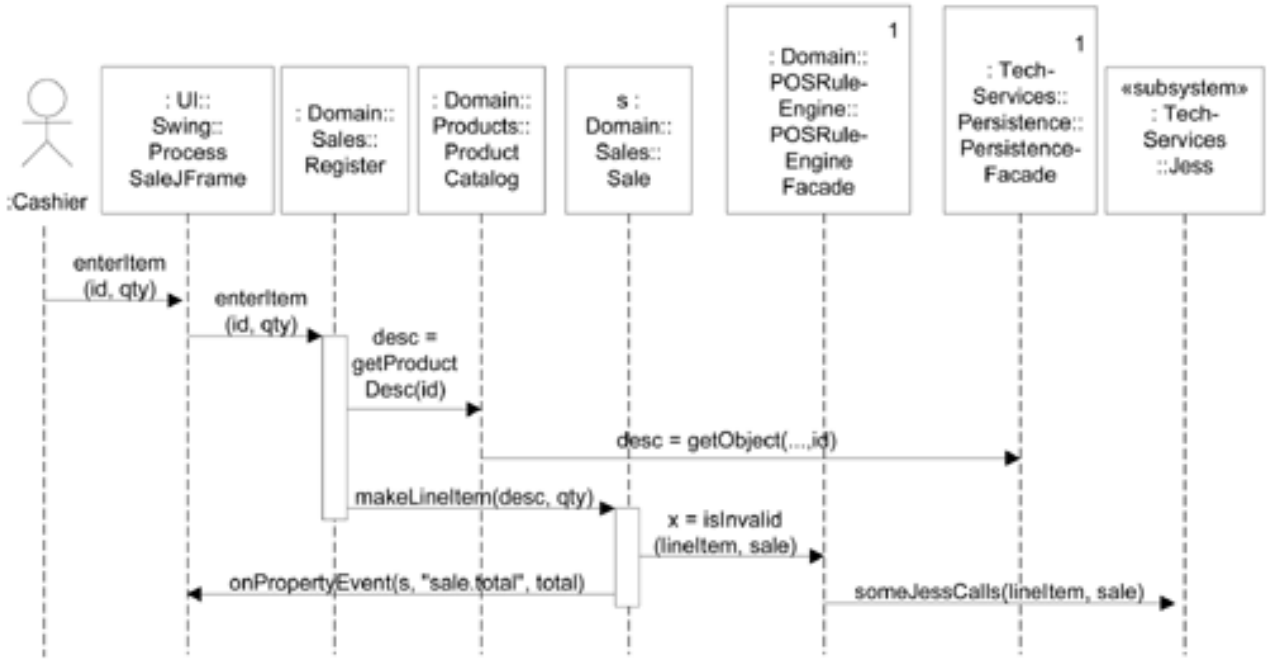
- For example,
  - How are all the files organized in terms of directories, and why?
  - How does a build and smoke test run?
  - How is version control used?

Use Case View

- Summary of the most architecturally significant use cases and their **non-functional requirements**.
  - Such use cases that, by their implementation, illustrate significant architectural coverage or that exercise many architectural elements.
    - For example, the Process Sale use case, when fully implemented, has these qualities.

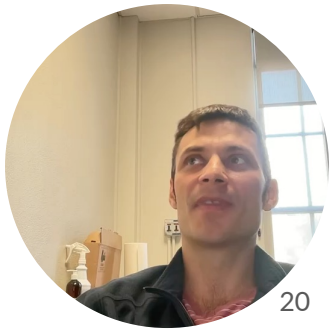
- A view onto the UP Use-Case Model, expressed in text and visualized with UML use case diagrams and perhaps with use-case realizations in UML interaction diagrams.

Use Case View



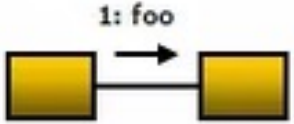
4+1 Architectural View Model (other sources)

- Logical View:
  - The functionality. The service.
- Process View:
  - Communication between processes and/or services.
- Physical View:
  - Deployment of your services.
- Development/Implementation View:
  - File/Folder Structure of your codebase. What you're looking at in your IDE/Editor



