Using intentional actor modeling to support the evolution of enterprise software architectures in organizations

iStar'10 workshop @ CAiSE 2010
Hammamet, Tunisia
June 2010

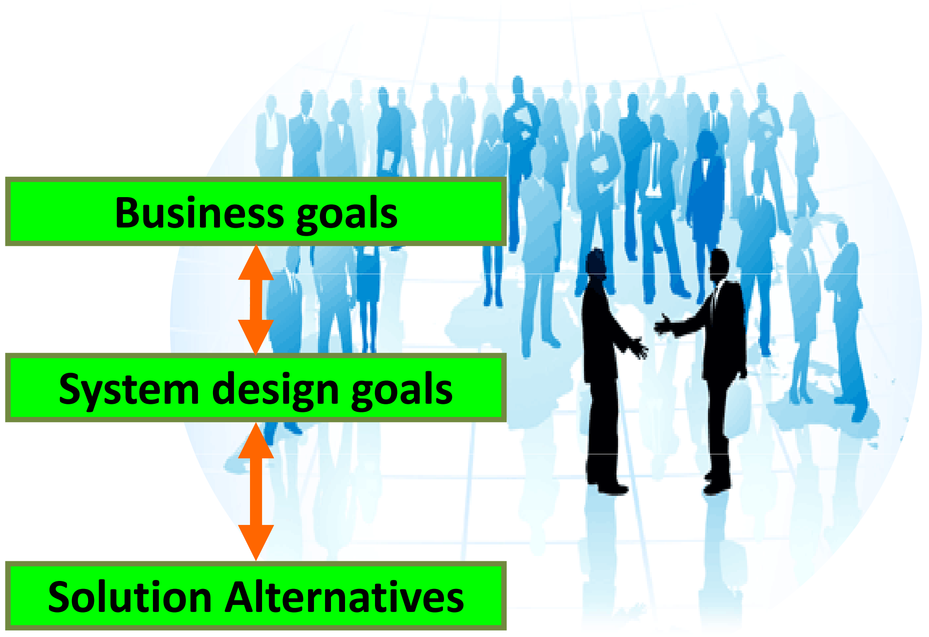
Daniel Gross & Eric Yu

UNIVERSITY OF TORONTO
FACULTY OF INFORMATION
The iSchool at Toronto

Highlights

- An exploration of applying i* in enterprise software architecture reasoning
  - A pilot case study
- A small extension to i*
  - Concept of an “intentional viewpoint”

From business goals to software architecture

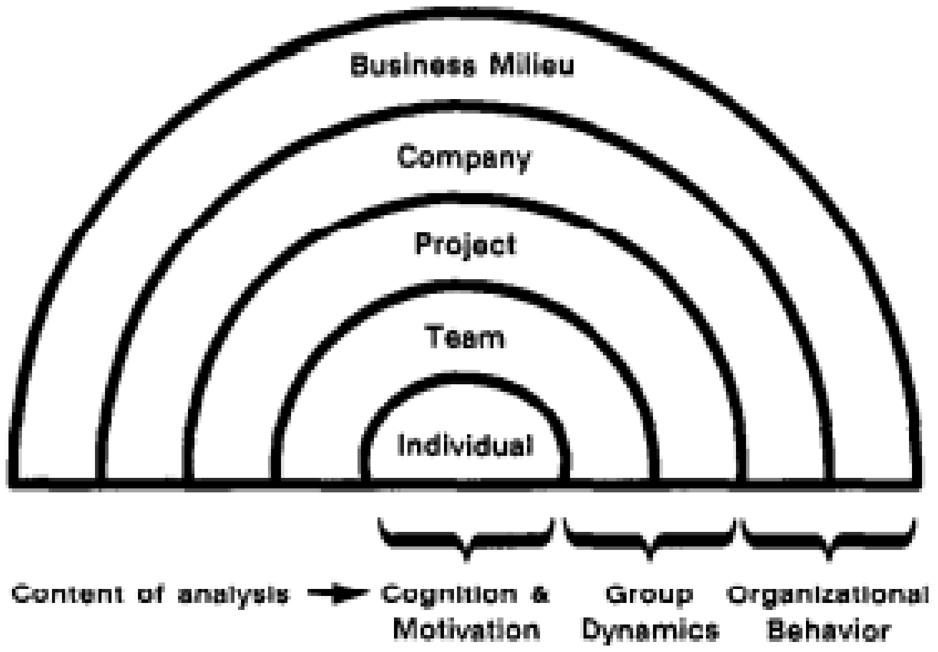


- Business goals justify design goals, which in turn guide design decision-making [3,4]
- Architectural design as an organizationally distributed decision-making process

[3] Kazman, R., & Bass, L. (2005). *Categorizing Business Goals for Software Architecture: SEI.*

[4] Tyree, J., & Akerman, A. (2005). Architecture decisions: demystifying architecture. *IEEE Software, 22(2), 19-27.*

Nature of decision-making in software development organizations e.g., [Curtis], [Herbsleb], [Grinter]



[2] Curtis, B., Krasner, H., & Iscoe, N. (1988). A Field Study of the Software Design Process for Large Systems. *Communications of the ACM, 31(11), 1268-1287.*

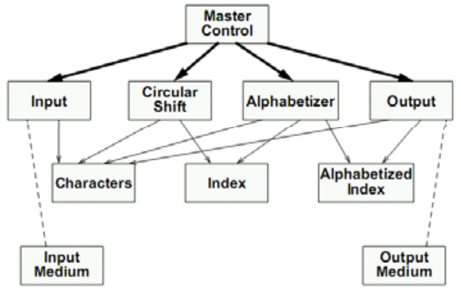
- Organizational decision behavior is inherently **distributed**, heterogeneous, and contingent on many (changing) factors
- Stakeholders and designers at different levels in the organization pursue goals, make and **delegate** decisions
- Stakeholders and designers goals are *heterogeneous*, *situated in different domains*, which often lead to **conflicting choices**
- Decision making (authority/autonomy) is distributed **"vertically" and "horizontally"**
- Decisions are made **locally**, but are interdependent, and often have systemic consequences
- "Linking" between decision-making behavior is ***social***, requires the identification and negotiation of shared interests

