WICSA/CompArch 2015

ARCHITECTURAL DECISION GUIDANCE ACROSS PROJECTS

Plenary Session 2 – Helping Architects Architect

IFS INSTITUTE FOR SOFTWARE

Olaf Zimmermann

Distinguished (Chief/Lead) IT Architect, The Open Group

Institute for Software, HSR FHO

Montreal, May 6, 2015

HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

FHO Fachhochschule Ostschweiz

Acknowledgments

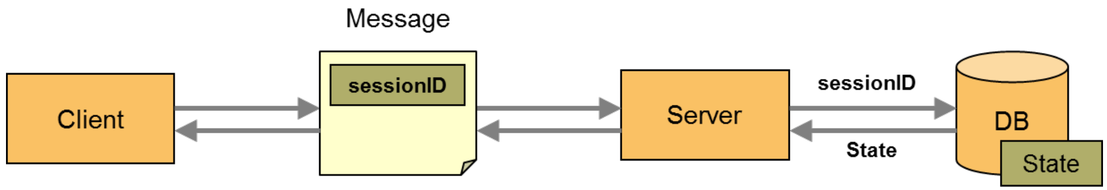
- **Joint work with ABB Corporate Research**
  - Funded by a 2014 Research Grant, Industrial Software Solutions program
  - Open Source Software release planned (pending)

Power and productivity
for a better world™ ABB

- **IT architect community input**
  - ABB business units and group architects
  - Cloud Computing Patterns book (Springer 2014) and supporting website
  - Softwareforen Leipzig, software architecture group meeting Nov. 2014
    - 26 architects from different companies (ICT, insurance, telecommunications)
    - Topic: workflow design
  - SATURN 2013 Architectural Decisions (AD) BoF session attendees
  - WICSA reviewers (2008-present)

Context and Motivation (by Example) (1/2)

- **AD capturing matters, e.g. ISO/IEC/IEEE 42010 has a rationale element**
  - But it remains an unpopular documentation task
    – particularly, but not only in agile communities
  - Effort vs. gain (feeding the beast)?
- **Example (from cloud application design): Session State Management**
  - Shopping cart in online commerce SaaS (e.g., Amazon) has to be stored while user is logged in; three design options described in literature

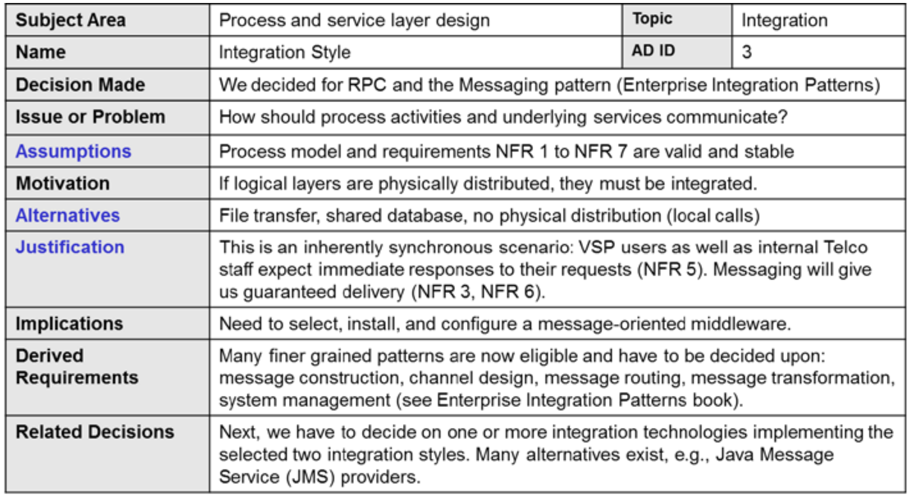


"*In the context of* the Web shop service, *facing the need to* keep user session data consistent and current across shop instances, *we decided for* the Database Session State Pattern from the PoEAA book (and *against* Client Session State or Server Session State) *to achieve* cloud elasticity, *accepting that* a session database needs to be designed, implemented, and replicated."

Reference: (WH)Y-template first presented at SEI SATURN 2012 and later published in IEEE Software and InfoQ, http://www.infoq.com/articles/sustainable-architectural-design-decisions
(inspired by decision part in George Fairbanks' Architecture Haiku, WICSA 2011 tutorial)

Context and Motivation (by Example) (2/2)

- **Filling out a template (e.g. arc42, IBM UMF, Tyree/Akerman) is even more time consuming – still practical for more than 10-20 ADs?**
  - Seven templates from 1998 to 2012 evaluated in paper
  - Selected in “unSLR” (criteria: adoption in practice, diversity, maturity)
  - Reviewed templates contain between 5 and 14 attributes/aspects of an AD



Subject Area	Process and service layer design	**Topic**	Integration
Name	Integration Style	**AD ID**	3
Decision Made	We decided for RPC and the Messaging pattern (Enterprise Integration Patterns)		
Issue or Problem	How should process activities and underlying services communicate?		
Assumptions	Process model and requirements NFR 1 to NFR 7 are valid and stable		
Motivation	If logical layers are physically distributed, they must be integrated.		
Alternatives	File transfer, shared database, no physical distribution (local calls)		
Justification	This is an inherently synchronous scenario: VSP users as well as internal Telco staff expect immediate responses to their requests (NFR 5). Messaging will give us guaranteed delivery (NFR 3, NFR 6).		
Implications	Need to select, install, and configure a message-oriented middleware.		
Derived Requirements	Many finer grained patterns are now eligible and have to be decided upon: message construction, channel design, message routing, message transformation, system management (see Enterprise Integration Patterns book).		
Related Decisions	Next, we have to decide on one or more integration technologies implementing the selected two integration styles. Many alternatives exist, e.g., Java Message Service (JMS) providers.		

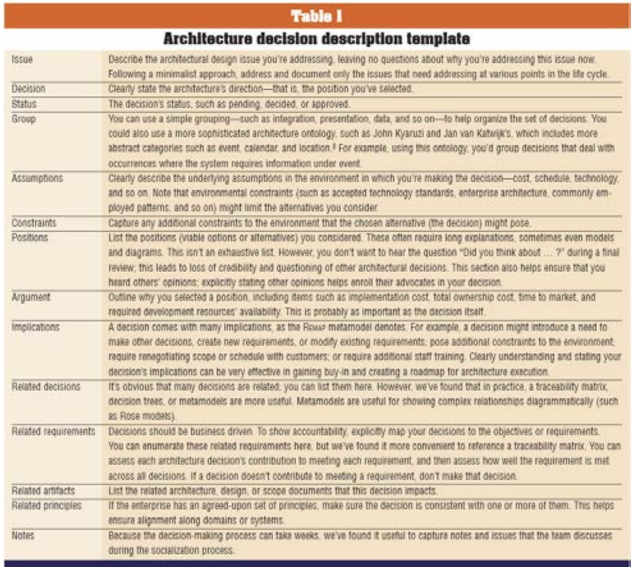


Table I

Architecture decision description template

Issue	Describe the architectural design issue you're addressing, leaving no questions about why you're addressing this issue now. Following a minimalist approach, address and document only the issues that need addressing at various points in the life cycle.
Decision	Clearly state the architecture's direction—that is, the position you've selected.
Status	The decision's status, such as pending, decided, or approved.
Group	You can use a simple grouping—such as integration, presentation, data, and so on—to help organize the set of decisions. You could also use a more sophisticated architecture ontology, such as John Kyaruzi and Jan van Katwijk's, which includes more abstract categories such as *event*, *calendar*, and *location*.[8] For example, using this ontology, you'd group decisions that deal with occurrences where the system requires information under *event*.
Assumptions	Clearly describe the underlying assumptions in the environment in which you're making the decision—cost, schedule, technology, and so on. Note that environmental constraints (such as accepted technology standards, enterprise architecture, commonly employed patterns, and so on) might limit the alternatives you consider.
Constraints	Capture any additional constraints to the environment that the chosen alternative (the decision) might pose.
Positions	List the positions (viable options or alternatives) you considered. These often require long explanations, sometimes even models and diagrams. This isn't an exhaustive list. However, you don't want to hear the question "Did you think about ... ?" during a final review; this leads to loss of credibility and questioning of other architectural decisions. This section also helps ensure that you heard others' opinions; explicitly stating other opinions helps enroll their advocates in your decision.
Argument	Outline why you selected a position, including items such as implementation cost, total ownership cost, time to market, and required development resources' availability. This is probably as important as the decision itself.
Implications	A decision comes with many implications, as the Remap metamodel denotes. For example, a decision might introduce a need to make other decisions, create new requirements, or modify existing requirements; pose additional constraints to the environment; require renegotiating scope or schedule with customers; or require additional staff training. Clearly understanding and stating your decision's implications can be very effective in gaining buy-in and creating a roadmap for architecture execution.
Related decisions	It's obvious that many decisions are related; you can list them here. However, we've found that in practice, a traceability matrix, decision trees, or metamodels are more useful. Metamodels are useful for showing complex relationships diagrammatically (such as Rose models).
Related requirements	Decisions should be business driven. To show accountability, explicitly map your decisions to the objectives or requirements. You can enumerate these related requirements here, but we've found it more convenient to reference a traceability matrix. You can assess each architecture decision's contribution to meeting each requirement, and then assess how well the requirement is met across all decisions. If a decision doesn't contribute to meeting a requirement, don't make that decision.
Related artifacts	List the related architecture, design, or scope documents that this decision impacts.
Related principles	If the enterprise has an agreed-upon set of principles, make sure the decision is consistent with one or more of them. This helps ensure alignment along domains or systems.
Notes	Because the decision-making process can take weeks, we've found it useful to capture notes and issues that the team discusses during the socialization process.

HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

FHO Fachhochschule Ostschweiz

IFS INSTITUTE FOR SOFTWARE

From Decisions Made to Decisions Required (Guidance)

- **Approach: Refactor decision capturing templates into problem-option-driver fragments and change tone, to separate concerns and to ease reuse**

“In the context of the Web shop service, facing the need to keep user session data consistent and current across shop instances, we decided for the Database Session State Pattern from the PoEAA book (and against Client Session State or Server Session State) to achieve cloud elasticity, accepting that a session database needs to be designed, implemented, and replicated.”

Curate {decision need, solutions, qualities} for reuse – but *not* the actual decision outcomes

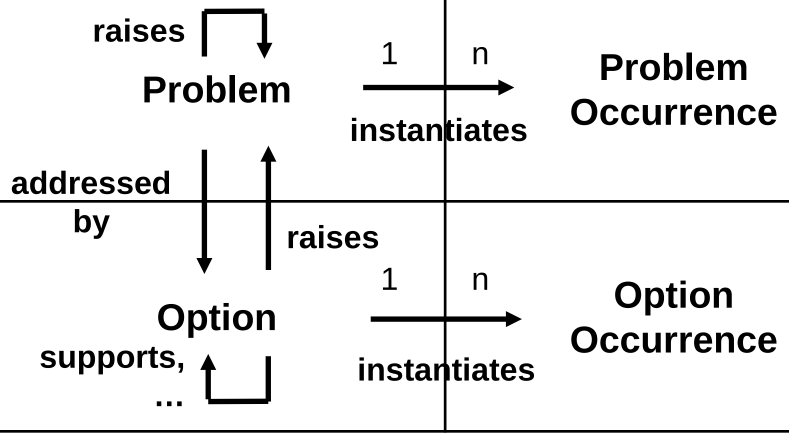
- “When designing a stateful user conversation (for instance, a shopping basket in a Web shop), *you will have to* decide whether and how session state is persisted and managed.” (question: is this a requirement or stakeholder concern?)
- “Your conceptual design options *will be* these patterns: Client Session State, Server Session State, and Database Session State.”
(question: are patterns the only types of options in AD making?)
- “The decision criteria *will include* development effort and cloud affinity.”
(question: what else influences the decision making?)

Research Questions and Contributions Overview

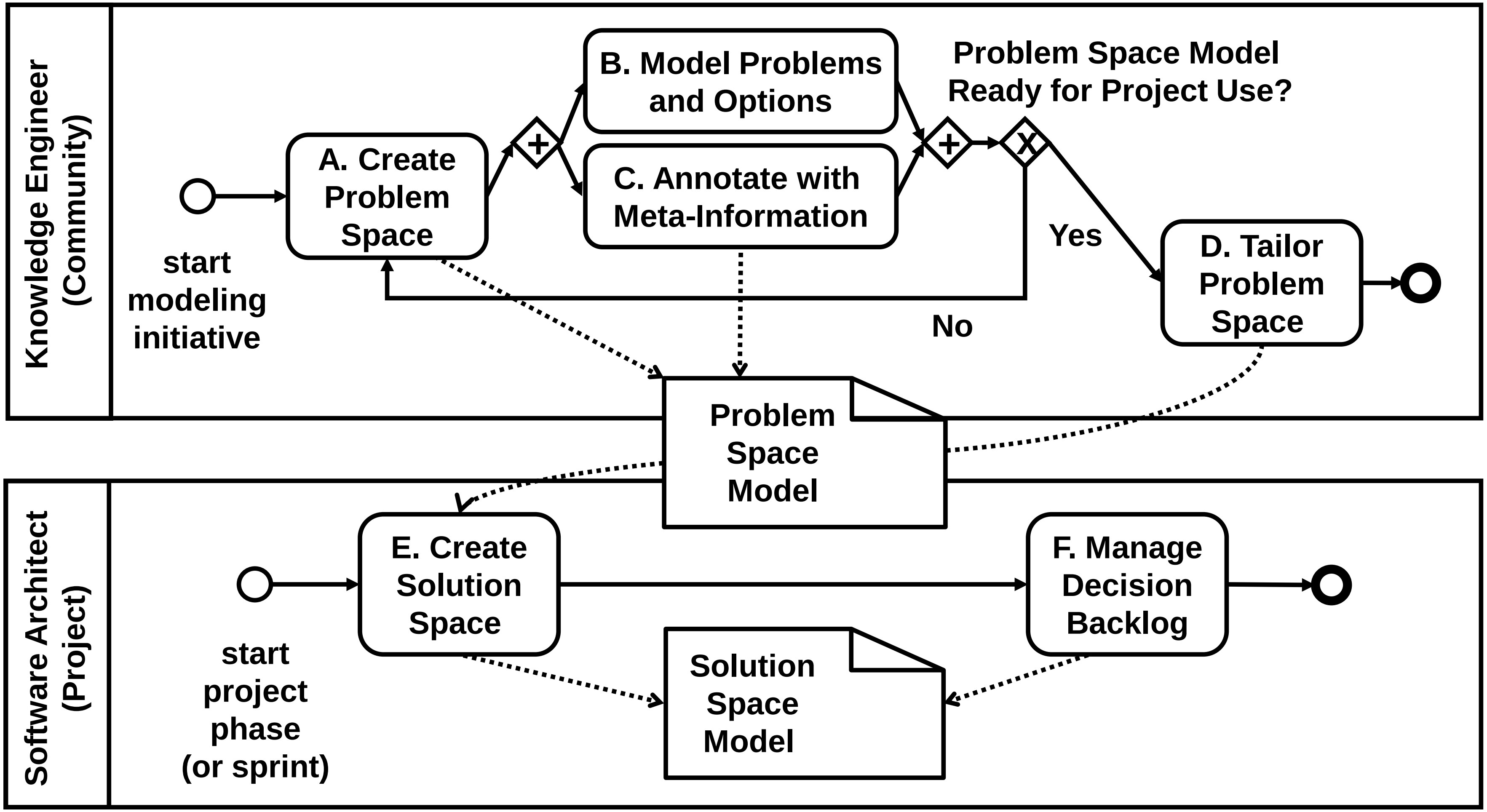
- ***RQ 1:* How to model decisions required so that a) they are applicable to diverse projects, b) do not age fast e.g. due to technology evolution, and c) are simple to maintain over time?**
  - To answer RQ 1, we supersede previous metamodels for decision capturing and sharing with lean knowledge quadruples that give decisions a guiding role that works effectively and efficiently both in traditional and in agile settings.
- ***RQ 2:* How to integrate decision modeling concepts into architecture design practices and tools commonly used by architects to evolve their designs and record decisions made along the way, without creating more effort than gains?**
  - To respond to RQ 2, we annotate the decision knowledge with meta-information, leveraging already existing organizing principles such as viewpoints, refinement levels, and project stages. Decision capturing is streamlined by leveraging lean documentation templates (from practitioner literature) flexibly.

Contributions (1/4): Refactored Domain Model (AD Quadruple)

Model Type	**Problem Space**	**Solution Space**
Reach/Level	Asset (Community)	Project
Owner	Knowledge Engineer	Software Architect
Purpose	Design Guidance	Decision (Back-)Log
Need for Architectural Decision	**Problem**	**Problem Occurrence**
Design Candidates	**Option**	**Option Occurrence**



Contributions (2/4): Knowledge Processing Workflow (BPMN)


Knowledge Engineer (Community)
start modeling initiative
A. Create Problem Space
B. Model Problems and Options
C. Annotate with Meta-Information
Problem Space Model Ready for Project Use?
Yes
No
D. Tailor Problem Space
Problem Space Model
Software Architect (Project)
start project phase (or sprint)
E. Create Solution Space
F. Manage Decision Backlog
Solution Space Model

HSR HOCHSCHULE FÜR TECHNIK RAPPERSWIL

FHO Fachhochschule Ostschweiz

IFS INSTITUTE FOR SOFTWARE

Contributions (3/4): Meta Information – Predefined, but Extensible

Name	Purpose, Rationale	Sample Value(s)
Intellectual Property Rights	Intellectual Property Rights (IPR) for model element, e.g. confidentiality level, copyright statement	Public, Company-Confidential, © Company X, 2015
Knowledge Provenance	Reference to a cited source and/or acknowledgment of contributor	CCP book, PoEAA website, Project Y, Architect Z
Refinement Level	The abstraction level on which this problem typically occurs	Conceptual Level, Technology Level
Project Stage	Gate, milestone, phase and/or elaboration point in incremental and iterative design (in which this problem is typically tackled)	Inception, Elaboration, Construction (in OpenUP)
Organizational Reach	Sphere of influence of the problem	Enterprise, Division, Business Unit, Project, Subsystem
Owner Role	The role (as defined e.g. in OpenUP) that is responsible and accountable for the decision	Application Architect, Integration Architect
Stakeholder Roles	People with an interest in this problem (note: the accountable person is annotated as owner role)	Enterprise Architects, Frontend Developers, Testers
Viewpoint(s)	e.g. one of the 4+1 views on software architecture or a Rozanski/Woods viewpoint	Logical Viewpoint, Deployment Viewpoint

