Towards Requirements and Architecture Co-evolution

João Pimentel[1], Jaelson Castro[1], Emanuel Santos[1], and Anthony Finkelstein[2]

[1] Universidade Federal de Pernambuco - UFPE, Centro de Informática, Recife, Brazil
{jhcp,jbc,ebs}@cin.ufpe.br
[2] University College London - UCL, Department of Computer Science, United Kingdom
a.finkelstein@ucl.ac.uk

Abstract. The relationship between requirements and architectures is an important research field on software engineering. One of its challenges is to provide proper support for their co-evolution, i.e., how to assess the mutual impact of requirements and architecture changes on each other, as well as how to react to these changes in order to prevent misalignment between them. We advocate the use of a single goal model to express both requirements and architectural concerns. In this paper we put forward an approach for requirements and architecture co-evolution considering such a model. Moreover, we outline the reasoning required in order to support forward and backward co-evolution of service oriented systems.

Keywords: System architecture, Requirements engineering, Software evolution, Self-Adaptation, Service-oriented architectures, Autonomics.

1 Introduction

Software evolution has become a key research area in software engineering [10]. Software artifacts and systems are subject to many kinds of changes, which range from technical adjustments due to rapidly evolving technological platforms, to modifications in the software systems themselves required by the natural evolution of the businesses and requirements supported by them. These modifications include changes at all levels, from requirements through architecture and design, as well as source code, documentation and test suites. For consistent evolution, all models and artifacts should remain aligned as the software evolves.

For instance, whenever requirements change we need to assess whether the current architectural configuration continues to meet the stakeholders' requirements. Similarly, if the properties of components in an architectural model are modified we need to analyze if these changes affect requirements satisfaction. In both cases, when there is a mismatch between architecture and requirements, an architectural reconfiguration may be considered. This is particularly relevant in the case of services, since they are very dynamic and may change in several ways (functional upgrades, varying quality-of-service, withdrawn, so on and so forth). In contrast, when a traditional COTS component evolves, the system using it may continue to use an older version of that component. In the services case, evolution cannot be prevented.

Since the abstraction level of software architecture is adequate for identifying and analyzing the ramifications of changes [14], it could be one of the software evolution pillars. Certainly, it is of paramount importance to identify when and why to perform changes, as well as to assess their impacts [4]. Recent advances in the Requirements Engineering and Software Architecture fields include methods and techniques to address the evolution, in isolation, of requirements and of architectural models. However, there is a lack of proposals for tackling the co-evolution of requirements and architecture.

In fact, the line that separates requirements from architecture is a blurred one, as argued in [5]. The Twin Peaks model highlights the intertwined characteristics of requirements and architectural models [27]. Requirements lie in the problem space, whilst architectures are part of the solution space. Thus, investigating how to define an architecture (solution) that satisfies the requirements (problem) is a key challenge in software engineering. Moreover, it is important to maintain this satisfaction throughout a system lifecycle [7].

In this paper we present a novel approach for dealing with requirements and architecture co-evolution. We define the co-evolution problem as the problem of assessing the impact of both requirements and architectural changes and responding to these changes.

The remainder of this paper is structured as follows. In Section 2 we present the case study used throughout the paper. In section 3 we describe the approach itself. Section 4 discusses related works. Lastly, Section 5 concludes the paper with a critical discussion of our proposed approach and indicates points for improvement.

2 Case Study

In this work we are expressing requirements using the *i** Framework [39]. It defines goal-based models to describe both the system and its environment in terms of intentional dependencies among strategic actors. The actors are refined using four kinds of elements: goal, softgoal, task and resource. Goals represent the actors' intentions, needs or objectives to fulfill its role within the environment in which they operate. Softgoals also represent the strategic interests of the actors, but in this case these interests are of subjective nature – it is generally used to express non-functional requirements. The tasks represent a way to perform some activity, i.e., they show how to perform some action to obtain the satisfaction of a goal or softgoal. The resources represent data, information or a physical resource that an actor may provide or receive. These elements are linked together within the actor boundaries using means-end, task-decomposition, and contribution links. The means-end links define which alternative tasks (means) may be performed in order to achieve a given goal (end). The task-decomposition links describe what should be done to perform a certain task (i.e., its sub-tasks). Finally, the contribution links suggest how a task can contribute (positively or negatively) to satisfy a softgoal. These contributions allow the selection of alternative tasks driven by the satisfaction of softgoals.

