VieCAR - Enabling Self-adaptive Collaboration Services [*]

Daniel Schall, Christoph Dorn, Schahram Dustdar
Distributed Systems Group
Vienna University of Technology
1040 Vienna, Austria
schall|dorn|dustdar@infosys.tuwien.ac.at

Ignazio Dadduzio
Dipartimento di Elettronica e Informazione
Politecnico di Milano
20133 Milano, Italia
ignazio.dadduzio@gmail.com

Abstract

Novel forms of collaboration increasingly distribute control among e-workers, thereby allowing agile and autonomous collaboration. However, this requires a novel blend of infrastructure and algorithms for self-adaptation of collaboration services. We present VieCAR (Vienna Collaborative Activity and Resource Management Framework), a framework that addresses the requirements of new collaborative service-oriented environments. Self-adaptive *collaboration services depend on the user's context. VieCAR combines service-oriented architectures with* activity-centric computing *enabling people to interact and collaborate regardless of their location and across organizational boundaries. Based on VieCAR's activity model, we present a ranking algorithm determining the relevant input for service adaptation.*

1 Introduction

In rapidly changing environments e-professionals need to collaborate regardless of location and across organizational boundaries. Traditional coordination styles following the "command and control" pattern are no longer efficient nor effective. Novel forms of collaboration increasingly distribute control among e-workers, thereby allowing agile and autonomous collaboration. However, this requires a novel blend of collaboration infrastructure. Web services are suitable as the basis for providing the needed resources and tools to conduct collaboration in a heterogeneous and distributed environment. Web services allow collaboration and interactions to span multiple organizations.

However, this novel way of collaboration and interaction requires adaptive collaboration services that adapt to the relevant view of individual e-workers. In this paper we present an activity-centric approach to collaboration and a framework that provides the fundamental infrastructure for self-managed collaboration services.

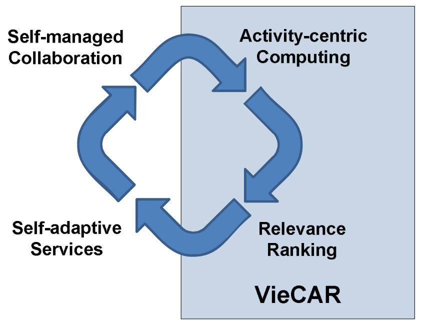


Figure 1. Adaptive collaboration and services.

Figure 1 shows the motivation of our approach. Ad-hoc collaborations resulting in complex interactions between e-workers and used resources, demand distributed control, thereby leading to **self managed collaboration**. This complex environment requires a set of **self adaptive services** providing the means to *coordinate*, *collaborate*, and *communicate*. Self-adaptation of services is based on the collaboration context of the users. We define context as the set of information (being used in the collaboration) that is relevant to the users' activities. Activities are the glue that bind together collaborative work. A technique to establish the context is **relevance ranking** based on activities. **Activity-centric computing** allows users to organize their work in a flexible yet structured manner.

Consider the following scenario: Mr. Marc Hash is an expert in the field of distributed systems and thus involved in over ten simultaneous consulting and development projects. Shortly before an important deadline, Mr. Hash needs to organize an ad-hoc meeting to sort out urgent problems. The challenge he faces is to find all relevant documents for preparation. For collaboration services sup-

[*]This work has been partially supported by the EU STREP project inContext (FP6-034718).

