Enhancing the Software Architecture Analysis and Design Process with Inferred Macro-Architectural Requirements

Plamen Petrov, Ugo Buy
Department of Computer Science
University of Illinois at Chicago
Chicago, IL, USA
ppetro2@uic.edu, buy@cs.uic.edu

Robert L. Nord*
Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA, USA
rn@sei.cmu.edu

***Abstract*— Traditionally the flow of authoritative information and control in requirements and software engineering is from requirements to architecture, design, development, implementation and testing. Iterative, spiral and agile methods, among others, have introduced increments and iterations in eliciting and discovering requirements within the project life cycle. Yet the authoritative flow of information across organizational boundaries within the enterprise continues to be from requirements to architecture to design. We argue that two additional implicit sources of information should be included in the requirements engineering process, contextual environment concerns and architectural patterns and heuristics. To account for these two sources of implicit requirements information we introduce the concept of forward and backward inferred macro-architectural requirements. Forward inferred macro-architectural requirements are elicited from contextual environment concerns. Backward inferred macro-architectural requirements are extracted through a reverse requirements elicitation process from architectural heuristics and patterns. We have observed significant improvements in the efficiency of the development processes and the quality of the final software products as a result of making inferred macro-architectural requirements explicit.**

***Keywords*-Enterprise architecture; software architecture; contextual concern; architectural heuristics; architectural requirements; requirements engineering; analysis and design.**

I. Introduction

Current software architecture methods and processes generally rely on requirements being elicited and provided as an input to the selection of the appropriate software architecture [2,4,13,29,30]. The requirements engineering process typically includes numerous practices and steps (e.g., requirements elicitation, requirements analysis, negotiation, requirements specification, system modeling, requirements validation, requirements management) [15]. A first assumption is that stakeholder concerns can be elicited and explicitly captured as requirements.

There are many software development methods and processes, some being mostly sequential in nature (e.g., waterfall), others being iterative, incremental, spiral, and agile. Most generally accepted and widely adopted requirements engineering techniques rely on a second assumption that the flow of authoritative information and control is from problem domain to solution domain, that is, from stakeholder requirements to system requirements, to architecture and design [15]. The second assumption does not preclude iterations between requirements and architecture, but rather speaks to the higher level of information authority assigned to functional and quality attribute (also known as non-functional) requirements compared to architectural considerations. As an example the Twin Peaks [24] model as adaptation of the spiral life-cycle model explicitly elevated architecture as a “peak” that together with the requirements “peak” is iterated upon early on. Yet the possibility that architectural considerations can be as authoritative in nature as functional and quality attribute requirements is not sufficiently explored in theory and rarely applied in practice. Iterative, spiral, and agile processes remove the chronological linearity of lifecycle phases and tasks, but the authoritative flow of information and control remains largely top down (from requirements to architecture).

A third assumption among software architecture analysis and design methods and processes is that functional requirements, quality attribute requirements and design constraints are the driving forces that determine the selection of the appropriate software architecture for the system under development [35,11]. Specifically the major focus of current methods and processes is on explicitly identifiable and project related functional and quality attribute requirements.

Our research and industrial experience identify some limitations of the three assumptions listed above and we propose extensions to address those limitations to support practical applications.

The problem with the first assumption is the belief that all types of concerns can be elicited from the stakeholders and explicitly captured as requirements that can drive the selection of an appropriate architecture. We argue that there are more compelling contextual environment concerns that can be appropriately translated into macro-architectural

* The views and conclusions contained in this document are solely those of the individual creator(s) and should not be interpreted as representing official policies, either expressed or implied, of the Software Engineering Institute, Carnegie Mellon University, the U.S. Air Force, the U.S. Department of Defense, or the U.S. Government.

heuristics, decisions and patterns through a decision analysis process relying on implicit knowledge, best practices and documented rationale.

In addition, we observe that certain architectural heuristics, decisions and patterns can be directly derived using implicit knowledge and best practices. Therefore, we seek to address the limitations of the second assumption and allow for requirements to be inferred from already made architectural decisions and formally captured as explicit architectural requirements. We call such requirements backward-inferred macro-architectural requirements.

Finally, the third assumption does not fully account for the realities of software development practice in which contextual environment concerns frequently determine the appropriateness of software product decisions. We introduce an additional driving force in the form of forward-inferred macro-architectural requirements that are explicitly captured by analyzing contextual environment concerns [27].

