Reliable Discovery and Selection of Composite Services in Mobile Environments

Lucia Del Prete
Dept. of Computer Science
University College London
Gower Street, London WC1E 6BT, UK
L.DelPrete@cs.ucl.ac.uk

Licia Capra
Dept. of Computer Science
University College London
Gower Street, London WC1E 6BT, UK
L.Capra@cs.ucl.ac.uk

Abstract

Service providers as we know them nowadays are the always-on "static" web service providers, that aim at Five9 availability (99.999%). Formal, or de-facto, standards, such as WSDL and BPEL, have become technology enablers for the easy discovery, use and coordination of such services. However, we envisage tomorrow's services to become increasingly pervasive, being deployed within buildings, transport systems, markets, as well as people portable devices. Such services will be, by their own nature, simple and fine grained; as a consequence, service composition will become crucial to deliver rich functionalities that satisfy end users requests. Composing services in mobile environments opens up significant challenges. In particular, the Five9 availability assumption no longer holds: the higher the dynamic nature of the environment, the higher the chances that services will move out-of-reach before the composition completes, causing the service as a whole to fail. We argue that, in order to enable the successful completion of compound services, the reliability of the composition must be measured and reasoned about. In order to do so, we propose to dynamically deploy a prediction model to estimate the duration of colocation between component services. These estimates are fed in input to a service composition semantics reasoner, which then autonomically selects those providers, within the current environment, that maximise the chances of successful compound service completion. We demonstrate the positive impact that the reliability reasoning has onto the ratio of successfully completed compound services in a typical human movement scenario.

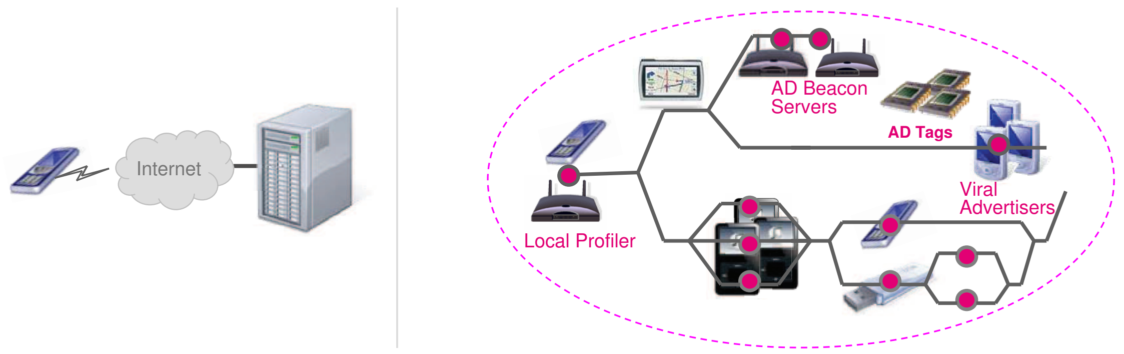
1 Introduction

Two major trends have been observed in recent years: the enormous evolution of mobile technology, and the transformation of the Internet user from consumer to producer of content. Portable devices (e.g., 3G mobile phones, portable digital assistants, etc.) have seen their computing capabilities (e.g., processing power and memory availability) grow according to Moore's law. Additional functionalities, such as digital cameras, MP3 players, GPS receivers, and the like, have been integrated on such devices, together with a variety of wireless network technologies of increasing bandwidth (e.g., Bluetooth 2, Zigbee, WiFi and WiMax), enabling the on-the-fly creation of networks of devices in proximity. In parallel, the Internet has seen a proliferation of blogs and personal content spaces, revealing a transformation of users from traditional consumers to active producers of content, thus becoming shapers of what the Web has to provide.

It will not be long before these two trends will converge, thus creating an integrated environment where, besides traditional services delivered by highly powerful server machines accessible via wide area networks (Figure 1 (a)), new services and content will be offered by users to users while on the go, via their portable devices (Figure 1 (b)). A user may, for example, access an information service made available locally within a building, she may use the navigation system of another user in reach, in exchange for her higher bandwidth Internet connectivity, and so on. These fine grained services, attached to people and the environment, will need to be composed to deliver more sophisticated functionalities to the end user.

In order to give users a positive mobile experience, such composite services will have to be perceived by the user as supplied by a unique entity, that is reachable and available for the duration of the service, despite the fact that services are actually mobile to each other. We propose to achieve this goal by means of a *service composer*, an entity that reasons about the mobility of service providers and consumer, in order to estimate the overall *reliability* of the composition. Only compositions that are considered reliable will then be executed, in order to minimise the number of failed attempts.

We first describe a scenario that exemplifies a variety of



(a) Today **(b) Tomorrow**

Figure 1. Paradigm Shift in Service Delivery

mobile service compositions (Section 2). We then present our *service composition model* (Section 3), which consists of two key components: a *mobility predictor component*, estimating the duration of colocation between component services, and a *semantic reasoner component*, that uses these colocation estimates to select those services, within the current environment, that provide the highest reliability to the composition. We demonstrate the positive impact that the mobility reasoning has onto the ratio of successfully completed compound services, in a typical human movement scenario (Section 4). We compare this work with ongoing research in this domain (Section 5), before concluding and discussing out future directions of research (Section 6).

2 Scenario

Let us consider a user Alice, who owns a next generation mobile phone, whose basic and enriched functionalities (e.g., phone calls, messaging, navigation, etc.) are delivered by smartly composing the services available at a given instant.

