A Requirements-Led Approach for Specifying QoS-Aware Service Choreographies: An Experience Report

Neil Maiden[1], James Lockerbie[1], Konstantinos Zachos[1], Antonia Bertolino[2], Guglielmo De Angelis[2], and Francesca Lonetti[2]

[1] School of Informatics, City University London, London, UK
[2] Istituto di Scienza e Tecnologie dell'Informazione "A. Faedo" CNR, Pisa, Italy
{N.A.M.Maiden@,James.Lockerbie.1@,kzachos@soi.}city.ac.uk,
{antonia.bertolino@,guglielmo.deangelis@,francesca.lonetti@}isti.cnr.it

Abstract. [Context and motivation] Choreographies are a form of service composition in which partner services interact in a global scenario without a single point of control. The absence of an explicitly specified orchestration requires changes to requirements practices to recognize the need to optimize software services choreography and monitoring for satisfaction with system requirements. **[Question/problem]** We developed a requirements-led approach that aims to provide tools and processes to transform requirements expressed on service-based systems to QoS-aware choreography specifications. **[Principal ideas/results]** The approach is used by domain experts to specify natural language requirements on a service-based system, and by choreography designers to adapt their models to satisfy requirements more effectively. Non-functional requirements are mapped to BPMN choreography diagrams as quality properties, using the Q4BPMN notation, that support analysis and monitoring facilities. **[Contribution]** We report the new integrated approach and provide lessons learned from applying it to a real-world example of dynamic taxi management

Keywords: service choreographies, requirements monitors, user task models, adaptive systems, quality properties, requirements-led life-cycle.

1 Introduction

Choreographies are a form of service composition in which, unlike orchestration, services interact to achieve a goal without a single point of control [1]. The increased flexibility of architectures based on choreographies can deliver more adaptive service-based systems that satisfy more ambitious requirements of certain types on these systems. However, we still need new techniques to design flexible choreographies that can be argued to satisfy system requirements, and new mechanisms for monitoring services invoked in these choreographies to show continued requirements satisfaction.

The traditional approach to service composition uses orchestration coordinators and arranges services according to a predetermined business logic and execution

C. Salinesi and I. van de Weerd (Eds.): REFSQ 2014, LNCS 8396, pp. 239–253, 2014.

order [2] based on design choices about the type and granularity of available services. This execution order and logic is often expressed as a workflow using notations such as BPEL [2]. Whilst service orchestration is a widely deployed form of system architecture, it can result in a failure to satisfy requirements in increasingly adaptive environments as the predefined execution order and/or business logic can be rendered invalid by changes to a service consumer's context. One advantage of service choreographies is that they impose fewer architecture-level constraints than orchestrations, and as a consequence have greater potential to deliver adaptive systems that continue to satisfy the evolving requirements on them. They make fewer design choices about the granularity of services to be invoked, an execution order for these services does not need to be specified, and business logic is specified independent of the services [3].

Requirements work is needed to specify the required behavior and qualities of choreography activities in a model. Quality of Service (QoS) has been acknowledged as a main concern in service-oriented computing (SOC) and in QoS-aware service composition. Relevant work in SOC includes quality-of-service ontologies and measurement [e.g. 4], however little research or practice traces service qualities back to the originating quality requirements on the systems. These specified behaviors and qualities are needed to guide local enactment of services and to enable the run-time monitoring of each choreography activity for continued requirements satisfaction. Indeed, this alternative paradigm for service technology creates new challenges, including how to:

1. Optimize the specification of choreography diagrams with respect to system requirements;
2. Associate specified system requirements with choreography activities in a choreography diagram;
3. Enhance choreography diagrams with quality properties that trace system requirements, to support analysis and monitoring facilities.

In this paper, we report results from the CHOReOS project (*www.choreos.eu*) to address the three challenges using a real-world dynamic taxi management example. The next section outlines the CHOReOS approach for specifying QoS-aware service choreographies. In sections 3 and 4 we report how this requirements approach uses user task models to generate a first-cut BPMN choreography diagram. Section 5 describes Q4BPMN (*http://labsedc.isti.cnr.it/tools/q4bpmn*) for extending BPMN models with specifications of different types of quality properties [5], and how these extended models can drive the instantiation of corresponding monitors. Section 6 presents lessons learned and the paper ends by looking at related work, reviewing current omissions in the approach, and outlining the next steps.

2 The CHOReOS Approach

Among the various challenges posed by the vision of the Future Internet (FI) [6], is how to provide user-centric processes to support the whole life cycle of SOC systems from their design, to their development, up to their maintenance and governance at run-time [7]. Requirements specification is a fundamental part of dealing with this

challenge. A distinctive feature of the FI vision affecting this activity is the active role of domain experts, who are intended to take the place of requirements analysts. The CHOReOS project tackled this challenge by elaborating an approach that considers the user as an active part in the choreography life cycle, and developed it from a business standpoint by leveraging realistic B2B and B2C scenarios provided by the industrial project partners e.g. the dynamic taxi management example in this paper. The result was a domain-expert centric approach supporting non-technical users to specify the desired choreography with respect to their business goals, requirements and quality expectations [8]. An overview of the approach is depicted in Figure 1.