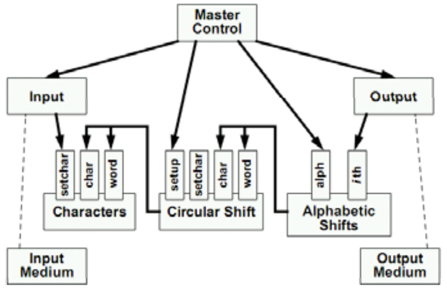
Research question

- Given that architectural decision making occurs in a larger context of organizationally distributed decision-making that involves business and system stakeholders and designers …
- … can such decision making benefit from using **social modeling** such as provided by i* framework?

- More specific questions:
  - **Is distributed reasoning and decision-making a problem in enterprise architecture evolution projects?**
  - **Does intentional agent modeling help in representing and communicating distributed decision-making amongst stakeholder and designers?**
  - **Does the inclusion of higher (management) level organizational participants and their goals into a model provide any value?**

Typical Software Architecture Reasoning





Shared Data Pros and Cons

+ Efficiency
  → shared data
  → efficient data representation
  → sequential data access
+ Intuitive structure
- Changeability
  → data format not abstracted away
  → functional elements dependent on data representation
- Support for reuse

ADT Pros and Cons

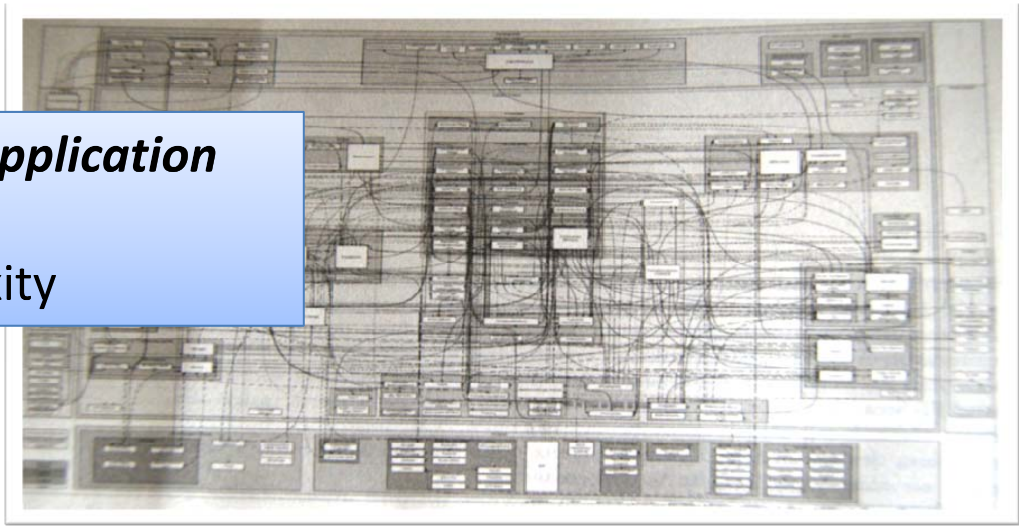
+ Intuitive structure
+ Changeability
  → data format abstracted away inside ADTs
  → modification of the processing algorithm isolated to individual modules
+ Support for reuse
  → fewer assumptions about the rest of the system
- Expansion of functionality [1]
  → sacrifice either conceptual simplicity or performance

But choosing alternatives in a business setting requires knowledge of business intents/goals

[1] Garlan, D., & Shaw, M. (1994). *An Introduction to Software Architecture (No. CMU-CS-94-166): Carnegie Mellon University.*

Consider Enterprise Application Landscapes

- Enterprise applications are **interconnected** via shared databases and/or interfaces



An ***Enterprise Application Landscape*** of high complexity

- Interesting to study because:
  - **Multiple stakeholders and designers** concurrently develop and evolve enterprise applications
  - **No single point of authority** possible -- inherently distributed and autonomous decision-making
  - Evolution towards **enterprise architecture** approach requires dealing with distribution of authority and decision making

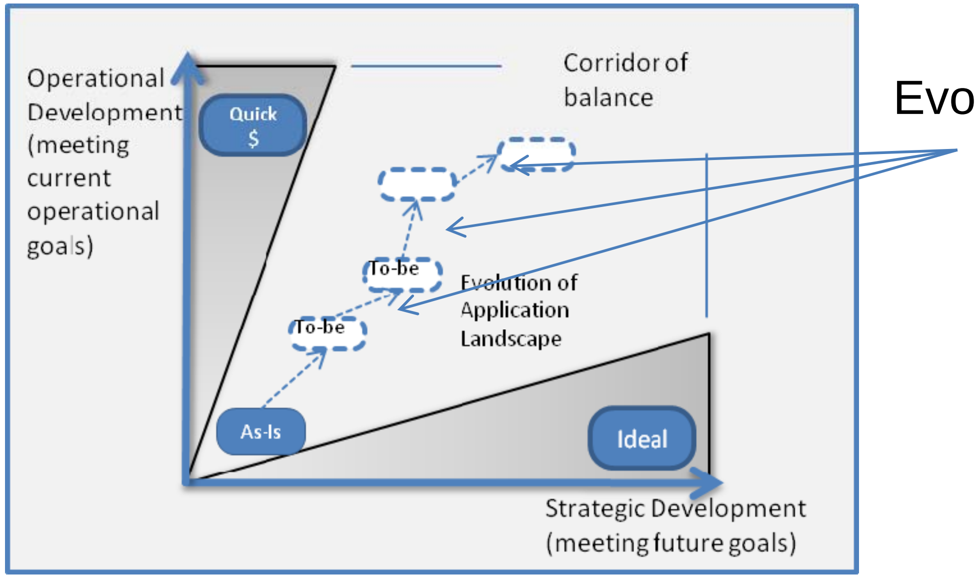
Why evolve towards an Enterprise Architecture

