A Quality- and Cost-based Selection Model for Multimedia Service Composition in Mobile Environments

Wolf-Tilo Balke and Jörg Diederich
L3S Research Center and University of Hannover, Hannover, Germany
E-mail: {balke,diederich}@l3s.de

Abstract

When moving from monolithic applications towards service-oriented multimedia frameworks, the composition of Web services to form complex multimedia workflows becomes a demanding problem. Especially mobile devices require a flexible composition strategy as they often have to move computationally complex or power-demanding tasks to powerful servers. Such a strategy also has to consider the changing environment due to movements of the device and it has to adapt to device-specific characteristics, e.g., the current battery level. Hence, mobile devices experience problems beyond the mere question of services' availability or successful execution. We propose the E^2Mon algorithm that monitors the execution chain of Web services and gracefully recovers from failures of individual services and network-specific or device-specific alarms. The sophisticated control flow dynamically chooses the quality-optimal and cost-optimal composition of available services handling both successive and parallel service execution.

1 Introduction

Service-oriented designs promise to offer the dynamic construction of even complex workflows from basic Web services fostering a maximum of reusability [1]. Such workflows often involve various services [5], which can either be executed sequentially or concurrently, and might also be time-critical as, for instance, in multimedia applications [12]. Web services are especially useful for mobile devices since they allow to move computationally complex or power-demanding tasks to stationary servers of the service providers. Previous approaches already discussed how to build workflows and developed powerful frameworks for their specification [6]. However, monitoring the actual execution especially of continuously running services (such as media streaming) and the automatic recovery from failures has not been addressed sufficiently yet.

In most scenarios a successful execution strongly depends on the application environment. Users generally want to accomplish a certain task, but are usually not interested in all intermediate steps. As an example, for streaming a video to hand-held devices, users want the video in the best possible quality and not exceeding a certain cost level. For this purpose, an automatic composition of services does not only have to deal with the service executions, but must also take technical requirements into account, for example, those related to the quality of the multimedia content and the display or battery capabilities of the mobile device [11]. Furthermore, it also has to consider requirements imposed by the user (or network provider) such as maximum costs [8].

Of course, service availability and matching capabilities are still crucial to ensure a successful task completion. This is especially true for mobile environments, where the availability of a service and its costs are very dynamic and can even depend on the current network provider. For example, a Web service might no longer be available when the user moves out of the service area of a Wireless LAN-based hotspot and has to switch to a wide-area network, such as UMTS [15]. Delivering multimedia content in such dynamic environments leads to two major problems:

1. If a service is no longer available, it has to be replaced on the fly to avoid service outages caused by the time necessary to discover alternative services.
2. If the environment changes (e.g., the cost of a service increases or the battery level of the user device becomes low), a new service chain might be better suited.

Therefore, it is very important to constantly monitor running Web services and keep track of alternative services to replace them in case of service failures. If several alternative services exists, it is important to choose the most efficient replacement strategy, for example, trying to reuse the outcome of already completed services in the service chain to avoid unnecessary service invocations.

In this paper, we describe the E^2Mon algorithm that monitors Web services regarding their costs and availability and keeps track of alternative Web services to be able to

quickly react to failures and alarms. For this purpose, the algorithm caches the descriptions of all available Web services that can be used to implement a certain task, together with their associated costs. Hence, a replacement service can be found quickly without a time-consuming and power-draining service re-discoveries. Furthermore, E^2Mon takes internal conditions into account, such as the current state of the battery, when computing the optimal service path.

The paper is organized as follows: Section 2 presents a sample application scenario and discusses related work. Section 3 describes the E^2Mon algorithm and its basic characteristics. Section 4 explains the monitoring algorithm along a use case scenario. The paper concludes with a short summary and outlook in Section 5.

2 Scenario: WorldCup 2006

Consider the following scenario: Bob is interested in the Soccer WorldCup 2006 and wants to follow events using his PDA while commuting home from work. His PDA has a small video screen and can utilize several network technologies such as Wireless LAN, UMTS and GPRS. For illustration purposes, the following example scenario is used throughout the paper (cf. Fig. 1). Bob starts at work us-

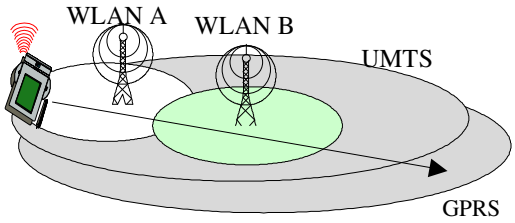

Figure 1. Mobile scenario

ing the company's Wireless LAN (WLAN A). Walking to the train station, he has to change to WLAN B (a hotspot's Wireless LAN network), and then to the city's UMTS network while riding the train. Near his house, there is no UMTS coverage yet, so he has to use a GPRS network there. Finally, his battery level may become low at the end of the day while walking home from the train station.

