Software Architectures for Collective Intelligence II

Advanced Software Engineering VO
(180.456)

Jürgen Musil
Juergen.Musil@tuwien.ac.at

Software Architectures for Collective Intelligence (CI)

1. Introduction (11.01.2018)
   - Overview on Collective Intelligence, Crowdsourcing & Human Computation.
   - What are Collective Intelligence Systems and what not.
   - Relevance of CIS today.
   - Getting started with architecting CIS.

2. **System & Technology Architectures (18.01.2018)**
   - **Architecting CIS**
   - **System and Technology Architecture Trade-Offs**
   - **Technology Stacks**
   - **Formats, Standards, APIs**

Collective Intelligence & Computer Science

- Definition: **"Groups of individuals doing things collectively that seem intelligent"** (Malone et al.,2009).
  - Connect people and computers to act collectively more intelligently -> socio-technical (eco-)system.
    - also: thrive on network effects.

- Harnessing collective intelligence requires to: **stimulate, aggregate, leverage, and distribute user contributions through an ICT system as mediator.**

Collective Intelligence – Our Take

1. Achieved by **hybrid systems** in which **humans and computers** interoperate and complement each others capabilities.

2. Potential for **highly effective collection** and distribution of hard-to-access knowledge.

3. Used for **Crowdsourcing, Social Web/Media, Social/Cognitive/Human Computing**.

Examples of CIS

- **Social network services**
  - Facebook, Twitter, LinkedIn, Foursquare, SinaWeibo

- **Media / Content Sharing**
  - YouTube, Flickr, Soundcloud, Slideshare, Thingiverse

- **Knowledge Creation**
  - Wikipedia, Crunchbase, Wikia, Yelp, Stack Overflow, Zooiverse

Recap: Challenges

1. **Designing** the “right” system.
   - Requirement elicitation of user needs and optimization potential.
   - Getting the basic workflows right.

2. **Perpetual beta** due to constant evolution of capabilities.
   - Continuous delivery

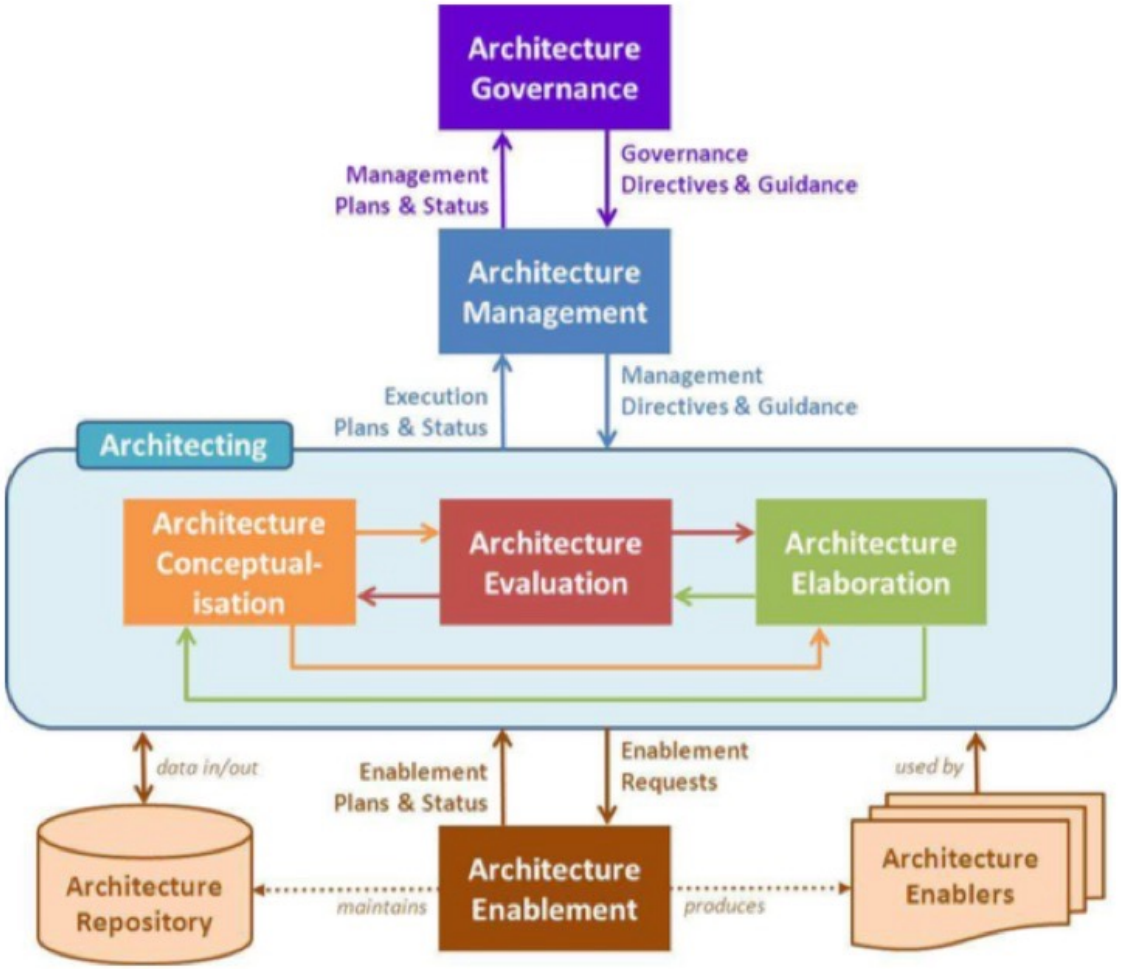
3. Fostering an **active community of contributors**.
   - Users are scarce resource: Competition with existing platforms.
   - Engagement (incentives, motivation)

4. **Scaling** up to (ultra) large-scale proportions.
   - Big data and Machine learning
   - Cloud computing
   - Global software development

ISO/IEC WD 42020 - Architecture Processes

- **Purpose:** Provide a defined set of processes in the life cycle of an architecture or the life cycle of systems related to that architecture.
- Architecture activity considered as **strategic** on **project** and **enterprise level**.
- Complements the architecture-related processes of **ISO/IEC/IEEE 15288**, **ISO/IEC/IEEE 12207** and **ISO 15704** with a set of requirements enabling architects to more effectively implement architecture practices.
- **Does not prescribe a specific** architecture or system life cycle model, development methodology, method, model or technique.

ISO/IEC WD 42020 - Architecture Processes



ISO/IEC WD 42020 - Architecture Processes

- **Architecture Governance**
  - Establish standards and policies related to one or more architectures of interest and their development, and to monitor and facilitate the alignment of the architecture(s) to stakeholder concerns, policies and standards, including organizational and environmental constraints.

- **Architecture Management**
  - Ensure execution of directives for development of the architectures, to ensure that the development runs according to these directives, to the expected timetables, to the assigned budgets, and that the architecture satisfies its objectives.

ISO/IEC WD 42020 - Architecture Processes

- **Architecture Conceputalization**
  - Generate architecture alternatives, to select one or more alternatives that address stakeholder concerns and meet relevant requirements, and to express them in a set of consistent views.

- **Architecture Evaluation**
  - Determine the degree to which the architecture meets architecture objectives and addresses stakeholder concerns.

- **Architecture Elaboration**
  - Create one or more architecture descriptions in a form that uses established notations and languages and captures this in a set of consistent views and models.

ISO/IEC WD 42020 - Information Flows

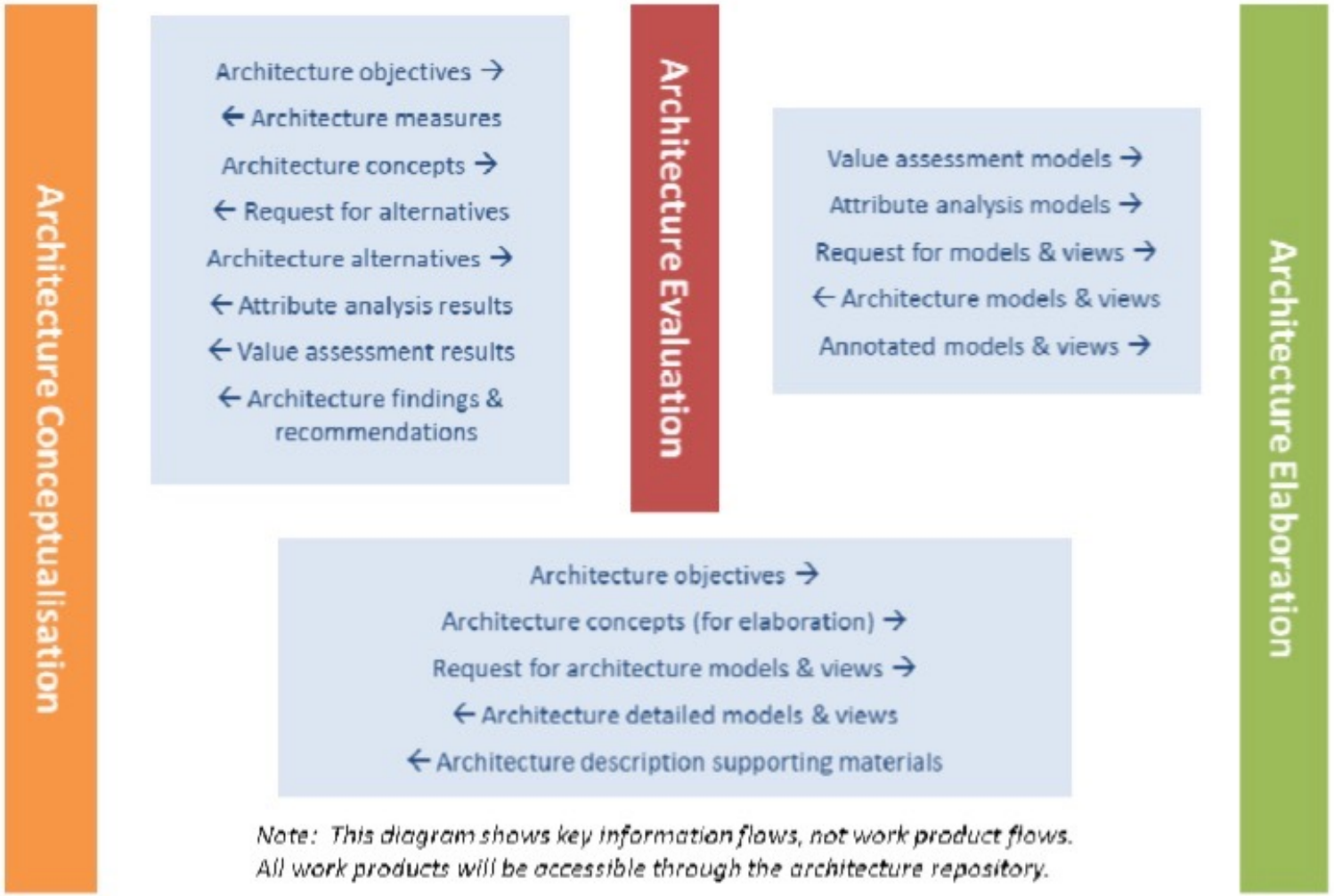


Figure 2: Key information flows between the architecting processes

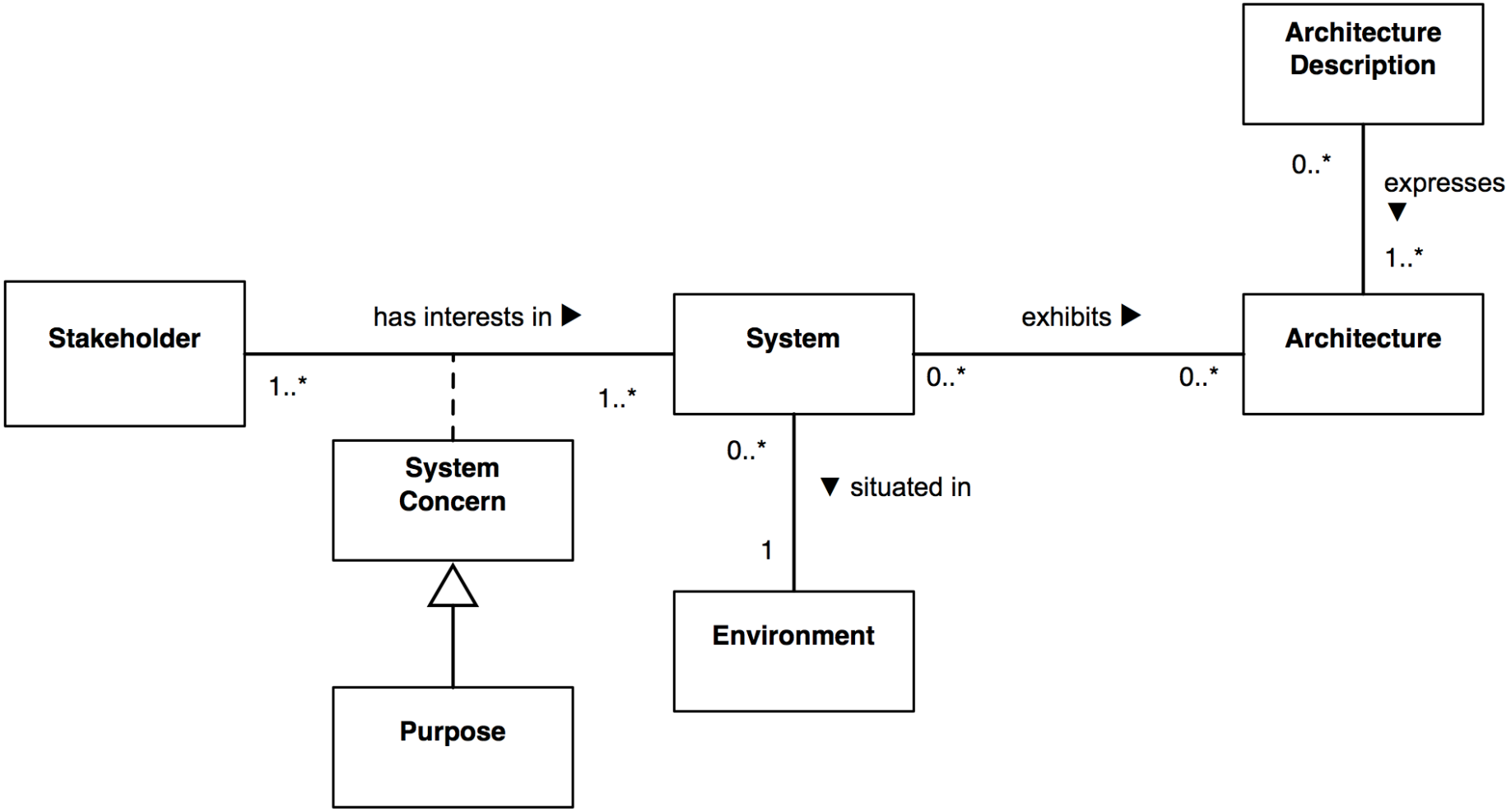
Software Architecture - Definition 1 (Bass et al., 2012)

The software architecture of a system is the **set of structures** needed to reason about the system, which comprise **software elements**, **relations** among them, and **properties** of both.

