# SAMVAAD: Speech Applications Made Viable for Access-Anywhere Devices

*Nitendra Rajput  Amit A. Nanavati  Mohit Kumar  Pankaj Kankar  Rajan Dahiya*
IBM India Research Laboratory, I.I.T., Hauz Khas, New Delhi-110016. Ph:+91-11-2686-1100
{rnitendr,namit,mohitkum,kpankaj}@in.ibm.com
Symposium 3: Ubiquitous Computing, Services and Applications

***Abstract*— The proliferation of pervasive devices has stimulated the development of applications that support ubiquitous access via multiple modalities. Since the processing capabilities of pervasive devices differ vastly, device-specific application adaptation becomes essential. We address the problem of speech application adaptation by dialog call-flow reorganisation for pervasive devices with different memory constraints. Given an *atomic* dialog call-flow $A$ and device memory size $m$, we present optimal deterministic algorithms,** RESEQUENCE **and** BALANCE-TREE**, which minimise the number of questions in the reorganised output call-flow $A_m$. Algorithms** MASQ **and** MATREE **produce $C_m$, minimally distant from input call-flow $C$, while accomodating the memory constraint $m$. These two minimisation criteria are capable of capturing various usability requirements important in dialog call-flow design. The following observation forms the cornerstone of all the algorithms in this paper: Two grammars $g_1$ and $g_2$ comprising of $|g_1|$ and $|g_2|$ elements respectively can be merged into a single grammar $g = g_1 \times g_2$ having $|g_1| \cdot |g_2|$ elements for the sequential case, and $g = g_1 + g_2$ having $|g_1| + |g_2|$ elements for the tree case.**

**Device-specific considerations lead us to introduce the concept of an $\langle m, q \rangle$-*characterisation* of a call-flow, defined as the set of pairs $\{(m_i, q_i) | i \in N\}$, where $q_i$ is the minimum number of questions required for memory size $m_i$. Each call-flow has a unique, *device-independent* signature in its $\langle m, q \rangle$-*characterisation* – a measure of its adaptability.**

**We present** SAMVAAD**, a system that implements these algorithms on call-flows authored in VXML containing SRGS grammars. The system was tested on an IBM voice browser using a sample airline reservation system call-flow reorganised for memories ranging from 64 MB to 210 KB. We ran an experiment with 14 users to obtain feedback on the usability of the adapted call-flows.**

**Keywords: Speech Processing, Pervasive Computing, Dialog Call-Flow Optimisation.**

## I. Introduction

Samir is driving to a theatre to watch a movie. He accesses the theatre's IVR system from his mobile to make a reservation. He gets disconnected from the IVR system twice during the conversation. It would be nice if Samir could connect once to download the application, use it locally and transmit the final response, thus avoiding network instabilities as well as connection charges.

Users are increasingly accessing remote applications on the internet and running a plethora of local applications from their mobile devices. From the users' point of view, they would like more and more applications to be accessible via various interfaces (voice, multimodal) from their pervasive devices. Pervasive devices are different from desktop computers in two fundamental ways. One, they occur in various sizes with vastly differing capabilities, and by virtue of mobility, are not always connected to the network. This combination gives rise to some very interesting challenges and possibilities.

From the application provider's point of view, an application composed on M pages to be accessed via N devices requires M x N authoring steps, and results in M x N presentation pages to be maintained. To address the application developer's nightmare, many application programming tools have been proposed [1], [2]. Such tools allow the programmer to develop a generic application which is automatically adapted for various devices. So far, these techniques address problems in the visual domain only.

We are interested in device-specific adaptation of speech applications. Traditionally, speech applications run on a remote server, and several client-server interactions take place in the course of a dialog. A client-server model incurs transmission costs, and is prone to transmission errors, which could result in degraded speech recognition accuracy. The use of compression for reducing transmission costs introduces other complications [7]. In order to circumvent such problems, speech recognition at the client offers a viable alternative.