Bob's PDA runs a sophisticated application automatically selecting a current game from the many available events, based on Bob's preferences or general recommendations, and plays it on his PDA. This basic task of the application can be accomplished in different ways, so the application designer may have anticipated several possible service chains with different workflows (cf. Fig. 2):

The first Web service S1 ('Recommender WS') determines the event particularly interesting for Bob given his profile. Following the InfoPyramid [14] approach, media can be delivered in different modalities based on technical constraints: as a video stream by a video Web service, an

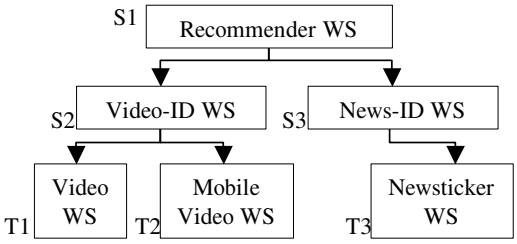

Figure 2. Possible service chains

audio stream by an audio Web service (with or without still pictures), or a text stream from a newsticker Web service. Our example scenario is limited to video and textual delivery (T3), but distinguishes between a 'Video WS' (T1) (for stationary and mobile users) and a 'Mobile Video WS' (T2) especially suited for mobile users with small displays (zooming into the scenery manually rather than automatically scaling it down). To be able to stream the data (video, mobile video, or newsticker) of a recommended event, the official ID of the event is necessary. For this purpose, there are separate Web services for videos and textual information, a 'Video-ID WS' (S2) and a 'News-ID WS' (S3), in between the recommender and the streaming services.

All services in Fig. 2 are to be executed sequentially. However the three services T1–T3 are so-called *concurrent units*, i.e., they act as a placeholder for a set of concurrent Web services required for the concrete implementation. In this scenario, an implementation of the 'Video WS', the 'Mobile Video WS', and the 'Newsticker WS' require several Web services to be executed concurrently. The

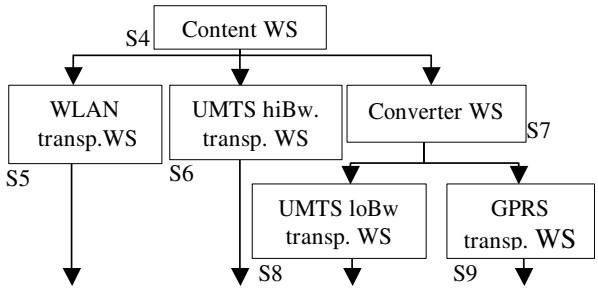

Figure 3. Video Web service concurrent unit

'Video WS' (cf. Fig. 3) is composed of a 'Content WS' (S4) that can deliver the video directly to a 'WLAN transport WS' (S5) or to a 'High-bandwidth UMTS transport WS' (S6). Alternatively, the video stream can be delivered to a 'Converter WS' (S7), where it is transformed (e.g., using transcoding, frame rate reduction, or color reduction) into a low-bandwidth stream. Afterwards, a 'Low-bandwidth UMTS transport WS' (S8) or a 'GPRS transport WS' (S9) can be used for actually transmitting the stream to Bob's

PDA. Regarding the remaining two concurrent units T2 and T3 , the 'Mobile Video WS' is instantiated by a 'Mobile Content WS' (S10) and a 'GPRS transport WS' (S9) only. The 'Newsticker WS' itself is composed of a 'News Content WS' (S11), which can stream the data either over a 'WLAN transport WS' (S5) or a 'GPRS transport WS' (S9). All possible service chains are summarized in Table 1.