Software Architecture - Concepts

- **Environment**
  - Every system is situated in the **context** of a defined environment: **setting** and technological, business, operational, organizational, political, social, regulatory **influences**.
- **Stakeholder**
  - **Individuals, groups or organizations** defining a system's purpose and having interests in a system.
  - Examples: system users, owners, operators, maintainers, architect, developers, suppliers, regulators, client, designer.
- **System/Stakeholder Concern**
  - Specific **interest of stakeholders** in a system.
  - Examples: business goals, cost, data access, data integrity, flexibility, maintainability, performance, privacy, usability, system properties, system features, resource utilization, reliability.

Software Architecture - Concepts (cntd.)



(ISO/IEC/IEEE 42010, 2011)

Software Architecture - Definition 2 (Jansen & Bosch, 2005)

Software architecture as a **set of architectural design decisions**: An architectural design decision is a description of the set of architectural additions, subtractions and modifications to the software architecture, the **rationale**, and the **design rules**, **design constraints** and **additional requirements** that (partially) realize one or more requirements on a given architecture.

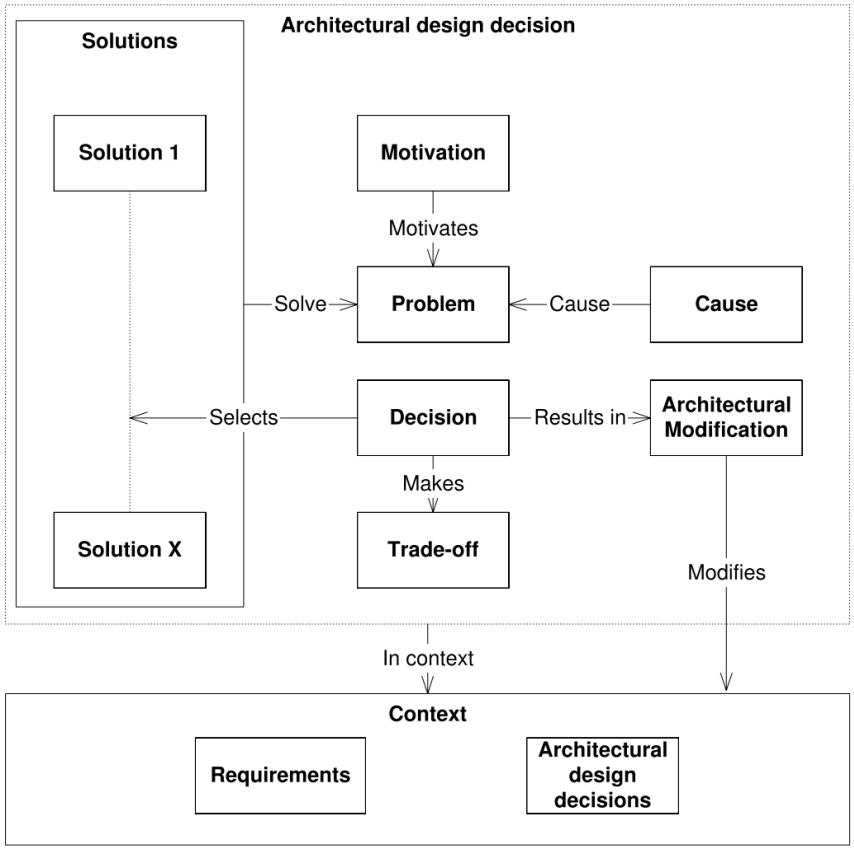
Software Architecture - Definition 2 (Jansen & Bosch, 2005)

1. **Rationale:** The reasons behind an architectural design decision are the rationale of an architectural design decision. It describes *why* a change is made to the software architecture.

2. **Design rules** and **design constraints** are prescriptions for further design decisions. Rules are mandatory guidelines, whereas constraints limit the design to remain sound.

3. **Design constraints** describe the opposite side of design rules. They describe what is not allowed in the future of the design, i.e. they prohibit certain behaviors.

4. **Additional requirements** A design decision may result in additional requirements to be satisfied by the architecture. These new requirements need to be addressed by additional design decisions.



ISO/IEC/IEEE 42010:2011 Standard

- International standard: **Systems and software engineering - Architecture description**
- Specification of definitions and requirements on the contents of
  - Architecture Descriptions of Systems
  - Architecture Frameworks
  - Architecture Description Languages (ADLs)
- Supports the **architecture** of a system of interest to describe what is considered fundamental about that system in the **context** of its **environment**.
  - Important to understand how the systems relates to and is situated in its environment in the context of its stakeholders.

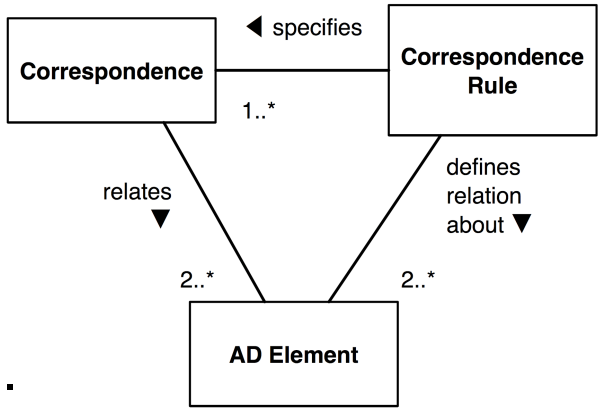
ISO/IEC/IEEE 42010:2011 Standard (cntd.)

Key Concepts:

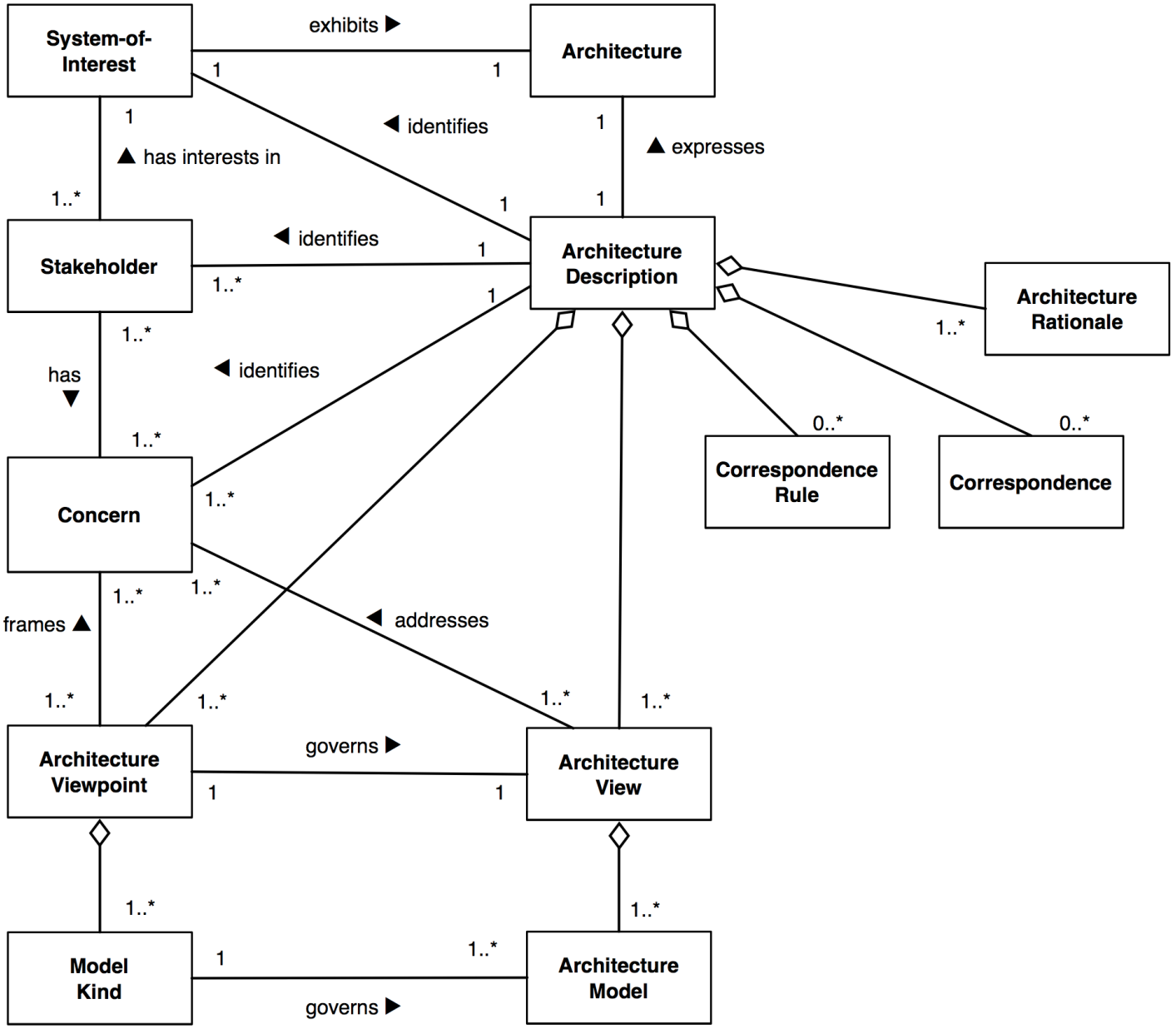
- **Architecture Description**
  - Documents one possible **architecture** for a system of interest and rationales for key **design decisions**.
  - Identifies **stakeholders** of the system of interest and their **concerns**.
  - Describes how an architecture meets the **needs** of the system's diverse stakeholders.
  - Illustrates with multiple chosen **architecture views** how the concerns of the stakeholders can be addressed.
- **Architecture View**
  - Describes a system of interest from a chosen **viewpoint**.
  - Comprises several **architecture models** expressing various aspects of a system architecture to address specific **concerns.**

ISO/IEC/IEEE 42010:2011 Standard (cntd.)

- **Architecture Viewpoint**
  - Promotes **reuse** of best practices.
  - Includes set of **model kinds** to frame **stakeholders** and a specific set of **concerns**.
  - Documents conventions (notations, models, techniques, rules, completeness) for constructing, analyzing, using and interpreting a particular kind of **architecture view**.
- **Correspondences**
  - Express **architecture relations** of interest between elements within an architecture description.
  - Examples of AD elements: stakeholder, concern, viewpoint, view, model kind.
- **Correspondence Rules**
  - Governs correspondences and enforce relations within an architecture description.