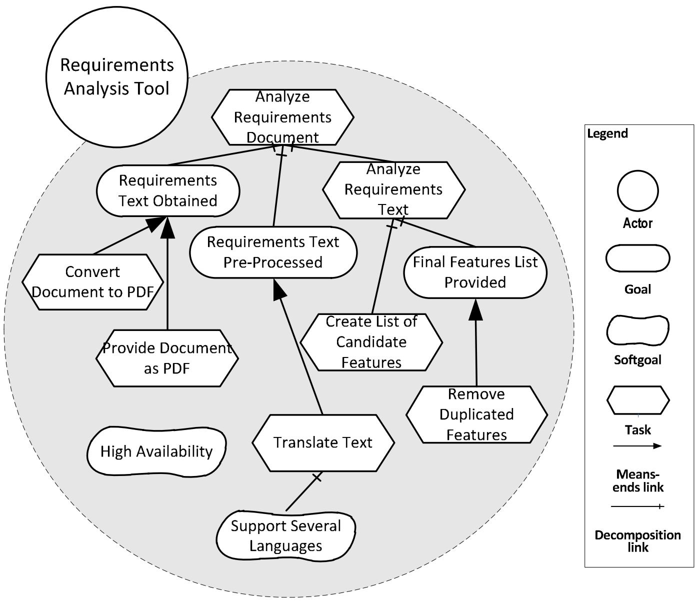
Requirements Analysis Tool
Analyze Requirements Document
Requirements Text Obtained
Analyze Requirements Text
Requirements Text Pre-Processed
Convert Document to PDF
Final Features List Provided
Create List of Candidate Features
Provide Document as PDF
Remove Duplicated Features
High Availability
Translate Text
Support Several Languages
Legend
Actor
Goal
Softgoal
Task
Means-ends link
Decomposition link

Fig. 1. The requirements model for the Requirements Analysis Tool

Fig. 1 presents the original requirements model of our system, which is a Requirements Analysis Tool. It is a web-based system that analyzes a textual requirements document and generates a list of candidate features. Thus, the main task of this system is to analyze a requirements document. In order to do so, it will need to obtain a requirements document, which will be provided by a user (here we are omitting dependency links to the system users). The user can either provide the document as a PDF file, or provide it in any usual file format (such as word processing documents and spreadsheets), which will be converted to PDF for our processing.

A common constraint on natural language text analysis is that it is highly dependent on the language being used. In order to enable the analysis of requirements documents in a wide range of languages, we decided to incorporate the functionality of translating the document to a reference language (the alternative would be to adapt the analysis algorithm for each language that we want to support). In order to reach a large user base worldwide, we defined that this translation must support several languages. The requirements analysis itself consists of creating a list of candidate features, and finally providing a consolidated list by removing duplicated features. Lastly, the *High Availability* non-functional requirement (softgoal) is important for our system, since it will be accessed anywhere, any time of the day. Please note that we have not yet defined how to satisfy this softgoal, since we have not taken any architectural decision yet. Alternatively, we could have already modeled all the different ways of satisfying this requirement – later we would only select which ones to use.

In Fig. 2 we present a possible structural architecture for the Requirements Analysis tool. In this architecture we rely on two kinds of services: Document Converter, which are services that provide file type conversion of documents; and Text Translator services, which are able to translate a given text. On the service consumer side, the client-server style was selected because it is well suited for web-based systems.