- Application landscapes have become increasingly complex, hard to understand and costly to maintain and evolve, e.g.,
  - Many different and incompatible design approaches
  - Unclear separation of business and/or technical concerns
  - Extensive use of Point to Point integrations across enterprise applications
  - Heterogeneous infrastructures
  - Multiple infrastructure suppliers/providers

- However, there are reasons for the current architectural structures!
  - Current enterprise applications are designed to support current business operations and do **meet (more or less) current organizational goals,** e.g., profitability, market share, …

Tradeoffs in enterprise application evolution

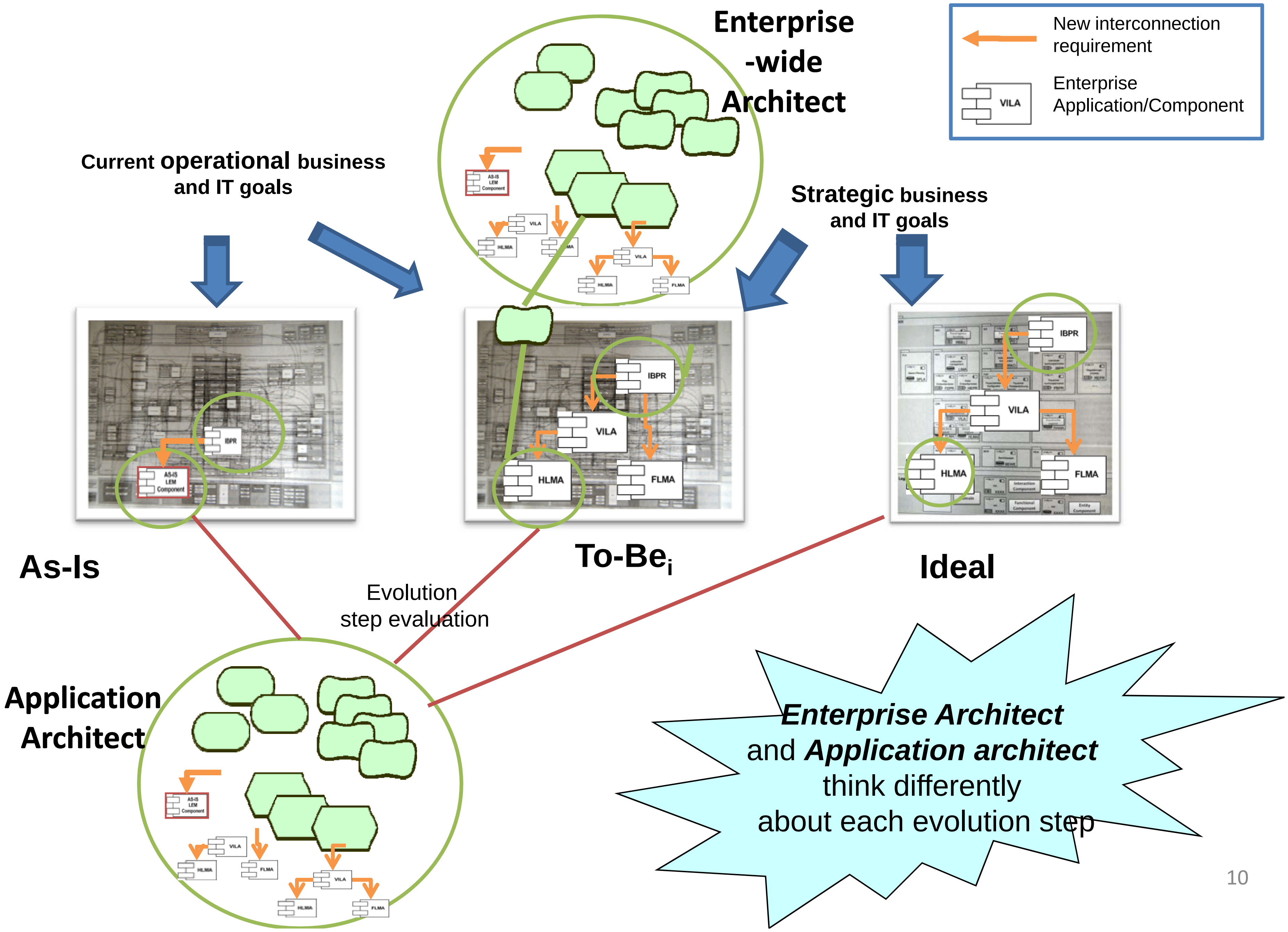
- Focusing on *operational business needs* only (here and now)
  - → leads long term to an un-maintainable application landscape
- Focusing on *strategic goals* only (enterprise as a whole, future-directed)
  - → usually too costly
- Need to find balance
  - Business requirements *and* IT requirements
  - Long-term *and* short-term


Operational Development (meeting current operational goals)
Quick $
Corridor of balance
Evolution steps
To-be
To-be
Evolution of Application Landscape
As-Is
Ideal
Strategic Development (meeting future goals)

IBM's SOMA and Cap Gemini/SD&M Quasar Enterprise methods, informally introduce goals to guide EA evolution

Engels, Hess, et. al (2008) "Quasar Enterprise: Anwendungslandschaften service-orientiert gestalten", dpunkt.verlag, Heidelberg

