Evaluating Consistency between BPEL Specifications and Functional Requirements of Complex Computing Systems using the NFR Approach

Anisha Vemulapalli
Department of Computer Science,
The University of Texas at Tyler, Texas, USA
avemulapalli@patriots.uttyler.edu

Nary Subramanian
Department of Computer Science,
The University of Texas at Tyler, Texas, USA
nsubramanian@uttyler.edu

Abstract – **Complex computing systems are usually composed of several subsystems interacting with each other. The functional requirements of the system being developed can be captured by BPEL (Business Process Execution Language) specifications, which are obtained from UML (Unified Modeling Language) diagrams. The captured BPEL specifications can be simulated in an environment such as NetBeans. The functional requirements of a complex system are known to change throughout the development due to changes in client's needs. Consistency between the BPEL specifications and the functional requirements is important to ensure that the system developed based on BPEL specifications satisfies the client's needs. In this paper we employ the NFR Approach, where NFR stands for non-functional requirements, to evaluate this consistency. We apply this approach to an example complex system, namely, the elevator control system, by first listing the functional requirements, capturing the requirements in BPEL, and evaluating the consistency between them using the NFR Approach. Based on this evaluation we are able to determine changes needed to BPEL specifications to match the functional requirements, justify the reasons for changes to the BPEL, and keep historical record of changes performed to the specifications during the system development process. It is our belief that practitioners in both industry and academia will find the NFR Approach to consistency evaluation a significant aid during complex systems development.**

Keywords – **Functional Requirements; Consistency; BPEL; NFR Approach**

I. INTRODUCTION

Dependence on complex computing systems has grown steadily from the past few years. A complex computing system (a complex system, henceforth) is usually composed of several subsystems which interact with each other [1]. The development of a complex system usually begins with an analysis of the business domain for capturing the user requirements. The user requirements of any software system, which typically constitutes the core of a complex computing system [1], consist of both functional requirements and non-functional requirements (NFR's). Functional requirements state what a software system should do while NFR's specify the overall properties of the system such as reliability, security, and performance. Both functional requirements and NFR's must be considered while designing a complex system. BPEL (Business Process Execution Language) is an executable language which provides a means to capture the functional requirements of a complex system [2] and simulate the behavior of the system in an environment such as NetBeans [3]. However, an important requirement for functional requirement is consistency [4] and this is especially important in complex systems when two different representations of the functional requirements exist – one in BPEL form and the other in non-executable form such as use-cases, user-stories, and written descriptions. Consistency between the two representations will help develop accurate scheduling and personnel estimates, identify test cases for user acceptance, and ensure that the complex system developed indeed meets user's needs. Consistency between functional requirements and BPEL specifications for a complex system is more than just mapping between the two representations since they are not always one-to-one; that is multiple functional requirements may be mapped to one set of BPEL specifications and vice-versa.

Some examples of complex systems mentioned in the literature include:

1. Elevator control system modeled as a multiagent system where each elevator is an agent collaborating with other elevators for reducing the waiting time for people to use an elevator [5].
2. Air traffic control system which is complex system that is responsible for displaying flight-related information to air traffic controllers [6].
3. Mycology Laboratory Information Management System (MYCO-LIMS), a web-based complex system capable of providing services such as managing microarray gene expression data, laboratory supplies, and patients [7].

BPEL specifications are often developed from the UML (Unified Modeling Language) models [8] that are created for the complex system as part of the functional requirements documentation for the system. However, inconsistencies between BPEL representation and UML

representation can arise due to any one or more of these reasons:

1. UML representations do not always transform into BPEL representation in a one-to-one manner
2. UML representations may be mis-understood
3. UML representations may not be complete
4. UML representation changes may not be propagated to the BPEL specifications, and
5. Not all UML aspects may be amenable to BPEL representations.

It would be extremely useful to practitioners in industry and academia if they could evaluate the consistency between BPEL specifications and functional requirements specifications on a continual basis so that they may identify the milestones during system development when the specifications are inconsistent with each other by more than tolerable margins.

