Requirements Engineering for Emergent Application Software

Pericles Loucopoulos(✉)

Department of Informatics and Telematics, Harokopio University of Athens, Athens, Greece
p.loucopoulos@hua.gr

Abstract. The field of Requirements Engineering is arguably one of the most sensitive areas in the development of not only software but more importantly in the development of systems and organisational structures and processes supported by such systems. As service systems play an increasingly important role in today's economy, the ability of software to respond to emergent real-world contexts becomes a key-enabling factor to developing new and unpredictable business models. This paper, which is partly based on the keynote lecture given by the author at ICEIS 2012, considers the field of emerged enterprise application software and critically examines the applicability of the methodology factors underpinning much of the practice in Requirements Engineering, to such systems.

Keywords: Requirements engineering · Emergent software · Service systems · Teleological methodology

1 Introduction

The last decade has witnessed the rising significance of services in world economies [1] to the extent that in some countries services account for over 65 % of their GDP. According to Eurostat "an essential source of innovation, knowledge-intensive services (KIS) and even more so, high-tech KIS, are often seen as major engines of growth in modern economies." In particular, "the value added by the EU's high-tech KIS sector increased at an average annual growth rate of 6.6 % between 2000 and 2005". Although high-tech KIS comprise more than just software, it is fundamentally software that provides the necessary added value to these systems. Therefore, the effective and efficient development of such systems can have a major impact on the economic value of KIS.

The aim for this paper is to examine the role of Requirements Engineering (RE) in contemporary software development in the context of *emergent application software*. Emergent application software is regarded as a key component in the service industry of tomorrow [2]. Some key challenges outlined in [2] include:

J. Cordeiro et al. (Eds.): ICEIS 2012, LNBIP 141, pp. 18–28, 2013.
DOI: 10.1007/978-3-642-40654-6_2,

- Providing dynamically adaptable services for new innovative business models from different service providers.
- Adaptable services to be compliant with Service Level Agreements (SLAs). In general dealing with non-functional requirements (NFRs) such as quality of service, security and privacy.
- Extending cloud computing to allow for semantically connecting software services across enterprise domains.
- Lining existing services efficiently towards new business processes.
- Dealing with evolvable and adaptable requirements and in general dealing with these at the user side.

In examining the role of RE in this changing landscape of software development and infrastructure systems it is worth looking at the contribution of RE to date. The past 40 years or so have witnessed the emergence of a variety of techniques for helping capture, represent, share, analyze, negotiate, and prioritize requirements. This is evidenced by the volume and impact of a plethora of requirements-related papers published in related journals and conferences [3]. At the same time the practical nature of RE has meant that some of this research has influenced the practice primarily in areas such as business and system modeling [4]. At the same time we have witnessed a number of initiatives that attempt to support the RE lifecycle, initiatives such as risk-driven methodologies [5], requirements tracing [6, 7], model re-use [8], use of scenarios [9], use of visualization [10], use of business rules [11, 12], enterprise modeling [13] and goal modeling [14, 15] to name but a few.

A recent study of practices [16] and workshops involving both academics and practitioners, the results of which were reported in [17, 18], revealed four challenges: (i) the intertwining of requirements and contexts, (ii) the continuing evolution of requirements, (iii) the influence of architectural considerations and (iv) coping with complexity. These challenges apply across the spectrum of application domains as well as across different development paradigms.

There is a realization that, as Software Engineering (SE) capabilities have grown, so have aspirations and expectations for future systems and these largely outstrip capabilities. Brooks' [19] statement more than 30 years ago, that "the complexity of software is an essential property, not an accident one" remains true to this day [20]. It could be argued that there is a whole set of new advanced design requirements [17] that demand now more than ever close attention to the effectiveness of the SE process. The field's focus and scope has shifted from engineering of individual systems and components towards the generation, adaptation and maintenance of software-intensive ecosystems consisting of software, hardware, human and organizational agents, business processes and more. Such software ecosystems require attention in three areas namely those of design and evolution, orchestration and control and monitoring and assessment [21].