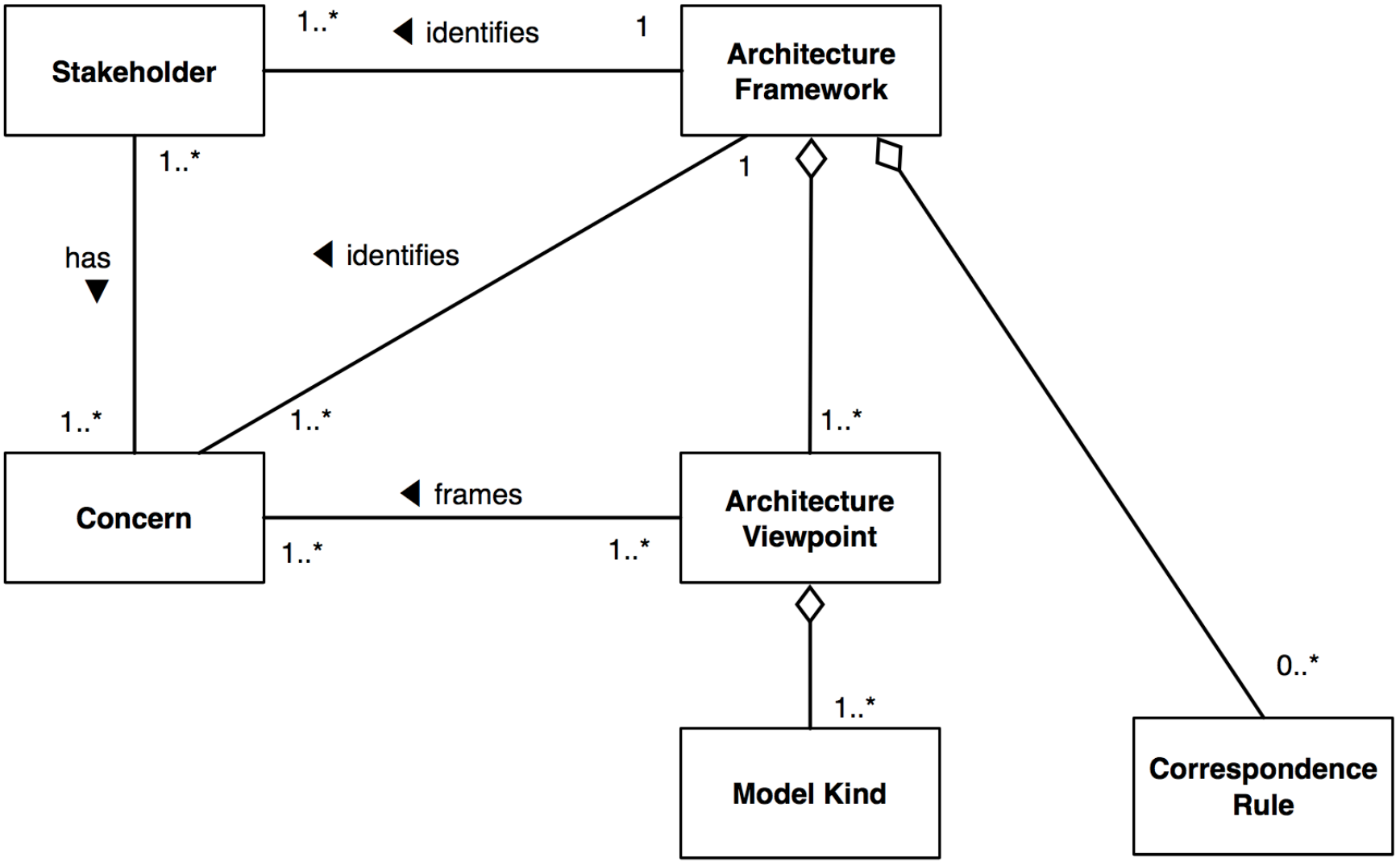
ISO/IEC/IEEE 42010:2011 Standard (cntd.)



ISO/IEC/IEEE 42010:2011 Standard (cntd.)

- **Architecture Framework**
  - Defines **conventions, principles and common practices** for creating, interpreting, analyzing and using architecture descriptions intended for a certain stakeholder community and/ or specific domain of application.
  - Specifies
    - addressed **concerns**.
    - **stakeholders** having those concerns.
    - integrated set of **architecture viewpoints** that frame those concerns.
    - **correspondence rules** integrating those viewpoints.
  - Examples: TOGAF, DoDAF, Kruchten's 4+1 View Model

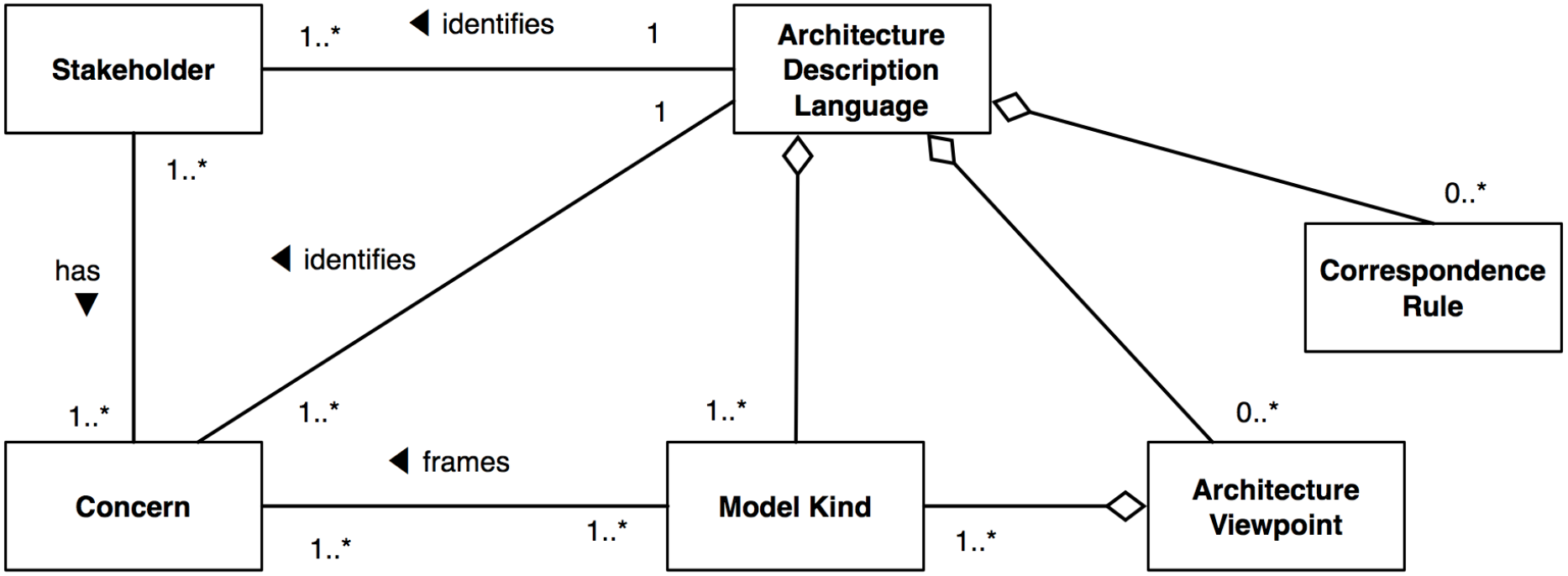
ISO/IEC/IEEE 42010:2011 Standard (cntd.)



ISO/IEC/IEEE 42010:2011 Standard (cntd.)

- **Architecture Description Language (ADL)**
  - **Form of expression** for use in architecture descriptions.
  - Specifies
    - addressed **concerns**.
    - **stakeholders** having those concerns.
    - **model kinds** implemented by the ADL which frame those concerns.
    - any **architecture viewpoints** and **correspondence rules**.
  - Examples: SysML, ArchiMate, xADL

ISO/IEC/IEEE 42010:2011 Standard (cntd.)



Multi-Agent System (MAS) - Definition (Wooldridge & Ciancarini, 2002)

A multi-agent system is a system composed of a **number of such agents**, which typically **interact with one-another** in order to **satisfy their goals**.

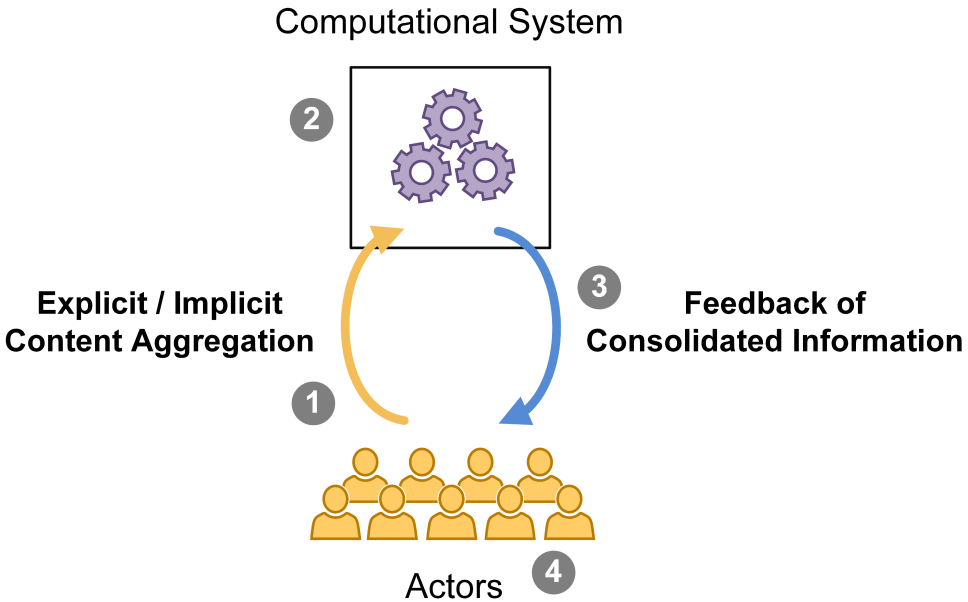
Architecting CIS

- Investigation of architectural foundations and principles of self-organizational CIS.

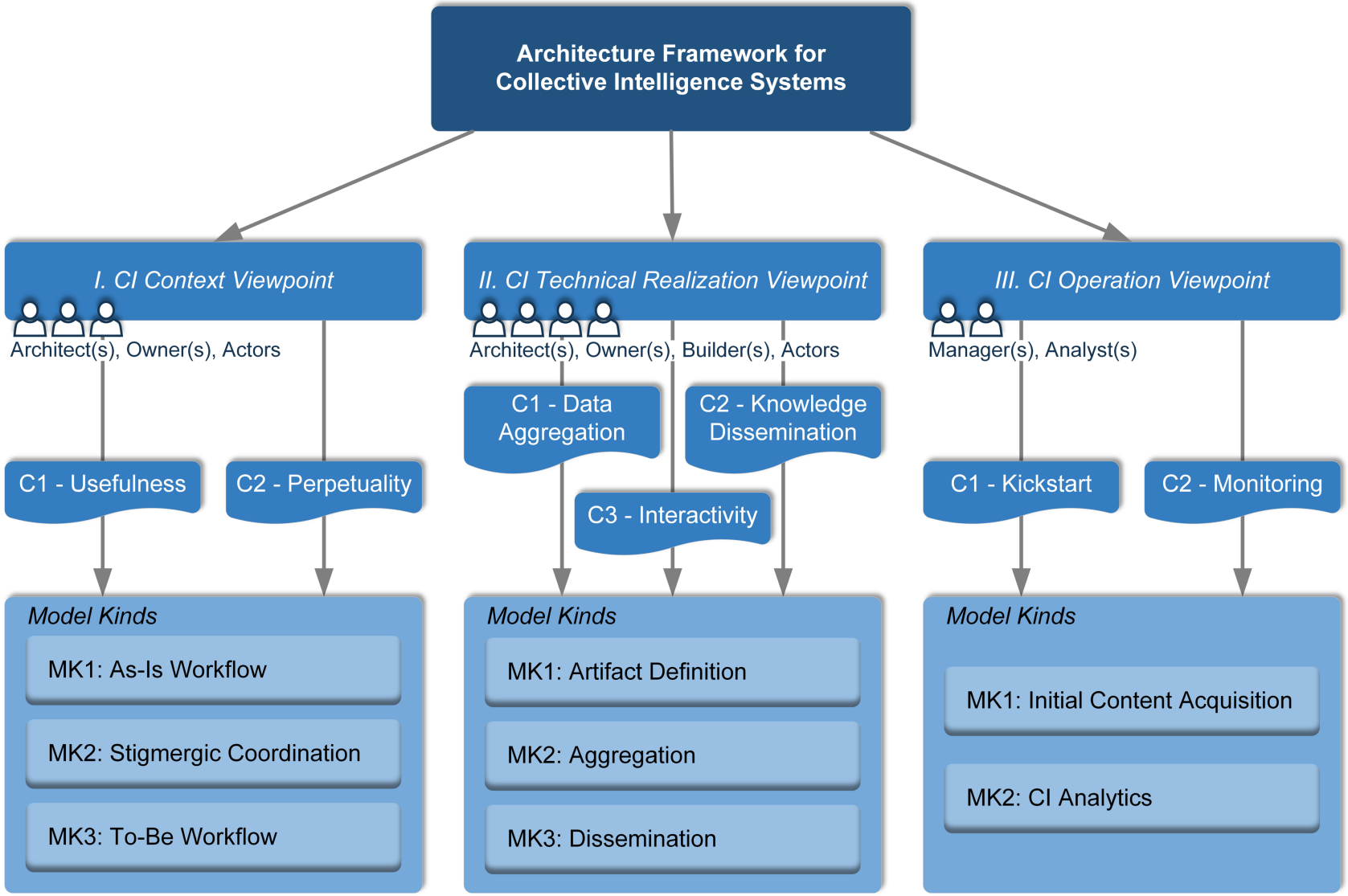
- Influences & Approaches
  - **Multi-Agent Systems (MAS)**
    - Socio-technical systems where agents interact with each other and with the surrounding environment.
    - Environment architectures for MAS.
    - **Agent-Oriented Software Engineering** (AOSE).
  - **CI-adapted Coordination Models**
    - Feedback loops, self-organization and self-adaption approaches.
  - **Software Architecture**
    - Standard-based software architecture frameworks and reference architectures.

CIS Concerns

- **Information Aggregation**
  - Bottom-up sharing of content through actors.
- **Knowledge Dissemination**
  - Distributing content among actors.
- **Perpetual Feedback Loop**
  - Continuous flow of user contributions.