Litman and Pan [4] claim that the performance of a conversational system can be improved by adapting dialog behavior to individual users. Jameson [3] discusses the cognitive aspects of conversational applications taking into consideration the user's available time and her ability to concentrate on the interaction. Dialog call-flow adaptation of a conversational system for improving the speech recognition accuracy has been addressed in [5]. Levin et al. [6] use learning techniques for designing a conversational system and modelling it as a Markov decision process. However, none of these efforts take device characteristics into consideration.

### A. Our Contribution

We investigate the problem of dialog call-flow reorganisation for pervasive devices with memory constraints. The crux of the reorganisation lies in altering the memory requirement of the underlying grammar. We achieve this by continually merging *atomic* (an atomic grammar is one which cannot be split into subgrammars) grammars till the resulting grammar size can be supported by the device. We present optimal deterministic algorithms, RESEQUENCE and BALANCETREE,

which provide solutions for the two types of call-flows, *sequential* and *tree-type* respectively. Typically, call-flows are designed based on various usability criteria. When such 'ideal' reference call-flows are available, minimising the number of *changes* to this ideal call-flow is a reasonable goal. MASQ and MATREE minimally alter a given dialog call-flow to accommodate it within a given memory constraint.

We introduce the concept of a *device-independent* characterisation of dialog call-flows. This signature can be constructed by finding the set of ⟨memory, minimum number of questions⟩ corresponding to each call-flow and provides a benchmark for the *adaptability* of a call-flow.

We have built a system SAMVAAD that takes as input a VXML dialog containing SRGS grammars and a memory size $m$. SAMVAAD has VXML and SRGS parsers that parse the input dialog to build a call-flow which is input to the above algorithms. The output of the algorithms is converted back to a VXML dialog.

Section II discusses the background of the problem. Section III details reorganisation algorithms RESEQUENCE, BALANCETREE, MASQ and MATREE, and introduces ⟨m,q⟩-*characterisation* of a call-flow. Section IV presents SAMVAAD which realises the reorganisation algorithms and describes some experiments with it. Section V concludes the paper.

## II. Background and Problem Setting

In this section we briefly discuss speech recognition systems, their grammars and memory requirements and the need for client-side processing.

*Client-side processing:* In server-side processing intensive systems, a typical approach to alleviate server bottlenecks and achieve scalability is to offload processing to the client side to the extent possible. The evolution of Javascript in the context of the Web is an example. Further, for pervasive devices, server connectivity comes at a cost and is not always robust. Together, these factors make a compelling case for disconnected, client-side processing of dialog call-flows.

*ASR:* An Automatic Speech Recognition (ASR) system consists of two main components, an acoustic model and a language model. The acoustic model estimates how a given word or 'phone' is pronounced. The complexity (hence memory requirement) of an acoustic model is dependent on the training data and is fixed once the model is built. The language model provides a probabilistic estimate of the likelihood of a sequence of words. In conversational systems, a language model is represented by a speech recognition grammar. The memory requirement of a grammar depends on the number of choices that it encapsulates. Therefore, the memory requirement of a call-flow can be altered by changing its underlying grammars.

*Memory optimisation of ASR systems:* While the available memory size on devices is increasing, so is their ability to support more and more complex conversational systems. Conversational systems range from single-word-based-recognition to phrase-recognition to complex-grammar recognition to large-vocabulary-recognition coupled with NLU, in the order of increasing memory requirement. The state of the art sophisticated speech recognition tasks ("how-may-I-help-you" type of tasks, which provide much better user experience) require more memory than is typically available on laptops (≈512 Mb). Therefore it becomes necessary to adjust the complexity of the conversational system for different devices.

We introduce, formulate and analyse device-specific adaptation of dialog call-flows. Based on the algorithms presented, we have built a system to assist a speech application developer. The system outputs the adapted call-flow for different devices, but the task of designing corresponding prompts and help messages need to be done by the developer. While the complete automation of dialog call-flow adaptation is a distant holy grail, this is a first step towards it.