This paper examines the traditional approach to RE and argues that the prescriptive way of using maps of the development process is not valid for software that needs to exhibit emergent properties (Sect. 2). The need to be adopt a user-driven approach for deliberating requirements for emergent software is considered in Sect. 3. Section 4 argues that an intention-driven approach provides the necessary abstraction and

flexibility that is required during the development and evolution of emergent software. The paper concludes in Sect. 5 with a reflection on different modeling dimensions that would support a development paradigm for emergent application software.

2 RE in the Context of Emergent System Development

The demand for emergent application software is due to the realization that software needs to adapt itself dynamically to support the ever-changing requirements of markets and enterprises. Enterprises making use of such software would explore this type of software to develop new business models that were not originally planned. Emergent application software is characterized by the combination of service components from a variety of providers in a dynamic and flexible manner. Let us examine the role of RE in attempting to design such systems. Is the traditional approach of using "maps of designs" (i.e. methods that follow a prescribed route from problem to solution) appropriate?

Developing emergent software falls in the category of design problems often referred to as ill-defined problems, or as Rittel and Webber called them "wicked problems" [22]. They demand that designers not only keep in pace with the technology's potential, but that they take business goals, social and human factors into account as well. In such a context, problem setting is no more a trivial activity. On the contrary, problem setting is, in a large part, intertwined with problem solving [23]. In ill-defined problems, defining the problem amounts to a large extent to solving it. Ill-defined problems are usually hard to solve because their definition, and not only their solution, is difficult. Usually there is no single definition, and subsequently no any single solution to be found. Ill-defined problems cannot be *optimized*; they can only be *satisficed*.

This point was argued by Fred Brooks in his talk at the workshop in "Design Requirements" [17], as follows: "*A point I want to emphasize in the requirements process is we do not usually know what the goal is. I will assert that this is a deep fact of reality that is ignored in much of the literature about requirements. We do not know what we are trying to build. The hardest part of most designers of complex systems is not knowing how to design it, but what it is you are trying to design. When we talk about eliciting requirements, we are talking about deciding what it is we are trying to design*". In other words the development tree emerges as we progress through designing and the development tree is not about decisions but about designs [20].

The traditional approach to designing is that of 'design by drawing'. In other words, an artifact is represented in terms of expressions of mental models. For example, a design drawing in the case of the architect may be the design of a building; or in the case of an information systems designer, the collection of conceptual schemata representing the form of the information system. This is an approach that concentrates entirely on the artifact itself. It attempts to visualize the end product prior to its implementation. However, a major disadvantage with this approach is that it ignores completely those issues that cannot be visualized, for example, social issues, issues of value conflict, etc. It is only by making the process open to inspection and critical evaluation that it is possible to model and therefore reason about such issues.

The expressions of mental models (the product of the design process) concerning the artifact are normally considered within a framework that *maps* the design process. These maps, expressed as different design activities, are advocated by method engineers as a way of planning the designer's work. On examination of these maps, it becomes obvious that rather than considering a description of the process, the maps are really a description of the products. A design map, let us take as an example the steps in a structured method, tells us what the designer should consider (e.g., data flows, stores, modules etc.) and what they should produce (e.g., a set of hierarchically organized structure charts) rather than how a designer actually works.

Design maps tend to be general and prescriptive and they are an attempt of method engineers to provide some ordering in the actions of the designer. In fact there is little evidence that design maps bear any resemblance to the way designers work. Although the field of Information Systems is rather poor in empirical work in this area, there is fortunately a considerable wealth of material from other disciplines that demonstrate that maps fail because problem analysis and problem solution are more closely knitted than usually thought, and this is what makes a problem ill-defined.

For example, Lawson [24] observed the design processes followed by different groups of people concluding that there was no generic pattern that was followed and much was dependent on the culture and background of the different groups. Similarly, Eastman [25] recorded the workings of experienced designers. The protocols revealed how the designers explored the problem as well as their attempts to a solution concluding that many requirements were not thought out in the abstract nor were they defined in advance of searching for solutions but rather they were discovered as a result of experimenting with different solutions.

One of their characteristics of ill-defined problems is that they have no definite formulation. Formulating them amounts, to a great deal, to solving them.