Nr.	Service chain	Description
1	S1-S2-S4-S5	Video-WLAN
2	S1-S2-S4-S6	Video-UMTS-high_bandwidth
3	S1-S2-S4-S7-S8	Video-UMTS-low_bandwidth
4	S1-S2-S4-S7-S9	Video-GPRS-low_bandwidth
5	S1-S2-S10-S9	Mobile Video-GPRS
6	S1-S3-S11-S5	Newsticker-WLAN
7	S1-S3-S11-S9	Newsticker-GPRS

Table 1. Possible service chains

Typically, there will be several providers that are offering implementations for each particular Web service. For brevity, the scenario in this paper is limited to only one implementation for each Web service being named as I1 (Implementation for S1), I2 (Implementation for S2) etc., with only the following three exceptions:

- We assume two implementations for the 'WLAN transport WS' (WLAN A (I5a) and WLAN B (I5b)).
- There exist two implementations for the 'Content WS', one providing videos in Bob's mother tongue German (I4a) and one providing English videos (I4b).
- Finally, there are two implementations for the 'Converter WS', one resulting in H.263 output (I7a) and one in H.264 output (I7b).

Of course, this scenario is easily extensible with a variety of further Web services, e.g. a buffering web service, which given the small storage amount on a PDA stores the last ten minutes of a stream to cater for TiVo-style interaction, anonymization services, services to add subtitles, etc.

2.1 Related Work

The problem of composing complex workflows from basic services has already attracted a lot of attention in the context of modeling reusable workflows for business applications, e.g., [5]. Being strongly supported by industry, BPEL4WS (currently continued as WS-BPEL and standardized by OASIS) has become the de-facto standard for Web service orchestrations [6].

On a more abstract level, the architecture in [7] uses planning techniques and a request language to instantiate service chains and also proposes limited monitoring capabilities by interleaving planning and execution. But discovering and selecting adequate services does not only depend on the services' capabilities, but also on user preferences, see e.g., [2, 3] for cooperative frameworks. Moreover, abstract characteristics like a service's reputation [10] or the estimated service quality [8] have been found necessary. However, the monitoring of such criteria (and adequate adaptations of the workflow instantiation) during workflow execution has not been fully addressed yet.

In terms of capability matching, especially Semantic Web approaches need to be considered building workflows and simulating execution with powerful formalisms like state machines, Petri nets, or description logics (see e.g., [4] or [13]). Ontology-based descriptions based on DAML+OIL recently led to the OWL-S [9] standard for service capability descriptions. Although these approaches are generally quite powerful in the workflow instantiation, again monitoring the actual execution is not considered.

3 A Cost-Based Algorithm

This section describes the E^2Mon (Execution and Environment Monitor) algorithm, which monitors both the execution of a service chain with sequential and concurrent units and the respective environment. If necessary, E^2Mon recomputes the service costs and switches to another service chain in case the currently executed service is no longer optimal or any service fails. Specifically, E^2Mon tries to reuse the results of already invoked services when computing new 'best' service chains.

3.1 The Cost Function

One important part of the algorithm is to efficiently find the 'best' service chain. Generally speaking, for assessing the value of any instantiation, not only the parameters from the actual execution (success or failure) have to be considered, but also a variety of costs depending on device capabilities or the execution environment. The approach in [8] proposes an extensible framework for modeling such costs for individual applications.

E^2Mon can in principle support arbitrary cost functions. However, in our example scenario a simple additive cost function is sufficient to show the capabilities of the algorithm (more complex monotonic functions have to be expected in real deployment scenarios):

$$F(I,t) = M(I) + Q(I) + w(t) * P(I) + U(I) \qquad (1)$$

For each service implementation I, the cost function includes provider costs $M(I)$, Quality of Service costs $Q(I)$ (the lower, the better the quality), power consumption costs $P(I)$ (low for low consumption), and some costs $U(I)$ based on user preferences (smaller for preferred instances).

For example, WLAN A might have lower provider costs $M(I)$ than WLAN B, if we assume that Bob can use his

company's WLAN without charges. To include the different states of the battery of Bob's PDA, the cost function can include time-variable weights like $w(t)$, which is 0 when the battery is full, but is set to 1 on a low-battery event (simultaneously raising an alarm in the monitoring service).

3.2 Assumptions

For ease of understanding, this paper makes the following assumptions:

- All Web services capabilities are described correctly (i.e., no malicious services) and semantically well-defined, independent of the service provider.
- All service implementations for one specific Web service type can be used interchangeably. Service providers will disclose correct costs and other information, e.g., the expected Quality of Service.
- There is one central instance (the monitor / proxy) keeping all results of previously executed Web services to reuse them for failure recovery.

3.3 The E^2Mon Algorithm

The E^2Mon algorithm for choosing, executing and monitoring adequate service compositions can basically be decomposed into four distinct phases:

I. **Workflow Enumeration**: all possible (or known) workflows are enumerated and their respective service chains extracted. We consider possible workflows to be explicitly given for a number of anticipated usage scenarios reflecting the idea of usage patterns. The problem of dynamically building such workflows for a given problem has been addressed in the literature, e.g., [6, 7], but is orthogonal to the scope of this paper.