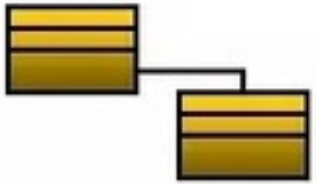
Process View

Diagrams:
- Sequence
- Communication

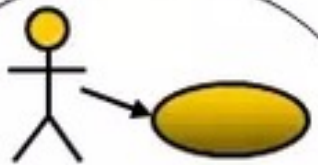


Logical View

Diagrams:
- Class
- Object
- Composite Structure



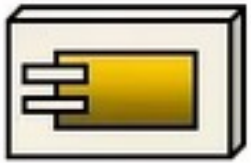
Conceptual / Logical



Scenario View
- Use Case
- User Stories

Implementation View

Diagrams:
- Component
- Package



Physical View

Diagrams:
- Deployment
- Nettwork Topology (not UML)

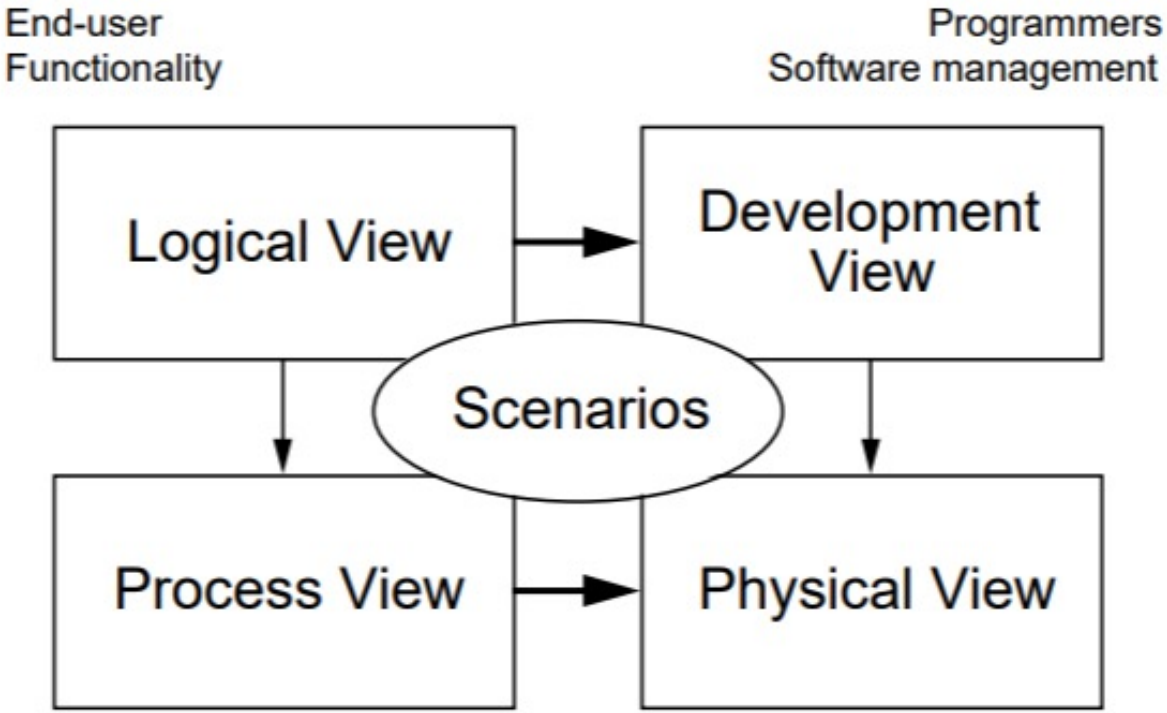


Figure 1 — The “4+1” view model

Architectural Factors

- Any and all of the FURPS+ requirements may have a significant influence on the architecture of a system, ranging from reliability, to schedule, to skills, and to cost constraints.
- For example, a case of tight schedule with limited skills and sufficient money probably favors buying or outsourcing to specialists, rather than building all components in-house.

- However, the factors with the strongest architectural influence tend to be within the high-level FURPS+ categories of functionality, reliability, performance, supportability, implementation interface.
- Interestingly, it is usually the non-functional quality attributes (such as reliability or perf that give a particular architecture its unique flavor, rather than its functional requireme

- Architecturally significant requirements

Software Architecture Document - structure

- Architectural Representation
  - Summary of how the architecture will be described in SAD,
    - Technical memos and the architectural views.
    - **Technical memo**: why did we choose this direction rather than another
      - records architectural decisions

- Architectural Factors
  - Reference to the Supplementary Specification to view the Factor Table.
- Architectural Decisions
  - The set of technical memos that summarize the decisions.
- List of views with motivation
  - Not all views necessary

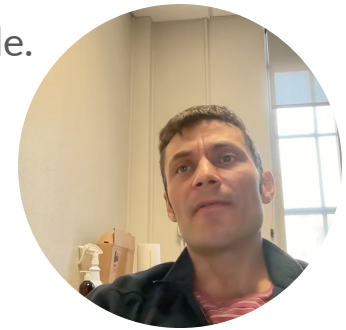
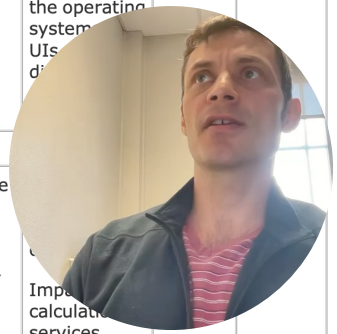


Table 33.2. Partial factor table for the NextGen architectural analysis.