Analyzing an evolution step using agents and goals



Enterprise -wide Architect
New interconnection requirement
Enterprise Application/Component
VILA
Current operational business and IT goals
Strategic business and IT goals
AS-IS LEM Component
IBPR
VILA
HLMA
FLMA
As-Is
To-Be$_i$
Ideal
Evolution step evaluation
Application Architect
Enterprise Architect and *Application architect* think differently about each evolution step

Case study

- Study site:
  - The Phoenix Insurance – a major insurer in Israel
  - enterprise systems evolving towards a service-oriented enterprise architecture
- Study objective:
  - Application of agent and goal modeling to enterprise architecture evolution
  - Utility of agent and goal-modeling for practitioners in their daily work
- Study approach:
  - Interview stakeholders
  - Analyze and model reported architectural evolution discussions
  - Identify linking of architectural decision making across organizational stakeholders to higher level system and business goals
  - Present agent and goal models to stakeholders for feedback

Argumentation about Messaging Approach between Consumer and Provider Components

Enterprise Architect design argumentation viewpoint:

Use Asynchronous Messaging!

- **Resource efficiency:** Asynchronously messaging requires less SOA infrastructure resources.
- **Improved extensibility:** Asynchronous messaging supports forwarding policy data to other relevant Providers, without needing the Consumer to know a-priori about additional destinations.
- **Simplify Exception handling:** How should the Consumer behave when one or more Providers return error codes while others complete processing successfully? Dealing with such cases is not obvious and complicates the design of the Consumer component. Asynchronous messaging delegates dealing with such issues to the ESB and simplifying the Consumer component.
- **Simplify processing of multiple Provider feedback:** Asynchronous messaging also simplifies support for multiple feedback messages returned by one or more Providers, and routing feedback messages to several interested Consumers. Asynchronous messaging directly supports such decoupling while synchronous messaging requires more design work for and across Consumer components.

Consumer Component design argumentation viewpoint:

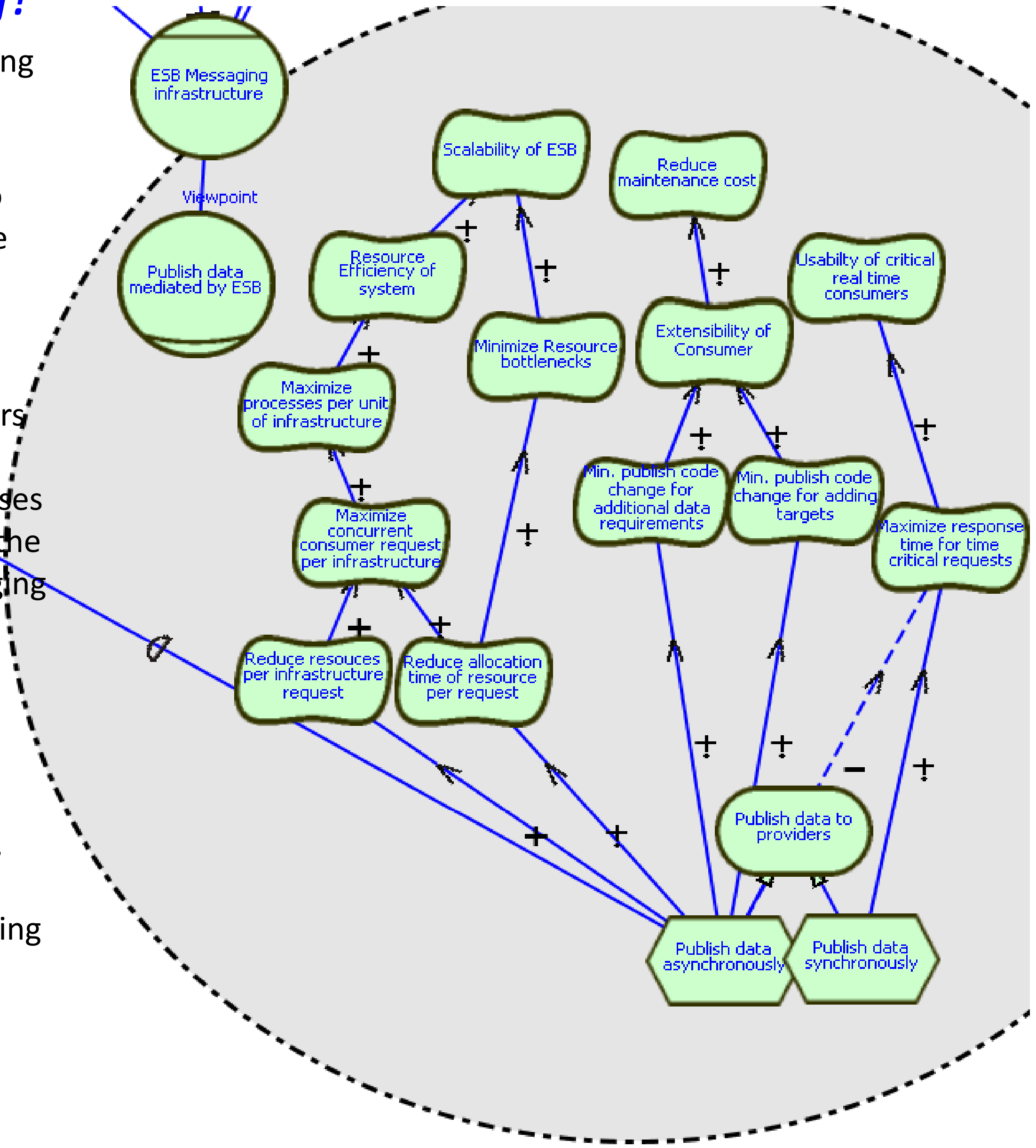
Use Synchronous Messaging!

