Interaction-driven Self-Adaptation of Service Ensembles

Christoph Dorn and Schahram Dustdar

Distributed Systems Group
Vienna University of Technology
1040 Vienna, Austria
lastname@infosys.tuwien.ac.at

Abstract. The emergence of large-scale online collaboration requires current information systems to be apprehended as service ensembles comprising human and software service entities. The software services in such systems cannot adapt to user needs based on autonomous principles alone. Instead system requirements need to reflect global interaction characteristics that arise from the overall collaborative effort. Interaction monitoring and analysis, therefore, must become a central aspect of system self-adaptation. We propose to dynamically evaluate and update system requirements based on interaction characteristics. Subsequent reconfiguration and replacement of services enables the ensemble to mature in parallel with the evolution of its user community. We evaluate our approach in a case study focusing on adaptive storage services.

1 Introduction

Over the past years we have observed a trend towards online collaboration. Web sites for social networking (e.g., Facebook, LinkedIn), collaborative tagging (e.g., Digg, Del.ici.us), content sharing (e.g., Youtube), or knowledge creation (e.g., Wikipedia) have attracted millions of users. People increasingly utilize such tools to pursue joint interests and shared goals.

The scientific community in particular comes to profit from a tight interweaving of social networks and technological networks [1]. Barabasi [2] highlights the tendency for research teams to grow in size. Guimera et al. [3] describe the impact of social network dynamics on team performance. Scientific teams emerge in an ad-hoc fashion, gather the persons with the required expertise, conduct research, and dissolve again.

The scientific community is one example where collaboration emerges in large-scale, heterogeneous systems. Kleinberg [4] notices the opportunity to observe the dynamics and complexity of such systems that arise from the convergence of social and technical networks in general. We refer to such systems as *service ensembles*.

The scale of online collaborations prevents a single ensemble entity from obtaining a complete picture of the overall service ensemble. Simultaneously, services lack adequate mechanisms that derive global-level interaction characteristics. Services will exhibit poor performance and slow reaction to a changing environment: promising collaborations dissolve prematurely; helpful services remain unavailable as nobody becomes aware of the demand. As a result, enabling interaction-driven adaptivity is a prime concern in evolving information systems.

Several challenges need to be addressed before a service system can adapt to global-level ensemble interaction characteristics. Users apply both direct and indirect interaction means as well as exchange services as they feel suitable. Any interaction metric needs to abstract from these low-level interaction details and has to derive user proximity across activities, services, and shared artifacts. Any distance measurement needs to take into account also the focus and magnitude of interaction among humans, among services, and between humans and services. Existing work on social network analysis (e.g., [5, 6]) provides insight into the community structure. Deriving requirements from this data, however, is non-trivial.

In this paper we provide models and mechanisms to align configuration and provided functionality of software services with the ensemble's interaction structure. The main goal is to establish the set of required service capabilities - i.e., what functionality and adaptability services need to support - but *not* how services achieve specific adaptation. Our mechanisms, for example, discover that ensemble users tend to interact in co-located groups, and derive requirements demanding location-aware services. We do not, however, specify how these services exploit information about the ensemble interaction structure.

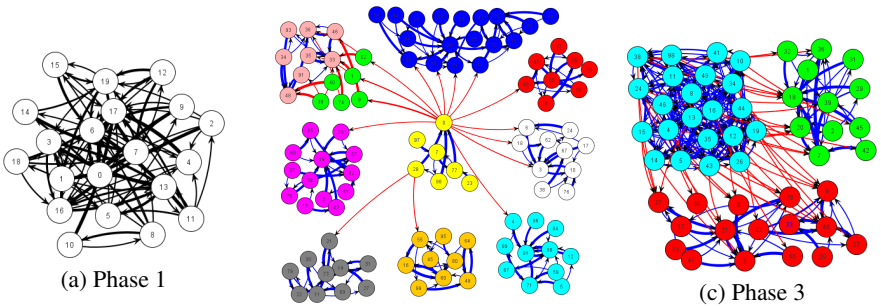
Specifically, we extend the traditional autonomic feedback loop with a secondary loop to monitor and analyze interactions across the complete service ensemble (Section 2). Interaction analysis relies on our bipartite interaction graph that tracks relations between humans and services across activities and artifacts. Distance measurements on this graph consider the interaction focus and global significance of individual ensemble entities (Section 3.2). Significant changes in the users' interaction structure trigger an update of the system's configuration via requirements rules. These rules take the interaction structure to derive necessary service capabilities (Section 3.3). Finally, we validate our approach (Section 4) based on the case study presented in the following section (Section 1.1).

1.1 Motivating Scenario

We observe a service ensemble comprising a group of 20 scientists working together closely on a research proposal. They utilize various services to coordinate their work, communicate on- and off-line, manage documents and figures, update and revise financial tables for example. In this scenario, we focus on the requirements and adaptation particular to storage services.

In such a relatively small service ensemble already, individual users find it considerably hard to gain an overview of the underlying interaction structure and the system requirements emerging from that structure. During the proposal drafting phase (Fig. 1a) users will collaborate on the various proposal sections in an ad-hoc manner, giving rise to short-lived interactions between a subset of users that dissolve again. A storage service will, therefore, need to provide maximum cooperation flexibility. A suitable service enables simple file storage and exchange between participants without requiring extensive configuration of file versioning or access control.