Factor	Measures and quality scenarios	Variability (current flexibility and future evolution)	Impact of factor (and its variability) on stakeholders, architecture and other factors	Priority for Success	Difficulty or Risk
ReliabilityRecoverability					
Recovery from remote service failure	When a remote service fails, reestablish connectivity with it within 1 minute of its detected re-availability, under normal store load in a production environment.	current flexibility - our SME says local client-side simplified services are acceptable (and desirable) until reconnection is possible. evolution - within 2 years, some retailers may be willing to pay for full local replication of remote services (such as the tax calculator). Probability? High.	High impact on the large-scale design. Retailers really dislike it when remote services fail, as it prevents them from using a POS to make sales.	H	M
Recovery from remote product database failure	as above	current flexibility - our SME says local client-side use of cached "most common" product info is acceptable (and desirable) until reconnection is possible. evolution - within 3 years, client-side mass storage and replication solutions will be cheap and effective, allowing permanent complete replication and thus local usage. Probability? High.	as above	H	M
ReliabilityRecoverability					
Supportability - Adaptability					
Support many third-party services (tax calculator, inventory, HR, accounting). They will vary at each installation.	When a new third-party system must be integrated, it can be, and within 10 person days of effort.	current flexibility - as described by factor evolution - none	Required for product acceptance. Small impact on design.	H	L
Support wireless PDA terminals for the POS client?	When support is added, it does not require a change to the design of the non-UI layers of the architecture.	current flexibility - not required at present evolution - within 3 years, we think the probability is very high that wireless "PDA" POS clients will be desired by the market.	High design impact in terms of protected variation from many elements. For example, the operating system UIs di... Impa... calculat... services.	L	H
Other - Legal					
Current tax rules must be applied.	When the auditor evaluates conformance, 100% conformance will be found.	current flexibility - conformance is inflexible, but tax rules can change almost weekly because of the many rules and levels of government taxation (national, state, ...) evolution - none			



Technical Memo

- How did we solve Architectural Factors
  - See previous slide
- Outline
  - Summary
  - Solution
  - Motivation
  - Unresolved
  - Considered Alternatives
- Why do we capture what we did not adopt?

Technical Memo: Issue: ReliabilityRecovery from Remote Service Failure

Solution Summary: Location transparency using service lookup, failover from remote to local, and local service partial replication.

Factors

- Robust recovery from remote service failure (e.g., tax calculator, inventory)
- Robust recovery from remote product (e.g., descriptions and prices) database failure

Solution

Achieve protected variation with respect to location of services using an Adapter created in a ServicesFactory. Where possible, offer local implementations of remote services, usually with simplified or constrained behavior. For example, the local tax calculator will use constant tax rates. The local product information database will be a small cache of the most common products. Inventory updates will be stored and forwarded at reconnection.

See also the *AdaptabilityThird-Party Services* technical memo for the adaptability aspects of this solutions, because remote service implementations will vary at each installation.

To satisfy the quality scenarios of reconnection with the remote services ASAP, use smart Proxy objects for the services, that on each service call test for remote service reactivation, and redirect to them when possible.

Motivation

Retailers really don't want to stop making sales! Therefore, if … offers this level of reliability and recovery, it will be a very attra… our competitors provide this capability. The small product … limited client-side resources. The real third-party tax … the client primarily because of the higher licensing … each calculator installation requires almost weekly … supports the evolution point of future customers wi… replicate services such as the tax calculator to each…

Unresolved Issuesnone

Alternatives Considered

A "gold level" quality of service agreement with remote cr… …es to improve reliability. It was available, but much too expensive…

How do they see documentation in Bass/Clem/Kazm?

- Chapter 18

- Creating an architecture isn't enough.
- It has to be communicated in a way to let its stakeholders use it properly to do their jobs

- The sad truth is that architectural documentation today, if it is done at all, is often treated as an afterthought, something people do because they have to.
  - Maybe a contract requires it. Maybe a customer demands it. Maybe a company standard.

