Goal-Oriented Analysis of System-wide Qualities During Software Architectural Composition

Lawrence Chung
Dept. of Computer Science
The University of Texas as Dallas

Abstract

Software development through architectural composition of existing systems, or their parts, is expected to significantly reduce development time, efforts and cost. In order for compositional software architecting to be useful and usable, however, the behavior of system-wide qualities needs to be clearly understood and analyzed in terms of their interactions and in the context of the target system. This paper presents a goal-oriented approach to the incorporation of the principle of contextuality and the analysis of synergistic or conflicting interactions among system-wide qualities of the component systems during software architectural composition. In this generational approach intended to help generate a "good enough" software architecture, requirements on system-wide qualities (*non-functional requirements* or *NFRs*) and their relationships are represented respectively as *softgoals* and *satisficing* interdependencies, and reasoned about in the context of the organizational policies, usage characteristics and priorities of the target system.

1 Introduction

> "One should ask for the meaning of a word only in the context of a sentence, and not in isolation." In the introduction to *Grundlagen der Mathematik (Frege, 1923, p. x)*

Traditionally, most software systems have been developed from scratch, going through a lifecycle of requirements specification, design, implementation, testing, etc. The process is often quite time-consuming, costly and error-prone, especially when the system is large and complex. A new approach, *compositional software architecting*, has started to emerge with seemingly a great promise, in which a software system is developed not from scratch but through integration of the architectural designs of some existing systems, or their parts, i.e., through *software architectural composition* (hereafter, the two terms will be used interchangeably unless there is any confusion). Since the source systems are expected to already have gone through rigorous software lifecycles, hence reasonably well analyzed, validated, verified, tested, and maintained (See, for example, [IWASS95][CSA98]), software system development through composition is expected to significantly reduce development time, efforts and cost. But how about quality?

One critical hurdle to the progress in the area of compositional software architecting has been the lack of understanding in, and systematic analysis of, the system-wide qualities of the component systems while they are composed into a bigger system. Can the qualities of two systems, for example, be additively preserved when they are composed? If not, what are perturbed and why, and how can we improve the quality of the target system?

This paper discusses why it is futile, more often than not, to wholistically and objectively preserve the qualities during software architectural composition. In light of the discussion, this paper presents a goal-oriented approach to the incorporation of the principle of contextuality and the analysis of synergistic or conflicting interactions among system-wide qualities of the component systems during compositional software architecting. In the approach, requirements on system-wide qualities (*non-functional requirements* or *NFRs*) and their relationships are represented respectively as *softgoals* and *satisficing* interdependencies, and reasoned about in the context of the organizational policies, usage characteristics and priorities of the target system. The approach is intended to help generate a "good enough" software architecture, by helping consider alternatives of software architectural styles, patterns, components, connections and constraints, and rationally choose among them.

The goal-oriented approach to be presented in this paper adopts the NFR Framework [CNYM98] in which NFRs are treated as softgoals to be achieved, not absolutely but in a "good-enough" sense, and design alternatives, decisions and rationale are all considered in relation to the softgoals. The Framework has been applied, among other things, to rationalize the software architecting process but primarily for building quality in the architecture of a single system from scratch [CNY95][CY98]. The present paper, however, deals with system-wide qualities of multiple component systems during software architectural composition, while reflecting the principle of contextuality. Although the analysis technique presented in the paper adopts the goal-oriented approach of the NFR Framework, the technique is intended for use with other goal-oriented approaches (e.g., [vDL98] [BI96] [DvF93]) and agent-oriented approaches (e.g., [Yu95] [Bubenko93]) as well.

The Running Example: An order processing system In order to illustrate the goal-oriented approach to software architectural composition, an order processing system will be used as the running example throughout the paper since it is a real system, simple to understand and has been used widely. The order processing system to be built should take orders from customers, check if the goods ordered are available, generate invoices if so, ship the goods together with the invoices, receive payments and issue receipts for the payments. The system should also check the level of goods/parts, place stock orders if it is below the re-order point, receive the goods ordered and make payments. The construction will involve the use of two existing systems, an invoice processing and an inventory control systems, as well as some minor modifications.

Section 2 revisits the principles of compositionality and contextuality as applied to compositional software architecting, and discusses the context-sensitive and interacting nature of system-wide qualities. Sections 3 and 4 present a goal-oriented approach to the analysis of system-wide qualities respectively of an individual software architecture and of a compositional software architecture. Section 5 provides a set of guidelines for the analysis of system-wide qualities during software architectural composition. At the end, a summary of contributions is provided along with future directions.

2 The Principle of Compositionality Revisited

The principle of compositionality of meaning is described as ([Janssen96]):

> "the meaning of a compound expression is a function of the meanings of its parts. ... the principle should not be considered an empirically verifiable restriction, but a methodological principle that describes how a system for syntax and semantics should be designed."

This principle of compositionality requires that words in isolation have a meaning and that from these meanings the meaning of a compound can be built. For example, propositional logic satisfies the principle: the meaning of a formula is its true value and the meaning of a compound formula is indeed a function of the truth values of its parts. When applied to the quality of a composed system, this principle would imply that (the meaning of) the quality of a composed system is a function of (the meaning of) the qualities of its parts. Is it?

