Centralize or Decentralize? A Requirements Engineering Perspective on Internet-Scale Architectures

Eric Yu
University of Toronto
http://www.fis.utoronto.ca/~yu

Large-scale systems are hard to design because of tensions among many interacting forces. For example, in considering whether to centralize or decentralize a function or resource, the designer has to make tough tradeoffs among competing requirements such as performance, cost, usability, reliability, security, time-to-market, maintainability, evolvability, etc. In the area of Requirements Engineering, techniques have been developed to support the managing of large numbers of requirements (functional and non-functional), detecting and analyzing their interactions, using requirements to guide the exploration, pruning and evaluation of design alternatives, and to support change. Such techniques typically emphasize the need to find suitable ways to represent knowledge, and to build tools that manipulate that knowledge in support of human design activities [MBY97]. A knowledge-based approach to software engineering thus may include the following components:

1. a representational framework (notations, models, languages, ontologies) that is expressive enough to deal with the subject matter: domain concepts, requirements, elaboration steps, design techniques, design configurations, design steps and process, alternatives, relationships, etc.,
2. analysis and design techniques that make use of the semantics of the modelling constructs to support the engineering activities, e.g., analyzing interactions among requirements, generating design options, evaluating implications of design alternatives, etc.,
3. collections of reusable knowledge (knowledge bases), ranging from case studies to generic knowledge such as common types of requirements and their possible elaboration, design principles, methods, rules, techniques, patterns of solutions to common design problems, architectures, frameworks, etc.,
4. methodologies for guiding the use of the approach, models, principles, techniques, etc., in various settings, and
5. tools which makes use of the structure and semantics of the knowledge to automate some aspects of the engineering activities, e.g., visualization, animation, simulation, verification, support for reasoning (e.g., qualitative, quantitative, case-based, etc.), and basic management facilities (maintaining design history, traceability, navigation, query, retrieval, version and change management, etc.).

The NFR framework [MCY99] [CNYM 00], for example, treats non-functional requirements such as performance, reliability, usability, etc. as goals applied to various aspects of a system. These are to be incrementally elaborated and refined until specific techniques for addressing them are identified. Each potential solution or refinement may have implications for other requirements. The conflicts and synergies may be detected via a knowledge base of correlations. The designer makes decisions by evaluating the tradeoffs among competing (and complementary) goals. The process involves the construction of a graph representing the design space. The elaboration, pruning, and evaluation of the graph is guided by previously accumulated reusable knowledge as well as case-specific judgements [CY98].

Considerations to centralize or to decentralize may come up at various points in the design process, and when considering different aspects of the system. From the viewpoint of a systematic, knowledge-based, requirements-driven approach to design, centralization and decentralization would refer to broad classes of design techniques or design patterns that have been invented over the years in a number of design areas. Thus the centralize/decentralize question may arise when considering transaction processing, long-term storage, system availability, security, or management functions. Specific techniques for addressing performance, security, reliability, etc. may have classes of solutions that are centralized or decentralized. For example, replication for speed of global access, distributed data close to source or user for local processing needs, redundancy for reliability, centralized management to reduce management costs, single database to avoid inconsistencies, fewer sites to reduce security exposures, etc. Each of these techniques tends to address one primary requirement, but typically have impacts on other requirements. A systematic approach therefore allows these complex interacting issues to be discerned, clarified, and analyzed.

The centralization/decentralization question typically arises within the context of many other potential techniques which may be or need to be used in conjunction, e.g., interface and protocol adoption or design, algorithms, database design, data integration issues, etc. Also, the centralization/decentralization question may arise at each of many layers in the architecture. A systematic design process that keeps track of all these steps and linkages can provide traceability and revision support through the proper tools.

In most systems, requirements come from many quarters – various kinds of users, operations personnel and management in the user organization, and developers, product managers, project managers, quality assurance, marketing, etc., in the development organization. Tradeoffs among competing requirements are therefore not among requirements as abstract impersonal goals, but as somewhat negotiable stakes among stakeholders [BI96]. Organizational issues therefore affect technical decisions in significant ways, resulting in architectures that reflect some aspects of organizational structure.

In Internet-based applications, the organizational factors are likely to be more pronounced since there may be many distinct economic and legal entities involved in the development, use, and management of the system, all with their distinct interests. The centralize-or-decentralize question arises at many levels not only because of the inherently distributed nature of the Internet, but also because there can be many ways of dividing up the scope of control at various levels. Some of these divisions include:

- administrative and management domains
- trust domains, from the viewpoint of each stakeholder or classes of stakeholders, e.g., application providers, network providers, user organizations, end-users, intermediaries, etc.
- developer domains – divisions of responsibility in the development organization(s)
- ownership domains
- operations management domains – e.g., failure recovery, performance optimization, load balancing, etc.
- subsystems, components, modules at various levels in the technical architecture.

These different ways of dividing up the scope of control may intersect in different ways too. For example, the trust domains may coincide with administrative domains, ownership domains may overlap with design domains, etc. Sometimes these alignments are by design, other times they are incidental. In any case, the alignments may be imperfect and may drift over time.

Each scope of control entails some localized tradeoffs among competing goals relevant to that domain (both at development time and during operations) as well as negotiations across domain scopes.

The systematic, goal-oriented modelling and support of the design process discussed above (e.g., as offered in [CNYM00] and [BI96]) therefore need to be extended to explicitly deal with the complexity of "organizational issues" arising from the many different ways in which stakeholder interests arise, how they are resolved and manifested in architectures and designs, and how they evolve over time.