Alice has installed on her phone the Smart Media Player application, an application streaming music and video for free from other devices, mocking the functionalities of radio and TV channels. Advertisements are injected from time to time, either interrupting the music/video streaming, or by means of interactive banners. The content to be played, as well as the adverts to be shown/reproduced, are selected based on what is currently available in the environment, taking into consideration Alice's profile. For example, adverts can be gathered from Internet services, when connectivity is available, as well as local advert broadcasters.

Alice leaves her office and walks to the nearest tube station. She is listening to some music played by her Smart Media Player. As she gets on the tube, she looses global connectivity, so the Smart Media Player application has to elect new service providers for music content. In order to do so, it first has to fetch Alice's profile, and to examine what content and content sources are now available in the new environment. On the basis of those, the Smart Media Player elects the items to be played next, and streams them.

This simple scenario describes a variety of mobile services and introduces a variety of composition semantics. For example, the media content selector and the advertising service both require to collect Alice's profile and context first; as such they need to be composed *sequentially* (in sequence) to an eventual context-aware user profiling service:

$$profilingService\ \textbf{seq}\ content\&advertService$$

Depending on the actual context and user preferences, advertising may be shown or played, either *in parallel* to the content selection and reproduction service, or *subsequent* to it. In order to enable the selection of one or the other strategy, a *choice* composition semantics will be needed. The compound service $content\&advertService$ can then be broken into:

```
choice $< guard\ condition >$ (
    $(contentSelector$ seq $advertisingService)$,
    $(contentSelector$ parallel $advertisingService)$
)
```

The Smart Media Player updates the list of the next-to-come songs or videos at a regular basis, so that the overall composition *loop* is started again. The full composition semantics of the Smart Media Player can thus be described as follow:

```
loop $< iterator, guard\ condition >$ (
  $profilingService$ seq (
    choice $< guard\ condition >$ (
      $(contentSelector$ seq $advertisingService)$,
      $(contentSelector$ parallel $advertisingService)$
    )
  )
)
```

As the above scenario shows, pervasive services (e.g., Smart Media Player) are often compound services, provided by aggregating more basic functionalities according to a variety of semantics (e.g., sequential, parallel, choice, loop, etc.). Some of these services will be local to the client's device (i.e., the device who is consuming the compound service), while others will be available from a combination of stationary providers (e.g., those embedded in the local space) and mobile providers (e.g., those provided by other people personal devices). Given the dynamicity of the target scenario, with services appearing and disappearing all the time as perceived by the client's device, it becomes crucial to: (1) reason about the providers' movement relative to the client's device; (2) select those providers that will maximise the chances of a successfully-completed compound service. In the next section, we present a service composition framework that supports this type of run-time reasoning and selection.

3 Service Composition Framework

In order to enable mobile users to seamlessly and successfully consume compound services while on the move, they must be given the impression of being interacting with a monolithic, local service. We propose to do so by means of a run-time service composition framework that takes care of finding, composing, and coordinating those services needed by the composition. Such services can be hosted by a mixture of stationary and mobile devices. The exact topology of services making up a composition varies depending on the current environment and is completely hidden to both application engineers and end users. We first provide an overview of the composition framework as a whole (Section 3.1), before focusing on the main contribution of this paper, that is, a service discoverer component (Section 3.2) that selects those services, among those currently available, that maximise the reliability of a composition. It does so by reasoning about the mobility of individual services (Section 3.2.1) with respect to the client's device, and the semantics of the composition (Section 3.2.2).

3.1 Overview

An overview of our composition framework is provided in Figure 2. As shown, it consists of four main modules: the *Service Manager* component is the access point to the composition framework. It provides an interface to request services, leaving the invoking application unaware of whether the requested service is single or compound, and whether it resides on the same device or it is accessible from other devices in the proximity (in particular, at a single hop distance).

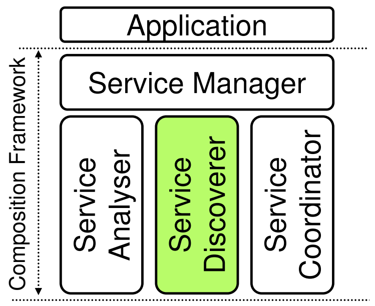


Figure 2. Composition Framework

Upon receiving a request for service s, the Service Manager invokes the *Service Analyser*, whose goal is to 'understand' the request: more precisely, the analyser decomposes s into component services $s_1, s_2, \ldots, s_n$, and returns a composition semantics for them (e.g., s_1 `seq` s_2 `seq` $\ldots$ `seq` s_n). Different decompositions are possible; the analyser returns the one that relies on the minimum number of components, and alternative decompositions are tried only if previous ones were not successful (i.e., providers were not found in the environment). In this paper, we assume a pre-defined taxonomy exists to map a requested service s to a decomposition $s_1, \ldots, s_n$ with associated semantics (e.g., [13]); this taxonomy could be either universal (and built, for example, on OWL-S) or can be specific to a domain. The analyser is also responsible for annotating the decomposition with 'aspects', that is, entry gates in the execution flow where a re-assessment of the current environment should take place (with potential re-biding of services - see below) before the service execution proceeds. For example, a possible aspect within the Smart Media Player compound service described in Section 2 is the entry to the loop.

The Service Manager then passes the annotated decomposition to the *Service Discoverer* component, whose main goal is to choose providers $p_1, \ldots, p_m$, among those available in the current environment, that will satisfy the request (i.e., that will be able to deliver services $s_1, \ldots, s_n$) and that will maximise the chances of successful service completion. Note that the binding between services and providers is 'consolidated' at each entry gate in the decomposition, and it is expected to remain stable only until the next entry gate in the flow: with reference to our example, whenever a new loop is about to start, providers of media content and of the adverts will be chosen by the discoverer, and they will be expected to remain available until the next aspect, that is, the next loop iteration.