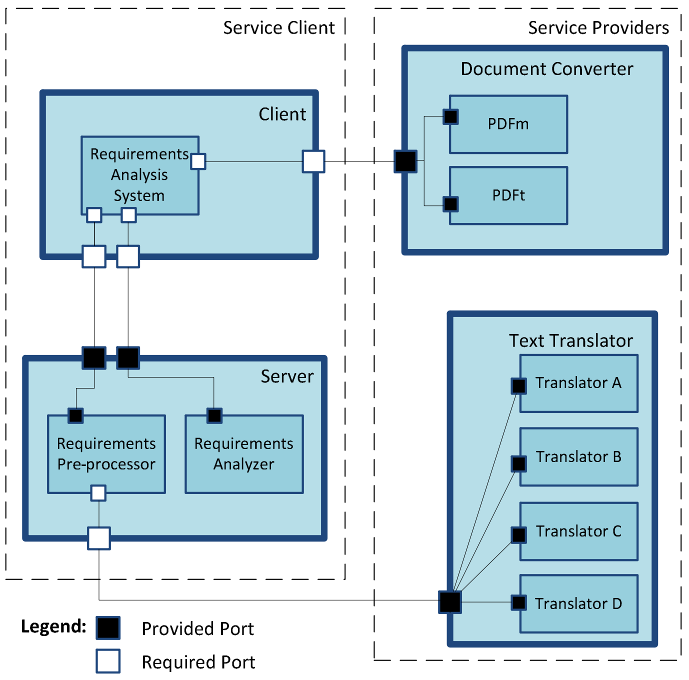

Fig. 2. Architectural model for the Requirements Analysis Tool, in Acme

3 Forward and Backward Evolution of Service-Oriented Systems

When dealing with requirements and architecture co-evolution two situations may arise. On one hand, changes in the system requirements may happen (this includes the system context, stakeholders' attitudes and quality constraints). In this case, some analysis is required to assess if these changes call for a reconfiguration, and whether there is some reconfiguration that satisfies the new requirements. We call this forward evolution, since it is from requirements to architecture.

On the other hand, there may also be changes in the architecture itself. For instance, the performance of a component may degrade. Thus, it is now required to check whether these architectural changes prevent requirements satisfaction. If this is the case, and this failure is unacceptable, it is necessary to attempt to identify a possible architectural reconfiguration that improves the level of the satisfaction of the requirements. However, if it is not feasible to reconfigure it, the system administrator could be prompted to either relax the affected requirements, or to perform offline evolution. This we call backward evolution – from architecture to requirements.

We propose to tackle the co-evolution problem by converging requirements and architecture models, i.e., working with architectural models that also contain requirements information. By doing so, we are able to perform the co-evolution reasoning in a single model. Moreover, this reduces the overhead of maintaining traceability between requirements models and architecture models.

In order to do so, we use a conventional requirements modeling notation to represent architectures – namely, *i**. This was preferred over modifying an Architectural Description Language (ADL) because *(i)* we did not find an architectural language expressive enough for presenting requirements; *(ii)* by using the same framework for both RE and architecture we can have a seamless approach to go from requirements to architecture; and *(iii) i** showed to be a suitable notation for expressing architectures.

Despite being an organizational modeling notation, *i** has shown to be particularly adequate for requirements modeling [39]. Recent works also showed that it is reasonably suitable for architectural modeling [16] [30]. More specifically, it has been used to model information services [26]. In [16] there are arguments in favor of using *i** extended models for architectural modeling. It is claimed that it can be used to describe main architectural concepts, such as components, connectors, constraints, non-functional properties and evolution. Moreover, *i** has suitable composition, abstraction and analysis mechanisms. However, it lacks proper support to promote reusability and heterogeneity, as well as it lacks proper support for configuring the models.

It is claimed that software architecture describes a system in a high-abstraction level, defining its components, the interaction among these components, their attributes and their functionalities [37]. Fig. 3 presents our approach for expressing service-oriented architectures using *i**, in the context of our case study. Here, conventional components were mapped onto *i** actors, service categories onto roles and the services themselves onto agents. A service category is a general definition of the service that is required, while an agent is a specific service that plays the role defined by a service category. E.g., Text Translator is a service category, whilst Microsoft Translator is a particular service of that category. With this mapping we are able to express the requirements related to each component of the architecture. Lastly, connectors are represented by dependencies. This allows expressing what is expected from a component (*dependum*), why is it expected (from the *depender*'s model) and how is it going to be provided (from the *dependee*'s model).

In Table 1 we present a summary of this mapping, considering the five major architectural elements [38]. However, note that the rationale, i.e., the information that explains the architectural decisions taken, cannot be properly captured by *i** elements. This is also the case for the majority of architectural modeling notations, where other artifacts are required to document the architectural diagrams [8].