## III. Reorganisation Algorithms

We assume that the input call-flow comprises of *atomic* dialogs[1] only. Such a call-flow is called an *atomic call-flow*. An atomic call-flow has the least memory requirement and also the most number of dialogs. $M$ denotes the memory constraint.

We present two sets of algorithms. The first set consists of two algorithms RESEQUENCE and BALANCETREE that minimise the number of dialogs (questions/prompts) given an atomic dialog call-flow with respect to a given memory constraint and operate on sequential and tree-type call-flows respectively. The second set of algorithms MASQ and MATREE *minimally alter* the given reference call-flow (sequential and tree-type respectively) to accommodate the given memory constraint. A reference call-flow $C^r$ may be the result of a myriad of considerations and serves as a guideline for reorganisation. $C^r$ represents a guideline and therefore is a *soft* constraint. It need not be atomic. For this set of algorithms, we naturally require a notion of distance to quantify minimal alteration. Apart from the memory constraint, all algorithms accommodate *reorganisation* constraints.

### A. Reorganisation Constraints

Reducing the number of questions improves usability and optimises memory usage. However, grammar merges across certain subdialogs may either not be possible or not preferable due to reasons such as data dependency and usability. We call these *reorganisational* constraints.

Reorganisational constraints represent *hard* constraints of two types. One, that insist that a certain group of dialogs be merged, and two, that forbid a certain group of dialogs from being merged. The first set *must-merge* is a set of sets of dialogs which must be merged. The second set *must-separate* is a set of sets of dialogs which should not be merged. For example, in an airline reservation system (figure 6), the grammar for the departure city cannot be merged with the grammar for the arrival city because the arrival city depends upon the departure city. From a usability standpoint, one may not want to merge the date of departure with the flight number. Another possible reason for preventing merges might be to

[1] dialogs which cannot be split into subdialogs - analogous to atomic grammars

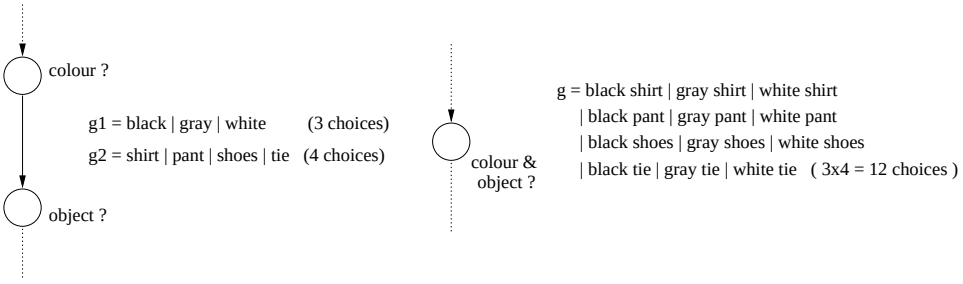


Fig. 1. Effect of merging/splitting a sequential grammar.

improve recognition accuracy. Reorganisational constraints thus provide a mechanism to incorporate various practical considerations and constraints to improve the overall usability and performance of a dialog call-flow.

### *B. Minimising the Number of Dialogs*

In this section, we present RESEQUENCE and BALANCE-TREE to minimise the number of dialogs in a sequential and tree-type call-flow respectively while respecting the memory and reorganisational constraints.

*1)* RESEQUENCE:

*Observation 1:* Two grammars $g_1$ and $g_2$ comprising of $|g_1|$ and $|g_2|$ elements respectively can be merged into a single grammar $g = g_1 \times g_2$ having $|g_1| \cdot |g_2|$ elements. Figure 1 shows an example, where as a result of a merge operation the memory requirement goes upto 12 from 4.