Finally, the Service Manager passes the service decomposition and the selected providers to the *Service Coordinator* who is then in charge of executing the request. If, at execution time, one or more providers become no longer available, the Service Coordinator notifies the Service Man-

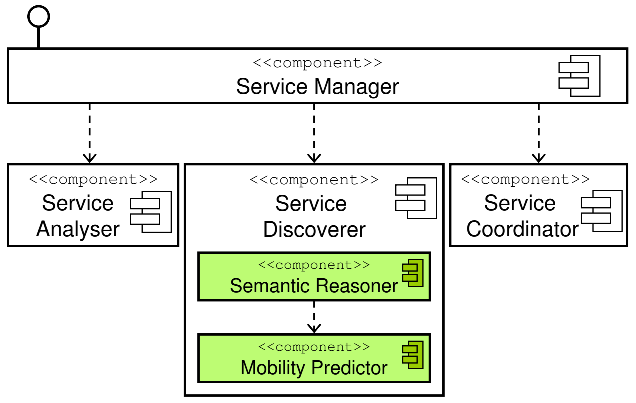


Figure 3. Service Discoverer Component

ager that, depending on the annotated decomposition, determines if the overall service can still be carried on (e.g., within a given time limit) or if a failure must be reported.

The reminder of the paper is devoted to describing the reasoning performed by the Service Discoverer component (Section 3.2), and to demonstrate experimentally how such reasoning maximises compound service completion success rate (Section 4).

3.2 Service Discoverer

The Service Discoverer module is responsible for the selection of the providers (or *services instances*), within the current environment, that will be relied upon to carry out the composition. Its goal is to determine a binding that will maximise the probability of compound service completion. In this paper, we do not consider QoS reasoning: while important, we believe that the ability to identify providers that will remain available for the duration of the service comes first, with QoS to be leveraged on top.

As illustrated in Figure 3, the Service Discover uses two main components to elect the service instances for a composition: the *Mobility Predictor* and the *Semantic Reasoner*. The former estimates for how long a given provider will remain colocated with the client's device (where the composition framework is running); the latter then uses these predictions, together with the specific composition semantics, to determine if a composition can be attempted and, if so, what instances to rely upon.

3.2.1 Mobility Predictor

The main goal of the Mobility Predictor is to estimate for how long a provider will remain directly connected (i.e., within single hop distance) to the Service Composer, and thus will be available to participate in the composition (i.e., we make the assumption that if a service provider is busy serving others client, it will not send a beacon to notify its presence).

The basic observation underpinning this work is that people show a high degree of regularity in their activities, often traveling to/from work on the same train, following routines during their working days, visiting the same pub or restaurant, and so on. Although the number of unknown devices we encounter will always be high, a non-negligible set of devices, either stationary or mobile, will be re-encountered regularly [8]. Based on this assumption, we have defined a simple yet effective prediction mechanism that aims at learning human behavioural patterns from past activities: for every day of the week d, and for every hour h within a day, a device i logs the duration of its encounters with any another device j. We use the symbol $\delta_{i,j}(d,h)$ to refer the historical colocations between devices i and j in the specific time slot (d,h). Recording colocation statistics incurs a small amount of storage and processor usage; this amount scales linearly with each additional host that is tracked. To reduce this overhead, only records about *familiar strangers*, that is, hosts we have been encountering with at least a certain frequency, can be kept.

Given two service instances i and j, and the current time t which falls in slot (d,h), the predicted duration of colocation is then computed as follows:

$$(1)\quad \sigma_{i,j}(t) = \begin{cases} 0 \\ \text{if avg}(\delta_{i,j}(d,h)) \leq \alpha\cdot \text{ stddev}(\delta_{i,j}(d,h)) \\ \\ \text{avg}(\delta_{i,j}(d,h)) - \alpha\cdot \text{ stddev } (\delta_{i,j}(d,h)) \\ \text{otherwise} \end{cases}$$

Intuitively, the higher the variation in past colocation durations, the lower the estimate made. In general, we prefer to underestimate the duration of colocation, rather than overestimating it, to later minimise the number of initiated but then failed compositions; we thus always set parameter α to be a positive constant value, whose impact on service completion rate will be analysed experimentally in Section 4.

3.2.2 Semantic Reasoner

In order to deliver a compound service s, the Service Discoverer module could use the colocation predictions computed by the Mobility Predictor in order to decide what service providers to bind to within the current environment, in order to have high probability that the composition will successfully complete (i.e., that no component will become unreachable during service delivery). For example, if n component services are needed to deliver s, and if it is estimated (e.g., from past experiences, from Service Descriptors, etc.) that it takes up to Δt seconds for s to complete, then the n instances $p_1, \ldots, p_n$ would be chosen so that each of them is estimated to remain colocated with the client's device at least Δt from the time the service request begins. In other words, we could require each component service to remain available for the whole duration of the composition.

This requirement may become quite stringent in highly dynamic environments, especially for compositions requiring the participation of many components and/or for long periods of time, resulting in many service deliveries not even being attempted. However, not all services are indeed needed for the whole duration of the composition. For example, if two services s_1 and s_2 are sequentially composed (i.e., s_1 `seq` s_2), and it is estimated that each will take 30 seconds, then s_2 will be needed for the whole duration of the composition (i.e., 60 seconds) while s_1 will be needed only for the first 30 seconds. The semantics of the composition quite precisely identify *when* and *for how long* a specific service instance is going to be needed. Based on colocation predictions, the *Semantic Reasoner* component thus leverages the specific composition semantics in use to set *minimum colocation requirements* for each service instance s_i individually.