Table 1. Mapping of architectural elements onto *i**

Architectural Element	*i** Element
Component	Actor, role, agent
Connector	Dependency links
Interface	Implicitly defined by the source and target elements of dependency links
Configuration	The graph itself
Rationale	Partially defined by internal elements (goals, softgoals, tasks, resources and their relationship)

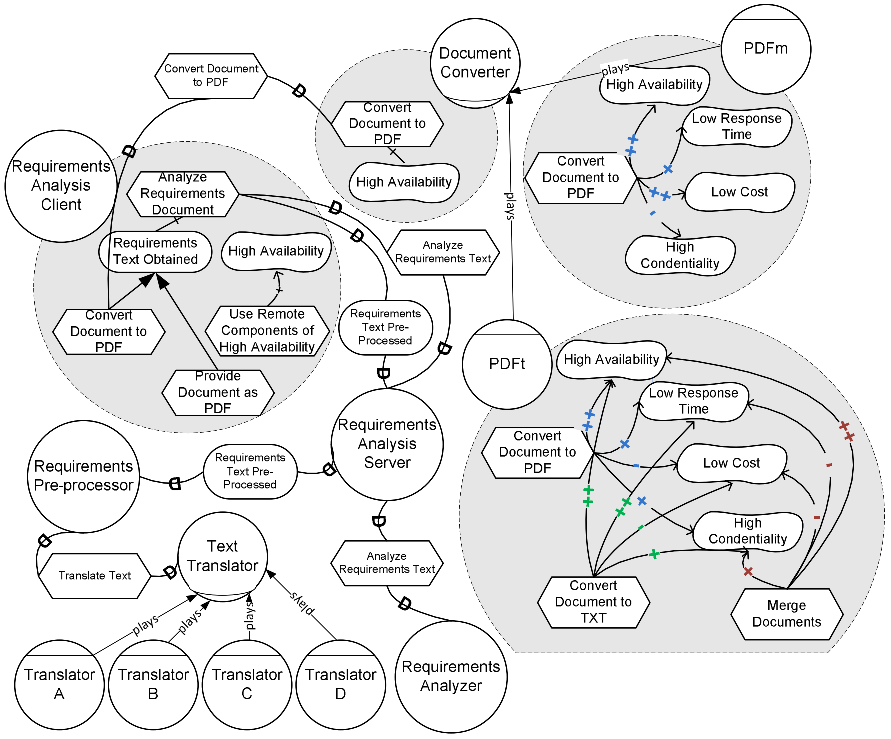

Fig. 3. Architectural model of the Requirements Analysis tool, using *i**. It replaces the former model presented in Fig. 2.

Another motivation for the use of *i** as an architectural description language is the current set of available reasoning mechanisms. Particularly relevant to our approach is the evaluation of the softgoals' satisfiability [15][19]. This allows selecting the best alternative to achieve a goal, considering the contributions of each alternative to the softgoals of interest (top-down reasoning). We can also assess whether a given alternative properly satisfies the selected softgoals.

3.1 Forward Evolution

Our concern here is to handle requirements changes related to the information services being used by the proposed system. These may come in two ways: a functional change, i.e., we want the service to satisfy a different goal or task; or a non-functional change, i.e., we define different quality constraints on how the service is supposed to support its goals or tasks.

When there are requirements changes, we are capable of checking whether the information service currently selected can satisfy the new requirements. For instance, consider that we are interested in the *Document Converter* service category (as described in Fig. 3) and that the *PDFm* service is currently selected. Several queries can be performed:

Query 1: Can PDFm support Convert Document to PDF with high Availability?
According to Fig 3. the service is highly available. However, if after deployment we notice that requirements documents are sometimes split in several documents according to some criteria (such as by sub-system, by viewpoints, and so on), we may decide that we also need the capability of merging documents. Thus, we can now pose a new question to check if *PDFm* provides this functionality as well, which is expressed in Query 2.

Query 2: Can PDFm support Convert Document to PDF and Merge Documents?
Since *PDFm* is unable to perform the *Merge Documents* task (Fig. 3), the answer to Query 2 is negative. Thus, we may ask the same question to other services of the same category. If one is found (eg. *PDFt*), we could then perform the required architectural reconfiguration, i.e., use *PDFt* instead of *PDFm*.

The same reasoning presented so far can be performed with softgoals as well. For instance, we may decide to go only for PDF conversion, as long as it is performed at low cost. In order to check whether *PDFm* satisfy this new requirement, we may ask Query 3:

Query 3: Can PDFm support Convert Document to PDF with low cost?
The assessment of softgoal satisfaction is trivial in the *PDFm* model (Fig. 3): since there is only one contribution link towards *Low Cost*, and it is a ++ contribution. Hence, the *Low Cost* softgoal is satisfied. For more complex cases, with different contribution links, one may refer to [15][19].