What are the implications of this? Mainly a realization that designing does not proceed in a well founded route from problem setting to problem solving but there is a continuous interaction between the two. This can be seen in a more appropriate model of the design process, that of 'generator-conjecture- analysis' [26] which postulates that designers first define what might be an important aspect of the problem, then they develop a tentative design on the basis of this and they subsequently examine it to see what else can be discovered about the problem. Analysis guides design and design guides analysis -and all in an effort to gain an understanding of the problem, of the situation at hand. In this 'trade off" situation, models and modelling play a crucial role. Models are not just outputs of the process but also inputs to the thought process.

Whilst the discussion thus far has focused on the process of development one has to be cognizant of the changes that are taking part partly driven by technology and partly by business and economic factors. Table 1 compares and contrasts some of the key differences between traditional and contemporary development settings.

Traditionally development of software-intensive systems assumed a reasonably stable business environment. Rapid market changes such as electronic commerce, deregulation, mergers, globalisation and increased competition have led to a business environment that is constantly and rapidly evolving.

Table 1. Traditional vs. Contemporary Development.

Traditional	Contemporary
The context is a reasonably stable system ecology	The context is a rapidly changing system ecology
Emphasis on business process improvement via IS	Emphasis on enterprise and market transformation via IS
A key issue is that of alignment	A key issue is that of innovation
System properties predictable	Emergent system properties
Development based on a decision paradigm	Development based on a design paradigm
Clear separation between system and user	The human is no longer outside the system but an integral part of it

This leads to the demands for exploiting information systems technologies not just for supporting business processes but also and perhaps more crucially in enabling organisations to transform their businesses.

This implies that nowadays there is less emphasis in carefully aligning a system to the business and instead seeking to innovate through the use of technology.

Whilst the properties of systems were traditionally predictable, there is a realization that systems need to adapt to their environment. This desire for systems with emergent properties raises issues with respect to requirements evolution, requirements tracing and requirements quality.

This uncertainty at the outset leads one to conclude that the decision-oriented approach i.e. one that has a pre-determined set of development actions is not applicable and a more design-oriented approach is required, an approach that affords, experimentation and re-work.

Finally, the user is nowadays an integral part of a system and is no longer a mere observer. This in turn raises demands for user-driven paradigms that enable users to define, design and review potential solutions.

3 RE for Emergent Software Services

As introduced already in Sect. 2, emergent application software is characterized by the combination of service components from a variety of providers thus exploiting Software as a Service (SaaS) as a software development and delivery approach. The salient feature of the SaaS approach is its flexibility, as it enables organizations to create new software applications dynamically to meet rapidly changing business needs. The SaaS approach has become a common software delivery model for many business applications, including accounting, customer relationship management (CRM), enterprise resource planning (ERP), invoicing, human resource management (HRM), content management (CM) and service desk management. The Open Group [27, 28], which is behind the development of Software Oriented Architecture (SOA) Reference model, defines SOA as “an architectural style that supports service

orientation. Service orientation is a way of thinking in terms of services and service-based development and the outcomes of services".

With the advent of cloud computing, the SaaS approach has become part of the nomenclature of the cloud stack, along with IaaS (Infrastructure as a Service) and PaaS (Platform as a Service). Armbrust et al. [28] argue that what is really new in cloud computing is utility computing as a service, as cloud computing is the sum of SaaS and utility computing. These authors eschew the terms such as IaaS (Infrastructure as a Service) and PaaS (Platform as a Service) because they argue that accepted definitions for these terms still vary widely and the line between "low-level" infrastructure and a "higher-level" platform is not crisp. They believe the two are more alike than different, and hence consider them together under the general term of "utility computing".

This viewpoint suggests that the success of cloud computing depends on close collaboration between these users at all levels of the cloud continuum. The development of cloud computing should therefore have strong user engagement and involvement. Typical cloud users are SaaS providers, who, on one hand bring the benefits of cloud computing to end users – SaaS users, and on the other hand, drive the development of cloud applications through solving the end user problems. SaaS applications should therefore be able to scale seamlessly at all levels of the cloud continuum, from higher-levels that support the SaaS application development, all the way to the cloud platforms.