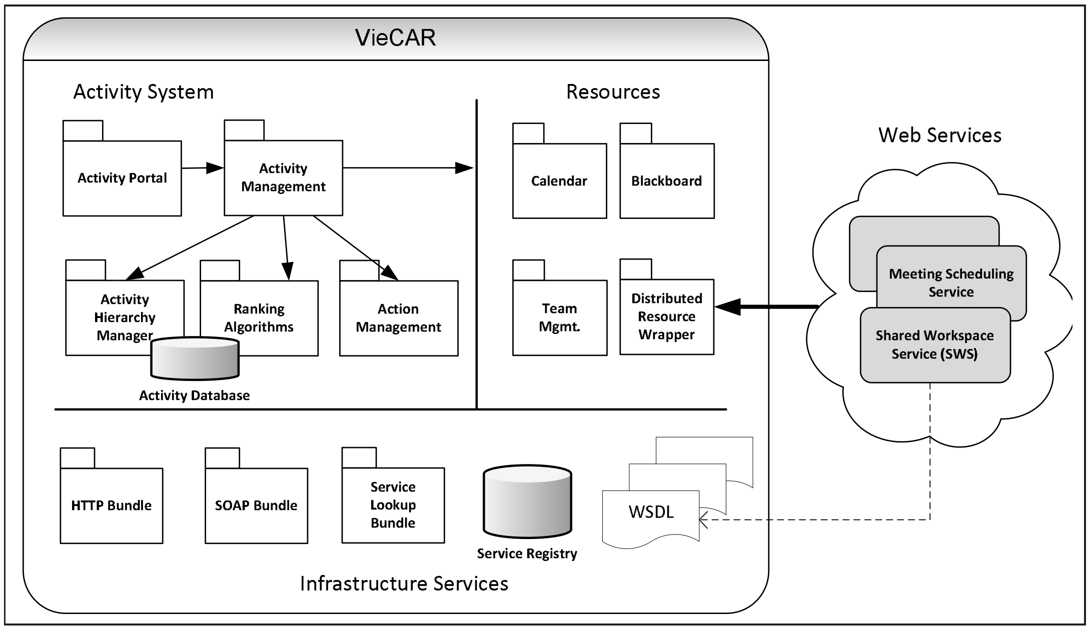


Figure 2. Middleware platform and architecture.

porting Mr. Hash, the problem is selecting the right places to search for such documents. This paper presents our contribution to tackle such issues.

Approach: Our approach is the design of VieCAR to manage collaborative activities and resources in distributed environments. Specifically, we have performed the following steps to enable self-adaptive collaboration:

- Design of an *activity model* allowing human actors to organize their work and related resources. The execution of activities may also encompass the usage of Web services. Thus, in our model, Web services are seen as resources in collaborations. Our activity model allows people to organize a set of activities hierarchically. Therefore, activities can be assembled to represent more complex work structures.

- Design and implementation and evaluation of the VieCAR *middleware platform*. VieCAR follows the service-oriented paradigm, thereby allowing services and resources to be discovered at run-time. To enable cross-organizational collaborations and to support a large number of teams, resources can be distributed across multiple platforms.

- The design and evaluation of an algorithm that enables *context-aware ranking* of resources based on the user's context. Our ranking approach aims at extracting emerging relations between members, activities, artifacts, and resources by taking multiple criteria into account.

Structure of the Paper: Section 2 introduces the VieCAR framework and middleware, a platform for managing collaborative activities and resources in distributed environments. In Section 3, the activity model is defined that captures the dynamic nature of ad-hoc collaborations. VieCAR's ranking approach to determine the most relevant context is presented in Section 4, followed by a discussion and performance study of the ranking algorithm in Section 5. Section 6 discusses related work. Finally, conclusion and future work are presented in Section 7.

2 VieCAR Framework

The VieCAR framework is shown in Figure 2 and consists of three main elements: *Activity System*, *Resources*, and *Infrastructure Services*.

Infrastructure Services: The basic infrastructure services are provided and implemented in an OSGi container environment[1]. OSGi is a lightweight platform (container) providing a set of management services (i.e., OSGi bundles) to realize service-oriented applications. Each service in a container can be discovered and used by other services, given an interface is provided and registered in the *Service Registry*. The *HTTP* and *SOAP* bundle allow local services to export features as Web services. For example, the *Activity System* uses the *HTTP* bundle to make the activity management features available to users via the *Activity Portal*. The *Service Registry* holds information regarding the location of

[1]`http://www.osgi.org/osgi_technology/`

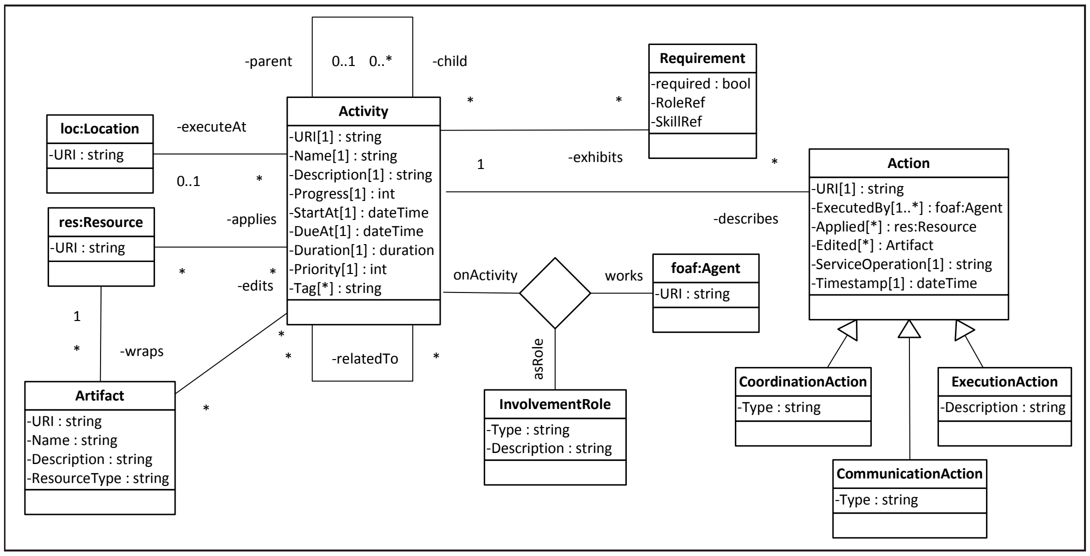


Figure 3. Activity and action model.

