Technische Universität Darmstadt

Department of Electrical Engineering and Information Technology
Department of Computer Science (Adjunct Professor)
Multimedia Communications Lab
Prof. Dr.-Ing. Ralf Steinmetz

The Virtual Assistant: Framework and Algorithms for User-Adaptive Communication Management

Technical Report

KOM-TR-2007-08

By

Johannes Schmitt, Matthias Kropff, Matthias Hollick, Ralf Steinmetz

Contact: [johannes.schmitt;matthias.kropff;matthias.hollick]@kom.tu-darmstadt.de
http://www.kom.tu-darmstadt.de

First published: December 12, 2007

For the most recent version of this report see
ftp://ftp.kom.tu-darmstadt.de/pub/TR/KOM-TR-2007-08.pdf

The Virtual Assistant: Framework and Algorithms for User-Adaptive Communication Management

Johannes Schmitt, Matthias Kropff, Matthias Hollick, Ralf Steinmetz

Today, a plethora of communication technologies promises ubiquitous connectivity. Aspects such as communication management and service control are crucial for efficient and productive usage of the existing communication services. However, the burden of complex configuration and parameterization of these aspects is typically left to the end user. We introduce the virtual assistant, which aids the end user to manage his diverse communication needs in a technological environment characterized by heterogeneity and dynamics. We thoroughly discuss the requirements for a virtual assistant that adapts to the communication behavior of the end user and propose a context-aware architecture to implement this service. While our architecture is generic, we here instantiate it for the special case of ubiquitous communications. Our architecture applies methods from the area of machine learning. This paper lays special emphasis on the selection of appropriate learning algorithms. As a proof-of-concept, our architecture has been implemented in a real world communication system.

I. Introduction

Existing communication systems offer a large number of services and functionalities to end users. However, communication services such as call forwarding or notification which are designed to assist the user in his daily work are rarely used, because services have to be maintained manually by the user. Managing communication services by rules or scripts is often complex and requires technical knowledge. Furthermore, static models like fixed rules have to be held up to date manually and increase the management overhead. As a consequence the added value of communication services is limited by the effort the user has to spend for their appropriate configuration.

At the same time the amount of digital devices, which can be used as information sources (sensors) in order to determine the user's actual context increases. Examples for information sources are calendars and address books on mobile phones, location from GPS systems, activity on a PC or other information from sensors in a user's environment. We envision that future communication systems will be able to support their users like a private virtual assistant. The virtual assistant might incorporate several information sources to handle a communication request in an optimal way. The virtual assistant generates a model for call management by observing the user's behavior. This model represents the user's communication demands depending on the actual context.

In this paper, we present a virtual assistant that provides a user adaptive communication management. This virtual assistant is designed to cope with changing environments and demands of the addressed user. The assistant provides mechanisms to improve the model generation by incorporating simple user feedback. Depending on the requirements, the virtual assistant may be implemented as service on a server or directly within the communication software framework.

A. Contribution and Outline

The paper discusses the general architecture for an *assistant service*, which provides the functionality and the necessary system components of the proposed virtual assistant. Figure 1 gives a graphical overview of the proposed architecture and provides references to the corresponding sections.

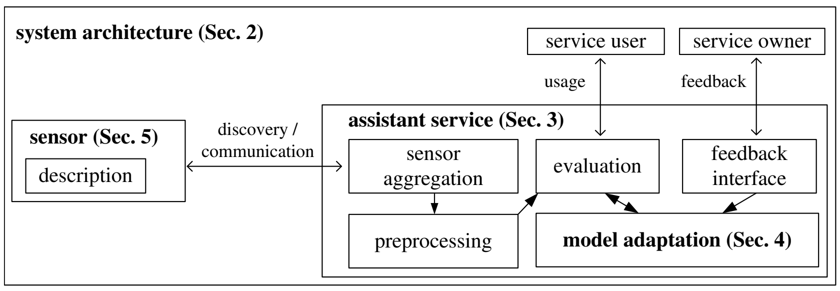

Fig. 1. Organization of the proposed architecture

Our contribution can be summarized as follows:

- We propose the novel idea of a virtual assistant, which autonomously generates and adapts a model of the user's communication behavior to automatically manage the users communication services.
- We outline the usage scenario for optimal communication management and discuss related work in this area in Section II.
- We design an appropriate system architecture for the virtual assistant based on the constraints and requirements for the selected scenario. This design is presented in Section III together with the correlated terminology of the system elements.
- We propose the architecture of our learning system as the central entity of the adaptive assistant in Section IV. This architecture comprises the processing phases of model evaluation and model adaptation as well as the description of the feedback interface.
- We analyze available algorithms for model building and address the specific challenges, benefits and novel demands of adaptive model generation within the assistant service. We give a comparison of the different classes of learning algorithm with focus on properties like robustness and efficiency in Section V.
- We conclude the paper with a short summary of the obtained results and highlight possible future fields of application for our virtual assistant in Section VI.

II. Usage Scenario and Background

A. Usage Scenario for Optimal Communication Management

Today users have to cope with an increasing number of new communication means, services and applications. However, the increasing number of communication means and the resulting continuous availability impose additional management effort to the user.