The ***i**** framework [Yu95] introduces the intentional actor as a modelling abstraction to deal with locality and distribution at an intentional level (***i**** stands for distributed intentionality). Actors are intentional in that they have goals, beliefs, abilities, commitments, and so on, and relate to each other in terms of these intentional properties. Actors depend on each other for goals to be achieved, tasks to be performed, and resources to be furnished. While each actor is free to pursue its interests autonomously, it must also consider the consequences of its decisions and actions because of its relationships with other actors.

The deliberations of each actor is modelled in a similar fashion as in the graph structure of the NFR framework (the Strategic Rationale model in ***i****). However, the design space is now carved up into many localized spaces. The intentional relationships among actors define the interfaces among the localized spaces (the Strategic Dependency model in ***i****). Actors have limited knowledge about the internal rationales of other actors.

Modelling intentional relationships among actors and their rationales offers a higher level description of complex distributed architectures. Conventional architectural descriptions typically model non-intentional aspects such as data and control flows and interfaces. Intentional models explicitly portray the kinds of design freedoms that each architectural unit has, how these freedoms are exercised during design to meet competing demands, and how the freedoms in one unit impinge on the freedoms of other units. Design alternatives can be analyzed in terms of what design freedoms are allocated to which architectural units, and the consequences of such allocations. Explicit representation of design goals allows means-ends analysis and exploration of alternatives.

The ***i**** framework was originally developed to model primarily human organizational issues (e.g., as they arise in business process modelling [YML96] or enterprise modelling [Yu99]). It is currently being extended to model technical system architectures [CGY99][GY00]. The intentional level of modelling offers a good common framework for understanding and analyzing the interactions among human organizational issues and technical system design issues, as is especially needed in Internet-scale applications. Research challenges exist in each of the five areas:

- representation: What modelling constructs are appropriate for describing architectures at an intentional level?
- analysis and design: What computational techniques are available for supporting the exploration and evaluation of intentional descriptions of architectures?
- collections of reusable design knowledge: What are the available bodies of knowledge to support this area of design, and how can they be encoded to take advantage of tool support?
- methodologies: How can these models and techniques be used in practical settings?

- tools: What computational environments and infrastructures are needed to make the approach effective?

The TWIST 2000 [TWIST00] workshop theme raises good opportunities for testing and further developing this approach and framework. Many current Internet-based systems are hastily constructed using ad hoc approaches and techniques, in order to meet tight time-to-market requirements. These systems risk becoming legacy systems, as market, technology, and personnel shifts continue at rapid rates. The requirements-oriented, knowledge-based approach will hopefully provide a more systematic way to support the development and evolution of Internet-scale systems.

References

[BI96] B. Boehm and H. In. Identifying Conflicts Among Quality Requirements. *Proceedings of the Second International Conference on Requirements Engineering.* (ICRE'96) Los Alamitos, Calif.: IEEE Computer Society Press. Also appeared in *IEEE Software* (March 1996).

[CNYM00] L. Chung, B.A. Nixon, E. Yu, J. Mylopoulos. *Non-Functional Requirements in Software Engineering*. Kluwer Academic Publishers, 2000.

[CGY99] L. Chung, D. Gross, E. Yu. Architectural Design to Meet Stakeholder Requirements. *Software Architecture*, Patrick Donohue, ed., Kluwer Academic Publishers. 1999. pp. 545-564. (TC2 First Working IFIP Conference on Software Architecture (WICSA1), 22-24 February 1999, San Antonio, Texas, USA.)

[CY98] L. Chung and E. Yu. Achieving System-Wide Architectural Qualities. *OMG-DARPA-MCC Workshop on Compositional Software Architectures*, Janurary 6-8, 1998, Monterey, California.

[GY00] D. Gross and E. Yu. From Non-Functional Requirements to Design through Patterns. *Proceedings of the 6th International Workshop on Requirements Engineering: Foundations for Software Quality (REFSQ 2000). June 5-6, 2000, Stockholm, Sweden.*

[MBY97] J. Mylopoulos, A. Borgida, and E. Yu. Representing Software Engineering Knowledge. *Automated Software Engineering*, Kluwer Academic Publishers, 4(3): 291-317. July 1997.

[MCY99] J. Mylopoulos, L. Chung, and E. Yu. From Object-Oriented to Goal-Oriented Requirements Analysis. *Communications of the ACM*, 42(1): 31-37, January 1999.

[TWIST00] The Workshop on Internet-scale Software Technologies – Organizational and Technical Issues in the Tension Between Centralized and Decentralized Applications on the Internet (TWIST 2000), July 13-14, 2000. University of California, Irvine, Irvine, California, USA. Call for participation. http://www.isr.uci.edu/events/twist/twist2000/ Last viewed May 15, 2000.

[Yu95] E. Yu. *Modelling Strategic Relationships for Process Reengineering*. Ph.D. Thesis. Dept. of Comp. Sci., University of Toronto. 1995.

[Yu99] E. Yu. Strategic Modelling for Enterprise Integration. *Proceedings of the 14th World Congress of International Federation of Automatic Control* (IFAC'99), July 5-9, 1999, Beijing, China. pp. 127-132. Permagon, Elsevier Science.

[YML96] E. Yu, J. Mylopoulos and Y. Lespérance. AI Models for Business Process Reengineering. *IEEE Expert: Intelligent Systems and Their Applications*, August 1996, pp. 16-23.