Coding Book

Composite Ontology Classes

A. **Design Issue**: expressed in ARPs through description for relevant architecture configurations. The described configurations concern either part of a planned design or an existing software system. Design issues could be described at different levels of abstraction. Most ARP questions provide a long description for the design issue in a couple of sentences, while during answers short references for known design issues are commonly used.
B. **Requirements**: Three types of requirements were found: 1) Quality attribute requirements are mentioned explicitly using the standard quality attribute terms. 2) Technology features requirements show the need of user to have certain technology features as part of the proposed solution. 3) Business requirements are expressed using their business domain terms.
C. **Constraints**: We found three types of constraints: 1) Team skills constraints express the level of knowledge of the user to a certain technology solution. 2) Development time constraint is indicated by expressing explicitly the priority of development time to the user. 3) Solution constraint is also expressed explicitly by indicating which technology solutions must be considered in the solutions.
D. **User Request**: exist in ARP question or title in a form of questions. User request complements design issue, requirements and constraints by showing the type of architecture activity (evaluation or synthesis) considered by the user in this post. The request might embody short references for design issues and requirements.
E. **Technology Solutions Features**: We found two main types of technology features: 1) Development features are expressed through certain programming activities (e.g. debugging) or programming features and tools (e.g. inheritance, code generation), 2) Behavioral features are expressed through technology specific component and class names. In addition, behavioral features could be further classified among different quality attributes (e.g. Interoperability features). Moreover, some of the behavioral features are explained through their implemented architectural patterns or their relationship with other technologies.
F. **Technology Solutions ASTAs:** users mention technology solutions' benefits and drawbacks, as part of their argument for recommending or excluding technology solutions. A key aspect which distinguishes ASTAs is the extensive usage of adjectives and adverbs in combination with technology features and quality attributes. The adjectives or adverbs are used to express the advantages or disadvantages of certain technology solutions or features.
G. **Technology Solutions Use-Cases**: These are either success or failure stories for the usage of technology solutions at certain contexts. The stories could be coming from personal experiences of users, or well-known examples for existing systems. The context associated with stories could include domain description, architecture configurations, infrastructure, and constraints.
H. **Design Decisions:** ADDs came in different forms: 1) Recommended ADDs represent the majority of ADDs. They are recommendation from users based on their experience or opinion for certain architectural solutions. 2) Taken ADDs are ADDs, which have been decided by the user who asked the question. Usually after discussions with other users. 3) Planned and existing system ADDs come usually as part of the design issue description. They represent ADDs which have been previously taken or implemented.
I. **Decision Rules**: Conditional recommendation for architectural solutions. They consists of a rule condition and recommendation. The condition might involve other ontology classes such as requirements, constraints, architectural configuration, and existing system description. On the other hand, recommendations involve recommended ADDs for certain technology solution or architecture configuration.

J. **Architecture configuration**: represents part of an architectural model. The ontology class is represented through a sentence, which consists of one or more component names or application types associated with a connector verb or name. (e.g. "Pushing data from the server to the client", "Rubby app sends a request to Java app")
K. **Component behavior**: A sentence which describes the behavior of a component. It give an overview about the type of implemented logic and complexity. In addition, sometimes component interface and internal operations are mentioned during the description. (e.g. "service can be viewed as the business layer of the application", "process will run asynchronously").
L. **Existing system description**: describe the architecture of an existing system, which a user is dealing with. The description depends on other ontology classes such as architecture configuration and existing system ADD. In addition, technologies and application types are commonly mentioned within the description. (e.g. "I am working on a RESTfull application", "An existing process changes the status field of a booking record in a table, in response to user input.", "I am using Apache MINA in my open source project")

The table below list examples for the aforementioned defined ontology classes.