3.2 Backward Evolution

On the other hand, when there is a change in properties of the information services, or when new candidate services are identified, a similar reasoning may follow. In this case, we may use a monitoring framework in order to retrieve updated information on the services' properties. For instance, the SALMon tool [28] is able to provide up-to-date data on web services' response time and availability, among others. With such monitoring capabilities, we are able to assess both at design time and at runtime the quality of the information services being used.

In our case study, consider that we require documents to be converted to PDF with a high availability service. Recall that before deployment we certified that the *PDFm* service satisfied this query; for this reason, we had selected it for use in our system. Nonetheless, after deployment we may have noticed a degradation of its availability. Thus, we need to check whether this service is still able to meet our requirements – i.e., we need to check if there is a possible solution for Query 1. I.e., the same query would be performed, now with the model updated for the new availability value. Provided that automated monitoring is available, this reasoning can be completely performed without human intervention. Hence, it is suitable for adaptive and autonomic systems, which could perform this checking at regular time intervals. Foresight methods may be used to define which requirements/architectural elements to monitor and at what time intervals [33].

If the experienced change prevents the information service from satisfying its related requirements, we may check if other services of the same service category are able to meet the requirements – in this case, *PDFt*.

3.3 Tolerance, Relaxation and Manual Evolution

On the last two sub-sections we outlined how we can reason to identify a mismatch between system requirements and information services. Moreover, we showed how we can attempt to solve this mismatch by searching for a possible reconfiguration. There are two questions that arise when performing this reasoning: (i) all mismatches must be solved or can we live with some mismatches? (ii) What happens when no reconfiguration is able to solve this mismatch?

In previous works we argued that not every failure requires compensation [32], acknowledging that distinct failures may have different impacts. In our specific case, we could rephrase it: not every mismatch between service and requirements (failure to satisfy requirements) demands a reconfiguration (compensation). We tackle this issue by allowing system administrators to define tolerance policies, using our previous framework [32]. Thus, the system administrator will be able to define different criteria to assess when a failure in satisfying requirements, resulting from architectural changes, needs to trigger a reconfiguration.

A main element in that framework is the tolerance policy, which consists of tolerance rules. These rules may be related to the system context as well as to a particular element of the goal model, or to the amount of failures that happened. With this framework we may decide to ignore when a service fails to support a given element of the architectural model (e.g., a quality constraint), in particular conditions. Hence, instead of searching for a possible reconfiguration, we will continue to use the same service.

Regarding the second question, if it is not possible to reconfigure to satisfy the evolved requirements, and assuming that the tolerance policy in place does not allow for that failure to be ignored, we envision two scenarios. On one hand, the system administrator will be prompted to adjust (relax) the current requirements so that there is at least one possible reconfiguration. Alternatively, manual (offline) evolution of the system may take place.

4 Related Works

The area of Software evolution has been largely studied. More recently, terms such as autonomics, self-adaptation and self-management have been used to describe systems that are able to dynamically evolve at runtime. Regarding requirements evolution, some approaches (such as Lapouchnian and Mylopoulos [23] and Ali et al. [2]) use the notion of context in order to identify which elements of the requirements model are active/enabled. Pimentel et al. builds on that to derive architectures that support requirements activation/deactivation [31]. Jian et al. [21] proposes mechanisms to allow the insertion of goals in the requirements model at runtime. The system is only capable to satisfy these new requirements by developing new modules for the system. Qureshi et al. [35] also allows the changing of goal models at runtime. It proposes a service lookup mechanism to identify services that may satisfy the new requirements. Franch et al. [13] define metrics related to non-functional requirements. In turn, the metrics are linked to service categories and services. Thus, its reconfiguration is based solely on the measurements of these metrics.

Similarly, there are several research works regarding architectural evolution. For instance, [9] defines adaptation conditions based on architectural properties as well as reconfiguration operations. Control events based on components' states are used in [3] to reconfigure the architecture connectors. Composition rules are deployed in [34] to dynamically define connections between components and aspects. Some of previous work also allowed the addition, removal, change and reconfiguration of components [32]. These works may have broader and more sophisticated mechanisms for architecture evolution than ours. However, they fail to relate this evolution to system requirements.