In the introduction to *Grundlagen der Mathematik (Frege, 1923, p. x)*, Frege presents a few principles, one being:

> "One should ask for the meaning of a word only in the context of a sentence, and not in isolation.'

which later acquired the name of "principle of contextuality". This formulation of contextuality disallows speaking about the meaning of words in isolation and is therefore incompatible with compositionality.

We take the premise that qualities are inherent properties of a system — just as weight, volume, color, etc. are of a physical substance, but, for the reasons below, they follow the principle of contextuality of quality:

> "One should ask for the meaning of quality only in the context of a system, and not in isolation."

The context of a software system is determined by a variety of things, such as the goals of the intended application domain the system is expected to help achieve, the characteristics of the users and usage patterns, organizational policies to be observed, resource limitations and priorities imposed, etc. During compositional software architecting, there is change in context: from the contexts of the source systems before composition, the *source contexts*, to the context of the target system after composition, the *target context*.

Now we explain why we take the premise that system-wide qualities follow the principle of contextuality of quality, using our running example described in Section 1, namely, the order processing system as the target system and the invoice processing and the inventory control systems as the source systems. Consider, for example, the meaning of "security". Before composition, it may mean guarding against unauthorized access in the invoice processing system. After composition, however, it may have a broader meaning, namely not only guarding against unauthorized access but allowing access to information when needed, hence system availability. Furthermore, the notion of security may apply to the information pertaining only to invoice processing before composition, but all types of information in the order processing system.

In moving from the source to the target contexts, system-wide qualities can change not only in their meanings but in the degree they are perceived satisfactory and in the manner they are made satisfactory. How can the degree to which qualities are perceived satisfactory change? For example, a simple password mechanism might have been considered sufficient before composition for assuring the security of the information about invoices, payments and receipts that the invoice processing system maintains. This perhaps is because the system was used in isolation, the information the system maintains was not highly confidential, or major stakeholders of the system simply felt that way, etc. After composition, however, that mechanism alone may no longer be considered satisfactory perhaps because the invoice processing system is only a component of a larger system, namely the order processing system which may be exposed to many different types of users, hence vulnerable to a variety of virus attacks. In order to maintain the same degree of security assurance, the manner security is made satisfactory may need to change, perhaps through both a simple password mechanism and virus filters.

In moving from the source to the target contexts, system-wide qualities also tend to interact with one another, either synergistically — yielding more than the sum of the parts — or antagonistically — yielding less. For example, performance may be the most important concern for the inventory control system, perhaps due to a large body of information about parts and suppliers that the system has to handle. In the target system, however, accuracy may become more important than performance, perhaps due to the increasing cases of intentional fraud reported recently on other similar systems.

During software architectural composition, thus, system-wide qualities cannot simply be additively composed but instead should be analyzed in consideration of the differences in contexts and their interactions in the target context. The remainder of the paper describes how to systematically carry out such an analysis which can be quite complex and error-prone, due to the various types of changes and their interactions to be considered, and due to many design alternatives to be considered and many decisions to be made.

3 The Goal-Oriented Approach

A useful and usable software architectural design requires a clear understanding and analysis of the behavior of system-wide qualities in relation to the requirements on the system, since it is requirements that contextually describe the functionality of, and the constraints on, the system.

Softgoal Interdependency Graph (SIG). In the goal-oriented approach being presented in the paper, requirements on system-wide qualities (i.e., NFRs) are treated as goals to be achieved. However, NFRs are often hard to precisely and objectively state as problems to be solved, tend to be conflicting or synergistic with one another, their solution space can be quite huge, and consequently solutions to such problems tend to be sub-optimal. For these reasons, NFRs are treated as (*NFR*) *softgoals* to be achieved, not in an absolutely clear-cut sense but in a *satisficing*, or satisfactory or "good enough", manner. Once stated, the meaning of NFRs is iteratively clarified through decomposition, satisficed through *operationaling sofgoals*, and evaluated through softgoal *labels* and a label propagation procedure. Throughout the process, softgoals are prioritized through a criticality attribute, design decisions are justified through *argument softgoals*, and the various types of softgoal inter-relationships are represented as *contributions*. All these activities are carried out in consideration of the particular context in which the projected system is to function. The goal-oriented approach offers a graphical notation, *softgoal interdependency graph (SIG)*, to organize and manage the three categories of softgoals, the status of their satisficing, their criticalities and contributions.

Contributions are categorized into contribution types including *AND* (if the offspring are satisficed, so will be their parent), *OR* (if any of the offspring is satisficed, so will its parent), *MAKE* (a softgoal makes a sufficient positive contribution to its parent), *HELP* (insufficient positive), *BREAK* (sufficient negative) and *HURT* (insufficient negative). A label, such as "weakly satisficed", "strongly satisficed", "weakly denied" and "strongly denied", indicates the degree to which it's associated softgoal is satisficed. More details on contribution types and how they are used in propagating labels can be found in [CNYM98].