The attached social implications demand that a user manually turns off its mobile phone during an meeting. In another scenario person A would like to leave his office for a limited time but is urgently expecting a call from person B. This however requires that A sets and later resets the call forwarding functionality to his mobile phone. Further, when a person C wants to talk to person A, A might or might not want to talk to C at the given time. If person A is not available, person C might want to send A an email, an instant message, or leave a message on A's answering machine. Accordingly, A always has to update the configuration of his communication services to define which and when calls should be accepted, rejected or forwarded.

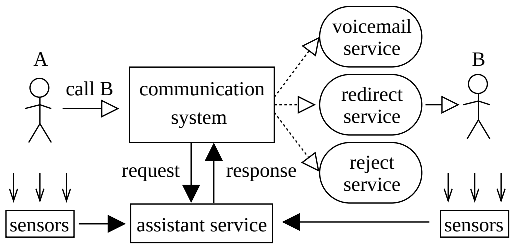

Fig. 2. Usage scenario – assistant service for VoIP call management

Users often want to use a communication service with a more detailed behavior, e.g., calls should be redirected to their office only during a certain time frame, whereas at home only private communication attempts are desired. The configuration of such detailed call setup management is usually done by using static models like rules or scripts, such as the Call Processing Language (CPL) [LS04].

We assume that our assistant service is able to automatically determine the demanded call handling method as shown in Figure 2. The assistant service selects the appropriate call handling method by using the context information of the communication partners and their environments.

B. User-Adaptive Communication Management

Our approach addresses an automatic generation of a model that adapts the user's demands. Benefits of automatic model generation are:

- Automatic model generation strongly reduces maintenance effort. The complexity of the model increases with the number of available information sources, the number of addressed communication services and the level of granularity of the behavior demanded by the user (e.g. the number of different contexts). Configuring the system manually by using scripts or rule based control mechanism is very challenging especially for technically unskilled users.
- A static configuration of a communication service has to be kept up to date. An inappropriate configuration would cause undesirable behavior, among others by wrong forwarded call attempts. Automatic model adaptation reduces maintenance effort as well.
- The more information sources related to the user are available the more precise the determination of optimal communication management can be. Using information sources from the user's environment requires the dynamic integration of dynamic and heterogeneous information sources into the

evaluation process. Our approach integrates new and unseen information sources dynamically into the model.

- The currently used information sources can be hidden from the user's perception. The user does not need to know which information sources are relevant and how they have to be processed.

User-adaptive communication management encapsulates two main tasks:

1) Process a communication request for the given situation in the most optimal way. This is accomplished by aggregating all available context information related to the communication attempt. The aggregated information is used as input for a model to determine the appropriate management decision (compare Figure 3).
2) Learning from user decisions. By observing the user's behavior or by receiving direct feedback the system adapts the model to the user's demands (compare Figure 4).

The addressed user-adaptive communication management is performed in an *assistant service* as described in more detail in Section IV.

C. Related Work

This work is novel in the sense that it combines several research topics, which to our knowledge have not been combined before. The first research area is a communication platform as target application area. The second research area is artificial intelligence and, in particular, its subfield supervised learning, which we are using as basis for the model generation. Third, context-awareness has become a hot topic in recent years because of the ongoing popularity that mobile communication devices enjoy.

Ferscha et al. [FMR03] take a similar approach: they use several simple sensors as a basis for suggesting a decision on the current context, with context names supplied a priori by the user. Their interpretation of the sensor concept is very close to ours. On the other hand, their work differs from the presented approach in some vital points. In contrast to our approach Ferscha et al. address mainly the *proactive context determination*, while we consider determining proper actions for the current situation. Proactivity focuses to forecast future events, but this aspect is negligible in our context.

Beigl et al. [BGS02] explore also the problem of automated call processing, but with a different focus. The authors provide examples for context classification on the basis of a sensor set and motivate the use of several heterogeneous sensors as well. They introduce the *Technology Enabling Awareness* (*TEA*) project [Lae99]. TEA includes a prototype mobile phone that can recognize contexts with a high probability, although this occasionally requires considerable computing time. The practical findings of the TEA project motivate a deeper analysis of machine learning algorithms in order to improve performance for model adaptation.

Microsoft Research proposes a similar approach within their *Notification Platform*[HK02]. The Notification Platform uses information from a user's PC to determine if a user shall be notified about an incoming e-mail or if the notification would disturb the user in his ongoing work flow. In contrast to our scenario, their approach is limited to a fixed set of input values and has fewer restrictions regarding the evaluation delay. Additionally the Notification Platform uses also an a fixed model, i.e., it does not face the challenge of handling changing user demands.

III. System Architecture

The following generic architecture of the assistant service can be used in various application scenarios. The only prerequisite is that there has to be a supervising entity [Wya97]. That means an entity (the person that uses the assistant service) has to train the assistant service with *correct* decisions. We aim at a smart communication system as application area for the assistant service. That implies a certain set of requirements. These requirements as well as the terminology of the system elements are described in this section.