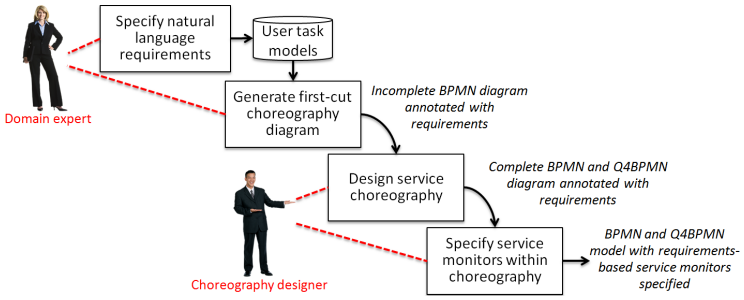

Fig. 1. The key stages of the user-centric requirements-led approach

A domain expert specifies the requirements on a future service-based software system in natural language using the new CHOReOS Requirements Tool. To maximize the uptake of the approach, we decided not to adopt more formal requirements specification techniques that would have necessitated trained analysts. Instead, the primary input to the approach is a small set of natural language requirements that need domain knowledge rather than analytic training to write (see Sections 3.1 and 3.2). To generate a first-cut choreography diagram that satisfies the requirements, the expert uses procedures adapted from previous work to retrieve user tasks models that match the requirements (see Section 3.3). The Requirements Tool then automatically reasons with retrieved models to generate a first-cut model of an under-constrained choreography diagram that can satisfy the matched requirements (Section 4). The choreography designer can select and refine the generated BPMN choreography activities that are the best-fit abstractions of the requirements problem to complete the design of the choreography. Finally, the requirements mapped to the functional BPMN choreography diagram can be used to specify non-functional systems engineering properties using the existing Q4BPMN notation, an extension of BPMN (see Section 5).

If it is applied successfully, the approach will transform functional and quality requirements on a system expressed in natural language into a single service choreography specification that is expressed using BPMN and Q4BPMN, and optimized to satisfy the requirements and the domain constraints extracted from matched user task models. In addition, the specification of such Q4BPMN properties of the choreography activities enables the definition of software modules monitoring the fulfillment of QoS constraints directly traced from the requirements [9].

3 From Natural-Language Requirements to First-Cut Choreography Specifications

An objective of our requirements-based approach was for it to be usable in the largest number of service-oriented projects possible. Natural language continues to be the most used form of requirements expression [e.g. 10], so our approach assumes that functional and quality requirements to be satisfied by the service choreography will be expressed in natural language by domain experts rather than trained analysts. It assumes that a domain expert gathers requirements from the consumers of its services – the domain expert acts as a surrogate for service consumers such as travelers needing a taxi, who are unlikely to participate directly in a requirements process.

The domain expert is supported in 3 stages. In the first stage, they are guided to express system requirements with associated qualities. In the second stage, the expert is guided to cluster requirements on a single choreography. In the third stage, the cluster is matched to a catalogue of user task models to guide the initial design of the requirements-driven service choreography.

3.1 Expressing Structured Natural Language Requirements

Whilst our experiences have shown that domain experts with minimum training can write functional requirements in natural language [e.g. 11], we do not believe that they can express measurable quality requirements such as performance and reliability as effectively. Therefore we developed requirements writing guidelines based on the notion of a qualifier from the anatomy of well-written requirements [10] to transform the functional root of a requirement into one or more quality requirements. These guidelines are built into tool support that guides the domain expert to add one or more qualifiers indicative of different qualities to each specified functional requirement.

The CHOReOS Requirements Tool provides support to the domain expert to define four different qualities considered the most relevant for the project scenarios: accuracy, reliability, performance and security. It tags and parses the functional requirements text written by the domain expert to extract keywords and synonyms, then applies simple rules to infer which of the four qualities, if any, might be associated with the described requirement. The expert is then prompted to consider adding the inferred qualifier to the requirement, which s/he can accept or reject. For example, the functional requirement written by a domain expert in taxi management: *The user shall receive a prompt notification of how long it will take for their taxi to arrive* is parsed to infer a possible performance qualifier based on the keyword *prompt*. Next, to make each quality requirement measurable [12], the expert simply indicates the quality rating on a Likert scale of 1 (very low) to 5 (very high). The expert can then apply additional qualifiers if required, however the tool ensures that only one quality can be selected as the most important, as shown in Figure 2. Each response on the scale for each quality type has been associated with a predefined range of measures to determine satisfaction with the requirement. We describe this in detail in Section 5.2.

The output from this stage is a set of structured natural language requirements on systems that can be implemented as service choreographies.

