Systematic Scalability Modeling of QoS-aware Dynamic Service Composition

LETICIA DUBOC, La Salle - University Ramon Lull, Spain
RAMI BAHSOON, University of Birmingham, United Kingdom
FAISAL ALREBEISH, King Abdulaziz City for Science and Technology, SA
CARLOS MERA-GÓMEZ, ESPOL Polytechnic University, Escuela Superior Politécnica del Litoral, ESPOL, Ecuador
VIVEK NALLUR, University College Dublin, Ireland
RICK KAZMAN, Software Engineering Institute - Carnegie Mellon University and University of Hawaii
PHILIP BIANCO, Software Engineering Institute - Carnegie Mellon University
ALI BABAR, The University of Adelaide, Australia
RAJKUMAR BUYYA, The University of Melbourne, Australia

In Dynamic Service Composition (DSC), an application can be dynamically composed using web services to achieve its functional and Quality of Services (QoS) goals. DSC is a relatively mature area of research that crosscuts autonomous and services computing. Complex autonomous and self-adaptive computing paradigms (e.g., multi-tenant cloud services, mobile/smart services, services discovery and composition in intelligent environments such as smart cities) have been leveraging DSC to dynamically and adaptively maintain the desired QoS, cost and to stabilize long-lived software systems. While DSC is fundamentally known to be an NP-hard problem, systematic attempts to analyze its scalability have been limited, if not absent, though such analysis is of a paramount importance for their effective, efficient, and stable operations.

This article reports on a new application of goal-modeling, providing a systematic technique that can support DSC designers and architects in identifying DSC-relevant characteristics and metrics that can potentially affect the scalability goals of a system. The article then applies the technique to two different approaches for QoS-aware dynamic services composition, where the article describes two detailed exemplars that exemplify its application. The exemplars hope to provide researchers and practitioners with guidance and transferable

The research leading to these results has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No. 712949 (TECNIOspring PLUS) and from the Agency for Business Competitiveness of the Government of Catalonia.

Authors' addresses: L. Duboc, La Salle - University Ramon Lull, Spain; email: l.duboc@salle.url.edu; R. Bahsoon (corresponding author), University of Birmingham, School of Computer Science, Edgbaston, Birmingham, B15 2TT, United Kingdom; email: r.bahsoon@cs.bham.ac.uk; F. Alrebeish, King Abdulaziz City for Science and Technology, SA; email: frebeish@kacst.edu.sa; C. Mera-Gómez, ESPOL Polytechnic University, Escuela Superior Politécnica del Litoral, ESPOL, Facultad de Ingeniería en Electricidad y Computación, Campus Gustavo Galindo Km 30.5 Vía Perimetral, P.O. Box 09-01-5863, Guayaquil, Ecuador; email: cjmera@espol.edu.ec; V. Nallur, University College Dublin, Ireland; email: vivek.nallur@ucd.ie; R. Kazman, Software Engineering Institute - Carnegie Mellon University and University of Hawaii; email: kazman@hawaii.edu; P. Bianco, Software Engineering Institute - Carnegie Mellon University; email: pbianco@sei.cmu.edu; A. Babar, The University of Adelaide, Australia; email: ali.babar@adelaide.edu.au; R. Buyya, The University of Melbourne, Australia; email: rbuyya@unimelb.edu.au.

1556-4665/2022/10-ART10 $15.00
https://doi.org/10.1145/3529162

knowledge in situations where the scalability analysis may not be straightforward. The contributions provide architects and designers for QoS-aware dynamic service composition with the fundamentals for assessing the scalability of their own solutions, along with goal models and a list of application domain characteristics and metrics that might be relevant to other solutions. Our experience has shown that the technique was able to identify in both exemplars application domain characteristics and metrics that had been overlooked in previous scalability analyses of these DSC, some of which indeed limited their scalability. It has also shown that the experiences and knowledge can be transferable: The first exemplar was used as an example to inform and ease the work of applying the technique in the second one, reducing the time to create the model, even for a non-expert.

CCS Concepts: • **Software and its engineering → Extra-functional properties**; • **General and reference → Evaluation**; • **Computing methodologies → Modeling methodologies**

Additional Key Words and Phrases: Scalability modelling, dynamic service composition, autonomous and adaptive systems

ACM Reference format:
Leticia Duboc, Rami Bahsoon, Faisal Alrebeish, Carlos Mera-Gómez, Vivek Nallur, Rick Kazman, Philip Bianco, Ali Babar, and Rajkumar Buyya. 2022. Systematic Scalability Modeling of QoS-aware Dynamic Service Composition. *ACM Trans. Auton. Adapt. Syst.* 16, 3-4, Article 10 (October 2022), 39 pages.
https://doi.org/10.1145/3529162

1 INTRODUCTION

In QoS-aware Dynamic Service Composition, instead of linking programs and libraries at compile time, an application can be dynamically composed using web-services to achieve its functional and non-functional targets. For such, application architects specify the functionality that the component parts should have, their budgetary resources, and other **Quality of Service (QoS)** constraints. With this information, an application can search a Service Registry for the services that it wants and can bind to them dynamically. This dynamic binding allows an application to change the QoS that it exhibits at runtime. Several autonomous and adaptive systems, operating at scale, such as cloud-based software systems, smart/mobile services, services discovery and composition in adaptive environments such as smart cities, have been leveraging QoS-aware Dynamic Service Composition as an underlying technique to maintain their dependability requirements, optimize for value, and enforce their **Service Level Agreements (SLA)** at runtime.

Selecting services from the service registry faces a search problem: Matching the functionality is constrained by the costs the application stakeholder is willing to pay while ensuring that the QoS advertised are acceptable to application's needs. Dynamic service selection, while optimizing QoS, is known to be an NP-hard problem [Yu et al. 2007]. This immediately implies that any solution that tries to find the optimal combination would find itself confronted with an exponentially increasing search space. Hence, most solutions focus instead on obtaining a "good enough" combination of services that meet the QoS constraints that an application specifies.

Given such a scenario, it is not surprising that scalability is an important concern in the domain of QoS-aware dynamic service composition. A previous literature review by the authors [Duboc et al. 2018, 2013b] revealed that most authors of such solutions present some kind of scalability evaluation. Two types of characteristics are normally considered: application domain characteristics, which belong to the domain of QoS-aware dynamic service composition, and system design characteristics, which belong to the specific approach implemented by the solution. Evaluations also take into consideration a variety of metrics.

Unsurprisingly, the literature review showed that system design characteristics were unique to the scalability evaluation of each solution. However, it also revealed that only two application

domain characteristics (*workflow size* and the number of *concrete services*) and one metric (*execution time*) are considered across most of the works. The rest are more rarely found, appearing in one to three papers. But, most importantly, the review reported that authors normally do not justify their choice of characteristics and metrics by any systematic reasoning.

With such a variety in the literature, how should the designers of QoS-aware dynamic service composition proceed to evaluate the scalability of their solutions? While the system design characteristics are most likely unique to specific solutions, can application domain characteristics be reused? The scalability of a solution is highly dependent on its goals [Duboc et al. 2013a]. Therefore, a one-size-fits-all set of characteristics and metrics for scalability analysis is unlikely to be effective. But how to know the ones that are likely to affect the scalability of a given approach and that may have critical consequences?

Duboc et al. [2013a] offer a goal-modeling technique for identifying characteristics and metrics that can potentially affect the scalability of software systems. By relating characteristics and metrics to the goals of these systems, the technique supports informed decision-making on scalability analyses. However, the application of goal-modeling techniques for scalability analysis may not be straightforward. This analysis can be further hampered when there are neither a compilation of inputs that can inform the elaboration of models (e.g., common scalability dimensions and metrics) nor exemplars that can guide a systematic analysis.

The above concerns are among the challenges that are faced by practitioners when evaluating the scalability of QoS-aware **Dynamic Services Composition (DSC)**, that we address in this article. Our novel contributions are as follows:

(i) The article reports on new application of Duboc et al. in the domain of DSC—an application domain in which goal-modeling scalability technique has not been applied to, and choices of variables and metrics are neither straightforward nor often justified. The new application leverages goal modeling in requirements engineering, but tailored to the scalability analysis and unique requirements of DSC. The modeling is informed by DCS domain characteristics and metrics that were compiled by a systematic survey by the authors. The modeling can be also informed by additional characteristics and requirements that are related to the DSC underlying theory, its formulation, and assumptions. The modeling is extensible and open; transferable knowledge and the expertise gained in modeling DSC, if available (e.g., existing exemplar and patterns) can further refine the modeling.

(ii) The contribution aims at assisting DSC designers and architects in identifying relevant characteristics and metrics that can potentially affect the scalability goals of a system. To assist this objective, the article compiles a set of application domain characteristics and metrics that relate to the DSC domain. Additionally, the article contributes to **two detailed exemplars**, each of which deals with a distinctive approach for QoS-aware service composition: The **Cloud-based Multi-Agent System (Clobmas)** and the Portfolio-based Service Composition System. The exemplars serve two intertwined objectives: to (a) exemplify and evaluate the usefulness of a systematic scalability evaluation technique in the DSC domain; and (b) to provide practitioners with guidance and transferable knowledge for evaluating the scalability of systems that fundamentally build on dynamic service composition (e.g., autonomous services, multi-tenant cloud-based services composition, smart services, smart cities).

Our experience has shown that: (i) we were able to identify in both exemplars application domain characteristics and metrics that had been overlooked in previous scalability analyses of these solutions—some of which indeed limited their scalability; (ii) it has also shown that the experiences and knowledge can be transferable: the first exemplar was able to inform and ease the work of applying the technique in the second one, reducing the time to create the model, even for a non-expert. The second exemplar has also confirmed some of the common scalability issues and identified others; and (iii) the systematic goal-obstacles modeling for scalability of DSC offers the flexibility and

openness to incorporate new scalability concerns and queries that were not commonly discussed in the DSC literature. This can be attributed, for example, to the infancy of a given environment (e.g., when extending the coverage of DSC beyond services registries to consider cloud marketplaces) and/or emerging behaviors as a result of the new environment. It can also be attributed to the fundamentals of the DSC solution itself—its formulation, underlying theory, assumptions, and implementation—which may require customized analysis but can still benefit from the existing dimensions, metrics, and exemplars.

The article is organized as follows: Section 2 describes the basics of scalability and its evaluation in QoS-aware dynamic service composition and summarizes the goal-modeling technique that we apply. Sections 3 and 4 present the exemplars demonstrating how to apply the goal-modeling technique for QoS-aware DSC solutions. Section 5 evaluates the technique, discusses the contributions of these exemplars, and presents a list of application domain characteristics and metrics that may be useful to other solutions in this domain. Section 6 explores the related work. Finally, Section 7 summarizes our main findings and lists the future work.

2 BACKGROUND

2.1 Relevance to Autonomous and Adaptive Systems

Dynamic Service Composition is an enabler environment and part of the underlying infrastructure for supporting a variant of **Adaptive and Autonomous Systems (AAS)** [D'Angelo et al. 2020; Weyns 2019], where the constituent components of the architecture are services. Additionally, Dynamic Service Composition as a problem is fundamentally linked to AAS; it embeds flavors for enacting the self-adaptive processes, including monitoring, analysis, planning, execution through composition, feedback loops, and knowledge management to enable the dynamic composition process of services supporting an application at runtime [Caporuscio et al. 2015]. The dynamic and adaptive composition process is often steered by changes in the monitored variables of the managed system; these variables may relate to changes in **Quality of Services (QoS)**, **Service Level Agreement (SLA)** violations and the need to enforce compliance, continuous search for added value in service provision with considerations for context, costs, risks of the service provision, among the many constraints.

Contributions on DSC over the years have developed propriety analysis and planning techniques and mechanisms to inform the dynamic and adaptive composition problem [Asghari et al. 2018; Vakili and Navimipour 2017]. In the context of self-adaptive systems, these techniques essentially sit in the heart of the managing system and act on the monitored variables to inform the next cycle(s) of adaptation. The dependability of DSC is often linked to the ability to provide "seamless" computation addressing the various QoS tradeoffs and constraints, making the consideration of scale essential. The extent to which the planning and analysis can scale requires systematic modeling and analysis of the relevant scaling goals and their objectives, considering the various unbounded variables that relate to scalability and risks hindering scale against metrics that are sensible for a given domain. Systematic analysis of scalability can particularly help designers of self-adaptive and managed service-oriented systems to design better self-adaptive policies and/or tune the managing mechanisms so they can better cater of uncertainties surrounding the satisfiability of the scaling goals. Our consideration of unbounded variables in the systematic modeling and analysis is recognition of the dynamism that AAS exhibit at runtime due to changes in QoS requirements, fluctuation of demands on the services, changes in the operating environments, and so forth. Our considerations of obstacles analysis in the scalability goal modeling is an acknowledgement that risks in AAS need to be modeled and analyzed. The analysis can help designers and engineers of these systems to inform refinements and design preventative and mitigation strategies.

The article considers two exemplars, which are essentially AAS. The first application is **Cloud-based Multi-Agent System (Clobmas)**. Clobmas leverages **Market Based Control (MBC)** [De Wolf and Holvoet 2006] to adaptively manage collaboration and market equilibrium in cloud marketplace, constituting of cloud services providers and cloud services consumers, represented as decentralized agents, and considers QoS requirements and price. For the the second exemplar, the application adaptively reduces probable risks of QoS performance fluctuation, a common characteristic in **Cloud Service Composition (CSC)** due to changes in supply and demand of shared computational infrastructure and resources [Dejun et al. 2009]. For a given adaptation cycle, the adaptation decisions are informed by the extent to which the diversification of services can reduce risks and improve dependability, subject to QoS and cost constraints.

2.2 Scalability

Scalability is the "ability of a system to maintain the satisfaction of its quality goals to levels that are acceptable to its stakeholders when characteristics of the application domain and the system design vary over expected operational ranges" [Duboc et al. 2013a].

Some observations can be made regarding this definition: (1) Scalability is related with the satisfaction of all QoS, even though performance and resource usage are more commonly considered. (2) The system's scalability might be affected by multiple characteristics of both the application domain and the system design. Some of these may exhibit a wide variation in values during the lifetime of the system. We refer to these as *scaling dimensions*. (3) Scalability is not an absolute concept; a system is scalable with respect to a set of quality goals and scaling dimensions. Therefore, claims of scalability should be based on thorough analysis that consider these elements.

2.3 Scalability in Dynamic Service Composition

In a long-lived service-oriented system, the desired QoS will change with time. For example, during peak hours of usage, an application might want to exhibit high throughput, whereas during non-peak hours, it may want to minimize cost while meeting minimum reliability standards. In this scenario, a dynamic composition of services allows an application to search at runtime for the appropriate set of constituent services that will enable it to meet its several QoS goals.

The overall QoS of a service-oriented application is dependent on its constituent services and the structure of its workflow. e.g., while the cost of the application may be calculated by summing the cost of each of the individual services, not all QoS attributes can be calculated by a simple Stochastic Workflow Reduction [Cardoso et al. 2004].

Therefore, calculating the end-to-end QoS for each attribute is a computationally expensive process [Nallur and Bahsoon 2013]. Given this scenario, it is not surprising that our previous literature reviews [Duboc et al. 2018, 2013b] showed that scalability is indeed a concern for QoS-based service dynamic composition. Most authors attempt to justify their (explicit or not) scalability claims with evaluations of software qualities given some variation in the application domain and system design characteristics (i.e., scaling dimensions). Nevertheless, these evaluations vary greatly.

With respect to the scaling dimensions belonging to the application domain, there is a general agreement that the *workflow size* and the *number of concrete services* should be considered. Some works also agree on the *number of QoS* as an important scaling dimension. However, for all other application domain characteristics, each paper has its particular concern. As for the scaling dimensions belonging to the system design, there is no consensus on variables. Examples of such characteristics are: the number of *ants* in an **ant colony optimization (ACO)** algorithm [Li and Chen 2010] and the *max non-improving generations*, a termination condition for an optimization algorithm [X. and H 2011].

Regarding the metrics used to measure the scalability of solutions, it is most agreed that *execution time* is an important concern, followed by *success/failure rates*, *utility,* and *costs*. The study also revealed many others, less popular, metrics. Furthermore, each work assessed their own combination of metrics.

All these differences in the scalability analyses are by no means surprising, given the variety of techniques used for dynamic service composition.

2.4 Scalability Goal Modeling

Duboc et al. [2013a] extended the KAOS framework [van Lamsweerde 2008] to devise a generic technique for elaborating scalability goals. The main concepts of these approaches are briefly explained below but will be illustrated in detail in the exemplars.