A. Design Principles and Requirements

There are several technical and social requirements for the design of the targeted assistant service:

a) Minimum effort for the user: The goal of the integration of an assistant service is to make life easier for the user. A key requirement is that the service does not take much time from the user. The user shall neither be burdened with programming scripts of any kind nor with having to acquire any kind of technical knowledge about the system. In order to achieve this functionality the system should learn autonomously.

b) Supervision by the user: It is important that the user retains control over the behavior of the system.

c) Adaptivity: The system has to adapt dynamically and quickly to the changing preferences of the user.

d) Evaluation effort: In our application scenario the requests have to be processed in real time.

e) Management effort: The assistant service should be scalable and applicable for many users in parallel, i.e., the system requirements per user and per request should be minimal.

f) Generic architecture: The assistant service should be generic, i.e., it should be flexible to provide its functionality for other application areas (e.g., in smart home scenarios).

g) Dynamic set of information: Applicability in heterogeneous and dynamic environments. Hence parameters such as the number and type of information sources are open. Also newly available information sources have to be incorporated into the evaluation process.

h) Variable decision model: Also the applied learning algorithm should not be fixed in order to apply the one that is best suited for the desired task and the specific environment.

i) Human in the loop: The assistant service is controlled by human feedback. Regarding this the assistant service should be able to handle also a certain amount of mistakenly wrong given feedback messages.

B. System Elements and Terminology

We propose a system that provides user-adaptive communication management. In particular our architecture includes the following components: assistant service, service owner, service user, feedback mechanism, sensors, sensor-data, context information, samples, learning algorithms, and models. In this section we introduce the individual components of our architecture.

A *service* is a functionality that a system offers to its users. The *assistant service* is a special type of service that is able to decide, which services shall be applied and how a service has to be parameterized. In our framework the assistant service is set up by one entity (the *service owner*) and another entity (the *service user*) requesting or using it.

Within in the given communication management scenario from Section II-A the VoIP system is the service user. The VoIP system can request the actual state of the called person who is in this scenario the service owner. The task of the service owner is to inform the assistant service whether its decision was correct or not – and if not, what the correct decision would have been. This can be done only by the service owner using the *feedback mechanism*.

During a request the assistant service evaluates all available and relevant information regarding the context of the request. In the scenario described in Section II-A examples for information sources are call specific data like caller-ID, callee-ID, time, device types or the used codec. This set of information can be supplemented by information sources that are related to both conversation partners, e.g., an address book, a calendar, the activity or the location.

Each entity, which can be requested and which provides information about a person, a situation or the environment (e.g., physical sensors, databases or web services) can be used potentially as information source. Such physical or logical entity is called *sensor*. We use an abstract definition of sensor: *A sensor is any entity that can give information about the current situation.* The information that can be obtained from a sensor is named *sensor-data*. *Context information* describes the set of sensor-data, which was applied on a certain evaluation process, i.e., a fixed set of information that was related to a specific situation.

The assistant service provides the functionality to adapt the evaluation process to the service owner's demands. For that the assistant service saves the context information, which was collected during recently performed evaluation processes. Another input value is the reaction of the service owner, i.e., his *feedback*. The *knowledge* of the assistant service about the service owner's demands is described by tuple of context information $\{x\} = X$ with the corresponding correct decision $f(X)$ given by feedback messages. We name this tuple $(\{x\}, f(X))$ *sample* (which is equivalent to the meaning of *instances* in other works).

The function $f()$ which derives the correct decisions out of the set of sensor-data $\{x\}$ represents the demanded behavior, but it is only known by the service owner. By a *learning algorithm* we consider a method, which determines a function $g()$ which approximates $f()$.

A learning algorithm is able to generate such a function $g()$, the so called *model* or *evaluation model*. Usually a model is not a function in a mathematical notation. Common models are rather a sequence of processing instructions. There are many different types of models, but each with own properties. Often a learning algorithm is strongly coupled with the generated model. Details about the appropriate learning algorithms and models for our scenario are described in Section V.

IV. Adaptive Assistant Service

In the given communication management scenario as described in II-A the assistant service is requested during the call setup. The assistant service has the key role is the user-adaptive communication management. Its task is to decide, which communication service is appropriate for the given context and how it has to be parameterized. This section describes the basic principles and the architecture of the assistant service.

A. Basic Principles

The assistant service makes decisions by aggregation and evaluation of information from the environment. The evaluation is performed by applying the information on a decision model. However, common models for evaluation of information are based on static rules or scripts. The proposed assistant service adapts the model by observing the service owner's reaction and learns the dependency between the user's behavior and the current situation. The basic tasks of the proposed assistant service can be separated into two phases: The evaluation phase and the adaptation phase.