There are also works on the requirements and architecture relationship such as [12][17][20][22]. However, they do not tackle this problem as we do, i.e. by considering the architecture model as a refinement of the requirements model, along the lines of what was developed for problem frames in [17].

Pahl et al. [29] proposes to dynamically define service collaboration through a coordination space, on which a service consumer expresses its need for a particular kind of service, which may be satisfied by a service provider. However, it does not consider the other elements of the software architecture.

5 Discussion

Considering the architecture as a reification of the system under consideration, and the increasing adoption of technologies that facilitate architectural changes (such as the technologies behind web services and cloud computing), it is of utmost importance to understand and reason on the relationships between requirements and architectural models. This calls for systems that are able to react to changes in requirements (i.e. according to the stakeholders expectations), as well as dealing with changes in the system itself (architecture). Architectural changes include structural changes – e.g., replacing a component (due to a new update) – and properties changes – e.g., the performance of a component may have degraded.

Throughout this paper we outlined our approach for requirements and architecture co-evolution. The main contribution of this approach is that it provides proper reasoning to handle the reciprocal impact between requirements and architecture – i.e., the requirements and architecture co-evolution. In the particular case of information services we are able to assess the impact of such changes, as well as to identify whether and which reconfiguration is possible to react to a given change. Given that proper monitoring tools are set up, this reasoning can be used at runtime to enable autonomic and self-adaptive behaviors.

In order to provide such reasoning, we advocate the use of architectural models enriched with requirements data. Such model may be derived from requirements models through a series of decision/transformations steps (e.g., [6]). In this research we propose the use of *i** for both requirements and architecture modeling [16][30]. This approach has some drawbacks, as follows:

Lack of familiarity – software architectures are already accustomed to conventional ADL. Thus, the need to learn a new notation would be a barrier for the adoption of this approach.

Poor readability – architectural models may become more difficult to handle in our approach due to the additional requirements information.

Lack of tools – there are several tools to support conventional ADL – e.g., for automatic code generation. The lack of similar tools to support *i** may prevent some architects to adopt it.

The first two drawbacks may be mitigated by using the *i** information hiding mechanism, by improving the *i** visual syntax [25] and by using modularization mechanisms [1][11]. The third problem may be softened by developing new tools for *i**, or by translating the *i** models to a conventional ADL as described in [6][24].

We believe that our approach is suited not only to service-driven architectures, but also for any kind of architecture on which components have some degree of intentionality. This is the case for socio-technical systems, on which some responsibilities are delegated not only to software and hardware components, but also to organizations and human participants. This is also the case for agent-based systems, on which each agent has its own goals, that may or may not converge to the overall system goals.

A key limitation of our approach is that we only consider the structural view of the architecture. Thus, an important advance in future works would be to include other views [36], as well as behavioral concerns. It is also important to notice that we intend to support only the derivation of architectural models – detailed design, class diagrams, code, and so on, are currently out of the scope of our approach. Thus, we do not define some service details, such as protocols, publishing mechanisms, and so on.

A major improvement for our approach would be to use Artificial Intelligence (AI) mechanisms in order to enhance the reasoning here proposed – for instance, simulation techniques [18]. This would be an important step towards Intelligent Software Engineering, i.e., Software Engineering that makes use of AI techniques.

Acknowledgments. This work was partially supported by Brazilian agencies CAPES and CNPq.

References

1. Alencar, F., Castro, J., Lucena, M., Santos, E., Silva, C., Araújo, J., Moreira, A.: Towards modular *i** models. In: 25th ACM Symposium on Applied Computing, pp. 292–297 (2010)
2. Ali, R., Dalpiaz, F., Giorgini, P.: A Goal Modeling Framework for Self-Contextualizable Software. In: Halpin, T., Krogstie, J., Nurcan, S., Proper, E., Schmidt, R., Soffer, P., Ukor, R. (eds.) BMMDS 2009 and EMMSAD 2009. LNBIP, vol. 29, pp. 326–338. Springer, Heidelberg (2009)