2.4.1 A Brief Introduction to the KAOS Framework. KAOS is a goal-oriented framework for eliciting, evaluating, and analyzing software requirements [van Lamsweerde 2008]. Its *goal model* depicts the system's functional and non-functional goals and how they contribute to each other. In KAOS, a *goal* is a prescriptive statement of intent that a system should satisfy through the cooperation of agents. *Agents* can be humans, hardware devices, or software components that satisfy goals. *Domain properties* are descriptive statements that are always true in the application domain, such as physical laws. *Domain hypotheses* are also descriptive statements about the application domain, but they are considered to be subject to change. The term *domain assumption* is used to refer to both domain properties and domain hypotheses.

A goal model organizes goals in a hierarchy. An *AND-refinement* link indicates that to satisfy a goal, all of its offsprings must be satisfied. An *OR-refinement* represents that there are alternative AND-refinements that can satisfy the parent goal. High-level goals require the cooperation of multiple agents to be satisfied, whereas leaf goals are the responsibility of a single agent. During the refinement process, goals are decomposed into subgoals until they can be assigned to a single agent [Letier and van Lamsweerde 2002]. A goal under the responsibility of a software-to-be agent is called a *software requirement*, while a goal assigned to an agent in the environment is referred to as a *domain expectation*.

The first goal model produced is often too ideal. It fails to consider exceptional conditions in the application domain that may violate goals, requirements, and assumptions. For this reason, the KAOS framework includes a *goal-obstacle analysis*, which systematically checks the model looking for exceptional conditions that prevents the goal from being satisfied, i.e., *obstacles*. The likelihood and criticality of obstacles are assessed, and these are resolved by adding or modifying the goals of the model and by applying resolution tactics in the KAOS catalog [van Lamsweerde 2008].

2.4.2 Elaborating Scalability Goals with KAOS. Duboc et al.'s technique [Duboc et al. 2013a] extends the KAOS framework with the concepts of *scaling assumption*, *scalability goal,* and *scalability obstacle*. It also refines the goal-obstacle analysis and the catalog of resolution tactics for identifying and resolving scalability obstacles.

A *scaling assumption* is a domain assumption that specifies how certain characteristics in the application domain are expected to vary over time and across deployment environments. Scaling assumptions may be described with varying levels of precision, depending on the system. For example, a scaling assumption could distinguish value ranges for different time periods (e.g, year-by-year) and categories of applications (e.g., e-commerce, scientific, CRM). It could also be represented by orders of magnitude or by a full probability distribution of the domain quantity under consideration. There are a number of approaches that can be used for such, e.g., requirements elicitation techniques, extreme scaling scenarios, technology roadmaps, publicly available

performance-related benchmarks (e.g., spec.org), and estimation-by-analogy [Bahsoon and Emmerich 2008; Shepperd et al. 1996; van Lamsweerde 2008].

Two points are worth emphasizing regarding scaling assumptions: (1) they denote measurable quantities that can be used in the scalability analysis, and (2) the *absence* of a scaling assumption for a domain quantity means that there is *no assumed constraint* on its possible values; that is, its value at any point in time potentially could be infinite. The latter forces system designers to explicitly define these assumptions so they can make better informed design choices.

A *scalability goal* is a goal whose definition and required levels of goal satisfaction make explicit reference to one or more scaling assumptions [Duboc et al. 2013a]. That is, it states that a goal must be achieved for a certain range of values defined in the scaling assumption. The level of satisfaction may be fixed or vary in response to the variations specified in the scaling assumptions. Scalability goals can be specified at different levels. Following the KAOS definitions, a *scalability requirement* is a scalability goal assigned to an agent in the software-to-be (i.e., it is a leaf scalability goal).

A *scalability obstacle* is a condition that prevents a goal from being satisfied when the load imposed by the goal on the agents involved in its satisfaction exceeds the capacity of the agents This obstacle uses the concept of *goal load* and *agent capacity* to denote measures that characterize the amount of work needed and the amount of resources available to the agent to satisfy the goal, respectively [Duboc et al. 2013a]. Therefore, a scalability obstacle takes the form Goal Load Exceeds Agent Capacity.

Scaling assumptions and scalability goals are elaborated through a *scalability goal-obstacle analysis*, as follows: (1) systematically *identifies scalability obstacles* that may obstruct the satisfaction of the goals, requirements, and expectations; (2) *assesses* the likelihood and criticality of those obstacles; and (3) *resolves* the obstacles by modifying existing goals, requirements, and expectations, or by generating new ones so as to prevent, reduce, or mitigate the obstacles. The latter is supported by a catalog of *scalability resolutions* [Duboc et al. 2013a].

This article is mostly concerned with the identification and assessment of scalability obstacles in the domain of QoS-aware dynamic service composition and therefore focuses on the steps one and two of the scalability goal-obstacle analysis. Yet, some resolutions are briefly mentioned for the sake of completeness.

3 FIRST EXEMPLAR: THE CLOUD-BASED MULTI-AGENT SYSTEM (CLOBMAS)

The ***Market Based Control (MBC)*** [De Wolf and Holvoet 2006] is a pattern for communication/collaboration in decentralized systems, in which agents act as buyers or sellers of goods collaborating to achieve market equilibrium. In a software context, this protocol is typically used for efficient resource allocation in shared infrastructures, such as CPU cycles, bandwidth, and disk-usage [Bichler et al. 1999; Borissov et al. 2010; Chen and Yeh 2010].

The **Cloud-based Multi-Agent System (Clobmas)** uses MBC to define a marketplace that allows individual applications to select web-services in a decentralized manner. An application is viewed as a composition of multiple web-services, each with an specific functionality and QoS levels. Thus, each web-service contributes to the total QoS of the application.

Application stakeholders desiring to compose applications should specify them as a *workflow* of abstract services. An *abstract service* describes the functional specification of a certain task. Service providers offer *concrete services* that implement abstract services, each with an associated QoS advertised through **service level agreements (SLAs)**. Concrete services, in a particular market, that meet the QoS requirements for a application are viewed as *candidate services*. When receiving an application request, Clobmas decomposes the global QoS and budget constraints creating alternative bids for abstract services to find the best combination of concrete services out of candidate services. Clobmas also ensures that concrete services respect their SLAs, replacing them if

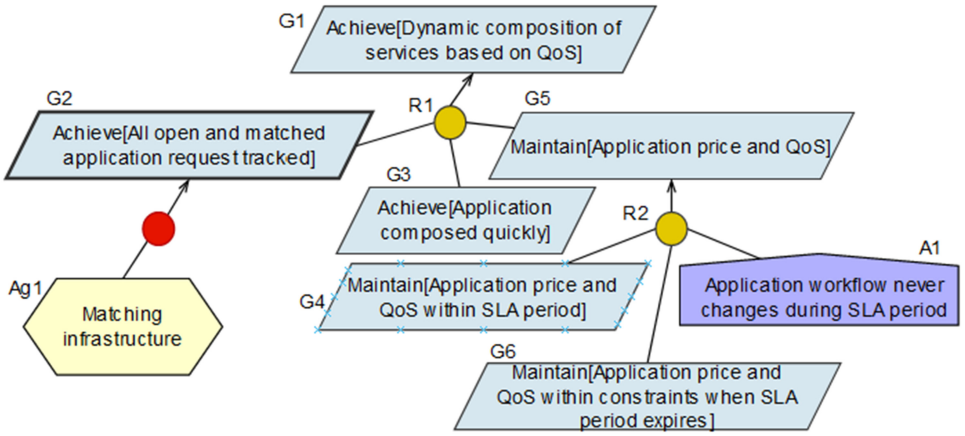

Fig. 1. Refinement of the goal Achieve [Dynamic composition of services based on QoS] of Clobmas.

necessary. Upon the expire of an SLA, Clobmas negotiates its renewal or finds a new set of concrete services that meets the application's global QoS and budget.

3.1 Initial Goal Model of Clobmas

In this section, we explain the original goal model of Clobmas. Since our aim is to provide an exemplar for the QoS-aware dynamic service composition community, the model is explained in detail. It is worth noting that the initial goal model of Clobmas, shown in Figures 1 to 7, use the standard goal-modeling notation and its reserved language constructs. In KAOS, details of the goals such as the QoS metrics are collected, how often these are verified, as well as application domain characteristics that define the "scalability level" of goals, do not appear in the diagram. Instead they are detailed in the textual description of the goals and assumptions, as we will demonstrate in detail in Section 3.2.

The main high-level goal of Clobmas is Achieve[Dynamic composition of services based on QoS]. Its refinement is represented by the goal G1 in Figure 1. The diagram represents goals as parallelograms, AND-refinements as arrows connected by a circle, and agents as hexagons.

To achieve this goal, the Clobmas framework must (a) keep information about open application requests for concrete services, (b) compose applications by selecting candidate services for the abstract services in the application workflow, and (c) maintain the agreed price and QoS for the concrete services composing the application. This is represented in the model by AND-refining (R1) the top goal into three goals Achieve[All open and matched application requests tracked] (G2), Achieve[Application composed quickly] (G3) and Maintain[Application price and QoS] (G4), respectively. The first goal is a leaf goal, assigned to the agent Matching Infrastructure (Ag1). The second goal will be refined in Figure 2. Finally, the third goal is further decomposed into Maintain[Application price and QoS within SLA period] (G5), stating that the QoS and price of concrete services composing the application should be maintained within the SLA period, and Maintain[Application price and QoS within constraints when SLA period expires] (G6), representing that when the SLA expires, it should be automatically renewed if both application stakeholder and service provider agree. When an agreement is not reached, the application stakeholder may compose the application out of a different combination of candidate services. In addition, the workflow of the application composed is assumed to never change within the SLA period; any change in the desired abstract services is treated as a new request. This is represented in the model by the domain assumption Application workflow never changes during SLA period (A1).

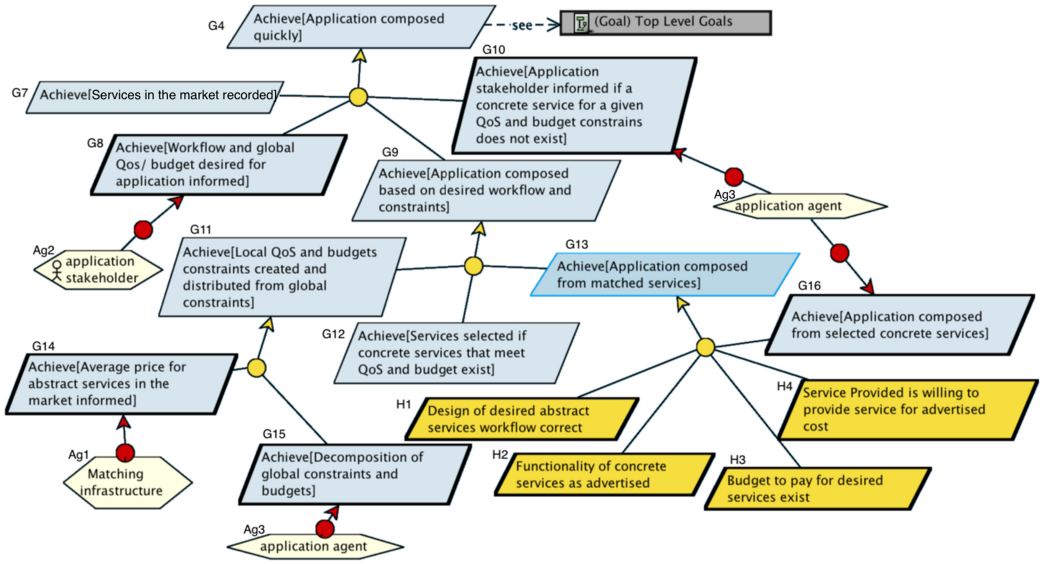

Fig. 2. Refinement of the goal Achieve [Application composed quickly].

The goal Achieve[Application composed quickly] (G4, Figure 1) is refined in Figure 2. To compose an application, the application stakeholder (Ag2) must inform the desired price and QoS for the entire application—the global constraints (G8). This information is used to search the separate concrete services that will compose the application. Clobmas must also keep track of all services in the market (G6). This goal requires the cooperation of agents and will be further refined in Figure 4. Composing the application (G9) requires breaking the global constraints into alternative combinations of price and QoS for abstract services (G11). This is accomplished by consulting the average price of concrete services available in the market (G14) and by decomposing the global constraints and budgets for the different services (G15). The latter goal is satisfied by the application agent (Ag3). Concrete services that meet the QoS and budget are then selected from the market (G12) and the application composed (G13). The latter can only be achieved if the following domain hypothesis are indeed true: that the design of the desired abstract services workflow was correctly defined (H1), that the functionality of concrete services are as advertised (H2), that the budget to pay for the desired services exists (H3), and that service provider is willing to provide the service for the advertised cost (H4). The selection of concrete services (G12) requires the cooperation of agents and will be further refined in Figure 3. Finally, the application agent (Ag3) is also responsible for informing the stakeholder if no concrete service could be found for the desired QoS and budget (G10) and for actually composing the concrete services (G16).

The goal Achieve[Service selected if concrete service that meet QoS and budget exist] (G11, in Figure 2) is refined in Figure 3. Satisfying this goal (G11) requires identifying a set of candidate services (G16), assuming that two hypotheses are true: that any combination of services that attends to the constraints is satisfactory (H5) and that concrete services do indeed implement the interface they advertise (H6). The service buying agent (Ag4) finds candidate services by requesting alternative combinations of cost and QoS (G17). Concrete services are then selected by choosing the best matches out of all possible matches (G19, G24, and G25). When all services are found, both their local and global costs and QoS are verified (G20 and G21), respectively, by the service buying agent (Ag4) and the application agent (Ag3). When a viable combination of services is not found (G18), the

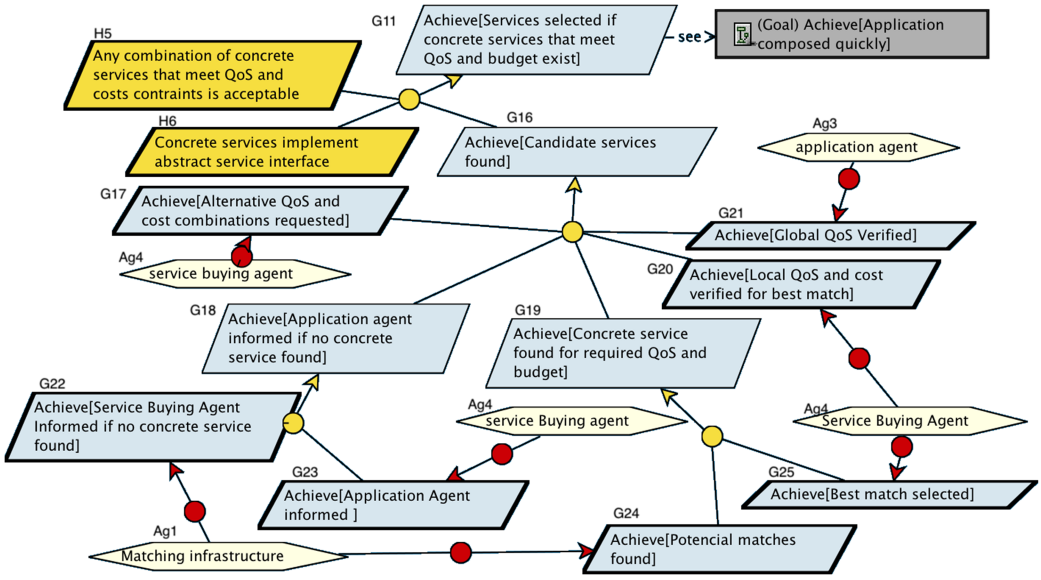

Fig. 3. Refinement of goal Achieve [Service selected if concrete service that meet QoS and budget exist].

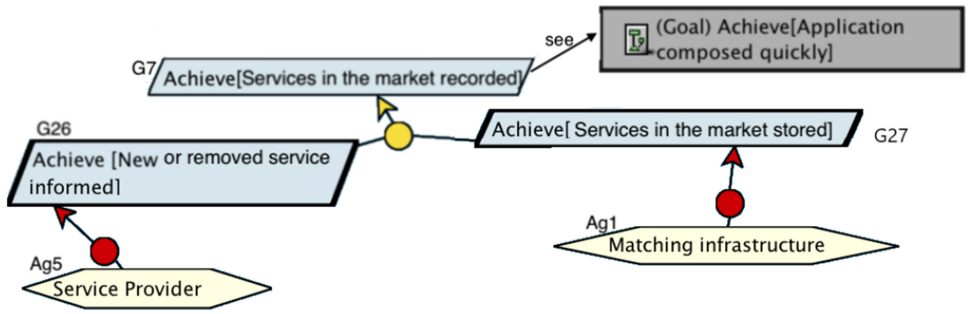

Fig. 4. Refinement of goal Achieve [Services in the market recorded].

matching infrastructure (Ag1) informs the service buying agent (Ag4) (G22), who in turn reports this fact to the application agent (Ag3) (G23).