Figure 1 shows a softgoal interdependency graph for an invoice processing system. In the figure, se-

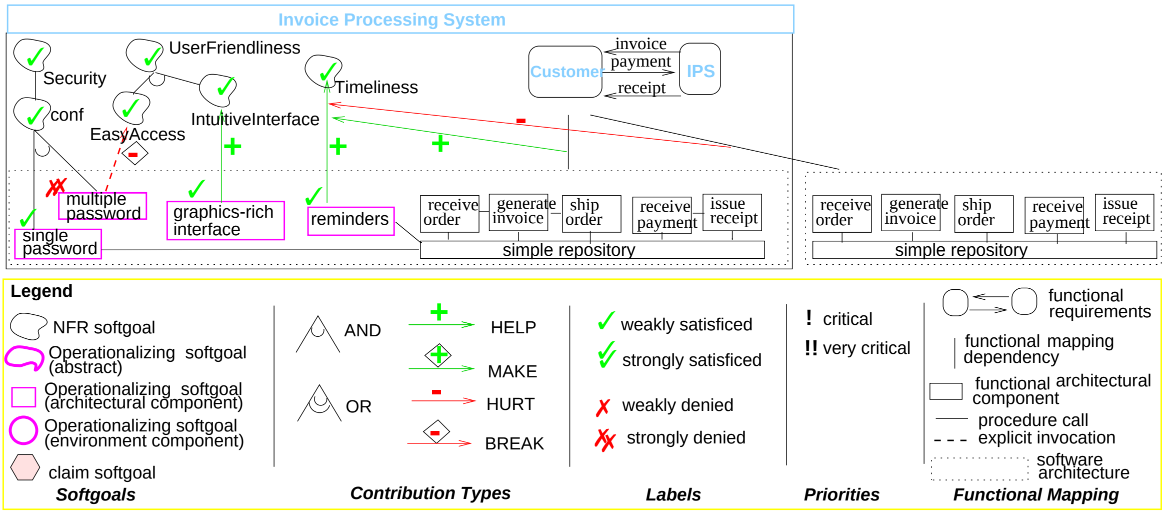

Figure 1: Application of the goal-oriented approach to an invoice processing system architecting.

curity, userfriendliness and timeliness requirements are all represented as NFR softgoals. The meaning of the security requirement is clarified in the context of the invoice processing system as confidentiality requirement, "guarding against unauthorized access", and the meaning of the userfriendliness requirement through decomposition as easy access and intuitive interface requirements. In order to satisfice the confidentiality requirement, two operationaling goals are considered: a "single password" mechanism and a "multiple password" mechanism. The "multiple password" mechanism is perceived to BREAK the easy access softgoal, hence not chosen as a component of the target architecture (hence a "strongly denied" label) but a "single password" mechanism is (hence a "strongly satisficed" label). The two operationaling goals, "graphics-rich interface" and "reminders", are chosen respectively to satisfice the intuitive interface and timeliness softgoals.

Functional Mapping Dependency Graph (FMDG). In the goal-oriented approach, NFR softgoals are satisficed not solely by those operationalizing softgoals specific to them but additionally by other parts of the software architectural design that are intended to meet functional requirements. This is in accord with our premise that qualities are inherent properties of a system, although they are context-sensitive. Put differently, an architectural design on the one hand is associated with different qualities depending on what kind of components it has and how the components interact with each other, and manifests qualities as it makes various types of partial or full, positive or negative, contributions to the satisficing of NFR softgoals, on the other.

Figure 1 depicts the *functional mapping dependency graph* (or *FMDG*) for the invoice processing system, involving the functional requirements and a couple of alternative architectures. Between the softgoal interdependency graph (SIG) and the FMDG are shown the kinds of contributions the mappings, from the functional requirements into the two architectures, would make towards the satisficing of the NFR softgoals. The mapping of the functional requirements into the first architecture is considered HELPing meet the timeliness NFR softgoal, as some of the components of the first architecture directly communicate with each other (e.g, an invoice is generated with no delay when an order is received). In contrast, the mapping into the second architecture is considered HURTing the same timeliness softgoal, since all components have to be invoked explicitly by the user and consequently delay is bound to occur between those components that are temporally inter-related.

The two graphs, SIG and FMDG, are used not only as a means for explicitly representing, and reasoning about, system-wide qualities during the software architecting process but as an historical record for later review, evolution, and particularly in the present paper for applying the goal-oriented approach to compositional software architecting.

Figure 2 shows another application of the goal-oriented approach to architecting inventory control system, which will be used in the next section. In the SIG part of the figure, userfriendliness, accuracy and

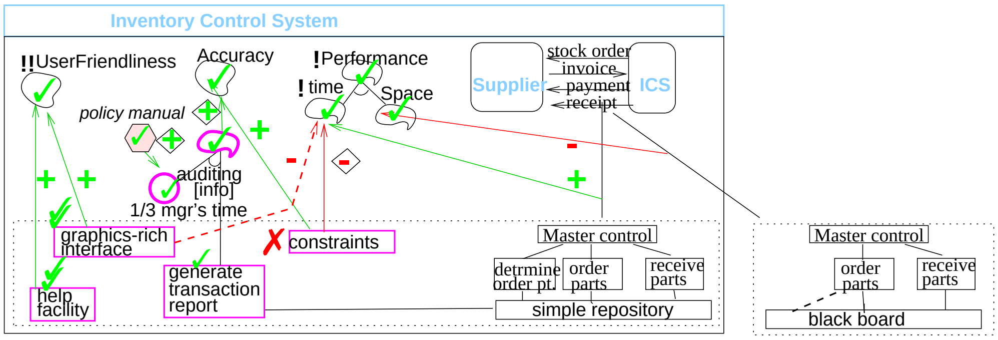

Figure 2: Application of the goal-oriented approach to an inventory control system architecting.