II. Requirements Engineering and Architecture Design as Decision Analysis Problem

We have advanced the proposition that the software architecture analysis and design process is a decision analysis discipline. We also introduced the contextual environment concept as an approach to support the design of an appropriate architecture for the system under construction that best fits the enterprise-level and system-level context [26,27].

In this paper we propose a novel approach to integrate the requirements engineering process [14,16,25] with the architecture analysis and design process through forward-inferred and backward-inferred macro-architectural requirements. In line with our earlier work we integrate concepts and techniques from the decision analysis theory and practice into the requirements and architecture processes.

In [27] we introduced the concept of macro-architecture – a set of broad, coarse grain, and highly impactful architectural decisions that can be made early on in the design process and that can be inferred and rationalized from the overall enterprise-wide and system-wide contextual environment. The concept of macro-architecture and macro-architectural requirements is best understood within the context of “decision frame” as presented in decision analysis theory.

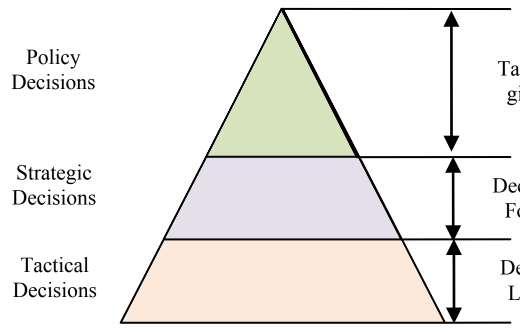

Figure 1: Decision Hierarchy Model.

Decision analysis introduces the decision hierarchy as a useful tool to define the frame for a decision problem [20]. The theory and practice of decision analysis identify the framing of the decision focus as a critical step in the decision process. The middle band in the decision hierarchy is the decision frame - the focus of the decision process - where the strategic decisions should be made. In the top tier are policy decisions, an umbrella designation for assumptions and preexisting decisions that constrain the problem under consideration. Finally, the lowest tier covers tactical decisions that can be decided later without affecting the quality of the decision process.

As part of our proposal to approach the early phase of the architecture analysis and design as a decision analysis problem we have mapped the decision hierarchy to an architectural decision hierarchy model.

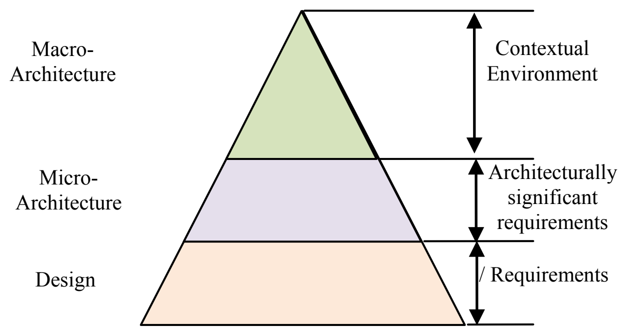

Figure 2: Architecture Decision Hierarchy Model.

We propose to capture explicitly and to include the contextual environment as part of the architectural process. We have observed that the contextual environment concerns–whether at the enterprise-level or the system-level–constrain the solution space to a subset of appropriate architectural options. We have observed that the contextual environment has not been traditionally treated explicitly and as a result the constrained solution space has been approached only implicitly. Our approach captures explicitly the constrained solution space that is affected by the contextual environment as a macro-architecture. The traditional software architecture – which we call micro-architecture – is constrained by the macro-architecture.

III. Multilevel Context-Aware Software Architecture

Figure 3 graphically describes the multilevel context-aware software architecture analysis and design process. The top arrow describes the traditional flow of functional requirements, quality attribute requirements and design constraints to the definition of a “fit for purpose” software architecture, called micro-architecture. The bottom arrow expresses the impact that the contextual environment has on the selection and definition of a “fit to context” software architecture, called macro-architecture. Global analysis (part of the Siemens Four Views architecture design approach

[13]) is an approach to expand the requirements engineering process to cover concerns broader than the immediate functional and quality attribute requirements [12], [23]. We build on top of the global analysis approach to formalize the concept of contextual environment concerns that additionally cover some implicit sources of information to discover architectural requirements.