II. **Service discovery**: for all extracted services, providers of suitable implementations are discovered. E^2Mon uses periodic re-discoveries during the execution phase, since performing a discovery only whenever a service fails is too time-consuming for time-critical multimedia applications such as media streaming. Moreover, whenever a more cost-optimal instantiation chain is discovered, a respective alarm can be raised and handled.

III. **Service chain selection**: the cost-optimal instantiation of all possible workflows is selected. The selection determines what workflow should be executed, usually the one having acceptable provider costs, good availability of the services involved, and sufficient Quality of Service guarantees.

IV. **Execution monitoring**: the selected service chain has to be executed either sequentially or in parallel (concurrent units) in a supervised fashion. If all service executions succeed, the task was completed successfully. On each service failure, the algorithm has to reassess costs and select a different service implementation or in some cases even a different service chain. A critical issue is whether the failed service can be replaced at all (dealing with time-critical operations, E^2Mon does not consider retry operations for failed services).

E^2Mon itself is run as Web service either on the client (e.g., Bob's PDA) or on a proxy machine to reduce the communication load of the PDA for Web service invocation and monitoring. Such a proxy, however, is only useful if alarms (e.g., battery low) are rare compared to external events (service failures), as otherwise communicating the alarms can lead to a high communication load.

In the remainder of the section, the E^2Mon algorithm will be explained in detail, starting with its main data structures. Let W be a map of possible workflows to deal with task T, S be an array of vectors to contain the necessary services of the individual service chains of different workflows, and P be an array of Boolean vectors stating whether each service within a service chain has to be executed atomically (false) or concurrently with the following service in the service chain (true). Furthermore, let S_{exec} be a Boolean vector stating whether a particular service chain in S is currently executable or not (i.e., whether Web service instances are actually available for all services in the service chain or not). Let C be an inverted service index containing discovered implementations for each service and K be a table storing the external costs of each implementation. Let V be an array of vectors to contain the cost-optimal implementation I for each service under cost function $F(I, t)$, where parameter t is a global variable related to the internal status of the end device. Furthermore, let V_{min} be an array of real values containing the current aggregated costs of all executable workflows, and V_{chain} an array of integers connecting one row in V with the accompanying service chain in S. Finally, let n be the number of different service chains to implement T, m be the index in V to that service chain with currently minimal costs, and r be the number of currently executable service chains (= size of V).

E^2Mon: **Monitoring the Service Chain Execution and the Environment**

I. Workflow Enumeration

/* Construct array S (the service chains) and array P (information about atomic services / concurrent units) */

0. Set $n := 0$ and get all internal device information (t)
1. If W does not contain entries for task T terminate with error, else for every valid service chain c in W do

1.1. Set $n := n + 1$
1.2. Set $j := 1$
1.3. For each service s in c do /* Iterate over the services of the current service chain c in W */
1.3.1. Set $S[n, j] := s$
1.3.2. Set $j := j + 1$
1.3.3. If s to be executed concurrently with next service in c Set $P[n, j] := true$
else Set $P[n, j] := false$
1.4. Set $S_{exec}\,[n] := true$ /* init */

II. Service Discovery

/* Construct the service index C and the cost table K */
2. For $i := 1$ to n do /* iterate over all service chains */
2.1. Set $j := 1$
2.2. While $S[i, j]$ not nil do /* iterate over chain elements */
2.2.1. If $S[i, j]$ is no element of index C
2.2.1.1. Create a key with name $S[i, j]$ in C, discover all possible implementations for service $S[i, j]$, and store them in C using that key.
2.2.1.2. For each discovered implementation: get all external information (costs) from the respective providers of the implementation and store it in K
2.2.2. If C does not contain any implementation for $S[i, j]$ and $S[i, j]$ has not already been successfully executed and the results cached: Set $S_{exec}\,[i] := false$
2.2.3. Set $j := j + 1$