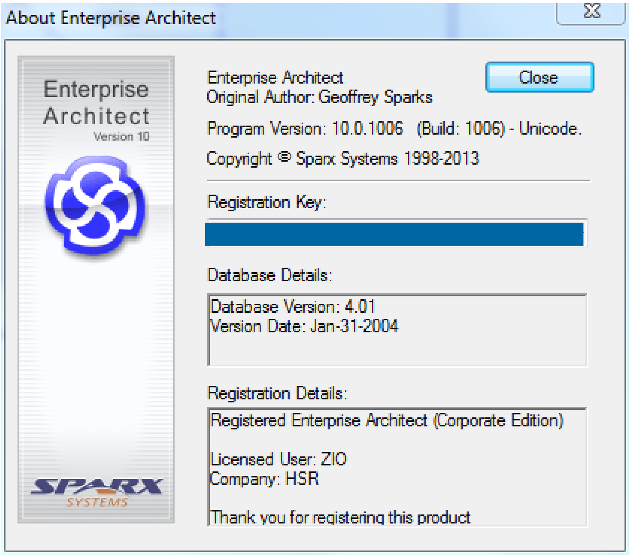
Contributions (4/4): Decision Backlog (Session State Example)

Problem Occurrence	**Status**	**Viewpoint**	**Owner Role**	**Complexity**	**…**
Session State Management Occurrence 1: Web Shop (Buyer Channel)	Decided	Functional	Web architect	High	…
Session State Management Occurrence 2: Call Center Channel	Decided	Functional	Web architect	High	…
Session Database Provider Occurrence 1: Web Shop (Buyer Channel)	Open	Information	Data Architect	Medium	…
Session Database Provider Occurrence 2: Call Center Chanel	Open	Information	Data Architect	Medium	…
…	…	…	…	…	…

- **No need to decide all open problems in next iteration/sprint**
- **Prioritization, search, filter according to metadata and project context**
- **Future work: add technical debt index, support architectural refactoring**
  - e.g. should-use vs. have-used (with assessment of principal and interest?)

Implementation : ADMentor Add-In to Enterprise Architect (EA)

- **EA profile for extended AD/AKM metamodel and supporting diagrams**
- **CRUD on metamodel instances (model elements), renaming, moving**
- **Package explorer, project explorer, matrices**
- **Rich text notes (with Web links)**
- **Model search**
- **Model patterns**
- **Model analytics**
- **Report template engine**
- **Custom link (stereo-)types**
- **…**



Enterprise Architect
Original Author: Geoffrey Sparks

Program Version: 10.0.1006 (Build: 1006) - Unicode.

Copyright © Sparx Systems 1998-2013

Registration Key:

[redacted]

Database Details:

Database Version: 4.01
Version Date: Jan-31-2004

Registration Details:

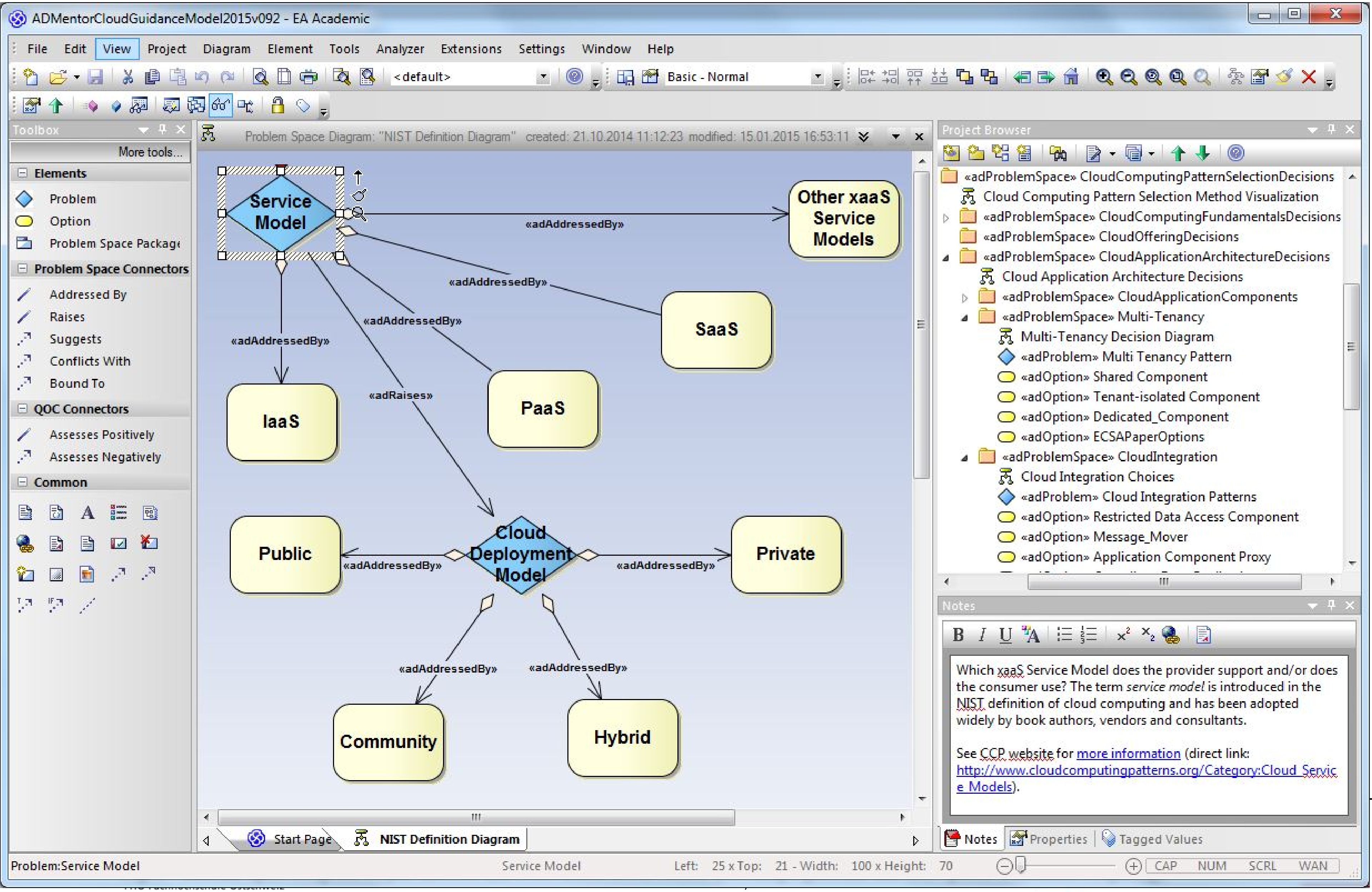
Registered Enterprise Architect (Corporate Edition)