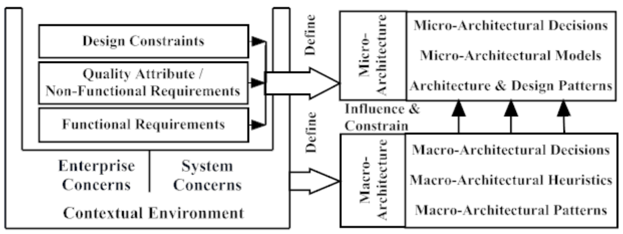
Design Constraints
Quality Attribute / Non-Functional Requirements
Functional Requirements
Enterprise Concerns
System Concerns
Contextual Environment
Define
Define
Micro-Architecture
Macro-Architecture
Influence & Constrain
Micro-Architectural Decisions
Micro-Architectural Models
Architecture & Design Patterns
Macro-Architectural Decisions
Macro-Architectural Heuristics
Macro-Architectural Patterns

Figure 3: Multilevel Context-Aware Architecture.

Based on architectural best practices, experience and rules of thumb, it can be practical to translate certain enterprise-level and system-level contextual environment concerns directly into macro-architectural heuristics, decisions and patterns, bypassing the step of translating the contextual environment into descriptive requirements. The traceability and proper documentation of the decisions is handled through the rationale section in the architectural heuristics and decisions artifacts [17,18,31,33].

Example 1–Suppose that the enterprise-level contextual environment includes a development organization committed to innovation that is composed of several geographically distributed teams each specializing in a certain functional domain and technology. Some core functional domains are funded as an allocation across the entire organization and managed as a portfolio through a rigorous development lifecycle. Others are funded as a direct charge back to the specific project with target profit and loss (P&L) and return on investment (ROI) guidelines and managed through lightweight agile processes. In this case, an architecture with a good chance of success should account for those contextual environment concerns by selecting macro-architectural patterns and heuristics that fit the context.

The context-aware macro-architecture guides the solution structure to reflect the structure of the development organization and partitions the functionality to reflect the unit specializations. Additionally, functionality that benefits the entire enterprise is partitioned and assigned to the shared services development unit supported through the allocation-based funding model. Since the organization is committed to innovation, high-risk and leading-edge capabilities could be assigned to the agile development units to avoid unnecessary overhead in a fluid environment. Alternatively, these capabilities could be assigned to the units that allow for charge back of high-risk tasks with a probability of failure in their funding model.

Since the organization is both geographically distributed and committed to innovation, the architectural partitioning emphasizes loose coupling to allow for frequent changes in the innovative modules, as well as simple, but rigorous interface contracts among modules to avoid integration challenges and undesired dependencies. All these considerations have a profound effect on the selection of the appropriate software architecture and the ultimate success of the product.

Notice that these considerations, related to the contextual environment, are rarely discussed or captured as requirements in the current requirements engineering and software architecture analysis and design practices; they would be quite hard to capture as explicit requirements. These considerations involve implicit knowledge, architectural rules of thumb, and practical experience. Yet an experienced and knowledgeable architect would likely identify those patterns and heuristics. These macro-architectural decisions should be captured as rationale and justifications in the architectural artifacts to allow for traceability, reasoning on the appropriateness of the decisions, and further tradeoff analysis.

Our approach bears some similarities to goal-oriented requirements analysis [1,7,9,10,19,28,32], and the evolution from task-oriented approach [3] and expands on the work to relate business goals to architectural requirements [6] and business goals to software architecture [5,22]. Traditional task and problem-oriented requirement analysis methods focus on what features and qualities the system under design should support. Goal-oriented analysis broadens the process to include the motivation and rationale of why the system should be constructed. We expand on these concepts by adding implicit enterprise-wide and system-wide contextual environment considerations to the requirements process that may not be viewed as explicit goals for the specific project, but have just as profound and significant impact on determining the appropriate architecture and design.

IV. Macro-Architectural Requirements

Our approach introduces the concept of macro-architectural requirements as an intermediate artifact facilitating the selection of an appropriate software architecture that fits the context. Macro-architectural requirements are a way to address the limitations of the second and third assumptions as described in Section 1. We integrate those macro-architectural requirements with the multilevel context-aware software architecture process to improve the software lifecycle efficiency and the quality of the target software product.

Figure 4 shows an overview of forward-inferred and backward-inferred macro-architectural requirements within the context of the multilevel context-aware software architecture process: The shaded boxes in Figure 4 represent the macro-architectural requirements. There are two types of macro-architectural requirements – forward-inferred and backward-inferred – that differ by how they have been elicited.