The refinement of the goal Achieve[Services in the market recorded]) (G6, in Figure 2) is shown in Figure 4. It is important that Clobmas keeps accurate track of all services registered in the marked and their associate QoS and price. For such, the matching infrastructure (Ag1) stores the information of all new services announced (G27) by the services providers(Ag5).

Clobmas attempts to maintain the price and QoS between application stakeholders and services providers for as long as possible. That includes making sure that QoS and price are maintained within the SLA period (G5, Figure 1). The refinement of this goal is shown in Figure 5. Clobmas assumes that the cost of a concrete service does not change within the SLA period (H7), therefore, it only has to deal with two cases: when the concrete services violates the SLA for QoS (G28) and when the application stakeholder changes its price and QoS requirements (G29). The former goal needs to be further refined, while the latter triggers another composition of the application (G31) based on the new QoS and budget desired (G30) by the application stakeholder (Ag6).

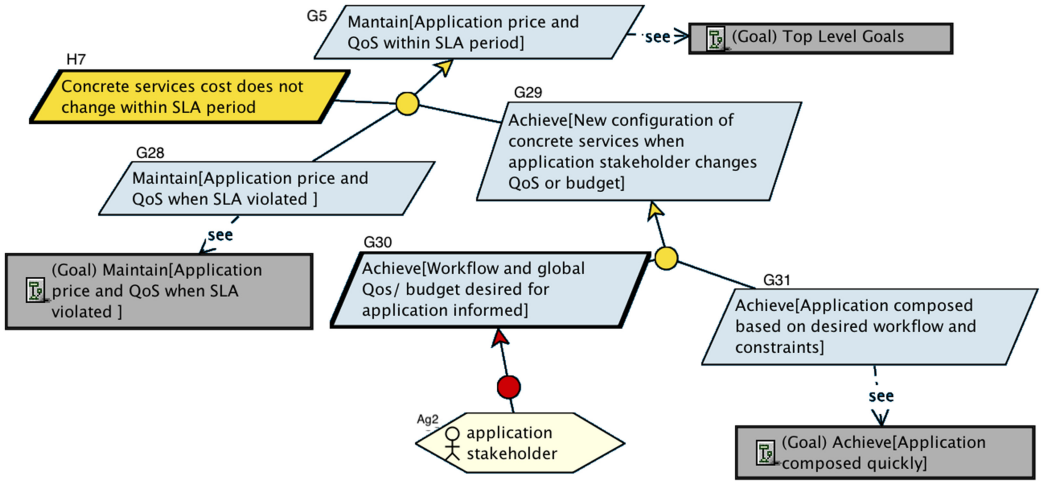

Fig. 5. Refinement of goal Maintain [Application price and QoS within SLA period].

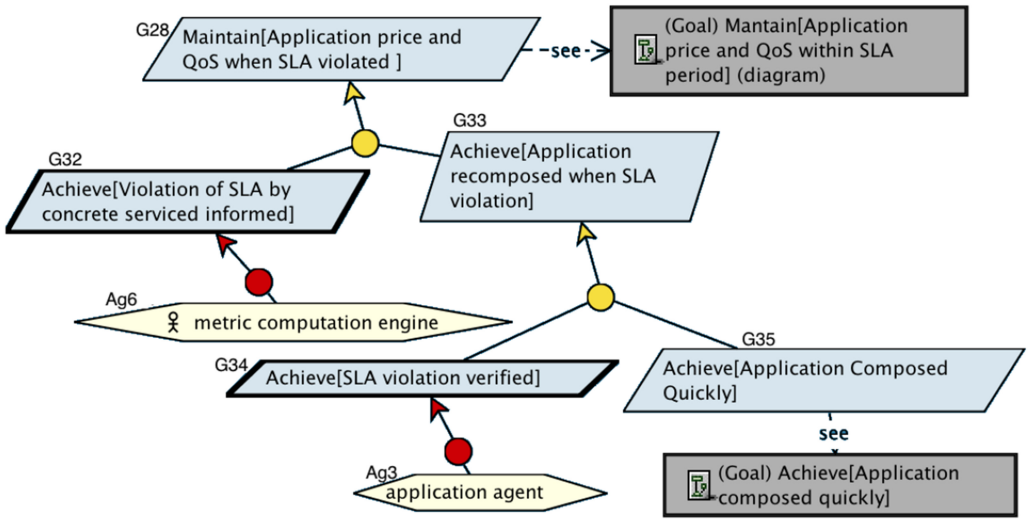

Fig. 6. Refinement of goal Maintain [Application price when SLA violated].

The refinement of goal Maintain[Application price when SLA violated](G28, Figure 5) is illustrated in Figure 6. Clobmas knows that an SLA is violated because the execution of concrete services are monitored by a third party metric computation engine (Ag7). When a violation is informed (G32), the application agent (Ag3) verifies the violation (G34) and if necessary recomposes the application out of a new set of concrete services (G35).

Figure 7 depicts the refinement of goal Maintain[Application price when SLA period expires] (G5, Figure 1). To maintain the price and QoS between application stakeholders and services providers for as long as possible, Clobmas attempts to maintain the agreements between application stakeholders and services provides even after the SLA expires (G5). For such, when the SLA expires, the service buying agent (Ag4) enquirers about any changes in the conditions of the previous SLA (G36). If the new conditions informed (G37) by the service provider (Ag5) are adequate, then the SLA

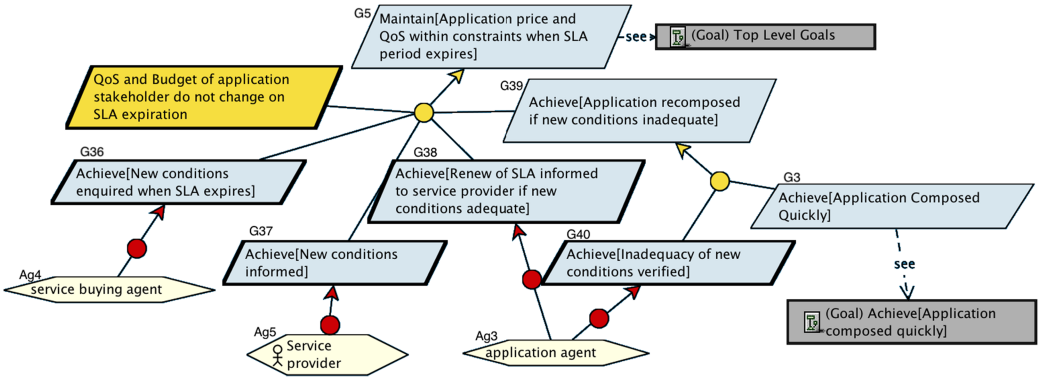

Fig. 7. Refinement of goal Maintain [Application price when SLA period expires].

is automatically renewed (G38) by the application agent (Ag3). Otherwise, Clobmas verifies that the new conditions are indeed inadequate (G40) and recomposes the application out of a new set of concrete services (G3).

Therefore, the initial goal model of Clobmas has six agents and 40 goals.

3.2 Scalability Goal-obstacle Analysis

Due to the size of the model, the goal-obstacle assessment of each goal cannot be fully explained in this article. Instead, we will exemplify the goal-obstacle analysis of **two selected goals**, followed by a **summary of the metrics and scaling dimensions** uncovered, and the assessment of **likelihood and criticality of selected scalability obstacles**. Some resolution to the scalability obstacles will be briefly mentioned but not discussed, since the **focus of this exemplar in on the identification and assessment of scalability obstacles**.

3.2.1 Sample Scalability Obstacles and Goals. This section exemplifies the scalability goal-obstacle analysis of the following goals: Maintain [Information about application requests], Achieve [Application composed from selected concrete services], and Maintain[Application price and QoS within SLA period]. For the sake of illustration, the first example will be described in greater detail.

Example 1: Achieve [All open and matched application request tracked]. This initial goal states that Clobmas must keep the relevant information of all application requests, the services available, their price, and QoS so it can compose them. As explained in Section 2.4.2, if domain quantities are not bound, then they are considered potentially infinite. Therefore, the agent Matching infrastructure cannot satisfy this goal for a potentially infinite number of application requests. Figure 8 shows how this situation is represented in the model as the **scalability obstacle** Load greater than Matching Infrastructure capacity, which is OR-refined into the sub-obstacles Unbounded number of matched requests, Unbounded number of open applications requests, and Unbounded number of concrete services.

We resolve these sub-obstacles by (1) introducing the **scaling assumptions** Expected open and matched application requests and Expected concrete services that bounds the number of concurrent application requests and concrete services expected by Clobmas, respectively; and (2) modifying the original goal to state that it only needs to be satisfied for the range defined in the scaling assumptions.[1] This modification, shown in Figure 9, corresponds to the application of the

[1]The range of scaling dimensions and acceptance criteria for goals can be elicited by a number of requirements engineering techniques (see Section 2.4.2).

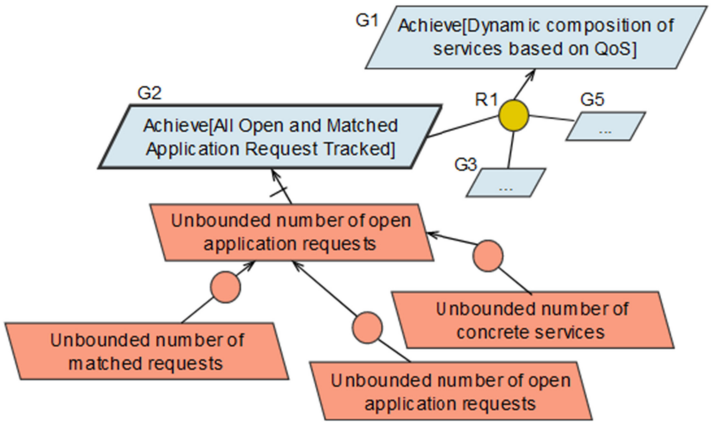

Fig. 8. Scalability obstacle to Maintain [Information about application requests].

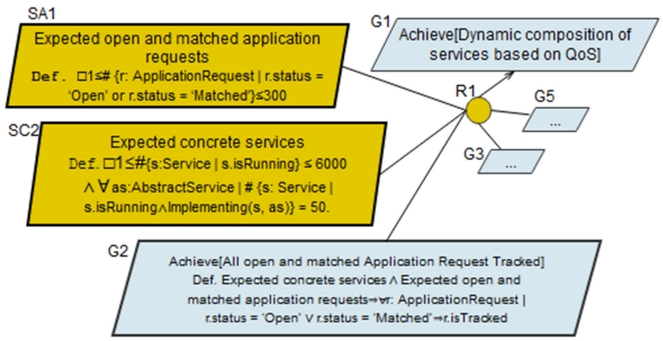

Fig. 9. Resolution of scalability obstacle to goal "Achieve [All open and matched application request Tracked]."

resolution tactics *Introduce scaling assumption* and *Weaken the goal with scaling assumption* [Duboc et al. 2013a]. The assumptions and the new goal are specified as follows:

Assumption Expected open and matched application requests

Category Scaling assumption

Definition The number of concurrent open and matched application requests is expected to vary between 1 and 300.

Formal Definition □ 1 ≤ # {r: ApplicationRequest | r.status = "Open" ∨ r.status = "Matched"} ≤ 300

Assumption Expected concrete services

Category Scaling assumption

Definition The number of concrete services is expected to vary between 1 and 6.000. The number of concrete services per abstract service is 50.

Formal Definition □ 1 ≤ # {s:Service | s.isRunning} ≤ 6000 ∧ ∀ as:AbstractService | # {s: Service | s.isRunning ∧Implementing(s, as)} = 50.

Goal Achieve [All open and matched application request tracked under "Expected open and matched application requests" and "Expected concrete services"]
Category Functional, scalability goal
Quality Variable storage space: application request → disk space
Definition Clobmas shall keep track of all open and matched application requests and their related information as long as the number of application requests and concrete services do not exceed the bounds stated in the scaling assumptions "Expected application requests" and "Expected concrete services."
Formal Definition ExpectedConcreteServices ∧ ExpectedApplicationRequests ⇒ ∀ r: ApplicationRequest | r.status = "Open" ∨ r.status = "Matched" ⇒ r.isTracked.[2]

A few points are worth emphasizing regarding the goal-obstacle analysis:

- In what concerns the scalability of the software system, the goal-obstacle analysis is only performed on the goals under the responsibility of agents belonging to the system; in this case: application agent, matching infrastructure and service buying agent. Goals under the responsibility of agents in the application domain are not analyzed as they cannot be controlled by the software system designer.
- The goal resulting from a goal-obstacle analysis is both a **functional** and a **scalability goal**. It is a functional goal because it determines a functionality of the system and it is also a scalability goal because its satisfaction criteria considers a range of values for the number of application requests.
- The quality variables (e.g., storage space) indicate the metric to evaluate the goal satisfaction, while the application domain characteristics that define the expected "scalability levels" of the goals (e.g., number of application requests, number of concrete services, number of matched requests) are defined in the scaling dimensions. These details are found in the textual descriptions of the scalability goals and assumptions.
- Ideally, the range of values for application domain characteristics for given scaling dimensions should reflect the values that the system designer realistically expects. However, if at the time of modeling the designer is uncertain about these values, then s/he can define broader ranges to explore the scalability limits of the system. This can be a good strategy for better informing the designers on realistic expectations that relate to these values. For example, in Clobmas, the initial range of values for the variables number of QoS and number of markets were set to 1–100, considering a hypothetical scenario. When simulations showed that the system could not scale well for this range, experts were consulted for revising these values, prioritizing markets and/or considering the tradeoffs involved when reducing the number of the markets to be searched.
- Resolving the scalability sub-obstacles as described above may not be sufficient to ensure that the agent—the Matching Infrastructure—is capable of handling the load imposed—the number of application requests and concrete services. Theoretically, that means that the scalability obstacle Load greater than Matching Infrastructure capacity may still exist, but would be now obstructing the scalability goal Achieve [All open and matched application request tracked under "Expected open and matched application requests" and "Expected concrete services"]. Resolving this obstacle will require the assessment of its criticality and likelihood, as well as possibly further resolutions. This assessment will be described in Section 3.2.2.

The next two examples are shown without the graphical representation of the goals, obstacles, and assumptions, but follow an equivalent rationale.

[2]In KAOS, the ⇒ symbol means entailment: "A ⇒ B" is a shorthand for "Always(A implies B)."

Example 2: Achieve[Application composed from selected concrete services]. This goal states that once the best matches for the concrete services are returned by the service buying agent, then the application agent is responsible for composing the application by combining these services as determined by the application workflow. This goal is satisfied when the application agent correctly combines the concrete service within a fraction of the total time Clobmas is expected to respond to an application request (i.e., timeToAchieveQoS). The load this goal imposes on the application agent is determined by the number of concurrent application requests and the size of the workflows (i.e., the number of concrete services the agent must combine). The application agent has finite capacity and cannot handle an unlimited number of concurrent application requests, neither compose an unlimited number of concrete services, causing the scalability obstacle Goal load exceeds application agent capacity. This obstacle can be further refined into Unlimited number of concurrent application requests and Unlimited workflow size. Their resolution is achieved by introducing the scaling assumptions Expected number of concurrent application requests and Expected workflow sizes, which bounds the range of values for both characteristics, and by replacing the original goal by the scalability goal Achieve[Application composed from selected concrete services under "Expected number of concurrent application requests and workflow sizes"]. Both the scaling assumptions and the scalability goals are presented below:

Assumption Expected number of concurrent application requests

Category Scaling assumption

Definition The number of concurrent application requests opened is expected to vary between `1 and 300`.

Formal Definition $\Box$ 1 ≤ # {r: ApplicationRequest | r.status = "Open"} ≤ 300

Assumption Expected workflow sizes

Category Scaling assumption

Definition The size of the workflow in any application is expected to `vary between 1 and 20`.

Formal Definition ∀ a: Application | 1 ≤ WorkflowSize(a) ≤ 20

Goal Achieve [Application composed from selected concrete services under 'Expected number of concurrent application requests and workflow sizes']

Category Functional, scalability goal

Quality Variable execution time: application request, workflow size → time

Definition Clobmas shall compose the application using the concrete services selected in the market within $timeToAchieveQoS/100$ as long as value range of the number of concurrent application requests and workflow size do not exceed the bounds stated in the scaling assumptions "Expected number of concurrent application requests" and "Expected workflow sizes."