services and allows local and distributed services to be discovered (*Service Lookup*).

Activity System: Comprises the *Activity Management*, which is responsible for basic operations such as creating, updating, or modifying activities. Additionally, the *Activity Hierarchy Manager* allows hierarchies of activities (*compositions*) to be created. Users can assemble a set of activities to a new hierarchy of activities or modify existing hierarchies. In our activity-based collaboration system, activity hierarchies are modeled as *parent-child* relations between activities . Furthermore, *Ranking Algorithms* are used to determine the importance of activities. Finally, activities are persistently saved in a database (see Section 5 for detailed implementation aspects).

Users can access the *Activity System* using a Web portal denotated as *Activity Portal* in Figure 2. Since VieCAR is built upon a service-oriented architecture using Web services, *Activity Management* can be accessed by using a Web service interface.

Resources: Figure 2 shows a number of collaboration resources such as locally deployed services including Calendar, Blackboard, etc. or distributed services that are made available within VieCAR by using Web services and the SOAP protocol to interact with remote resources.

3 Activity-based Collaboration

Activities are the glue that bind together collaborative work. Examples of collaborative activities at various levels of granularity are *sending an email*, *reviewing a paper*, *organizing a workshop*, and *managing a multi-national research project*. A single activity by itself provides basic collaborative data. It describes the work to be done, defines the temporal constraints, and lists the involved people. That data is often sufficient in static collaborative settings where all members are aware of the overall working environment. In dynamic and distributed collaborative environments, however, we have to explicitly model the embedding of a single activity in the overall collaboration context. The context contains the overall structure of activities, dependencies between activities, the temporal flow of (future) activities, and history of activity changes. This provides the core structure of collaborative work. In addition, the collaboration context describes the involvement of members, their roles, required and applied skills, work artifacts, and resources. Consequently, an expressive activity model needs to support such relations.

Existing activity-based tools provide very limited means to connect activities, members, and resources. Therefore, the design of the activity information model focuses extending the existing activity-like data structures such as iCal RFC 2445[2]. Figure 3 presents a UML model of the core activity structure.

We focus on elaborating in detail the relations that link a single activity into the overall collaboratin context. Activity properties such as *Name*, *Description*, or *StartAt* are inspired by RFC 2445 and not discussed in detail here.

The primary activity structure is a hierarchy. Based on parent-child relations, child activities specify in more detail the required work to fulfill the goal of the parent activity. Depending on the purpose, such a hierarchy can reflect project topology, activity types, or any other structure suitable to group related activities. In addition, the generic

[2]`http://www.ietf.org/rfc/rfc2445.txt`

relatedTo property provides a flexible mechanism to link to any other activity in the hierarchy. Typically, multiple members work on the same activity having different roles. An *InvolvementRole* identifies the creator, observer, contributor, responsible, and supervisor of an activity. *Requirements* define skill-based and role-based restrictions on the members working on an activity. Involved workers apply a set of *Resources* to execute their work.

Resources can be generic, e.g., describing a virtual blackboard, but also specific such as documents or Web sites containing activity-relevant information. Activities do not describe resources in detail but instead refer to resources managed by the VieCAR framework. When work focuses on creating or modifying an object, this object is represented as a shared *Artifact*. An artifact in one activity can become a resource in a subsequent activity. Finally, location-dependent activities specify the position at which the work is to be carried out.

We apply the concept of an *Action*, as introduced in [3], to capture the changes to the activity, the interaction between members, and the actual work carried out. An action is an atomic event that describes the progress of the activity toward its collaborative goal.

4 Relevance-based Ranking

With decentralized control, coordination and collaboration happens very locally, focused on a small set of activities. Nevertheless, decisions on collaboration and coordination in individual activities need to be based on the relevant global view. Our novel relevance ranking algorithm determines this view - the *activity context*. There are multiple definitions of *Context*. One of the most prominent definitions in the domain of computer science is by Dey and Abowd [1]:

> [...] any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves.