Licensed User: ZIO
Company: HSR

Thank you for registering this product

ADMentor Tool Demo @ 6pm in Lobby area

User Interface – Seamless Integration into EA Modeling Platform

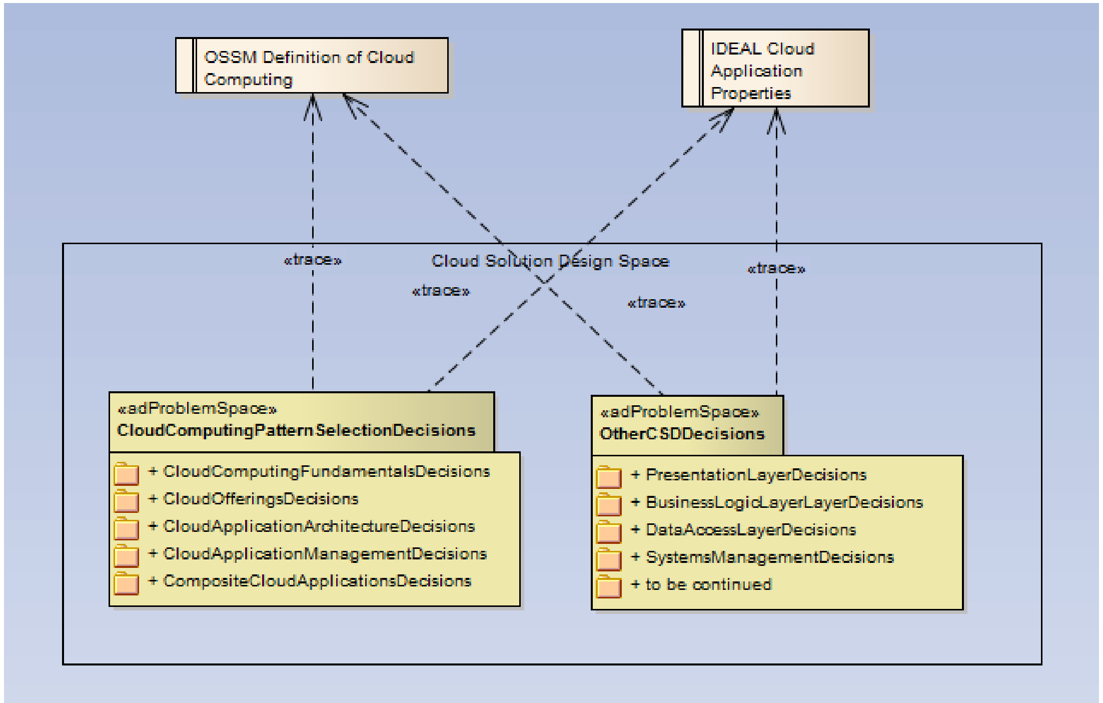


«adProblemSpace» CloudComputingPatternSelectionDecisions
Cloud Computing Pattern Selection Method Visualization
«adProblemSpace» CloudComputingFundamentalsDecisions
«adProblemSpace» CloudOfferingDecisions
«adProblemSpace» CloudApplicationArchitectureDecisions
Cloud Application Architecture Decisions
«adProblemSpace» CloudApplicationComponents
«adProblemSpace» Multi-Tenancy
Multi-Tenancy Decision Diagram
«adProblem» Multi Tenancy Pattern
«adOption» Shared Component
«adOption» Tenant-isolated Component
«adOption» Dedicated_Component
«adOption» ECSAPaperOptions
«adProblemSpace» CloudIntegration
Cloud Integration Choices
«adProblem» Cloud Integration Patterns
«adOption» Restricted Data Access Component
«adOption» Message_Mover
«adOption» Application Component Proxy

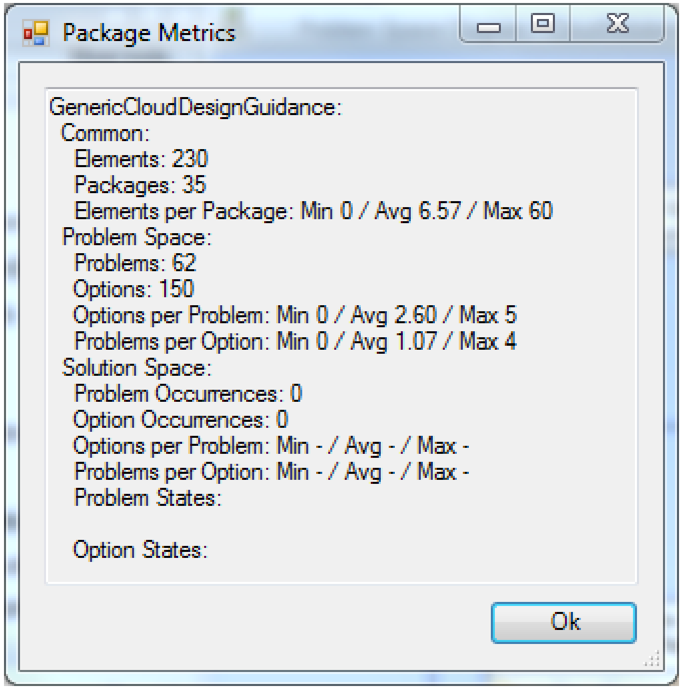
Which xaaS Service Model does the provider support and/or does the consumer use? The term *service model* is introduced in the NIST definition of cloud computing and has been adopted widely by book authors, vendors and consultants.

See CCP website for more information (direct link: http://www.cloudcomputingpatterns.org/Category:Cloud_Service_Models).

Validation 1: Cloud Pattern Language as Problem Space


OSSM Definition of Cloud Computing
IDEAL Cloud Application Properties
«trace»
Cloud Solution Design Space
«trace»
«trace»
«trace»
«adProblemSpace»
CloudComputingPatternSelectionDecisions
+ CloudComputingFundamentalsDecisions
+ CloudOfferingsDecisions
+ CloudApplicationArchitectureDecisions
+ CloudApplicationManagementDecisions
+ CompositeCloudApplicationsDecisions
«adProblemSpace»
OtherCSDDecisions
+ PresentationLayerDecisions
+ BusinessLogicLayerLayerDecisions
+ DataAccessLayerDecisions
+ SystemsManagementDecisions
+ to be continued

- **Model and tool applied to ABB architecture(s)**
  - Positive feedback regarding effort and effect (usefulness)

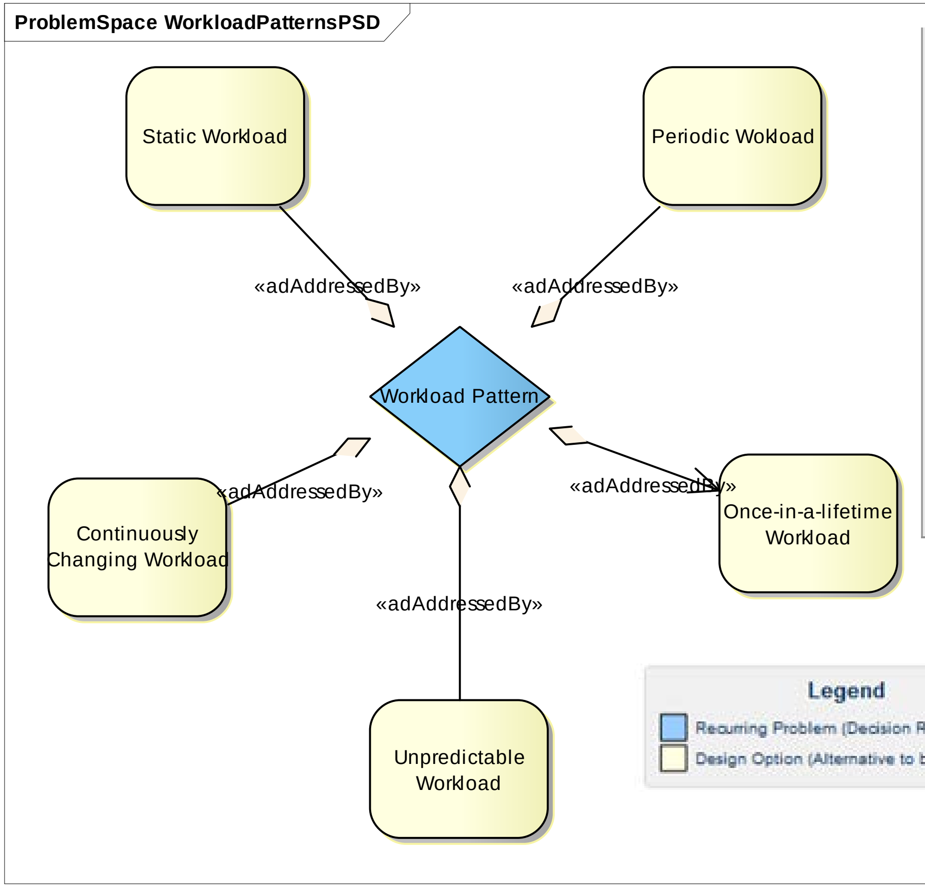


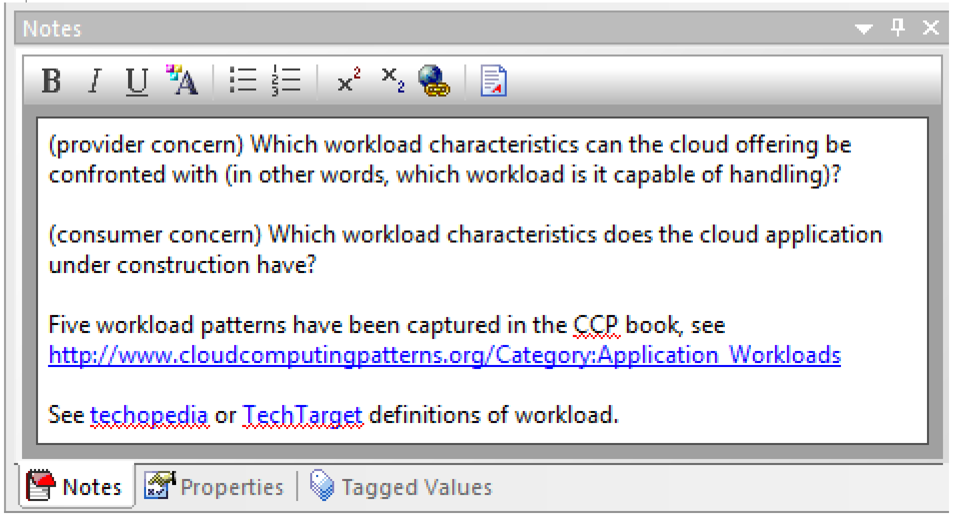
```
GenericCloudDesignGuidance:
 Common:
  Elements: 230
  Packages: 35
  Elements per Package: Min 0 / Avg 6.57 / Max 60
 Problem Space:
  Problems: 62
  Options: 150
  Options per Problem: Min 0 / Avg 2.60 / Max 5
  Problems per Option: Min 0 / Avg 1.07 / Max 4
 Solution Space:
  Problem Occurrences: 0
  Option Occurrences: 0
  Options per Problem: Min - / Avg - / Max -
  Problems per Option: Min - / Avg - / Max -
  Problem States:

  Option States:
```

- **CCP book fully modelled in ADMentor**
  - Rich text snippets and Web links over full self-contained meta model instance (unlike in previous work)

Cloud Guidance Model – Example: Workload Pattern Selection


ProblemSpace WorkloadPatternsPSD
Static Workload
Periodic Wokload
«adAddressedBy»
«adAddressedBy»
Workload Pattern
Continuously Changing Workload
«adAddressedBy»
«adAddressedBy»
Once-in-a-lifetime Workload
«adAddressedBy»
Unpredictable Workload
Legend
🟦 Recurring Problem (Decision Required)
□ Design Option (Alternative to be Considered)



(provider concern) Which workload characteristics can the cloud offering be confronted with (in other words, which workload is it capable of handling)?