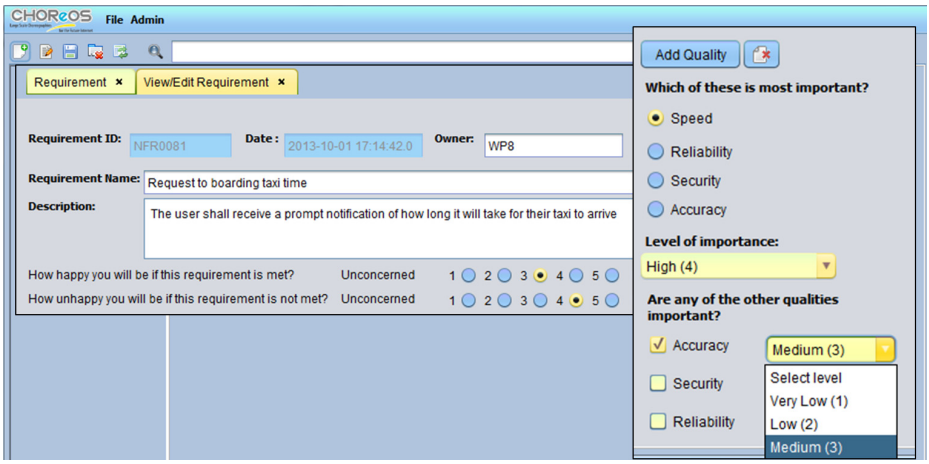

Fig. 2. Expressing quality requirements in the *CHOReOS Requirements Tool*

3.2 Clustering Requirements

Domain experts write requirements on systems rather than service choreographies. However, before these requirements can be mapped onto choreography activities, work is needed to cluster similar requirements that will map onto a single choreography and its elements such as activities and roles. This is because a domain expert acts as the ghost author of requirements from multiple service consumers, and more than one requirement can specify the same or a similar function or quality. Therefore, to optimize the specification of a single choreography, similar and overlapping requirements need to be discovered and handled.

To do this, the Requirements Tool provides the capability to calculate the semantic similarity between any pair of requirements in the set of requirements, similar to a linguistic approach for large-scale requirements management reported in [13]. The algorithm for computing semantic similarity is based on one element of an existing algorithm that computes measures of similarity between natural language requirements and service descriptions during service discovery [14]. The Requirements Tool invokes the algorithm as a third party service, taking the selected requirement and computing a measure of its similarity with every other requirement in the set. A description of the algorithm and an evaluation of its effectiveness are provided in [14].

Figure 3 shows an example of a cluster of requirements on the `Request Taxi` choreography. The similarity algorithm function reorders the *Requirements List* table according to a match score (the most relevant at the top) and the expert can simply select and add the requirements into the *Requirements Cluster*. Similarly, there is also a keyword search that moves the matched requirements to the top of table.

The output from this stage is a cluster of system requirements that are matched to user task models to inform the design of a first-cut service choreography.

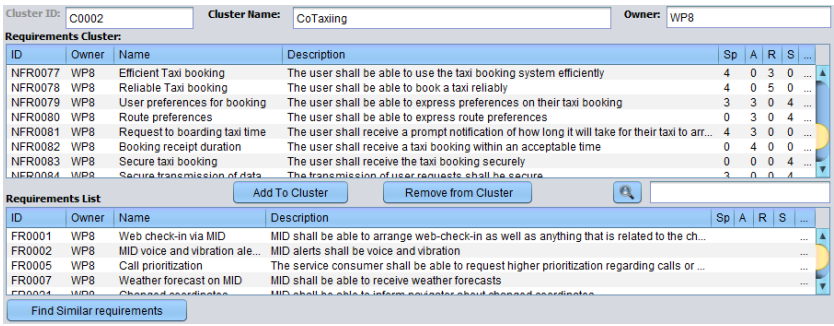

Fig. 3. A completed requirements cluster for the *taxi management* example

3.3 User Task Models for Choreography Specification

A user task model is a description of the structured activities that are often executed by a user during the interaction with a system in its contextual environment to attain goals [15]. Research exploiting user task models to design service-based systems is scarce. For example, Paterno et al. [16] delivered an environment to support tasks and services matching with CTT task models to develop user interfaces but the association between tasks and services was manually established and not cost-effective. Ruiz et al. [17] proposed a method for designing web services that analyzed user task descriptions to identify a web application's required operations. Unlike Paterno's work, this approach was automated but did not include activities such as discovering, selecting and composing software services specific to the design of service-based applications.

Our approach utilizes class-level user task models expressed in the ConcurTaskTrees (CTT) formalism [15] to apply an engineering approach to user task modeling. The precise semantics of CTT enable greater automated guidance with which to specify choreographies, whilst the CTT models can bridge the semantic gap between natural language specifications and formal choreography specifications. For the approach, we built upon the existing library of domain-independent CTT models developed in S-Cube, the EU-funded Network of Excellence for Software Services (*www.s-cube-network.eu/*). We used commonly occurring class-level tasks such as requesting and booking and extended them with knowledge from the applied domains. The CTT models were developed manually in a process reported in [18].

Figure 4 depicts the CTT model *Request taxi*. This CTT model follows a common 3-tier structure we specified. The top-level user goal, *Request taxi*, is decomposed into intermediate level tasks such as *Retrieve data*. In turn, this task is decomposed into the application task level. Application tasks include *detect current location*; *retrieve date and time*; and optional tasks *retrieve preferences* and *retrieve taxi membership*. A description of the full semantics of CTT models can be found in [19].

Important to this design guidance, is a set of mappings between CTT and BPMN semantics that we use to generate a first-cut BPMN choreography diagram. The mapping rules are simple, and we can demonstrate them using the *Request taxi* CTT model. The sub-task *Retrieve Data* occurs concurrently (|[]|) with *Provide taxi request details.* These two tasks enable (>>) the sub-task *Submit query*. CTT semantics define enables (>>) sub-tasks as interleaved, so our rule specifies that both sub-tasks can be

undertaken in the same choreography activity. The *Submit query* sub-task enables and passes on information ([] >>) to the *Process query* sub-task. CTT semantics indicate that information is passed between the sub-tasks, so our rule specifies a boundary between choreography activities across which messages are exchanged. Finally, the CTT operator choice ([]) denotes a split in the tasks followed, so our rule specifies a split in the flow of the choreography, represented in BPMN as a gateway or loop.