Let us assume, the proposal is accepted and the regular project phase commences (Fig. 1b). As the number of project participants rises, we expect stable subgroups to



(a) Phase 1 (b) Phase 2 (c) Phase 3

Fig. 1. Scenario: (Colors online) - (a) Phase 1: single cluster, dense interactions. (b) Phase 2: intra organizational interaction with ties to the coordinator. (c) Phase 3: task-centric interaction, feedback from T1 (upper left) to T2 (right) and T3 (bottom). Line thickness indicates the amount of interactions. Red links denote cross-cluster interaction, blue links represent intra-cluster interactions.

form that work on various project tasks. Subsequently, users require services that reflect this structure. A suitable service, for example, reduces information overload of individual users by focusing the interactions to users within a subgroup. Other adaptation dimensions include services keeping content within user proximity, limiting resource access to users of the same organization, or providing notifications to users with specific roles or skill.

Towards the end of the project (Fig. 1c), users have to collaborate across organizational boundaries again to integrate their research results and prototypes. Communication remains mostly within the scope of the various tasks, with users involved in demonstration activities giving feedback to users working on scientific dissemination as well as to users planning industrial exploitation. Derived requirements demand a shift from organization-centric to task-centric services.

Throughout the evolution of the ensemble, no single user obtains a complete overview of interactions and respective system requirements. Continuous interaction monitoring and analysis is therefore quintessential to enable software services to evolve in line with the overall ensemble structure.

2 From Autonomous to Evolving Service Ensembles

A service ensemble consists of humans (i.e., the users), software services, and a service infrastructure. The infrastructure typically provides a service registry and a graphical user portal. The registry contains a comprehensive set of service descriptions. At any time, however, only a subset of those services is actively deployed in the ensemble. We refer to the complete set of software elements as the *system*. Whereas an autonomous system aims to fulfill a static set of requirements, an evolving system replaces and reconfigures its parts (i.e., services) to match the dynamically changing requirements of the ensemble's user base. Here, dynamic system requirements ultimately cause the replacement, deployment, removal, or reconfiguration of services.

Most traditional autonomous systems follow a Monitor- Analyze- Plan- Execute+ Knowledge (MAPE-K) approach [7, 8] - although these phases are sometimes named differently [9, 10]. The core cycle in Fig. 2 visualizes the basic MAPE-K steps 1a, 2a, 3a, and 4. In a service ensemble, *System Monitoring* observes changes in services capabilities and QoS parameters, service dependencies, and service load. *System Analysis* compares the current requirements with the current system state. *System Planning* initiates changes when capabilities or QoS degrade, with *System Execution* enforcing the replacement and reconfiguration of the actual service instances.

We introduce a parallel (incomplete) MAPE-K cycle to turn an autonomous system into an evolving system (Fig. 2). *Ensemble Monitoring* (1b) observes the interactions and properties of the user set. *Ensemble Analysis* (2b) extracts interaction patterns and changes thereof. Given the structure of collaboration, user properties, shared artifacts, and service usage, *Ensemble Planning* (3b) updates the system requirements to match the dynamic user characteristics. Direct *Ensemble Control* of human behavior is not possible, thus the outer MAPE-K cycle remains incomplete. Any desirable impact is induced via system reconfiguration. Consequently, the system MAPE-K cycle reacts not only to system events but also to updates of the system requirements.

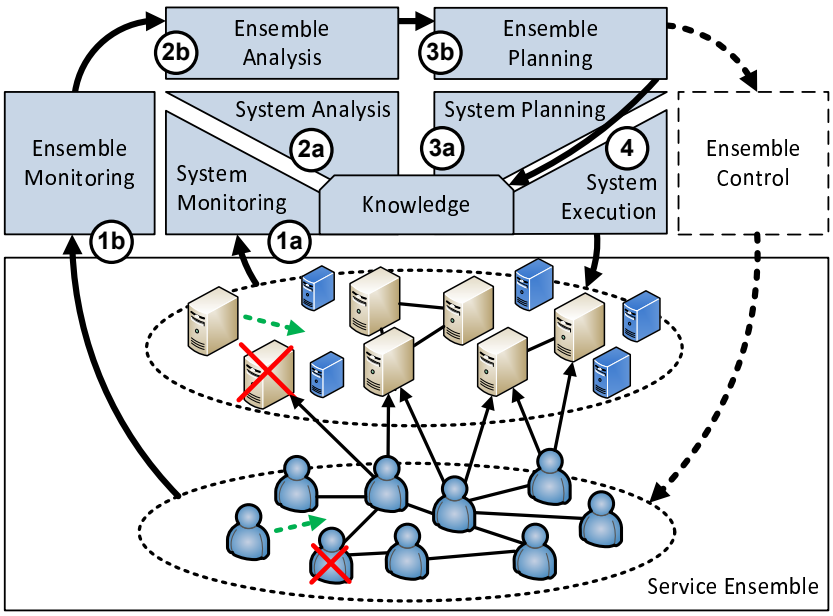


Fig. 2. Approach: from autonomous to evolving service ensembles.

Interactions play a most important part in our approach. Extraction of the ensemble structure from interactions not only reveals relevant system requirements but also provides the additional information on the system level to analyze service dependencies in more detail. In the following sections we will, therefore, focus on the interaction-centric

models and mechanisms for realizing steps 1b, 2b, 3b, and 3a. Specific adaptation and reconfiguration mechanisms in step 4 or within individual services remain outside the scope of this paper.