Techniques for evaluating consistency exist in the literature. A quantitative approach has been proposed by [9] for evaluating design quality, especially, that of consistency. However, design follows requirements and this technique is not easily applied to evaluating consistency between two requirements representations. Failure of consistency has been evaluated as a failure of reliability in [10] – this can related to specifications consistency by evaluating their reliability first; however, in our opinion reliability and consistency are two different aspects and therefore we considered this technique inappropriate. In [11], a program is verified to correspond with its specifications using theorem-proving techniques - however, in our opinion; this approach requires expertise and large amount of time to ensure correctness in the software and is not directly applicable to proving consistency between specifications. In [12] an approach for checking models for consistency automatically via state exploration has been proposed - but this approach analyses consistency between requirements and design by considering the events in the state transitions which is not only often impractical but also does not help evaluate consistency between specifications.

In this paper we employ the NFR Approach [13, 15] to evaluate consistency between the BPEL specifications and functional requirements. The advantages offered by the NFR Approach for this evaluation include:

1. Ability to capture varying definitions of consistency
2. Ability to capture justifications for evaluations
3. Provides an easy to use graphical artifact to capture evaluation decisions
4. Provides a means for retaining historical decision-making
5. Ability to evaluate both qualitatively and quantitatively.

This paper is structured as follows: in Section II we describe how functional requirements are converted into BPEL specifications for an elevator control system; Section III describes briefly the NFR Approach used for evaluating consistency and applies the Approach to the BPEL specifications developed in Section II; and Section IV concludes the paper with a discussion of the experiences during this evaluation process and also presents some directions for further research.

II. THE ELEVATOR CONTROL SYSTEM

In this section we describe the functional requirements for an elevator control system [5] and develop BPEL specifications for the system.

A. Description of the System

An elevator control system controls the motion of multiple elevators and responds to the elevator user requests at various floors. Each elevator has a set of destination buttons, a set of floor lamps, and a hoist motor which responds to the direction commands – up, down and stop, and returns a status. Also each elevator has a door motor to control the doors to open or close and return the status. Each floor has an elevator arrival sensor. Each floor except the top and bottom floors have up and down floor buttons and corresponding lamps to indicate the direction the elevator is heading to. The top and bottom floors have a single down and up floor button and corresponding lamps, respectively.

B. Functional Requirements for the System

The elevator control system enables elevator users to control elevators with a simple button-based interface. A user is anyone that uses an elevator in a building serviced by the system. The user is assumed to know how to interact with push buttons. An elevator will require finite time to arrive the destination floor. The elevator button starts illuminating when the user presses it, and stops illuminating after it reaches the requested floor. The floor lamps illuminate when the user presses the floor button and stops illuminating as the elevator reaches the requested floor. The minimum functional requirements of the elevator control system can be listed as follows:

F1. User can request elevator at any floor by pressing the floor button.
F2. User can request elevator to stop at any floor serviced by an elevator by pressing the elevator button.
F3. When there are no requests for an elevator it remains at current floor with doors closed.
F4. The elevator moves to the appropriate floor requested by the user.

Based on the above functional requirements, UML diagrams were developed as shown in Figure 1, which depicts the use case model diagram for the system, class diagram for the system, the activity diagram for the ‘elevator controller’ class, and two sequence diagrams for two scenarios: serving a floor request and serving the elevator request.