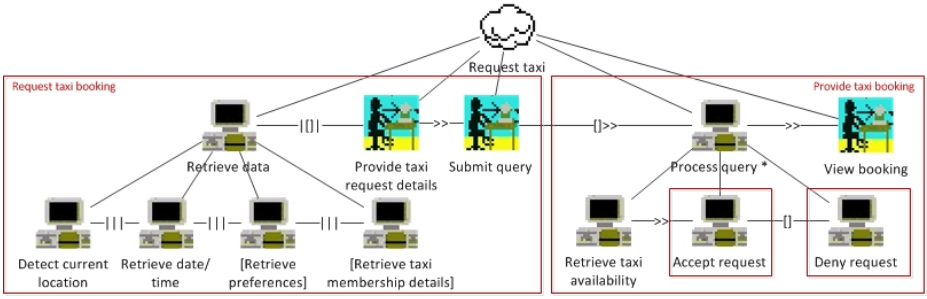

Fig. 4. The CTT specification of the user task *Request Taxi*

Returning to the Requirements Tool, the cluster of requirements is fired at the TEDDiE service, which applies sophisticated information retrieval techniques to identify CTT models relevant to the requirements. Of course, the approach's effectiveness depends upon the accurate automatic retrieval of user task models that match each requirement from the cluster to each CTT model. An evaluation of the retrieval process can be found in [18]. The normal course output from a single invocation of the TEDDIE engine is retrieval of one or more CTT models from the library, and each model is mapped to one or more individual requirement statements in the cluster. First evaluations have revealed that, because the CTT models in the library express class-level tasks rather than more concrete and hence complete business processes, most requirements clusters are likely to match to more than one CTT model.

The output is documented in a XML file that specifies the data with which to generate a first-cut BPMN choreography diagram.

4 Generating a First-Cut Choreography as a BPMN Choreography Diagram with Associated Requirements

Our approach uses the Business Process Model and Notation (*www.bpmn.org*), which is an emerging standard for business process modeling and the specification of service choreographies. The choreography designer designs BPMN choreographies using the MagicDraw visual modeling tool (*www.nomagic.com*), which we configured to accept the XML files output from the Requirements Tool. As a result, the choreography designer receives explicit requirements-based guidance for designing a service choreography based on a first-cut template model annotated with requirements information.

A first-cut BPMN choreography diagram generated for the taxi example is shown in Figure 5. One consequence of the BPMN formalism being more complete than the CTT formalism is that each choreography diagram automatically generated from CTT

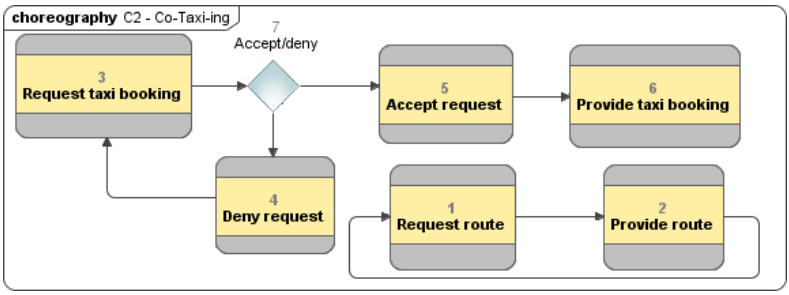

Fig. 5. Automatically generated first-cut BPMN choreography diagram in *MagicDraw*

models will be incomplete. As well as integrating model elements generated from more than one CTT model, the choreography designer may need to add choreography tasks, as well as define the instance-level participant names in the choreography.

All of the design refinement should be directly informed by the requirements linked to each choreography element, presented in MagicDraw as a requirements-choreography task matrix. The matrix maps each original requirement to one or more choreography tasks in the choreography diagram. The designer can use the requirement traces imported from each matched user task model to refine the specification of the required levels of quality-of-service, and from them support the application of quality properties. An example of a requirements matrix is depicted in Figure 6.