(consumer concern) Which workload characteristics does the cloud application under construction have?

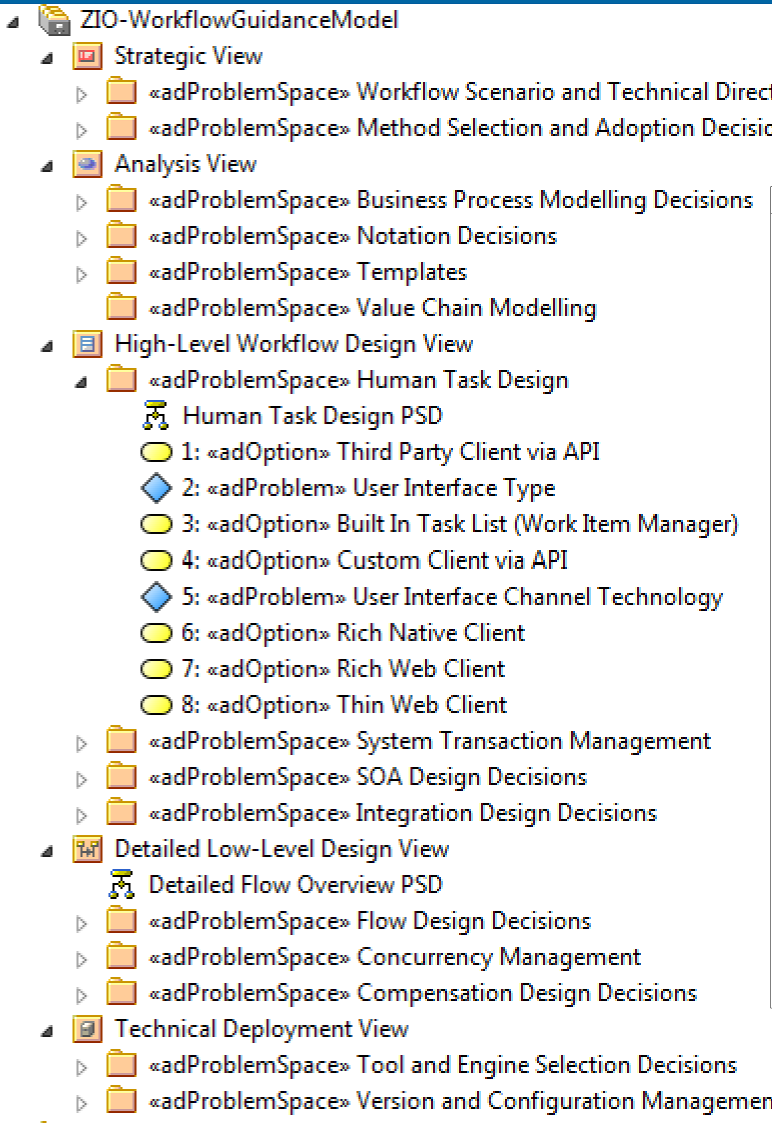
Five workload patterns have been captured in the CCP book, see http://www.cloudcomputingpatterns.org/Category:Application_Workloads

See techopedia or TechTarget definitions of workload.

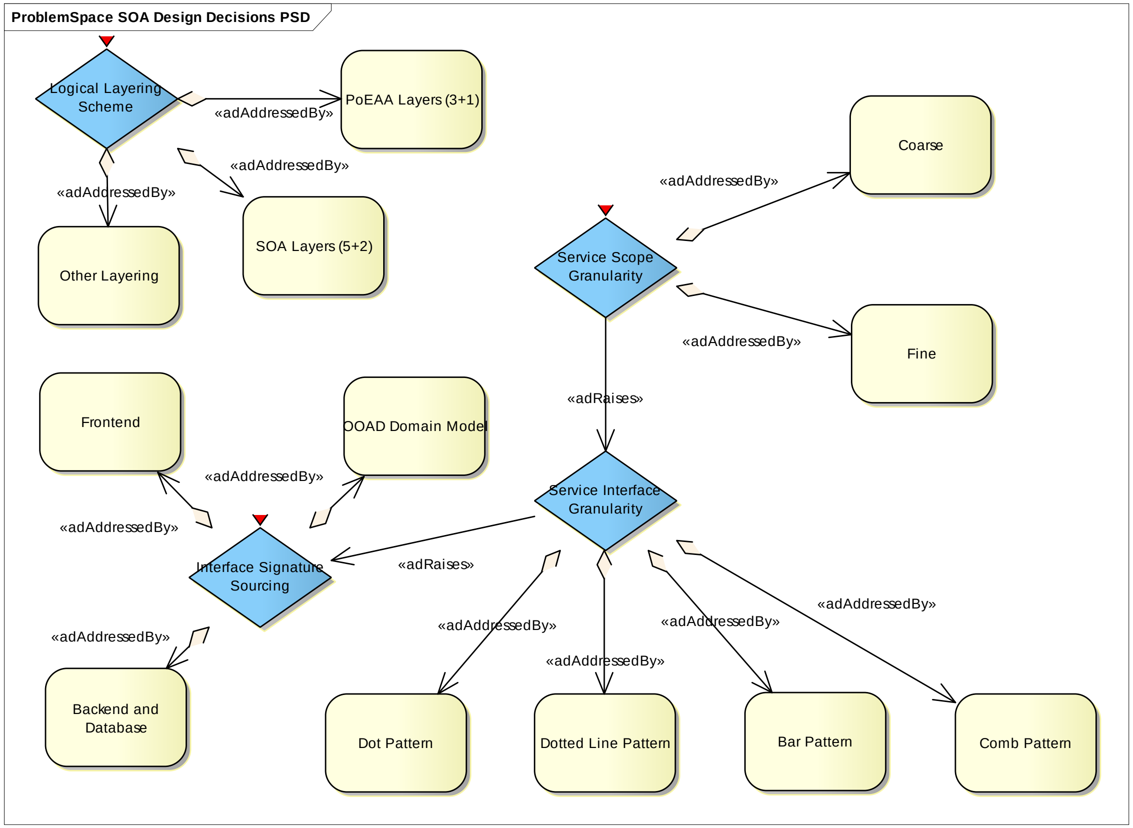
- **Problem descriptions:**
  - Motivating question
  - Link to pattern category
- **Option descriptions:**
  - Link to pattern
  - List of known uses (partial)

- **Light text descriptions by intent**
  - Rich(er) content is available online

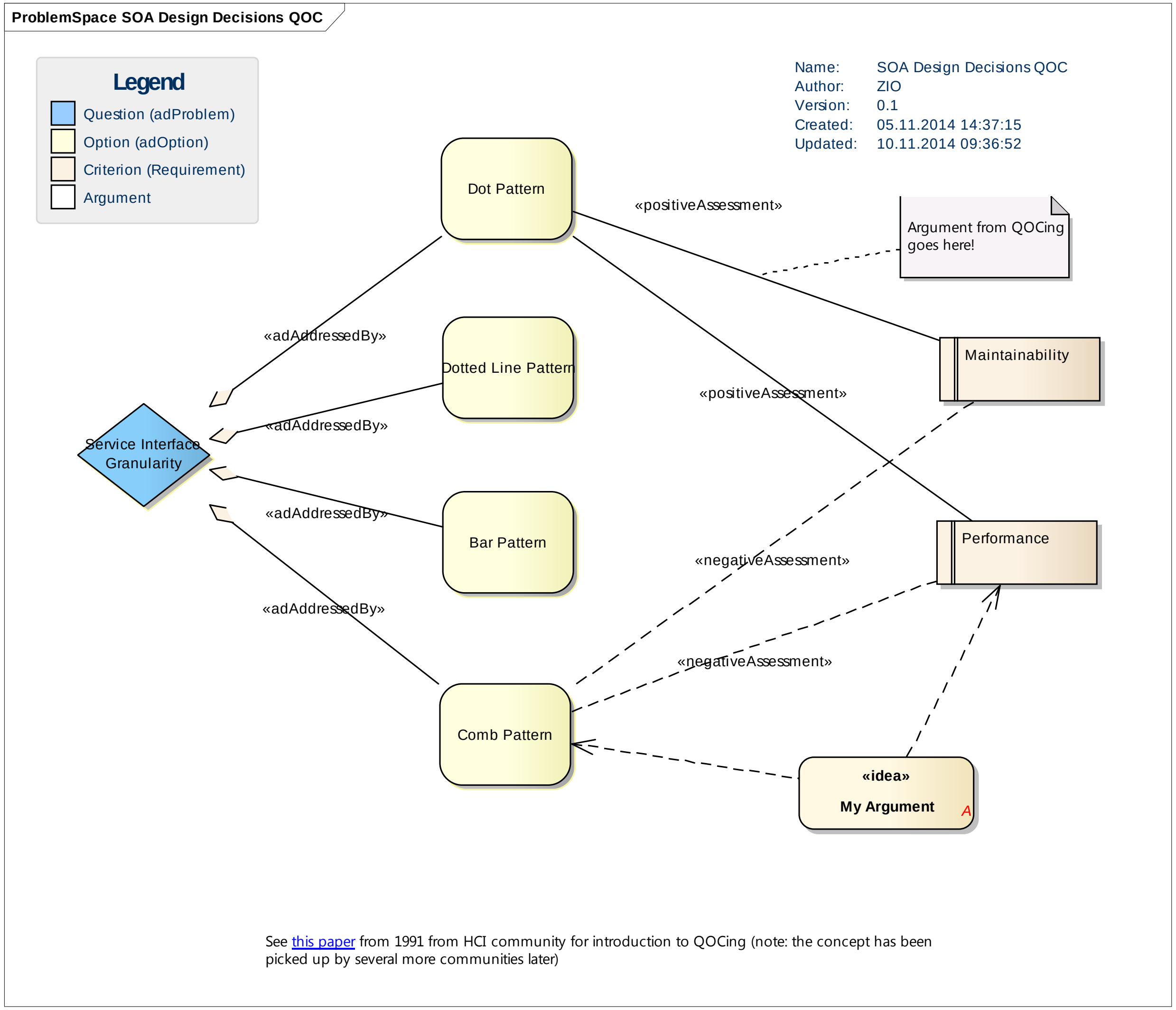
Validation 2: SOA and Workflow Problem Space Diagram