performance requirements are posted as softgoals. In order to satisfice the userfriendliness softgoal, "help facility" is considered as an operationaling goal and chosen as a component of the target architecture. An additional operationalizing goal, "graphics-rich interface", is also considered but can HURT the response time softgoal. In order to resolve this conflict, the priorities associated with the userfriendliness and response time softgoals are taken into consideration: the decision is to deploy "graphics-rich interface", since the userfriendliness softgoal is more critical to satisfice than the response time softgoal. Another use of priorities in conflict resolution is also seen, this time between the accuracy and response time softgoals. Automatic enforcement of "constraints" is considered HELPing accuracy but BREAKing response time, and the decision is in favor of response time since it has a higher priority than accuracy. The SIG also shows a domain resource allocation as an operationalization. Auditing can MAKE the inventory control system accurate, only if the transaction report can be examined carefully. The SIG shows the allocation of a chunk of a manager's time to the task, which is justified through the policy of the intended organization, which is in turn represented as a supporting argument softgoal.

The FMDG in figure 2 shows a couple of architectural alternatives. In the first architecture, parts are ordered after the determination of re-order points and the examination that the stock is below the re-order points, which are all explicitly controlled by the master control component. In the second architecture, in contrast, the re-order points are stored in a blackboard which triggers the ordering of parts when the stock falls below the minimum level. The links between the SIG and the FMDG show the relative contributions that the two architectures make towards the satisficing of the response time softgoal: compared to the first one, the second architecture can HURT response time due to the processing overhead involved in the detection and announcement of events.

4 System-wide Qualities During Compositional Architecting

Compositional software architecting involves merging the source architectures, detecting and resolving inconsistencies, and improving the target architecture. As discussed in Section 2, however, system-wide qualities cannot simply be merged, even in the absence of architectural inconsistency, but instead should be analyzed in consideration of the differences between the source and target contexts and of the interactions between system-wide qualities in the target context.

In the goal-oriented approach, such an analysis amounts to merging the source softgoal interdependency graphs (SIGs), as well as the functional mapping dependency graphs (FMDGs). The analysis should continue, in consideration of the target context, with: the determination of the meaning of NFR softgoals, the degree to which they are satisficed and the manner in which they are satisficed, as manifested respectively by their priorities and decompositions, their labels and their operationalizations; the detection of antagonistic and synergistic interactions between NFR and operationalizing softgoals respectively represented as negative and positive contribution types; and improvement on the quality of the target architecture by expanding the set of NFR softgoals or increasing the degrees of satisficing.

Figure 3 illustrates the application of the goal-oriented approach to the analysis of system-wide qualities during software architectural composition, using the invoice processing and inventory control systems as the source systems and the order processing system as the target system.