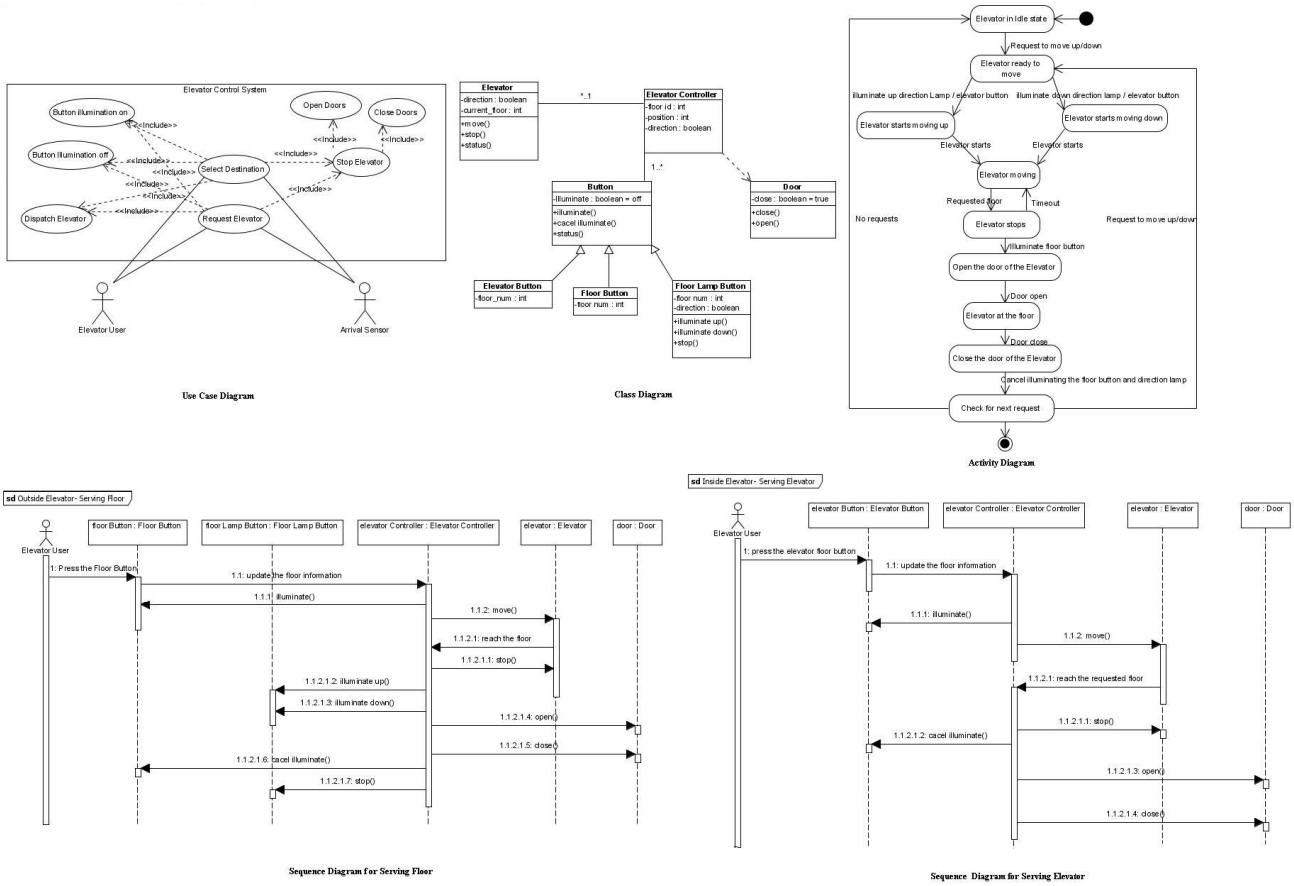

Figure 1. UML Diagrams for Elevator Control System Captured as Part of the Functional Requirements

```
<sequence name="ServicingFloor">
    <receive name="start" createInstance="yes" partnerLink="FloorButton"
    operation="RequestForElevator"
    xmlns:tns="http://j2ee.netbeans.org/wsdl/ECS"
portType="tns:RequestElevatorPT"/>
    <assign name="CopyFloorInfo"/>
    <invoke name="UpdateFloorInfo" partnerLink="ElevatorController"
    operation="RequestForElevator" xmlns:tns="http://j2ee.netbeans.org/wsdl/ECS"
    portType="tns:RequestElevatorPT"/>
    <invoke name="UpdateFloorButtonInformation"
partnerLink="ElevatorController"
    operation="FloorButtonIlluminateOn"
xmlns:tns="http://j2ee.netbeans.org/wsdl/ECS"
    portType="tns:RequestElevatorPT"/>
   <assign name="MoveElevator"/>
    <if name="IfRequestedFloorIsUp">
      <sequence name="Up">
        <receive name="ReadFloorInfo" partnerLink="ElevatorController"
        operation="RequestForElevator"
        xmlns:tns="http://j2ee.netbeans.org/wsdl/ECS"
        portType="tns:RequestElevatorPT"/>
       <assign name="CopyFloorInfo1"/>
        <invoke name="IncrementFloor" partnerLink="Elevator"
        operation="ElevatorMovingUp"
        xmlns:tns="http://j2ee.netbeans.org/wsdl/ECS"
        portType="tns:ElevatorMovementPT"/>
        <receive name="ReachRequestedFloor" partnerLink="Elevator"
        operation="StopMovement"
        xmlns:tns="http://j2ee.netbeans.org/wsdl/ECS"
        portType="tns:ElevatorMovementPT"/>
        <assign name="CopyFloorInfo2"/>
        <receive name="ReadFloorInformation" partnerLink="ElevatorController"
        operation="FloorLampDirectionUp"
        xmlns:tns="http://j2ee.netbeans.org/wsdl/ECS"
        portType="tns:RequestElevatorPT"/>
        <assign name="CopyLampIlluminationUp"/>
        <receive name="ReadFloorButtonInformation"
partnerLink="ElevatorController"
        operation="FloorButtonIlluminateOff"
        xmlns:tns="http://j2ee.netbeans.org/wsdl/ECS"
        portType="tns:RequestElevatorPT"/>
        <invoke name="InvokeElevatorController" partnerLink="Doors"