III. Service Chain Selection

/* Construct the 'best implementation array' V and V_{min} */
3. Set $r := 0$ /* number of implemented service chains */
4. For $i := 1$ to n do /* iterate over service chains */
4.1. if $S_{exec}\,[i] == true$
4.1.1. Set $j := 1$
4.1.2. While $S[i, j]$ not nil do
4.1.2.1. Compute the cost function $F(I, t)$ for every implementation I of service $S[i, j]$ using the cost table K. Choose I_{min} having the minimum value (ties can be broken arbitrarily).
4.1.2.2. Set $V[r, j] := I_{min}$
4.1.2.3. If no result of execution of I_{min} cached: Set $V_{min} := V_{min} + F(I_{min}, t)$
4.1.2.4. Set $V_{chain}\,[r] := i$
4.1.2.5. Set $j := j + 1$
4.1.3. Set $r := r + 1$

IV. Execution Monitoring

5. If $r == 0$, terminate with error, else set m to the index of the minimum value in V_{min} and j to 1+ the length of the common prefi x of the successfully fi nished part of the previously executed service path and the new service path m
6. While $V[m, j]$ is not nil do
6.1. Set $y := V_{chain}[m]$ /* index of the currently executed service path */
6.2. Set $k := 0$
6.3. Set $L := V[m, j]$ /* set of concurrent services */
6.4. While $P[y, j + k] == true$
6.4.1. Set $k := k + 1$
6.4.2. Set $L := L \cup V[m, j + k]$
6.5. Execute all service implementations in L concurrently
6.6. Listen for the next event
6.7. **On event: k successful terminations**
6.7.1. For all service chains c in V with the same prefi x $V[m, 1] \ldots V[m, j + k]$, do set $V_{min}[c] := F(V[m, j] + V[m, j + k], t)$
6.7.2. Set $j := j + k + 1$
6.7.3. Cache the result of all service executions for later reuse
6.8. **On event: service execution of $V[m, f]$ failed and $j \leq f \leq j + k$:**
6.8.1. Remove $V[m, f]$ from C for key $S[y, j]$
6.8.2. Stop all running services in L
6.8.3. If there is no further implementation in C for key $S[y, j]$
6.8.3.1. For all service chains s in S which contain $S[y, j]$: Set $S_{exec}[s] := false$
6.8.3.2. Remove $S[y, i]$ from C
6.8.4. Proceed with step 3 /* fi nd alternative implementation or new service chain */
6.9. **On event: time-out** (i.e., periodic rediscovery)
6.9.1. Initialize S_{exec} to true
6.9.2. Delete C and K and execute step 2
6.9.3. Generate event: local /* i.e., go to step 6.10.1 */
6.10. **On event: local** (e.g., battery low)
6.10.1. Update the internal device information (t) according to the kind of alarm and recompute V and V_{min} (like in steps 3 + 4)
6.10.2. Let m_{new} be the new index of the minimal value in V_{min}
6.10.3. if $y == V_{chain}[m_{new}]$
6.10.3.1. Set $k' := 0$
6.10.3.2. Set $L' := V[m, j]$ /* set of concurrent services */
6.10.3.3. While $P[y, j + k'] == true$
6.10.3.3.1. Set $k' := k' + 1$
6.10.3.3.2 Set $L' := L' \cup V[m, j + k']$
6.10.3.4. if $L == L'$ continue with 6.6
6.10.4. Stop all services in L + proceed with step 5

3.4 Discussion: Correctness & Efficiency

The correctness of the E^2Mon algorithm is given by its event handling: the algorithm will always execute the cost-

optimal service chain with a maximum offset in effi ciency given by the periodic rediscovery time span.

Due to the enumeration of all workfbws that can be instantiated, and the assumption of interchangeable instantiations for identical services within each individual workfbw chain, E^2Mon can choose a cost-optimal instantiation after phases I and II. Afterwards, there are only three ways to violate the cost-optimality:

- A new/existing service chain becomes executable/non-executable (i.e., there is a change in C and/or S_{exec}),
- the current costs of already discovered services change (i.e., a change in K),
- or the utility function $F(I,t)$ changes, usually triggered by device-dependent alarms.

When a service chain becomes non-executable (e.g., after changing the network or in case of provider problems) there are two cases: either it affects the currently executed chain, in which case the event is handled by our service failure recovery, or it is a currently not executed chain. In the latter case, the periodic rediscovery will detect non-availabilities of not yet executed services, mark all affected service chains as not executable, and recompute the cost-optimal chain. If a new service chain becomes executable (or a new, different implementation for any service becomes available), the periodic rediscovery event will instantiate the service chain. After recomputing all service costs, it will replace the currently executed chain if necessary. In the latter cases, the offset is at most the periodic rediscovery time span, whereas a failure event is handled immediately.