We define the concept of a context element based on our activity model describing activities, people, resources, devices, location, and deadlines. The proposed algorithm ranks context elements to establish the most relevant context of a user based on the current activity. Thus the goal of the algorithm is determining which context elements are relevant to the situation at hand. The resulting activity context is the primary input for adaptation of collaboration services.

The two main relevance criteria are activity distance and temporal distance. Activity distance is based on the activity graph (see Section 4.1) created by the activity model's parent-child relations. Calculating the difference between a context element's timestamp and the current time results in temporal distance. Combining both techniques results in a very powerful algorithm as we will demonstrate in the following section.

4.1 Ranking Algorithm

In this section we discuss our ranking approach. (1) We define an *activity graph* whose nodes are represented by activities. Each activity has a set of associated *context elements* (artifacts, resources, etc. see Figure 3). (2) We define an algorithm that labels the activity graph by assigning *activity distance* weights based on the relative distance of each activity to the user's current activity (i.e., the user's activity context). (3) We apply an algorithm that finds the set of activities that match a given set of context elements, which can be determined by, for example, a user-define query. (4) Finally, we rank obtained context-elements based on a set of metrics using an LSP-like algorithm.

4.1.1 Definitions

Let us define $\mathcal{A} = \{a_1, a_2, \ldots, a_n\}$ as the set of activities and the activity graph as $AG = (\mathcal{A}, R)$, whose vertices represent the set of activities $\mathcal{A}$ and $R = \{r_{ij} | a_i \overset{r_{ij}}{\rightarrow} a_j\}$ the set of relations as directed links between activities. In our activity model, AG is represented as a hierarchical tree structure of activities. Therefore, the set of relations R in AG correspond to $\texttt{child}(a)$, the set of child activities connected to a, $\texttt{parent}(a)$ denoting the parent activity of a with set cardinality $\|\texttt{parent}(a)\| = 1$, and $\texttt{sibling}(a)$ as the set of activities with the same parent activity as a. Let us denote $\{ce\}_i = \{ce_i^j \in \{ce\}_i | j = 1 \ldots n, i \in \{\mathcal{A}\} \equiv i \mapsto a_i\}$, as the set of context elements associated with activity a.

Matching function. Let us define a matching function $match(\{ce\}_\alpha, \{ce\}_\beta)$ to determine the containment of $\{ce\}_\alpha$ in $\{ce\}_\beta$ such that $\{ce\}_\alpha \subseteq \{ce\}_\beta$. The function *match* is calculated as $\bigwedge_{\{ce\}_\alpha} \bigcap_{ce_\alpha^i} \{ce\}_\beta = \top \vee \bot$ (*true* or *false*), where $\bigcap = ce_\alpha^i \wedge ce_\beta^1 \vee ce_\alpha^i \wedge ce_\beta^2 \vee \ldots \vee ce_\alpha^i \wedge ce_\beta^n$. Thus, $ce_\alpha^i \cap \{ce\}_\beta = \top$, if $\{ce\}_\beta$ contains an element of the same concept or class as ce_α^i.

Remark. One example of such a matching function is determining the set of relevant people (`foaf:Agent`) outside the current activity.

4.1.2 Labeling Activities

Let us define the weight of each activity, based on the relations r_{ij} in AG, as ($a = a_i$):

$$weight(a_j) = \begin{cases} W_c & \text{if } r_{ij} \in \texttt{child}(a) \\ W_p & \text{if } r_{ij} \in \texttt{parent}(a) \\ W_s & \text{if } r_{ij} \in \texttt{sibling}(a) \end{cases} \quad (1)$$

The weights W_c, W_p, and W_s can be seen as *distance* measures of activities that are directly related (linked) to a. However, in our proposed ranking algorithm we not only consider activities that are directly related to a, but also the activities in $\mathcal{A} \backslash a$. Let us define a labeling function $\Lambda(a, AG)$ that determines the relative distance of each activity $\mathcal{A} \setminus a$, given a, in the activity graph AG.

Algorithm 1 Labeling function $\Lambda(a, AG)$.

```
function CHILD($a$, child($a$))
    for all $a_j \in$ child($a$) do
        $weight(a_j) = W_c + weight(a)$
        if child($a_j$) $\neq \emptyset$ then
            Compute Child for $a_j$
        end if
    end for
end function
function SIBLING($a$, sibling($a$))
    for all $a_j \in$ sibling($a$) do
        $weight(a_j) = W_s + weight(a)$
        if child($a_j$) $\neq \emptyset$ then
            Compute Child for $a_j$
        end if
    end for
end function
function PARENT($a$, parent($a$))
    $weight(a_j) = W_p + weight(a)$
    if sibling(parent($a$)) $\neq \emptyset$ then
        Compute Sibling for parent($a$)
    end if
    if parent(parent($a$)) $\neq \emptyset$ then
        Compute Parent for parent(parent($a$))
    end if
end function
```

Algorithm 1 shows that the *importance* of each activity a_j is calculated as the aggregated weight when traversing AG from a given (selected) activity a. Thus, the weight of a_j depends recursively on activity a_i. Calculating the activity-based distance for activities and not for individual context elements, accelerates subsequent queries, as the measurements are reused.