       operation="DoorsOpen"
       xmlns:tns="http://j2ee.netbeans.org/wsdl/ECS"
       portType="tns:DoorMovementPT"/>
      </sequence>
<else>
      <sequence name="Down">
          <receive name="ReadFloorInfo1" partnerLink="ElevatorController"
          operation="RequestForElevator"
          xmlns:tns=http://j2ee.netbeans.org/wsdl/ECS portType="tns:RequestElevatorPT"/>
         <assign name="CopyFloorInfo3"/>
          <invoke name="DecrementFloor" partnerLink="Elevator"
          operation="ElevatorMovingDown"
          xmlns:tns="http://j2ee.netbeans.org/wsdl/ECS"
          portType="tns:ElevatorMovementPT"/>
          <receive name="ReachRequestedFloor1" partnerLink="Elevator"
          operation="StopMovement"
          xmlns:tns="http://j2ee.netbeans.org/wsdl/ECS"
          portType="tns:ElevatorMovementPT"/>
          <assign name="CopyFloorInfo4"/>
         <receive name="ReadFloorInformation1" partnerLink="ElevatorController"
          operation="FloorLampDirectionDown"
          xmlns:tns="http://j2ee.netbeans.org/wsdl/ECS"
          portType="tns:RequestElevatorPT"/>
          <assign name="CopyLampIlluminationDown"/>
          <receive name="ReadFloorButtonInformation1" partnerLink="ElevatorController"
          operation="FloorButtonIlluminateOff"
          xmlns:tns="http://j2ee.netbeans.org/wsdl/ECS"
          portType="tns:RequestElevatorPT"/>
          <invoke name="InvokeElevatorController" partnerLink="Doors"
          operation="DoorsOpen"
          xmlns:tns="http://j2ee.netbeans.org/wsdl/ECS"
          portType="tns:DoorMovementPT"/>
        </sequence>
  </sequence>
```

Figure 2. Part of the BPEL Specification for the Elevator Control System

From these UML diagrams, BPEL specifications were created; more details can be found in [8, 17]. A part of the generated BPEL Specification is shown in Figure 2 which corresponds to the sequence diagram for serving the floor request in Figure 1.

The consistency between the BPEL specifications and UML diagrams captured were evaluated using the NFR Approach, which is discussed next.