Architecture Framework for CIS

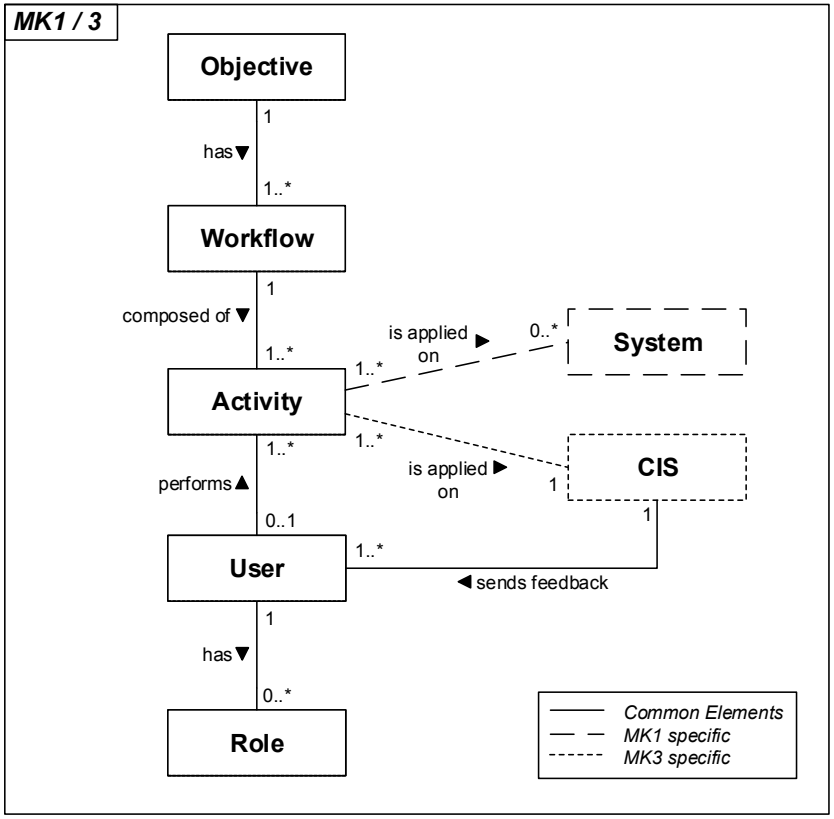


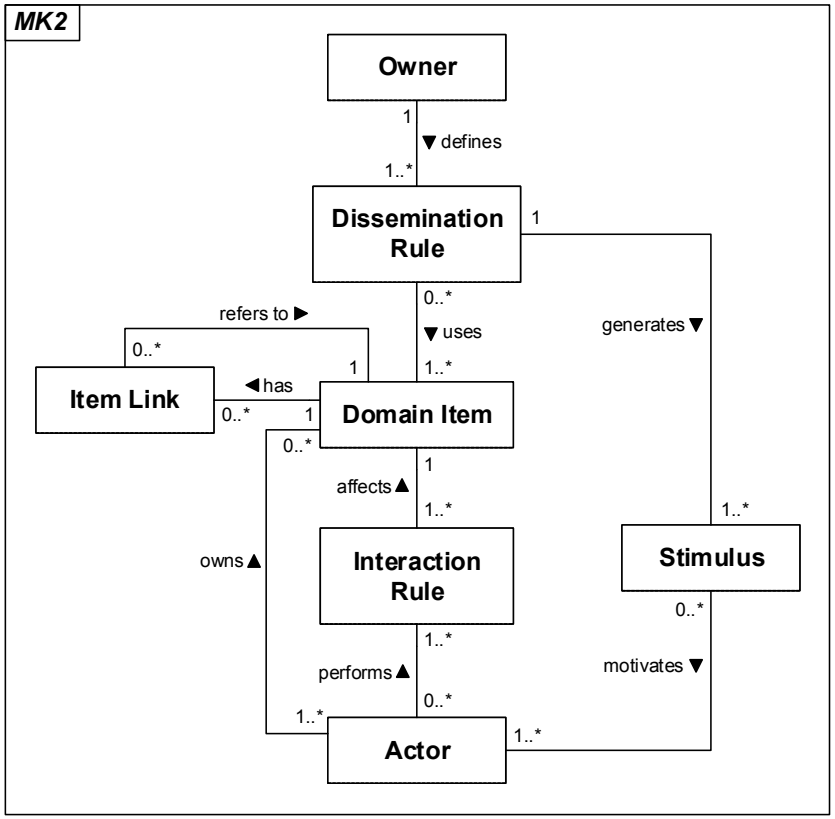
Architecture Framework for CIS (cntd.)

I. CI Context Viewpoint

- **Designs CI-specific system capabilities** and defines models for new **CIS construction** and **capture of design decisions**.
- Stakeholders
  - **Architect(s), Owner(s), Actors**
- Concerns
  - **Usefulness** - Process limitations addressable with CIS?
  - **Perpetuality** - Identify coordination-supporting process for the application scenario?
- Model Kinds
  - **MK1 - As-Is Workflow**: Current workflow of interest in the organization.
  - **MK2 - Stigmergic Coordination**: Describes domain items of interest in organization, the rules to interact with the domain items, and the feedback loop that provides stimuli to users.
  - **MK3 - To-Be Workflow**: Describes activities performed by users and CIS, and its feedback mechanisms.

Architecture Framework for CIS (cntd.)



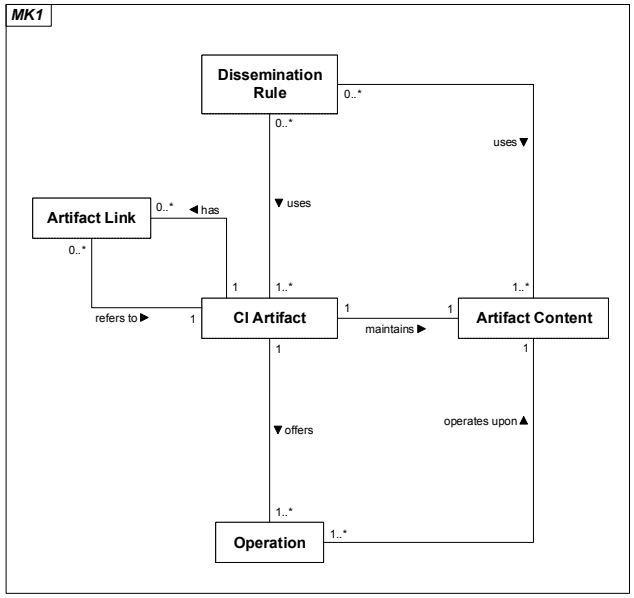


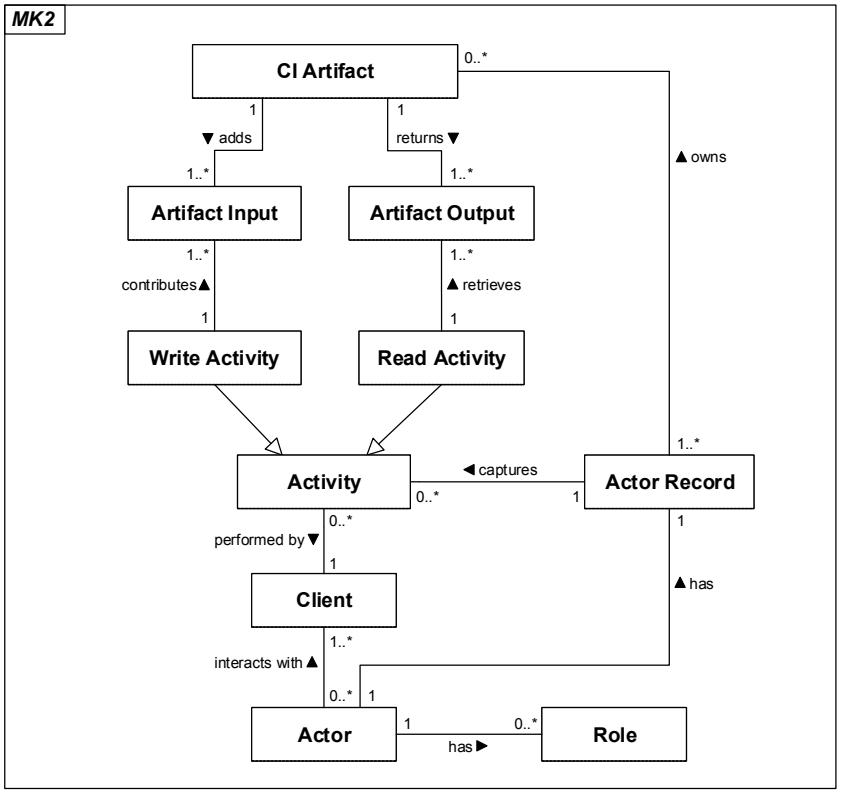
Architecture Framework for CIS (cntd.)

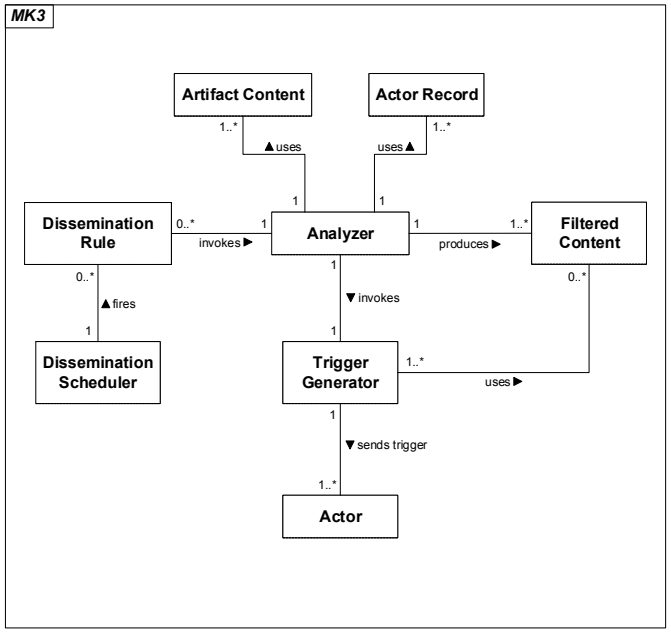
II. CI Technical Realization Viewpoint

- **CIS realization** and defines models to model **collective knowledge**, the **aggregation of data** and **stigmergy-based dissemination of knowledge**.
- Stakeholders
  - **Architect(s), Owner(s), Builder(s), Actors**
- Concerns
  - **Data Aggregation, Knowledge Dissemination, Interactivity**
- Model Kinds
  - **MK1 - Artifact Definition**: Details CI artifact structure, linking, and the operations to interact with artifact content.
  - **MK2 - Aggregation**: Describes actor activities, what kind of data is aggregated from actors, and activity logging.
  - **MK3 - Dissemination**: Details dissemination content and rules.

Architecture Framework for CIS (cntd.)





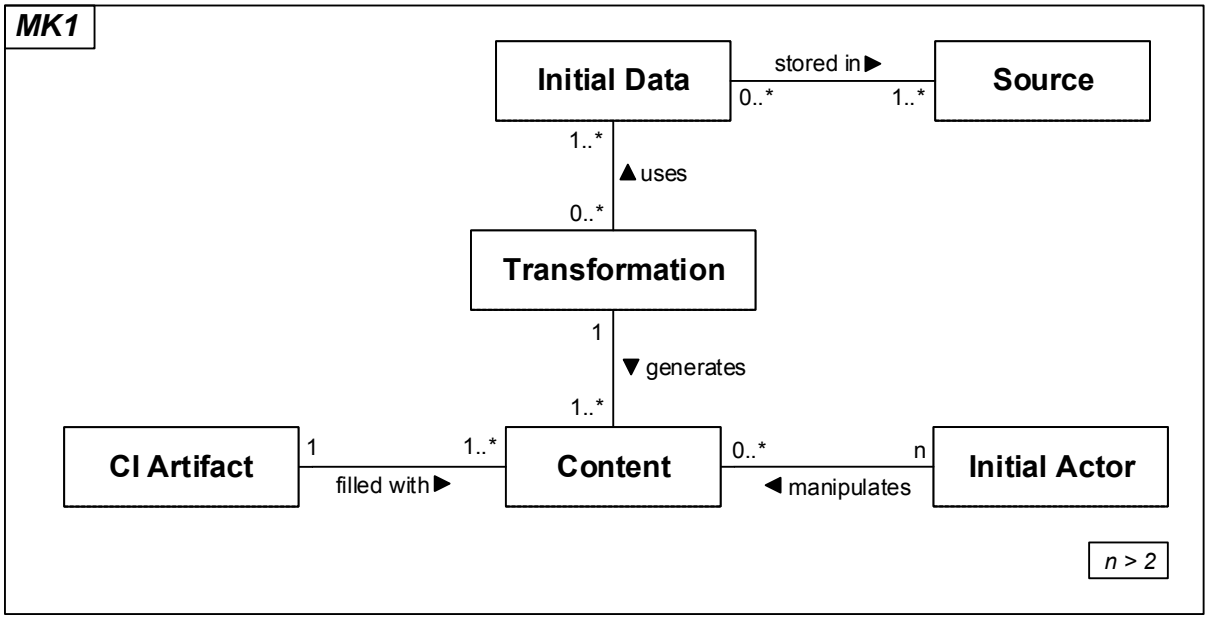


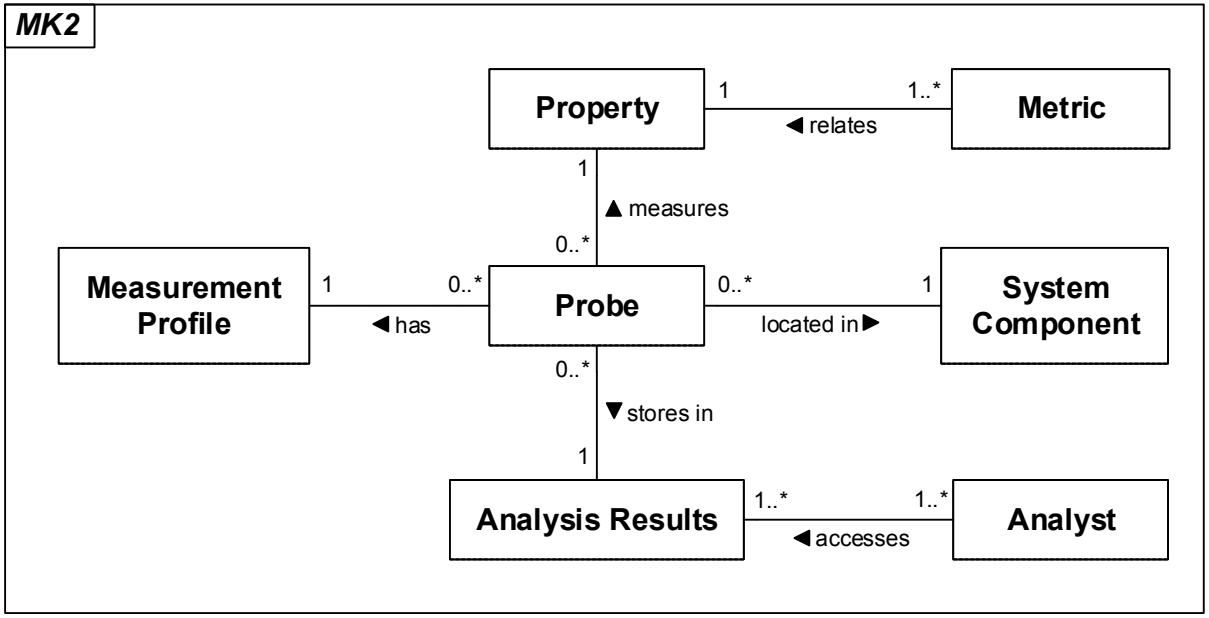
Architecture Framework for CIS (cntd.)