3 Interaction-based Continuous Adjustment

3.1 Interaction Monitoring

The key to extracting meaningful information from interaction data is monitoring the context in which the various interactions take place (Fig. 2 Step 1b). Introduced in previous work [11], the principle interaction event is an action that describes the co-occurrence of users, services, and artifacts (e.g., documents) in a common activity at the given time. In this manner, actions describe interaction among users, among services, and between users and services.

Additional information sources from the environment such as user profiles available from online social platforms or enterprise directories (in corporate settings) complement action events by putting them in perspective. The raw events are captured by distributed logging [12], monitoring [13], or sensing [14] mechanisms. Within individual services, autonomous toolkits (e.g., [15–17]) provide monitoring techniques to provide an up-to-date view on capabilities and QoS values.

3.2 Interaction Structure

We propose to integrate multiple interaction types into a single distance measurement to reflect more accurately the tight inter-dependency between humans and services (Fig. 2 Step 2b). Most existing approaches to interaction analysis observe one interaction type only, for example, either emails, friendship links, or posting threads. Furthermore, little attention is put on the focus and magnitude of links between entities, i.e. the distribution of re-occurring interactions across established links, respectively the amount of re-occurring interactions. The key aspect of our approach is weighting the entity's local interactions according to its global significance.

We map the captured actions into a bipartite interaction graph $\mathcal{AG}_2(V, E)$. Our bipartite graph defines two vertex categories: actions ($\mathcal{V}_a$) and ensemble entities ($\mathcal{V}_e$). Edges are undirected and exist only between vertices of different category. Fig. 3a displays an example bipartite interaction graph comprising actions (a1, a2, a3, a4) and ensemble entities of type user (u1, u2, u3, u4), activity (t1, t2), artifact (o1, o2), and service (s1, s2, s3). An action's weight $w(a)$ corresponds to the number of times that action has occurred. Two actions are considered identical when they exhibit the same set of involved ensemble entities.

We measure the distance between two vertices of the same ensemble entity type by aggregating their involvement in shared *and related* actions. The distance, for example, between two users is based on actions involving joint activities, joint resources, and joint artifacts. In the example graph (Fig. 3a) users u2 and u3 become linked to user u4 through the shared artifacts o1 and o2. The process of calculating the distance between two entities requires determining the similarity between any two actions first.

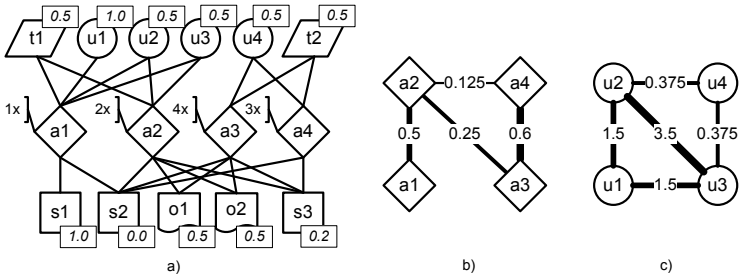


Fig. 3. Bipartite interaction graph (a) combining user, activities, artifacts, service, and actions; action similarity graph (b); user distance graph (c).

Action similarity For each pair of actions (a_i, a_j), we consider the amount of shared entities. The set of common entities, however, is insufficient to accurately establish the distance between two actions. We also consider the global significance of a shared entity in contributing to the overall distance measurement. For example, service $s2$ has edges to all actions. It should not be considered as it does not add any information to distinguish the similarity of two actions.

We apply Shannon's entropy definition [18] $H(w) = -\sum(w * log(w))$ to describe the information content of an entity's edge set, thus deriving the global significance $sig(v)$. The significance of entity v is defined as[1]:

$$sig_v = 1 - \frac{log(deg(v))}{log(|a|)} \qquad \forall \qquad v \in \mathcal{V}_e \; and \; deg(v) > 1 \tag{1}$$

where $deg(v)$ is the degree of vertex v and $|a|$ denotes the total number of actions. The normalization yields a significance value in the interval $[0, 1]$. When a vertex v links to all actions, it exhibits no focus (i.e., minimum entropy) and thus significance becomes 0. Vertices with one neighbor yield $sig(v) = 1$. Italic numbers in Fig. 3a provide the global significance values for the example entities.

The Jaccard similarity metric determines the similarity of two actions based on their common entities. We consider, however, only those entities which exceed a given significance threshold γ. The Jaccard similarity is defined as 1 minus the difference between set union and set intersection, divided by the set union:

$$J_\delta = 1 - \frac{|A \cup B| - |A \cap B|}{|A \cup B|} \tag{2}$$

where sets A and B contain the references to users, services, activities, and artifacts when comparing two actions a_i and a_i. J_δ becomes 1 when the two sets contain the same entities and yields 0 when the two sets are completely disjoint. The complete action similarity graph is denoted $\mathcal{AG}_{action}(\mathcal{V}_a, E_a)$ shown in Fig. 3b for the example bipartite graph with $\gamma = 0.3$.

[1] Reduced from $sig_v = 1 - \frac{-deg(v) * (\frac{1}{deg(v)} * log(\frac{1}{deg(v)}))}{log(|a|)}$

Interaction distance The distance measurement between two ensemble entities of same type takes two aspects into account:

- The distance between two entities decreases with increasing number of shared actions. A large amount of shared actions, however, does not necessarily imply low distance.
- The distance between two entities decreases with increasing re-occurrence of interactions. The more often two entities have interacted (i.e., high action weight) the closer they become. Two entities having interacted only once in the scope of three actions, yield higher distance than two entities having interacted 10 times via a single action. Users u2 and u3, for example, yield lower proximity than u2 and u1.