- Perhaps the most important concept associated with software architecture documentation i
  that of the ***view***.
  - A view is a representation of a set of system elements and relations among them—not all syste
    but those of a particular type.

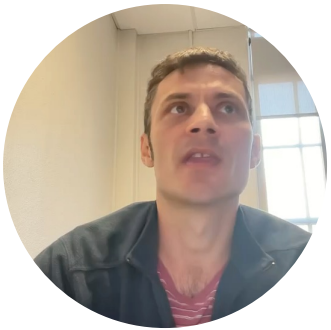
View again

- Views let us divide the multidimensional entity that is a software architecture into a number of interesting and manageable representations of the system.
- The concept of *views* gives us our most fundamental principle of architecture documentation:
  - Documenting an architecture is a matter of documenting the relevant views
    - and then adding documentation that applies to more than one view.
- For instance,
  - a *layered view* will let you reason about your system's portability,
  - a *deployment view* will let you reason about your system's performance and reliability, ar

Module Views

- A module is an implementation unit that provides a coherent set of responsibilities.
- A module might take the form of a class, a collection of classes, a layer, an aspect, or any decomposition of the implementation unit.
- Determines how a system’s source code is decomposed into units.

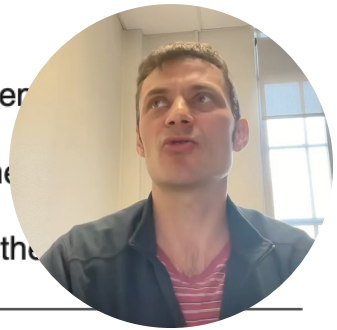
- Includes global data structures that impact and are impacted by multiple units.
- Module structures often determine how changes to one part of a system might affect other parts and hence the ability of a system to support modifiability, portability, and reuse.



Module Views

TABLE 18.1 Summary of the Module Views

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 views 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, such as limitations on the visibility between modules.
Usage	▪ Blueprint for construction of the code ▪ Change-impact analysis ▪ Planning incremental development ▪ Requirements traceability analysis ▪ Communicating the functionality of a syster ture of its code base ▪ Supporting the definition of work assignme tion schedules, and budget information ▪ Showing the structure of information that th to manage



Component-and-Connector Views

- Show elements that have some runtime presence, such as processes, objects, clients, servers, and data stores. These elements are termed ***components***.
- And elements of the pathways of interaction, such as communication links and proto-cols, information flows, and access to shared storage. Such interactions are rep resented as ***conn*ectors.**
- Sample C&C views are service-oriented architecture (SOA), client-server, or communicating process views.

C & C

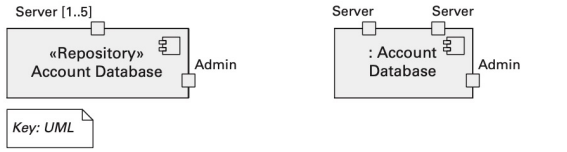


FIGURE 18.1 A UML representation of the ports on a C&C component type (*left*) and component instance (*right*). The Account Database component type has two types of ports, Server and Admin (noted by the boxes on the component's border). The Server port is defined with a multiplicity, meaning that multiple instances of the port are permitted on any corresponding component instance.

TABLE 18.2 Summary of Component-and-Connector Views

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 (interfaces) 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. The case is similar for the roles of a connector.
Constraints	▪ Components can only be attached to connectors, not directly to other components. ▪ Connectors can only be attached to components, not directly to other connectors. ▪ Attachments can only be made between compatible ports and roles. ▪ Interface delegation can only be defined between two compatible ports (or two compatible roles). ▪ Connectors cannot appear in isolation; a connector … to a component.
Usage	▪ Show how the system works. ▪ Guide development by specifying structure and b… elements. ▪ Help reason about runtime system quality attribute… mance and availability.

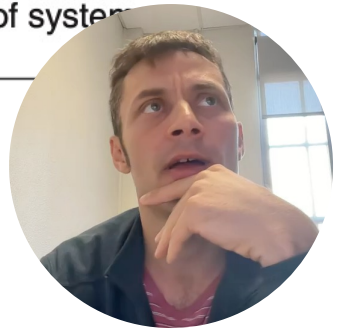
Allocation Views

- Describe the mapping of software units to elements of an environment in which the software is developed or in which it executes.
- ~ Deployment

Allocation

TABLE 18.3 Summary of the Characteristics of Allocation Views

Elements	▪ *Software element.* A software element has properties that are *required* of the environment. ▪ *Environmental element.* 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 view.
Constraints	Varies by view
Usage	▪ For reasoning about performance, availability, security, and safety. ▪ For reasoning about distributed development and allocation of work to teams. ▪ For reasoning about concurrent access to software versions. ▪ For reasoning about the form and mechanisms of syste installation.