4.1.3 Selecting Activities

Let us define a two-step algorithm that uses in the first step the labeling function $\Lambda(a, AG)$ to assign weights to each activity, given the selected activity a, and in the second step creating a vector **A** — the *candidate activities* that contain matching context elements. Algorithm 2 adds an activity a_i to the vector **A**, containing candidate activities, if a_i has a set of context elements associated with it that satisfy (match) the given input set $\{ce\}_\alpha$. The input set can be determined by a user define query to search for relevant context elements, given the user's current activity a.

Algorithm 2 Activity selection function *getCandidates*(a, AG).

```
$\mathbf{A} \leftarrow 0$
$AG_L \leftarrow \Lambda(a, AG)$            ▷ Labeled graph $AG_L$ based on $a$
for all $a_i \in AG_L$ do
    if $match(\{ce\}_\alpha, \{ce\}_{a_i}) = \top$ then
        $add(\mathbf{A}, a_i)$                  ▷ Adds $a_i$ to $\mathbf{A}$
    end if
end for
    return A
```

4.1.4 Ranking Context Elements

Let us define the set of metrics $M = \{m_1, m_2, \ldots, m_n\}$ that are associated with each context element and ω_{m_i} as the weight assigned to metric m_i such that $\sum_{i=0,\ldots,n} |\omega_{m_i}| = 1$. The score of each metric (i.e., the level of satisfaction) for a particular metric of ce is defined as follows:

$$score(ce(m)) = \begin{cases} 1 - \frac{\mathtt{max}(m)-m}{\mathtt{max}(m)-\mathtt{min}(m)} & \text{if } \omega_m \geq 0 \\ \frac{\mathtt{max}(m)-m}{\mathtt{max}(m)-\mathtt{min}(m)} & \text{otherwise} \end{cases} \quad (2)$$

Parameter	Description
ω_m	Weight assigned to metric m.
$\mathtt{max}(m)$	Maximum value within metrics of context elements.
$\mathtt{min}(m)$	Minimum value within metrics of context elements.
m	Metric value of a given context element.

Table 1. Score based on metrics.

In the second step, we rank individual context elements according to the selected set of metrics. Our approach to rank context elements based on metrics is using a simplified LSP method (see [2] for a detailed overview). The global score of a context element is given as:

$$E(ce) = \sum_{i=1}^{n} |\omega_{m_i}| * score(ce(m_i)), ce \in \{ce\} \quad (3)$$

The rank of each context element can be computed as shown in Algorithm 3. First, the candidates are selected from the activity graph based on the desired set of context elements (*containment* of query). The function *getCandidateElements* retrieves those context elements from a given activity a. Each context element is ranked using the LSP method by weighting and aggregating individual scores into a global score $E(ce)$.

The final set of rankings includes for every candidate context element ce the corresponding overall rank. The algorithm is parametric, since the user can assign a ranking

Algorithm 3 Ranking function $R(M, AG)$.

```
Require: $\sum_{|\omega|} = 1$
  $E \leftarrow 0$                                  ▷ Set of ranked context elements
  $\mathbf{A} \leftarrow getCandidates(a_s, AG)$    ▷ Selected activity $a_s$ for which $AG$ is labeled
  for all $a \in \mathbf{A}$ do
      for all $ce \in getCandidateElements(a)$ do
          for all $\omega_{m_i} \neq 0$ do
              $E[ce] \leftarrow E[ce] + |\omega_{m_i}| * score(ce(m_i))$
          end for
      end for
  end for
    return $E$                                      ▷ Ranked context elements
```

weight to each metric. In addition, the distance measurement for traversing the activity graph AG from the activity to its siblings, parent, and child activities is configurable. Details on the applied values are presented in the following section.

5 Experiments

5.1 Discussion Ranking Algorithm

In the given example in Figure 5, the applied metrics and weights for the Algorithm 3 are (1) activity-distance $w(d)$ and (2) temporal distance $w(t)$ of context elements. For both metrics we prefer lower values over higher values, i.e., closer activities and shorter time difference indicate more relevant context elements. Hence, both weights $w(d)$ and $w(t)$ have to be negative. Adjusting the weights determines the importance of one metric over the others. We demonstrate the algorithm's significance to self-adaptation of collaboration services, applying the same ranking weights ($|0.5|$) for $w(d)$ and $w(t)$ for every activity and context element respectively. For labeling AG, we set the edge weights as: parent, $W_p = 2$, child $W_c = 1$, and siblings, $W_s = 2$.