Formal Definition Expected number of concurrent application requests ∧ Expected workflow sizes ⇒ ∀ m:Market, a: Application, c: ConcreteService | c ∈ m.SelectedServices(a) ⇒ $\Diamond_{\leq timeToAchieveQoS/100}$ Compose(a, m.SelectedServices(a))

Example 3: Maintain[Application price and QoS within SLA period]. This goal states that Clobmas should ensure that the composed application maintains the overall agreed price and QoS during the time established by the SLA. Unlike the two previous examples, this is not a leaf-goal, meaning that its satisfaction is determined by the satisfaction of its descendants. To make sure that we are not overlooking any scalability obstacle or dimension, scalability goal-obstacle analyses are normally carried out in leaf-goals. However, we choose to show the analysis of this goal to illustrate how it would look like for a non-leaf goal and because we believe that it better illustrates

aspects of AAS. More attentive readers will notice that the result that this analysis is equivalent to applying the goal-obstacle analyses to all its leaf-goals descendants.

The goal Maintain[Application price and QoS within SLA period] is satisfied when several agents work together to monitor the QoS and price of the application, recomposing its workflow when the SLA is violated or when the stakeholder changes its desired QoS or budget. The refinement of this goal is shown in Figure 5. As a combination of several leaf-goals, its satisfaction is measured by a number of metrics: execution time, storage space, communication bandwidth and utility, with the latter determining the selection of the best combination of concrete services based on their QoS. The load this goal imposes on the agents that have to collaborate to sense its satisfaction is determined by the scaling of many application domain characteristics: number of concurrent application requests, size of the workflows, number of changes in QoS or price in existing concrete service, number of not matched abstract services, number of SLA violations at a given time, number of concrete services per abstract service in market, number of markets per abstract services, number of QoS per abstract service, number of constraints for qoS per abstract service, and types of QoS constraints in application workflow. These agents have finite capacity and cannot handle an unlimited number of these characteristics, causing the scalability obstacle Goal load exceeds application agent capacity. Its resolution is achieved by introducing the scaling assumptions that bound the range of values for each of these characteristics and by replacing the original goal by the scalability goal Achieve[Application price and QoS within SLA period under "Expected number of concurrent application requests, workflow sizes, etc."]. The textual descriptions of this scalability goal and scaling dimensions are similar to the ones in Examples 1 and 2.

We define the remaining scalability obstacles and scaling assumptions of Clobmas following a similar rationale. As a result, we uncovered four metrics and 17 application domain characteristics that may be relevant to the scalability analysis of Clobmas. These are:

Scaling dimensions: number of concurrent application requests in the market, number of running applications, workflow size, number of QoS per abstract service, number of constraints for QoS in application workflow, type of QoS constraints, number of markets per abstract service, number of concrete services per abstract service in market, number of alternative QoS and cost requests, number of unmatched abstract services, number of expired SLAs at a given time, number of renewed SLAs at a given time, number of SLA violations at a given time, number of changes in QoS or budget of existing concrete services, number of concrete services per abstract services in the market, number of new concrete services added to the market, number of matched abstract services.

Metrics: execution time, disk storage space, satisfaction rate, and application utility. The latter determines the selection of the best combination of concrete services based on their QoS.

Table 1 shows the metrics and the scaling dimensions for all goals under the responsibility of the system-to-be agents. These have been identified following an analogous rationale to the examples above.

3.2.2 Obstacle Assessment and Resolution. Once the previous step is completed—i.e., the scalability assumptions and scalability goals have been identified—the system designer should assess the likelihood and the criticality of the remaining scalability obstacles. This evaluation helps him or her to define which obstacles should be further examined and resolved.

As with scalability goals, scalability obstacles can also occur at any level of the goal model. Low-level goals can be very useful for identify application domain characteristics that vary during the system lifetime. However, in the assessment of scalability obstacles, one may choose to consider

Table 1. Metrics and Unbounded Variables in Clobmas Goals under the Responsibility of the System-to-be-agents

ID	Goal	Metric	Unbounded variables
G2	Mantain[Information about application requests and concrete services]	storage space	workflow size, num. of QoS per abstract service
G10	Achieve[Application stakeholder informed if a concrete services for a give QoS and budget constraint does not exist]	execution time	num.of unmatched abstract services
G14	Achieve[Average size per abstract service in the market informed]	execution time	workflow size, num.of concurrent application requests, num.of concrete service per abstract service in the market, number of markets per abstract services
G15	Achieve[Decomposition of global constraints and budget]	execution time	num. of QoS per abstract services, num. of constrains for QoS in the application workflow, type of QoS constraints, workflow size, num. of concurrent application requests
G16	Achieve[Application composed from selected concrete services]	execution time	num. of concurrent application requests, workflow size
G17	Achieve[Alternative QoS and cost combination requested]	utility, execution time	num. of market per abstract services, num. of alternative QoS and cost requests
G20	Achieve[Local QoS and cost verified for best match]	execution time, cost	execution time, num. of QoS per abstract service, num. of concurrent application requests
G21	Achieve[Global QoS verifies]	execution time	num. of concurrent application requests, num. of constrains for QoS in the application workflow, type of QoS constraints, workflow size
G22	Achieve[Service buying agent informed if no concrete service found]	execution time	num. of unmatched abstract services
G23	Achieve[Application agent informed]	execution time	num. of unmatched abstract services
G24	Achieve[]Potential matches found]	satisfaction rate, execution time	num. of concurrent concrete services per abstract service in a application workflow
G25	Achieve[Best match selected]	execution time	num.of matched abstract services, workflow size, num. of constrains for QoS in the application workflow
G27	Achieve[Services in the market stored]	storage space, execution time	num. of concrete services per abstract service in the market, num.of QoS per abstract service, num. of changes in QoS or price of existing concrete services
G34	Achieve[SLA violation verified]	execution time	num. of SLA violations in a give time
G36	Achieve[New conditions enquired when SLA expires]	execution time	num. of SLA violations in a give time
G38	Achieve[Renew of SLA informed to service provider if new conditions adequate]	execution time	num. of expired SLAs at a given time, num. of QoS per abstract service, type of QoS constraint, workflow size
G40	Achieve[Inadequacy of new conditions verified]	execution time	num. of expired SLAs at a given time, num. of QoS per abstract service, num. of constraints for QoS in the application workflow, workflow size

higher-level obstacles, which can help to quickly identify which scaling variables affect the scalability of the system. Then, the system designer can explore the sub-goals to find the root of the problem.

When analyzing the likelihood and criticality of obstacles, it is advisable to keep the following in mind: While the **likelihood** of scalability obstacles is highly influenced by the scaling dimensions affecting the obstructed goal, its **criticality** is directly related to the goals that these dimensions affect. Therefore, although the analyst may choose to assess higher-level scalability obstacles, he or she should note which are the goals in its hierarchy that are being affected by the scaling dimensions. Furthermore, it is important to ensure that the scalability obstacles chosen cover all leaf-goals and application domain characteristics in the model.

Table 2 exemplifies the assessment of **selected scalability obstacles** of Clobmas. For the sake of illustration, we show obstacles at different levels. We use these examples to highlight some issues to consider when performing the analysis of the likelihood and criticality:

- Obstacle to leaf goals, such as Maintain[Information about application requests and concrete services] (G2), are written at the level of the system-to-be agent; that is, using the format Load greater than agent capacity. Additionally, a leaf-goal may refer to several scaling dimensions. The analyst may choose to consider as a single obstacle affected by several application domain characteristics (as row 1 from Table 2) or as separate sub-goals (as discussed in the next point).
- Obstacles to higher-level goals, such as Achieve [Services selected if concrete services that meet QoS and budget exists] (G9), are written at the level of the system; that is, using the format Load greater than system capacity. Additionally, a higher-level goal theoretically can be affected by the scaling dimensions of all its leaf-goals. In this example, G9 has in its hierarchy 10 leaf goals (G14, G15, G16, G17, G20, G21, G22, G23, G24, G25, Figure 2) and, consequently, 10

Table 2. Assessment of Likelihood (Li) and Criticality (Cr) of Scalability Obstacles, Classified as High (H), Moderate (M), and low (L) for Clobmas

ID	Scalability Obstacle	Li	Cr	Rationale
1	The number of concrete services and application requests exceed the matching infrastructure ability to update information about concrete services price and QoS in the market	H	H	The goal `Maintain [Updated information about concrete services price and QoS in the market]` (G2) requires that Clobmas maintain fast and ever-increasing storage, since the number of transactions can only increase monotonically. The multiple double-auction mechanism is designed to achieve fast matching and not storage.
2	The number of concrete services available exceeds system ability to compose application based on the desired workflow and constraints	H	H	Since the cloud is an open environment, popular abstract services will attract many service providers, which will increase the number of potential service matches. The goal `Achieve [Application composed based on desired workflow and constraints]` (G9) will be affected if the number of potential matches rises very fast.
3	The number of QoS attributes exceeds system ability to compose application based on the desired workflow and constraints	L	M	The goal `Achieve [Application composed based on desired workflow and constraints]` (G9) is affected if the number of QoS attributes changes arbitrarily. However, the number of QoS that an application is interested in, does not change very frequently.
4	Workflow size exceeds system ability to compose application based on the desired workflow and constraints	L	H	More complex workflows will result in a greater probability of concrete services not being found, affecting the goal `Achieve [Application composed based on desired workflow and constraints]` (G9). However, it is assumed that application designers generally do not change their workflows very frequently and will typically enter markets where they are confident of finding concrete services.
5	The number of markets exceeds the system's ability to compose application based on desired workflow and constraints	L	M	The goal `Achieve [Application composed based on desired workflow and constraints]` (G9) requires that services are found in a market, whenever an application makes a request, else the application will have to move to another market. However, markets that do not host popular services will eventually shut down due to low volumes of trading. Market forces will ensure that services will move to markets that have some popularity.

scaling dimensions that may affect its satisfaction (see goals above in Table 1). To ease the assessment of the likelihood and criticality, the analyst may choose to consider separate sub-obstacles to individual or subsets of scaling dimensions (as in rows 2–5 of Table 2).

In Clobmas, once the likelihood and criticality of scalability obstacles were estimated, the ones with high to medium likelihood or criticality were further evaluated through simulation. In sequence, we briefly discuss the simulation for two obstacles (rows three and five of Table 2) that refer to the goal Achieve[Application composed based on desired workflow and constraints]. This goal defines that the growth in execution time should be at most low-order polynomial.

Obstacle 1: The number of QoS attributes exceeds the system ability to compose the application based on desired workflow and constraints: Simulations of the system using a hypothetical scenario have considered up to 100 QoS attributes. This is clearly an idealistic goal that can rarely materialize in practice, unless the QoS conceptualizes and expresses various concerns linked to the SLA. The application of such a scenarios has showed that the increase of QoS attributes caused two phases of growth in the time to select the services. Up to an extent, the growth was bi-quadratic and then it became exponential. The system designer can be faced with a tradeoff: exhaustive consideration of QoS vs. selective strategy and performance. To resolve this tradeoff, the designer may, for example, avoid exponential growth by applying the resolution "Weaken goal definition by strengthening scaling assumption" [Duboc et al. 2013a]. In the Clobmas case, the designer consulted capacity planning experts and weakened the goal to verify that the range suggested by the experts

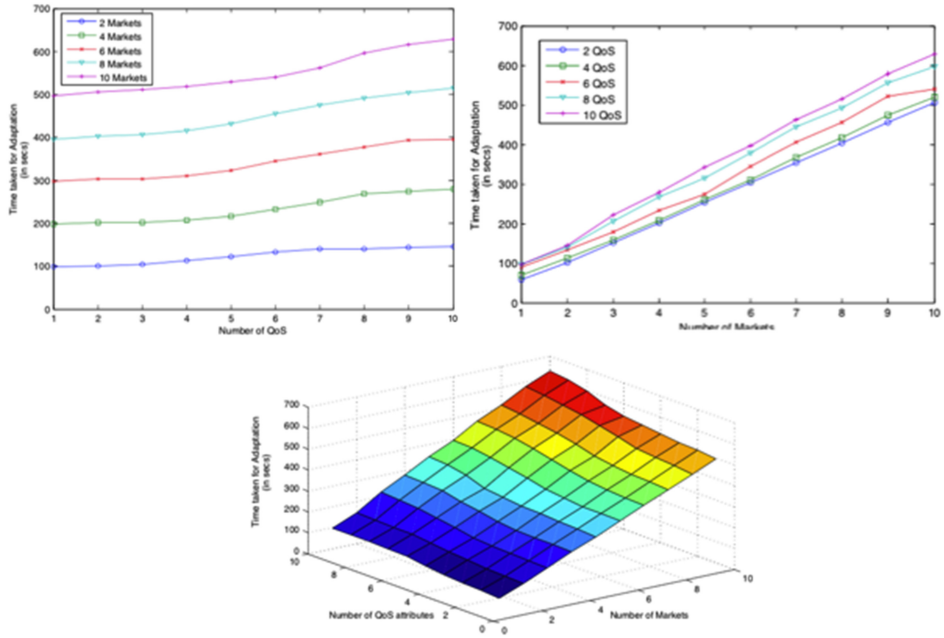

Fig. 10. Time-to-adapt (or time-to-achieve-QoS) against the number of QoS and markets.

could be handled by the system. The analyst observed that considering 10 critical QoS to be a sensible input for resolving this tradeoff (i.e., providing meaningful QoS consideration while avoiding exponential growth in the search).

Obstacle 2: The number of markets exceeds the system's ability to compose the application based on desired workflow and constraints: Simulations of the system using a hypothetical scenario have considered up to 100 markets of comparable size of constituent service offerings. The results again showed a two-phases growth: bi-quadratic followed by exponential. In this case, the system architect had initially hoped to support up to 50 markets but tested 100 to explore the bounds of the system. However simulations showed that the search using 50 markets experienced exponential growth challenging the scalability of the system. This has consequently made the search time-demanding and unrealistic in online scenarios. As with the QoS, arriving on a sensible number of markets to support is another tradeoff decision. In this case, the same resolution "Weaken goal definition by strengthening scaling assumption" [Duboc et al. 2013a] was applied, and 10 markets revealed to be a sensible input for this tradeoff decision, reconciling expert's input and observations from the goal analysis/simulation.

Figure 10 shows how within the redefined bounds, the scaling of the system is acceptable; that is, it satisfies the high-level goal Achieve[Dynamic composition of services based on QoS] of being at most a low-order polynomial.

4 SECOND EXEMPLAR: PORTFOLIO-BASED SERVICES COMPOSITION SYSTEM

The portfolio-based service composition [Alrebeish and Bahsoon 2015] aims to reduce probable risks of QoS performance fluctuation, a common characteristic in **Cloud Service Composition (CSC)** due to changes in supply and demand of shared computational infrastructure and resources [Dejun et al. 2009]. The mechanism adaptively modifies the selection of concrete services in response to changes in the market. For a given adaptation cycle, the adaptation decisions are

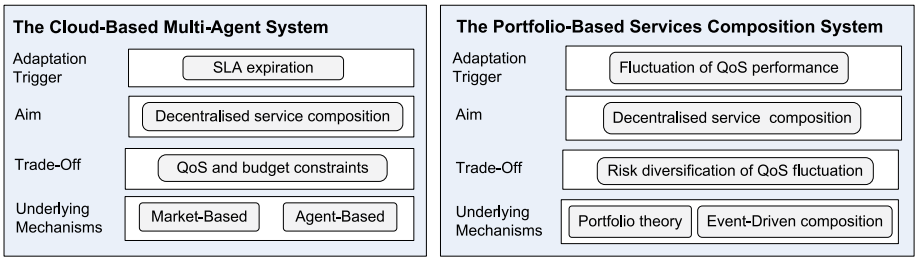

Fig. 11. Layered architecture diagram to compare exemplars.

informed by the extent to which the diversification of services can reduce risks and improve dependability, subject to QoS and cost constraints.

This system differs from Clobmas in the sense that it uses Portfolio theory [Markowitz 1952] to inform the composition. Moreover, the adaptation in the Clobmas is event-driven, where recomposition reacts to changes in the cloud service provision. That is to say, the adaptation is based on trust assumptions and solely driven by the published provision and promises of the SLA. This second system challenges these assumptions, taking a proactive approach to adaptation through continuous monitoring for fluctuation of QoS: If the fluctuation rate drops below the promised threshold, then adaptation is triggered. Figure 11 depicts a layered architecture diagram to illustrate the connection between the two exemplars at a high level of abstraction.

4.1 Initial Goal Model of the Portfolio-based System

This exemplar applies the same goal-modeling technique exemplified above. Most importantly, it uses the Clobmas model as a starting point to model this system.

The refinement of the goal Achieve[Dynamic composition of services with minimum QoS performance fluctuation] is shown in Figure 12 (G1). This is the main goal of portfolio-based allocation. To achieve this goal, the framework must (a) keep information about open requests for cloud services compositions and buyer preferences (G2); (b) compose applications by selecting a set of diversified concrete services for the abstract services in the application workflow (G3); and (c) maintain updated information about concrete services in the market (G4). The first goal has been assigned to the buyer agent (Ag1), while the other two need to be further refined.