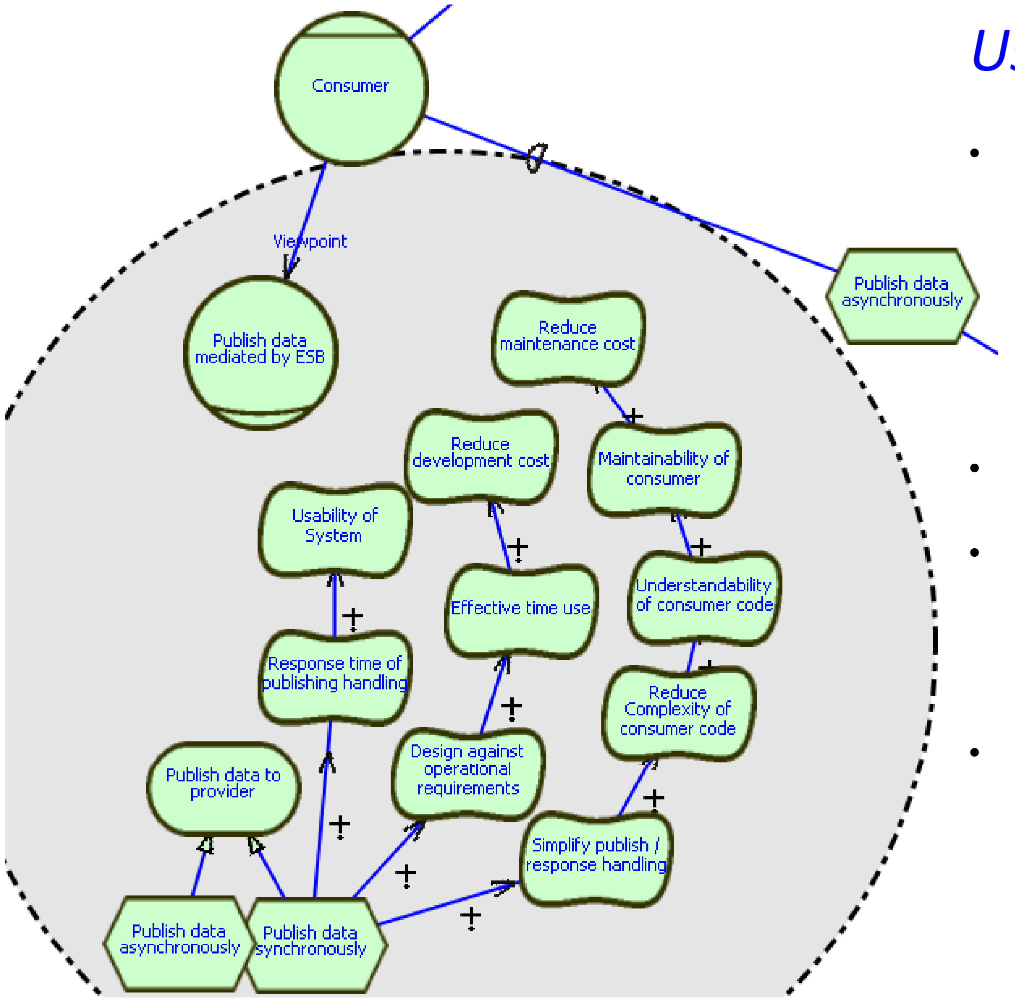
- **Simplifying Consumer component**: Synchronous messaging simplifies the Consumer component design and code. It simplifies sending the new policy request to a specific Provider, and also simplifies handling the response received from the Provider. Simplifying is important since it make code more understandable which contributes to reduce the component's maintenance cost.
- **Simple is cost efficient:** Simpler design is easier and faster to write, which reduces development cost.
- **Synchronous is fast:** Synchronous messaging returns an immediate response. The ESB works to immediately fulfill a synchronous request. This improves response time for the Consumer component and contributes to the customer's quality of service perception.
- **Improved design accountability:** Having a response returned immediately, improves the control the Consumer component designer has over the overall policy submission process. Using synchronous messaging allows the Consumer component designer to directly design for failure contingencies.

Enterprise Architect design argumentation viewpoint:

Use Asynchronous Messaging!

- **Resource efficiency:** Asynchronously messaging requires less SOA infrastructure resources.
- **Improved extensibility:** Asynchronous messaging supports forwarding policy data to other relevant Providers, without needing the Consumer to know a-priori about additional destinations.
- **Simplify Exception handling:** How should the Consumer behave when one or more Providers return error codes while others complete processing successfully? Dealing with such cases is not obvious and complicates the design of the Consumer component. Asynchronous messaging delegates dealing with such issues to the ESB and simplifying the Consumer component.
- **Simplify processing of multiple Provider feedback:** Asynchronous messaging also simplifies support for multiple feedback messages returned by one or more Providers, and routing feedback messages to several interested Consumers. Asynchronous messaging directly supports such decoupling while synchronous messaging requires more design work for and across Consumer components.