3. Allen, R., Douence, R., Garlan, D.: Specifying and Analyzing Dynamic Software Architectures. In: Astesiano, E. (ed.) ETAPS 1998 and FASE 1998. LNCS, vol. 1382, pp. 21–37. Springer, Heidelberg (1998)
4. Andersson, J.: Issues in Dynamic Software Architectures (2000)
5. Boer, R., Vliet, H.: On the Similarity between Requirements and Architecture. The Journal of Systems and Software 82(3), 544–550 (2009)
6. Castro, J., Lucena, M., Silva, C., Alencar, F., Santos, E., Pimentel, J.: Changing attitudes towards the generation of architectural models. Journal of Systems and Software 85(3), 463–479 (2012)
7. Cleland-Huang, J., Marrero, W., Berenbach, B.: Goal Centric Traceability: Using Virtual Plumblines to Maintain Critical Systemic Qualities. IEEE Transactions on Software Engineering 34(5) (2008)
8. Dermeval, D., Soares, M., Alencar, F., Santos, E., Pimentel, J., Castro, J., Lucena, M., Silva, C., Souza, C.: Towards an *i**-based Architecture Derivation Approach. In: Proceedings of the 5th International *i** Workshop, Italy, pp. 66–71 (2011)
9. Dowling, J., Cahill, V.: The K-Component Architecture Meta-model for Self-Adaptive Software. In: Matsuoka, S. (ed.) Reflection 2001. LNCS, vol. 2192, pp. 81–88. Springer, Heidelberg (2001)
10. Fernandez-Ramil, J., Perry, D., Madhavji, N.H. (eds.): Software Evolution and Feedback: Theory and Practice. Wiley, Chichester (2006)
11. Franch, X.: Incorporating Modules into the *i** Framework. In: Pernici, B. (ed.) CAiSE 2010. LNCS, vol. 6051, pp. 439–454. Springer, Heidelberg (2010)
12. Franch, X., Botella, P.: Putting Non-functional Requirements into Software Architecture. In: 9th International Workshop on Software Specification and Design (1998)
13. Franch, X., Grünbacher, P., Oriol, M., Burgstaller, B., Dhungana, D., López, L., Marco, J., Pimentel, J.: Goal-driven Adaptation of Service-Based Systems from Runtime Monitoring Data. In: 5th IEEE Workshop on Requirements Engineering for Services, Germany (2011)
14. Garlan, D., Perry, D.: Introduction to the Special Issue on Software Architecture. Journal IEEE Transactions on Software Engineering 21(4) (1995)
15. Giorgini, P., Mylopoulos, J., Nicciarelli, E., Sebastiani, R.: Formal Reasoning Techniques for Goal Models. In: 21st International Conference on Conceptual Modeling (2002)
16. Grau, G., Franch, X.: On the Adequacy of *i** Models for Representing and Analyzing Software Architectures. In: Hainaut, J.-L., Rundensteiner, E.A., Kirchberg, M., Bertolotto, M., Brochhausen, M., Chen, Y.-P.P., Cherfi, S.S.-S., Doerr, M., Han, H., Hartmann, S., Parsons, J., Poels, G., Rolland, C., Trujillo, J., Yu, E., Zimányie, E. (eds.) ER Workshops 2007. LNCS, vol. 4802, pp. 296–305. Springer, Heidelberg (2007)
17. Hall, J., Jackson, M., Laney, R., Nuseibeh, B., Rapanotti, L.: Relating software requirements and architectures using problem frames. In: IEEE Joint International Requirements Engineering Conference (2002)
18. Hill, T., Supakkul, S., Chung, L.: Confirming and Reconfirming Architectural Decisions on Scalability: A Goal-Driven Simulation Approach. In: Meersman, R., Herrero, P., Dillon, T. (eds.) OTM 2009 Workshops. LNCS, vol. 5872, pp. 327–336. Springer, Heidelberg (2009)
19. Horkoff, J., Yu, E.: Qualitative, Interactive, Backwards Analysis of *i** Models. Computer, 43–46 (2008)
20. Inverardi, P., Muccini, H., Pelliccione, P.: Checking consistency between architectural models using SPIN. In: Workshop From Software Requirements to Architectures (2001)
21. Jian, Y., Li, T., Liu, L., Yu, E.: Goal-Oriented Requirements Modelling for Running Systems. In: 1st International Workshop on Requirements at Run-Time (2010)