In the case of user distance, we take measurements across services, activities, and artifacts to derive interaction distance for direct and indirect communication. One user, for example, uploads a document, while another user downloads it. This process consists of two actions, each involving a single user, and the same document. Total distance $dist_{total}(u_i, u_j)$ between two users derives, thus, not only from shared actions ($dist_1(u_i, u_j)$), but also related actions ($dist_2(u_i, u_j)$):

$$dist_{total}(u_i, p_j) = dist_1(u_i, u_j) + dist_2(u_i, u_j) \tag{3}$$

We define two action sets. Set A_i contains all neighboring actions of u_i. There exists an analogous set A_j. The shared action distance is defined as the sum of actions weights w:

$$dist_1(u_i, u_j) = \sum_{n}^{|N|} w(n) \tag{4}$$

where $N(u_i, u_j)$ is the interaction of A_i and A_j.

The related actions distance $dist_2(u_i, u_j)$ defines the edge set $E_a(a_k, a_l)$ in the action similarity graph $\mathcal{AG}_{action}$ such that $a_k \in A_i$, $a_l \in A_j$, and $a_k \neq a_l$.

$$dist_2(u_i, u_j) = \frac{\sum_E (w_e * min[w(a_k), w(a_l)])}{|E|} \tag{5}$$

The set of pairwise distance measurements generates the entity interaction distance graph $\mathcal{AG}_{user}(\mathcal{V}_P, E_P)$ — here specific to users — required in the subsequent analysis step (Section 3.3). The corresponding graph for the four users in the example is displayed in Fig. 3c.

3.3 Interaction-driven Requirements Adjustment

System requirements need to reflect the significant changes in the ensemble's interaction structure (Fig. 2 Step 3b). One important structural property of service ensembles is the number and the size of subgroups that emerge from the interaction distance graph and what factors cause this structure. Graph analysis based on interaction affinities (e.g., [19]) or community detection algorithms (e.g., [20] or [21]) describes the underlying structure from which we subsequently derive appropriate system requirements. Fig. 4

displays the UML representation of the interaction analysis data model. The model lists the set of involved entities of type users, activities, artifacts, and services. It states for each entity the centrality in the overall interaction graph and the global significance value applied during distance calculation. Each *ClusterAnalysis* element describes the observed entity type and the context property that most likely caused the clusters to form (e.g., location, organization, task, or role). Each detected *Cluster* states the interaction-centric metrics such as density, average cardinality, interaction direction (interaction within the cluster $[0 \rightarrow 1]$ or external $[0 \rightarrow -1]$), and cluster name (e.g., the organization's name, particular location, or role identifier). The membership degree specifies the degree to which an entity belongs to a particular cluster, and its centrality within that cluster.

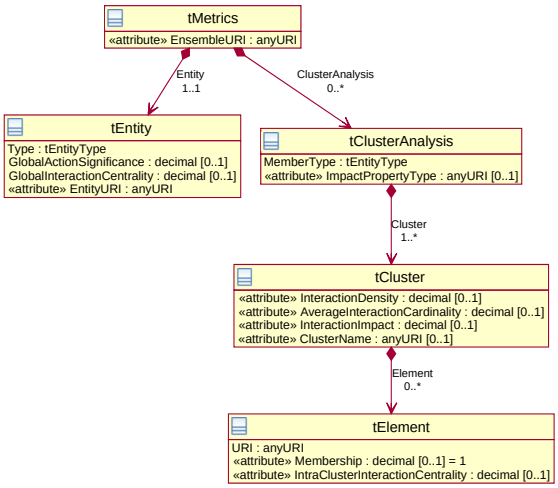


Fig. 4. Excerpt of the ensemble model specifying the interaction analysis result.

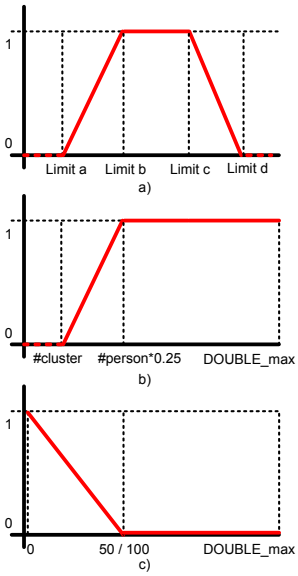


Fig. 5. Utility function: (a) generic form, (b) fct. utilized in the example rule, and (c) fct. used in the evaluation.

Our previous work [22] introduced a capability model for services. The capability model specifies non-functional properties such as storage space per user, cost, or a service's adaptability of resource access to location or organization context. System requirements are expressed in terms of necessary capabilities. Coupling these system requirements with rules enables us to dynamically adjust the system configuration to match the interaction structure of the service ensemble.