III. CI Operation Viewpoint

- **CIS operation startup** and defines models to **identify initial content**, **actor groups**, and **measures** for CIS aggregation and dissemination performance.
- Stakeholders
  - **Manager(s), Analyst(s)**
- Concerns
  - **Kickstart** - How to derive initial content from existing data?
  - **Monitoring** - Detail metrics and probes for monitoring?
- Model Kinds
  - **MK1 - Initial Content Acquisition**: Describes sources of initial content for artifacts and initial actor groups for community building.
  - **MK2 - CI Analytics**: Describes metrics and probes to capture necessary data for measure calculation.

Architecture Framework for CIS (cntd.)





Overview of Web Application Frameworks

	Rails	Django	Java EE
Category	Web Application Framework	Web Application Framework	Platform
OS	Cross-platform	Cross-platform	Cross-platform
Programming Language	Ruby	Python	Java
Model-View-Controller	X	X	X
Object-relational mapping	Active Record	Django ORM	Java Persistence API
Extension/Plug-in	RubyGems	Django Packages	jar
Focus	Fast & Agile	Fast & Agile	Scalability & Enterprise IT Integration

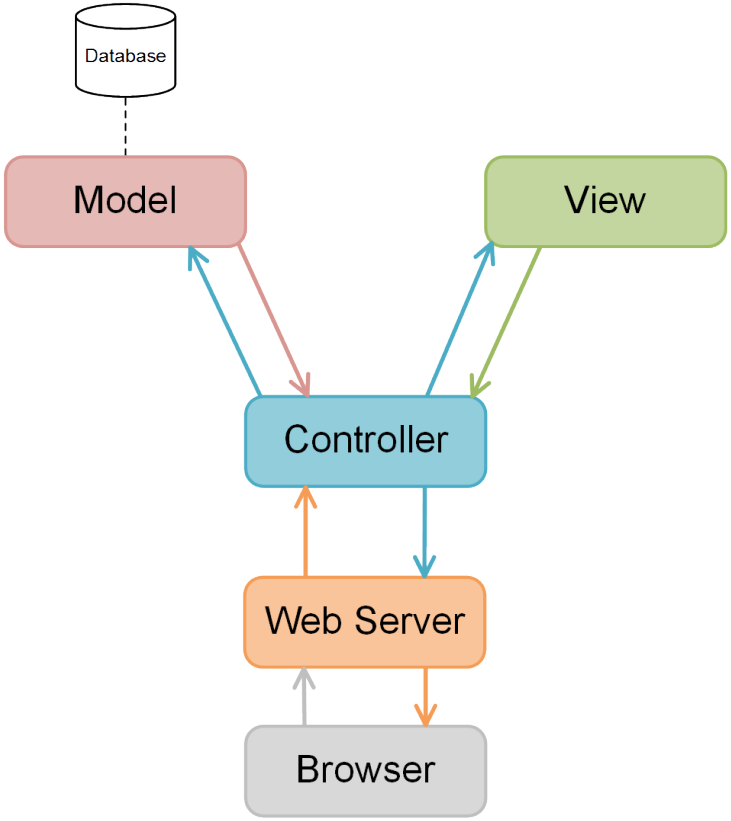
Ruby on Rails

- Rails is a **web application framework** which runs on the Ruby programming language.
- Encourages to use **REST**ful routes (e.g. index, show, create).
- Uses **MVC pattern** to organize application:
  - Default setup: “app” folder containing
    - **“controllers”:** component that responds to external requests from the web server to the application by determining which view file to render and also has to query model(s) directly for data.
    - **“models”:** a model maps to a database table.
    - **“views”:** user interface, which is converted to HTML at run-time.
- Bundler: maintains consistent environment.
  - Tracks applications code and dependencies (gems) it needs for execution.

Model-View-Controller (MVC) Pattern

- **Software pattern** for implementing applications with UI.
- Separation of presentation (**view**), control logic (**controller**) and domain model (**model**).

- **Model**
  - Consists of application data and domain logic.
- **View**
  - Generates visual representation of the model in the UI.
- **Controller**
  - Gets requests/input by the user.
  - Manipulates the model.
  - Responses by updating the view's presentation.



Java EE

- Oracle's enterprise Java platform providing APIs and runtime environment for developing and running enterprise applications
- Model
  - **Java Persistence API (JPA)**
  - Heavily influenced by major ORM frameworks, e.g. Hibernate or EclipseLink (former Oracle TopLink)
- View
  - **JavaServer Faces API**
  - Building component-based user interfaces for web applications
  - JSF frameworks: ICEfaces, PrimeFaces, Apache MyFaces, …
- Controller
  - **Enterprise JavaBeans**

Ruby on Rails vs. Java EE

Ruby on Rails

Pros:

+ **Easy & quick** project setup from scratch and configuration.
+ Rails scripts facilitate **fast development**.
+ Enables **short trial & error cycles**.
+ **Less boilerplate code** thus smaller, more maintainable codebase.

Cons:

- **Harder customization due to obscurity**, since feature implementations are hidden in Rails libraries.
- **Scaling**: RoR is often slower to run and requires more memory, thus it becomes increasingly inefficient (e.g. ActiveRecord ORM).
- **Slower execution** of Ruby prog. language.

Ruby on Rails vs. Java EE (ctnd.)

Java EE

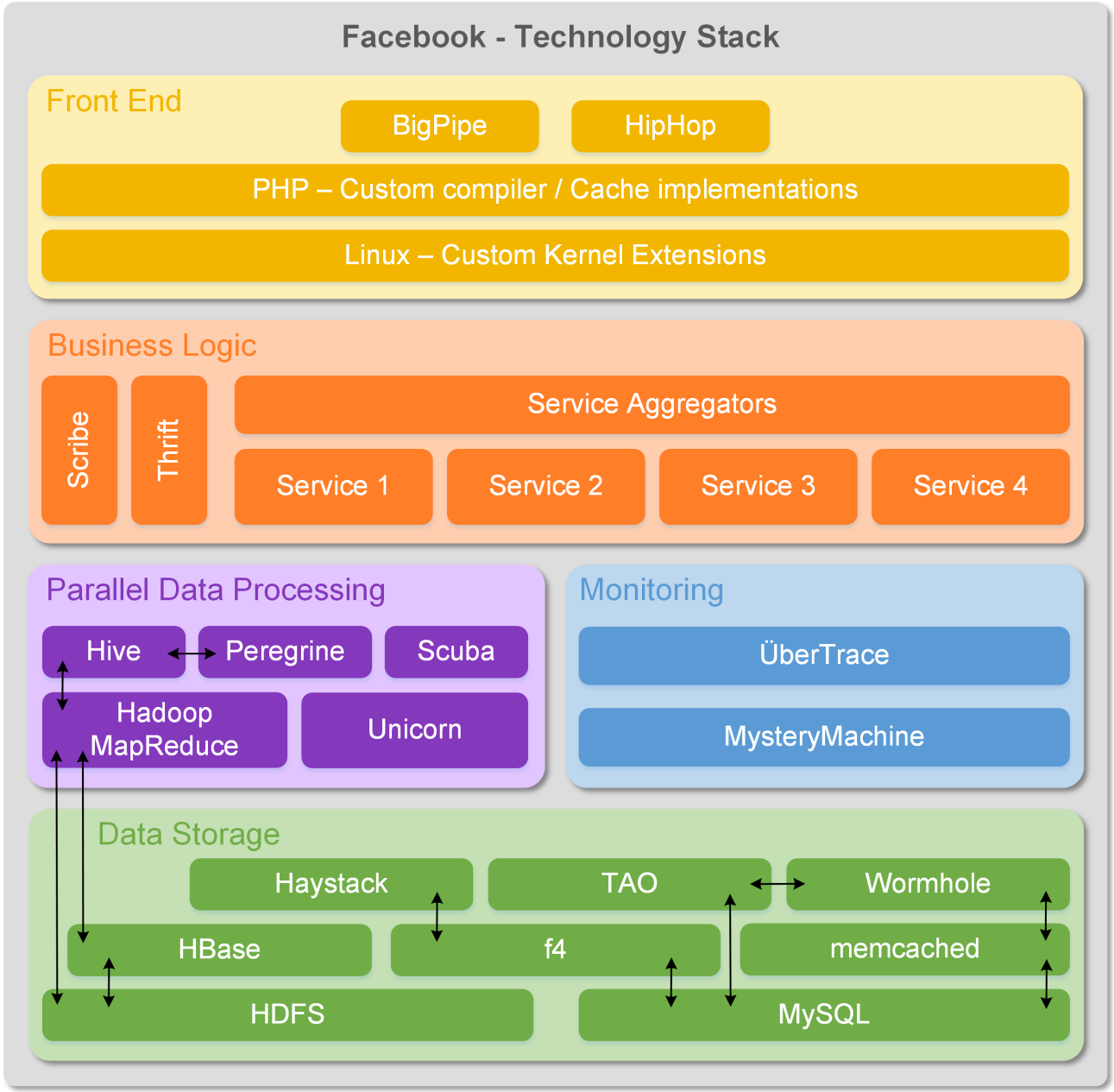
Pros:

+ Tools/APIs support **performance, stability and maintainability**.
+ **Scaling:** Optimized, robust infrastructure.
+ **Long-term support** due to standardization.
+ Java EE libraries (e.g. Hibernate) are often **more flexible and powerful** than Rails libraries (e.g. ActiveRecord).

Cons:

- **Complex setup/configuration**.
- Due its standardization, **slow response to industry changes**.
- Libraries can also lead to **coding and maintenance overhead**.

Facebook's Technology Stack



"Facebook Architecture - Breaking it Open", Aditi Technologies
[http://www.slideshare.net/AditiTechnologies/facebook-architecture-breaking-it-open]

Facebook – Front End

- **PHP**
  - Server-side scripting language.

- **HipHop**
  - Virtual Machine for PHP developed by Facebook.
  - Converts PHP code to C++.
  - Reduces CPU usage significantly.

- **BigPipe**
  - Dynamic web page serving system developed by Facebook.
  - Decomposes web pages into multiple small chunks called “pagelets” and pipelines them through several execution stages inside web servers and browsers.
  - Implemented entirely in PHP and JavaScript.

Facebook – Business Logic

- **Apache Thrift**

Apache Thrift ™

  - Software framework for scalable, cross-language services development.
  - Combines a software stack with a code generation engine to build services between e.g. C++, PHP, Python, Java, Ruby.

- **Scribe**
  - Logging server for aggregating real-time client data.
  - Designed to be scalable to a very large number of nodes, extensible, and robust to failure of the network or any specific machine.
  - Logging a wide array of data.
  - Built on top of Thrift.
  - Developed by Facebook.

Facebook – Parallel Data Processing

- **Apache Hive**
  - Data warehouse software built on top of Apache Hadoop.
  - Facilitates reading, writing, and managing of large data sets residing in distributed storage using SQL-like language named HiveQL.
  - Provides command line tool and JDBC driver to connect users to Hive.

- **Peregrine**
  - Distributed query engine for ad hoc analysis built on top of Apache HDFS and the Hive-Metastore.
  - Provides support for low-latency interactivity queries.
  - Designed to be lean and simple with high performance.

- **Scuba**
  - Fast, scalable, distributed in-memory database.
  - Provides real-time, ad hoc analysis of arbitrary data.

Facebook – Parallel Data Processing (cntd.)

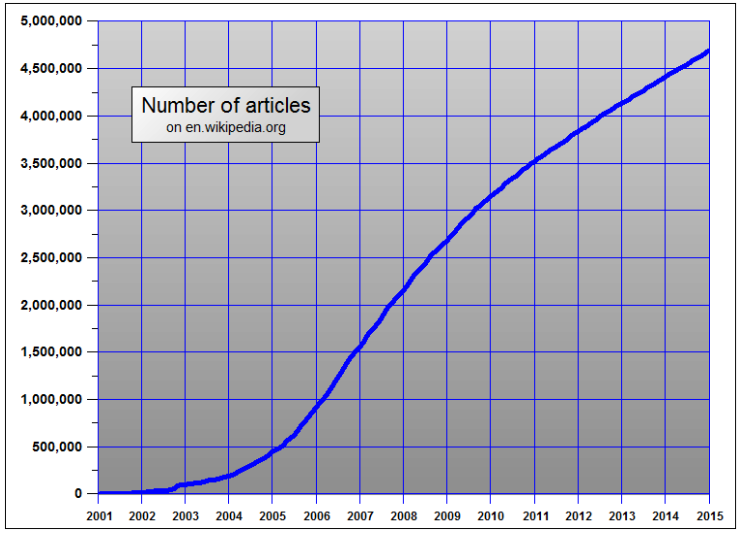
- **Apache Hadoop**
  - Software framework for distributed storage and processing of large data sets across clusters of computers.
  - Designed to be reliable and scalable (support from single server to thousands of machines).
  - Core consists of:
    - distributed file-system: **Hadoop Distributed File System (HDFS)**
    - large scale parallel data processing: **MapReduce**

- **Unicorn**
  - In-memory social graph-aware indexing system.
  - Quickly and scalably searches structured information on the social graph and performs complex set operations on the results.
  - Supports social graph retrieval and social ranking.