For example, given n services composed sequentially s_1 `seq` $\ldots$ `seq` s_n, and assuming each service s_i takes Δt_i seconds to execute, then the minimum colocation requirement for any provider of s_i is $\Delta t_i^* = \Delta t_i + \sum_{j=0}^{j<i} \Delta t_j + \Delta t_{j,j+1}$, where $\Delta t_{j,j+1}$ is the maximum tolerated interval of time between the completion of s_j and the launch of s_{j+1}. If services are composed in parallel, then their minimum colocation requirement is $\Delta t_i^* = \Delta t_i$ instead. The complete set of composition semantics, and relative colocation requirements, can be found in a separate technical report [7].

The Semantic Reasoner on client device c decides whether to launch a composition, and (if so) on what instances to rely on, using Algorithm 1. To begin with, providers of services needed in the composition are found in the environment (step 1). A prediction of the remaining colocation between each of these providers and the client's device is computed, and only those providers who are expected to remain available for the minimum colocation requirement are kept (step 2). If more than one provider is available for a given component s_i, then the one with the longest predicted colocation time will be selected (step 3). If, at the end of the process, set P contains one provider for each component $s_i \in s$, then the composition is attempted.

Note that the minimum colocation requirement could be set even looser than what we have described thus far. In Algorithm 1, we require providers of all component services to be available at the beginning of a composition, for the service to be started; however, depending on the composition semantics (e.g., sequential), some services may only be needed at a later stage. A composition could thus be started even if instance s_i is currently not available, provided that there is a high probability that s_i will become

Algorithm 1 Service Components Selection Algorithm

```
Input Parameters
    - semantics de-composition of service $s$ into $\{s_1, \ldots, s_n\}$;
    - $\Delta t_i$ the maximum time to execute each component service $s_i$ (i.e., obtained from monitoring past experiences, from Service Descriptors, etc.);
    - $\Delta t_i^*$ the interval of time $s_i$ is requested to be available for, as estimated based on the composition semantics;
    - $\Delta t_{i,j}$ the maximum tolerated interval of time between the completion of $s_i$ and the launch of $s_j$.
Returns: set $P$ of service instances $\{p_1, \ldots, p_n\}$ to bind to, to deliver the composite service $s$.
$P = \emptyset$
{Step (1) - Functional Matching}
$F = \emptyset$
for all $p_k$ in the environment do
  if $s_k$ delivered by $p_k$ belongs to $s$ then
    $F = F \cup \{p_k\}$
  end if
end for
{Step (2) - Stability Filtering}
$S = \emptyset$
for all $s_i \in s$ do
  if $\exists p_k \in F \mid (s_k \equiv s_i) \wedge (\sigma_{c,k} \geq \Delta t_k^*)$ then
    $S = S \cup \{p_k\}$
  end if
end for
{Step (3) - Stability Maximisation}
for all $s_i \in s$ do
  $p_k = max_{\sigma c,j}\{p_j | p_j$ provides service $s_i \in S\}$
  $P = P \cup \{p_k\}$
end for
return $P$
```

available by the time it is needed. Such probability could be computed based on: the maximum waiting time by which s_i will be needed (e.g., for services composed sequentially, the *time before* s_i is needed is $\sum_{j=0}^{j<i} \Delta t_j + \Delta t_{j,j+1}$); and historical/contextual information about what services where available at a given place and time in the past. If historical information were not available, an estimate could be obtained by looking at the dynamicity of the mobile environment (the average growth per second of the number of services available, as perceived by the service composer), and the homogeneity of the available services. We have not yet experimentally evaluated this looser version of the selection algorithm, and we thus leave its details outside the

scope of this paper.

In the next section, we experimentally evaluate the accuracy of our prediction technique, and demonstrate the gain obtained in terms of successfully complete composite services in realistic mobile environments.

4 Evaluation

4.1 The Dataset

In order to evaluate our service composition model in a realistic pervasive environment, we have used the connection logs from real life Bluetooth devices, available from the CRAWDAD [1] resource archive. In particular, we have elected the MIT Reality Mining dataset [2] as our reference scenario. This dataset contains the Bluetooth id of devices in proximity, as detected by one hundred Nokia 6600 phones, which were given to MIT staff and students, over a period of nine months. Each phone had been configured to scan the environment and log colocated devices (i.e., within one hop distance). The final dataset contains in excess of 500,000 hours of data about human activity.

The reasons we focus on this dataset are many: to begin with, unlike many real data sets, this one is big enough to enable accurate evaluation of our model, without having to rely on synthetic mobility traces (e.g., random mobility). Second, it has been collected in what we consider a typical setting for the consumption of composite services (i.e., a university campus, with some services being available centrally, others from devices embedded in buildings, and others still from peer Bluetooth devices). Finally, the dataset spans a long enough period of time for temporal behavioral patters to emerge, and for our model to learn and exploit them.

4.2 Simulation Setup

We have conducted two main sets of experiments, both on top of the previously described dataset: the first set aimed to assess the accuracy of the Mobility Predictor component, while the second set aimed to quantify the gain obtained by the Semantic Reasoner component, in terms of successfully completed compositions.