If the costs of already discovered services change (e.g., the expected Quality of Service, or execution costs), either the provider has to advertise the change or the costs changed because of changes in the environment (e.g., roaming to a new network, or a low battery warning). In both cases either a local event or the rediscovery would initiate the change to the cost-optimal solution. In contrast, the utility function can only change by local events (including user interaction), which are always handled immediately.

4 A Use Case Scenario for Mobile Multimedia Service Composition

To provide a better understanding of all data structures and the selection and instantiation of the best service chains, the E^2Mon algorithm is applied to the scenario introduced in Sect. 2. This includes an illustration of how the algorithm reacts to scenario dynamics, i.e., when Bob moves from WLAN A to WLAN B, then to a UMTS networks and to a GPRS network, where eventually his battery level becomes low.

4.1 Initialization

In the workfbw enumeration phase I (steps 0 and 1), E^2Mon populates the array of the service chains S, marks in P whether a particular service is atomic or has to be executed concurrently with the following service, and sets S_{exec} to whether each service chain is currently executable or not (i.e., if an implementation exists for all services). For simplicity, all three data structures are shown in one table (cf. Table 2) with P being the background color of each cell (white=false, gray=true) and S_{exec} the last column. As a result, $n := 7$ (i.e., there are seven alternative service chains).

Nr.	1	2	3	4	5	6	S_{exec}
1	S1	S2	S4	S5	nil		True
2	S1	S2	S4	S6	nil		True
3	S1	S2	S4	S7	S8	nil	True
4	S1	S2	S4	S7	S9	nil	True
5	S1	S2	S10	S9	nil		True
6	S1	S3	S11	S5	nil		True
7	S1	S3	S11	S9	nil		True

Table 2. Array S, P (shaded), and S_{exec}

In phase II, the service discovery phase, the service index C (cf. Table 3) and the cost table K (cf. Table 4) are built.

S1	S2	S4	S5	S6	S7	S8	S9	S3	S11
I1	I2	I4a	I5a	I6	I7a	I8	I9	I3	I11
		I4b	I5b		I7b				

Table 3. Service index C

Implementation	M	Q	P	U	$F(I,t)$ ($w(t)$=0)
I1	0	0	2		0
I2	1	0	2		1
I4a	0	0	64	4	4
I4b	1	0	64	1	2
I5a	0	0	4		4
I5b	1	0	4		5
I6	15	1	8		23
I7a	2	4	0		6
I7b	3	2	0		5
I8	6	4	4		10
I9	3	15	4		18
I3	1	0	2		1
I11	1	17	2	0	18

Table 4. Cost table K & cost function $F(I,t)$

In this scenario, all services are advertised in all networks except for the 'Mobile Video Content WS' (I10), which is only advertised in the GPRS network.

The cost table K contains all individual costs to compute the cost function $F(I,t)$, i.e., the provider costs M as advertised in the registry, the quality costs Q, the power consumption P, and a user preference U that is only provided for content-related services (e.g., to implement a language preference for video services). Please note that how to determine necessary costs for specifi c applications is beyond the scope of this paper, which only demonstrates their impact on the E^2Mon algorithm. As S10 is the only service with no discovered implementation, only $S_{exec}[5] := false$ for now.

In phase III (service chain selection; steps 3 and 4) the best implementation of each service chain in S is computed (using the cost function F) and stored in V, together with V_{min}, as aggregated costs of a service chain (cf. Table 5; V_{chain} represents the index of the service chain in S). As a result, $r := 6$ as service chain 5 cannot be executed (due to not having an implementation for S10).

Nr.	1	2	3	4	5	6	V_{min}	V_{chain}
1	I1	I2	I4b	I5a	nil		7	1
2	I1	I2	I4b	I6	nil		26	2
3	I1	I2	I4b	I7b	I8	nil	18	3
4	I1	I2	I4b	I7b	I9	nil	26	4
5	I1	I3	I11	I5a	nil		23	6
6	I1	I3	I11	I9	nil		37	7

Table 5. Array V (best implementation of S), V_{min} (minimal costs), V_{chain} (index into S)

From the three services with alternative implementations, I4b was chosen due to Bob's language preferences, which dominate the higher provider costs in this case. I5a (WLAN A) is preferred because of the lower provider costs, and I7b (the H.264-based converter) as a higher quality dominates higher provider costs.