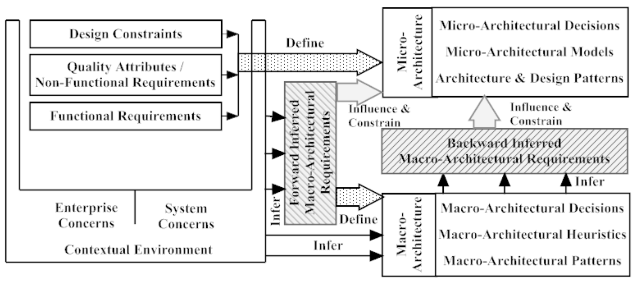
Design Constraints
Quality Attributes / Non-Functional Requirements
Functional Requirements
Enterprise Concerns
System Concerns
Contextual Environment
Define
Infer
Forward Inferred Macro-Architectural Requirements
Influence & Constrain
Backward Inferred Macro-Architectural Requirements
Micro-Architecture
Micro-Architectural Decisions
Micro-Architectural Models
Architecture & Design Patterns
Macro-Architecture
Macro-Architectural Decisions
Macro-Architectural Heuristics
Macro-Architectural Patterns

Figure 4: Macro-Architectural Requirements.

The forward-inferred macro-architectural requirements are inferred from the contextual environment–enterprise-level and system-level contextual environment concerns. Those explicitly captured, intermediate requirements serve both to influence and constrain the traditional software architecture process (described as micro-architecture), as well as to facilitate the decision analysis process that defines the macro-architecture (described in the form of macro-architectural decisions, heuristics, and patterns).

As shown in Figure 4, backward-inferred macro-architectural requirements are inferred *post factum* from an already pre-defined macro-architecture. The macro-architecture that serves as the basis for the backward-inferred architectural requirements is derived earlier from the contextual environment through decision analysis processes as described elsewhere [27]. Those explicitly captured, backward-inferred requirements serve to influence and constrain the traditional software architecture process described as micro-architecture in the figure above.

We propose a reverse requirements elicitation process through which we extract the backward-inferred macro-architectural requirements from the architectural decisions, heuristics and patterns which constitute the macro-architecture. The distinction between forward-inferred and backward-inferred macro-architectural requirements is somewhat arbitrary and depends largely on the practicality of capturing architectural requirements directly from the contextual environment (in the case of forward-inferred requirements) or from a pre-defined macro-architecture that serves as an architectural translation of some more elusive and implicit contextual environment concerns.

A. *Forward Inferred Macro-Architectural Requirements*

Our research and practical experience tell us that significant macro-architectural requirements inferred from the contextual environment are typically ignored and missed. We believe that the primary reason for this is the perception that architectural requirements are somehow secondary to the requirements elicited from business stakeholders. Evidently, some requirements with mostly technical and architectural implications are captured as either quality attribute requirements or design constraints. However, those are typically the obvious requirements that can be relatively easily detected by sophisticated business or senior systems stakeholders.

The more subtle architectural requirements typically can only be identified by experienced and knowledgeable architects. The more subtle requirements and concerns are usually relegated to the follow up software architecture and design processes. They are rarely captured as explicit requirements, and are considered as a lower priority with respect to functional and quality attribute requirements elicited directly from the business stakeholders. As a result, they are rarely considered with sufficient urgency and focus while making the initial architectural decisions. Our experience tells us that serious issues can result from missing those macro-architectural considerations.

Forward inferred macro-architectural requirements are becoming especially meaningful as the software development and information technology industries are continuously evolving away from mostly “contextually greenfield” to predominantly “contextually constrained” development and operational environments. Contextual environment concerns that should now play a significant role in the selection of the appropriate software architecture include legacy libraries for reuse, legacy applications for integration, legacy and dominant development and operational skill sets within the organization, geographic location of development and operational units, maturity and track record of development and operational units with functional domain, technologies, and methodologies, strategic technology vendor partnerships and long lasting licensing agreements, among others.

The availability, desirability and inherent constraints of third-party libraries and commercial off-the-shelf (COTS) products can be appropriately modeled as forward-inferred macro-architectural requirements. (See related work [34].) Several recent trends have further increased the usefulness of macro-architectural requirements, including cloud computing and open source software. Cloud computing in its different forms (Software as a Service / SaaS, Platform as a Service / PaaS, and Infrastructure as a Service / IaaS) introduces significant peculiarities that can be best captured as macro-architectural requirements and inferred by experienced architects.