The example requirement rule in Listing 1.1 demands common storage space for each tightly connected subgroup (i.e., a cluster with interaction impact between 0 and 1). A requirements rule consists of four parts:

```
rule "StorageClusterSupport"
dialect "java"
when
    em : EnsembleMetrics ((UserCA.clusters.size() >= 2) &&
                          (UserCA.clusters.size() <
                              (0.25*em.getEntities().getUsers().size()))
then
    int clusterCount = UserCA.clusters.size();
    int maxClusterSize = 0;
    for (int i = 0; i<clusterCount; i++)
    {
        int size = UserCA.cluster.get(i).elements.size()
        if (size > maxClusterSize)                  maxClusterSize = size;
        if (UserCA.cluster.get(i).getImpact() < 0)  clusterCount--;
    }
    URI impactTypeURI = UserCA.getImpactPropertyType();
    TCapabilitySelectionRequirement req1 = RequirementsFactory.getSelectConstraint(
        "Require Adaptability to Interaction Cluster Impact",
        URIs.SELECTCAP_ADAPTABILITY,
        new String[]{impactTypeURI},
        ChoiceUtilityHigh.class.getSimpleName(), ChoiceUtilityHigh.UTILITY_TYPE
        0.8d);
    TSimpleDecimalConstraint req2 = RequirementsFactory.getConstraint(
        "Support folders for all clusters",
        URIs.CAP_ResStorage, URIs.PROP_MaxFoldersPerAccount_ResStorage,
        ValueUtility.UTILITY_TYPE, ValueUtility.class.getSimpleName()
        new double[]{  clusterCount, 0.25*em.getEntities().getUsers().size(),
                       Double.MAX_VALUE, Double.MAX_VALUE},
        0.5d);
    TSimpleDecimalConstraint req3 = RequirementsFactory.getConstraint(
        "Support sufficient concurrent users within folders",
        URIs.CAP_ResStorage, URIs.PROP_MaxConcurrentUsersPerFolder_ResStorage,
        ValueUtility.UTILITY_TYPE, ValueUtility.class.getSimpleName()
        new double[]{  maxClusterSize*0.9, maxClusterSize*2,
                       Double.MAX_VALUE, Double.MAX_VALUE},
        0.4d);
    knowledge.addRequirement( em.getEnsembleURI(), req1);
    knowledge.addRequirement( em.getEnsembleURI(), req2);
    knowledge.addRequirement( em.getEnsembleURI(), req3);
end
```

Listing 1.1. Example DROOLS requirement rule generating three storage capability constraints when interaction clusters have emerged.

1. the interaction-based condition triggering the rule. In the example we wait for two or more clusters of users. Simultaneously, we limit that number to a quarter of the total number of users to mitigate potential errors during the clustering process.
2. the computation of utility function parameters (optional).
3. the generation of the desired capability constraints. A constraint specifies the desired capability, the utility function type, utility function parameters, and the importance of the constraint. Fig. 5 visualizes the general utility function (a), and the instance (b) utilized in requirement *req*2. In this example, we require a storage service to be aware of the source causing the clustering (e.g., clusters mapping to organization, locations, or roles). We also demand that the service supports as many folders as there are clusters, and that each folder can be accessed simultaneously by at least as many users as contained in the largest cluster.
4. the update of the knowledge component of the system's MAPE-K cycle.

3.4 System Planning and Execution

The new capability constraints need to be matched against the current system configuration (Fig. 2 Step 3a). Along with the capability model, we have introduced a constraint matching and selection algorithm in [22].

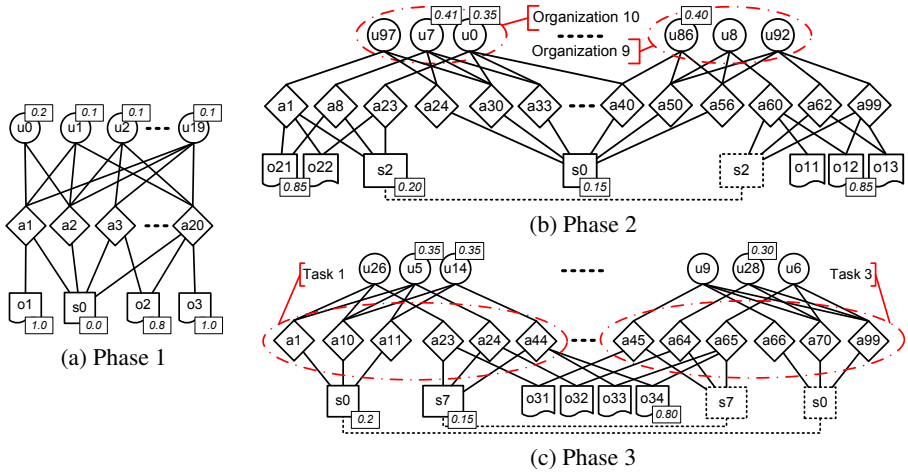
The ultimate execution of adaptation plan involves removal, replacement, or deployment of new services. Dynamic invocation frameworks such as VRESCO [23] provide the appropriate mechanisms to exchange service endpoints during runtime. We rely on VRESCO's runtime SOAP message interceptors to perform necessary message transformation operations. This addresses potential service interoperability problems and allows interaction-based requirements analysis to work on a common capability model without having to take heterogeneous service interfaces into account.