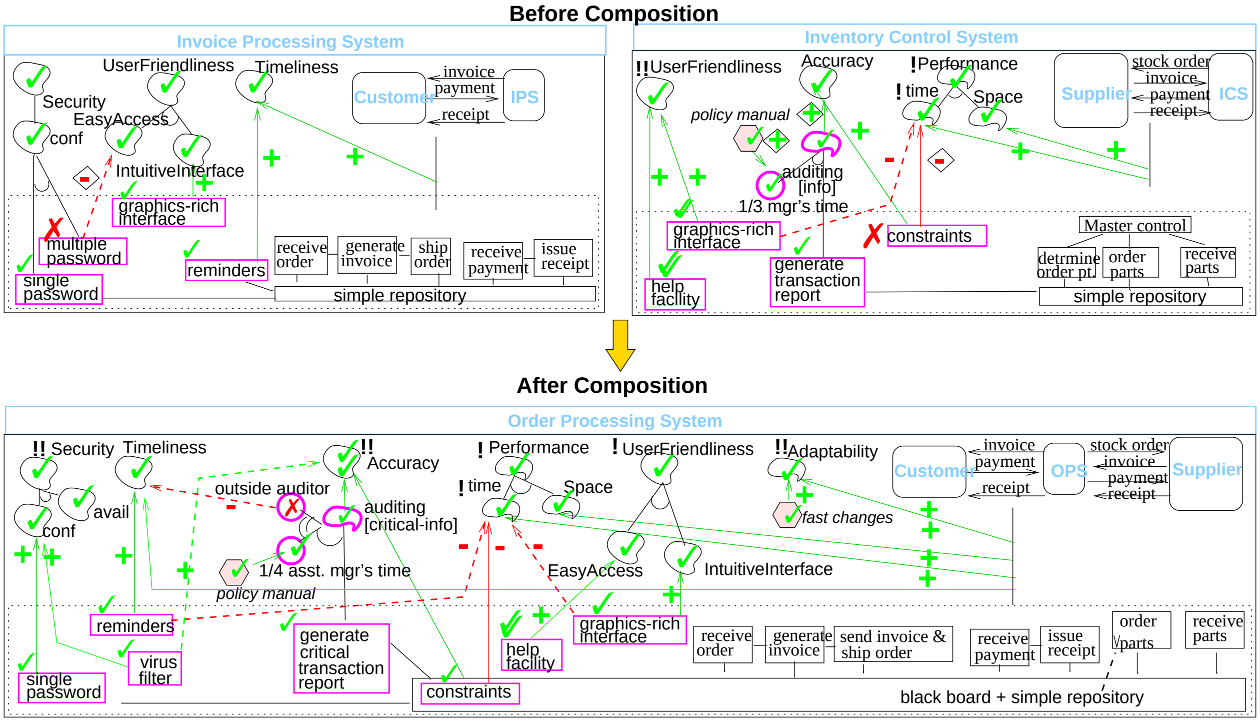

Figure 3: Goal-oriented analysis of system-wide qualities during composition.

Softgoal Interdependency Graph. Figure 3 shows change in the meaning of the security softgoal, through decomposition, due to change in context: from "guarding against unauthorized access" (confidentiality), in the invoice processing system, to more broadly "allowing access for retrieval of general information when needed" (availability) as well, in the order processing system. For another example, the meaning of userfriendliness, which is common in both of the source SIGs, is adopted only from that in the invoice processing system, namely, easy access and intuitive interface, hence different from that in the inventory control system.

Change in priority can induce change in the manner NFR softgoals are satisficed. Automatic enforcement of integrity constraints can reduce errors, hence HELPing accuracy, but at the same time induce significant processing overhead, hence HURTing response time. For this reason, the decision before the composition was not to allow integrity constraints in the inventory control system, since response time had a higher priority than accuracy. After the composition, however, accuracy is considered more importance than performance perhaps due to the past experience involving serious user complaints about inaccurate order information. This leads to re-consideration of those alternatives for enhancing accuracy that were earlier discarded (the history record still remains, of course), here in terms of the SIG of the inventory control system, resulting in change in the decision and reviving the constraint enforcement mechanism.

Change in priority can also induce change in the degree to which softgoals are satisficed. Before the composition, the userfriendliness softgoal had a higher priority than the response time softgoal in the inventory control system. After the composition, however, the two softgoals have the same priority in the order processing system. As a result, less of graphical images is used for "graphics-rich interface" in order to avoid HURTing response time too much, but just enough to satisfice the intuitive interface softgoal.

The manner whereby softgoals are satisficed can change as well. Due to its open-nature, for example, the target system is vulnerable to virus attacks, and subsequently the security softgoal is satisficed not solely by a simple password mechanism but additionally by virus filters. Change in the availability of resources, such as office workers, hardware and other software systems, can also change the manner whereby softgoals are satisficed. The accuracy softgoal in the inventory control system was operationalized through auditing to be carried out by a manager. The policy of the target organization, however, may prohibit a manager from spending a chunk of her time as an auditor. At least a couple of alternatives may seem feasible as replacement: an outside auditor and an assistant manager. An outside auditor seems less readily available than an assistant manager who can spend as part of her regular duty some of her time examining the accuracy of information. Thus, an assistant manager is chosen for the task of auditing, but with a reduced scope on only critical information. This change in the availability of resources also leads to change in the behavior of the system, namely, from the generation of report for all transactions to that for only critical transactions.

The SIG in the figure also illustrates a synergistic interaction, between the security softgoal from the invoice processing system and the accuracy softgoal from the inventory control system. Virus filters, which HELP improve the confidentiality aspect of security, can also reduce intentional fraud, hence HELPing accuracy as well. An antagonistic interaction is also visible in the SIG, namely, between the timeliness softgoal from the invoice processing system and the performance softgoal from the inventory control system. Generation of reminders HELPs maintaining up-to-date information, but can HURT performance due to the processing overhead.

Functional Mapping Dependency Graph. Figure 3 shows the mapping of the target functional requirements into a target architecture, along with the kinds of contributions the mapping makes towards the satisficing of the target NFR softgoals. Although several alternative architectures were considered, the figure shows only the finally chosen architecture due to space limitation. The architecture has been chosen since it is considered to have the right kinds of components and component interactions, hence the desired system-wide properties. Recall that the invoice processing system adopted a simple repository style for its architecture but with direct communication between its components, and the inventory control system again a simple repository but with a master control coordinating its sub-components. In the target system, however, adaptability becomes a vital quality concern. One of the two source architectural styles can be chosen, but neither is considered "good enough" for accommodating the components and connections of the target system such that they become easily replaceable, modifiable and reconfigurable. The decision is to adopt a hybrid style: a repository stores data for both invoice processing and inventory control as before, but direct communication applies only to the components coming from the invoice processing system, while implicit invocation applies to the ordering of parts when the stock falls below the reorder points (this is resurrection of one of the architectural alternatives for the inventory control system which was earlier discarded).