AK Ontology Class	Example
Design Issue	4741713 → "I want to send a batch of 20k JMS messages to a same queue. I'm splitting the task up using 10 threads, so each will be processing 2k messages. I don't need transactions."
Requirement	4473567 → "Our criteria: 1. Short roundtrip time. 2. Low roundtriptime standard deviation. (We understand that garbage collection pauses and network usage spikes can affect this value). 3. High availability. 4. Scalability (we may want to have multiple instances of Ruby and Java app exchanging pointtopoint messages in the future). 5. Ease of debugging and profiling. 6. Good documentation and community support."
Constrain	Team skills constrains: 13016406 → "I have never used Netty" Solution constrain: 12783677 → "This needs to adhere to WCF REST standards"
User Request	1582952 → "How do I choose between WCF, REST, POX and RIA services for a new Silverlight application"
Technology Feature	1429318 → "EMS is centralized (hub and spoke) on a specific server(s) and can traverse subnets no problem" 10375137 → "ActiveMQ is a widely used message broker that offers FIFO queues"
Technology ASTA	Benefit: 100993 → "It is much easier to debug Webservices over the wire as the data is SOAP/HTTP , which can be easily captured via sniffing tools for debugging" Drawback: 19758215 → "performance difference will be negligible and in many cases worse for NIO (Netty with thread sharing)"
Technology Use-case	12783677 → "An application I'm working on has a similar architecture, and I'm planning to use SignalR to push updates to clients, using long polling techniques (...) I have implemented this now, and it works very well"
Design	361491 → "I would highly recommend using WCF; and use the WCF

Decision	Service Library project over the Silverlight-enabled web service”
Decision Rule	17806977 → “If performance is your main criteria, you should definitely look at ZeroMQ.”

Simple ontology classes:

1. Technology name: represented with the name of a technology (e.g. “WCF”, “Netty”, “Biztalk”).
2. Technology type name: they represent a family of technologies (e.g. “Message queues”, “SOAP Library”, “message protocol”)
3. Pattern name: represented with the name of a pattern (e.g. “Messaging”, “Rest”, “FIFO” “Queuing”).
4. Quality attribute name: represented with the name of quality attributes based on ISO standard (Ref) (e.g. “Performance”, “Reliability”, “Interoperability”, “Scalability”)
5. Application types: They are usually represented with words “application” or “app” associated with an adjective to define the type of the application. (e.g. “distributed application“, “web app”, “Java application”, “Mobile app”).
6. Component name: As part of describing a software architecture. Processing units or storage components are mentioned using different terms. (e.g. “server”, “database”, “service”, “back end”, “system”, “client”, “process”)
7. Component element: They are elements which constitute a component. (e.g. “operation”, “method”, “job”, “event”, “interface”, “field”)
8. Connector verb/name: used to express communication to or from a certain component. They could be expressed using verbs as well as terms. (e.g. “send”, “receive”, “read”, “write”, “communication”, “connection”)
9. Connector data: they are the data need to be transferred through the connector to or from a component. (e.g. “data”, “request”, “response”, “message”, “object”)
10. Infrastructure term: they are terms used to describe an infrastructure or networking. (e.g. “firewall”, “internet”, “NAT”, “port”, “load balanced”, “data center”)
11. Cost: identified with words like “budget” or “cost”.
12. Programming activity: Describing common programming activities. (e.g. “debug”, “deploy”, “write code”)
13. Programming element: Describing common programming concepts. (e.g. “class”, “method”, “attribute”, “inheritance”, “query”)
14. Feature terms: Behavioral features are expressed through several technology component terms, as well as class names. (e.g. “Authenticator”, “SoapServer”, “serialize”, “endpoint”, “binding”, “socket”, “stream”). On the other hand, development features are expressed through programming elements, development tools, and programming activities.

Decision bound between ontology classes

1. Difference between technology solution features and requirements: Technology features describe the capabilities of the technologies. Usually verbs like “support” or “provide” are used to express that a certain technology offer a certain capability. On the other hand, requirements describe the needs of the user. Usually verbs like “require” or “need” or “would like” are used to express the wishes of users for certain quality attributes or technology features.
2. Difference between technology solutions use-cases and design decisions: Technology solution use cases describe a story in the past from the experience of the user. The story describes several context details. On the other hand, design decisions are presented as recommendations for using a certain technology solution using simple present tense (e.g. “use”, “go with”) as a response to the requirements and user request mentioned in the question section of the post.

3. Difference between design decisions and decision rules: Decision rules are a more complex structure than design decisions. Decision rules always have a conditional statement, while design decisions are coming in imperative or normal statements. Moreover, decision rules contain design decisions. However, design decisions could also come independently without decision rules.
4. Difference between technology solution features and benefits: even though technology solution features and benefits have similar structure. Technology benefits are differentiated through the extensive use of adjectives and adverbs (e.g. “very”, “fast”, “easy”, “better”), as well as the usage of keywords like “advantage” or “benefit”.