4 Case study Evaluation

We base our evaluation on the three phases of the motivating scenario in Section 1.1. Throughout the evaluation we focus on a limited number of requirements and service specific storage capabilities for sake of clarity. Table 1 (upper part) lists the monitored capabilities and their support by ten example services (S1 ... S10). The underlying interaction data (Fig. 1a-c) for the three phases are synthetic, but yield the same global interaction characteristics we observed in our past research projects. Excerpts from the corresponding bipartite action graphs are visualized in Fig. 6a-c. Table 7 provides the results of the cluster analysis subsequently applied to the rules in Listing 1.1. We also apply an additional cost constraint. The respective cost utility function yields maximum utility at \$0 , and minimum utility from \$50 (Phase 1), respectively \$100 (Phase 2 and 3), onwards. Table 1 (lower part) lists the final service ranking result, while the right part supplies the parameters for the respective utility functions. The ranking process utilizes the Logic Scoring of Preferences (LSP) algorithm [24] - a weighted sum of utility scores.

Phase 1: During the project proposal drafting the 20 initial participants interact closely without being affected by location, their role, or organizational boundaries (Fig. 1a). The bipartite action graph excerpt for phase 1 (Fig. 6a) has most users involved in every action using an email service ($sig(S0) = 0$) . S0 is thus not considered for action similarity measurements. The corresponding interaction cluster analysis for users does not reveal emergence of distinct subgroups, thus the only *task* of preparing the proposal becomes the best explanation for the dense interaction graph (Fig. 7). As outlined in the motivation, services which are cheap and offer the minimum required amount of support for users and folders rank highest. Here, we select service S2.

Phase 2: The participant number raises sharply shortly after the project kickoff. In our scenario, ten organizations allocate between 6 and 18 members for a total number of 100 participants. Project coordinator is Organization 10, exhibiting a neutral impact due to a balance of organization internal and external interaction. Most members keep interaction organizations internal, with exception to Organization 2, who's members focus purely on coordinating members in Organization 8 (Fig. 1b). The underlying bipartite action graph (Fig. 6b) emphasizes the uptake of service S2 in Phase 1 as an indirect



(a) Phase 1

(b) Phase 2

(c) Phase 3

Fig. 6. Bipartite action graph excerpt - activities omitted and services s2 and s7 split for the sake of clarity. Significance values are given only for a subset of entities and are based on the complete bipartite graph.

Phase	Property	ClusterName	Impact	Size
1	Task	Proposal	1	20
2	Org.	Org. 1	1	9
		Org. 2	-1	6
		Org. 3	1	10
		Org. 4	1	10
		Org. 5	1	13
		Org. 6	1	18
		Org. 7	1	11
		Org. 8	1	8
		Org. 9	1	8
		Org. 10	0	7
3	Task	Dem. (T1)	0.83	22
		Dissem. (T2)	1	12
		Exploit. (T3)	1	16

Fig. 7. Interaction Cluster Analysis.

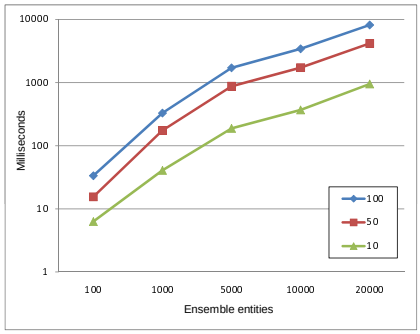


Fig. 8. Interaction analysis performance measurements (in milliseconds) for service ensemble graphs exhibiting 100 to 20000 entities, featuring 10, 50, and 100 properties.

interaction tool via shared artifacts. Simultaneously, users continue to utilize the email service (S0), albeit, mostly for organization internal communication. Both services thus provide useful (but low significance) information for the calculation of action similarity. The subsequent cluster analysis reveals a strong impact of the organizational topology on the interaction structure (Fig. 7). The same requirement rule of Phase 1 will now request services exhibiting adaptability to the organizational structure, while supporting the required number of users within each cluster. In our scenario, we consequently switch from service S2 to S7.

Phase 3: The number of project participants decreases to 50 towards the end of the project (Fig. 1c). The bipartite action graph (Fig. 6c) draws attention to the task internal direct communication, and cross-task indirect interaction via shared artifacts. We also note the switch of storage services as recommended in the previous phase (S7 instead of S2). Interaction analysis now identifies the task structure as the most likely factor to give rise to clusters (Fig. 7). Derived requirements demand a change from organization-centric to task-centric services. Subsequently, we replace service S7 with service S5.

	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10	Para 1	Para 2	Para 3	w
Org-adaptability	-	-	x	-	-	-	x	-	x	x	-	x	-	0.8
Loc-adaptability	-	-	-	x	-	-	x	-	-	x	-	-	-	0.8
Task-adaptability	-	-	-	-	x	-	-	x	x	x	-	-	x	0.8
Role-adaptability	-	-	-	-	-	x	-	x	x	x	-	-	-	0.8
FoldersPerAccount	5	5	20	50	50	100	50	250	100	250	1;5	9;25	3;12.5	0.5
UsersPerFolder	10	50	100	20	50	10	10	50	100	100	18;40	16.2;36	19.8;44	0.4
Costs $	0	0	60	50	20	100	50	30	90	200	0;50	0;100	0;100	0.5
Rank Phase 1	56	**100**	44	3	**78**	0	0	36	44	44				
Rank Phase 2	0	25	**98**	49	34	41	**100**	46	**94**	76				
Rank Phase 3	0	19	42	44	**100**	10	18	**98**	**89**	71				

Table 1. Storage service capabilities, utility function parameters, constraint weight, and ranking results (top services in bold font).