Facebook – Data Storage

- **Haystack**
  - High-performance object storage system optimized for efficient photo storage.

- **HBase**
  - Non-relational, distributed, scalable big data store (Hadoop database).
  - Provides a fault-tolerant way of storing large quantities of data (big data).



Facebook – Data Storage (cntd.)

- **MySQL**
  - Open source relational database management system (RDBMS).

- **memcached**
  - Distributed in-memory caching system.
  - Stores data from MySQL DB in cache to reduce number of times an external data source must be read.

- **TAO (The Associations and Objects)**
  - Distributed data store that provides efficient and timely access to the social graph using a fixed set of queries.
  - Serves thousands of data types and handles read and write requests very fast.

Facebook – Data Storage (cntd.)

- **Wormhole**
  - Publish-subscribe (pub-sub) system that identify updates and transmit notifications to interested applications.
  - Supports reliable replication of changes among several other services including TAO, Graph Search and Memcache.

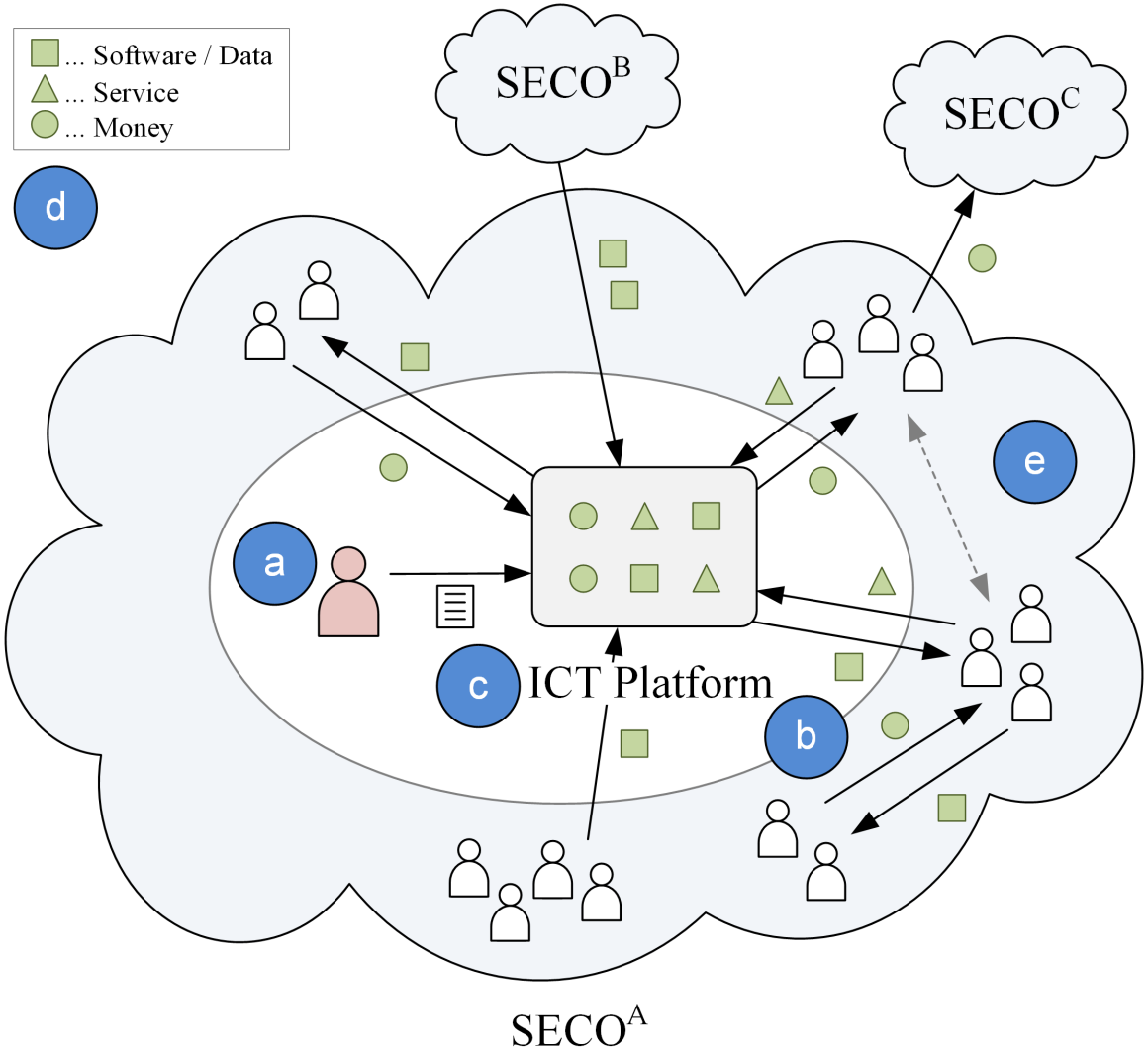
- **f4**
  - Efficient object storage system for photos, videos, and other Binary Large OBjects (BLOBs).
  - Designed to be simple, modular, scalable, and fault tolerant.
  - Request rate for its content is lower than that for content in Haystack.

Software Ecosystems

- **Definition:** *“A networked community of organizations or actors, which base their relations to each other on a common interest in the development and use of a central software technology”*. [Hansen12]

- **Examples:** iOS App Store, Android, Eclipse IDE, Django, Rails, Facebook Apps.

Software Ecosystem Elements



Musil, J., Musil, A., & Biffl, S. (2013). *Elements of Software Ecosystem Early-Stage Design for Collective Intelligence Systems*. In Proc. of the Int'l Workshop on Ecosystem Architectures (WEA '13) (pp. 21–25). ACM.

Software Ecosystem Elements (cntd.)

a) Central Reference Organization

Main beneficiary of the SECO. Oversees and guides SECO growth and evolution. Also, exerts control on SECO actors. i.e. private companies (Apple), consortia (Apache Foundation).

b) Networked Character

Network effect between SECO actors and organisations. Organization internal and external links. Causes hard-to-replicate “stickiness” of platform.

c) ICT Platform

Technological core that is used beyond a single organization. Foundation around the SECO evolves. i.e. app store, social networking service, technical standards, programming frameworks.

Software Ecosystem Elements (cntd.)

d) Shared Values

Can be extrinsic (software product, complementary services, business domain), or intrinsic (contribute to common good, increase of reputation).

e) Self-Regulation

Community-centric way of collaboration and coordination. Direct and indirect interactions among SECO actors. SECO governance and management bottom-up or top down.

Software Ecosystem - Granularity Levels

1. **Code**
    - Libraries and components using a particular programming language or framework. e.g. Swift, Ruby Gems, Docker files.

2. **Packaging**
    - Packaging technology compliant to a particular platform. e.g. iOS/Android App, Ubuntu Snappy, Docker container.

3. **Application Extensibility**
    - Provide extension points within applications and platforms to extend functionality or access data. e.g. iOS 10 iMessage extensions, Eclipse Plugins, Twitter Streaming API.

4. **Platforms (Operating System, Hubs, Marketplaces, CIS)**
    - Platforms mediating the exchange of software technology, data and information about it. e.g. iOS, Play Store, Facebook Apps, Wikipedia.

5. **Ecosystem System-of-Systems**
    - Multiple SECOs linked together creating a compound mega system. e.g. Wikimedia, Facebook, Google, Apple.

Privacy by Design & GDPR[1]

- **Motivation**
  - Threat privacy as first class concern throughout the whole engineering process.
- **Contrast to: Privacy by Default**
  - Stricktest privacy settings should be applied as default settings.

- **Privacy by Design and Privacy by Default are both codified in General Data Protection Regulation (GDPR) with applies by May 2018.**
  - Privacy becomes major concern for software development and also the organisational structure.
  - Rigorous sanctions in case of non-compliance.

- **Known Limitations:**
  - Top-down policy making: Guidelines are vague and ignore SW dev processes.
  - Privacy engineering approaches ignore current SW dev methods and lifecycle. -> each architect has to figure out solution on their own.
  - No agreed privacy/data auditing procedures.

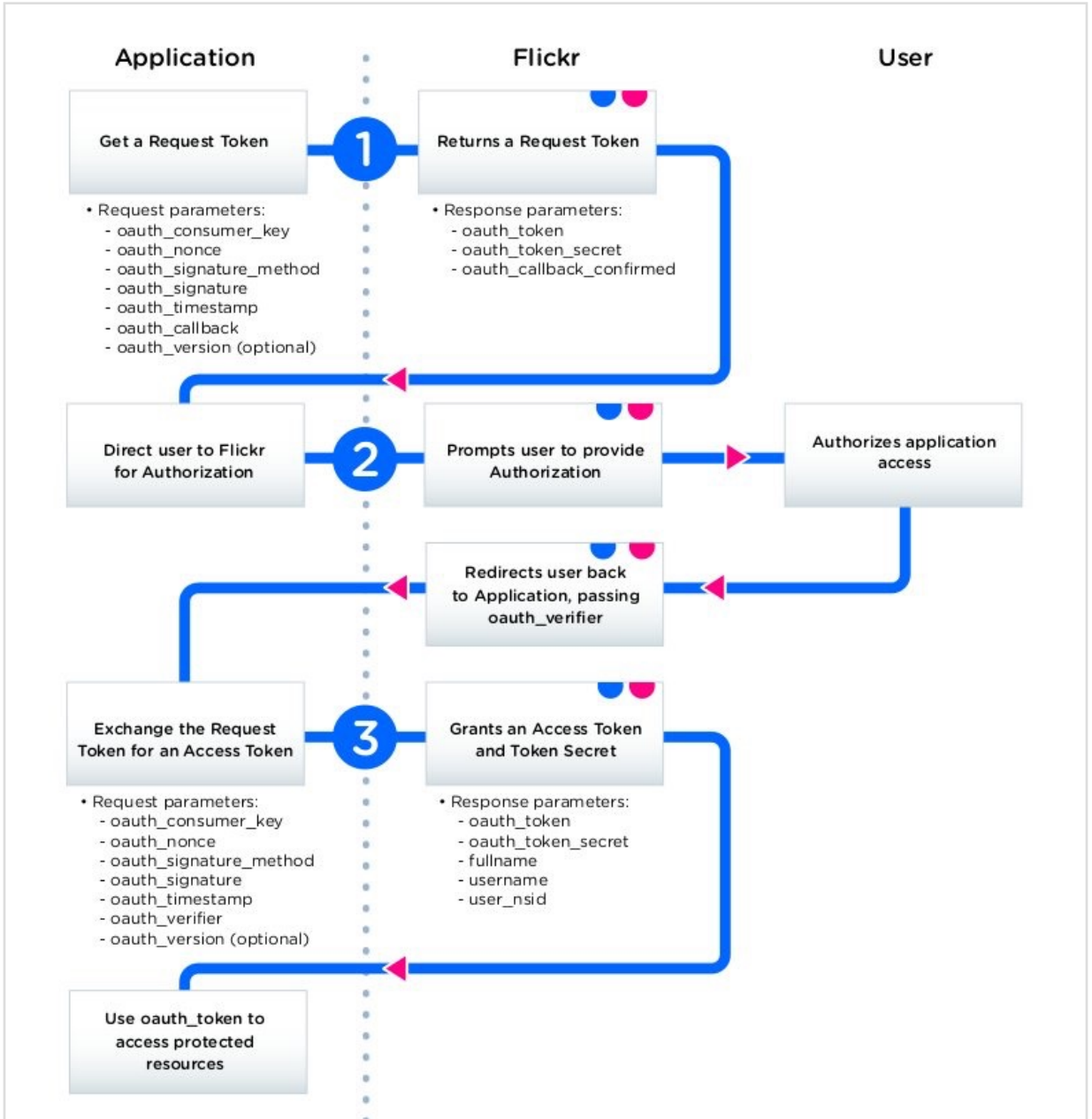
1. Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation). http://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX:32016R0679

OAuth Authorization Framework

- Open standard for authorization (current protocol version: 2.0).
- Specifies process for users to authorize third-party websites or applications to obtain limited access to their information on other websites but without sharing their credentials.

- OAuth defines 4 roles:
  - **Resource owner:** Capable of granting access to a protected resource (e.g., an end-user).
  - **Resource server:** Server hosting the protected resources, capable of accepting and responding to protected resource requests using access tokens.
  - **Client:** An application making protected resource requests on behalf of the resource owner and with its authorization.
  - **Authorization server:** Server issuing access tokens to the client after successfully authenticating the resource owner and obtaining authorization.