We have developed an intelligent meeting scheduling Web service, which helps participants to search for relevant documents. In dynamic collaboration, the user should not need to configure the scheduling service where to look for relevant documents. Instead, the scheduling service applies the relevance ranking algorithm to retrieve the most promising locations — in the given example *Shared Workspace Services*.

The activity graph in Figure 5 displays context elements of type Shared Workspace Service (SWS): Activity distance is given in upper right corner. Past dates indicate last access of SWS; whereas future dates planned documents available as templates in SWS. The graph represents part of the underlying collaboration structure, created via the Activity Management GUI depicted in Figure 4. We start to schedule a meeting in the scope of activity A8 on the 28th January. As we invoke the scheduling service, it queries the *Activity Manager* (Figure 2) for relevant context elements. The context elements used in our ranking example are *Resources* (Figure 3) of type SWS. Candidate SWS are those who are referenced within *Action* elements. In addition, we include SWS that are in future scheduled for managing already existing document templates. As both actions and activities contain a timestamp, the ranking Algorithm 3 is able to rank these context elements. Table 2 presents the resulting activity-distance rank $R(d)$, time-distance rank $R(t)$, and the aggregated rank $R(M, AG)$.

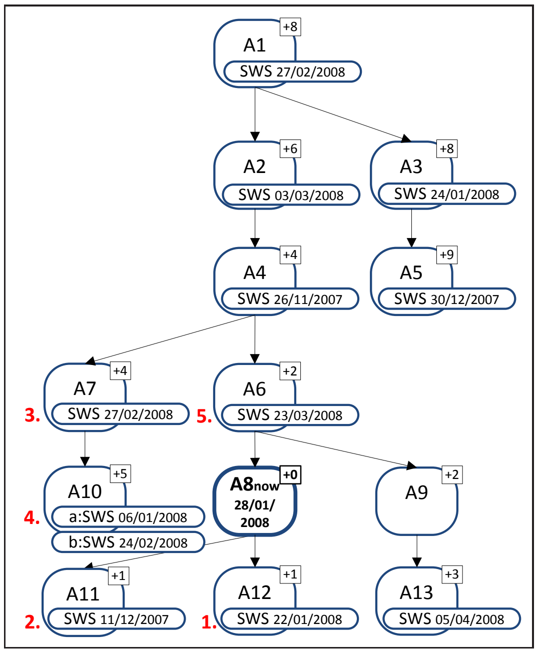
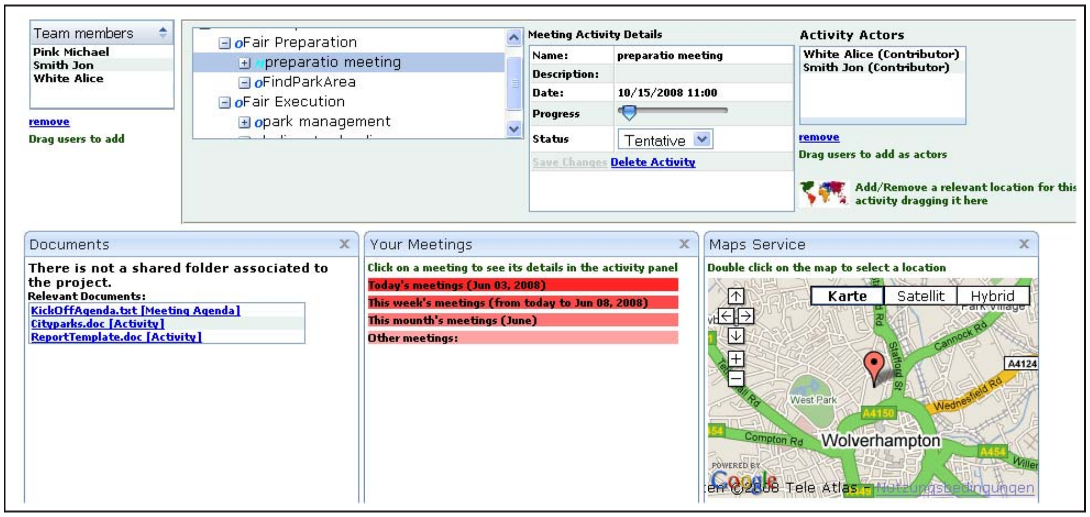