B. *Backward Inferred Macro-Architectural Requirements*

As we explain in Example 1 above, there are contextual environment concerns that are hard or impractical to elicit as explicit requirements. Some of those more subtle contextual environment characteristics are most naturally translated into macro-architectural heuristics, decisions and patterns as a result of best practices, rules of thumb, and architectural experience. Once those contextual environment characteristics are implicitly translated in a target macro-architecture and documented as decision rationale and justification, some key architectural requirements can be reverse elicited and explicitly captured as backward-inferred macro-architectural requirements.

Looking at Example 1, the selected macro-architecture that translates the described contextual environment may include an explicit integration contract rigorously defined as a Web Services Description Language (WSDL) platform independent interface. The interface accommodates the geographically distributed development organization that has expertise and legacy functionality in several heterogeneous technologies. The geographical and technological distribution of expertise will result in functional partitioning of architectural components that will be developed independently, modified frequently, and yet integrated seamlessly with little overhead.

Some details of the proposed architecture and approach are best captured implicitly as part of the justification and rationale of the architectural decisions. There are other specifics that are best-captured explicitly as macro-architectural requirements. It could have been hard and impractical to try to elicit them in a forward manner from the contextual environment; however, they can be extracted quite easily from the defined macro-architecture and its rationale and justification.

Some trends in the software development and information technology industries make the backward-inferred architectural requirements especially applicable. The wide adoption of Service Oriented Architecture (SOA) as an architectural style applicable to numerous scenarios is one such major trend. Selecting the appropriate architectural style and respective technology standards can be implicitly captured as justification and rationale in the macro-architectural decisions, but then some key architectural decisions should be explicitly extracted and documented as backward-inferred macro-architectural requirements. (See [39] for WS* web services standards and [8] for RESTful web services architectural style).

Another major trend is the coming of age of software ecosystems as a widely adopted model for software development and delivery [21]. In many current scenarios the development and delivery of software functionality relies on a software ecosystem to be economically viable and practical. Software ecosystems include the development and delivery of mobile application through the Apple iStore and Google Play (formerly known as the Android market), accepting online payments from payment ecosystems like PayPal [38]. An additional example is given by healthcare applications compliant with the HL7 reference information model for semantic interoperability [36] and with the IHE technical framework and integration profiles [37]. These ecosystems entail some very involved and sophisticated architectural decisions, which would be hard to capture as forward-inferred macro-architectural requirements. They can be easily expressed as significant backward-inferred architectural requirements by a reverse elicitation process.

V. Reverse Requirements Elicitation process

We propose a reverse requirements elicitation process as the mechanism through which backward-inferred macro-architectural requirements can be extracted from the macro-architectural heuristics, decisions, and patterns that constitute the macro-architecture of a system.

The reverse requirements elicitation process relies on architectural justification and rationale to be included in the documentation of the macro-architecture. Once the macro-architecture is defined and documented as an implicit translation of the enterprise-level and system-level contextual environment concerns, a cross-functional team consisting of a senior software architect and a senior requirements analyst analyze the macro-architectural decisions and heuristics and extract explicit requirements in the same form used to capture functional and quality attribute requirements from the different stakeholders. In Example 1, the following set of backward-inferred macro-architectural requirements can be extracted, assuming an application project from the insurance industry.

- The system shall be partitioned into subsystems that are loosely coupled and integrated through standard Web Services implementations and described in WSDL.
- The WSDL interface definitions will be centrally defined and managed by the shared service department in San Jose.
- The pricing engine shall be developed by the rating team in South Carolina as an extension to the existing Cobol pricing engine and will be accessible through the standardly defined WSDL interface as specified by the shared services department.
- The benefit contract engine shall be developed by the account benefits team in Virginia as an extension to the existing Java contracting engine and will be accessible through the standardly defined WSDL interface as specified by the shared services department.

VI. Sample Case Study

In this section we capture one representative sample case study. This is one of about twenty representative samples that we studied to date.

Example 2–As part of our research and industrial experience we studied and then experimented with a real-work project from a US-based company. This example follows three versions of a system as it evolved over time. Version #1 was developed following traditional, industry-accepted requirements engineering and architectural analysis and design methodology. Version 2 was based on an initial iteration of the multi-level context-aware software architecture analysis and design method that explicitly treated the definition of a macro-architecture as an early step in the architecture process. Version 3 was based on a later iteration that further refined the process to include the concept of backward-inferred architecture requirements.