OAuth Process

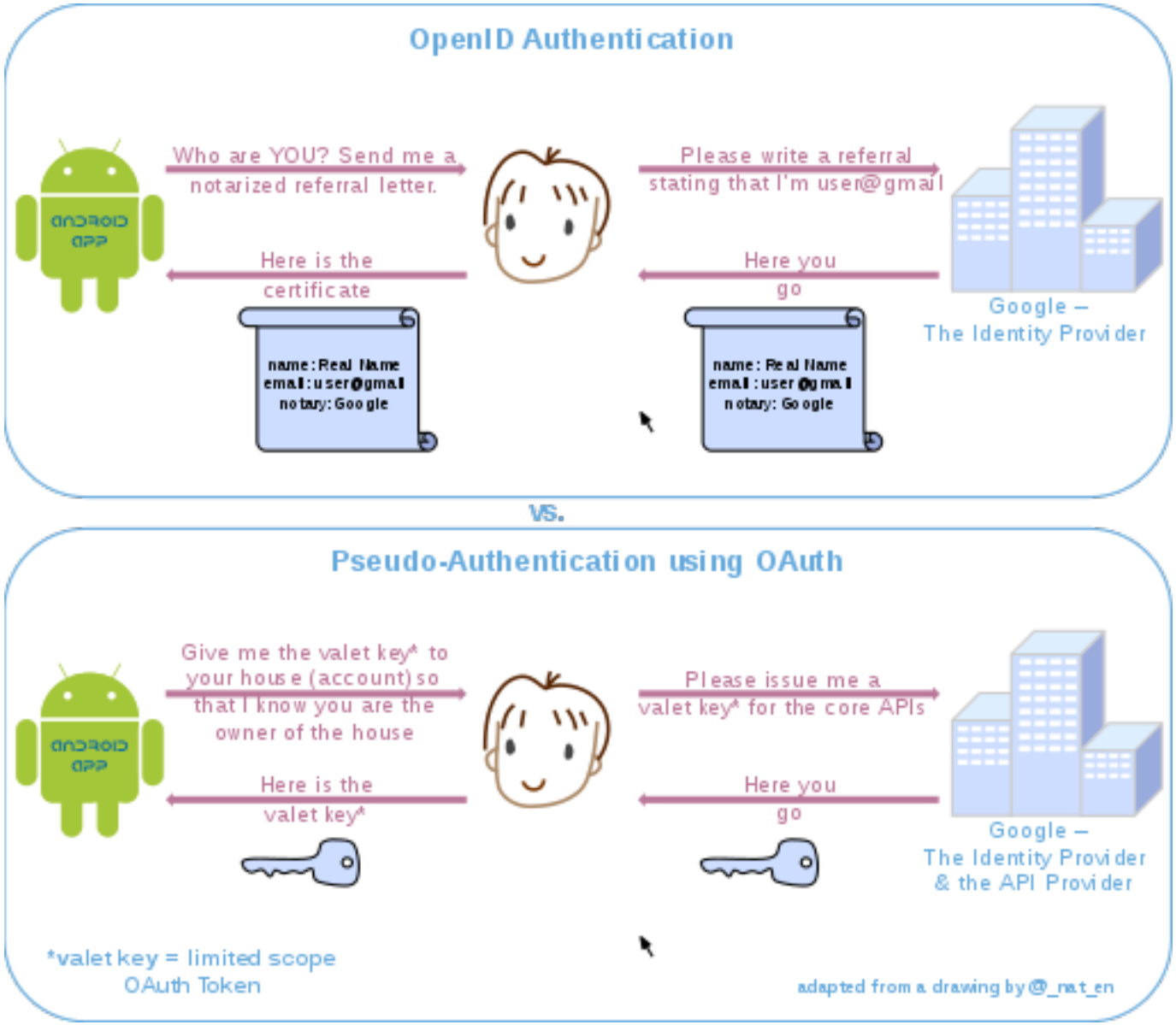


[http://s.yimg.com/pw/images/en-us/flickr_oauth_flow.jpg]

OpenID

- **Open standard** and **decentralized authentication** protocol.
- Specifies process for users to control an identifier that can be used to **verify their identity** so that third-party websites or applications don’t need access to a user’s credentials.
- User creates account by choosing OpenID identity provider and use this account to sign into any other website which accepts OpenID authentication - needs to remember just **single login credentials**!
- **Identifier:** http" or "https" URI
- **OpenID provider:** OpenID authentication server on which a third-party website relies for an assertion that the user controls an identifier.
- Decentralized design: **no central authority** must approve or register third-parties or OpenID providers.
- Uses only standard HTTP(S) requests and responses.

OpenID vs. OAuth



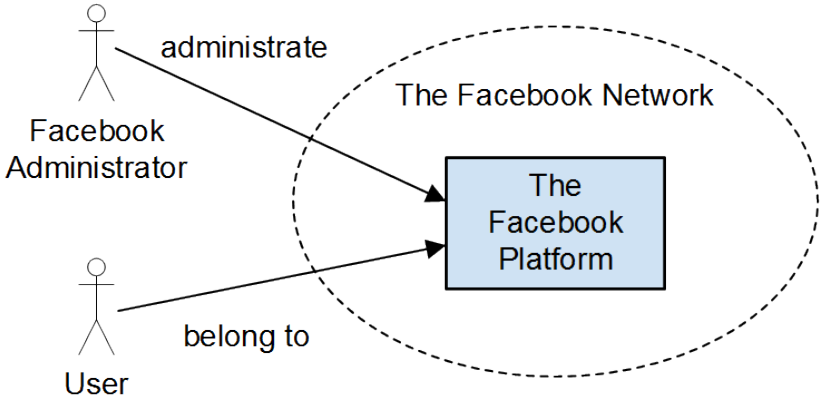
[https://upload.wikimedia.org/wikipedia/commons/thumb/3/32/OpenIDvs.Pseudo-AuthenticationusingOAuth.svg/512px-OpenIDvs.Pseudo-AuthenticationusingOAuth.svg.png]

XACML (eXtensible Access Control Markup Language)

- Standard consists of:
  - An **attribute-based access control policy language** for expressing policies for information access over the Internet.
  - An **access control architecture** and and a processing model describing how to evaluate access requests according to the **rules defined in policies**.
- Formal definitions for the XACML language are provided in XML Schema Definitions.
- Provides **common terminology and interoperability** between access control implementations by multiple vendors.

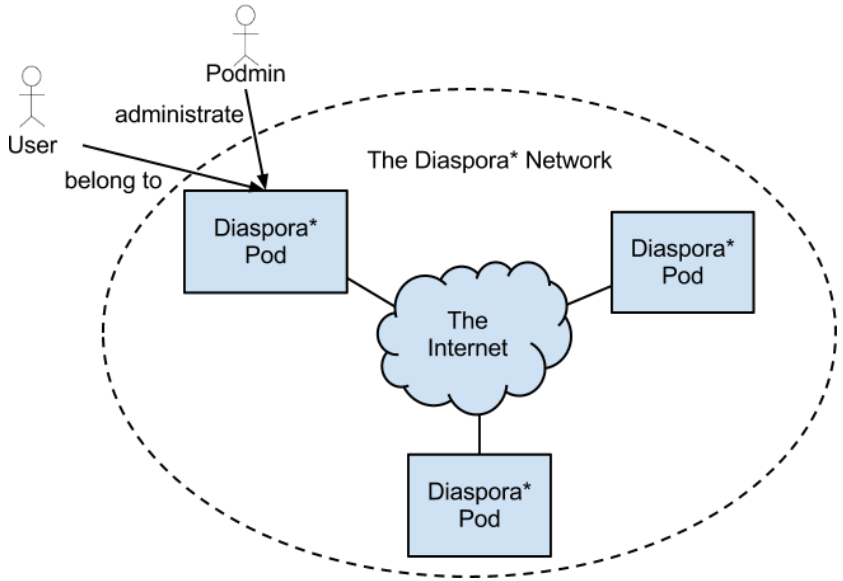
Centralized CIS

- CIS which acts as **“one” platform** and is **operated by a single provider**.
- **Central** administration, development and content curation.
- **Data** concentrated in a **single system**.
- Most CIS are centralized: Facebook, Wikipedia, Yelp.



Decentralized/Federated CIS

- A CIS that is **decentralized** and **distributed across multiple providers**.
- NOT the same as centralized CIS running on multiple servers.
- Overall system **consists of independent CIS instances** (node/pod/…) which are able to communicate with each other.



Decentralized CIS Examples

GNU Social

- FOSS microblogging platform.
- Merged from StatusNet, Free Social and GNU social projects.
- Tech: PHP, OStatus, XMPP.

Diaspora

- Social networking service + personal web server (Unicorn)
- Diaspora network is build out of a network of individual Diaspora system instances (pods).
- Tech: Ruby on Rails, Unicorn, backbone.js.

Formats and Protocols for Decentralized CIS

JavaScript Object Notation (JSON)

- Format to transmit data objects consisting of attribute-value pairs in human readable text.
- Standardized (IETF RFC 4627).
- Used for data transmit between server and web application.

JSON Activity Streams

- Specification of how to serialize social activity streams in JSON format.
- Version 2.0 introduced as IETF Draft (Nov. 2013).
- Used in: Facebook, MySpace, Windows Live, OStatus-based Systems.

Formats and Protocols for Decentralized CIS (ctnd.)

PubSubHubbub

- Protocol for server-to-server, webhook-based publish/subscribe for web resources.
- Used to provide **real-time updates via pushed HTTP notifications** (= no more client-side polling of feeds).

- 3 Concepts: Publishers, Subscribes and Hubs.
- **Hub:**
  1. Hubs can be run by publishers or 3rd parties (e.g. Google).
  2. Publisher provides topic URL to hub server.
  3. Subscribers subscribe to hub (instead to publisher).
  4. Hub multicasts publisher updates to registered subscribers.
- Used in: Tumblr, MySpace, Diaspora, Wordpress.com

Formats and Protocols for Decentralized CIS (cntd.)

Salomon

- Message exchange protocol.
- Enables decentralized commenting/annotating against feed articles.
- Used in: StatusNet, Diaspora.

Webfinger

- Protocol used for discovery of addressable entities (people/ things) via URI (Link-Based Resource Descriptor – LRDD).
- Specified in IETF RFC 7033.
- REST-based web service.
- Returns JSON document as response to valid query.
- Used in: StatusNet, Diaspora (discover users on federated nodes).

Communication in Decentralized CIS

OStatus

- Specification for distributed status updates or microblogging.
- Lets people on different social networks follow each other.
- Superseded OpenMicroBlogging protocol.
- Systems: GNU Social, Friendica.

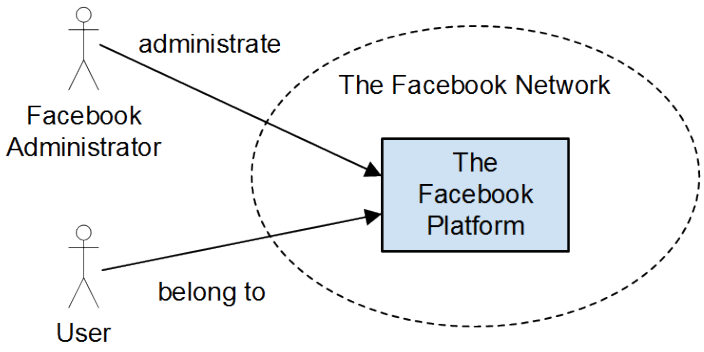
Bringing it all together: Following a feed in a microblog

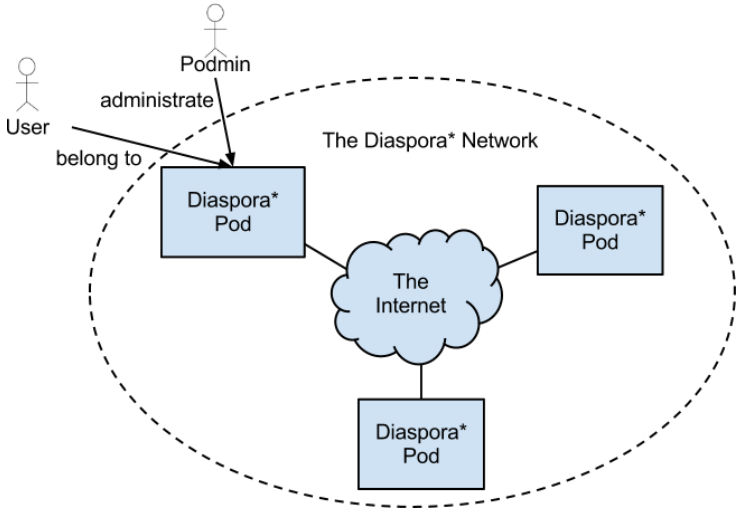
Bob wants to subscribe to Alice's public feed and Alice should be informed about it.

1. Bob discovers the LRDD of Alice using Webfinger.
2. Bob subscribes to Alice's public feed using PubSubHubbub.
3. Bob sends an ActivityStream Atom Entry to Alice using the Salomon protocol to notify her about the event.

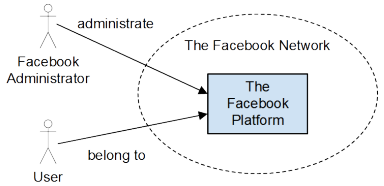
Trade-Off Centralized/Distributed CIS

Going centralized or distributed comes with trade-offs – favoring one style over the other depends on the individual application context.





Trade-Off Centralized/Decentralized CIS

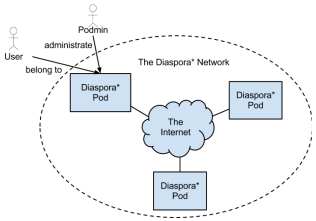


Centralized CIS

+ Constant quality of service for all participants.
+ Single point of access.
+ More resources for system maintenance, data security, platform evolution.
+ Single organization to hold accountable (e.g. privacy issues).
+ Effective information exchange due to very-large scale user base.

- Single point of failure.
- Prone to censorship and systematic infiltration by government organizations.
- Often closed/proprietary system source code.
- Lots of influence concentrated in a single organization

Trade-Off Centralized/Decentralized CIS (cntd.)