Discussion During the scenario users switch between different communication types (direct such as email, and indirect such as shared artifacts). They also exchange services. Nevertheless, we are still able to track those interactions and thus derive accurate user proximity as we map all actions onto our bipartite graph. The significance of particular entities changes throughout the scenario. Dynamically assessing their impact on similarity guarantees accurate distance measurements during all scenario phases.

A global-level view on interactions is vital to establish the underlying interaction structure such as emerging clusters. The interaction analysis detects the cause of clustering, such as organizational, spatial, or task-related constraints. The respective requirements demand exactly those services that are able to adapt to and support these constraints. Observation of individual user or service properties alone can never reveal such requirements.

The performance evaluation of the interaction analysis algorithm for various configurations of service ensemble size and property count is displayed in Figure 8. The algorithm exhibits exponential runtime behavior as expected for a growing interaction network. However, even for the most complex scenario of 20000 entities each exhibiting 100 properties, the interaction impact is derived within 10 seconds which is sufficiently fast.

5 Related Work

Interaction-based adjustment of service systems builds on concepts and techniques from the autonomic computing domain and service adaptation community. Current general-

purpose autonomic techniques and toolkits (e.g., [15–17]) primarily apply context about the software environment. These frameworks adhere to the basic MAPE-K feedback loop, limiting the application of user context to properties such as location or device.

Jennings et al. [25] propose an architecture for autonomic management of communication networks that comes close to our approach of monitoring a large-scale system. No human interaction characteristics, however, are considered.

Colman [26] proposes a hybrid approach to self-organization services through hierarchical structuring of autonomic managers and services. An autonomic manager monitors and controls all composed services, but lacks insight into the interaction characteristic of the environment. In our work, adaptation to the environment is sensed and analyzed in parallel to regular system monitoring. Any effect on the environment, however, can only be achieved through the system MAPE-K cycle.

The main shortcoming of the presented autonomic computing approaches is their fixed set of requirements. These remain valid throughout the system's lifetime or require manual configuration. In large-scale ensembles neither the services' providers nor individual users can grasp the system functionality required by the overall ensemble.

Specifically in service-oriented systems, extensive research efforts focus on service selection based on Quality-of-Service (QoS) attributes (e.g., [27–29]), context information (e.g.,[30, 31]), or trust (e.g., [32–34]). Recently, Skopik et al. [35] extended the notion of trust to cover both humans and services in mixed service-oriented systems.

These approaches consider only a subset of an ensemble context during composition. Adaptation and selection criteria usually build upon QoS metrics or context about the service execution environment. Involved user context comprises mostly location, devices, and preferences. None of these efforts evaluates the complex user interaction structure within the service ensemble.

Social network analysis investigates the interaction characteristics of online communities. Information (e.g., [5, 6]) that potentially serves as context for adaptation is usually not available in near-realtime, nor does it include aspects beyond human-to-human communication. Our approach derives similarity from users' involvement in joint activities, use of common services, or modification of shared artifacts. Additionally, we take into consideration the global significance of individual elements when computing distance measurements.

6 Conclusion and Outlook

We have demonstrated the importance of monitoring global-level user interaction characteristics to adapt system requirements. Based on captured user actions, our bipartite interaction graph enables proximity measurements across users, activities, resources, and artifacts. Subsequent interaction analysis identifies those properties that determine the emergence of clusters. Clustering results and other interaction metrics provide meaningful data for interaction-centric requirements rules. These rules generate service capability constraints which serve as system requirements. Final matching against service capability profiles provides recommendations for system reconfiguration.

The current approach evaluates the strongest interaction impact for the complete service ensemble. We plan to introduce context-dependent analysis that observes dif-

ferent parts of the ensemble independently to simultaneously adapt to multiple impact factors. The second line of future research activities focuses on deriving and describing reoccurring global-level interaction patterns. So far changes in the interaction structure cannot be anticipated. We expect that prediction techniques combined with patterns enable the early detection of structural changes and facilitate such transitions through a-priori service reconfiguration.

Acknowledgment

This work has been partially supported by the EU STREP project Commius (FP7-213876).

References

1. Jones, B.F., Wuchty, S., Uzzi, B.: Multi-University Research Teams: Shifting Impact, Geography, and Stratification in Science. Science **322**(5905) (2008) 1259–1262
2. Barabasi, A.L.: SOCIOLOGY: Network Theory-the Emergence of the Creative Enterprise. Science **308**(5722) (2005) 639–641
3. Guimera, R., Uzzi, B., Spiro, J., Amaral, L.A.N.: Team Assembly Mechanisms Determine Collaboration Network Structure and Team Performance. Science **308**(5722) (2005) 697–702
4. Kleinberg, J.: The convergence of social and technological networks. Commun. ACM **51**(11) (2008) 66–72
5. Bird, C., Gourley, A., Devanbu, P., Gertz, M., Swaminathan, A.: Mining email social networks. In: MSR '06: Proceedings of the 2006 international workshop on Mining software repositories, New York, NY, USA, ACM Press (2006) 137–143
6. Valverde, S., Solé, R.V.: Self-organization and hierarchy in open source social networks. Technical report, DELIS – Dynamically Evolving, Large-Scale Information Systems (2006)
7. Kephart, J.O., Chess, D.M.: The vision of autonomic computing. Computer **36**(1) (January 2003) 41–50
8. IBM: An architectural blueprint for autonomic computing (2005)
9. Parashar, M., Hariri, S.: Autonomic computing: An overview. In: UPP. (2004) 257–269
10. Dobson, S., Denazis, S., Fernández, A., Gaïti, D., Gelenbe, E., Massacci, F., Nixon, P., Saffre, F., Schmidt, N., Zambonelli, F.: A survey of autonomic communications. ACM Trans. Auton. Adapt. Syst. **1**(2) (2006) 223–259
11. Schall, D., Dorn, C., Dustdar, S.: Viecar - enabling self-adaptive collaboration services. In: 34th EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA), IEEE Computer Society (September 2008)
12. Dorn, C., Truong, H.L., Dustdar, S.: Measuring and analyzing emerging properties for autonomic collaboration service adaptation. In: ATC '08: Proceedings of the 5th international conference on Autonomic and Trusted Computing, Berlin, Heidelberg, Springer-Verlag (2008) 162–176
13. Moser, O., Rosenberg, F., Dustdar, S.: Non-intrusive monitoring and service adaptation for ws-bpel. In: WWW '08: Proceeding of the 17th international conference on World Wide Web, New York, NY, USA, ACM (2008) 815–824
14. Akyildiz, I., Su, W., Sankarasubramaniam, Y., Cayirci, E.: A survey on sensor networks. Communications Magazine, IEEE **40**(8) (Aug 2002) 102–114