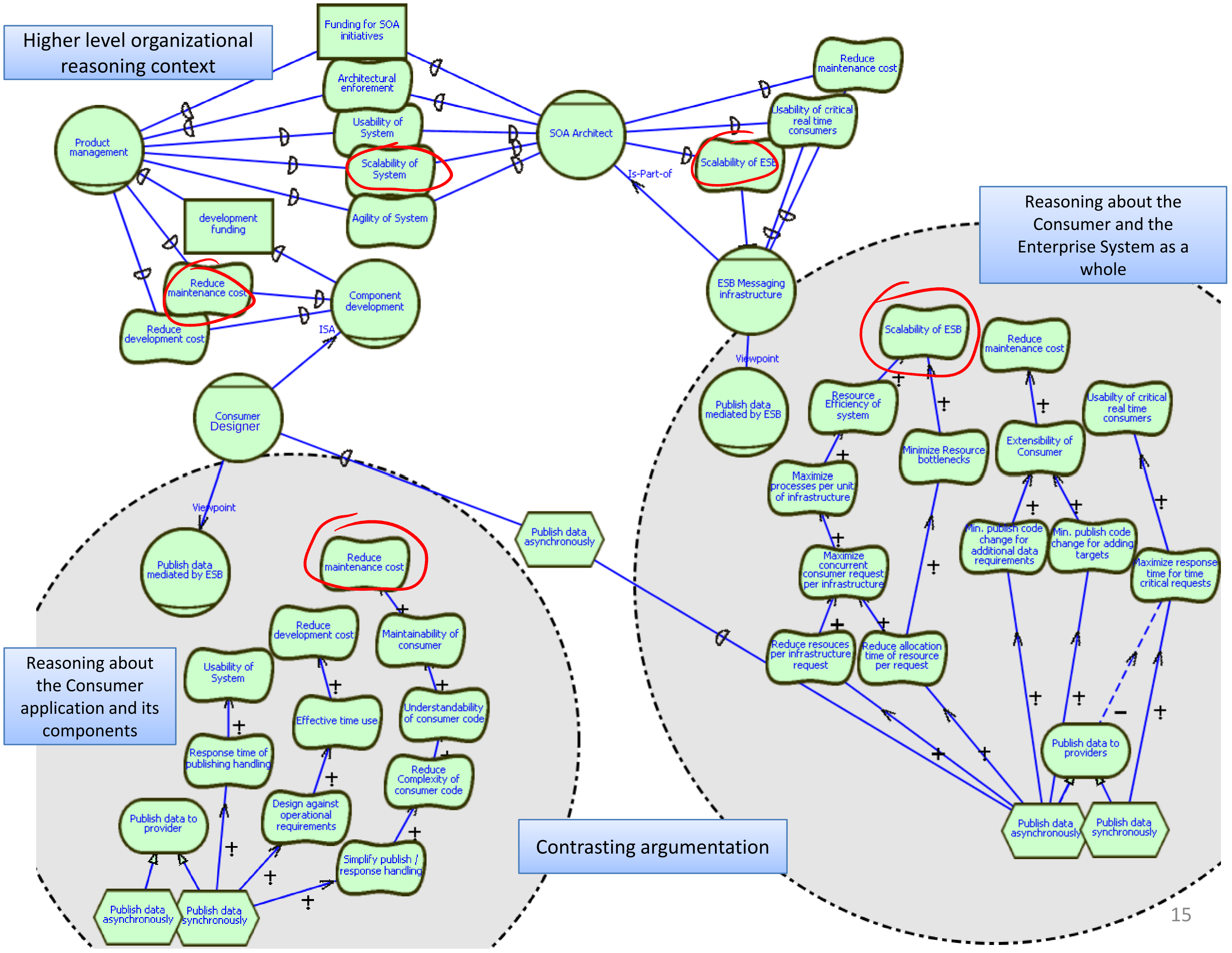

ESB Messaging infrastructure
Viewpoint
Publish data mediated by ESB
Scalability of ESB
Reduce maintenance cost
Resource Efficiency of system
Usabilty of critical real time consumers
Minimize Resource bottlenecks
Extensibility of Consumer
Maximize processes per unit of infrastructure
Min. publish code change for additional data requirements
Min. publish code change for adding targets
Maximize concurrent consumer request per infrastructure
Maximize response time for time critical requests
Reduce resouces per infrastructure request
Reduce allocation time of resource per request
Publish data to providers
Publish data asynchronously
Publish data synchronously


Consumer
Viewpoint
Publish data mediated by ESB
Publish data asynchronously
Reduce maintenance cost
Reduce development cost
Maintainability of consumer
Usability of System
Effective time use
Understandability of consumer code
Response time of publishing handling
Reduce Complexity of consumer code
Design against operational requirements
Publish data to provider
Simplify publish / response handling
Publish data asynchronously
Publish data synchronously