5 Guidelines for the Analysis Process

Analysis of system-wide qualities is hard during software architectural composition, since it involves multiple source systems which can be inconsistent with one another and just about anything can change including contextuality and quality interactions. This section describes a set of guidelines for detecting inconsistency in the target SIG during composition and restoring it to a consistence state, making and propagating changes, and improving the quality of the target system. The guidelines are based on the structure of the softgoal interdependency and functional mapping dependency graphs, in particular, the three categories of softgoals and the way softgoals are prioritized, decomposed, operationalized, inter-related, and evaluated. Although currently for manual use by software architects, the guidelines can be formalized and built into a CASE tool which would then support the analysis process.

The target architecture is produced through an iterative and interleaving process of integration of the source SIGs into a target SIG and the source FMDGs into a target FMDG, and improvement on the quality of the target system. Throughout the process, inconsistencies should be detected, changes should be made and propagated, and traceabilities, both forward and backward, have to be established at the end, assuming they have already been for the sources. Forward traceability is established if each NFR softgoal is satisficed through an operationaling softgoal or the target architecture as a whole; and backward traceability if each operationalizing softgoal is traceable to a NFR softgoal, which in turn should be traceable to its sources, such as domain agents or policies. This is similar in spirit to the guidelines for maintaining desirable properties of SIGs in dealing with system evolution, as described in [CNY96], although here we are concerned with composition of software architectures involving multiple source systems.

Producing a unified target SIG. Integration of the source SIGs into a *consistent target SIG* should be done in consideration of relationships among the source SIGs. The source SIGs can be either completely *disjoint* from, or at least partially *overlapping* with, each other. If the source SIGs are disjoint from each another, so will their softgoals be; otherwise there should be at least one softgoal that is common to more than one source SIG. For example, the SIGs of the invoice processing and inventory control systems are overlapping with each other, since a userfriendliness softgoal is common to both SIGs.

When the source SIGs are disjoint from each other, there would be nothing in common among the source SIGs. In this case, a first-cut approximation of a consistent target SIG can be obtained simply by merging the source SIGs. The target SIG is only a first-cut approximation, since there is no guarantee that a softgoal will retain the same relative priority, or label, as before in the target SIG, which after all must have a pool of softgoals larger in number than any of its source SIGs has. This discussion leads to another notion of consistency, *semantic consistency*, which, although unlikely, would hold if there are no contextual differences between the source and target systems and no interactions between system-wide qualities from difference source systems. In other words, a consistent target SIG would be produced through merging only if the source SIGs are disjoint and semantically consistent.

Unlike disjoint source SIGs, overlapping source SIGs can be *syntactically inconsistent* if a softgoal from one source SIG has a different priority, or label, than the same softgoal from another SIG; otherwise they are *syntactically consistent*. For example, the SIGs of the invoice processing and inventory control systems are overlapping with each other and syntactically inconsistent, since a userfriendliness softgoal is common to both but with different priorities.

If overlapping source SIGs are syntactically consistent, a first-cut approximation of a consistent target SIG can be obtained by superimposing the source SIGs. The target SIG is again only a first-cut approximation, since the type of contribution between the same two softgoals can vary from one source SIG to another, i.e., due to *contribution inconsistency*. In other words, a consistent target SIG would be produced through superimposing the source SIGs that overlap, only if the source SIGs are consistent, contribution-wise, syntactically and semantically.

In order to deal with syntactic, semantic or contribution inconsistency, all the elements of the source SIGs need to be examined. In the goal-oriented approach, one way to deal with inconsistent elements is to treat them as OR alternatives (e.g., two different priorities of the same softgoal), and discard all but one as the chosen one (e.g., the highest priority), possibly after a tradeoff analysis, or even replace them all with a completely new one (e.g., a new priority) if none of them is suitable.

Also a decomposition can be introduced in the target SIG when several softgoals from different source SIGs can be hierarchically related to each other. For example, the scope of accuracy may vary from information about all customers in one source SIG to information about high-priority customers in another, information about regular customers in yet another, etc. In this case, accuracy softgoals can be represented along a hierarchy in the target SIG in which accuracy of information about all customers becomes the parent of accuracy of information about high-priority customers and accuracy of information about regular customers, etc. This kind of hierarchy then can be used in dealing with inconsistent operationaling softgoals, for example, auditing of information about high-priority customers by a manager, auditing of information about regular customers by an assistant manager, etc.

As a rule-of-thumb, a high-priority softgoal should be given more time and effort in traversing (usually in a top-down manner) the source SIGs and resolving conflicts. The goal of such a traversal should include: detecting and resolving inconsistency in softgoal priorities and decompositions (i.e., the same parent has different types, or numbers, of offspring in different SIGs); validating operationalizing softgoals which are environmental constraints or agent assignments, especially in relation to argument softgoals if there is any; detecting and resolving contribution inconsistency, and establishing new correlations, often between NFR softgoals from one source and operationalizing softgoals from another; and detecting and resolving inconsistency in softgoal labels, and iteratively propagating them upward.