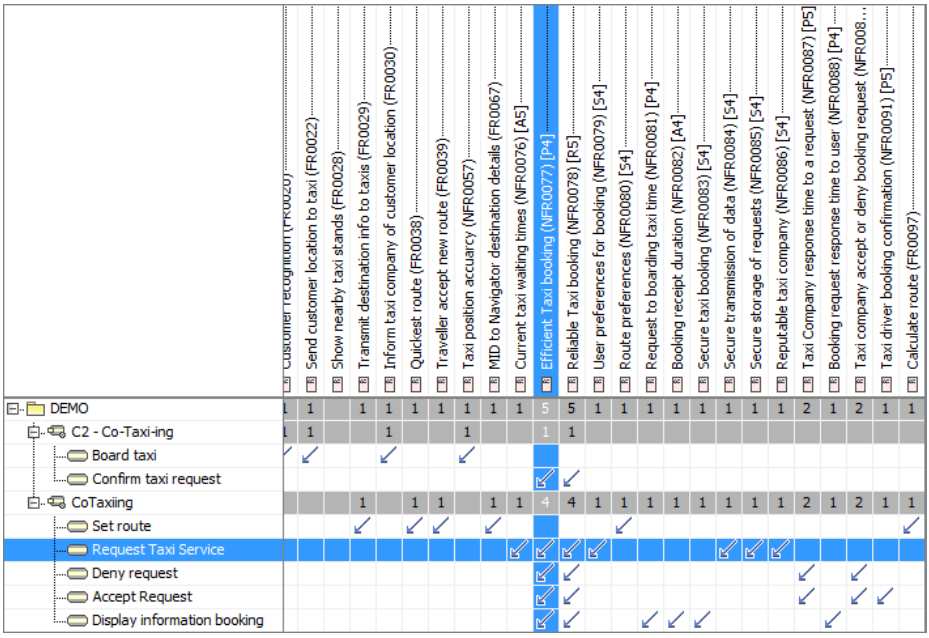

Fig. 6. Completed matrix mapping system requirements to choreography tasks in *MagicDraw*

The output of this stage is a first version of a service choreography diagram specified using the constraints derived from matching user task models, and with each element of the choreography that is annotated with requirements information derived from the original system requirements from domain experts and service consumers. For reference, a final elaborated version of the first-cut choreography diagram shown in Figure 5 can be found at *http://labsedc.isti.cnr.it/tools/q4bpmn/co-taxing*.

5 Monitorable Service Qualities from Requirements

To express qualities on service choreographies we used Q4BPMN (Quality for BPMN), a semi-formal notation for specifying quality annotations at the choreography level [5]. It allows designers to extend BPMN models with quality properties that the services entering the choreography will have to abide by. With Q4BPMN, the choreography designer can state the quality requirements for the choreography and its roles at the same level of abstraction of the tasks' flow without the need for additional models. The explicit introduction of non-functional constraints within a choreography specification supports the verification and validation of their impact on the overall quality requirements. At the same time, prospective participants can use this information to understand the quality level required on their part.

5.1 The Q4BPMN Notation

Q4BPMN is an implementation of the Property Meta-Model (PMM) with which to specify the quality properties, metrics and observable events of a system [20]. Q4BPMN is implemented within MagicDraw as a design tool, and was conceived to support the specification of non-functional properties within a service choreography expressed in the BPMN notation. It provides designers and analysts the means to annotate a choreography diagram with quality requirements [5]. Specifically, Q4BPMN enables the definition of non-functional systems engineering properties that can be directly linked either to a single task (i.e. «Q4Task»), specific participant of a task (i.e. «Q4Participants»), or whole choreography (i.e. «Q4Choreography») [21].

Q4BPMN supports several kinds of properties, which correspond to the class of properties inherited from the PMM meta-model. Specifically, it currently defines four classes of properties: (i) dependability properties concerning the availability of the system and failure rates; (ii) performance properties related to time, mainly from a software and human interaction point of view; (iii) security properties related to encryption of operations and trustworthiness of the business activities; and (iv) accuracy properties that relate to time and space dimensions.

5.2 Mapping Requirements to Q4BPMN

Although Q4BPMN provides the means for expressing quality properties within a BPMN choreography diagram, it does not bind to any specific methodology for defining which values, dimensions or context characterize these properties. In this sense, as

commonly happens during their specification, non-functional requirements can conceal underspecified aspects that can allow many different interpretations depending on the context these desired system properties are offered [22]. For example, a user expressed performance requirement such as *NFR0077: The user shall be able to use the taxi booking system efficiently* is useful in the early stages of the requirement elicitation process, as this kind of natural language requirement can be easily understood by non-technical stakeholders (e.g. final users). However, quantifying such a high level user requirement could imply a different interpretation depending on the context and the functionalities intended to impact on it.

To address this, the non-functional requirements and their various potential interpretations were mitigated by quantifying each abstract property in a set of application contexts. From the analysis of our application domains we identified three major concerns representing the different contexts for achieving the quality goals intended by the elicited non-functional requirements. Specifically, such concerns are:

1. *Software System:* the properties specified under this concern represent those quality attributes quantifying either the behavior of software components, or their interactions;
2. *Human–Computer Interaction*: the properties specified under this concern represent those quality attributes quantifying any interactions between a human and any part of the considered software system;
3. *Business Activities:* the properties specified under this concern represents those quality attributes quantifying the admissible constraint used in order to characterize the activities of both the whole system (i.e. software + human related activities), and its actors from a business perspective.

The model shown in Figure 7 shows the requirements to Q4BPMN mapping which includes these concerns, modeled as agent-based quality goals. The left side of the model shows a user expressed systems requirement and the four main qualities that can be associated with it, indicated on scales of 1 to 5 as described in Section 3.1. These user expressed systems requirements correspond directly to four user quality goals on the wider service-based system – accuracy, dependability, performance and security. These goals are then decomposed into the 3 concerns described above: Software System, HCI and Business, and shown as agent-based quality goals. Finally, the model shows the definition of the qualitative properties that could be used by the agent-based goals. In this case, we have instantiated the model for our application to the taxi management scenario. Specifically, for each of the properties we identified: the type and the dimension it addresses; but also its name and the values prescribed for each specific satisfaction level.

In order to define the abstract properties, we drew upon research in disciplines such as software reliability and human-computer interaction to determine prototypical measures of different qualities that the expert uses to refine the requirement measure. For example, we considered the Secure Sockets Layer (SSL) protocol for security encryption-type requirements and Miller's work [23] describing threshold levels of human attention for HCI time performance-type requirements. We then utilised the expert input available to us in our taxi management application domain to review the provided definitions and quantify the predefined ranges.