RE can play a very significant and central role in bridging the gap between technology and users by extending the abstraction level of SaaS towards the user end, which is conceptually depicted in Fig. 1.

A key factor in the scheme outlined in Fig. 1 is that of the "application independent model". These models would augment the typical way of SaaS working. SaaS focuses on delivering software on demand by enabling the composition of service applications, which is delivered at the point of need, executed and then discarded. Current efforts concentrate on the underlying technology and pay less attention to the ways that users may be engaged. Users may get engaged if the 'language' and the method deployed are closer to their perception. This would then enable users to develop and modify software without necessarily being information technology professionals. The application independent models would be archetypal models that would fall into three categories:

- *Component Models*. This type of model supports the development of component services. Examples are requirements models [29] and design models [30]. Component services are fine-grained service components that can be reused for composing different types of services. Component developers can use these models to match and select suitable component services offered by cloud providers or use them as a baseline for designing new component services.
- *Service Models.* This type of model supports the development of services. Examples are service descriptions, business rules [31], and quality of service specifications [32]. Services are higher-level, business-oriented components which are composed from one or more service components. Service developers can use these models to

match and select suitable services offered by cloud providers or use them as a baseline for designing new services.
- *Business Models.* This type of model supports the development of service systems or ecosystems as shown. Examples are business process models [33], service interaction and orchestration models. Service systems are business applications composed from one or more services. Developers can use these models to compose service systems on demand or use them to integrate service systems into an ecosystem.

There are many advantages in deploying the scheme shown in Fig. 1. Specifically:

- Services would be closer to user needs.
- Applications would be more adaptable to user needs.
- Implementation would be independent of any vendor.
- Implementation would be independent of any platform.
- Adaptivity would be easier for vendors.

Application independent models would represent a knowledge base to support the RE process, exploiting knowledge about specific domains [34, 35], developing and maintaining archetypical, generic solutions [36, 37], as well as assisting in the process of progressing from fuzzy expressions of requirements to formal specifications [38, 39].

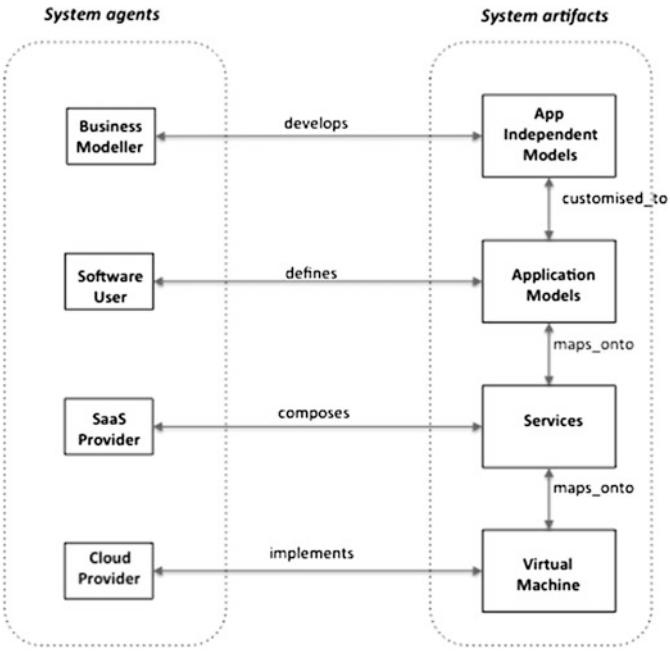

Fig. 1. Extending the level of abstraction towards the user-end.

4 Intention-Driven RE for Emergent Software Services

Design actions are cognitive actions, and are mirrored in the conceptual models employed by the designers. An important way of working in developing, managing and exploiting these conceptual models (for example, in Fig. 1 customizing the application independent models, to application models), is that of the teleological approach [15, 40]. The term teleological being used to convey the fact that *whatever* we do, we do to identify the *means* by which we shall be able to fulfill the *ends* stated (by means we refer to the lower level goals whose achievement is necessary -and, we hope, enough- for the high level ends). When we substitute a goal for a requirement, this goal is no more only a goal for the enterprise, the project, or whatever we might be working for; *it becomes a goal for us as well* (i.e. all those involved in the 'change' process).