The company sponsored a strategically significant business project to develop a system to nominate, assess, and designate a vendor from its supply chain as a partner of excellence within a certain field of specialization. We discuss three versions of the system.

Version 1: The initial version of the system was developed following the standard software development lifecycle used in the company. The initial design and implementation was awarded to an external consulting company. After several years in production the system was experiencing significant technical issues and customer dissatisfaction. We were invited to perform an analysis of the system and to provide a recommendation on how to address the issues.

We reviewed the requirements of the system and performed an analysis based on an early version of the multilevel context-aware software architecture methodology. We concluded that at the root of the problem was an architecture that was inappropriate for the context, although the architecture was reasonable to satisfy the technical purpose of the system. More specifically, analyzing the enterprise-level contextual environment we noted that development was outsourced to an organization that specialized in software development based on leading edge and immature open source software. At the same time we noted that maintenance and operations were performed by the in-house organization that had skills and experience to work with fully supported commercial software from top-tier technology vendors like IBM and Microsoft.

The macro-architectural dissonance was significant – the maintenance and enhancement organization was unable to keep up with the immature and advanced technology expectations of an open source environment and even trivial defects ended up causing major system outages. The enhancement organization was unable to keep up with the enhancement requests, since the custom-built system included a domain specific language (DSL) that was rather immature and restrictive, yet the enhancement and maintenance organization did not have the expertise to support and enhance compiler and code generation technologies.

Additionally, the custom-built application had a tightly integrated architecture in order to support the original, specific business requirements, but was unable to keep up with the usability and graphical user interface features of GUI rich commercial surveying products. Trying to keep up with what was perceived by the business stakeholder as trivial usability features became a major consumer of development resources and cost that quickly reached unacceptable levels.

Version 2: After taking into account key enterprise-level contextual environment concerns (organizational skillset and outsourced development model) and key system-level contextual environment concerns (expectation to match the GUI features of commoditized COTS surveying tools), the architects within the organization defined a macro-architecture that partitioned the system into a surveying capability and other business logic and workflow capability. Additionally, the business stakeholder in collaboration with the architect included forward-inferred requirements that the surveying capability should be implemented by integrating a COTS package with rich GUI capabilities.

After another year we were brought in again to perform an assessment of the system. This assessment benefited from the earlier work assessing Version 1 of the system and confirmed the benefits of multi-level context-aware architecture analysis and design, since none of the issues observed in Version 1 of the system were found in Version 2. There were, however, new major issues with Version 2; analyzing those issues and their root cause played a role in formalizing the concept of backward-inferred macro-architectural requirements. Customer satisfaction with Version 2 of the system was again slipping. The system was experiencing instability and high number of defects. Additionally, the cost of enhancing and maintaining the system was unacceptably high, especially considering that a large portion of the system was integrating a commoditized, low-cost COTS platform.

Our assessment identified that during the earlier redesign iteration, the organization has failed to reverse elicit backward-inferred macro-architectural requirements. The selection of a COTS product to implement the surveying functionality introduced a set of architectural and usability constraints that were not explicitly captured and were ignored and contradicted. As a result, the loosely coupled partitioning of the system into a surveying subsystem, a workflow subsystem, a business rules subsystem, and a reporting subsystem was violated. Tight coupling and integration were attempted and the benefits of COTS modules were negated.

Additionally, backward-inferred usability requirements that resulted from the selection of a COTS surveying subsystem were not captured and understood by the business stakeholders as attempts to customize the COTS system were made. As a result, Version 2 was already running on a custom-built version of the COTS surveying product that introduced instability by running a specially-built, sparsely tested product and that cost significantly more to maintain and support than the general use, commodity product available to the market in general.

Version 3: In this version we required that a macro-architecture be defined to reflect the specific contextual environment that accounted for both enterprise-level and system-level contextual environment concerns. As a next step, forward-inferred macro-architectural requirements and backward-inferred macro-architectural requirements were explicitly extracted and reverse elicited, documented, reviewed, agreed upon, and prioritized together with the remaining functional and quality attribute requirements elicited directly from the different stakeholders.