III. THE NFR APPROACH

In order to evaluate consistency we apply a qualitative reasoning approach called the NFR Approach which is a goal-oriented approach and the goal here is to achieve good consistency between UML specifications and BPEL specifications. The NFR Approach views consistency as a non-functional requirement (NFR) that needs to be satisfied by the specifications. In order to apply the NFR Approach a structure called the Softgoal Interdependency Graph (SIG) is developed which will be used to qualitatively evaluate consistency. The SIG development consists of four steps:

1. Decomposition of the factors that affect consistency into an AND-OR-EQUAL graph: two factors (also referred to as softgoals) are related by an AND relation if both the factors are important to satisfy the parent factor; two factors are related by an OR relation is either of the factors is important to satisfy the parent factor; a factor is related to another by EQUAL relation if the child factor is important to satisfy the parent. Generally, factors are named using the convention Type [Topic1, Topic2…] where Type is the factor and Topic is the field of application of Type [15]; Topic is optional.
2. Determine the various features in the relationship between the two specifications such as, for example, completeness, comprehension, acceptability of the system by the users, and the like; we refer to these as the features of the specifications relationship (SpecRel from here on).
3. Determine the extent to which SpecRel features affect the leaf consistency factors identified in the first step by calculating the contributions the SpecRel features make to the consistency factors; the contributions can be one of four types: strongly positive (++) or MAKE, positive (+) or HELP, negative (-) or HURT, and strongly negative (--) or BREAK.
4. Capture the justifications for the contributions in step 3 so that a historical record of rationale for contribution changes is maintained.

It should be noted that the factor decomposition in step 1 may also assign priorities to some of the factors. Another point to note is that the contributions to consistency factors can either satisfice or deny that factor – satisficing is a term derived from economics and refers to relative satisfaction as opposed to absolute satisfaction. By using the satisficing attitude, the propagation rules permit collections of contributions to be given one level of satisficing. Once the SIG is computed, the propagation rules of the NFR Approach are applied to propagate the contributions in step 3 above to the top of the SIG to evaluate the root NFR softgoal, namely, consistency. Sample propagation rules are given below (in the following TYPE is used to stand for one of MAKE, HELP, HURT, or BREAK); more details may be found in [13, 15, and 16]:

R1. In the case of AND-related factors, if all child softgoals are TYPE-satisficed then the parent softgoal is TYPE-satisficed.
R2. In the case of OR-related factors, if all child softgoals are TYPE-satisficed then the parent softgoal is TYPE-satisficed.
R3. In the case of EQUAL-related softgoals (only one child) the parent is TYPE-satisficed if the child is TYPE-satisficed.

The result of this evaluation is that consistency score for each set of specifications will be qualitatively categorized into one of MAKE, HELP, HURT, or BREAK, with their ranking being:

MAKE (++) > HELP (+) > HURT (-) > BREAK (--),

where, MAKE refers to excellent consistency, HELP to good consistency, HURT to poor consistency, and BREAK to very poor consistency.

A. *Application of the NFR Approach*

The SIG developed for evaluating the consistency of specifications is shown in the upper part of Figure 3. The root NFR softgoal (cloud-shape at the top) is Consistency. We now discuss the application of 4 steps of NFR Approach to elevator control system for evaluating consistency in the following manner.

Step 1: The softgoal Consistency is decomposed, based on suggestions in the literature [9], into three child softgoals (OR-decomposition indicated by the double arc) - Quality, Reliability and Robustness. The factor Consistency is named Consistency [Mapping] using the convention Type [Topic1, Topic2…], where Consistency is the factor and Mapping is the field of application of Consistency. Similarly the child softgoals are named Quality [Mapping], Reliability [Mapping] and Robustness [Mapping]. Again, taking hints from a literature survey [10, 14], softgoal Quality is decomposed (AND- decomposition, indicated by single arc) into Maintainability and Performance. Likewise, the softgoal Reliability is further refined (AND-decomposition) into Failure Reportability and Design, and the softgoal Robustness is further refined (AND-decomposition) into Predictability, Time Consumption and Risk Assessment softgoals (topic is Mapping for these softgoals). Using the naming convention the sub softgoals Maintainability, Performance, Design and Predictability are named Maintainability [Mapping], Performance [Mapping], Design [Mapping] and Predictability [Mapping].