The teleological approach advocates that everything done is incited by some goal. It is *not* a prescriptive way-of-working and in this sense it avoids the problems discussed in Sect. 2.

The task of someone using enterprise knowledge modelling is to determine the means by which an ultimate goal, let us call it G_0, will be realized. In attempting to achieve this, the process is governed by causal relationships between goals in a network of goals. That is, a directed edge from a goal G_i to another goal G_j implies that the achievement of G_i depends on the achievement of G_j. At every step of the process, the process is controlled or driven by the goal in hand.

The actions chosen for attaining the goals represent *working hypotheses*. As goals and sub-goals are established, these are tentative at least until they are tested about their fitness of purpose, i.e. the satisfiability of higher goals. This observation has its roots in the Principle of Rationality [41] which states that "if an agent has knowledge that one of its actions will lead to one of its goals then the agent will select that action". However, an agent may not have complete knowledge to make the appropriate action, or the set of actions may be so complex that it is impossible to determine the single correct set of actions that will lead to the achievement of the goal. In this more general sense, the Principle of Bounded Rationality [42] which states that "given a goal, an agent may not possess perfect or complete knowledge of, or be able to economically compute or access, the correct action (or sequence of actions) that will lead to the attainment of the goal" will apply. Therefore, any actions chosen by an agent in order to achieve a given goal will, in general, be a *hypothesis* that the actions will lead to the achievement of the goal.

We argue therefore that in developing emergent software systems, which cannot be prescribed at the outset, the teleological approach offers many advantages over other traditional techniques. The approach comprises both goals conceived as requirements and goals conceived as instances of personal efforts without making any assumptions about the sequence with which such goals are produced.

Using the teleological approach one is engaged into designing which itself constitutes the establishment of a series of causal relationships between goals. The result of this analysis could be visualised as goal graphs. These graphs therefore, would represent, a posteori, the design decisions showing successively refined situations that

ultimately lead to the originally stated goal being satisfied. The goal graph will also show alternative design paths. It follows that the goal graph would by itself be a model of the design process. Or more accurately, the decisions adopted during the design process [43]. In this way it would be possible to trace decision from code to requirements, to observe and analyse the rationale used for certain design decisions and to achieve fast and effective changes to the software knowing the effects of these changes to the entire problem–solution frame.

5 Conclusions

This paper has put forward the position that the new class of emergent application software systems requires a user-driven paradigm, supported by a multi-perspective modelling approach. Such an approach would provide a convenient way of examining the different aspects that impact on the development of such systems. The three dimensions are those of: *product representation*, *change process* and *deliberation*.

The product representation dimension refers to the models that may be developed, browsed, manipulated etc. during the change process. The change process dimension refers to the steps, and activities made during the transition from a current situation to a future one. Finally the deliberation dimension refers to the participants, the arguments used, the choices made and the issues resolved during the transition process. The three dimensions are orthogonal in the sense that each aspect of one has a reflection on each one of the other two. For example, if we are engaged in discovering the existing situation (an aspect of change process) we may develop a business process model (an aspect of the product representation) and in doing so a number of stakeholders may engage into discussion and agreement (an aspect of the deliberation).

The task of emergent application software development should be viewed as a co-operative activity that exploits the contribution of different modelling views, each encompassing a specific type of knowledge. Within this multi-perspective approach enterprise analysis is based on two mechanisms: reasoning within a perspective; and reasoning across different perspectives in order to allow each individual step in the analysis process to exploit the most appropriate knowledge source.