Mobility Predictor - In order to assess the accuracy of the Mobility Predictor component, independent of any service composition, we have conducted the following experiment. Whenever two devices i and j entered within connectivity range, and thus an event was logged within the MIT Reality Mining dataset, we have predicted $\sigma_{i,j}(t)$, that is, for how long device i expects to remain colocated with device j, based on the colocation statistics observed up until time t (see Formula (1) - Section 3.2.1). We have then compared our estimated colocation duration with the actual one and, in case our estimate was lower or equal to the actual one, we recorded a success, otherwise we recorded a failure. Note that, in our framework, it is important not to overestimate colocation durations, to minimise the risk of started but uncompleted compositions due to service/device departure. Nonetheless, an overly conservative model that excessively underestimate colocations would inhibit compositions that would otherwise been successful. For each successful (under)estimation, we have thus quantified the actual percentage of underestimation as:

$$(2) \quad \frac{\text{actual colocation time} - \text{estimated colocation time}}{\text{actual colocation time}}$$

The experiments have been executed for different values of α (see Formula (1) - Section 3.2.1), and the results will be shown based on different minimum *familiarity levels*. Here, by familiarity level of a device j for weekday d and hour h, we intend the percentage of occurrences that the observing (client) device i was connected to j on d days and at h hour, among the number of co-occurrences between i and j, as observed prior to the current time. Our expectation is that, the more familiar a device is, the more accurate the prediction becomes. Results about this set of experiments are reported in Section 4.3.1.

Semantic Reasoner - In order to evaluate the Semantic Reasoner component, we have uniformly associated, to each device within the MIT Reality Mining dataset, a service of type $(id \bmod n)$, where n is a parameter indicating the number of services being composed in a given experiment; we have then replayed the connectivity traces. In all experiments, we have made the assumption that each device provides exactly one type of service, and that each service composition requires n distinct types of services. A first set of experiments only considered situations where at least $n \times 3$ devices were colocated, in order to assess the quality of the selection made by the Semantic Reasoner when faced with choice; a second set of experiments then followed, where all situations with at least n colocated devices were considered for composition. In both sets of simulations, we have recorded the percentage of successfully completed compositions, out of all those started, when using the Semantic Reasoner on top of the Mobility Predictor, and compared the results against a random selection of providers among those currently in reach. Similarly to what done for the Mobility Predictor, we have also quantified the number of compositions that we do not start but that would have been successfully completed. The experiments have been conducted on different values of Δt_i (i.e., average amount of time to execute a single service) and n (i.e., number of services within a composition), and considering all services as if they were composed in *sequence*. The obtained results are reported in Section 4.3.2.

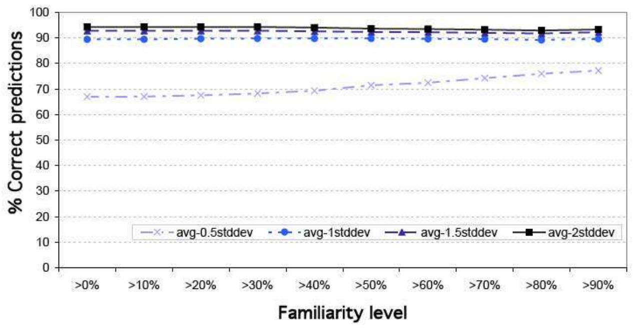


Figure 4. Mobility Predictor - Correct Predictions

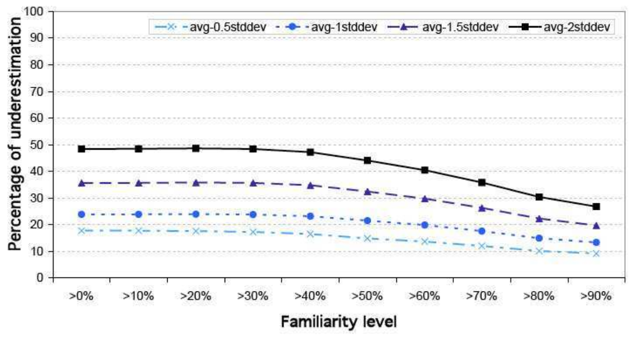


Figure 5. Mobility Predictor - Underestimation

4.3 Results

4.3.1 Mobility Predictor

Figure 4 illustrates the percentage of correct predictions (i.e., predictions which are equal or lower to the actual colocation time) made by the Mobility Predictor, for different values of α, and broken down for different levels of familiarity between each pair of devices. As expected, the higher the value of α, the more conservative the model becomes, with the percentage of correct predictions reaching 95% for $\alpha = 2$. However, the price to pay is loss of accuracy. Figure 5 illustrates the amount of underestimation, computed as per formula (2), for the same values of α: as shown, the prediction can miss up to 50% of the actual colocation duration when $\alpha = 2$. A good balance between correct predictions and underestimation can be achieved for $\alpha = 1$, where the former reaches roughly 90%, while the latter varies between 22% for strangers, down to 12% for familiar devices. In the following experiments, we have thus set the value of α to 1.

It is interesting to note the impact of familiarity level on the behaviour of the Mobility Predictor: familiarity plays little importance in the percentage of correct predictions (Figure 4), unless the standard deviation is only marginally considered ($\alpha = 0.5$), thus revealing a predictor model that tends toward the cautious side; however, it does play an important part in reducing the amount of underestimation, which neatly decreases the more frequently the pair of devices have met (Figure 5).