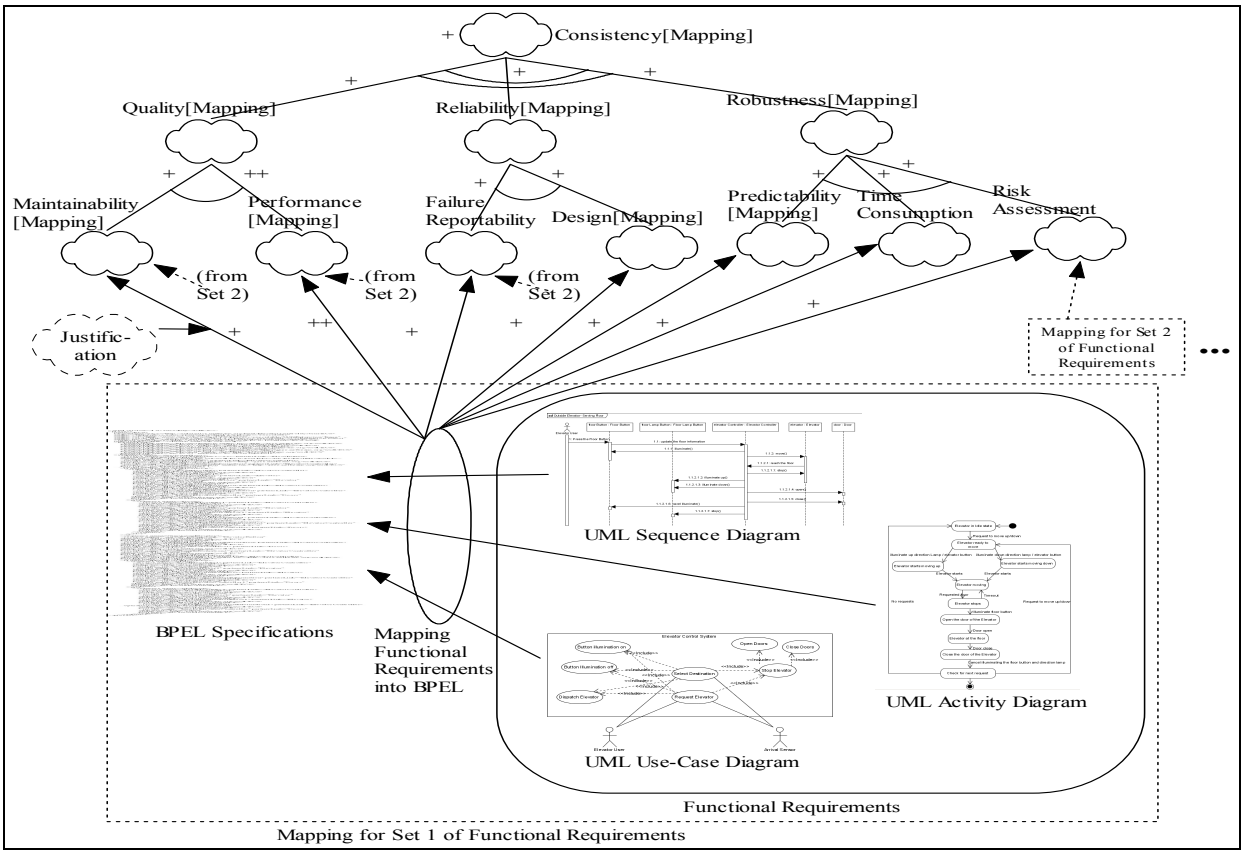

Figure 3. Softgoal Interdependency Graph for Evaluation of Consistency for Elevator Control System

Step 2: For each of set of functional requirements of the elevator control system, such as the set consisting of F1 through F4, the BPEL specifications were derived from the corresponding UML diagrams (a part of the BPEL specifications is shown in Figure 2). For the mapping between the diagrams and specifications for F1 through F4, the SpecRel had the features Maintainability, Performance, Failure Reportability, Design, Predictability, Time Consumption and Risk Assessment.