Figure 13 illustrates the refinement of goal Maintain[An updated information about concrete services price and QoS in the market] (G4, Figure 12). As the cloud is a dynamic environment, the framework must track changes in all services registered in the market. This includes recording the updated price and QoS of each service (G5 and G6), storing the historical record of QoS (G7), and maintaining an updated risk evaluation of the fluctuation and the correlation between QoS for the services (G8). A recomposition will be triggered whenever there is a change in one of the services composing the application or when an SLA contract expires (G9). This recomposition must be effective (G10), that is, it assumes that the benefit of the recomposition outweighs the cost of relocating the services (H1).

The goal Achieve[Effective diversification of services composition] (G3, Figure 12) is refined in Figure 14. To diversify the selection of services composition, the buyer agent (Ag1) needs to complement this process with information regarding the buyer preferences and QoS constraints (G12). Then, buyer preferences and constraints are used to search the market registry and to compose the system based on the desired workflow and constraints (G10). For that, the market regulator (Ag3) must keep updated information about the concrete services available (G13), so concrete services that meet the criteria can be selected (G16). When compatible candidate services cannot be found,

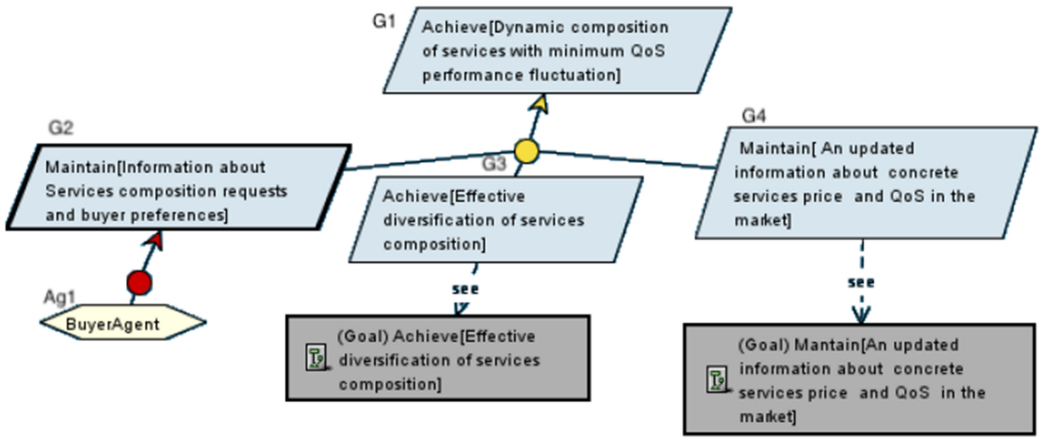

Fig. 12. Refinement of the goal Achieve [Dynamic composition of services with minimum QoS performance fluctuation].

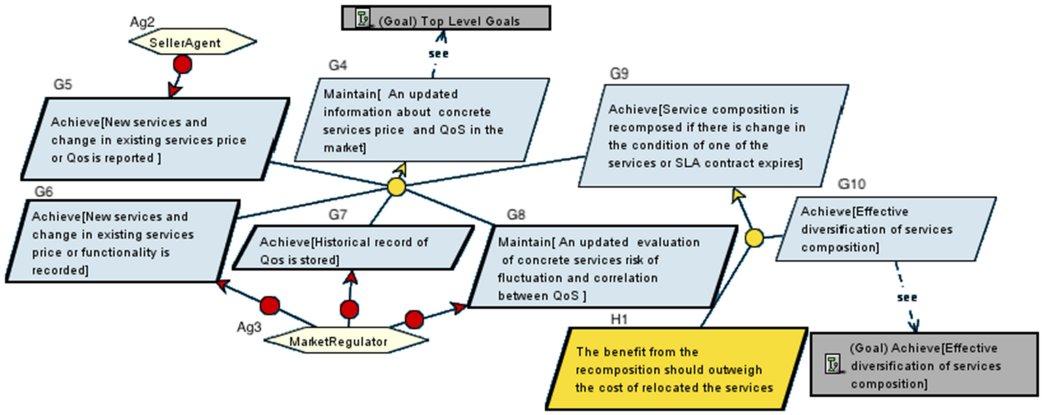

Fig. 13. Refinement of goal Maintain [An updated information about concrete services price and QoS in the market].

the buyer agent (Ag1) issues a warning to the buyer (G11). However, if a group of compatible candidate services exist in the market, then the buyer agent (Ag1) will use portfolio theory to effectively select a set of diversified set of services (G14). This means that the services will be composed from selected candidate services that share minimum correlation between their performance (G15). The composition can only happen if the following assumptions are indeed true: the design of the workload is correct (G12), the functionality corresponds to what has been advertised (H3), the budget exists (H4), the provider is willing to provide the service for the advertisement costs.

The refinement of goal Achieve[Services selected if concrete services that meet QoS and budget exist] (G16, Figure 14) is shown in Figure 15. Satisfying this goal requires identifying a group of compatible candidate services, assuming that any services that satisfy the constraints are selected as candidates (H6) and that these services indeed implement the interface they advertise (H6). The MarketRegulator (Ag3) reports all services found (G18) to the buyer agent before they are considered in the diversification process. The Buyer agent (Ag1) then explores all the possible services composition to ensure the optimality of the selected **Cloud Service Compositions (CSC)** (G17).

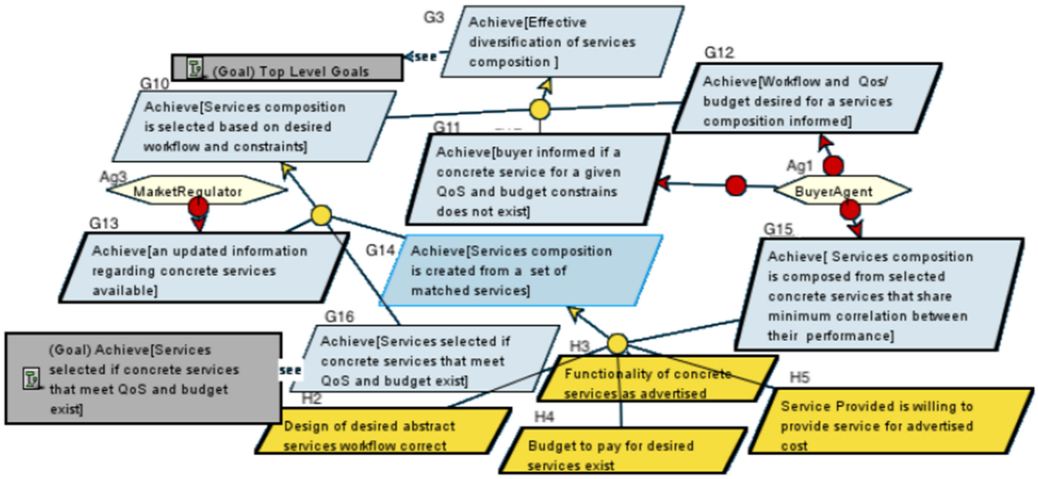

Fig. 14. Refinement of goal Achieve [Effective diversification of services composition].

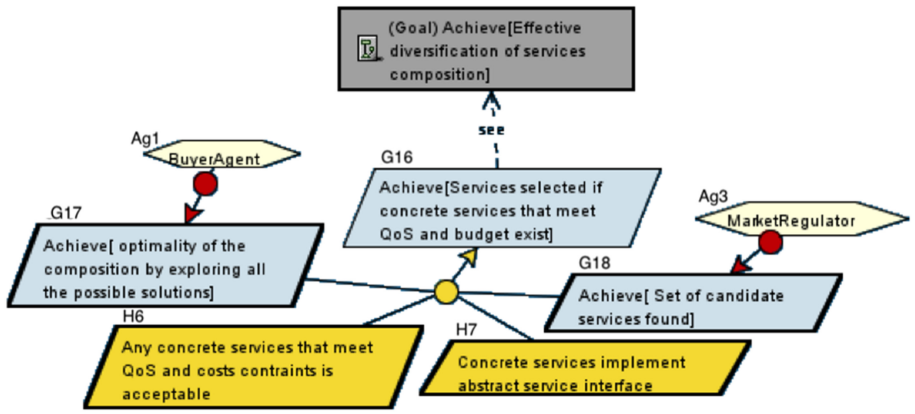

Fig. 15. Refinement of goal Achieve [Services selected if concrete services that meet QoS and budget exist].

Therefore, the initial goal model of the portfolio system has 18 goals and three agents.

4.2 Scalability Goal-obstacle Analysis

Like in the previous exemplar, this section exemplifies the scalability goal-obstacle analysis of two goals: Maintain[Updated evaluation of concrete services risk of fluctuation and QoS correlation between services] and Achieve[Effective diversification of services composition]. The analysis of the remaining goals follow an analog rationale.

Example 1: Maintain[Updated evaluation of concrete services risk of fluctuation and QoS correlation between services]. This goal states that the risk of QoS fluctuation for each service registered in the market and correlation between QoS are evaluated by the Market Regulator. This goal is satisfied if all the required information is evaluated and stored within a reasonable time and without exhausting the Market Regulator processing capacity or the storage space available. The load of this goal is determined by the number of services in the market, the number of QoS, the number of changes in QoS, and the length of the historical record of each QoS. The Market Regulator agent

has a finite capacity and cannot handle an unlimited scaling of these characteristics. Therefore, the **scalability obstacle** Goal load exceeds Market Regulator agent capacity can be further refined into the sub-obstacles Unlimited number of services, Unlimited number of QoS, Unlimited changes in QoS, and Unlimited length of the historical record. These sub-obstacles are resolved by introducing the **scaling assumptions** Expected number of services, Expected number of QoS, Expected number of QoS changes, and Limited length of the historical record, and by replacing the original goal by the **scalability goal** Maintain[Updated evaluation of concrete services risk of fluctuation and QoS correlation for expected number of services requests, QoS, Qos changes and limited length of the historical record]. These scaling assumptions and scalability goal below are shown below:

Assumption Expected number of services
Category Scaling assumption
Definition Over the next year, the number of services registered in the market is expected to be 1.000 and 10.000.
Formal Definition $\forall$ m:Market, s:Service $\Rightarrow \Diamond_{\leq 1\ \text{year}}$ { $1.000 \leq \#\{s \mid \text{Registered}(s,m)\} \leq 10.000$ }

Assumption Expected number of QoS in concrete service
Category Scaling assumption
Definition The number of QoS in concrete services is expected to be between 2 and 10.
Formal Definition $\forall$ s:Service $\Rightarrow 2 \leq \#\{s.QoS\} \leq 10$

Assumption Expected number of changes in QoS in concrete services
Category Scaling assumption
Definition The number of changes in QoS in concrete services is expected to be updated daily.
Formal Definition $\forall$ s:Service | LastUpdate(s.QoS) $\leq$ 24h

---

Assumption Expected length of the historical record of each QoS
Category Scaling assumption
Definition The length of the historical record of each QoS is expected to cover the last 30 days.
Formal Definition $\forall$ s:Service | HistoryLength(s.QoS) = 30 days

---

Goal Maintain[Updated evaluation of concrete services risk of fluctuation and QoS correlation between services for expected number of services requests, QoS, Qos changes, and limited length of the historical record]
Category Functional, scalability goal
Quality Variables storage space: application requests, QoS, QoS changes, length of historical record $\rightarrow$ disk space; execution time: application requests, QoS, QoS changes, length of historical record $\rightarrow$ time
Definition The system shall keep an up-to-date evaluation of the risk of fluctuations and QoS correlation between the services, as long as the value range of the relevant characteristics do not exceed the bounds stated in the scaling assumptions "Expected number of services requests," "Expected number of QoS in concrete services," "Expected number of changes in QoS in concrete services," and "Expected length of the historical record of each QoS."
Formal Definition Expected number of services requests $\wedge$ Expected number of QoS in concrete services $\wedge$ Expected number of changes in QoS in concrete services $\wedge$ Expected length of the historical record of each QoS $\Rightarrow \Box \forall$ s: Services | Status(s.fluctuation_risk) = "up to date" $\wedge$ Status(s.QoS_correlation) = "up to date"

Example 2: Achieve[Effective diversification of services composition]. This goal states that once a group of compatible services exist in the market, the buyer agent is responsible for effectively diversify the selection of services so its composition has a low risk of QoS fluctuation . This goal is satisfied when the buyer finds an optimal set of candidate services within an acceptable time. The load this goal imposes on the buyer is determined by the number of concurrent application

requests and number of QoS constraints. As the buyer agent has a limited capacity, it cannot handle an unlimited scaling of these application domain characteristics. This is represented in the model by the **scalability obstacle** Goal load exceeds Buyer agent capacity. As in the previous example, we introduced the **scaling assumptions** and replaced the original goal by a **scalability goal** as follows:

Assumption Expected number of concurrent application requests
Category Scaling assumption
Definition The number of concurrent application requests is expected to be between 1 and 50.
Formal Definition $\Box$ 1 $\leq$ # {r: ApplicationRequest | r.status = "Open"} $\leq$ 50

Assumption Expected number of QoS constraints
Category Scaling assumption
Definition The number of QOS attribute that the Application cares about is expected to be between 1 and 10.
Formal Definition $\forall$ a:Application $\Rightarrow$ 1 $\leq$ # {a.QoS_constraint} $\leq$ 10

Goal Achieve[Effective diversification of services composition under expected number of concurrent application requests and number of constraints for QoS]
Category Functional, scalability goal
Quality Variables execution time: application requests, QoS constraints $\rightarrow$ time
Definition The system diversifies the service composition in 5 minutes , as long as the value range of the relevant characteristics do not exceed the bounds stated in the scaling assumptions "Expected number of application requests" and "Expected number of QoS constraints"
Formal Definition Expected number of concurrent application requests $\wedge$ Expected number of QoS constraints $\Rightarrow$ $\forall$ m:Market, a: Application, s: Service | NeedRecomposition(a) $\wedge$ s $\in$ m.SelectedServices(a) $\Rightarrow$ $\Diamond_{\leq 5min}$ Compose(a, m.SelectedServices(a))

The same rationale can be applied to the analysis of the rest of the goals. As a result, three metrics and 12 application domain characteristics that may be relevant to the scalability analysis of this system have been uncovered. These are:

Application domain characteristics: number of concurrent application requests, workflow size, length of the historical record for QoS, number of QoS per abstract service in application workflow, number of QoS constraints in application workflow, number of changes in QoS or price of services, number of concrete services per abstract service in market, number of concrete services in the market, number of new services in the market, number of running applications, number of expired SLA at a given time, number of virtual machines.

Metrics: QoS fluctuation, execution time, disk storage space, optimality, cost, satisfaction rate

---

Table 3 depicts how this metrics and scaling dimensions relate to the responsibility of the system-to-be agents.

4.2.1 Obstacle Assessment and Resolution. As in the Clobmas exemplar, an obstacle assessment considered the likelihood and the criticality of obstacles to decide which obstacles should be further examined and resolved. Here again, we chose to do a first assessment on high-level scalability goals. Some of the obstacles are exemplified in Table 4. To ease the analysis of the likelihood and criticality of scalability obstacles, when a goal concerned more than one scaling dimensions, we created one

Table 3. Metrics and Unbounded Variables in Portfolio-based System Goals

ID	Goal	Metric	Unbounded variables
G2	Maintain[Information about services composition request and buyer preferences]	storage space	workflow size, num. of concurrent application requests, num.of constraints per QoS in application workflow
G5	Achieve[New services and change in existing services price or QoS is reported]	execution time	num. of new concrete services added to the market, num. of changes in QoS or price of existing concrete services
G6	Achieve[New services and change in existing services price or QoS is recorded]	storage space	num. of new concrete services added to the market, num. of changes in QoS or price of existing concrete services
G7	Achieve[Historical record of QoS stored]	storage space, execution time	num. of concrete services per abstract services in the market, length of historical QoS record, num. of changes in QoS or price of existing concrete services, num. of QoS per abstract service
G8	Maintain[An updated evaluation of concrete services risk of fluctuation and correlation between QoS]	storage space, execution time	workflow size, num. of concrete services per abstract service in the market, num. of changes in QoS and price of existing concrete services, num. of QoS per abstract service
G11	Achieve[Buyer informed if a concrete service for a given QoS and budget constraint does not exist]	execution time	num. of unmatched abstract services
G12	Achieve[Workflow and QoS budget desired for a service composition informed]	execution time	workflow size, num. of concurrent application requests, num.of constraints per QoS in application workflow
G13	Achieve[An updated information about concrete services available]	execution time	num. of concrete services in the market
G15	Achieve[Service composition is created form selected concrete services that share minimum correlation between their performances]	execution time, QoS fluctuation	workflow size, num. of concurrent application requests, num.of constraints per QoS in application workflow, num. of concrete services per abstract service in the market
G17	Achieve[Optimality of the composition isrecomposition by exposin all possible solutions]	execution time, QoS fluctuation, storage space, cost	num. of concrete services per abstract service in the market, workflow size, num.of constraints per QoS in application workflow
G18	Achieve[Set of candidate services found]	execution time	workflow size, num. of concrete services per abstract service in the market

obstacle to each dimension. See, for example, how the system architect created separate obstacles to the goal Achieve[Dynamic Composition of Services with Minimum QoS Fluctuation in 5 minutes] (in rows 3–6).