4.3.2 Semantic Reasoner

The first set of experiments we conducted on the Semantic Reasoner aimed to assess its reliability (in terms of percentage of successfully completed compositions), and compare it with an approach that selects service instances at random, among those available in the environment. To explicitly assess the reliability of the *choices* made using the two different approaches, we have considered environments where $n \times 3$ devices are colocated, while varying the number n of services that make up a composition, and the duration of a component service Δt_i (maximum values of n and Δt_i where chosen so to test even the limit cases in the available dataset). The results are reported in Figure 6.

As shown, the reliability of the composition performed by the Semantic Reasoner is consistently very high, in excess of 97% succesfull completion rate, regardless of the actual duration of the compound service. In fact, it remains almost constant while varying the number n of services being composed, and the time Δt_i required to execute each of them. On the contrary, the reliability of randomly made compositions is highly susceptible to increases of n and/or Δt_i: the achieved reliability is high only when just a couple of services that run for less than 10 seconds each are considered, as device mobility plays a smaller role in these situations; however, for compositions consisting of 3 services or more, and for service instances running for 20 seconds or longer each, the rate of successfully executed random compositions dramatically drops, with a decrease of up to 35% with respect to the Semantic Reasoner achieved reliability.

Under the same experiment setup, we have also quantified the number of compositions that the Semantic Reasoner chose not to start (as they were considered unlikely to complete), but that would have been successfully completed instead. Figure 7 illustrates the results. As shown, at most 8% of the compositions were mistakingly not started, and these refer to the more risky cases where long running compositions were considered (i.e., Δt_i in the order of minutes). Note also that the Semantic Reasoner does not attempt any composition involving 5 devices or more (no cases of successful compositions were reported in Figure 6 for $n = 5$), and it correctly does so, as the error (miss) rate is exactly 0% in this case. These results demonstrate that the Semantic Reasoner achieves reliability without being too conser-

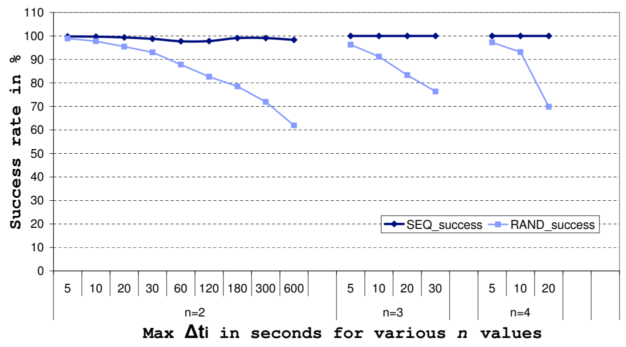


Figure 6. Semantic Reasoner - Successfully Completed Compositions

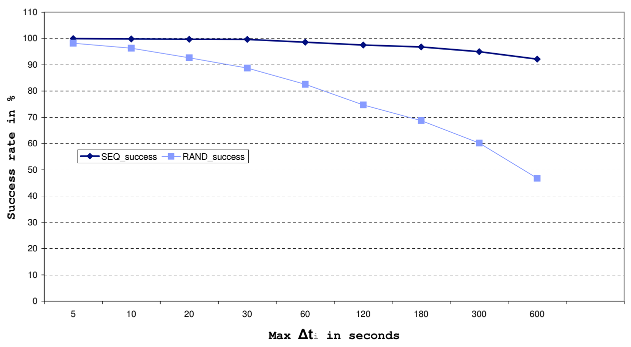


Figure 8. Semantic Reasoner - Success with respect to Δt_i, for $n = 3$

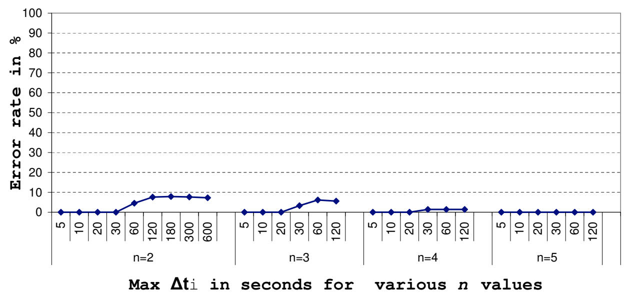


Figure 7. Semantic Reasoner - Missed Opportunities

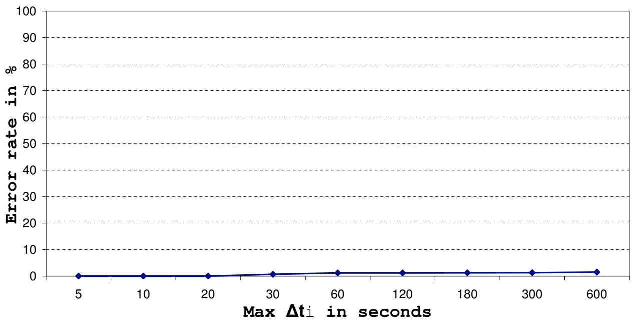


Figure 9. Semantic Reasoner - Misses with respect to Δt_i, for $n = 3$

vative in initiating service compositions.

We have then conducted a second set of experiments, to test the sensitivity of the Semantic Reasoner with respect to Δt_i and to n individually. We have first set $n = 3$ and increased Δt_i up to 600 seconds (Figure 8 and 9), and then set $\Delta t_i = 20''$ and increased n up to 5 (Figure 10 and 11). Once again, the maximal values of Δt_i and n were chosen based on limit situations recorded in the Reality Mining dataset. To increase the number of test cases, we have removed the constraint of having at least $n \times 3$ devices connected, thus considering all situations with at least n colocated devices (for compositions of n services).