References

1. Spohrer, J., Riecken, D.: Special section on service science. Comm. ACM **49**, 31–90 (2006)
2. ISTAG: Software Technolgies - The Missing Key Enabling Technology: Toward a Strategic Agenda for Software Technologies in Europe. ISTAG - Information Society Technologies Advisory Group, Brussels, Belgium (2012)
3. Cheng, B.H.C., Atlee, J.M.: Current and Future Research Directions in Requirements Engineering. In: Lyytinen, K., Loucopoulos, P., Mylopoulos, J., Robinson, B. (eds.) Design Requirements Engineering. LNBIP, vol. 14, pp. 11–43. Springer, Heidelberg (2009)
4. OMG (2003): UML Specifications. Object Management Group (2003)
5. Boehm, B.W.: A spiral model of software development and enhancement. Computer **21**, 61–72 (1988)

6. Gotel, O.C.Z., Finkelstein, C.W.: An analysis of the requirements traceability problem. In: Proceedings of the First International Conference on Requirements Engineering, pp. 94–101 (1994)
7. Ramesh, B., Jarke, M.: Toward reference models for requirements traceability. IEEE Trans. Software Eng. **27**, 58–93 (2001)
8. Rolland, C., Loucopoulos, P., Grosz, G., Nurcan, S.: A framework for generic patterns dedicated to the management of change in the electricity supply industry. In: International DEXA Conference (1998)
9. Kavakli, E., Loucopoulos, P., Filippidou, D.: Using Scenarios to Systematically Support Goal-Directed Elaboration for Information Systems Requirements. In: IEEE Symposium and Workshop on Engineering of Computer-Based Systems, pp. 308–314. IEEE Computer Society (1996)
10. Lalioti, V., Loucopoulos, P.: Visualisation of Conceptual Specifications. Information Systems **19**, 291–309 (1994)
11. Loucopoulos, P., Katsouli, E.: Modelling business rules in an office environment. SIGOIS Bulletin **13**(2), 28–37 (1992)
12. Tsalgatidou, A., Loucopoulos, P.: An object-oriented rule-based approach to the dynamic modelling of information systems. In: International Conference on Dynamic Modelling of Information Systems (1990)
13. Loucopoulos, P.: From information modelling to enterprise modelling. In: Brinkkemper, S., Lindencrona, E., Solvberg, A. (eds.) Information Systems Engineering: State of the Art and Research Themes, pp. 67–78. Springer (2000)
14. Kavakli, E., Loucopoulos, P.: Experiences with goal-oriented modelling of organisational change. IEEE Trans. Syst. Man and Cybern. - Part C **36**, 221–235 (2006)
15. Yu, E.S.K., Mylopoulos, J.: Why Goal Oriented Requirements Engineering. Department of Computer Science, University of Toronto (1998)
16. Hansen, S., Berente, N., Lyytinen, K.: Requirements in the 21st Century: Current Practice and Emerging Trends. In: Lyytinen, K., Loucopoulos, P., Mylopoulos, J., Robinson, B. (eds.) Design Requirements Engineering. LNBIP, vol. 14, pp. 44–87. Springer, Heidelberg (2009)
17. Lyytinen, K., Loucopoulos, P., Mylopoulos, J., Robinson, B. (eds.): Design Requirements Engineering. LNBIP, vol. 14. Springer, Heidelberg (2009)
18. Jarke, M., Loucopoulos, P., Lyytinen, K., Mylopoulos, J., Robinson, W.: The brave new world of design requirements. Information Systems **36**, 992–1008 (2011)
19. Brooks, F.J.: No silver bullet: essence and accidents of software engineering. IEEE Computer **20**, 10–19 (1987)
20. Brooks, F.P.: The Design of Design: Essays from a Computer Scientist. Addison-Wesley, New York (2010)
21. SEI: Ultra Large Scale Systems: The Software Challenge of the Future. Software Engineering Institute (2006)
22. Rittel, H.W.J., Webber, M.: Dilemmas in a General Theory of Planning. In: Cross, N. (ed.) Developments in Design Methodology, pp. 135–144. Wiley, Chichester (1984)
23. Jarke, M., Loucopoulos, P., Lyytinen, K., Mylopoulos, J.: The Brave New World of Design Requirements: Four Key Principles. In: Pernici, B. (ed.) CAiSE 2010. LNCS, vol. 6051, pp. 470–482. Springer, Heidelberg (2010)
24. Lawson, B.: How Designers Think: The Design Process Demystified. Butterworth, Cambridge (1990)
25. Eastman, C.M.: On the Analysis of Intuitive Design Processes. In: Moore, G.T. (ed.) Emerging Methods in Environmental Design and Planning. M.I.T Press, Cambridge (1970)