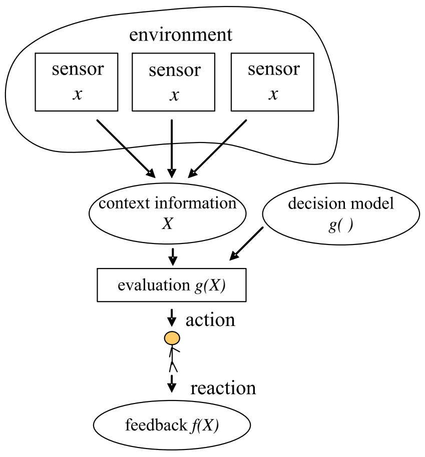

Fig. 3. Phase 1: Evaluation

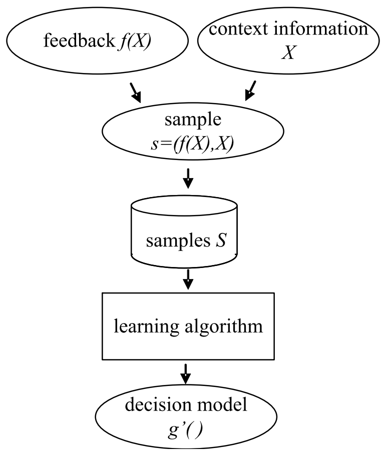

Fig. 4. Phase 2: Adaptation

1) Phase 1 - Evaluation.: This phase is used to perform decisions as shown in Figure 3. Decisions are performed by applying the information X on a model $g()$. This model $g()$ ideally approximates the demands of the service owner $f()$. The result of the evaluation can be used for controlling other systems, i.e., triggering of actions or parameterization of other services. The user's reaction on the chosen decision (his *feedback*) is equivalent to $f(X)$. If feedback on a performed decision is available, it can be used for adaptation of the model. In this case the adaptation-phase can be triggered.

2) Phase 2 - Adaptation.: If feedback on a recent evaluation process is available, adaptation of the model can be performed as shown in Figure 4. The feedback is related to a recent evaluation process respectively to the context information that was used for this evaluation process. The combination of this information with the feedback composes a sample. The knowledge of the system about the demands of the service-owner are reflected by the collected set of samples. The evaluation of large sets of samples during the evaluation phase has to be performed within a minimum amount of time according to the described requirements in Section III-A. Hence it is necessary to determine and extract the dependencies, which are described by the samples in advance. For that a learning algorithm processes those samples within this phase. The found and extracted dependencies are used as new model $g()$ in the following evaluation phases. The challenge is to generate a model $g()$, which ideally approximates the demands of the service owner $f()$. Ideally the function $g()$ not only fulfills $g(x_1, x_2, ..., x_n) = f(x_1, x_2, ..., x_n)$ in preferably all recently observed situations. The higher goal is to achieve that $g(x'_1, x'_2, ..., x'_n) = f(x'_1, x'_2, ..., x'_n)$ is true for a new and unknown situation X'. That means to deduce decisions also for sets of input values (x') for which the correct decision is not known a priori.

The combination of both phases composes a closed cycle. Beyond that the assistant service can be trained by the service owner by filling up the sample set. This principle makes it possible to control a system by the usage of feedback messages (as described in Section IV-B). Instead of the manual creation of the model $g()$, the system can be trained to adapt also a complex behavior.

B. Feedback as Interface for Supervision

A requirement is that the service owner should be able to retain control over the behavior of the assistant service. The service owner must have the ability to supervise the learning algorithm. The service owner can supervise the system by responding with *feedback*. If a recent decision of the assistant service was correct, the service owner may respond with a positive feedback and in case of a false decision, the service owner may give negative feedback. Depending on the applied learning algorithm and the usage scenario the feedback may contain also the corrected decision (e.g., the service owner responds with the expected result). If the demands of the service owner change, his decision will be different and he will give negative feedback. We call this case *concept drift* (this effect is discussed in Section V-C).

1) Feedback Methods.: Which feedback methods can be applied depends on the application scenario. Further it can be differentiated between two ways of using the feedback mechanism:

a) Implicit Feedback: occurs as direct observed reaction of the service owner on the decision made by the assistant service. This method is only applicable in scenarios where the service owner has an appropriate device for giving feedback. E.g., in the communication scenario the callee (who is in this case the service owner) may react directly on an incoming call by pressing *call rejection* on his phone. This observation can be applied as implicitly given negative feedback. Implicit feedback can be used to obtain simple feedback. Usually it can be used to determine whether the decision was correct and desired. In some other cases the service owner may choose one simple feedback out of a small set.

b) Explicit Feedback: is applied, when the service owner gives feedback on recently made decisions. Hence also more detailed feedback can be given, i.e., feedback, which contains the expected result can be generated. On the other hand this method requires also the access to appropriate devices for displaying the history and for definition of the feedback value.

C. Assistant Service Architecture

The basic scheme for the assistant service results from the combination of the workflows of the evaluation and adaption phase workflow and the feedback mechanism. The architecture of the assistant service as well as the interface of the services are depicted in Figure 5. The cyclic mode of operation of