Quality Views

- Previous views are structural
- Can be tailored for specific stakeholders or to address specific concerns
  - Security
  - Communication
  - Exceptions
  - Reliability
  - Performance

Security view

- A ***security** view* can show all of the architectural measures taken to provide security.
- Shows the components that have some security role or responsibility, **how those components communicate**, any data repositories for security information, and repositories that are of security interest.
- Shows other security measures (such as physical security) in the system's environment.
- **Shows the operation of security protocols** and where and how humans interact with the security elements.
- Also captures **how the system would respond to specific threats and vulnerabilities**.

Communications view

- A ***communications*** *view* might be especially helpful for systems that are globally dispersed and heterogeneous.
- This view show all of the **component-to-component channels**, the various network channels, **quality-of-service** parameter values, and areas of **concurrency**.

- It can be used to analyze certain kinds of performance and reliability
  - (such as deadlock or race condition detection).
- The behavior part of this view could show how network bandwidth is dynamically allocated

Exception or *error-handling view*

- An ***exception*** or *error-handling view* could help illuminate and draw attention to error reporting and resolution mechanisms.
- Shows how components detect, report, and resolve faults or errors.
- Help to identify the sources of errors and appropriate corrective actions for each.
- Root-cause analysis in those cases could be facilitated by such a view.

Reliability view

- A ***reliability*** view would be one in which reliability mechanisms such as **replication** and switchover are modeled.
- Also depicts timing issues and **transaction integrity**.

Performance view

- A ***performance*** view would include those aspects of the architecture useful for inferring the system's performance.
- Shows **network traffic models, maximum latencies for operations**, and so forth.

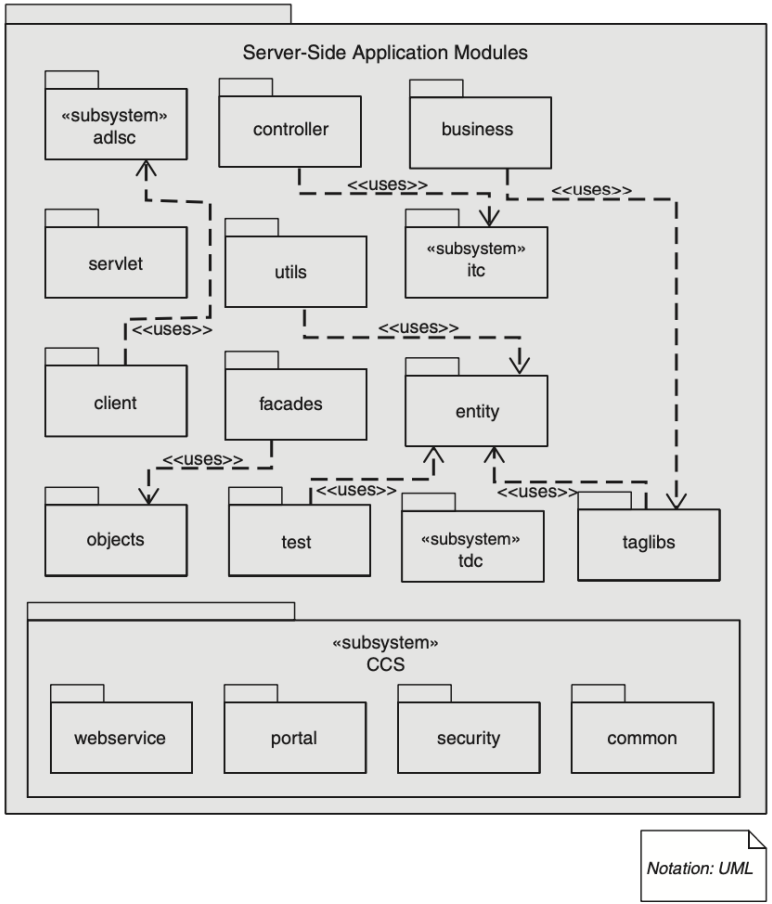


FIGURE 18.2 A decomposition view overlaid with “uses” information, to create a decomposition/uses overlay.

Documenting a View

Figure 18.3 shows a template for documenting a view.