Then, we further evaluated through simulation the obstacles with medium to high likelihood or criticality. In Table 4, these are the obstacles that refer to the Achieve[Dynamic Composition of Services with Minimum QoS Fluctuation in 5 minutes] (obstacles three to six):

- **Obstacle 3**: The **number of concurrent application requests** exceeds the buyer agent capacity to achieve a dynamic service composition of services with minimum QoS fluctuation in less than 5 minutes
- **Obstacle 4**: The **number of QoS in application workflow** exceeds the buyer agent capacity to achieve a dynamic service composition of services with minimum QoS fluctuation in less than 5 minutes
- **Obstacle 5**: The **workflow size** exceeds the buyer agent capacity to achieve a dynamic service composition of services with minimum QoS fluctuation in less than 5 minutes
- **Obstacle 6**: The **number of candidate services per abstract services** exceeds the buyer agent capacity to achieve a dynamic service composition of services with minimum QoS fluctuation in less than 5 minutes

As shown in Figure 16, simulations revealed that while the system could cope with the number of candidate services per abstract services (obstacle 6), QoS attributes (obstacle 4), and concurrent application requests (obstacle 3), it could not handle scaling of the abstract services in the workflow (obstacle 5). More precisely, the scalability obstacle happened when both the number of candidate services and the workflow size scale concomitantly. This issue had not been identified in the previous analysis of the portfolio-based system.

It is worth noting that, for the scalability obstacle 4, the problem was caused by the sub-goal Achieve[Optimality of the composition by exploring all the possible solutions], which ensured the minimum QoS fluctuation. The scalability obstacle was resolved by applying the resolution *weaken the goal objective function* [Duboc et al. 2013a], replacing that goal by Achieve[Near-Optimal Composition by Exploring Part of the Possible Solutions].

Table 4. Assessment of Likelihood (Li) and Criticality (Cr) of Scalability Obstacles, Classified as High (H), Moderate (M), and Low (L) for Clobmas

ID	Scalability Obstacle	Li	Cr	Rational
1	The **length of the historical record for QoS** exceeds market regulator capacity in term of storage space.	L	L	We will consider only recent recorded, which is limited to the last three months. Because of that the likelihood of representing scalability obstacle is low. Moreover, the system goal Achieve [Historical record of Qos is stored] dictate that historical record must be stored. This process will represent low criticality in system scaling, as additional storage is cheap and can easily outsource from the cloud market.
2	The **number of new services add to the market** exceed market regulator ability to maintain an updated information about concrete services price and QoS.	L	L	We assume that number of new services can reach hundreds of services each day. However, the services will be added at different time through the day that will make the likelihood of having a larger number of new services add to market at given time low. Moreover, the system goal Achieve [New services and change in existing services price or functionality is recorded] and Maintain [An updated information about concrete services price and QoS] stat that new services must be recorded and monitored. However,the number of new services will represent low criticality on the system scalability as a limited activity are performed in new services addition. These activities include adding the service information to the market repository and start monitoring and recording its performance.
3	The **number of concurrent application requests** exceeds the buyer agent ability to achieve a dynamic composition of services with minimum OoS fluctuation in less than 5 minutes.	H	H	As the cloud market represents an economical and attractive option for publishing web services, we assume that the number of its users will grow which will make the likelihood of having concurrent application request high. Moreover, a specified by the goal Maintain[Information about services composition request and buyer preference] each composition request will create an additional load on the buyer agent as specified by the goal, which is high critically on system scalability.
4	The **number of QoS in application workflow** exceeds the buyer agent ability to achieve a dynamic services composition with minimum QoS fluctuation in less than less than 5 minutes.	H	H	We assume that the number of QoS in application workflow can reach 10 QoS dimensions. For that, the likelihood of representing scalability obstacle is high. The system goal Achieve[Services composition is selected based on desired workflow and constraints] state that QoS constraints should be maintained when selecting a solution .where each QoS dimension will expand the number of factors needed to be considered at different stages of the service selection process. First, in the initiation screening of the services for compatibility. Then, in the optimisation process where each QoS dimension will multiply the calculations required to find a solution. This is because each Qos dimension will represent an objective of the selected services. For that increasing number of QoS dimensions will represent high criticality on the system scalability.
5	The **workflow size of an application** exceeds the buyer agent ability achieve a dynamic services composition with minimum QoS fluctuation in less than less than 5 minutes.	H	H	We assume that The workflow size can reach 20 services for a complex application where it is necessary to select an optimal candidate service for each service in the workflow. For that the likelihood of representing scalability obstacle is high. Moreover, the system goal Achieve [Services composition is selected based on desired workflow and constraints] state the system should consider each service in the workflow. In this case, each additional service in the workflow will dramatically increase the size of the problem and that will reflect on the time needed to find a solution. For that, any increase in the workflow size will represent high criticality on the system scalability.
6	The number of candidate services per abstract services in the application workflow exceeds the buyer agent ability to achieve a dynamic services composition with minimum QoS fluctuation in less than less than 5 minutes.	H	H	We assume that the candidate services per abstract services can be high when we consider the large number of the compatible services offered in the market . For that, the number of candidate services per abstract services have high likelihood of presenting a scalability obstacle. Moreover, the system goal Achieve[Optimality of the composition by exploring all the possible solutions] states that, in the optimisation process we need to consider every candidate services available in the market to ensure that we achieve optimality of the selected composition. For that increasing number of candidate services per abstract services will represent high criticality on the system scalability.

5 EVALUATION

As explained in Section 1, our **Research Objectives (RO)** were (i) to evaluate the usefulness of a systematic scalability evaluation technique in the domains of QoS-aware dynamic service composition; and (ii) to provide practitioners with guidance and transferable knowledge for evaluating the

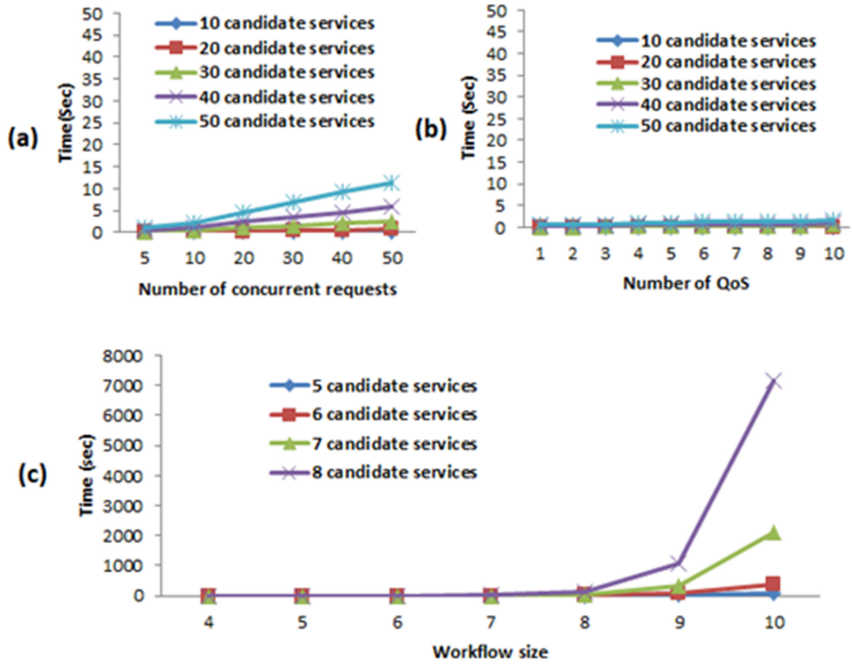

Fig. 16. The execution time of the portfolio-based composition as both of the number of candidate services and (a) number of requests, (b) number of QoS, and (c) workflow size increases.

scalability of systems that fundamentally build on dynamic service composition (e.g., multi-tenant cloud-based services composition, smart cities).

We now discuss how we have achieved these objectives. Then, we comment on other issues related to the use of goal modeling for scalability analysis.

5.1 RO1: Usefulness of the Systematic Technique

We evaluate the usefulness of the systematic scalability goal-modeling technique in the domain of dynamic service composition from two perspectives. First, we contrast the scalability analysis carried out as a result of the goal-modeling technique with the previous analysis that had been made for both systems. Second, we compare the list of the application domain characteristics and metrics normally considered in the literature with the ones uncovered by the goal-modeling technique.

5.1.1 Comparison with Previous Scalability Analysis. Prior to this research, the chosen application domain characteristics and metrics used for scalability analysis were heavily informed by seminal work in the area and to provide common grounds for comparison with these seminal techniques. Following this work, a number of additional characteristics and metrics were uncovered for both systems. Based on them, we have conducted a number of simulations for the most likely and critical obstacles in both systems. The simulation has revealed scalability problems that have been missed when analysts relied on ad hoc judgment and/or mimicked previous approaches.

The goal-oriented scalability analysis therefore rendered previous observations about scalability of these systems obsolete and challenged the conclusions that relied on previous metrics. Additionally, as the new set of application domain characteristics and metrics were uncovered through systematic modeling, they also provided a more comprehensive and less biased set of concerns

Table 5. Scaling Dimensions and Ranges Uncovered in Previous Literature Review

Scaling Dimensions (number of occurrences)	Metrics (number of occurrences)
Workflow size (44), *number of candidate services per abstract service* (53), *number of QoS dimensions* (4), *number of QoS constraints* (6), *number of global QoS constraint* (1), *number of quality levels* (1), *number of state nodes* (4), *number of supplementary services per abstract services* (1), *number of breaches of services* (1), *number of user QoS requirements* (2), *number of business process tasks*(1), *number of requests per day* (1), *arrival rate* (1), *number of users* (3), *number of requests per time interval* (1), *size of historical record* (1), *number of QoS registries* (1), *correlation percentage* (2), *amount of data transmission between services* (1).	*Execution Time* (38), *satisfaction/failure rates* (7), *utility* (8), *optimality* (13), *cost* (2), *fitness value of the algorithm* (2), *availability* (1), *speedup* (1), *average of violated quality of service* (1), *time complexity* (1), *interruption time caused by reselection process* (1), *overall aggregated QoS* (1), *memory usage* (1), *impact on QoS weight* (1), *effectiveness of the performance prediction* (1) *interrupting time caused by reselection process* (1), *operating costs of operationalization* (1), *learning rate* (1), *service providers* (1).

that the evaluation shall consider and can be sensitive to. Below, we discuss our observations for each system.

Simulations of Clobmas. Informed by the literature, the original scalability analysis of Clobmas considered execution time against the workflow size and the number of candidate services per abstract service. That analysis showed that the system performed very well with respect to these dimensions, with only a very low polynomial growth in adaptation time.

Following this work, **15 other application domain characteristics** and **three other metrics** that are potentially relevant to the scalability of Clobmas have been revealed. As explained in Section 3.2.2, two of application domain characteristics were enough to challenge the scalability of the system: number of markets and number of QoS attributes, both of which were unbounded in the first goal model, leading to a exponential growth in adaptation time.

Simulations of Portfolio-based system. Scalability analysis of portfolio-based composition considered execution time and QoS fluctuation against the number of candidate service for each abstract service. Results had shown that the portfolio-based techniques tend to scale and perform well on these dimensions. In contrary, goal-oriented scalability analysis has revealed **11 other application domain characteristics** and **three new metrics** that were overlooked. Based on the new input, the system architect revisited the experiments. While the system scaled well for dimensions such as the number of QoS attributes and the number of concrete application requests, it exhibits an exponential growth in execution time when we increased the workflow size of the application. These results challenged the preliminary conclusions, rendering the system far from being scalable.

Based on these observations, we conclude that the systematic goal-oriented scalability analysis technique was effective in uncovering characteristics and metrics that were relevant to the system at hand and in guiding the scalability experiments through the consideration of the likelihood and criticality of scalability obstacles.

5.1.2 Comparison with Scalability Analysis in the Literature. We now discuss the usefulness of the goal-modeling technique in the domain of QoS-aware dynamic service composition by looking at the characteristics and metrics considered in existing work and their occurrences and discussing how the systematic procedure can provide a useful alternative.

Table 5 evidences the application domain characteristics (i.e., scaling dimensions) and metrics considered in the literature of QoS-aware dynamic service composition, along with their occurrences. We base our findings in our previous reviews [Duboc et al. 2018, 2013b] and an updated scalability analysis of newer works, which is summarized in Table 6.

Table 6. Summary of Selected Scalability Analysis in QoS-Aware Service Composition

Initiative	Scalability Dimension				Evaluation Metric for Scalability				
	Workflow Size	# of Candidates per Abstract Service	# of QoS Constraints	Other	Execution Time	Satisfaction/Failure Rates	Utility	Optimality	Other
[Moustafa and Zhang 2013]	–	✓	–	–	–	–	–	✓	–
[Wang et al. 2016]	–	✓	–	✓	–	✓	–	–	✓
[Wang et al. 2017]	–	✓	–	✓	–	–	–	✓	–
[Moustafa and Ito 2018]	–	✓	–	✓	✓	–	–	✓	–
[D'Angelo et al. 2020]	✓	–	✓	–	–	–	–	✓	–
[Caporuscio et al. 2015]	✓	✓	–	–	–	–	✓	✓	–
[Souri et al. 2020]	–	–	✓	–	✓	–	–	–	✓
[Wang et al. 2019]	✓	✓	–	–	✓	–	–	✓	–
[Liang et al. 2021]	✓	✓	–	–	–	–	–	✓	–
[Wang et al. 2020]	–	✓	–	✓	✓	–	–	✓	–
[Guidara et al. 2020]	–	✓	✓	–	✓	✓	✓	–	–
[Yuan et al. 2018]	–	✓	–	–	✓	–	✓	–	–
[Dahan et al. 2021]	✓	✓	–	–	✓	–	–	–	–
[Hosseini Bidi et al. 2021]	–	✓	–	–	✓	–	✓	–	–
[Peng et al. 2020]	–	✓	–	–	–	–	–	✓	–

When looking at this table, the first thing to note is the lack of conscience about the dimensions and metrics for scalability per say, as most dimensions and metrics are considered in at most one paper. It is also evident that there is representation for some of the dimensions and metrics under different naming and therefore a lack of "unified" terminology to describe the issues. Finally, although there is common agreement between the most obvious scaling dimensions and metrics (i.e., see the ones with the most occurrences), a closer analysis of the papers reveals that in most cases no justification is given for the choice of variables.

Conversely, in the scalability goal-obstacle analysis, the justification of variables is grounded on sound modeling for the system's goals and their obstacles. We argue that the modeling is worth the effort, as it helps to identify: (i) the application domain characteristics and metrics that are relevant to the system and (ii) the goals such characteristics are affecting, and consequently to visualize and mitigate the risks associated with the scaling of these characteristics. Such judgment could be hard in the absence of systematic modeling. Moreover, the goal-modeling technique can overcome the chaos and ad hoc attempts, and it may provide a useful tool to **unify/aggregate the scaling dimensions and metrics** in the domain of QoS-aware dynamic service composition.

To give a first step towards this aim, we summarize in Table 7 the variables uncovered in the scalability goal-obstacle analysis of Clobmas and the portfolio-based system. Some observations must be made regarding this table:

(1) Some of the dimensions and metrics were distinct to Clobmas and the Portfolio-based technique. This can be attributed to (i) the fundamental differences in the solution formulation, each when informed by the chosen theory that underlies the architecture (e.g., one leverages market-based while the other uses Modern Portfolio); (ii) assumptions, design decisions, and architecture styles that induced the implementation of the solutions. Henceforth, it is imperative that the architect and analyst should account for the above factors when analyzing for scalability.

(2) Yet, it is interesting to note that many characteristics and metrics are common between the two exemplars. For this reason, we are inclined to think that this compilation of application domain characteristics and metrics can be useful as a starting point to inform the scalability analysis of other systems in QoS-aware dynamic services composition. This can be particularly useful when the analysis can be linked to the goals of the system and consequently to the likelihood and criticality of scalability obstacles; the architect can then prioritize and justify the exercise. However, this is a hypothesis that requires further research to be validated.