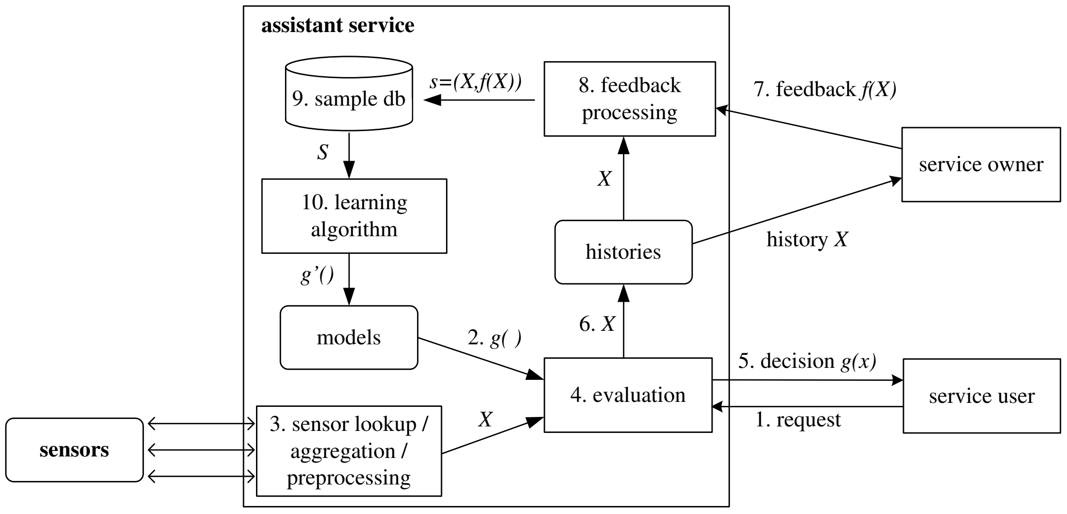

Fig. 5. Generic architecture of the assistant service

the assistant service is as follows:

1) A *request* made by a service user triggers the system.
2) When a request is received, the model $g()$ of the service owner is loaded.
3) Information from all sensors $X = (x_1, x_2, ..., x_n)$ is collected and prepared. The sensor aggregation process can be very complex depending on the applied scenario, the given network architecture and the types of sensors.

4) In the next step the aggregated sensor-data X is applied for model *evaluation* $g(X)$. The sensor-data X describes the situation for which the evaluation is performed.
5) The result of the model evaluation is immediately sent back to the service user.
6) After the evaluation all aggregated sensor-data is stored as *history* entry within a database.
7) The service owner can determine whether the made decision was correct using a *feedback* mechanism as described in Section IV-B. A feedback is a simple message that contains the confirmed or corrected result $f(X)$ and a reference to the related set of sensor values.
8) On receiving feedback a sample $s = (X, f(X))$ is created by merging the feedback with the sensor-data that was stored as history entry.
9) The *sample database* contains a set of samples $S = (X, f(X))$ per user. A set of samples reflects the knowledge of the assistant service about the user's demands.
10) The *learning algorithm* has the task to generate the model $g'()$ on the basis of the given samples S, which includes also the newly given sample. This model replaces the old model $g()$ when it is stored.

During the operation of the system, the sample database is enriched with new samples.

V. Models and Algorithms for Model Adaptation

The ability of the assistant service to adapt to the user's demands precisely plays a central role in the design of our framework. The choice of the appropriate model type and also the appropriate algorithm for model building is very important. The properties of the obtained sensor-data, the sensor-data format, the real time requirement as well as the dynamical adaptation requirement influence the choice of the appropriate learning algorithm (as described in Section V-A). Usually a model is associated to a specific set of algorithms for model building. However, each model and each algorithm feature has individual properties regarding the effort for evaluation, the effort for adaptation, the error tolerance and other special characteristics. The learning algorithm and the related model is basically decoupled from the rest of the system. This ensures the flexible usage in many different environments. If a specific learning algorithm proves to be inadequate for a certain application, it can be replaced by another one. In this section we discuss which model building algorithms are appropriate for our application scenario. Towards the determination of the optimal model building algorithm, we first describe the requirements, which can be derived from the design principles given in Section III-A. We subsequently classify existing learning algorithms. This section concludes with an analysis and an overview of the advantages and disadvantages of the surveyed classes of algorithms.

A. Requirements for Model Generation and Processing

The implications of the systems requirements of Section III-A applied to the model and the algorithm for model generation for usage within the assistant services are as follows:

c) Minimum delay for request processing.: The focus in this work is on providing an assistant service for real time communication services. The model has to be evaluated in real time including the collection of sensor data. Each model has distinctive properties such as the necessary evaluation effort (see Section V-D). The delay for model processing and also the delay for model generation strongly depends on the applied algorithm type as described in Section V-E.

B. Accuracy of the obtained results.

Obviously, the quality of decisions strongly depends on the quality of the applied information (Quality of Information - QoI). Common learning algorithms generally assume the sample data set to be correct, complete and consistent. However, in reality this can not be assured due to the following reasons:

1) In a sample the wrong decision is assigned to a set of sensor values. This can happen if users give wrong feedback by mistake.
2) Errors in the sensor values. Some sensor values can be false due to noise and blur in the sensor or the communication between the sensor and the server.
3) Missing sensor values (e.g. temporary unavailable sensor). A problematic situation occurs when a specific sensor type should be used for evaluation within the model but no adequate sensor or comparable sensor is available. Either an interpolation technique should be applied to determine a substitution for this value, or a model should be used, which also can handle also missing values.

Accurate decisions can be achieved if the situation can be described by sensors, i.e., the assistant service can switch to the desired behavior only if a sensor (that is significant for the decision) changes its state. However, self-learning algorithms are designed to deal with erroneous data, i.e., a certain amount of errors can be handled, depending on the applied algorithm and the count of samples and sensors.