22. Pohl, K., Sikora, E.: The Co-Development of System Requirements and Functional Architecture. In: Conceptual Modeling in Information Systems Engineering, pp. 229–246 (2007)
23. Lapouchnian, A., Mylopoulos, J.: Modeling Domain Variability in Requirements Engineering with Contexts. In: Laender, A.H.F., Castano, S., Dayal, U., Casati, F., de Oliveira, J.P.M. (eds.) ER 2009. LNCS, vol. 5829, pp. 115–130. Springer, Heidelberg (2009)
24. Lucena, M., Castro, J., Silva, C., Alencar, F., Santos, E., Pimentel, J.: A Model Transformation Approach to Derive Architectural Models from Goal-Oriented Requirements Models. In: Meersman, R., Herrero, P., Dillon, T. (eds.) OTM 2009 Workshops. LNCS, vol. 5872, pp. 370–380. Springer, Heidelberg (2009)
25. Moody, D., Heymans, P., Matulevicius, R.: Visual syntax does matter: improving the cognitive effectiveness of the *i** visual notation. Requirements Engineering Journal 15(2), 141–175 (2010)
26. Morales, E., Franch, X., Martinez, A., Estrada, H.: Considering Technology Representation in Service-Oriented Business Models. In: 5th IEEE Workshop on Requirements Engineering for Services, Germany, pp. 482–487 (2011)
27. Nuseibeh, B.: Weaving the Software Development Process Between Requirements and Architectures. IEEE Computer 34(3), 115–117 (2001)
28. Oriol, M., Franch, X., Marco, J., Ameller, D.: Monitoring Adaptable SOA-Systems using SALMon. In: Workshop on Service Monitoring, Adaptation and Beyond, pp. 19–28 (2008)
29. Pahl, C., Gacitua-Decar, V., Wang, M., Bandara, K.Y.: A Coordination Space Architecture for Service Collaboration and Cooperation. In: Salinesi, C., Pastor, O. (eds.) CAiSE Workshops 2011, Part VI. LNBIP, vol. 83, pp. 366–377. Springer, Heidelberg (2011)
30. Pimentel, J., Franch, X., Castro, J.: Measuring Architectural Adaptability in *i** Models. In: 14th Ibero-American Conference on Software Engineering, CIBSE, April 27-29 (2011)
31. Pimentel, J., Lucena, M., Castro, J., Silva, C., Alencar, F., Santos, E.: Deriving Adaptable Software Architectures from Requirements Models: The STREAM-A approach. Requirements Engineering Journal (2011) (published online)
32. Pimentel, J., Santos, E., Castro, J.: Conditions for ignoring failures based on a requirements model. In: 22nd International Conference on Software Engineering and Knowledge Engineering, USA, pp. 48–53 (2010)
33. Pimentel, J., Santos, E., Castro, J.: Anticipating Requirements Changes – Using Futurology in Requirements Elicitation. International Journal of Information System Modeling and Design 3(2), 89–111 (2012)
34. Pinto, M., Fuentes, L., Troya, J.M.: DAOP-ADL: An Architecture Description Language for Dynamic Component and Aspect-Based Development. In: Pfenning, F., Macko, M. (eds.) GPCE 2003. LNCS, vol. 2830, pp. 118–137. Springer, Heidelberg (2003)
35. Qureshi, N., Perini, A., Ernst, N., Mylopoulos, J.: Towards a Continuous Requirements Engineering Framework for Self-Adaptive Systems. In: 1st RE @ Run-Time (2010)
36. Razavizadeh, A., Cîmpan, S., Verjus, H., Ducasse, S.: Software System Understanding via Architectural Views Extraction According to Multiple Viewpoints. In: Meersman, R., Herrero, P., Dillon, T. (eds.) OTM 2009 Workshops. LNCS, vol. 5872, pp. 433–442. Springer, Heidelberg (2009)
37. Shaw, M., Garlan, D.: Software Architecture: Perspectives on an Emerging Discipline. Prentice Hall (1996)
38. Taylor, R., Medvidovic, N., Dashofy, I.: Software Architecture: Foundations, Theory, and Practice. John Wiley & Sons (2009)
39. Yu, E., Giorgini, P., Maiden, N., Mylopoulos, J. (eds.): Social Modeling for Requirements Engineering. The MIT Press, Cambridge (2011)