In phase IV (execution and monitoring phase; steps 5+6), $m = 1$ initially, since service chain 1 (Video-WLAN), has the lowest costs ($V_{min} = 7$). Then, $j = 1$ and the service chain is executed in step 6, starting with the recommender service I1 (atomic as $P[1,1]$=$false$). Since all service chains start with I1, V_{min} is decreased in step 6.7.1 for all service chains by the cost of I1. Then $j = 2$ and I2 is executed successfully. As a result of step 6.7.1, V_{min} is decreased by 1 (the total costs of I2) for the indices 1–4 (all service chains starting with I1 and I2), resulting in $V_{min} = \{6, 25, 17, 25, 23, 37\}$. Finally, I4b and I5a are executed concurrently to stream the video data to Bob's PDA.

4.2 Monitoring Dynamics

Let's assume that the current WLAN transport service fails, as Bob moves out of the coverage area of WLAN A. Thus, I5a fails and is removed from C for the key S5. Also, I4b is stopped and I5b (WLAN B) is found in C as alternative for I5a (WLAN A). Hence, V, V_{min}, and V_{chain} are recomputed in the service chain selection phase, where I5a is replaced by I5b in service chains 1 and 6 in V. Furthermore, $V_{min}+ = 1$ because $(F(I5b,t) - F(I5a,t)) = 1$. As a result, $V_{min} = \{7, 25, 17, 25, 24, 37\}$. In the last step to recover from the failure of WLAN A, $j = 3$ (the previously executed service chain is the same, so j is set to the next service to be executed according to step 5) and steps 6.1–6.5 restart the video streaming using I4b and I5b (WLAN B). To invoke I4b, the cached results of I2 are utilized.

On failure of I5b (WLAN B), s5 is deleted entirely from C and the service chains 1 and 6 are marked as 'not executable' ($S_{exec}[1] = S_{exec}[6] := false$) because there is no alternative implementation for s5. Compared to the previous state, service chains 1 and 6 are deleted from V and V_{min} and the best service chain to be executed is the service chain 3 ($V_{min} = 17$) using the 'Converter WS' I7b and the 'Low-bandwidth UMTS transport WS' I8 (cf. Table 6).

Nr.	1	2	3	4	5	6	V_{min}	V_{chain}
1	I1	I2	I4b	I6	nil		25	2
2	I1	I2	I4b	I7b	I8	nil	17	3
3	I1	I2	I4b	I7b	I9	nil	25	4
4	I1	I3	I11	I9	nil		37	7

Table 6. V, V_{min},V_{chain} after failure of WLAN B

When the UMTS network fails, service chains 2 and 3 are marked as non-executable (no alternative implementations available) and are removed from V and V_{min}. Hence, service chain 4 (using the 'Converter WS' and the 'GPRS transport WS') becomes the best implementation.

Due to the periodic service discovery in step 6.9, the PDA discovers the implementation I10 of the 'Mobile Video WS' and updates the service index C. Now, service chain 5 becomes executable ($S_{exec}[5]$=$true$). Assuming $M(I10) = 2, Q(I10) = 2, P(I10) = 30$, and $U(I10) = 1$, the cost function becomes $F(I10) = 5$ (for $w(t) = 0$). Finally, V and V_{min} are recomputed (cf. Table 7) and service chain 3 is executed reusing again the cached result of the invocation of I2 as input for invoking I4b.

Nr.	1	2	3	4	5	6	V_{min}	V_{chain}
1	I1	I2	I4b	I7b	I9	nil	25	4
2	I1	I3	I11	I9	nil		37	7
3	I1	I2	I10	I9	nil		24	5

Table 7. V, V_{min}, V_{chain} after discovery of I10

Finally, the battery of Bob's PDA runs low and creates a 'low battery' event. Hence, weight $w(t) = 1$ and the cost function F is recomputed for all implementations in C.

	I1	I2	I4a	I4b	I7a	I7b	I9	I3	I11	I10
$F(I,t)$, $w(t)$=0	0	1	4	2	6	5	18	1	18	5
$F(I,t)$, $w(t)$=1	2	3	68	65	6	5	22	3	20	35

Table 8. Cost function after battery-low event

As a result, $V_{min} = \{92, 45, 57\}$ for the service chains 4, 7, and 5 and the 'Newsticker WS' (service chain 7) is executed, as it consumes far less power than the video services (cf. Table 4). Note that service chain 7 is cheaper although only the cached results of I1 can be reused to execute I3 atomically and to start I10 and I9 concurrently afterwards.