Step 3: It was easy to keep the BPEL specifications in sync with UML specifications (HELP (+) satisfices Maintainability), it was not time or space-consuming to perform and store this synchronization (MAKE (++) satisfices Performance), any errors in the relationship were easy to identify (HELP (+) satisfices Failure Reportability), the mapping between the specifications was straightforward (HELP (+) satisfices Design), changes needed to keep the two specifications in sync were easily predicted (HELP (+) satisfices Predictability), it was not time-consuming to keep the two specifications in sync (HELP (+) satisfices Time Consumption), and risk assessment was not necessary for the relationship between these two specifications (HELP (+) satisfices Risk Assessment).

Step 4: The rationale for contributions in Step 3 is captured by argumentative softgoals (shown in Figure 3 as "Justification" in a dashed-bordered cloud shape): thus "Justification" stands for "easy to keep BPEL specifications in sync with UML specifications since for each change in UML specifications a corresponding change, if needed, was performed for BPEL specifications". Similar justifications for all contributions in Step 3 are captured in the SIG to maintain a historical record of rationale for contributions.

We next applied the propagation rules of the NFR Approach [14, 15]. We traverse from the sub NFR softgoals namely Maintainability [Mapping], Performance [Mapping], Failure Reportability [Mapping], Design [Mapping], Predictability [Mapping], Time consumption and Risk Assessment to the root NFR softgoal Consistency. We observe a HELP (+) satisfice for Maintainability [Mapping] and a MAKE (++) satisfice for Performance [Mapping], and the child softgoals Maintainability [Mapping] and Performance [Mapping] are the AND-decomposition for the NFR softgoal Quality [Mapping]. So, from the rule of AND-decomposition, R1, we determine that the parent Quality [Mapping] is HELP satisficed. Based on R1 again, the parent NFR softgoals Reliability [Mapping] and Robustness [Mapping] are both HELP-satisficed since all their respective children are HELP-satisficed as well.

Next we observe that the NFR softgoals Quality [Mapping], Reliability [Mapping] and Robustness [Mapping] are the OR-decomposition for the parent NFR softgoal Consistency [Mapping]. Hence, from the rule of OR-decomposition, R2,

we concluded that the root NFR softgoal, namely, Consistency [Mapping], is HELP-satisfied as well, which means that SpecRel – Maintainability, Performance, Failure Reportability, Design, Predictability, Time Consumption and Risk Assessment are consistent. However, SpecRel features for another set of functional requirements may be different and they may impact the NFR softgoals differently – the NFR Approach allows considerations of all sets of mappings between functional requirements and their BPEL specifications and allows us to evaluate consistency of the entire set of mappings using a systematic method.

However, the biggest advantage of the NFR Approach application is that we are able capture any definition of consistency by means of the NFR softgoal decomposition in the upper part of the SIG and our evaluation will conform to our definition of the root NFR softgoal. We are also able to identify the reasons for poor (or good) consistency – in our example; we understand there is scope for improvement by taking necessary actions to change HELP's to MAKE's. Moreover, the rationale for each contribution to the leaf NFR softgoal is captured by an argumentative softgoal; the rationales help us determine the causes for poor (or good) consistency – in our example; we understand there is scope for improvement by taking necessary actions to change HELP's to MAKE's. Moreover, the rationale for each contribution to the leaf NFR softgoal is captured by an argumentative softgoal (shown in Figure 3 as "Justification" in a dashed-bordered cloud shape); the rationales help us determine the causes for poor scores. Moreover, the change history is preserved by different SIG versions – thus the development team needs to only study the SIG's and the rationales for understanding the current scores.

IV. CONCLUSION

Complex systems are systems that can be decomposed into subsystems. By their very nature complex systems' specifications are usually extensive and automated means of understanding their specifications will be extremely helpful to practitioners. BPEL has emerged as an important language for capturing functional requirements for software-intensive complex systems since BPEL specifications can be easily executed in an environment such as NetBeans [3]. However, since BPEL specifications are derived from the functional specifications, it is important that the two representations are consistent with each other so that the system being developed meets the client's needs.