Figure 5. Activity Graph excerpt

Act.	d	Date	$R(d)$	$R(t)$	$R(M, AG)$
A12	1	22.01.08	(1) 1,00	(2) 0,96	(1) 98,24
A11	1	11.12.07	(1) 1,00	(9) 0,47	(2) 73,53
A07	4	27.02.08	(5) 0,63	(6) 0,68	(3) 65,37
A10-a	5	06.01.08	(7) 0,50	(3) 0,78	(4) 63,82
A06	2	23.03.08	(3) 0,88	(10) 0,39	(5) 63,16
A10-b	5	24.02.08	(7) 0,50	(4) 0,72	(6) 60,88
A03	8	24.01.08	(10) 0,13	(1) 0,99	(7) 55,66
A02	6	03.03.08	(9) 0,38	(8) 0,62	(8) 49,93
A13	3	05.04.08	(4) 0,75	(12) 0,24	(9) 49,26
A04	4	26.11.07	(5) 0,63	(11) 0,29	(10) 45,96
A01	8	27.02.08	(10) 0,13	(6) 0,68	(11) 40,37
A05	9	30.12.07	(12) 0,00	(5) 0,69	(12) 34,71

Table 2. Rankings example.

Context elements receiving high ranks in both time-distance metric and activity-distance metric will also rank



Pink Michael
Smith Jon
White Alice

oFair Preparation
opreparatio meeting
oFindParkArea
oFair Execution
opark management

Name:	preparatio meeting
Description:	
Date:	10/15/2008 11:00
Progress	
Status	Tentative

White Alice (Contributor)
Smith Jon (Contributor)

Add/Remove a relevant location for this activity dragging it here

There is not a shared folder associated to the project.
Relevant Documents:
KickOffAgenda.txt [Meeting Agenda]
Cityparks.doc [Activity]
ReportTemplate.doc [Activity]

Click on a meeting to see its details in the activity panel
Today's meetings (Jun 03, 2008)
This week's meetings (from today to Jun 08, 2008)
This mounth's meetings (June)
Other meetings:

Double click on the map to select a location

Figure 4. Activity Management GUI screenshot.

high the overall results (e.g., SWS in A12). A good position in a single ranking, however, does not guarantee final relevance, as this depends on the distribution of the other context elements. For example, the SWS in A11 is ranked 9th place in the time-distance metric, respectively ex aequo 1st in the activity ranking, ending up 2nd place in the overall ranking. In contrast, the SWS in A03 ranked 1st in the time-distance metric, but ranked 10th in the activity-distance metric, winds up in the middle on 7th place. The reason lies in the distribution of metric values. When a number of values (i.e., timestamps) lie close together (compared to the overall range) their intra metric ranking becomes less significant and the other metrics (i.e., activity distance) become dominant.

5.2 Performance Study

In this section we discuss basic performance characteristics of our ranking algorithm. The chart in Figure 6 shows the algorithm's execution time using a Java implementation of an object-oriented database. In our experiments we use db4objects[3]. Performed steps:

1. *Database Access* loads database into memory containing all activities and the graph AG which determines the activity structure.

2. *Populate AG* - selecting among all activities those activities that belong to AG.

3. *Graph Labeling and Ranking* - creating a new graph AG_L that can be smaller or equal to AG. In our ranking algorithm, we can choose the depth to which the ranking should be done to increase performance if AG is large. However, in our studies we have set the number of activities in AG_L to be equal as in AG.

[3]http://www.db4o.com/

In Figure 6 we see the *algorithm execution time* evaluated with different numbers of activities. The execution time linearly increases with the number of activities, showing that our algorithm will also scale to a larger number of activities.

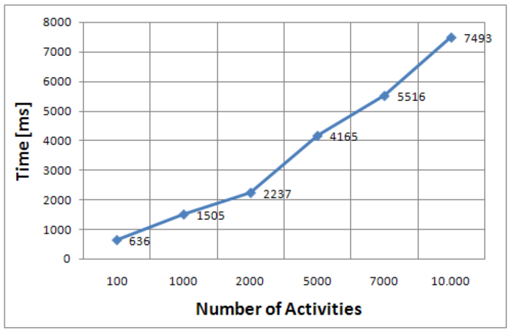


Figure 6. Algorithm execution time.

The performance analysis mainly focuses on run-time characteristics (shown in Figure 6) by generating activity graphs with different numbers of activities or varying levels of graph-depths. However, at this stage we only used small sets of context elements associated with activities.