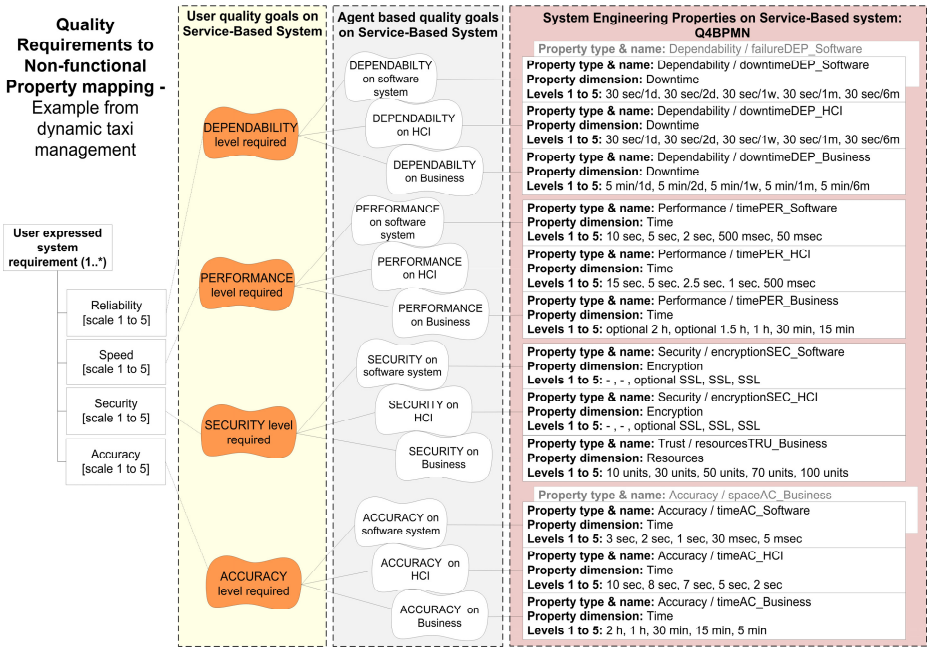

Fig. 7. Non-functional requirements mapping to quality properties in Q4BPMN

Table 1, for example, shows mappings between two original system requirements and the quality properties they relate to. In the first, the traveller details are required to be transmitted with a performance level of 5, which translates to the software time performance property. The required time performance for all software interactions within this choreography task is therefore quantified as `50msec`. However, the second requirement is mapped to a participant property limited specifically to the *Taxi Company* operating in the choreography task. It specifies that the taxi company shall be reputable, which maps a onto a trust property related to the business security quality goal. The instantiation of these properties reflect the specification introduced in Figure 7.

Table 1. An example of mapping original system requirements to quality properties

Choreography element, Name	**Original requirement Name, ID, description, quality**	**Quality Property**
Task Request Taxi Service	MID send customer details (NFR0012): *MID shall be able to transmit traveller (customer) details to the taxi company* [Performance 5]	`timePer_Software_L5` ▪ isHard = true ▪ metrics – DefaultMaxDurationMetric ▪ nature = PRESCRIPTIVE ▪ operator = LESS EQUAL ▪ propertyClass – PERFORMANCE ▪ unit = “msec” ▪ value = “50”
Participant property Taxi Company	Reputable taxi company (NFR0086): *The taxi company shall have an established reputation* [Security 4]	`taxiCompanyBusinessTrust` ▪ NF Properties = resourcesTRU_Business_L4 ▪ ParticipantRef = Taxi Company

Once complete, the quality properties into which the requirements are mapped can be used to inform the generation of property-based service monitors.

5.3 Software Monitors from Q4BPMN Specifications

Software monitors can be accurately developed to monitor for specified qualities of a service choreography at run-time. A monitor is *a software system that observes a target system's behavior for qualities of interest such as satisfying the target system's requirements* [24]. Effective monitoring is key for adaptation to ensure that the target system can bind to and invoke new services in new contexts to ensure continued requirements satisfaction. Robinson distinguishes software and requirements monitors. He defines a requirements monitor as a software component that *determines the requirements status from a stream of significant input events* [24], where the events are observed and recorded by software monitors.

Our approach explicitly supports the choreography designer to map QoS requirement into monitorable properties on a choreography model, the individual choreography tasks in the model, and on the roles in a task. By leveraging on generative techniques within the context of the model-driven engineering, the Q4BPMN properties support the synthesis of QoS monitoring modules of the service choreography. Specifically, each generated monitoring module determines whether a property associated to a quality requirement is satisfied using observed data and messages. Given the need to map requirements of different types onto different types of event filters, our approach uses a flexible event-based monitoring infrastructure tailored to observe and analyze the behavior of distributed systems and services [25]. The reference implementation of such event-based monitor includes a complex event-processing engine based on Drools Fusion [26]. The details about the model-to-code transformation process of the Q4BPMN properties in presented at length in [27].

Our use of Q4BPMN is related to current requirements metrics such as Planguage, the keyword-driven language for writing measurable quality requirements [12]. Planguage's use of measures and metrics is similar to the Q4BPMN's use of abstract, descriptive or prescriptive properties [27], however we believe that the grounding of these property types in observable data that can be collected using software monitors offers it a distinct advantage for service-based systems.