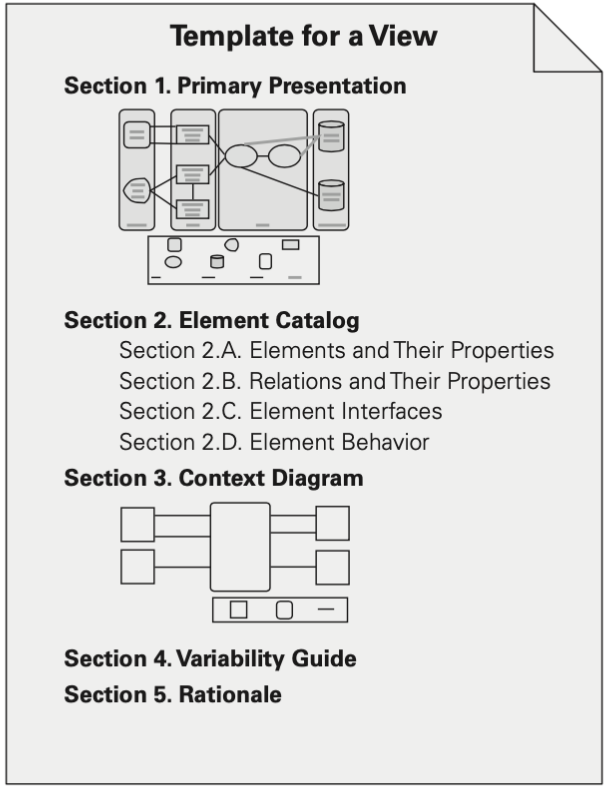


FIGURE 18.3 View template

Our reconstruction automation attempt for Microservices

- Microservice Architecture (MSA)
- Code-analysis
  - Analyze individual modules and merge results
    - Data model
    - Remote-calls
    - Deployment descriptors
- Accepted publication
  - Walker, Laird and Cerny, *On Automatic Software Architecture Reconstruction of Microservice Appl*

Software Architecture Reconstruction (SAR).

- The construction of a simplified overview of the application can help developers understand the application’s full scope, even beyond their modules.

Systematic Architecture Reconstruction

SAR has historically been defined with four distinct phases:

- ***Extraction***: This phase **collects all of the artifacts** needed during the next three phases of SAR.
  - The artifacts collected are relevant to the ”views” that are being constructed.
  - A ”view” is defined as a set of related artifacts that cover a concern of the architecture of the system.
- ***Construction***: This phase creates a **canonical representation** of the views and usually stores them in some form, like a database.
- ***Manipulation***: This phase **combines the views** to allow for the answering of more complicated questions in the next phase.
  - How the views are combined is relevant to the specific application being reconstructed.
- ***Analysis***: This phase **answers questions** about a system given the overall views of the archi constructed in the previous phases.
  - There are almost infinite questions that can be asked, covering multitudes of domains, including n and code quality.

Considered Views

- – Domain View: This view covers the domain concerns of an MSA application.
  - entity objects of the system + data source connections.
- – Technology View: This view focuses on the technology aspect of an application.
  - technologies used for microservice implementation and operation.
- – Service View: The view focuses on service operators.
  - the service models that specify microservices, interfaces, and endpoints.
- – Operation View: This view focuses on the ops concern of a system
  - service deployment and infrastructure, such as containerization, service
    monitoring.