- ZIO-WorkflowGuidanceModel
  - Strategic View
    - «adProblemSpace» Workflow Scenario and Technical Directions
    - «adProblemSpace» Method Selection and Adoption Decisions
  - Analysis View
    - «adProblemSpace» Business Process Modelling Decisions
    - «adProblemSpace» Notation Decisions
    - «adProblemSpace» Templates
    - «adProblemSpace» Value Chain Modelling
  - High-Level Workflow Design View
    - «adProblemSpace» Human Task Design
      - Human Task Design PSD
      - 1: «adOption» Third Party Client via API
      - 2: «adProblem» User Interface Type
      - 3: «adOption» Built In Task List (Work Item Manager)
      - 4: «adOption» Custom Client via API
      - 5: «adProblem» User Interface Channel Technology
      - 6: «adOption» Rich Native Client
      - 7: «adOption» Rich Web Client
      - 8: «adOption» Thin Web Client
    - «adProblemSpace» System Transaction Management
    - «adProblemSpace» SOA Design Decisions
    - «adProblemSpace» Integration Design Decisions
  - Detailed Low-Level Design View
    - Detailed Flow Overview PSD
    - «adProblemSpace» Flow Design Decisions
    - «adProblemSpace» Concurrency Management
    - «adProblemSpace» Compensation Design Decisions
  - Technical Deployment View
    - «adProblemSpace» Tool and Engine Selection Decisions
    - «adProblemSpace» Version and Configuration Management



QOC Support – Demonstrates Feasibilty of Custom Extensions



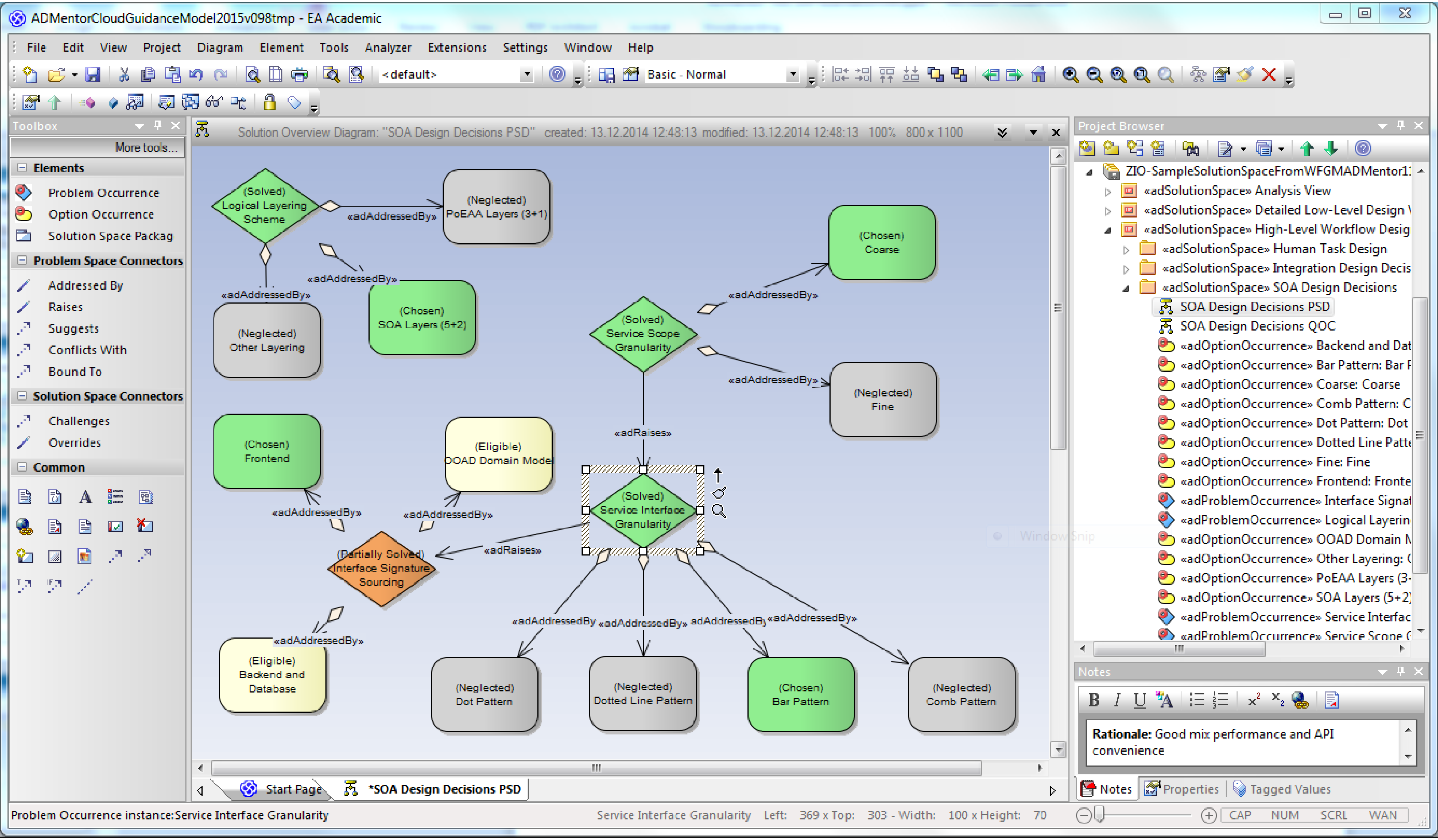
Name: SOA Design Decisions QOC
Author: ZIO
Version: 0.1
Created: 05.11.2014 14:37:15
Updated: 10.11.2014 09:36:52

Argument from QOCing goes here!

See this paper from 1991 from HCI community for introduction to QOCing (note: the concept has been picked up by several more communities later)

- **Design space visualization**
  - Originally from HCI community
  - Some popularity in AKM
- **Elements:**
  - Questions (Q)
  - Options (O)
  - Criteria (C)
- **Plus assessment relations**