The successful completion rate of the Semantic Reasoner remains remarkably high even for long compositions, while a random compositions fails up to 50% of the attempted ones, as Δt_i increases (Figure 8). This confirms the importance of reasoning about device mobility and composition semantics when managing service instances whose running time is $20''$ or above. Figure 9 confirms the observation that reliability does not come at the price of missed opportunities, as the percentage of unattempted but would-have-been successful compositions is constantly well below 5%.

The last set of experiments confirms the above observations: increasing the number of composed services n, when fixing $\Delta t_i = 20''$, has a negative impact on the reliability of random compositions, while only a marginal decrease is observed for compositions performed by the Semantic Reasoner (Figure 10). The number of unattempted compositions is also marginal (below 0.4%), and it becomes void when four or more services are being composed (Figure 11).

Based on the results presented in this section, we can thus confirm that our model enables the reliable composition of services in realistic human movement environments. The achieved reliability is only marginally dependent on the duration of the compound service, and it does not come at the expense of missed opportunities. This is in sharp contrast to the performance achieved by a random selection of service instances, where the successful completion rate starts to dramatically drop as soon as service duration exceeds $\Delta t_i = 20''$.

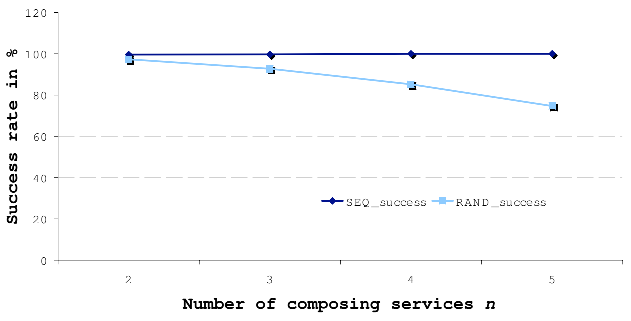


Figure 10. Semantic Reasoner - Success with respect to n, for $\Delta t_i = 20''$

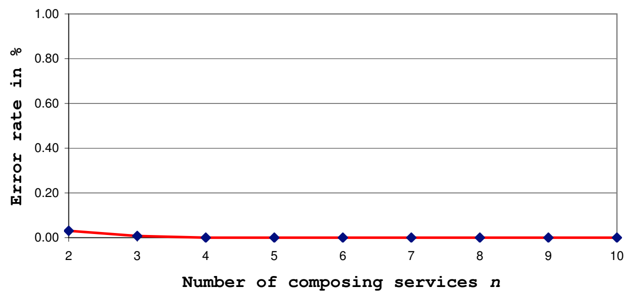


Figure 11. Semantic Reasoner - Misses with respect to n, for $\Delta t_i = 20''$

5 Related Work

Service composition [6] has attracted a lot of interest since the advent of Web Services, and it has become a workable and broadly adopted technology thanks to real or defacto standards such as WSDL [19], SOAP [21] and UDDI [17]. This attention has mainly concerned the Internet and hence wired-environments, where the service providers are static and well known.

In this domain, research has followed two main directions: one stream of research has focused on developing languages that aimed at adding semantic information (e.g., WSDL-S [20], OWL-S [14], METEOR-S [15]) and/or protocol information (BPEL4WS [3], WSCDL [23], METEOR-S [15], OWL-S [14], YAWL [18]) to service descriptions. A second stream of research has then focused on methodologies to aggregate these services, which can be broadly classified into: manual, semi-automatic and automatic [4]. Manual service composition entails the requester to browse a registry of services, find the desired service operations, and model their interactions into a flow structure (mainly with BPEL); the final service is then exposed as a unique service using WSDL. This methodology is effectively employed today within the Web Service Industry. Semiautomatic composition of services usually involves a service composition system that interacts with the requester in an iterative manner in order to obtain information about the requested service, and to construct aggregate services out of the registered ones. The automatic composition, instead, demands the existence of a discovery agent that receives a service request and generates a structure of services/operations of some pre-registered services based on the information provided in the request. These automatic aggregation approaches rely on services to be richly described by means of the previously cited Semantic Web Service languages.

Service composition in pervasive environments requires this kind of automatic (de)composition. However, the very nature of the environment opens the door to new challenges that limit the applicability of current technologies. For example, resource limitations on the target devices have called for novel solutions to enable efficient ontology-based semantic matching of services [13]; more flexible approaches enabling the on-demand creation of an agreed ontology are being investigated too [22]. As stated in Section 3, our work is orthogonal to the issue of semantic service matching, and we thus leverage on top of existing solutions. More closely related to our work are approaches that take into consideration mobility of devices, and thus services. In [5, 11] (single) service discovery protocols have been proposed that aim to find the device capable of delivering the best quality of service, given the dynamicity of the current environment; the discovery protocol has also been extended to consider multi-hop networks [16]. We argue that, while important, QoS reasoning must come *after* colocation reasoning, especially for services that require more than just a few seconds to complete. For the same reason, multi-hop service discovery and delivery is promising only for services that execute very fast, and/or are deployed in fairly stable scenarios. We first explored colocation reasoning in [12], where device mobility was considered when choosing from what (single) device to download some content; this paper extends our prior work by enabling the mobility-aware discovery of sets of component services, and by reasoning about the composition semantics to deliver a reliable service composition experience to the end user.

Research on service composition in mobile environments is still in the very early days, and other issues, which were just marginal for infrastructure-based environments, must now be looked into, including session management [9, 10], on-the-fly adaptation, on-demand composition, dynamic execution monitoring, failure recovery mechanisms, and so on, just to name a few.