26. Hillier, B., Musgrove, J., O'Sullivan, P.: Knowledge and design. In: Cross, N. (ed.) Developments in Design Methodology, pp. 245–264. Wiley, New York, (1984)
27. OPEN-GROUP: SOA Reference Architecture. The Open Group http://www.opengroup.org/soa/source-book/soa/soa.htm (2009)
28. Armbrust, M., Stoica, I., Zaharia, M., Fox, A., Griffith, R., Joseph, A.D., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A.: A view of cloud computing. Commun. ACM **53**, 50–58 (2010)
29. Hagge, L., Lappe, K.: Patterns for the RE process. In: Proceedings of the 12th IEEE International Conference on Requirements Engineering, pp. 90–99 (2004)
30. Rapanotti, L., Hall, J.G., Jackson, M., Nuseibeh, B.: Architecture-driven problem decomposition. In: Proceedings of the 12th IEEE Internal Requirements Engineering Conference, pp. 80–89. IEEE Computer Society, Washington (2004)
31. Loucopoulos, P., Wan-Kadir, W.M.N.: BROOD: business rules-driven object oriented design. J. Database Manag. **19**, 41–73 (2008)
32. Heidari, F., Loucopoulos, P., Kedad, Z.: A Quality-Oriented Business Process Meta-Model. In: Barjis, J., Eldabi, T., Gupta, A. (eds.) EOMAS 2011. LNBIP, vol. 88, pp. 85–99. Springer, Heidelberg (2011)
33. Grefen, P., Ludwig, H., Dan, A., Angelov, S.: An analysis of web services support for dynamic business process outsourcing. Inf. Softw. Technol. **48**, 1115–1134 (2006)
34. Loucopoulos, P., Champion, R.E.M.: Knowledge-based support for requirements engineering. Inf. Softw. Technol. **31**, 123–135 (1989)
35. Loucopoulos, P., Champion, R.E.M.: Concept acquisition and analysis in requirements specifications. Software Engineering Journal **5**, 116–124 (1990)
36. Prekas, N., Loucopoulos, P., Rolland, C., Grosz, G., Semmak, F., Brash, D.: Developing Patterns as a Mechanism for Assisting the Management of Knowledge in the Context of Conducting Organisational Change. In: Bench-Capon, T.J.M., Soda, G., Tjoa, A.M. (eds.) DEXA 1999. LNCS, vol. 1677, pp. 110–122. Springer, Heidelberg (1999)
37. Seruca, I., Loucopoulos, P.: Towards a systematic approach to the capture of patterns within a business domain. The Journal of Systems & Software **67**, 1–18 (2003)
38. Loucopoulos, P.: The F^3 (From Fuzzy to Formal) View on Requirements Engineering. Ingénierie des Systèmes d'Information **2**, 639–655 (1995)
39. Al Balushi, T.H., Sampaio, P.R.F., Dabhi, D., Loucopoulos, P.: ElicitO: A Quality Ontology-Guided NFR Elicitation Tool. In: Sawyer, P., Heymans, P. (eds.) REFSQ 2007. LNCS, vol. 4542, pp. 306–319. Springer, Heidelberg (2007)
40. Loucopoulos, P., Kavakli, E.: Enterprise Modelling and the Teleological Approach to Requirements Engineering. Int. J. Intell. Coop. Info. Syst. **4**, 45–79 (1995)
41. Newell, A.: The knowledge level. Artif. Intell. **18**, 87–127 (1982)
42. Simon, H.A.: Models of Bounded Rationality. MIT Press, Cambridge (1982)
43. Louridas, P., Loucopoulos, P.: A generic model for reflective design. ACM Trans. Software Eng. Methodol. **9**, 199–237 (2000)

http://www.springer.com/978-3-642-40653-9

Enterprise Information Systems
14th International Conference, ICEIS 2012, Wroclaw, Poland, June 28 - July 1, 2012, Revised Selected Papers
Cordeiro, J.; Maciaszek, L.A.; Filipe, J. (Eds.)
2013, XX, 490 p. 171 illus., Softcover
ISBN: 978-3-642-40653-9