In this paper we employ the NFR Approach to evaluate consistency between these two representations of software specifications. Our study indicates that the NFR Approach is very beneficial in evaluating the consistency since it provides justifications for the evaluations, retains a historical record of changes to the consistency, and can be evaluated on a continual basis during the development of the complex system.

For the future, we plan to develop a quantitative evaluation for this consistency so that numbers may be assigned. We also plan to apply the NFR Approach to other complex systems to ensure wider applicability. Another possibility for further research is tool development for automated evaluation. However, we believe the NFR Approach based evaluation of consistency will be significantly useful to practitioners in industry and academia.

ACKNOWLEDGMENT

The authors wish to thank the anonymous reviewers of this paper for their detailed and thorough comments which helped us significantly improve this paper.

REFERENCES

[1] A. Stoyenko, "Engineering complex computer systems: A challenge for computer types everywhere," *IEEE Computer*, vol. 28, no. 9, pp. 85-86, Sept. 1995.

[2] L. Zhang, W. Jiang, "Transforming business requirements into BPEL: a MDA-based approach to web application development," *IEEE Trans. Semantic Computing and Systems*, pp. 61–66, July 2008.

[3] F. Jennings and D. Salter, *Building SOA-based composite applications using NetBeans IDE*, Pakt Publishing, 2008.

[4] IEEE Std. 830-1993, IEEE Recommended Practice for Software Requirements Specifications, December 1993.

[5] J. R. Clymer, *Simulation-Based Engineering of Complex Systems*, 2nd ed., 2009, pp. 376–382.

[6] J. Coleman, C. Jones, I. Oliver, A. Romanovsky, and E. Troubitsyna, "RODIN (Rigorous open development environment for complex systems)," *Proceedings of 5th European Dependable Computing Conference*, EDCC-5 supplementary volume, pp. 23–26, April 2005.

[7] A. Shaban-Nejad, O. Ormandjieva, M. Kassab, and V. Harrslev, "Managing requirement volatility in an ontology-driven clinical LIMS using category theory," *International Journal of Telemedicine and Applications*, vol. 2009.

[8] A. Vemulapalli and N. Subramanian, "Transforming functional requirements from UML into BPEL to efficiently develop SOA-Based systems," *Proceedings of 8th International workshop on System/Software Architectures*, pp. 337–349, November 2009.

[9] N-L. Hsueh, P-H. Chu and W. Chu, "A quantitative approach for evaluating quality of design patterns," *Journal of Systems and Software*, vol. 81, pp.1430–1439, August 2008.

[10] K. P. Birman and B. B. Glade, "Reliability through consistency," *IEEE Trans. Software*, vol. 12, pp. 29–41, May 1995.

[11] C. A .R. Hoare, H. D. Mills and E. W. Dijkstra, "The roots of structured programming," *ACM SIGCSE Bulletin*, vol. 10, pp. 243–254, February 1978.

[12] M. Chechik and J. Gannon, "Automatic analysis of consistency between requirements and designs," *IEEE Trans. Software Engineering*. vol. 27, pp. 651–672, July 2001.

[13] L. Chung and N. Subramanian, "Process-oriented metrics for software architecture adaptability," *Proceedings of the International Symposium on Requirements Engineering*, IEEE Computer Press, Aug-Sep. 2001, pp. 310-311.

[14] G. Taguchi and D. Clausing, "Robust quality." *Harvard Business Review*, Jan-Feb. 1990, pp.65–75.

[15] L. Chung and N. Subramanian, "Software architecture adaptability: an NFR approach," *Proceedings of the 4th International Workshop on Principles of Software Evolution*, pp. 52–61, 2001.

[16] S. Supakkul and L. Chung, "Integrating FRs and NFRs: A use case and goal driven approach," *Proceedings of the 2nd International Conference on Software Engineering Research*, Management and Applications, pp. 30–37, May 2004.

[17] K. Mantell, "From UML to BPEL: model driven architecture in a web services world," http://www.ibm.com/developerworks/webservices/library/ws-uml2bpel/, September 2003.