C. Concept Drift.

A sample set is created by assigning a set of sensor values to the *correct* decision (i.e. that is demanded by the service owner). The correct decision depends on the behavior of the service owner and is named as

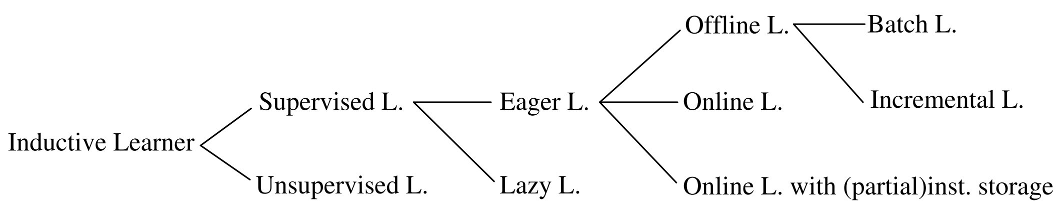

Fig. 6. Considered Classes of Learners

concept. Usually a person (that is the service owner) changes his behavior over time. This case is defined as *concept drift*. Since the interaction of the system with the user should be minimized, we can not assume to receive an explicit notification when concept drift occurs. Consider the case when the feedback for a given set of sensor values contradicts the actual model. The system has to differentiate whether this is a case of concept drift, or whether it is merely a false feedback that has to be discarded.

Reusing in the formal specification of Section III-B, the concept drift problem is as follows: A sample is a tuple $s = (X, f(X))$. The user's behavior is represented by a set of samples $S = \{s\}$. Imagine a number of $c = |S|$ samples containing the result $f(X)$ in a situation X. When the user's behavior changes, the user further on responds with the feedback $f'(X)$ in a similar situation X. In case of a learning algorithm where all existing samples are weighted equally (see Section V-E.1), the system will need $c+1$ samples related to the new behavior $(X, f'(X))$ until the model generated by the self-learning algorithm represents the new behavior. Concept drift processing is done in order to minimize the required new examples until the new behavior of the user is adapted and represented by the generated model.

D. *Qualified Basic Types of Learning*

In this section existing learning algorithms are categorized into type classes regarding their properties like model building strategy or targeted application area. Each type class will be analyzed in order to show if algorithms of this type class are qualified for appliance in the virtual assistant (i.e. if they fit to the given requirements of Section III-A).

The task to learn by observation is performed by inductive learning algorithms (Inductive Learner). All inductive learning algorithms share the common feature of incorporating a hypothesis, but the exact way of how and when to incorporate the hypothesis may vary, which leads to different classes of Learner. An overview of the considered classes of inductive learning algorithms is shown in Figure 6.

A key requirement is that the service owner shall retain control about the behavior of the system. A supervising entity must have the ability to influence the decision process. Hence algorithms which support *supervised learning* are basically qualified for the further selection process of learning algorithms. One intention of using a model is to represent the service owner's demands. In our use case the usage of a model should also shift processing effort from the evaluation phase to the adaptation phase as shown in Section IV-A. With respect to this, the set of inductive learning algorithms is commonly separated into two categories: *Lazy Learner* and *Eager Learner* [Dom97].

1) Lazy Learner.: During the adaptation phase a Lazy Learner usually stores all samples in an n-dimensional space (with n being the number of sensors involved). Since Lazy Learner project all features in an n-dimensional space, they directly deal with continuous features and there is no need for prior discretization. Finally, Lazy Learners are well suited to accommodate the notion of *incremental learning*, i.e., the gradual addition of examples to the training set. A Lazy Learner is lossless, i.e., the model contains all information of the applied samples. K-Nearest Neighbor is a well known Lazy Learner algorithm. While evaluating a new sample, Lazy Learners take into account similar examples from this n-dimensional space. Hence during evaluation, the algorithm performs lookups within this n-dimensional space. A Lazy Learner produces very few effort during the adaptation phase, but (depending on the amount of samples and their dimensionality) much more during the evaluation phase. This property makes algorithms of the *Lazy Learner* category unusable for fulfilling our time critical evaluation requirement.

2) Eager Learner.: Abstractly described an Eager Learner performs a pattern matching on the set of samples and generates a model, which describes the extracted patterns. In comparison to Lazy Learning only a subset of processing instructions have to be performed during the model evaluation phase. For a complexity analysis each algorithm and its model has different dependencies like the number of samples, sensors, or the size of the resulting set. For example, the evaluation of a Naïve Bayes model has linear complexity, whereas the evaluation of a Decision Tree model usually has logarithmic complexity regarding the amount of related sensors. Models of Eager Learner are usually able to fulfill the requirement of bounded maximum processing time. On the other hand a model of an Eager Learner is lossy, i.e., the model does not contain all information, which comes from the applied samples. This property may gain importance when an Eager Learner is applied as Online- or Incremental Learner (see also Section V-E).

Batch Learning	**Incremental Learning**
1) model initialization 2) processes *all* samples 3) create decision model	1) model initialization 2) process *one new* sample 3) adapt model 4) if new sample left then go to step 2

TABLE I

COMPARISON OF BATCH- AND INCREMENTAL LEARNING. ACCORDING TO [SAR04].

E. Model Building Strategies

Algorithms with different strategies for model building exist. They were designed in order to fulfill the requirements and constraints for different fields of application. Model building is not as time critical than model evaluation. On the other hand, while the human supervisor can give feedback to the system, he probably would like to see or to test immediately if the given feedback was adapted by the system in the desired way.