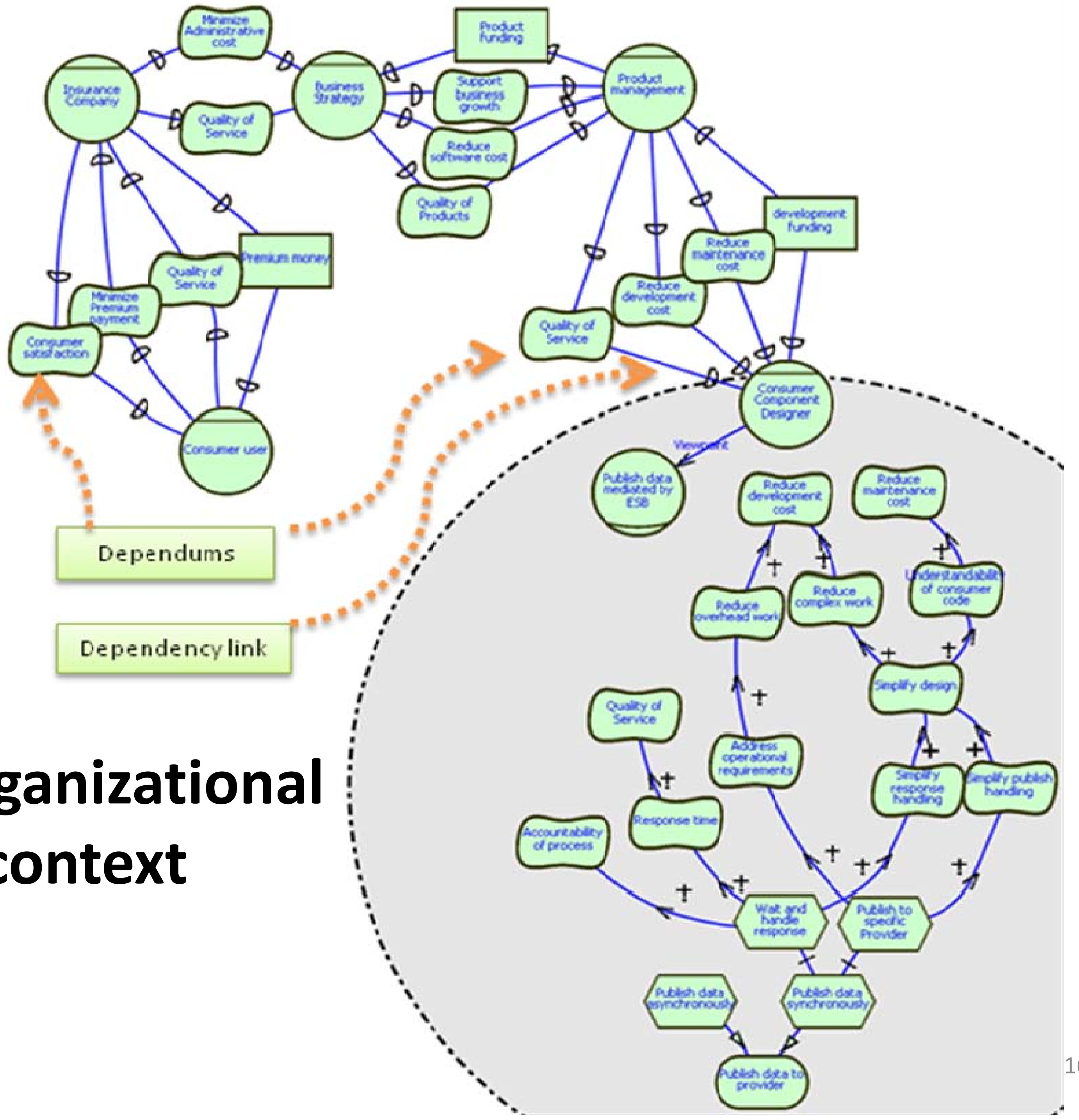
Consumer Component design argumentation viewpoint:

Use Synchronous Messaging!

- **Simplifying Consumer component**: Synchronous messaging simplifies the Consumer component design and code. It simplifies sending the new policy request to a specific Provider, and also simplifies handling the response received from the Provider. Simplifying is important since it make code more understandable which contributes to reduce the component's maintenance cost.
- **Simple is cost efficient:** Simpler design is easier and faster to write, which reduces development cost.
- **Synchronous is fast:** Synchronous messaging returns an immediate response. The ESB works to immediately fulfill a synchronous request. This improves response time for the Consumer component and contributes to the customer's quality of service perception.
- **Improved design accountability:** Having a response returned immediately, improves the control the Consumer component designer has over the overall policy submission process. Using synchronous messaging allows the Consumer component designer to directly design for failure contingencies.