Solution Space Diagram and Occurrence State Management



ADMentorCloudGuidanceModel2015v098tmp - EA Academic
File
Edit
View
Project
Diagram
Element
Tools
Analyzer
Extensions
Settings
Window
Help
<default>
Basic - Normal
Toolbox
More tools...
Elements
Problem Occurrence
Option Occurrence
Solution Space Packag
Problem Space Connectors
Addressed By
Raises
Suggests
Conflicts With
Bound To
Solution Space Connectors
Challenges
Overrides
Common
(Solved) Logical Layering Scheme
«adAddressedBy»
(Neglected) PoEAA Layers (3+1)
«adAddressedBy»
«adAddressedBy»
(Neglected) Other Layering
(Chosen) SOA Layers (5+2)
(Chosen) Coarse
«adAddressedBy»
(Solved) Service Scope Granularity
«adAddressedBy»
(Neglected) Fine
«adRaises»
(Chosen) Frontend
(Eligible) OOAD Domain Model
«adAddressedBy»
«adAddressedBy»
(Solved) Service Interface Granularity
«adRaises»
(Partially Solved) Interface Signature Sourcing
«adAddressedBy»
(Eligible) Backend and Database
«adAddressedBy»
«adAddressedBy»
«adAddressedBy»
«adAddressedBy»
(Neglected) Dot Pattern
(Neglected) Dotted Line Pattern
(Chosen) Bar Pattern
(Neglected) Comb Pattern
Start Page
*SOA Design Decisions PSD
Project Browser
ZIO-SampleSolutionSpaceFromWFGMADMentor1
«adSolutionSpace» Analysis View
«adSolutionSpace» Detailed Low-Level Design
«adSolutionSpace» High-Level Workflow Desig
«adSolutionSpace» Human Task Design
«adSolutionSpace» Integration Design Decis
«adSolutionSpace» SOA Design Decisions
SOA Design Decisions PSD
SOA Design Decisions QOC
«adOptionOccurrence» Backend and Dat
«adOptionOccurrence» Bar Pattern: Bar F
«adOptionOccurrence» Coarse: Coarse
«adOptionOccurrence» Comb Pattern: C
«adOptionOccurrence» Dot Pattern: Dot
«adOptionOccurrence» Dotted Line Patte
«adOptionOccurrence» Fine: Fine
«adOptionOccurrence» Frontend: Fronte
«adProblemOccurrence» Interface Signat
«adProblemOccurrence» Logical Layerin
«adOptionOccurrence» OOAD Domain M
«adOptionOccurrence» Other Layering: (
«adOptionOccurrence» PoEAA Layers (3-
«adOptionOccurrence» SOA Layers (5+2)
«adProblemOccurrence» Service Interfac
«adProblemOccurrence» Service Scope (
Notes
Notes
Properties
Tagged Values
CAP
NUM
SCRL
WAN

Solution Overview Diagram: "SOA Design Decisions PSD" created: 13.12.2014 12:48:13 modified: 13.12.2014 12:48:13 100% 800 x 1100

Rationale: Good mix performance and API convenience

Problem Occurrence instance:Service Interface Granularity

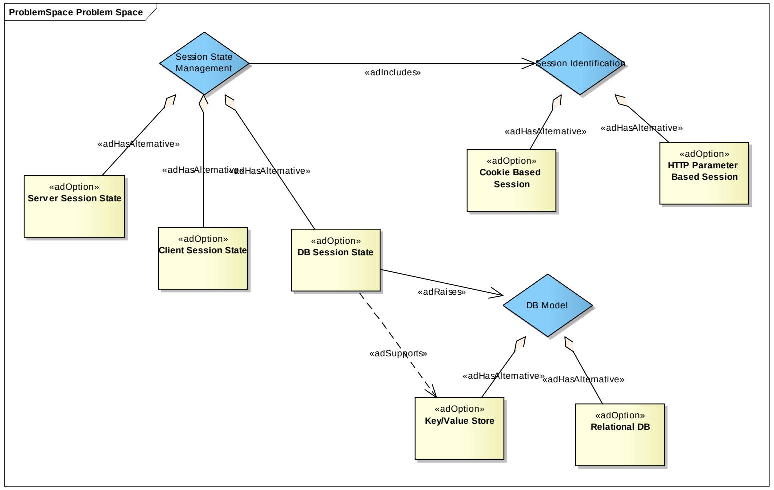
Service Interface Granularity Left: 369 x Top: 303 - Width: 100 x Height: 70

Summary (1/2): Context and Contributions

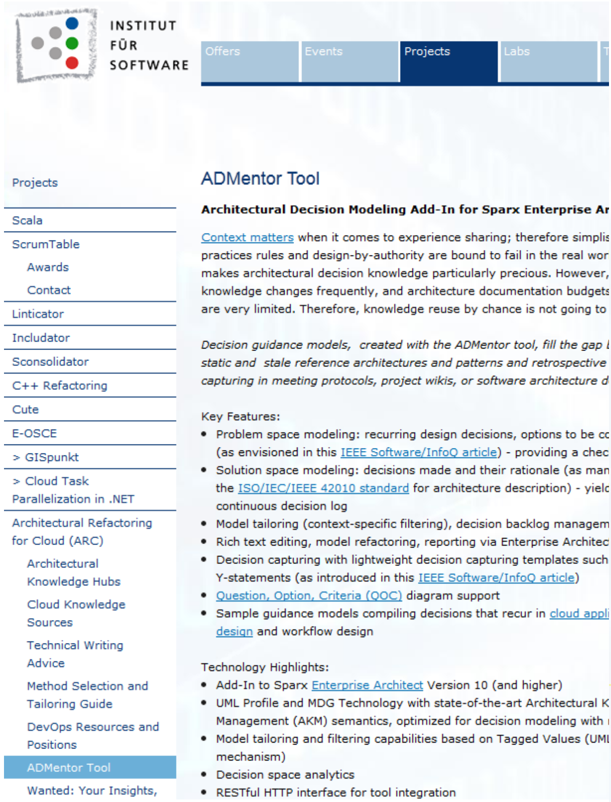
- **Architectural decision making is a key responsibility of IT architects which is often underestimated and underrepresented in existing methods and tools.**
  - AD capturing templates vary – supporting tools must accommodate that
  - Metadata can help with AD tailoring and integration
- **In cloud application design and other domains, many architectural decisions recur. This makes it possible to reduce the documentation effort and to share architectural decision knowledge in a consumable way:**
  - Decisions required vs. decisions made
  - Benefits: design acceleration and quality assurance
- **Tool support for decision modeling with reuse is emerging**
  - Decision Architect, ADMentor; Advise, Software Architecture Warehouse
- **Collaboration opportunities abound…**
  - … do you have input to (or a need for) a cloud/SOA/workflow design space?
  - … do you have a need/use for an AKM data set (e.g. cloud/workflow)?

Summary (2/2): ADMentor Implementation

- **Joint work, HSR FHO and ABB Corporate Research**
  - Tool website: http://www.ifs.hsr.ch/ADMentor-Tool.13201.0.html?&L=4
- **Add In for Sparx Enterprise Architect that supports AD reuse and sharing (on top of AD documentation features of other tools)**
  - Problem and Option vs. Problem Occurrence and Option Occurrence
  - Leverages standard product features as much as possible (e.g. rich text editor, reporting, model refactoring, links)



More Information: Project Websites @ IFS HSR



ADMentor Tool

Architectural Decision Modeling Add-In for Sparx Enterprise Ar

Context matters when it comes to experience sharing; therefore simplis practices rules and design-by-authority are bound to fail in the real wor makes architectural decision knowledge particularly precious. However, knowledge changes frequently, and architecture documentation budgets are very limited. Therefore, knowledge reuse by chance is not going to

Decision guidance models, created with the ADMentor tool, fill the gap l static and stale reference architectures and patterns and retrospective capturing in meeting protocols, project wikis, or software architecture d

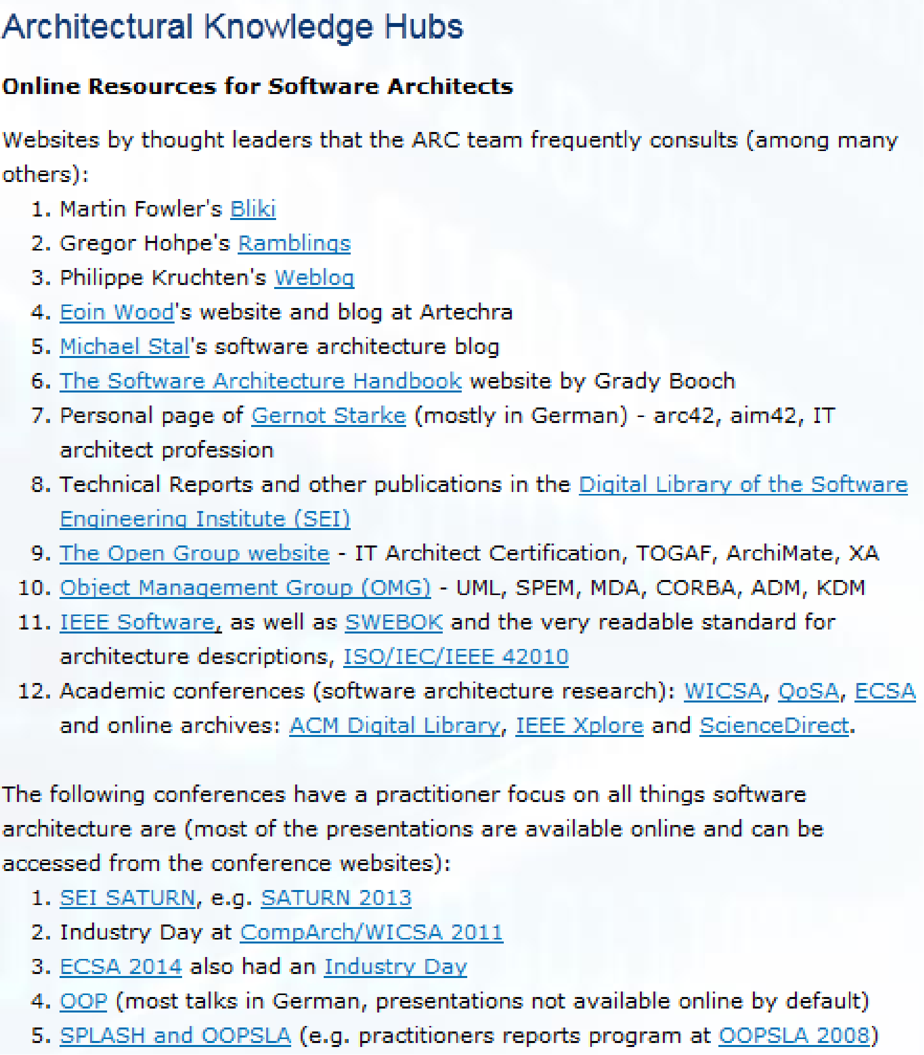
Key Features:

- Problem space modeling: recurring design decisions, options to be cc (as envisioned in this IEEE Software/InfoQ article) - providing a chec
- Solution space modeling: decisions made and their rationale (as man the ISO/IEC/IEEE 42010 standard for architecture description) - yielc continuous decision log
- Model tailoring (context-specific filtering), decision backlog managem
- Rich text editing, model refactoring, reporting via Enterprise Architec
- Decision capturing with lightweight decision capturing templates such Y-statements (as introduced in this IEEE Software/InfoQ article)
- Question, Option, Criteria (QOC) diagram support
- Sample guidance models compiling decisions that recur in cloud appli design and workflow design

Technology Highlights:

- Add-In to Sparx Enterprise Architect Version 10 (and higher)
- UML Profile and MDG Technology with state-of-the-art Architectural K Management (AKM) semantics, optimized for decision modeling with
- Model tailoring and filtering capabilities based on Tagged Values (UMl mechanism)
- Decision space analytics
- RESTful HTTP interface for tool integration

(screen captions clickable)



Architectural Knowledge Hubs

Online Resources for Software Architects

Websites by thought leaders that the ARC team frequently consults (among many others):

1. Martin Fowler's Bliki
2. Gregor Hohpe's Ramblings
3. Philippe Kruchten's Weblog
4. Eoin Wood's website and blog at Artechra
5. Michael Stal's software architecture blog
6. The Software Architecture Handbook website by Grady Booch
7. Personal page of Gernot Starke (mostly in German) - arc42, aim42, IT architect profession
8. Technical Reports and other publications in the Digital Library of the Software Engineering Institute (SEI)
9. The Open Group website - IT Architect Certification, TOGAF, ArchiMate, XA
10. Object Management Group (OMG) - UML, SPEM, MDA, CORBA, ADM, KDM
11. IEEE Software, as well as SWEBOK and the very readable standard for architecture descriptions, ISO/IEC/IEEE 42010
12. Academic conferences (software architecture research): WICSA, QoSA, ECSA and online archives: ACM Digital Library, IEEE Xplore and ScienceDirect.

The following conferences have a practitioner focus on all things software architecture are (most of the presentations are available online and can be accessed from the conference websites):

1. SEI SATURN, e.g. SATURN 2013
2. Industry Day at CompArch/WICSA 2011
3. ECSA 2014 also had an Industry Day
4. OOP (most talks in German, presentations not available online by default)
5. SPLASH and OOPSLA (e.g. practitioners reports program at OOPSLA 2008)

WICSA/CompArch 2015

ARCHITECTURAL DECISION GUIDANCE ACROSS PROJECTS

BACKGROUND MATERIAL

IFS INSTITUTE FOR SOFTWARE

Prof. Dr. Olaf Zimmermann

Institute for Software, HSR FHO

Montreal, May 6, 2015

HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

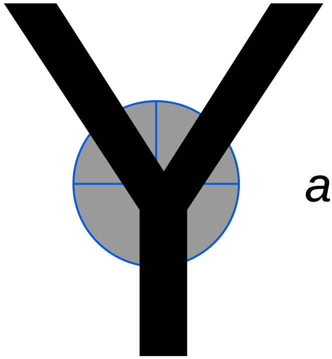
FHO Fachhochschule Ostschweiz

Y-Template

- **Presented at SATURN 2012 (Haiku-style rationale with some traces):**

In the context of <use case uc and/or component co>,

… facing <non-functional concern c>,



We chose <options o1>,

and neglected <options o2 to on>,

… to achieve <quality q>,

… accepting downside <consequence c>.

Good and Bad Justifications, Part 1

Decision driver type	Valid justification	Counter example
Wants and needs of external stakeholders	Alternative A best meets user expectations and functional requirements as documented in user stories, use cases, and business process model.	End users want it, but no evidence for a pressing business need. Technical project team never challenged the need for this feature. Technical design is prescribed in the requirements documents.
Architecturally significant requirements	Nonfunctional requirement XYZ has higher weight than any other requirement and must be addressed; only alternative A meets it.	Do not have any strong requirements that would favor one of the design options, but alternative B is the market trend. Using it will reflect well on the team.
Conflicting decision drivers and alternatives	Performed a trade-off analysis, and alternative A scored best. Prototype showed that it's good enough to solve the given design problem and has acceptable negative consequences.	Only had time to review two design options and did not conduct any hands-on experiments. Alternative B does not seem to perform well, according to information online. Let's try alternative A.

Source: Zimmermann O., Schuster N., Eeles P., Modeling and Sharing Architectural Decisions, Part 1: Concepts. IBM developerWorks, 2008

Good and Bad Justifications, Part 2

Decision driver type	Valid justification	Counter example
Reuse of an earlier design	Facing the same or very similar NFRs as successfully completed project XYZ. Alternative A worked well there. A reusable asset of high quality is available to the team.	We've always done it like that. Everybody seems to go this way these days; there's a lot of momentum for this technology.
Prefer do-it-yourself over commercial off-the-shelf (build over buy)	Two cornerstones of our IT strategy are to differentiate ourselves in selected application areas, and remain master of our destiny by avoiding vendor lock-in. None of the evaluated software both meets our functional requirements and fits into our application landscape. We analyzed customization and maintenance efforts and concluded that related cost will be in the same range as custom development.	Price of software package seems high, though we did not investigate total cost of ownership (TCO) in detail. Prefer to build our own middleware so we can use our existing application development resources.
Anticipation of future needs	Change case XYZ describes a feature we don't need in the first release but is in plan for next release. Predict that concurrent requests will be x per second shortly after global rollout of the solution, planned for Q1/2009.	Have to be ready for any future change in technology standards and in data models. All quality attributes matter, and quality attribute XYZ is always the most important for any software-intensive system.

Source: Zimmermann O., Schuster N., Eeles P., Modeling and Sharing Architectural Decisions, Part 1: Concepts. IBM developerWorks, 2008

Recurring Issues (1/2)

Artifact	Decision Topic	Recurring Issues (Decisions Required)
Enterprise architecture documentation [SZ92, ZTP03]	IT strategy	Buy vs. build strategy, open source policy
	Governance	Methods (processes, notations), tools, reference architectures, coding guidelines, naming standards, asset ownership
System context [CCS07]	Project scope	External interfaces, incoming and outgoing calls (protocols, formats, identifiers), service level agreements, billing
Other viewpoints [Kru95]	Development process	Configuration management, test cases, build/test/production environment staging
	Physical tiers	Locations, security zones, nodes, load balancing, failover, storage placement
	Data management	Data model reach (enterprise-wide?), synchronization/replication, backup strategy
Architecture overview diagram [Fow03, CCS07]	Logical layers	Coupling and cohesion principles, functional decomposition (partitioning)
	Physical tiers	Locations, security zones, nodes, load balancing, failover, storage placement
	Data management	Data model reach (enterprise-wide?), synchronization/replication, backup strategy
Architecture overview diagram [Eva03, Fow03]	Presentation layer	Rich vs. thin client, multi-channel design, client conversations, session management
	Domain layer (process control flow)	How to ensure process and resource integrity, business and system transactionality
	Domain layer (remote interfaces)	Remote contract design (interfaces, protocols, formats, timeout management)
	Domain layer (component-based development)	Interface contract language, parameter validation, Application Programming Interface (API) design, domain model
	Resource (data) access layer	Connection pooling, concurrency (auto commit?), information integration, caching
	Integration	Hub-and-spoke vs. direct, synchrony, message queuing, data formats, registration

Source: O. Zimmermann, Architectural Decision Identification in Architectural Patterns. WICSA/ECSA Companion Volume 2012, Pages 96-103.

Recurring Issues (2/2)

Artifact	Decision Topic	Recurring Issues (Decisions Required)
Logical component [ZTP03]	Security	Authentication, authorization, confidentiality, integrity, non-repudiation, tenancy
	Systems management	Fault, configuration, accounting, performance, and security management
Logical component [ZZG+08]	Lifecycle management	Lookup, creation, static vs. dynamic activation, instance pooling, housekeeping
	Logging	Log source and sink, protocol, format, level of detail (verbosity levels)
	Error handling	Error logging, reporting, propagation, display, analysis, recovery
Components and connectors [ZTP03, CCS07]	Implementation technology	Technology standard version and profile to use, deployment descriptor settings (QoS)
	Deployment	Collocation, standalone vs. clustered
Physical node [YRS+99]	Capacity planning	Hardware and software sizing, topologies
	Systems management	Monitoring concept, backup procedures, update management, disaster recovery

Source: O. Zimmermann, Architectural Decision Identification in Architectural Patterns. WICSA/ECSA Companion Volume 2012, Pages 96-103.