Producing a unified target FMDG. As with a unified target SIG, the goal-oriented approach can be used to represent as alternatives inconsistency in the architectural components and component interactions from the different source FMDGs, as well as in constraints, patterns and styles. The approach can also be used to subsequently carry out tradeoff analyses, select among the alternatives or even replace them with an entirely new one, and justify design decisions and record design rationale. Design decisions should be made in relation to the contributions that different mappings, from functional requirements into architectural designs, make towards NFR softgoals in the target SIG.

Improving quality. The quality of a software architecture can be improved through increase in the degree of satisficing of the NFR softgoals in the target SIG or addition and satisficing of new NFR softgoals.

A useful rule-of-thumb is the *80-20 rule* (or the *Pareto principle)* [Juran79]: 20% of the population have 80% of the entire wealth as it applied originally to economics. In the goal-oriented approach, this implies satisficing those NFR softgoals with the highest priority that are either denied or weakly satisficed. They can be improved through a top-down traversal of the SIG and determination and satisficing of their key offspring. For an AND contribution, all offspring are important; for OR, at least one should be satisficed; a MAKE offspring is more important to be satisficed than a HELP offspring; and a BREAK offspring is more important to be denied than a HURT offspring.

A top-town traversal of the target SIG may indicate that there is only one operationalizing softgoal that can satisfice some critical NFR softgoals or their offspring. For example, the denial of the operationalizing softgoal implies the denial of one or more critical NFR softgoals. In this case, the operationaling goal should be treated as a high priority item and satisficed more strongly. On the other hand, a top-town traversal may indicate the presence of multiple ways of improving the same important NFR softgoals, e.g., multiple MAKE offspring that are either denied or weakly satisficed, and multiple BREAK offspring that are satisficed. Due to resource limitation in the target context, only some of them may be made satisficed or denied. In this situation, another rule-of-thumb can be used, *maximum synergy and minimum conflict rule* which in the goal-oriented approach typically implies satisficing and deploying those operationalizing softgoals (i.e., they should have an S label) that make strongest positive, and weakest negative, contributions to as many NFR softgoals as possible, and denying and discarding those (consequently, a D label) with strongest negative, and weakest positive, contributions to NFR softgoals.

6 Conclusions

In spirit of the principle of contextuality, this paper has examined the context-sensitive and interacting nature of system-wise qualities during compositional software architecting. By treating architectural components, connections and styles as operationalizations of system-wise qualities, the paper has adapted the goal-oriented approach of the NFR Framework to the representation of system-wide qualities as softgoals to be satisficed and architectural alternatives as satisficing solutions during software architectural composition. This adaptation allows for a systematic analysis of synergistic or conflicting softgoal interactions in consideration of the differences between the source and target contexts. Using the structure of the softgoal interdependency and functional mapping dependency graphs, the paper has also presented a set of guidelines for detecting and restoring inconsistency among the system-wide qualities of the component systems, for making and propagating changes, and for improving the quality of the target system.

The approach presented in this paper requires further refinement, especially concerning the qualities as manifested by varying granularities of architectural components, connections, constraints, patterns and styles. Another future research concerns a tool development involving the NFR Assistant and the Architectural Mapping Assistant which respectively would support the process of constructing softgoal interdependency and functional mapping dependency graphs. The two Assistant projects have been underway somewhat independently from each other, hence in need of integration as well as improvement in conjunction with the guidelines presented in the current paper. In order for the goal-oriented approach to compositional software architecting to be useful and usable in practise, real(istic) and bigger case studies are also needed, perhaps involving the distributed, cooperative processing paradigm.

References

[BI96] B. Boehm and H. In, "Identifying Quality-Requirement Conflicts", *IEEE Software*, pp. 25–35, March 1996.

[Bubenko93] J. A. Bubenko, "Extending the Scope of Information Modelling", *Proc. 4th Int. Workshop on the Deductive Approach to Information Systems and Databases,* Lloret-Costa Brava, Catalonia, Sept. 20–22, 1993. pp. 73–98.

[CNY95] L. Chung, B. Nixon and E. Yu, "Using Non-Functional Requirements to Systematically Select Among Alternatives in Architectural Design," *Proc., 1st Int. Workshop on Architectures for Software Systems,* Seattle, April 24-28, 1995., pp. 31-43.

[CNY96] L. Chung, B. A. Nixon and E. Yu, "Dealing with Change: An Approach Using Non-Functional Requirements," *Requirements Engineering Journal*, **1**(4), pp. 238–259, 1996.

[CNYM98] L. Chung, B. A. Nixon, E. Yu and J. Mylopoulos, *Non-Functional Requirements in Software Engineering,* Kluwer Publishing (to appear).

[CSA98] *OMG-DARPA-MCC Workshop on Compositional Software Architectures,* Monterey, CA, January 6-8, 1988.

[CY98] L. Chung and E. Yu, "Achieving System-Wide Architectural Qualities", *OMG-DARPA-MCC Workshop on Compositional Software Architectures,* Monterey, CA, January 6-8, 1988.

[DvF93] A. Dardenne, A. van Lamsweerde and S. Fickas, "Goal-Directed Requirements Acquisition", *Sciences of Computer Programming,* 20, 1993. pp. 3–50.

[IWASS95] D. Garlan (ed.), *1st International Workshop on Architectures of Software Systems*, Seattle, WA. 1995.