(3) When compared to Table 5 and accounting for the difference in terminology, one can see that many of the application domain characteristics in the literature—or proxies of these variables—have also been considered relevant in the modeling technique (see emphasized characteristics on Table 5). Others are not considered relevant for the systems at hand, such as the number of QoS registries (assumed to be one per market), the cost of concrete services (which is a constraint on the application requests), and the amount of data transmission between systems (which will only affect the scalability of the composed applications themselves). Admittedly, the correlation percentage (which refers to the fact that the QoS attributes of a service are not only dependent on the service itself but also correlate to other services) has been overlooked in the modeling.

(4) Most scalability analyses of QoS-aware dynamic service composition systems in literature are measured against a single metric, normally execution time. The systematic goal modeling allows a more careful exploration of the system metrics by looking at the criteria for the satisfaction of goals. For example, if there is a goal to store historical data, then storage

Table 7. Application Domain Characteristics and Metrics in the QoS-aware Dynamic Service Composition Domain

Application domain characteristic	**Description**	**Exemplar**
Num. of concurrent application requests	Number of requests to compose applications dynamically that are issued at the same time	Both
Num. of running applications	Number of applications running at the same time	Both
Num. of alternative QoS and cost requests	Number of requests broken down into different combinations of QoS and costs for the services composing the application	Clobmas
Workflow size	The size of the workflow that defines how the concrete services should be combined to compose the application	Both
Num. of QoS per abstract service	Number of quality-of-service attributes required for each abstract service in the workflow	Both
Num. of concrete services in the market	Number of concrete services that co-exist in a given market	Portfolio
Num. of concurrent concrete services per abstract service in the market	Number of concrete services in the market that offers the functionality of a given abstract service in the workflow	Both
Num. of new concrete services added to the market	Number of concrete services that are added to a particular market at some point in time	Both
Num. of markets per abstract service	Number of markets in which an abstract service may be found	Clobmas
Num. of constraints for QoS in the application workflow	Each QoS attribute that the Application cares about and requires a minimum value	Both
Type of QoS constraints	Either summative (e.g., cost) or projective (e.g., reliability)	Clobmas
Num. of changes in the QoS or price of existing concrete services	Number of changes that a given concrete service may have in its announced QoS properties and costs in a given time interval	Both
Num. of matched abstract services	Number of abstract services in an application request that have been matched to a concrete service in a market	Clobmas
Num. of unmatched abstract services	Number of abstract services in an application request that have not been matched to a concrete service in a market	Clobmas
Num. of expired SLAs at a given time	Number of SLAs of concrete services composing an application that expire in a given time period	Both
Num. of renewed SLAs at a given time	Number of SLAs of concrete services composing an application that are renewed in a given time period	Clobmas
Num. of SLAs violations at a given time	Number of SLAs of concrete services composing an application that violated in a given time period	Clobmas
Length of historical QoS record	Number of day before old QoS data are deleted from the system.	Portfolio
Length of changes in QoS and price of existing services	Number of changes in QoS and price of services composing an application.	Portfolio
Num. of virtual machines	Number of virtual machines available in the data center	Portfolio
Metric	**Description**	**Exemplar**
Execution time	Time to complete an operation of interest	Both
Storage capacity	How much disc space is used to store data	Both

(Continued)

Table 7. Continued

Application domain characteristic	Description	Exemplar
Utility	QoS values normalized and summed across all QoS that an application is interested in (both, required, and achieved)	Clobmas
QoS fluctuation	The standard deviation of QoS dimensions	Portfolio
Cost	The overall cost of the composed application	Both
Satisfaction rate	Number of applications requests that were satisfied	Both

space will be a concern. Conversely, if there is no goal stating the desired availability of the system, then this quality will not be measured in a scalability analysis. As a result, one will notice that the metrics of interest in the exemplars are limited when compared with the variety listed in Table 5. This happens because they reflect the goals, and consequently, the metrics with which the system architect was concerned with.

(5) The modeling can help the analyst to consider emerging concerns/dimensions for scalability that were not previously discussed in the literature. As an example, consider the case of the number of markets: The majority of works on QoS-aware service composition assume the existence of one market or a single services registry. As the concept of cloud markets started to materialize, this dimension emerged to be among the dimensions that the analysis should consider in assessing the scalability of their solutions. The modeling provided sound grounds for conducting systematic analysis for the goals and obstacles that relate to this dimension. As we have seen from the exemplar, the analysis provided means to discuss tradeoff decisions that relate to scalability vs. the number of markets and to reconcile inputs from experts and systematic modeling/simulation.

(6) The modeling is generic enough and can be further refined to consider scalability related properties of the underlying environment. For example, end-to-end performance guarantee for cloud service composition [Huang et al. 2015] can entail environment-specific properties for scalability that relate to the underlying network and cloud configuration, computation overheads, transmission and delivery mode, performance targets, and so on.

5.2 RO2: Guidance and Transferable Knowledge

One of the objectives of the study has been to provide guidance and transferable knowledge by means of (i) a detailed explanation of the goal-modeling technique in the two exemplars and (ii) a list of application domain characteristics and metrics that might be relevant to their solutions. To evaluate this research objective, we discuss how the modeling and exemplars can potentially help the architects of QoS-aware dynamic service composition by reflecting on our own experience.

As with any modeling activity, the elaboration of the goal model was not trivial. For the Clobmas, the model was mostly built by the first author of the article, who has extensive experience with KAOS. Constructing the model and performing the scalability goal-obstacle analysis required a number of iterations with the Clobmas architect. This task took the primary modeler around two weeks, spread over a couple of months. Considering also the time spent in discussions and joint modeling, we estimate the total effort in 90 man-hours. In the first "week," the exercise involved several meetings between the modeler and Clobmas architect to understand the purpose of the system, its working procedure, implementation, assumptions, and so on. As a result, a preliminary model was developed. The subsequent "week" involved another round of meetings between the modeler and Clobmas architect, resulting in asynchronous iterations and subsequent refinement of the models for performing the scalability analysis.

The portfolio-based system, however, used the Clobmas model as starting point. The model was built by the third author of this article, who was the architect of the system with no previous experience with KAOS. It took him four days to create the first model and two days for performing the scalability-obstacle analysis and refining the model (i.e., 48 man-hours). We believe this happened because these solutions have many goals and assumptions in common within the problem domain (e.g., creating several combinations of QoS for the abstract services on the workflow, finding candidate services to an abstract service, informing the buyer if matches were found, dealing with broken SLAs). This commonalities have eased the partial reproduction of goals in Clobmas on the portfolio-based system. However, one should also note that each system is unique, and although they belong to the same problem domain and share a number of concerns, they will also have their own specific goals. The portfolio-based system, for example, was concerned with the QOS fluctuation, which was not relevant to the Clobmas. As a result, nine application domain characteristics and three metrics were common between both systems.

In interpreting these results, one should also note that KAOS has a steep learning curve. Usually, first-time modelers make modeling mistakes and need to correct their models until they grasp the KAOS concepts. This happened in Clobmas: The first attempt to build the model was done by a Clobmas architect himself, with no experience with KAOS. The model contained the typical modeling mistakes and had to be rebuilt nearly from scratch by the first author. The story was different with the portfolio-based system. The first-time modeler, who is also the portfolio-based architect, built the model with almost no input from the KAOS expert.

Although reduced in the second study, the initial modeling effort should be taken into consideration. We do, however, believe that we have demonstrated the advantages of using a systematic technique to a scalability analysis, as well as shown that the resulting model can be used as a starting point to other solutions in the realm of dynamic QoS-aware services composition.

5.3 Observations on Goal Modeling for Scalability Analysis

We now discuss other issues related to the use of goal modeling for scalability analysis.

5.3.1 Completeness of the Model. It is worth emphasizing that using goal modeling will not guarantee that application domain characteristics or metrics will not be overlooked; these can only be as good as the model itself. Nevertheless, a systematic approach offers a sensible alternative to ad hoc practices and random guesses and is more likely to reveal the variables that may be relevant to the scalability analysis.

5.3.2 Unbounded Scaling Dimensions and Metrics. Not all application domain characteristics will lead to scalability problems. In practice, many of them will vary within small ranges, imposing very little load on the agent. When going through the assessment stage of the scalability goal-obstacle analysis, potential scalability obstacles associated with these variables may be considered unlikely and may be left unresolved. Take, for example, the goal Achieve [Historical record of QoS stored] in the Portfolio-based system. The system is only required to store the data regarding the past 30 days. Therefore, the obstacle Length of the historical record for QoS exceeds market regulator capacity can be left unresolved.

The same can be said about the metrics associated with each goal. For the Clobmas, for example, one may observe in Table 1 that "execution time" is considered relevant to many goals. This happens because these goals contribute to the high-level goal Achieve [Application composed quickly], whose satisfaction criteria is determined with respect to execution time. In practice, a number of goals will have very little impact on the metric and may be left out of the scalability analysis.

This, however, must be a conscious decision, which requires these variables to be at least identified and an assumption about their possible range of values documented.

5.3.3 Decomposition of Software-to-be Agents. KAOS was not conceived for modeling the solution space. For this reason, in a goal model exercise, normally one stops refining when a goal is assigned to the software-to-be, not further decomposing it into separate agents. Nevertheless, as observed by Duboc et al. [2013a], sometimes it is worth decomposing the software-to-be into finer-grained agents. In the case of a decentralized solution, as the ones modeled in this article, further decomposition can be very useful for identifying scaling dimensions that otherwise could be overlooked. Take, for example, the extract of the goal model in Figure 14. In reality, the goal Achieve[Effective diversification of service composition] is achieved by a combination of two finer-grained software-to-be agents (MarketRegular and BuyerAgent). By decomposing this goal, one makes the sub-goal Achieve[Buyer informed if a concrete service for a given QoS and budget constraints does not exist] explicit, which leads to the identification of the scaling dimension number of unmatched concrete service, which did not affect any other goal on the model.

5.3.4 System Design Characteristics. Decomposing the software-to-be agents can also lead to the identification of scaling dimensions belonging to the system design. In the exemplars, while we have decomposed the software-to-be to a degree, we avoided entering too much into the solution space to create goal models that could be used as a starting point to model other applications in the realm of QoS-aware dynamic service composition.

5.3.5 Unstated Hypotheses Affecting Scalability. The systematic goal modeling can also reveal unstated hypothesis that can have an effect on the scalability of the system. Take, for example, the goal Achieve[Service selected if concrete service that meet QoS and budget exists]. When analyzing this goal, one may wonder what happens with the scheduling and allocations of services tasks in a shared and multi-tenant environment. The order/priority services scheduled and executed can be a critical input for scalability analysis in multi-tenant and shared environment. Not all services should be given the same priority, as resource-hungry services can affect the scalability of the system. The above analysis made the architect of the portfolio-based system aware of an unstated assumption that All services have equal priorities, prompting him to rethink the priorities of services on future applications that leverage his solution.

6 RELATED WORK

6.1 Goal-modeling Technique

Goals and requirements for dynamic adaptive systems can be considered at one of the following levels [Berry et al. 2005; Krupitzer et al. 2015]: (1) at the overall definition of the system; (2) at runtime when the system changes to adapt itself to its environment; (3) at design time for the assessment of the adaptation aspects that will enable the system to perform adaptations as needed at level 2; and (4) at research to explore adaptation mechanisms. To our knowledge, our work is the first effort to apply a systematic scalability analysis at level 3.

The goal modeling of self-adaptive systems has available several frameworks [Weyns 2019] such as KAOS [Dardenne et al. 1993], i* [Yu 1997], Tropos [Castro et al. 2002], or LoREM [Goldsby et al. 2008]. However, KAOS is the most commonly used by the community [Yang et al. 2014]. Moreover, it is the most extended to produce new frameworks such as FLAGS [Baresi et al. 2010]; or be combined with requirements languages such as RELAX [Whittle et al. 2010] or AutoRELAX [Fredericks et al. 2014], whose constructs are oriented towards explicit support for specifying and dealing with runtime uncertainty in goals due to environmental changes.

In this context, we decided to adopt KAOS for this research. Our decision was also guided by two major considerations. First, the flexibility of KAOS as a generic modeling for goals of the system, obstacle on goals, subsequent refinements, and tradeoffs analysis [Bennaceur et al. 2019; Whittle

et al. 2010]. When KAOS is used to support the design of AAS, the modeling can inform the design of adaptation decisions and mechanisms, considering constraints and likely obstacles [Krupitzer et al. 2015]. Second, we have already extended KAOS in a previous work [Duboc et al. 2013a] to elaborate scalability goals through obstacle analysis. To the best of our knowledge, we are the first to tailor our goal-obstacle analysis for scalability modeling in the domain of dynamic service composition.

Vakili and Navimipour [2017] classified service composition approaches as framework-based, agent-based, and heuristic-based. Framework-based approaches refer to mechanisms that implement a completely original approach based on specific assumptions, concepts, practices, and values; agent-based methods are built on computational models that simulate autonomous agents interacting among them to assess the adaptations of the composition; and heuristic-based approaches are built on either heuristic or meta-heuristic algorithms. Although our technique decomposes goals until each is assigned to a single agent, it is not only for the scalability analysis of agent-based service composition approaches [Wang et al. 2017, 2016] but also for any other work whose underlying design depends on agents such as the heuristic-based approach of Moustafa and Zhang [2013] that combined multi-objective optimization with reinforcement learning.

6.2 Scalability Analysis in Dynamic Service Composition

Scalability analysis has been considered in the evaluation of dynamic service composition solutions from perspectives attached to specific deployment environments such as cloud [Jula et al. 2014; Vakili and Navimipour 2017], IoT [Arellanes and Lau 2020; Asghari et al. 2018], or other distributed environments [Caporuscio et al. 2015; Su et al. 2008]. However, a system that has not been built with scalability in mind cannot become scalable simply by being deployed in a scalable platform [Duboc et al. 2013a]. Therefore, different from previous approaches, we propose a methodology that supports the reflection on the scalability of the dynamic service composition solution at design time independently from the deployment environment. Moreover, we make emphasis on the dimensions and metrics for scalability, which are loosely coupled to the dynamic service provision of the underlying environment.

The research on QoS-aware dynamic service composition has mainly focused on availability, reliability, response time, cost, among others [Hayyolalam and Kazem 2018; Jatoth et al. 2015; Kumar 2021]. Then, although there exists previous works related to the analysis of scalability [Duboc et al. 2018; Moghaddam and Davis 2014; Sykes et al. 2011], there is little consensus in terms of the scaling dimensions, which rarely go beyond the *workflow size* [Calinescu et al. 2010] and *number of candidate services* [Caporuscio et al. 2015]. Most notably, existing works fail to derive their dimensions or metrics systematically [Hassan 2019]. Conversely, this article presents two exemplars that use goal modeling to systematically unveil dimensions and metrics that can be used in the scalability analysis. Not all dimensions will be necessarily used in an analysis; nevertheless, this technique helps the stakeholder to make informed decisions based on the goals of the system and to document the rationale.

In addition to the scalability analysis of individual solutions, there are works that compare different strategies for services composition. For example, Ghezzi et al. [2015] compare different service selection strategies based on a framework that supports performance estimation. Their work is based on only one metric (response time) and two application domain characteristics (number of clients and number of service providers on a concrete target set). Therefore, if a scalability problem were not caused by the scaling of these characteristics, then it could be overlooked. Our work differs from theirs in the sense that it allows to identify application domain characteristics and metrics that are potentially relevant to a specific system.

7 CONCLUSIONS AND FUTURE WORK

In this work, we have discussed the problem of scalability evaluation of QoS-aware dynamic service composition. Complex autonomous services computing systems (e.g., multi-tenant cloud services, mobile services, services discovery, and composition in smart environments) have been leveraging DSC to dynamically and adaptively maintain the desired QoS, cost, and to stabilize long-lived software system. Combining services while optimizing for QoS is known to be an NP-hard problem. Therefore, scalability is a major concern in the field. A previous systematic review of the literature by the authors [Duboc et al. 2018, 2013b] revealed that the analysis carried out in the field considers the scaling of a variety of characteristics that belong to the application domain and/or to the system design. While the latter are specific to each approach, the former may be common across solutions. Yet, only a couple of characteristics (workflow size and number of concrete services) are generally considered. Attempts to analyze scalability carry little justification for the chosen dimensions and metrics; effort seems to heavily mimic seminal work and/or rely on existing literature to arrive on the dimensions and metrics and/or to provide the groundwork for comparison with existing solutions.

We advocate for a systematic evaluation of scalability in the domain of QoS-aware **Dynamic Service Composition (DSC)**. A systematic evaluation can help to uncover the characteristics that might affect the scalability of the solution; to make informed decisions about which characteristics to consider in the analysis; and/or to test for more specific scenarios that are particularly likely and critical for the problem of DSC.