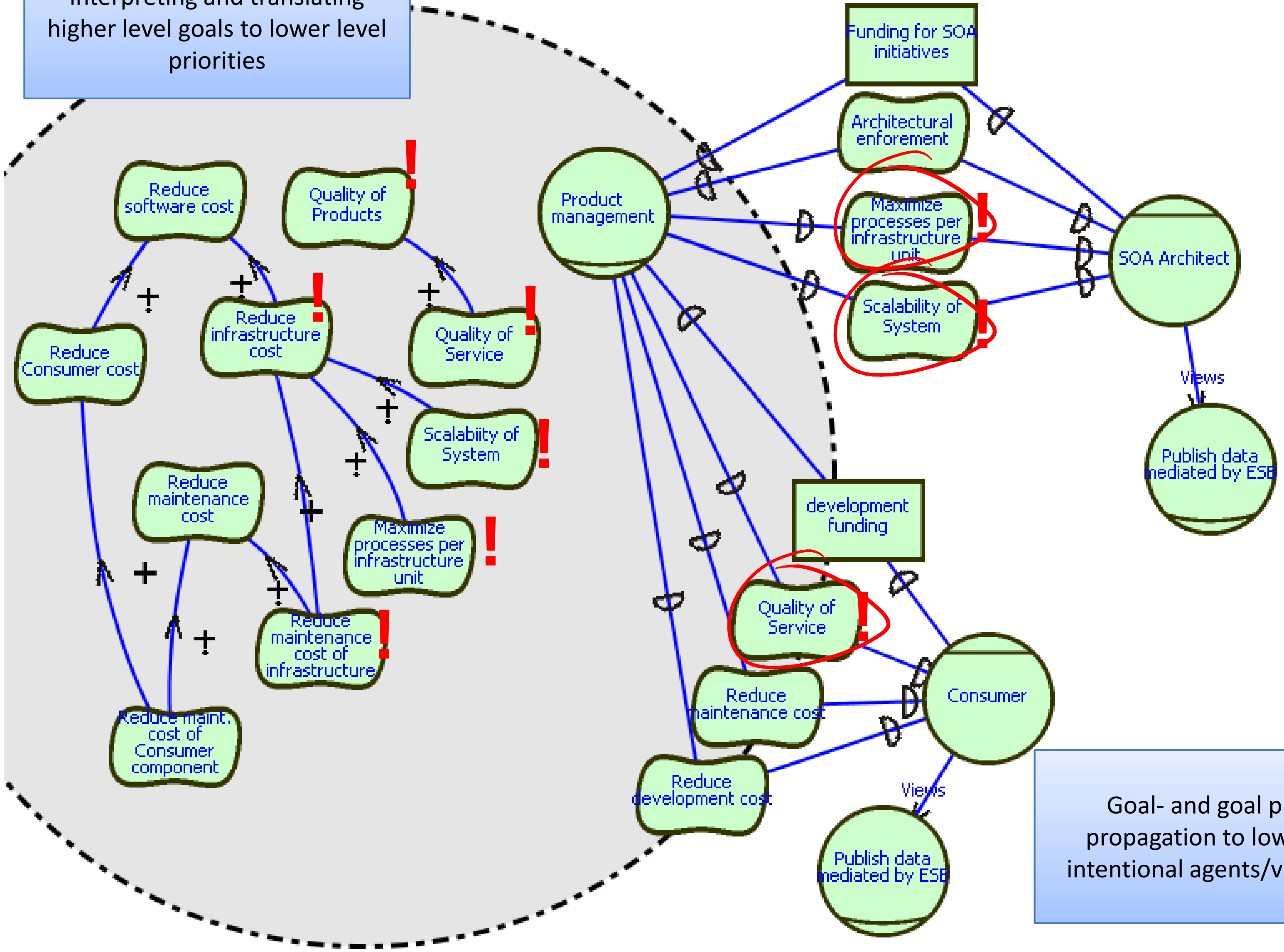


Higher level organizational reasoning context
Funding for SOA initiatives
Architectural enforement
Usability of System
Scalability of System
Agility of System
Product management
development funding
Reduce maintenance cost
Reduce development cost
Component development
ISA
Consumer Designer
SOA Architect
Is-Part-of
Reduce maintenance cost
Usability of critical real time consumers
Scalability of ESB
ESB Messaging infrastructure
Viewpoint
Publish data mediated by ESB
Reasoning about the Consumer and the Enterprise System as a whole
Scalability of ESB
Reduce maintenance cost
Resource Efficiency of system
Usabilty of critical real time consumers
Extensibility of Consumer
Minimize Resource bottlenecks
Maximize processes per unit of infrastructure
Min. publish code change for additional data requirements
Min. publish code change for adding targets
Maximize response time for time critical requests
Maximize concurrent consumer request per infrastructure
Reduce resouces per infrastructure request
Reduce allocation time of resource per request
Publish data to providers
Publish data asynchronously
Publish data synchronously
Publish data asynchronously
Viewpoint
Publish data mediated by ESB
Reduce maintenance cost
Reduce development cost
Maintainability of consumer
Usability of System
Effective time use
Understandability of consumer code
Reduce Complexity of consumer code
Response time of publishing handling
Design against operational requirements
Simplify publish / response handling
Publish data to provider
Publish data asynchronously
Publish data synchronously
Reasoning about the Consumer application and its components
Contrasting argumentation



Broader organizational reasoning context

Product management interpreting and translating higher level goals to lower level priorities



Goal- and goal priority propagation to lower level intentional agents/viewpoints

Case study results/Contributions

- **Is distributed reasoning and decision-making a problem in enterprise architecture evolution projects?**
  - CTO and SOA enterprise-wide architect report **struggling to convince** other decision-making stakeholders for the need of adopting SOA design principles
  - **Intentional viewpoint** was identified as needed to deal with design reasoning amongst designers with overlapping design responsibilities
- **Do intentional viewpoints help *in representing and communicating* distributed decision-making in case of overlapping responsibilities**
  - Feedback indicate intentional viewpoint modeling helps in **documenting side-by-side argumentation** of different stakeholders and designers during a design discussion, providing a useful communication tool
  - Simple actor models help provide reminders to relevant stakeholders why SOA principles were adopted
  - Such documentation helps **reduce the need for face-to-face** discussions between the SOA architect and enterprise application designers
- **Does including higher level organizational participants into a model provide any value?**
  - The CTO and SOA architect **perceived value** in putting design discussions into broader organizational decision-making context.
  - Was also seen as contribution to **IT Governance** – the need to justify architectural (SOA) choices in light of organizational strategic goals and directions.

Related work

- **Architecture decisions as "first class" modeling elements:**
  - Jansen & Bosch,
    - work on architecture decision modeling
  - Olaf Zimmerman
    - work on SOA architecture decision modeling for reuse (SOAD) – approach not SOA specific

No goals, no organizational concept, no distributed decision-making

- **i* applied to Architecture**
  - Grau, Franch
    - i* as architectural description language
  - Kolp, Mylopolous, Castro
    - i* architecture as organizational structures

No distributed agent-oriented reasoning (use of global SRs),

No organizational stakeholders included in distributed reasoning

Castro, J., Kolp, M., & Mylopoulos, J. (2001). A Requirement-Driven Software Development Methodology. Proc of the 13th International Conference on Advanced Information Systems Engineering CAiSE 01.
Grau, G. and X. Franch, On the Adequacy of i* Models for Representing and Analyzing Software Architectures. Proceedings of the First International Workshop on Requirements, Intentions and Goals in Conceptual Modeling (RIGiM'07), 2007: p. 296-305.

Kolp, M. and J. Mylopoulos. Software Architecture as Organizational Structures. in Proceedings ASERC Workshop on "The Role of Software Architectures in the Construction, Evolution, and Reuse of Software Systems. 2001. Edmonton, Canada.
Jansen, A., Bosch, J., 2005. Software architecture as a set of architectural design decisions. In: Proceedings of the Fifth Working IEEE/IFIP Conference on Software Architecture, November 2005. IEEE Computer Society, Washington, DC, pp. 109–1
Zimmermann, O., et al., *Reusable Architectural Decision Models for Enterprise Application Development, in Software Architectures, Components, and Applications. 2008, Springer Berlin /Heidelberg. p. 15-32.*

Future work

- Additional Phoenix study data analysis
- Simplify models, adapted to different types of stakeholders and designers (maintainers, designers, reuse managers, middle and upper management, etc.)
- Systematic comparing and contrasting of intentional viewpoint reasoning, while dealing with different decision scopes and levels of abstraction
- Larger scale documentation and analysis of enterprise architecture decision making, by use of agent types, and inheritance, instantiations, etc.
- Integration with enterprise architecture modeling approaches
- Tool support in enterprise setting