6 Lessons Learned

We return to the 3 challenges reported earlier to assess whether applying our approach helped to resolve them and report the most important lessons learned:

Optimize the specification of choreography diagrams with respect to system requirements. For the dynamic taxi management scenario, 97 system requirements were successfully specified and used to generate meaningful first-cut choreography diagrams. The simple interface of the requirements tool enabled system requirements to be specified effectively without training, and the similarity algorithm helped the user cluster requirements for specifying a single choreography. However, there was a

usability concern with the similarity function, as matching the selected requirement to the other 96 requirements took over 5 minutes using a standard laptop on the City University London network. An option for improving the performance of this function is to store the match results in a requirements matrix to remove the need for invoking the similarity algorithm web service for every requirement pair each time it is run.

The requirements specified retrieved the domain specific *Request taxi* CTT model and the generic *Calculate route* model. Minor refinements were made to the generated choreography tasks, shown in figure 5, and the final choreography tasks listed in Figure 6. Additional tasks included *Confirm taxi request* and *Board taxi* situated at either end of the process flow. While the use of CTT models provided valuable design guidance for the choreography design in this example, varying degrees of success were experienced for other choreographies. For example, in the context-aware traffic management choreography, the first-cut BPMN model elements generated by the retrieved CTT models *Provide traffic information* and *Calculate route* were either discarded or revised beyond recognition. Therefore, further work is needed to expand the catalogue of CTT models and to evaluate the effectiveness of the process in future case studies.

Associate specified system requirements with choreography activities in a choreography diagram. In our example, 38 out of the 40 clustered requirements were automatically mapped to choreography tasks and provided a useful starting point for the user. However, our approach can only match requirements to each retrieved CTT model, therefore matched requirements are automatically allocated to the first choreography task generated from each CTT model. Future work on the TEDDiE service to match the requirements to specific choreography tasks would improve the process and reduce the level of human input required. Also, it was evident that users need to be better supported where requirements are relevant to more than one choreography task in the model. As exemplified by the requirement NFR0077, which is mapped across 5 of the choreography tasks (see Figure 6), high level user requirements could imply different interpretations depending on the context and the functionalities intended to impact on them. One possible way of addressing this would be to use satisfaction arguments, as defined in [29], to provide a means to reason about the relationships between user expressed quality requirements and lower-level system requirements. We have already made an initial attempt to implement satisfaction arguments in the Requirements Tool (see [30]), but we need to explore effective ways of implementing this, or a similar approach, in the MagicDraw modeling environment.

Enhance choreography diagrams with quality properties that trace system requirements, to support analysis and monitoring facilities. Although the Q4BPMN notation already existed, this was the first time that we traced the originating set of system requirements through the choreography specification process. Reconciling the user expressed quality scores with actual values for the quality properties was a challenge. This required a significant level of domain expert input as ultimately context was everything and generic values reported in literature were not necessarily suitable.

Finally, it is worth mentioning the limitation that there is no backwards compatibility between the MagicDraw environment and the CHOReOS Requirements Tool. This restricts the possibility of revising the originating requirements without starting the process again and is an area that needs to be addressed in future.

7 Conclusion and Future Work

In this paper we report an integrated approach for designing service choreographies from system requirements to deliver more adaptive software systems. We integrated new and existing work from different sub-disciplines to develop a pragmatic solution to a pressing problem – how to engineer increasingly adaptive service-based software. We believe the combination of techniques for natural language requirements expression, user task models, quality model extensions to business processes, and transformations of these models to construct requirements-based software monitors, is unique. Not only have we developed an end-to-end approach for generating service-based systems that can be traced to their originating system requirements, but also we have developed an integrated toolkit based on BPMN modeling in MagicDraw.

Of course the approach we presented is not complete – it has some important omissions. One is the lack of discovery techniques to select and bind candidate services to choreographies at run-time. CHOReOS has developed such techniques linked to software service repositories [28]. Another is the lack of support to develop service-level agreements from requirements. Such agreements are needed to manage contractual relationships with the providers of the services invoked in choreographies, and should be derived from requirements. Although the approach provides the foundations for requirements-led development of SLAs, it still has to be extended it to deliver it.

Although we successfully applied the methods and tools in the demonstrated example, the next stage of our work is to evaluate the approach formatively in the development of other service-based systems. We plan to report results from these formative evaluations in future work.

Acknowledgment. The research is supported by CHOReOS project n° 257178 of the FP7 European program: FP7-ICT-2009-5.

References

1. Peltz, C.: Web Services Orchestration and Choreography. IEEE Computer 36(10), 46–52 (2003)
2. Ouyang, C., Verbeek, E., van der Aalst, W.M.P., Breutel, S., Dumas, M., Hofstede, A.H.M.: Formal Semantics and Analysis of Control Flow in WS-BPEL. Science of Computer Programming 2(3), 162–198 (2007)
3. Ben Hamida, A., et al.: An Integrated Development and Runtime Environment for the Future Internet. In: Álvarez, F., et al. (eds.) FIA 2012. LNCS, vol. 7281, pp. 81–92. Springer, Heidelberg (2012)
4. Sawyer, P., Hutchinson, J., Walkerdine, J., Sommerville, I.: Faceted Service Specification. In: Proceedings SOCCER Workshop at RE 2005 Conference, Paris (2005)
5. Bartolini, C., Bertolino, A., Ciancone, A., De Angelis, G., Mirandola, R.: Quality Requirements for Service Choreographies. In: Proceedings WEBIST 2012, pp. 143–148 (2012)
6. ERCIM News. Special Theme: Future Internet Technology. Number 77 (April 2009)
7. Shaw, M.: The Challenge of Pervasive Software to the Conventional Wisdom of Software Engineering. In: Keynote speech at ESEC/FSE (August 2009), `http://www.esec-fse-2009.ewi.tudelft.nl/downloads/ESECFSE09-shaw.pdf`
8. Autili, M., Di Ruscio, D., Inverardi, P., Lockerbie, J., Tivoli, M.: A Development Process for Requirements Based Service Choreography. In: Proceedings RESS 2011, pp. 59–62 (2011)