In this article, we have reported on a new application of the goal-oriented scalability analysis technique [Duboc et al. 2013a] in the DSC domain and created transferable exemplars. Moreover, the systematic goal/obstacles modeling for scalability of DSC offers the flexibility and openness to incorporate new scalability concerns and queries that were not commonly discussed in the DSC literature. This, for example, can be attributed to the infancy of the environment (e.g., when extending the coverage of DSC beyond services registries to consider cloud marketplaces) and/or emerging behaviors as a result of the new environment. It can also be attributed to the fundamentals of the DSC solution itself—its formulation, underlying theory, assumptions, and implementation—which may require customized analysis but can still benefit from the existing dimensions, metrics, and exemplars.

The two detailed exemplars aim to assist architects and designers in the systematic evaluation of systems that underlie DSC. The resulting models of the two exemplars can be re-used by architects and designers of QoS-aware dynamic composition solutions as preliminary input to guide the modeling of other systems. Our experience in creating these exemplars shows that: (1) the technique helped to identify characteristics and metrics that had been overlooked in previous scalability analysis of these applications and, (2) by having a existing model as a starting point, the modeling effort was reduced and could be performed by a non-expert. However, by no means should the exemplars be considered as a complete and definite compilation of the concerns into modeling. Subsequent refinements and elaboration of these models are expected before they qualify to act as reference models. Variants models can be also expected to reflect on domain characteristics.

In addition to the models, we compiled a list of application domain characteristics and metrics that may be relevant to other solutions in the field; either as a quick reference or as an input for creating other models. The usefulness and completeness of this list will be assessed in future research.

Future research directions would be to gather and synthesize independent modeling attempts for the problem of DSC from practitioners and researchers in the field. These attempts can benefit from our contribution as a preliminary guide. The objective is to arrive on a holistic reference

goal model, metrics and resolution tactics developed for dynamic QoS-aware service composition, with adaptation guidelines covering wider range of environments, applications, and underlying techniques for DSC.

ACKNOWLEDGMENT

The authors thank Emmanuel Letier for discussions on the model of Clobmas.

REFERENCES

F. Alrebeish and R. Bahsoon. 2015. Stabilising performance in cloud services composition using portfolio theory. In *Proceedings of the IEEE International Conference on Web Services (ICWS)*. IEEE, New York, NY, 1–8. DOI: https://doi.org/10.1109/ICWS.2015.11

Damian Arellanes and Kung-Kiu Lau. 2020. Evaluating IoT service composition mechanisms for the scalability of IoT systems. *Fut. Gen. Comput. Syst.* 108 (2020), 827–848.

Parvaneh Asghari, Amir Masoud Rahmani, and Hamid Haj Seyyed Javadi. 2018. Service composition approaches in IoT: A systematic review. *J. Netw. Comput. Applic.* 120 (2018), 61–77.

R. Bahsoon and W. Emmerich. 2008. An economics-driven approach for valuing scalability in distributed architectures. In *Proceedings of the 7th Working IEEE/IFIP Conference on Software Architecture (WICSA'08)*. IEEE Computer Society, 9–18.

Luciano Baresi, Liliana Pasquale, and Paola Spoletini. 2010. Fuzzy goals for requirements-driven adaptation. In *Proceedings of the 18th IEEE International Requirements Engineering Conference*. IEEE, 125–134.

Amel Bennaceur, Thein Than Tun, Yijun Yu, and Bashar Nuseibeh. 2019. Requirements engineering. In *Handbook of Software Engineering*. Springer, Cham, 51–92.

Daniel M. Berry, Betty H. C. Cheng, and Jia Zhang. 2005. The four levels of requirements engineering for and in dynamic adaptive systems. In *Proceedings of the 11th International Workshop on Requirements Engineering Foundation for Software Quality (REFSQ)*.

Martin Bichler, Marion Kaukal, and Arie Segev. 1999. Multi-attribute auctions for electronic procurement. In *Proceedings of the 1st IBM IAC Workshop on Internet-based Negotiation Technologies*. Citeseer, 18–19. Retrieved from http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.17.8499&rep=rep1&type=pdf.

N. Borissov, D. Neumann, and C. Weinhardt. 2010. Automated bidding in computational markets: An application in market-based allocation of computing services. *Auton. Agents Multi-Agent Syst.* 21, 2 (2010), 1–28. Retrieved from http://www.springerlink.com/index/41272l185u852475.pdf.

Radu Calinescu, Lars Grunske, Marta Kwiatkowska, Raffaela Mirandola, and Giordano Tamburrelli. 2010. Dynamic QoS management and optimization in service-based systems. *IEEE Trans. Softw. Eng.* 37, 3 (2010), 387–409.

Mauro Caporuscio, Vincenzo Grassi, Moreno Marzolla, and Raffaela Mirandola. 2015. Go prime: A fully decentralized middleware for utility-aware service assembly. *IEEE Trans. Softw. Eng.* 42, 2 (2015), 136–152.

Jorge Cardoso, Amit Sheth, John Miller, Jonathan Arnold, and Krys Kochut. 2004. Quality of service for workflows and web service processes. *Web Semant.: Sci., Serv. Agents World Wide Web* 1, 3 (Apr. 2004), 281–308. DOI: https://doi.org/10.1016/j.websem.2004.03.001

Jaelson Castro, Manuel Kolp, and John Mylopoulos. 2002. Towards requirements-driven information systems engineering: The Tropos project. *Inf. Syst.* 27, 6 (2002), 365–389.

Yee-Ming Chen and Hsin-Mei Yeh. 2010. Autonomous adaptive agents for market-based resource allocation of cloud computing. In *Proceedings of the 9th International Conference on Machine Learning and Cybernetics*. IEEE, 2760–2764.

Fadl Dahan, Wojdan Binsaeedan, Meteb Altaf, Mahfoudh Saeed Al-Asaly, and Mohammad Mehedi Hassan. 2021. An efficient hybrid metaheuristic algorithm for QoS-aware cloud service composition problem. *IEEE Access* 9 (2021), 95208–95217.

Anne Dardenne, Axel Van Lamsweerde, and Stephen Fickas. 1993. Goal-directed requirements acquisition. *Sci. Comput. Program.* 20, 1–2 (1993), 3–50.

Tom De Wolf and Tom Holvoet. 2006. *A Catalogue of Decentralised Coordination Mechanisms for Designing Self-organising Emergent Applications*. Technical Report CW458. Katholieke Universiteit Leuven, Department of Computer Science.

Jiang Dejun, Guillaume Pierre, and Chi-Hung Chi. 2009. EC2 performance analysis for resource provisioning of service-oriented applications. In *Proceedings of the International Conference on Service-oriented Computing (ICSOC/ServiceWave'09)*. Springer-Verlag, Berlin, 197–207. Retrieved from http://dl.acm.org/citation.cfm?id=1926618.1926641.

Leticia Duboc, Faisal Alrebeish, Vivek Nallur, and Rami Bahasoon. 2018. Summary of a literature review in scalability of qos-aware service composition. *CoRR* abs/1810.033014 (Apr. 2018), 5.

L. Duboc, E. Letier, and D. S. Rosenblum. 2013a. Systematic elaboration of scalability requirements through goal-obstacle analysis. *IEEE Trans. Softw. Eng.* 39, 1 (Jan. 2013), 119–140. DOI: https://doi.org/10.1109/TSE.2012.12

Leticia Duboc, Vivek Nallur, and Rami Bahsoon. 2013b. Escalabilidade em Composição Dinâmica de Serviços baseada em QoS: uma Revisão Sistemática. *Cadernos do IME. Série Informática* 35 (2013), 7–22. Retrieved from http://www.e-publicacoes.uerj.br/index.php/cadinf/article/view/7991.

Mirko D'Angelo, Mauro Caporuscio, Vincenzo Grassi, and Raffaela Mirandola. 2020. Decentralized learning for self-adaptive QoS-aware service assembly. *Fut. Gen. Comput. Syst.* 108 (2020), 210–227.

Erik M. Fredericks, Byron DeVries, and Betty H. C. Cheng. 2014. AutoRELAX: Automatically RELAXing a goal model to address uncertainty. *Empir. Softw. Eng.* 19, 5 (2014), 1466–1501.

Carlo Ghezzi, Valerio Panzica La Manna, Alfredo Motta, and Giordano Tamburrelli. 2015. Performance-driven dynamic service selection. *Concurr. Computat.: Pract. Exper.* 27, 3 (2015), 633–650. DOI: https://doi.org/10.1002/cpe.3259

Heather J. Goldsby, Pete Sawyer, Nelly Bencomo, Betty H. C. Cheng, and Danny Hughes. 2008. Goal-based modeling of dynamically adaptive system requirements. In *Proceedings of the 15th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems (ECBS'08)*. IEEE, 36–45.

Ikbel Guidara, Nawal Guermouche, Tarak Chaari, and Mohamed Jmaiel. 2020. Time-aware selection approach for service composition based on pruning and improvement techniques. *Softw, Qual, J,* 28, 3 (2020), 1245–1277.

Sara Hassan. 2019. *Modelling and Evaluation of Microservice Granularity Adaptation Decisions*. Ph.D. Dissertation. University of Birmingham.

Vahideh Hayyolalam and Ali Asghar Pourhaji Kazem. 2018. A systematic literature review on QoS-aware service composition and selection in cloud environment. *J. Netw. Comput. Applic.* 110 (2018), 52–74.

Arshia Hosseini Bidi, Zahra Movahedi, and Zeinab Movahedi. 2021. A fog-based fault-tolerant and QoE-aware service composition in smart cities. *Trans. Emerg. Telecommun. Technol.* 32, 11 (2021), e4326.

Jun Huang, Qiang Duan, Song Guo, Yuhong Yan, and Shui Yu. 2015. Converged network-cloud service composition with end-to-end performance guarantee. *IEEE Trans. Cloud Comput.* 6, 2 (2015), 545–557.

Chandrashekar Jatoth, G. R. Gangadharan, and Rajkumar Buyya. 2015. Computational intelligence based QoS-aware web service composition: A systematic literature review. *IEEE Trans. Serv. Comput.* 10, 3 (2015), 475–492.

Amin Jula, Elankovan Sundararajan, and Zalinda Othman. 2014. Cloud computing service composition: A systematic literature review. *Expert Syst. Applic.* 41, 8 (2014), 3809–3824.

Christian Krupitzer, Felix Maximilian Roth, Sebastian VanSyckel, Gregor Schiele, and Christian Becker. 2015. A survey on engineering approaches for self-adaptive systems. *Pervas. Mob. Comput.* 17 (2015), 184–206.

Satish Kumar. 2021. *Technical Debt-aware and Evolutionary Adaptation for Service Composition in SAAS Clouds*. Ph.D. Dissertation. University of Birmingham.

Emmanuel Letier and Axel van Lamsweerde. 2002. Agent-based tactics for goal-oriented requirements elaboration. In *Proceedings of the 24th International Conference on Software Engineering*. ACM, New York, NY, 83–93. DOI: https://doi.org/10.1145/581339.581353

Shuhua Li and Minghua Chen. 2010. An adaptive-GA based QoS driven service selection for web services composition. In *Proceedings of the International Conference on Computer Application and System Modeling (ICCASM'10)*. IEEE, V13–416–V13–418. DOI: https://doi.org/10.1109/ICCASM.2010.5622827

Huagang Liang, Xiaoqian Wen, Yongkui Liu, Haifeng Zhang, Lin Zhang, and Lihui Wang. 2021. Logistics-involved QoS-aware service composition in cloud manufacturing with deep reinforcement learning. *Robot. Comput.-Integ. Manuf.* 67 (2021), 101991.

Harry Markowitz. 1952. Portfolio selection. *J. Fin.* 7, 1 (1952), 77–91. Retrieved from http://www.jstor.org/stable/2975974.

Mahboobeh Moghaddam and Joseph G. Davis. 2014. Service selection in web service composition: A comparative review of existing approaches. In *Web Services Foundations*. 321–346.

Ahmed Moustafa and Takayuki Ito. 2018. A deep reinforcement learning approach for large-scale service composition. In *Proceedings of the International Conference on Principles and Practice of Multi-agent Systems*. Springer, 296–311.

Ahmed Moustafa and Minjie Zhang. 2013. Multi-objective service composition using reinforcement learning. In *Proceedings of the International Conference on Service-oriented Computing*. Springer, 298–312.

Vivek Nallur and Rami Bahsoon. 2013. A decentralized self-adaptation mechanism for service-based applications in the cloud. *IEEE Trans. Softw. Eng.* 39, 5 (2013), 591–612. DOI: https://doi.org/10.1109/TSE.2012.53

Shunshun Peng, Hongbing Wang, and Qi Yu. 2020. Multi-clusters adaptive brain storm optimization algorithm for QoS-aware service composition. *IEEE Access* 8 (2020), 48822–48835.

M. Shepperd, C. Schofield, and B. Kitchenham. 1996. Effort estimation using analogy. In *Proceedings of the 18th International Conference on Software Engineering*. IEEE Computer Society, 170–178.

Alireza Souri, Amir Masoud Rahmani, Nima Jafari Navimipour, and Reza Rezaei. 2020. A hybrid formal verification approach for QoS-aware multi-cloud service composition. *Clust. Comput.* 23, 4 (2020), 2453–2470.

Sen Su, Fei Li, and FangChun Yang. 2008. Iterative selection algorithm for service composition in distributed environments. *Sci. China Series F: Inf. Sci.* 51, 11 (2008), 1841–1856.

Daniel Sykes, Jeff Magee, and Jeff Kramer. 2011. FlashMob: Distributed adaptive self-assembly. In *Proceedings of the 6th International Symposium on Software Engineering for Adaptive and Self-managing Systems*. Association for Computing Machinery, New York, NY, 100–109.

Asrin Vakili and Nima Jafari Navimipour. 2017. Comprehensive and systematic review of the service composition mechanisms in the cloud environments. *J. Netw. Comput. Applic.* 81 (2017), 24–36.

Axel van Lamsweerde. 2008. *Systematic Requirements Engineering: From System Goals to UML Models to Software Specifications*. John Wiley & Sons.

Hongbign Wang, Xin Chen, Qin Wu, Qi Yu, Xingguo Hu, Zibin Zheng, and Athman Bouguettaya. 2017. Integrating reinforcement learning with multi-agent techniques for adaptive service composition. *ACM Trans. Auton. Adapt. Syst.* 12, 2 (2017), 1–42.

Hongbing Wang, Mingzhu Gu, Qi Yu, Yong Tao, Jiajie Li, Huanhuan Fei, Jia Yan, Wei Zhao, and Tianjing Hong. 2019. Adaptive and large-scale service composition based on deep reinforcement learning. *Knowl.-based Syst.* 180 (2019), 75–90.

Hongbing Wang, Guicheng Huang, and Qi Yu. 2016. Automatic hierarchical reinforcement learning for efficient large-scale service composition. In *Proceedings of the IEEE International Conference on Web Services (ICWS)*. IEEE, 57–64.

Hongbing Wang, Jiajie Li, Qi Yu, Tianjing Hong, Jia Yan, and Wei Zhao. 2020. Integrating recurrent neural networks and reinforcement learning for dynamic service composition. *Fut. Gen. Comput. Syst.* 107 (2020), 551–563.

Danny Weyns. 2019. Software engineering of self-adaptive systems. In *Handbook of Software Engineering*. Springer, 399–443.

Jon Whittle, Pete Sawyer, Nelly Bencomo, Betty H. C. Cheng, and Jean-Michel Bruel. 2010. RELAX: A language to address uncertainty in self-adaptive systems requirement. *Requir. Eng.* 15, 2 (2010), 177–196.

X. Wang Z. Jing, and H. Yang. 2011. Service selection constraint model and optimization algorithm for web service composition. *Inf. Technol. J.* 10 (2011), 1024–1030. DOI: https://doi.org/10.3923/itj.2011.1024.1030

Zhuoqun Yang, Zhi Li, Zhi Jin, and Yunchuan Chen. 2014. A systematic literature review of requirements modeling and analysis for self-adaptive systems. In *Proceedings of the International Working Conference on Requirements Engineering: Foundation for Software Quality*. Springer, 55–71.

Eric S. K. Yu. 1997. Towards modelling and reasoning support for early-phase requirements engineering. In *Proceedings of the 3rd IEEE International Symposium on Requirements Engineering*. IEEE, 226–235.

Tao Yu, Yue Zhang, and Kwei-Jay Lin. 2007. Efficient algorithms for web services selection with end-to-end QoS constraints. *ACM Trans. Web* 1, 1 (May 2007), 6. DOI: https://doi.org/10.1145/1232722.1232728

Yuan Yuan, Weishi Zhang, and Xiuguo Zhang. 2018. A context-aware self-adaptation approach for web service composition. In *Proceedings of the 3rd International Conference on Information Systems Engineering (ICISE)*. IEEE, 33–38.

Received 23 August 2020; revised 31 January 2022; accepted 25 March 2022