6 Related Work

Work related to VieCAR includes research in the areas of *activity-centric* collaboration, *ranking* of resources and *self-adaptive* services in collaboration. Caramba [3] is a system supporting virtual teams in ad-hoc as well as collaborative processes. Caramba supports ad-hoc processes and therefore members of a team do not need to model a process in advance to achieve process-awareness. *Coordination models* are used to denote dependencies of work activities in a team. The activity model used in VieCAR allows the user to assemble activities in a hierarchical structure to manage collaborations. The Unified Activity Management (UAM) has been presented in, for example, [5, 6]. UAM defines a framework for supporting collaborative work around the concept of human activities. In contrast to UAM, VieCAR is a framework that allows distributed resources including Web services to be used in activity-centric collaborations. Furthermore, in contrast to above mentioned system, VieCAR comprises ranking algorithms to determine the relevancy of resources associated with activities.

A context-aware resource recommendation system has been presented in [7]; using ontologies to describe tools used in collaborations. VieCAR considers interactions that arise in collaborations, which act as source for the ranking of various resources based on a set of metrics. A task-centric approach for self-adaptation has been proposed in [4, 9], however, only considering user tasks and not the collaboration context based on joint activities. Previous approaches (e.g., see [8] and [10]) to personal service adaptation also focus on the user's context but do not consider the overall collaboration context. *Activity-centric computing* and relevance-based ranking strongly improve techniques for personal service adaptation.

7 Conclusion and Future Work

VieCAR's service-oriented architecture and *activity-centric* approach enables the management of interactions and collaborations in cross-organizational, distributed environments. Through VieCAR, human actors collaborate by defining joint activities that allow work-items to be structured in a flexible yet structured manner. Activities link together different collaborative resources such as people, documents, and Web services — i.e., the *activity context*. It becomes ever more important to recommend the most relevant resources based on the user's context. We have presented VieCAR's novel ranking approach to rank the relevancy of resources based on activities.

In the next steps we plan to utilize our *relevance ranking* algorithm to determine the most relevant context that can be used for *self-adaptation* of services and furthermore, ultimately, to achieve *self-managed collaboration*. Thus, services can respond in an optimal, context-aware way to dynamically changing situations in flexible collaboration environments. Specifically, our future work will focus on extending the ranking algorithm to include automatic adaptation of metric weights based on a feedback loop. This will reflect the embedding of a single activity within the overall collaboration context, leading to improved ranking results. Furthermore, we will analyze communication and resource use to reflect the distance between activities more accurately. In addition, we plan to extend the set of available metrics to allow for more adaptation specific rankings.

References

[1] A. Dey and G. Abowd. Towards a better understanding of context and context-awareness. In *Workshop on the What, Who, Where, When, and How of Context-Awareness at CHI 2000*, April 2000.

[2] J. J. Dujmovic. Continuous preference logic for system evaluation. In *IEEE Transactions on Fuzzy Systems*, volume 15, pages 1082–1099. IEEE Computer Society, 2007.

[3] S. Dustdar. "Caramba Process-Aware Collaboration System Supporting Ad hoc and Collaborative Processes in Virtual Teams". *Distributed Parallel Databases*, 15(1):45–66, 2004.

[4] D. Garlan, V. Poladian, B. R. Schmerl, and J. P. Sousa. Task-based self-adaptation. In *WOSS*, pages 54–57, 2004.

[5] B. L. Harrison, A. Cozzi, and T. P. Moran. Roles and relationships for unified activity management. In *International ACM SIGGROUP Conference on Supporting Group Work GROUP '05*. ACM Press.

[6] P. Moody, D. Gruen, M. J. Muller, J. Tang, and T. P. Moran. Business Activity Patterns: A New Model for Collaborative Business Applications, 2006.

[7] K. Ning, R. Gong, S. Decker, Y. Chen, and D. O'sullivan. A context-aware resource recommendation system for business collaboration. *Int. Conf. on E-Commerce Technology and the 4th IEEE Int. Conf. on Enterprise Computing (CEC/EEE 2007).*, pages 457–460, 23-26 July 2007.

[8] Q. Z. Sheng, B. Benatallah, Z. Maamar, M. Dumas, and A. H. H. Ngu. Enabling personalized composition and adaptive provisioning of web services. In *CAiSE*, pages 322–337, 2004.

[9] J. P. Sousa, V. Poladian, D. Garlan, and B. R. Schmerl. Capitalizing on awareness of user tasks for guiding self-adaptation. In *CAiSE Workshops (2)*, pages 83–96, 2005.

[10] Y. Yang, F. Mahon, M. H. Williams, and T. Pfeifer. Context-aware dynamic personalised service re-composition in a pervasive service environment. In *UIC*, pages 724–735, 2006.