9. Bartolini, C., Bertolino, A., Ciancone, A., De Angelis, G., Mirandola, R.: Apprehensive QoS Monitoring of Service Choreographies. In: Proceedings SAC 2013, pp. 1893–1899 (2013)
10. Alexander, I., Stevens, R.: Writing Better Requirements. Addison-Wesley (2002)
11. Maiden, N.A.M., Jones, S.V., Manning, S., Greenwood, J., Renou, L.: Model-Driven Requirements Engineering: Synchronising Models in an Air Traffic Management Case Study. In: Persson, A., Stirna, J. (eds.) CAiSE 2004. LNCS, vol. 3084, pp. 368–383. Springer, Heidelberg (2004)
12. Gilb, T.: Competitive Engineering: A Handbook For Systems Engineering, Requirements Engineering, and Software Engineering Using Planguage. Elsevier (2005)
13. Nattoch Dag, J., Gervasi, V., Brinkkemper, S., Regnell, B.: A linguistic engineering approach to large-scale requirements management. IEEE Software 22(1), 32–39 (2005)
14. Zachos, K., Maiden, N.A.M., Zhu, X., Jones, S.: Discovering Web Services To Specify More Complete System Requirements. In: Krogstie, J., Opdahl, A.L., Sindre, G. (eds.) CAiSE 2007 and WES 2007. LNCS, vol. 4495, pp. 142–157. Springer, Heidelberg (2007)
15. Paterno, F., Santoro, C.: Preventing User Errors by Systematic Analysis of Deviations from the System Task Model. International Journal of Human-Computer Studies 56(2), 225–245 (2002)
16. Paterno, F., Santoro, C., Spano, L.D.: User Task-based Development of Multi-device Service-oriented Applications. In: Proceedings of the International Conference on Advanced Visual Interfaces, Roma, Italy (2010)
17. Ruiz, M., Pelechano, V., Pastor, O.: Designing Web Services for Supporting User Tasks: A Model Driven Approach. In: CoSS International Workshop on Conceptual Modeling of S-oSS, pp. 193–202 (2006)
18. Zachos, K., Kounkou, A., Maiden, N.A.M.: Exploiting Codified User Task Knowledge to Discover Services at Design-Time. IJSSOE 3(2), 30–66 (2012)
19. Paterno, F., Mancini, C., Meniconi, S.: ConcurTaskTrees: A Diagrammatic Notation for Specifying Task Models. In: Proceedings of the IFIP TC13 International Conference on Human-Computer Interaction, pp. 362–369 (1997)
20. Di Marco, A., Pompilio, C., Bertolino, A., Calabrò, A., Lonetti, F., Sabetta, A.: Yet another meta-model to specify non-functional properties. In: Proceedings QASBA 2011, pp. 9–16 (2011)
21. Bartolini, C., Bertolino, A., Ciancone, A., De Angelis, G., Mirandola, R.: Non-functional analysis of service choreographies. In: Proceedings PESOS 2012, pp. 8–14 (2012)
22. Pohl, K.: Requirements Engineering - Fundamentals, Principles, and Techniques. Springer (2010)
23. Miller, R.B.: Response time in man-computer conversational transactions. In: Proceedings of the Joint Computer Conference, pp. 267–277. ACM, New York (1968)
24. Robinson, W.: A Roadmap for Comprehensive Requirements Monitoring. IEEE Computer, 64–72 (May 2010)
25. Bertolino, A., Calabrò, A., Lonetti, F., Sabetta, A.: GLIMPSE: a generic and flexible monitoring infrastructure. In: Proceedings EWDC 2011, pp. 73–78 (2011)
26. Drools Fusion: Complex Event Processor, `http://www.jboss.org/drools/drools-fusion.html`
27. Bertolino, A., Calabrò, A., Lonetti, F., Di Marco, A., Sabetta, A.: Towards a Model-Driven Infrastructure for Runtime Monitoring. In: Troubitsyna, E.A. (ed.) SERENE 2011. LNCS, vol. 6968, pp. 130–144. Springer, Heidelberg (2011)
28. Ali, M., De Angelis, G., Polini, A.: ServicePot – An Extensible Registry for Choreography Governance. In: Proceedings SOSE 2013, pp. 113–124 (2013)
29. Hammond, J., Rawlings, R., Hall, A.: Will it work? In: Proceedings 5th IEEE International Symposium on Requirements Engineering, pp. 102–109. IEEE Computer Society (2001)
30. `http://www.choreos.eu/bin/view/Documentation/Requirements_Tool`