15. Sterritt, R., Smyth, B., Bradley, M.: Pact: personal autonomic computing tools. In: EASe Workshop at ECBS 2005. (2005) 519–527
16. Bigus, J.P., Schlosnagle, D.A., Pilgrim, J.R., Mills, W.N., Diao, Y.: Able: A toolkit for building multiagent autonomic systems. IBM Systems Journal **41**(3) (2002)
17. IBM: Autonomic computing toolkit: Developer's guide. http://www-128.ibm.com/developerworks/autonomic/books/fpy0mst.htm (2004)
18. Shannon, C.E.: A mathematical theory of communication. Bell system technical journal **27** (1948)
19. Dorn, C., Schall, D., Dustdar, S.: A model and algorithm for self-adaptation in service-oriented systems. In: IEEE European Conference on Web Services (ECOWS). (November 2009)
20. Capocci, A., Servedio, V., Caldarelli, G., Colaiori, F.: Detecting communities in large networks. Physica A: Statistical Mechanics and its Applications **352**(2-4) (2005) 669 – 676
21. Newman, M.E.J.: Modularity and community structure in networks. PROC.NATL.ACAD.SCI.USA **103** (2006) 8577
22. Dorn, C., Schall, D., Dustdar, S.: Context-aware adaptive service mashups. (December 2009) IEEE Asia-Pacific Services Computing Conference (APSCC) - (short paper).
23. Michlmayr, A., Rosenberg, F., Platzer, C., Treiber, M., Dustdar, S.: Towards recovering the broken soa triangle - a software engineering perspective. In: 2nd International Workshop on Service-oriented Software Engineering (IW-SOSWE'07 @ ESEC/FSE'07). (September 2007)
24. Dujmovic, J.J.: Continuous preference logic for system evaluation. In: IEEE Transactions on Fuzzy Systems. Volume 15., IEEE Computer Society (2007) 1082–1099
25. Jennings, B., van der Meer, S., Balasubramaniam, S., Botvich, D., Foghlu, M., Donnelly, W., Strassner, J.: Towards autonomic management of communications networks. Communications Magazine, IEEE **45**(10) (October 2007) 112–121
26. Colman, A.: Exogeneous management in autonomic service compositions. In: ICAS '07: Proceedings of the Third International Conference on Autonomic and Autonomous Systems, Washington, DC, USA, IEEE Computer Society (2007) 25
27. Yu, T., Lin, K.J.: Adaptive algorithms for finding replacement services in autonomic distributed business processes. In: Autonomous Decentralized Systems, 2005. ISADS 2005. Proceedings. (April 2005) 427–434
28. Wang, X., Vitvar, T., Kerrigan, M., Toma, I.: A qos-aware selection model for semantic web services. In: ICSOC. (2006) 390–401
29. Rosenberg, F., Leitner, P., Michlmayr, A., Celikovic, P., Dustdar, S.: Towards composition as a service - a quality of service driven approach. (29 2009-April 2 2009) 1733–1740
30. Yang, Y., Mahon, F., Williams, M.H., Pfeifer, T.: Context-aware dynamic personalised service re-composition in a pervasive service environment. In: UIC. (2006) 724–735
31. Baresi, L., Bianchini, D., Antonellis, V.D., Fugini, M.G., Pernici, B., Plebani, P.: Context-aware composition of e-services. In: TES. (September 2003) 28–41
32. Vu, L.H., Hauswirth, M., Aberer, K.: Qos-based service selection and ranking with trust and reputation management. In: OTM Conferences (1). (2005) 466–483
33. Maximilien, E.M., Singh, M.P.: Toward autonomic web services trust and selection. In: ICSOC '04: Proceedings of the 2nd international conference on Service oriented computing, New York, NY, USA, ACM (2004) 212–221
34. Maximilien, E., Singh, M.: Self-adjusting trust and selection for web services. (June 2005) 385–386
35. Skopik, F., Schall, D., Dustdar, S.: The cycle of trust in mixed service-oriented systems. In: 35th Euromicro Conference on Software Engineering and Advanced Applications (SEAA). (August 2009)