5 Summary and Outlook

The flexible composition of basic Web services to instantiate more complex workflows promises a maximum of component reusability in service-oriented architectures. Recently, a lot of effort has been invested into the automatic orchestration of services and the graceful recovery from service failures. However, for effective service provisioning a closer supervision of the actual service execution is vital. This is especially true in the case of mobile devices, where environmental variables may rapidly change.

In this paper, we presented E^2Mon, a novel monitoring scheme for the execution of Web service compositions that dynamically adapts to the execution environment. E^2Mon is designed for the deployment of service compositions on mobile devices, which due to their limited capabilities and possible network changes (roaming) also need a more dynamic cost-control. Costs in this case have to be understood as complex, application-specific utility functions that may involve actual provisioning costs, Quality of Service constraints, user preferences, etc.

We have shown E^2Mon to be efficient with a maximum offset to the cost-optimal solution of a periodic rediscovery time span. The applicability of E^2Mon was showcased in a multimedia service scenario with media streaming, where services have to be composed in a sequential fashion as well as in concurrent units. E^2Mon dynamically responds to a wide variety of events, including service failures, network changes, the discovery of new service implementations, and device-specific alarms, like low battery warnings.

Our future work will focus on the creation of a testbed with mobile clients, where the practical scalability of the E^2Mon approach will be investigated further for more sophisticated applications beyond simple media streaming. We will also investigate good application-dependent defaults for the necessary cost functions and for parameters like the optimal periodic rediscovery time span.

Acknowledgements

This work was funded within the Emmy Noether Program of the German Research Foundation (DFG).

References

[1] G. Alonso, F. Casati, H. Kuno, and V. Machiraju. *Web Services: Concepts, Architectures, and Applications*. Springer, 2004.

[2] W.-T. Balke and M. Wagner. Cooperative Discovery for User-Centered Web Service Provisioning. In *IEEE International Conference on Web Services*, pages 191–197, 2003.

[3] W.-T. Balke and M. Wagner. Towards Personalized Selection of Web Services. In *International World Wide Web conference; Alternate track on Web Services*, 2003.

[4] T. Bultan, X. Fu, R. Hull, and J. Su. Conversation specification: a new approach to design and analysis of e-service composition. In *International World Wide Web conference*, pages 403–410, 2003.

[5] F. Casati, S. Ilnicki, L.-J. Jin, V. Krishnamoorthy, and M.-C. Shan. Adaptive and Dynamic Service Composition in eFlow. In *International Conference on Advanced Information Systems Engineering (CAISE)*, pages 13–31, 2000.

[6] F. Curbera, Y. Goland, J. Klein, F. Leyman, D. Roller, S. Thatte, and S. Weerawarana. Business Process Execution Language for Web Services (BPEL4WS) 1.0, 2002.

[7] A. Lazovik, M. Aiello, and M. P. Papazoglou. Planning and Monitoring the Execution of Web Service Requests. In *International Conference on Service-Oriented Computing*, pages 335–350, 2003.

[8] Y. Liu, A. H. Ngu, and L. Z. Zeng. QoS computation and policing in dynamic web service selection. In *International World Wide Web conference on Alternate track papers & posters*, pages 66–73, 2004.

[9] D. L. Martin, M. Paolucci, S. A. McIlraith, et al. Bringing Semantics to Web Services: The OWL-S Approach. In *International Workshop on Semantic Web Services and Web Process Composition*, pages 26–42, 2004.

[10] E. M. Maximilien and M. P. Singh. Conceptual model of web service reputation. *SIGMOD Rec.*, 31(4):36–41, 2002.

[11] S. Mohapatra, R. Cornea, N. Dutt, A. Nicolau, and N. Venkatasubramanian. Integrated power management for video streaming to mobile handheld devices. In *ACM International Conference on Multimedia*, pages 582–591, 2003.

[12] K. Nahrstedt and W.-T. Balke. A taxonomy for multimedia service composition. In *ACM International Conference on Multimedia*, pages 88–95, 2004.

[13] S. Narayanan and S. A. McIlraith. Simulation, verification and automated composition of web services. In *International World Wide Web conference*, pages 77–88, 2002.

[14] J. R. Smith, R. Mohan, and C.-S. Li. Scalable multimedia delivery for pervasive computing. In *ACM international conference on Multimedia*, pages 131–140, 1999.

[15] Q. Zhang, C. Guo, Z. Guo, and W. Zhu. Efficient Mobility Management for Vertical Handoff between WWAN and WLAN. *IEEE Communications Magazine*, 41(11):102–108, 2003.