[Janssen96] T. M. V. Janssen, "Compositionality", in J. van Benthem and A. ter Meulen (eds.), *Handbook of Logic and Language*, Elsevier Pub. 1996.

[Juran79] J. M. Juran, Frank M. Gryna Jr., and R. S. Bingham Jr. (Eds.), *Quality Control Handbook,*, 3rd Ed., New York: McGraw-Hill Book, 1979.

[MCY99] J. Mylopoulos, L. Chung and E. Yu, *From Object-Oriented to Goal-Oriented Requirements Analysis,* submitted for publication.

[vDL98] A. van Lamsweerde, R. Darimont and E. Lieter, *Managing Conflicts in Goal-Driven Requirements Engineering,* submitted for publication.

[Yu95] E. Yu, *Modelling Strategic Relationships for Process Reengineering,* Ph.D. Thesis, Dept. of Computer Science, Univ. of Toronto, 1995.

II. *IPS* — An Invoice Processing System

The system shall issue invoices to customers, receive payments, issue receipts for these payments and reminders for unpaid invoices.

Once a week, payments which have been made are reconciled with the invoices. This might be done by associating invoice #, date, amount and customer # with each payment, and by associating invoice #, date, amount and customer with each invoice.

For those invoices which have been paid, a receipt is issued, and its record is kept in order to handle any potential complaints from customers in future. Issuing a receipt might be done by associating, customer #, name, address and credit period with each customer, and by associating invoice #, date, amount and customer # with each receipt.

For those invoices which have not been paid within the allowed payment time, a reminder is issued. This might involve comparing invoices with payments and identifying all payments which are due.

Depending on the architectural style chosen, certain modules may be needed to deal with I/O of various sources and destinations (e.g., reading invoices from a database, reading customer information from a database, writing payments into a database, writing receipts into a database, etc.).

II. EOPS — Electronic Order Processing System

Recent surveys indicate that a war is breaking out, if not there already, a war which is going to change the way we live. A war of electronic commerce, for faster, more accurate, more convenient, more flexible, more mobile, more nomadic, cheaper, etc. Your team's task is to build a software architecture for an electronic order processing system (hereafter, *EOPS*). The *IPS* can be considered as an important part of the EOPS, which in turn an important part of the architecture for electronic commerce.

A customer places an order (e.g., software-on-demand) electronically by filling in an order entry form. The form is taken by the EOPS and results in the generation of a concrete order form, which is to be used for the purpose of accounting, ordering goods or services (from other agents), and shipping.

The process of generating an order form can involve matching the needs of the customer against the goods and services the company offers. This process will be based on the order entry form which is usually incomplete in the beginning, but becomes more concrete as more information is gathered at a later stage. The process can also involve informing the customer of the choices she has, possibly by displaying parts of the catalogue the company maintains which of course continuously changes. For each selected item, the customer can have a list of purchase options which can include, for example, an immediate delivery of high-quality goods or services at a high cost, a poor-quality goods or services whose delivery date is uncertain, but cheaper, a one-year warranty plan, a five-year free upgrade plan, etc. EOPS should also be able to negotiate with the customer concerning the type, the price and the delivery date.

When the customer is satisfied with the result of the process, EOPS should log the order to generate the shipping addresses, shipping dates and shipping lists. Based on the shipping addresses, each (virtual or physical) mailing label is generated; based on shipping lists, each invoice is generated, and (hopefully) the customer pays.

Each customer can place more than one order at a time, where each order can involve several copies (or quantities). The customer can also add to her previous order(s), cancel them, or change them within certain pre-set date from the time the order is place.

The activity of filling an order is completed when mailing labels and shipping invoices are used for composing (virtual or physical) packing slips. The activity of processing an order, in turn, is completed when each order is shipped back to customer either electronically or physically using packing slips. As with *IPS*, the system shall issue invoices to customers, receive payments, issue receipts for these payments and reminders for unpaid invoices (This mode of operation could be preferable to the current mode in practise, namely pre-payment). EOPS should carry out all these activities electronically.

[Handbook of Logic and Language, Eds. J. van Benthem and A. ter Meulen,
Elsevier, 1996]
The principle of compositionality of meaning:
the meaning of a compound expression is a function of the meanings of its parts.
...the principle should not be considered an empirically verifiable
restriction, but a methodological principle that describes how a system for
syntax and semantics should be designed.

>>>KLC: "function of" --- functional; non-functional

... factorize the design problem
... Thus compositionality forms a reformulation of old wisdom,
attributed to Philippus of Macedonia: divide et impera (divide and conquer)

... Many abhistory of the principle of compositionality call it
{\it Frege's Principle}...

...In the introduction to Grundlagen der Mathematik (Frege, 1923, p. x), Frege
presents a few principles he promises to follow, one being:

One should ask for the meaning of a word only in the context of a sentence,
and not in isolation.

Later this principle acquired the name of 'principle of contextuality'

Propositional logic satisfies the principle:
the meaning of a formula is its true value and the meaning of a compound
formula is indeed a function of the truth values of its parts.

For example, input verification may help both accuracy and confidentiality, hence the two behaving mutually synergistically, but may hurt performance, hence accuracy and performance behaving antagonistically with each other.