6 Conclusion

In this paper, we have proposed a model and framework that increase the reliability of service composition in mobile environments. We do so by means of a prediction model, that is capable of estimating the duration of colocations between component services based on past encounters, and a semantic reasoner that exploits the predicted colocations and the composition semantics to choose the component services to rely upon to maximise the chances of successful completion of the compound service. We have demonstrated that, in a large scale human movement scenario, the percentage of successfully completed composite services is much higher than when selecting services without mobility and composition semantics in mind.

Our plans for the future spans three different directions. In the short run, we intend to run experiments on a broader set of human movement traces, and to quantify the gain obtained for each composition semantics our framework supports (i.e., parallel, loop, choice, etc.). In the medium run, we intend to investigate self-healing algorithms to react to changes in the environment, thus enabling the smooth adaptation of the composite service in between entry gates of its execution flow, as anticipated in Section 3. Preliminary results suggest that monitoring signal strength (and its sudden variation) can provide valuable information about when run-time adaptation should be carried out, in response to, for example, the unforeseen departure of a service provider [7]. Ultimately, our goal is to deliver a lightweight framework that will realise these techniques and that will run seamlessly on mobile devices, thus becoming an enabling technology for the realisation of pervasive computing. Note that we are focusing on real-time data and services that cannot be pre-fetched; caching techniques, as well as server-side composition, may be preferred and incorporated in our approach to increase the reliability of the composition, in the presence of good connectivity to stable providers.

References

[1] Community Resource for Archiving Wireless Data At Dartmouth. http://crawdad.cs.dartmouth.edu/.

[2] The Reality Mining Project. http://reality.media.mit.edu/.

[3] BPEL4WS Coalition. Business Process Execution Language for Web Services (BPEL4WS) Version 1.1. http://download.boulder.ibm.com/ibmdl/pub/software/dw/specs/ws-bpel/ws-bpel.pdf, May 2003.

[4] A. Brogi and R. Popescu. Contract-based Service Aggregation. Technical Report TR-06-12, Dept. of Computer Science, University of Pisa, July 2006.

[5] L. Capra, S. Zachariadis, and C. Mascolo. Q-CAD: QoS and Context Aware Discovery Framework for Adaptive Mobile Systems. In *IEEE International Conference on Pervasive Services*, Santorini, Greece, July 2005.

[6] D. Chakraborty, A. Joshi, Y. Yesha, and T. Finin. Service Composition for Mobile Environments. *Journal on Mobile Networking and Applications, Special issue on Mobile Services*, 2004.

[7] L. del Prete. Dynamic Service Composition in Mobile Environments. Technical Report RN/08/04, Dept. of Computer Science, University College London, 2007.

[8] N. Eagle and A. Pentland. Reality Mining: Sensing Complex Social Systems. *Personal and Ubiquitous Computing*, 10(4), 2006.

[9] C. Julien. Adaptive Preference Specification for Application Sessions. In *Proceedings of the 4th International of Service-Oriented Computing*, pages 78–89, Chicago, IL, USA, December 2006.

[10] C. Julien and D. Stovall. Enabling Ubiquitous Coordination Using Application Sessions. In *Proceedings of the 8th International Conference on Coordination Models and Languages*, pages 130–144, Bologna, Italy, June 2006.

[11] J. Liu and V. Issarny. QoS-Aware Service Location in Mobile Ad-Hoc Networks. In *Proceedings of the IEEE International Conference on Mobile Data Management*, pages 224–235, 2004.

[12] L. McNamara, C. Mascolo, and L. Capra. Content Source Selection in Bluetooth Networks. In *International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services*, Philadelphia, USA, 2007.

[13] S. B. Mokhtar, A. Kaul, N. Georgantas, and V. Issarny. Efficient Semantic Service Discovery in Pervasive Computing Environments. In *Proceedings of the ACM/IFIP/USENIX 7th International Middleware Conference*, Melbourne, December 2006.

[14] OWL-S Coalition. OWL-S: Semantic Markup for Web Services Version 1.1. http://www.daml.org/services/owl-s/1.1/overview/.

[15] A. Patil, S. Oundhakar, A. Sheth, and K. Verma. METEOR-S Web service Annotation Framework. In *Proceedings of the Thirteenth International World Wide Web Conference*, 2004.

[16] F. Sailhan and V. Issarny. Scalable Service Discovery in MANET, booktitle = Proceedings of the 3rd IEEE International Conference on Pervasive Computing and Communications, year = 2005, pages = 235–244, address = Hawaii, USA, month = March.

[17] UDDI Coalition. The UDDI Technical White Paper. http://www.uddi.org/, September 2000.

[18] W. M. P. van der Aalst and A. H. M. ter Hofstede. YAWL: Yet Another Workflow Language. *Information Systems*, 30(4):245–275, 2005.

[19] W3C. Web Services Description Language (WSDL) 1.1. http://www.w3.org/TR/wsdl, March 2001.

[20] W3C. Web Service Semantics - WSDL-S. http://www.w3.org/Submission/WSDL-S/, November 2005.

[21] W3C. SOAP: Simple Object Access Protocol. http://www.w3.org/TR/SOAP/, April 2007.

[22] A. Williams, A. Padmanabhan, and M. Blake. Experimentation with Local Consensus Ontologies with Implications for Automated Service Composition. *IEEE Transactions on Knowledge and Data Engineering*, 17(7):969–981, July 2005.

[23] WSCDL Coalition. Web Services Choreography Description Language Version 1.0. http://www.w3.org/TR/ws-cdl-10/, November 2005.