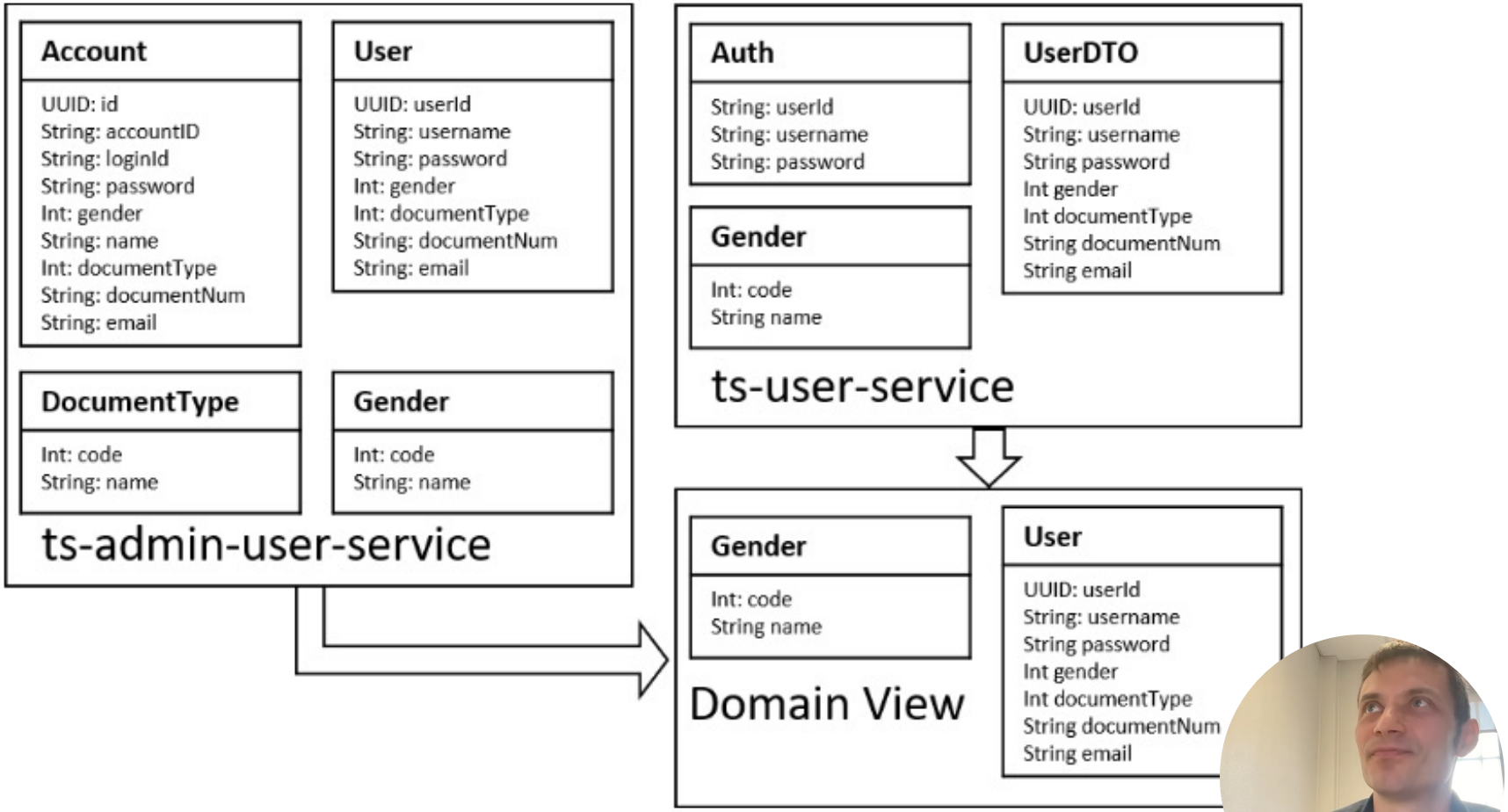


Fig. 1. Merged Domain View from TrainTicket

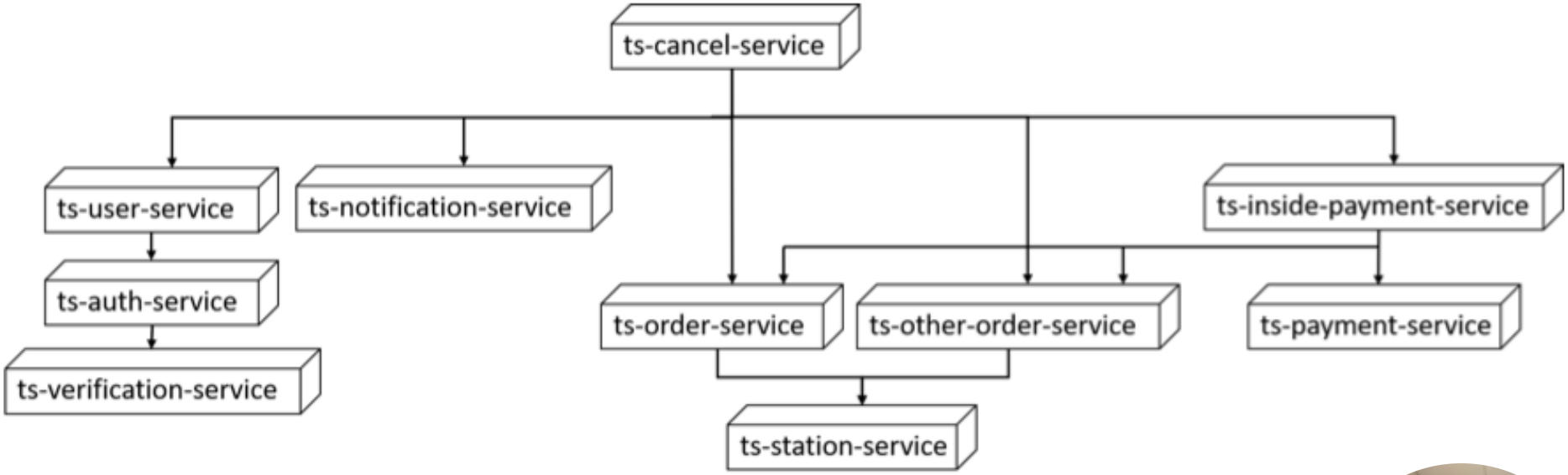