Models and the algorithms for model building can be also divided into two basic strategies: *Offline-Learner* and *Online-Learner*. Per definition Online Learner do not store samples, i.e., after using a sample for model adaptation the sample can be discarded. On the other hand Offline Learner do store samples, i.e., that Offline Learner have access to all stored samples [Sar04].

The Offline Learner can be divided into two classes concerning the mode of operation: *Batch Learner* and *Incremental Learner*. Table I compares the methods of operation for both types.

In the previous section a distinction on the basis of the strategies for model building was performed. Please note that one can not strictly separate between Online- and Offline Learner respectively Batch and Incremental Learner, i.e., also hybrid variants exist (e.g. Online Learner with Partial Instance Storage).

1) Offline-Batch Learner.: Learning algorithms which belong to the class of Batch Learner are commonly known from data-mining applications. Batch Learner use a priori constructed samples and stop learning once these samples have been processed. Those learning algorithms are commonly applied on datasets of large size, with different formats and even containing empty values. Algorithms for model building are usually applied for pattern matching on two dimensional datasets. Hence in principle they can be used within the assistant service, whereby one dimension is equivalent to one sample and the second dimension is achieved by sorting the samples of a user using the time of creation.

Among others we are considering algorithms also known from data-mining [HK01], [FW00] or machine learning application [Mit97]. Batch Learner provide a relative long history of development and many approaches are commonly used. The ability to interpret the whole amount of samples provides the main advantage for Offline Learner. By taking all samples into account, global optimizing strategies can be applied and finally more precise models can be generated. On the other hand Batch Learner are basically not designed for incremental adaptation of the model. Thus they produce much more effort during model building because they require a complete rebuild of the model, processing all samples. A main disadvantage is the missing capability of handling concept drift (see Section V-C).

2) Offline-Incremental Learner.: Historically machine learning has focused on non incremental learning tasks [GC00]. Algorithms of the Incremental Learner class provide the functionality to add one sample to an existing model without rebuilding the complete model. The effort for model building can be reduced, but global optimizing strategies can not be applied. Incremental Learner are often based on Batch Learner algorithms, i.e. some Batch Learner can be extended with an *update-method*. Incremental Learner are preferable regarding the demanded minimal system requirement of the assistant service because they need less effort for model adaptation. However, under the assumption concept drift they produce the same problems as Batch Learner.

3) Online Learner.: There is a relation between Incremental Learner and Online Learner. Online Learner are based on an incremental strategy. In contrast to Online Learner, which basically do not store any samples, an Incremental Learner may also save and may take all recent samples into account for model building. Usually an Online Learner provides *intelligent forgetting*, i.e., samples which are old or conflicting will be discarded. Examples for Online-Learner are the algorithms STAGGER and Winnow [Lit88].

An Online Learner uses only the recent model $g()$ and the new sample $(X, f(X))$ for building the new model. Hence an Online Learner may be more imprecise than an Offline Learner, but this can be compensated by involving statistic factors and heuristics [Sar04]. Every time when a new sample is processed the model is adapted, whereas usually a new sample overrules old conflicting behavior descriptions. For this reason an Online Learner provides fast adaptation to changing user demands. An Online Learner is implicitly designed to handle the effect of concept drift (as explained in Section V-C).

4) Online Learner with Partial Instance Storage.: The partial instance storage is a hybrid solution between Offline and Online Learner. These solutions store a certain amount of samples. However, usually only actual samples are saved and old detached samples are thrown away [Mal02]. A pure Online Learner uses only the model and the new samples. An Online Learner with Partial Instance Storage uses a subset of the information, which is given by the samples. The option to access the samples which are actually *active* (used within the model), is used to compensate the loss of information which is gained during model building by generalization and discretization. The algorithms LAIR, HILLARY and FLORA are based on the concept of STAGGER and Winnow but they use partial instance storage [EW91],[IWL88],[WK96].

F. Meta Algorithms for Online Learning

An additional way to perform Online Learning can be obtained by the use of a special type of meta algorithm. The meta algorithm implements the tasks for sample (instance) management in the same way as an Online Learner does. Further the meta algorithm encapsulates a common Offline Learner. By applying such a meta algorithm on an Offline Learner can be converted to an Online Learner (usually with partial instance storage). A main advantage of the usage of a meta algorithm is the ability to incorporate the features of Offline Learner and to change the encapsulated algorithm as needed. MetaL(B) and MetaL(IB) are examples for such Meta Algorithms for Online Learning [Wid97]. Through this dynamic instance management a Meta Learner provides also an effective way to handle concept drift. When the meta algorithm adds a new sample, the advantages of incremental learning can be used. On the other hand, in case of an Offline Learner a rebuild of the whole model has to be performed when a sample has to be removed from the model.

G. Summary – Qualified Models and Algorithms

Table II gives a high level comparison of the different classes of learning algorithm. We compare the following properties: effort for model building, capability of concept drift handling and the functionality/quality of available approaches. The functionality/quality value represents an overall rating of the functionality provided by the algorithm (capability to process continuous values, missing sensor values, optimizing strategies, etc.) and the quality of the resulting model.