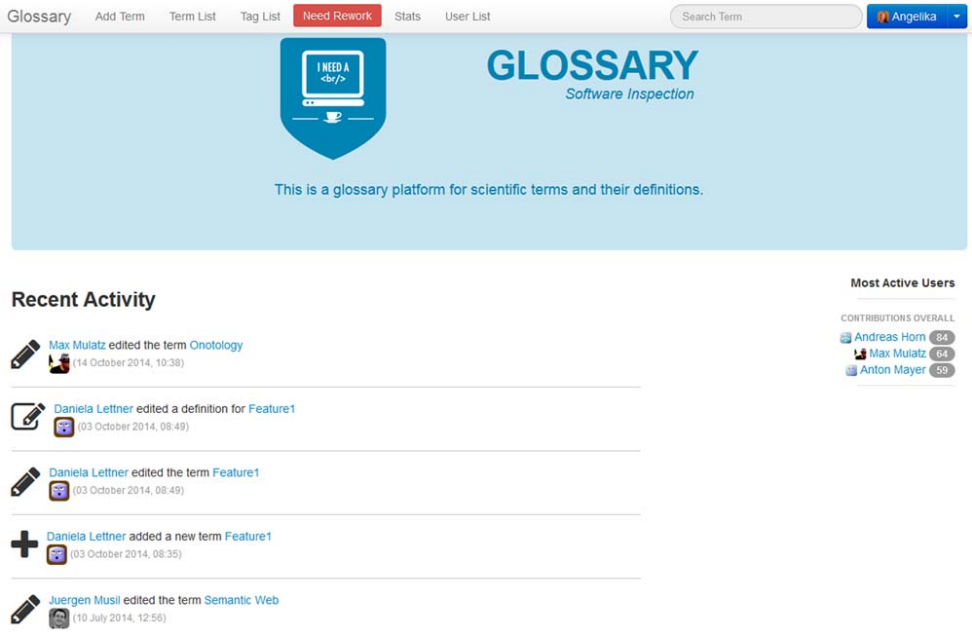
Decentralized CIS

+ Multiple points of access.
+ Robust: Infiltration only of individual points; Easy hosting of new nodes.
+ Often system source code is open source (= auditable).
+ Easier to operate on non-profit basis due to lower resource needs of individual nodes.

- Quality of service dependent on individual node.
- Each node is responsible for its maintenance and data security.
- Less effective information exchange due to fragmentation of user base.
- User contributions are stored on an individual node.

Example Tech Stack: Scientific Glossary

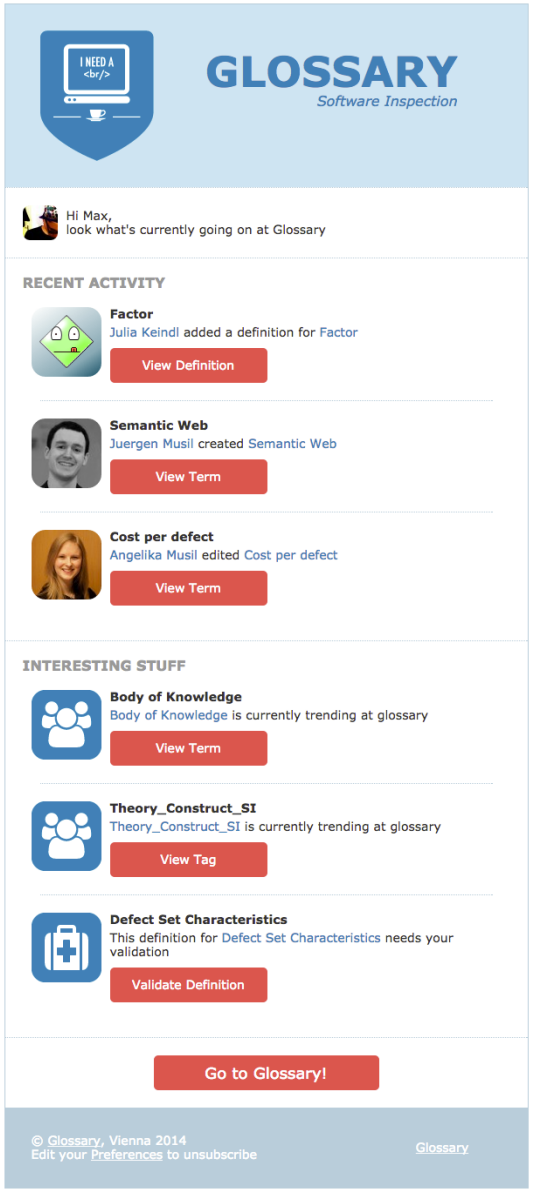
- Ruby v2.2.3
- Rails v4.2.4
- Database: PostgreSQL
- Git Repository
- Heroku as PaaS
- Useful Gems:
  - *pg:*
    interface to PostgreSQL DB
  - *bootstrap-sass/jquery-rails:*
    user interface design
  - *haml:*
    well-indented, clear mark-up
  - *cancancan:*
    authorization and role management
  - *public_activity/paper_trail:*
    activity tracking



Example Tech Stack: Scientific Glossary (cntd.)

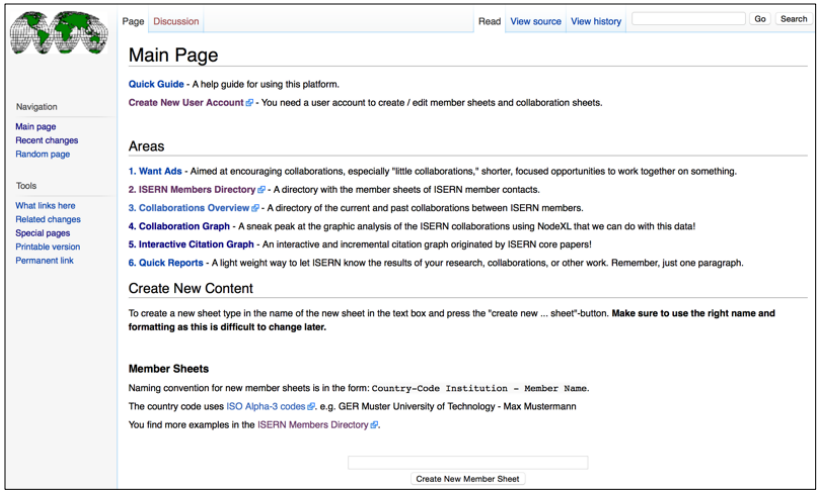
Notification concept to support awareness in the collaboration process:

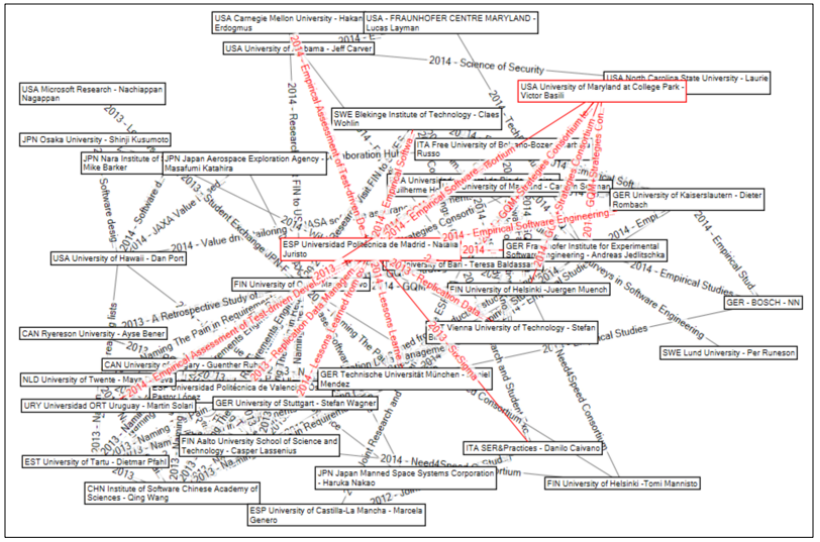
- Activity Feed
- Top Contributors
- Notification Messages:
  1. Global Digest
  2. Personal Digest
  3. Ranking Mail



Example: ISERN Collabhub - Repurposed MediaWiki

- **Goal:** Collect profiles of ISERN members and track their collaborations.
- **Approach:** Repurpose MediaWiki application to meet CI needs:
  - Pages -> Membersheets
  - Categories -> Collaborations
  - Categories -> Collab Categories
- No programming needed, but redefinition of semantics, design of „house rules“, templates and workflows to steer user contributions to meet information aggregation goals.
- **Limitations:**
  - Rework analytics.
  - Poor dissemination mechanisms.





Getting Started with Ruby on Rails

1. Start with well-known example of a Blog application:
   http://http://guides.rubyonrails.org/getting_started.html
   - Installing Ruby and Rails
   - Creating Blog application example
   - Configuring/creating a database
   - Running application locally

2. Look also at this good tutorial to learn Ruby on Rails:
   http://https://www.railstutorial.org/book

3. Developing first CIS step by step:
   http://medium.com/rails-ember-beyond/how-to-build-a-social-network-using-rails-eb31da569233

Conclusion and Summary

- ISO/IEC WD 42020 draft
  - Architecture Processes
- ISO/IEC/IEEE 42010:2011 standard
  - Specification of key architecture concepts
- Architecture Framework for CIS
  - Following ISO/IEC/IEEE 42010 standard
  - Addressing CI system concerns: information aggregation, knowledge dissemination, perpetual feedback loop
  - 3 Viewpoints: Context, Technical Realization, Operation

Conclusion and Summary (cntd.)

- No definite technology architecture:
  - But: start light and grow over time.
- Software Ecosystems
- Centralized and distributed architectures.
- Distributed architectures more complex to engineer.
- Using a Ruby on Rails stack:
  - Relatively easy to start with.
  - Lots of existing program and knowledge resources to build upon.

SW Architectures for CI Systems I/II – Key Takeaways

- What are CI Systems?
  - Main constructive directions, challenges, success/risk factors.
  - How do CI Systems differ to Crowdsourcing Systems (process, elements) ?
- CI design patterns.
- ISO/IEC WD 2020 draft on architecture processes
- ISO/IEC/IEEE 42010 standard specifying key architecture concepts.
- Architecture framework for CI systems.
  - Concerns, stakeholders and viewpoints.
- Centralized/decentralized CI systems.
  - Characteristics and trade-offs.
- Software Ecosystems
  - OAuth
- Communication in decentralized CI systems.
  - OStatus.
  - Formats and protocols.
- Ruby on Rails VS Java EE.

References

1. Bass, L., Clements, P.,Kazman, R. (2012). Software Architecture in Practice (3rd ed.). Addison-Wesley Professional.

2. ISO/IEC/IEEE 42010 (2011). Systems and Software Engineering - Architecture Description. Retrieved from http://www.iso-architecture.org/42010/.

3. Jansen, A., & Bosch, J. (2005). Software Architecture as a Set of Architectural Design Decisions. In *Proceedings of the 5th Working IEEE/IFIP Conference on Software Architecture (WICSA '05)*. IEEE.

4. Malone, T. W., Laubacher, R., & Dellarocas, C. (2009, February). Harnessing Crowds : Mapping the Genome of Collective Intelligence.

5. Musil, J., Musil, A., Weyns, D., & Biffl, S. (2015). An Architecture Framework for Collective Intelligence Systems. In *Proceedings of the 12th Working IEEE/IFIP Conference on Software Architecture (WICSA '15)*. IEEE.

6. Musil, J., Musil, A., & Biffl S. (2015) Introduction and Challenges of Environment Architectures for Collective Intelligence Systems. In *Agent Environments for Multi-Agent Systems IV*. Springer.

References (2)

7. Wooldridge, M. & Ciancarini, P. (2002). Agent-oriented software engineering. In *Handbook of Software Engineering and Knowledge Engineering: Fundamentals* (Volume 1). World Scientific.

8. ISO/IEC CD 42020. (2016). Systems and software engineering - Architecture Processes. Retrieved from http://www.iso.org/iso/catalogue_detail.htm?csnumber=68982

9. Manikas, K., & Hansen, K. M. (2013). Software Ecosystems - A Systematic Literature Review. Journal of Systems and Software, 86(5), 1294–1306. http://doi.org/10.1016/j.jss.2012.12.026

Online Resources

- **CI Tech Stacks**
  - LinkedIn: A Professional Social Network Built with Java Technologies and Agile Practices. http://www.slideshare.net/linkedin/linkedins-communication-architecture
  - Facebook Architecture – Breaking it Open. http://www.slideshare.net/AditiTechnologies/facebook-architecture-breaking-it-open
  - Diaspora Architecture Overview https://wiki.diasporafoundation.org/Architecture_overview
  - Twitter Engineering Blog https://engineering.twitter.com/
  - Etsy Engineering Blog http://codeascraft.com/
  - High Scalability Blog on Platform Architectures http://highscalability.com

Online Resources

- **Formats & Protocols**
  - JSON: http://json.org/
  - Activity Streams: http://activitystrea.ms/
  - PubSubHubbub: http://code.google.com/p/pubsubhubbub/
  - Salomon Protocol: http://www.salmon-protocol.org/
  - Webfinger: http://code.google.com/p/webfinger/
  - OStatus: http://www.w3.org/community/ostatus/

  - OAuth: https://oauth.net/2/
  - OpenID: http://openid.net
  - XACML: http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-en.html

Helpful Links: Ruby on Rails / Python & Django

- Ruby: http://www.ruby-lang.org/
- Ruby on Rails: http://rubyonrails.org/
- Documentation: http://guides.rubyonrails.org/
- Overview of categorized gems: http://www.ruby-toolbox.com/
- Gem hosting service: http://rubygems.org/
- Screencasts: http://railscasts.com/

- Python: http://www.python.org/
- Django: http://www.djangoproject.com/
- Documentation: http://docs.djangoproject.com/
- Overview of Django Packages: http://www.djangopackages.com/
- Tutorial: http://www.djangobook.com/