A call-flow can be represented by a sequence $L = \{1, \ldots, n\}$ of *atomic* dialogs representing the order in which the dialogs are presented. The goal is to merge as many questions as possible while respecting the memory constraint. The memory requirement $m(g_i)$ for each $g_i$ is known. We construct a graph $G$ as follows. The vertex set $V(G)$ contains precisely the elements of $L$. For each vertex $i$ in $G$, we add edge $(i, j)$ if $\Pi_{k=i}^{j} m(g_k) \leq M$ $(i < j \leq n)$, i.e, the memory requirement of the merged grammars $g_i$ through $g_j$ can be accommodated within memory constraint $M$. As a result of this, $G$ becomes a directed acyclic graph, possibly disconnected. Now, we need to find the shortest path (or set of paths) from 1 to $n$, by finding the shortest path for each connected component of $G$. Each edge in the shortest path (set of paths) denote the subsequence of questions being merged. The sets of dialogs in *must-merge* are merged as a preprocessing step, and dialogs merged as a result of this step are *considered atomic*. $L_m$ denotes the output call-flow with the minimum number of dialogs. $L_m$ may contain merged (non-atomic) dialogs. Figure 2 shows an example of a graph with 7 nodes.

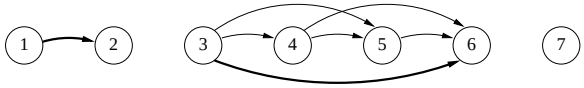


Fig. 2. An example directed acyclic graph of a call-fbw.

The edges of this graph represent the possible merges in the call-flow. The dotted edges identify the nodes that are not allowed to merge due to reorganisation constraints. The shortest path for the graph is indicated by thick edges in Figure 2.

---

RESEQUENCE

1) input: atomic sequential call-fbw $L_a$.
2) output: sequential call-fbw $L_m$ with the minimum number of questions.
3) Construct a graph $G(V, E)$ as follows:
   a) Merge all *must-merge* dialogs in $L_a$ to obtain $L_a^m$.
   b) Represent all dialogs by vertices labelled {1,...,n}
   c) for each vertex $i (1 \leq i \leq n)$
      i) for each vertex $j (i \leq j \leq n)$
      ii) if $(\Pi_{x=i}^{j} m(g_x) \leq M)$ && $\{i, j\} \notin must-separate$, add $(i, j)$ to $G$.
4) Find the shortest (set of) path(s) as follows:
   a) start = 1. $L_m = \emptyset$.
   b) while $(start \leq n)$
      i) $L_m = L_m \cup \{start\}$.
      ii) select $max_j$ such that $(start, j) \in E$.
      iii) start = j+1.
5) output $L_m$.

---

*Claim 1:* RESEQUENCE is correct and runs in $O(n^2)$ time. The graph construction phase takes $O(n^2)$ time to check every pair of vertices for adding edges. The shortest path phase takes $O(n)$ time, since at each vertex the largest adjacent vertex can be chosen greedily to yield the shortest path.

*2)* BALANCETREE:

*Observation 2:* Two grammars $g_1$ and $g_2$ comprising of $|g_1|$ and $|g_2|$ elements respectively can be merged into a single grammar $g = g_1 + g_2$ having $|g_1| + |g_2|$ elements. Figure 3 shows an example. As a result of the merge operation, the memory requirement goes up from 2 to 4 ($g1$ to $g1'$).

---

BALANCETREE

1) input: tree-type call-fbw $T$.
2) output: tree-type call-fbw $T_m$ with the minimum number of questions.
3) initialise $T_m = T$; boolean changed = false.
4) do
   a) Find the longest path in $T_m$ and identify its lowest 2-subtree $t_2$.
   b) if (*shorten*$(T_m, t_2)$) changed = true.
5) while (changed) ;
6) output $T_m$.
7) *shorten*$(T_m, t_2)$
   a) while $(t_2 \neq$ "$root''$)
   b) do