As a conclusion to our case study, Version 3 of the product was built on an architecture that was both "fit for purpose" as well as "fit to context" as defined by the macro-architecture and the forward-inferred and backward-inferred macro-architectural requirements.

VII. Conclusion and Future Work

We have identified and described limitations in the current state of the art in the theory and practice of requirements engineering and software architecture analysis and design, where information traditionally flows from requirements to design. We propose an integrative framework that improves the efficiency of software development processes and quality of the overall software product under development. We have developed the proposed methods through observation and experimentation with real world industrial scenarios and by studying published research.

As part of our on-going research and industrial work we plan on refining the forward-inferred and backward-inferred macro-architectural requirements techniques in conjunction with our on-going work to further develop and formalize the multilevel context-aware software architecture analysis and design method. We plan on publishing further details of our work and empirical results as our work progresses.

References

[1] A. I. Anton, "Goal-Based Requirements Analysis," in Second International Conference on Requirements Engineering (ICRE 96) , IEEE, 1996, pp. 136–144.

[2] L. Bass, P. Clements, and R. Kazman, *Software architecture in practice*. Boston: Addison-Wesley, 2003.

[3] D. Bolchini and J. Mylopoulos, "From task-oriented to goal-oriented Web requirements analysis," in *Proceedings of the 7th International Conference on Properties and Applications of Dielectric Materials (Cat. No.03CH37417)*, Rome, Italy, 2003, pp. 166–175.

[4] P. Clements, F. Bachmann, L. Bass, D. Garlan, J. Ivers, R. Little, P. Merson, R. L. Nord, and J. Stafford, *Documenting software architectures : views and beyond*, 2nd ed. Upper Saddle River NJ: Addison-Wesley, 2011.

[5] P. C. Clements and L. Bass, "Using Business Goals to Inform a Software Architecture," in *RE*, 2010, pp. 69–78.

[6] P. C. Clements and L. Bass, "Relating Business Goals to Architecturally Significant Requirements for Software Systems," CMU/SEI, Technical Note CMU/SEI-2010-TN-018, May 2010.

[7] A. Dardenne, A. van Lamsweerde, and S. Fickas, "Goal-directed requirements acquisition," *Science of Computer Programming*, vol. 20, no. 1–2, pp. 3–50, Apr. 1993.

[8] R. T. Fielding, "Architectural styles and the design of network-based software architectures," University of California, Irvine, 2000.

[9] P. Giorgini, J. Mylopoulos, E. Nicchiarelli, and R. Sebastiani, "Reasoning with Goal Models," in *Conceptual Modeling — ER 2002*, vol. 2503, S. Spaccapietra, S. T. March, and Y. Kambayashi, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 167–181.

[10] D. Gross and E. Yu, "Evolving system architecture to meet changing business goals: an agent and goal-oriented approach," in *Proceedings Fifth IEEE International Symposium on Requirements Engineering*, Toronto, Ont., Canada, pp. 316–317.

[11] C. Hofmeister, P. Kruchten, R. L. Nord, H. Obbink, A. Ran, and P. America, "A general model of software architecture design derived from five industrial approaches," *Journal of Systems and Software*, vol. 80, no. 1, pp. 106–126, Jan. 2007.

[12] C. Hofmeister, R. L. Nord, and D. Soni, "Global Analysis: moving from software requirements specification to structural views of the software architecture," *IEE Proc., Softw.*, vol. 152, no. 4, 2005.

[13] C. Hofmeister, R. L. Nord, and D. Soni, *Applied software architecture*, 6. printing. Reading Mass. ;;Harlow: Addison-Wesley, 2006.

[14] E. Hull, K. Jackson, and J. Dick, "Requirements Engineering in the Problem Domain," in *Requirements Engineering*, London: Springer London, 2011, pp. 93–114.

[15] E. Hull, K. Jackson, and J. Dick, "Requirements Engineering in the Solution Domain," in *Requirements Engineering*, London: Springer London, 2011, pp. 115–136.

[16] E. Hull, K. Jackson, and J. Dick, "A Generic Process for Requirements Engineering," in *Requirements Engineering*, London: Springer London, 2011, pp. 25–45.

[17] A. Jansen and J. Bosch, "Software Architecture as a Set of Architectural Design Decisions," in *5th Working IEEE/IFIP Conference on Software Architecture (WICSA'05)*, Pittsburgh, PA, USA, pp. 109–120.