Fig. 2. Service View from TrainTicket

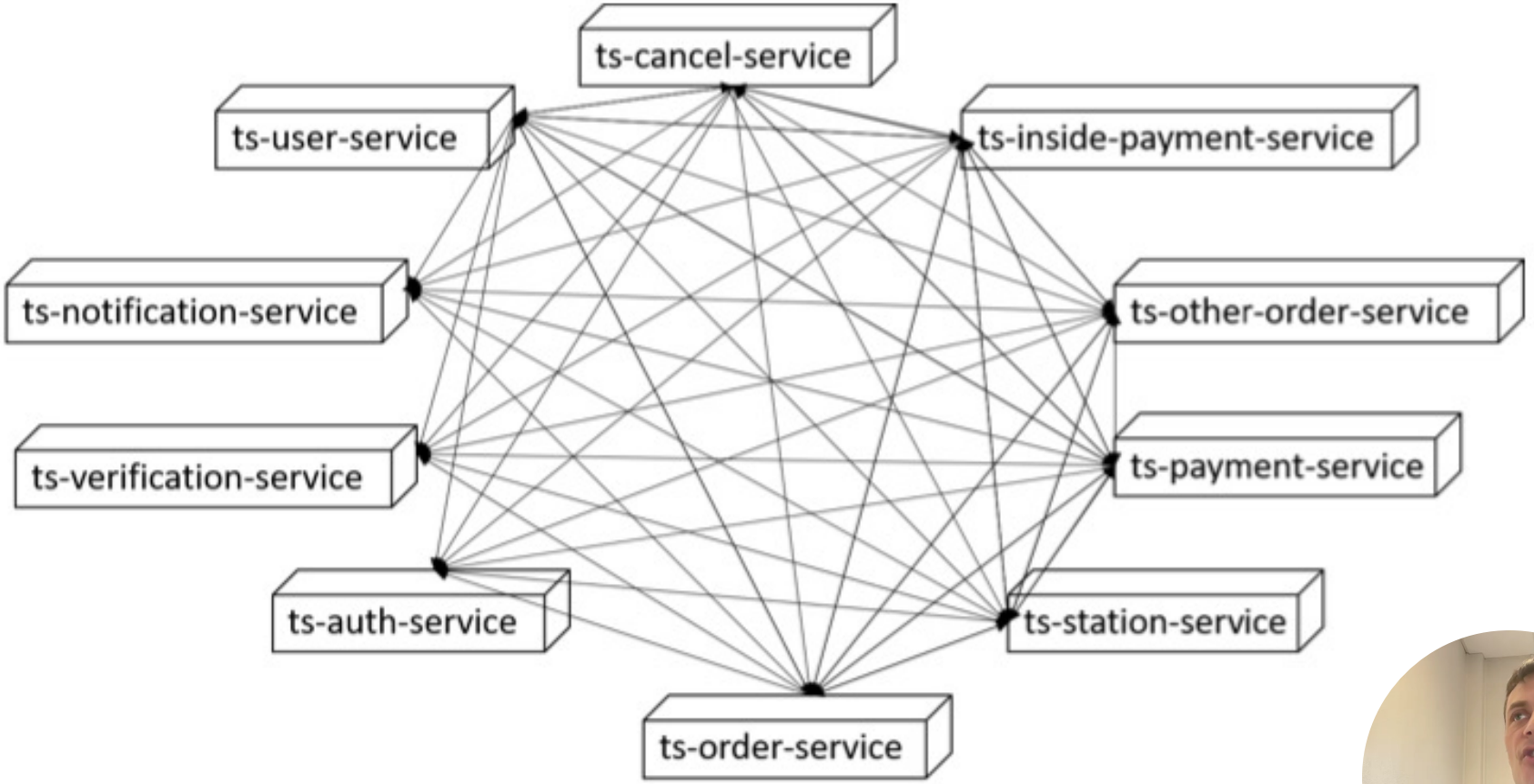


Fig. 3. Operation View from TrainTicket