Concept drift handling is a major issue for the addressed virtual assistant. That means the algorithm has to provide Online Learning properties. Offline Learner in conjunction with a meta algorithm and Online Learner (with Partial Instance Storage) are qualified for the deployment within the assistant service.

	build efficiency	concept drift	functionality/quality
Offline-Batch	-	- -	+ +
Offline-Incremental	+	- -	+ +
Online	+ +	+ +	- -
Online-Part.-Storage	+	+ +	-
Meta+Batch	- -	+	+
Meta+Incremental	-	+	+

TABLE II

HIGH LEVEL COMPARISON OF LEARNING ALGORITHM CLASSES

Further restrictions are given by requirements of the target application scenario of the assistant service (see Section III-A). Depending on these requirements the optimal algorithm has to be chosen. Common use cases will perform learning in an incremental manner. In these use cases Batch Learner and Meta Learner will produce generally more processing time for model generation than Incremental Learner or Online Learner. On the other hand the accuracy and functionality provided by the algorithm has to be taken into account. Common Offline Learner are considered more mature in comparison to the majority of the Online Learner approaches.

VI. CONCLUSION

In this paper, we proposed the virtual assistant architecture, which is able to assist a user in performing complex communication management tasks. To this end we analyzed the design conditions and system requirements and described the proposed control cycle for the learning process. Further, we investigated models and algorithms for model building and selected the qualified ones for our exemplary telephony application. In our approach model generation is assisted by related sensors information and verified by only a minimum amount of active end user feedback.

We relieve management complexity from the end user, helping to increase the consumer acceptance for existing and future communication services. We envision, that due to the generality of the assistant service, the proposed architecture can be applied in further scenarios. E.g., the usage of the assistant service in combination with an IP communication system will offer mechanisms to prevent the user being bothered by Spam over IP telephony (SPIT). This goal can be accomplished by learning call patterns (caller ID, time, source domain) that match with SPIT calls to suppress such communication requests later on.

We have built a prototypical implementation of our assistant service to validate the applicability of our virtual service manager. Novel applications and value-added services become feasible by applying the new assistant service.

REFERENCES

[BGS02] Michael Beigl, Hans W. Gellersen, and Albrecht Schmidt. Multi-sensor context awareness in mobile devices and smart artifacts. *Mobile Networks and Applications*, 7(5):341–351, 2002.

[Dom97] Pedro Domingos. Control-sensitive feature selection for lazy learners. *Artificial Intelligence Review*, 11(1):227–253, 1997.

[EW91] Renée Elio and Larry Watanabe. An incremental deductive strategy for controlling constructive induction in learning from examples. *Machine Learning*, 7(1):7–44, July 1991. http://dx.doi.org/10.1007/BF00058925.

[FMR03] Alois Ferscha, Rene Mayrhofer, and Harald Radi. Recognizing and predicting context by learning from user behavior. In *Proceedings of the 1st International Conference on Advances in Mobile Multimedia*, volume 171, pages 25–35, 2003.

[FW00] Eibe Frank and Ian H. Witten. *Data mining*. Morgan Kaufmann, San Francisco, 2000.

[GC00] Christophe Giraud-Carrier. A note on the utility of incremental learning. *AI Communications*, 13(4):215–223, December 2000. http://www.cs.bris.ac.uk/Publications/Papers/1000535.pdf.

[HK01] Jiawei Han and Micheline Kamber. *Data Mining. Concepts and techniques.* Morgan Kaufmann, San Francisco, 2001.

[HK02] E. Horvitz and P. Koch. Coordinate: Probabilistic forecasting of presence and availability, 2002.

[IWL88] W. Iba, J. Woogulis, and P. Langley. Trading simplicity and coverage in incremental concept learning. In *Proceedings of the Fifth International Conference on Machine Learning*, pages 73–79, San Francisco, 1988. Morgan Kaufmann.

[Lae99] K. Van Laerhoven. Online adaptive context awareness, 1999. http://www.teco.edu/tea/thesis99.ps.

[Lit88] Nick Littlestone. Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. *Machine Learning*, 2(4):285–318, April 1988. http://dx.doi.org/10.1007/BF00116827.

[LS04] Jonathan Lennox and Henning Schulzrinne. Call Processing Language (CPL): A Language for User Control of Internet Telephony Services, October 2004. Request for Comments 3880.

[Mal02] Mark Maloof. Incremental learning with partial instance memory, June 2002. http://www.cs.georgetown.edu/~maloof/pubs/ismis02.pdf.

[Mit97] Tom M. Mitchell. *Machine learning*. McGraw Hill, New York, 1997.

[Sar04] Warren Sarle. What are batch, incremental, on-line, off-line, June 2004. http://citeseer.ist.psu.edu/697724.html.

[Wid97] Gerhard Widmer. Tracking context changes through meta-learning. *Machine Learning*, 27(3):259–286, June 1997. http://dx.doi.org/10.1023/A:1007365809034.

[WK96] Gerhard Widmer and Miroslav Kubat. Learning in the presence of concept drift and hidden contexts. *Machine Learning*, 23(1):69–101, April 1996. http://dx.doi.org/10.1007/BF00116900.

[Wya97] Jeremy Wyatt. Supervised learning. In Alan Bundy, editor, *Artificial Intelligence Techniques*, pages 111–112. Springer, Berlin, 1997.