[18] A. Jansen, J. Der Ven, P. Avgeriou, and D. Hammer, "Tool Support for Architectural Decisions," in *2007 Working IEEE/IFIP Conference on Software Architecture (WICSA'07)*, Mumbai, India, 2007

[19] S. Liaskos, S. A. McIlraith, S. Sohrabi, and J. Mylopoulos, "Integrating Preferences into Goal Models for Requirements Engineering," in *2010 18th IEEE International Requirements Engineering Conference*, Sydney, Australia, 2010, pp. 135–144.

[20] Matheson, James E., "Decision Analysis = Decision Engineering," in *Emerging Theory, Methods, and Applications*, INFORMS, 2005.

[21] D. G. Messerschmitt and C. Szyperski, Software ecosystem : understanding an indispensable technology and industry. Cambridge Mass.: MIT Press, 2003.

[22] J. A. Miller, R. Ferrari, and N. H. Madhavji, "Characteristics of New Requirements in the Presence or Absence of an Existing System Architecture," in *2009 17th IEEE International Requirements Engineering Conference*, Atlanta, Georgia, USA, 2009, pp. 5–14.

[23] R. L. Nord and D. Soni, "Experience with Global Analysis: A Practical Method for Analyzing Factors that Influence Software Architectures," in *2nd International Workshop on SofTware Requirements to Architectures (STRAW '03), co-located with the International Conference on Software Engineering (ICSE)*, 2003.

[24] B. Nuseibeh, "Weaving together requirements and architectures," *Computer*, vol. 34, no. 3, pp. 115–119, Mar. 2001.

[25] A. Osterwalder and Y. Pigneur, "An ontology for e-business models," in *Value Creation from E-Business Models*, Oxford: Elsevier Butterworth-Heinemann, 2004, pp. 65–97.

[26] P. Petrov and U. Buy, "A Systemic Methodology for Software Architecture Analysis and Design," 2011, pp. 196–200.

[27] P. Petrov, U. Buy, and R. L. Nord, "The Need for a Multilevel Context-Aware Software Architecture Analysis and Design Method with Enterprise and System Architecture Concerns as First Class Entities," 2011, pp. 147–156.

[28] R. S. Sangwan and C. J. Neill, "How Business Goals Drive Architectural Design," *Computer*, vol. 40, no. 8, pp. 85–87, Aug. 2007.

[29] M. N. Shaw and D. Garlan, *Software architecture : perspectives on an emerging discipline*. Upper Saddle, New Jersey: Prentice Hall, 1996.

[30] R. N. Taylor, N. Medvidovic, and E. M. Dashofy, *Software architecture : foundations, theory, and practice*. Hoboken N.J.: Wiley, 2010.

[31] J. Tyree and A. Akerman, "Architecture Decisions: Demystifying Architecture," *IEEE Softw.*, vol. 22, no. 2, pp. 19–27, Mar. 2005.

[32] A. van Lamsweerde, "Goal-oriented requirements engineering: a guided tour," in *Proceedings Fifth IEEE International Symposium on Requirements Engineering*, Toronto, Ont., Canada, pp. 249–262.

[33] J. S. V. D. Ven, A. G. J. Jansen, J. A. G. Nijhuis, and J. Bosch, "Design Decisions: The Bridge between Rationale and Architecture," in *Rationale Management in Software Engineering*, 2006,

[34] K. Wallnau, S. Hissam, and R. C. Seacord, *Building systems from commercial components*. Boston: Addison-Wesley, 2002.

[35] R. Wojcik, F. Bachmann, L. Bass, P. C. Clements, P. Merson, R. L. Nord, and W. G. Wood, "Attribute-Driven Design (ADD), Version 2.0," CMU/SEI, CMU/SEI-2006-TR-023, 2006.

[36] "HL7 - Health Level 7 Reference Information Model Standards." [Online].Available: http://www.hl7.org/implement/standards/rim.cfm. [Accessed: 27-Jul-2012].

[37] "Integrating the Healthcare Enterprise (IHE)." [Online]. Available: http://www.ihe.net/. [Accessed: 27-Jul-2012].

[38] "PayPal Developer Network." [Online]. Available: https://www.x.com/. [Accessed: 27-Jul-2012].

[39] "OASIS Web Serices Standards." [Online]. Available: http://www.oasis-open.org/specs/. [Accessed: 27-Jul-2012].