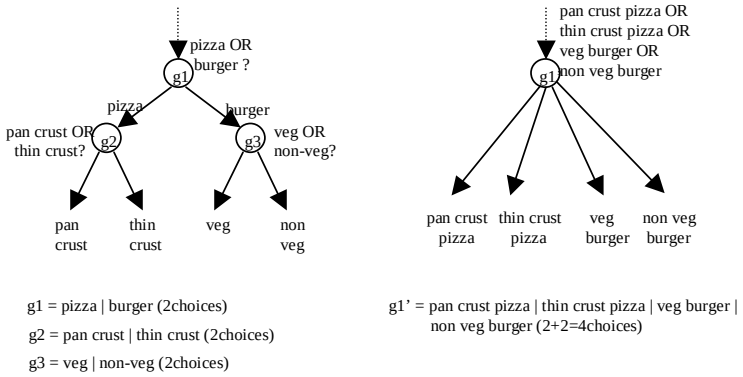


Fig. 3. Effect of merging/splitting a tree-type grammar.

i) if (*fold*$(t_2)$) return true.
ii) else $t_2 = parent(t_2)$.

c) done
d) return false.

8) *fold*$(t_2)$

a) if (($\Delta(t_2) \geq$ degree(children$(t_2)$)) return true.
b) return false.

---

*Definition 1:* The *degree* of a vertex is the number of its children.

*Definition 2:* A *2-subtree* of a vertex $v$ is a tree of depth 2 with $v$ as the root.

*Definition 3:* A 2-subtree of a vertex $v$ is *balanced* if all the leaves of the 2-subtree are at distance 2 from $v$, i.e., no child of $v$ is childless. A 2-subtree of a vertex $v$ is *1-balanced* if at least one child of $v$ is childless. A 2-subtree is either balanced or 1-balanced.

*Definition 4:* Let the maximum degree of any vertex in a call-flow tree be denoted by $\Delta$. The *vacancy* of a vertex $v$ is defined as ($\Delta$ - *degree(v)*).

*Definition 5:* The *fold* operation is defined on root $v$ of a 2-subtree and allows $v$ to directly inherit all its grandchildren if the $\Delta \geq \Sigma_i degree(child_i(v))$. As a result of this operation, all the grandchildren of $v$ become its own children, and the original children are removed. This operation reduces the height of the tree by 1.

*Claim 2:* The greedy application of the folding operation cannot lead to suboptimal solutions.

*Proof:* A greedy application of the folding operation on root $v$ of a 2-subtree can lead to two possibilities. As a result of a *fold(v)* operation, a subsequent *fold(parent(v))* is possible, or it is not. In first case, since both *fold* operations must be done optimality is preserved. In second case, it turns out that only one of *fold(v)* or *fold(parent(v))* could have been applied, either of which would lead to a height reduction of 1. ■

Claim 2 suggests that a bottom-up approach on the longest paths in the tree one 2-subtree at a time might provide a solution. This is the essence of BALANCETREE. At each step, the longest path is found, its height reduced by 1, if a *fold* operation is possible at any vertex from the grandparent of the leaf in the longest path to the root. Note that *shorten* traverses up the tree till it is able to reduce the height by 1. After this reduction, the longest path is calculated again and the same procedure is applied. If at any time, the longest path cannot be reduced, the algorithm terminates. Since the longest path is found globally at each step, and since the height of the tree is reduced only 1 at a time, we obtain a maximal height reduction.

*Claim 3:* BALANCETREE is correct and runs in $O(n^2)$ time where $n$ is the number of vertices in the tree. Since the *fold* operation is the dominating cost, consider the degenerate case of a tree of depth $n$ with one vertex at each level (a path). Suppose the root has vacancy $n$, then each vertex folds into its parent bottom-up one at a time. This accounts for $O(n^2)$ *fold* operations. Each vertex is examined a maximum of 2 times for each level it visits, as a child for its degree and as a parent for its vacancy amounting to a cost of $2n^2$.

*3) Hybrid Call-flows:* In general, *hybrid* call-flows may contain sequential parts as well as tree-type parts. The algorithms RESEQUENCE and BALANCETREE can operate on the separate parts independently of each other. Without loss of generality, we can execute RESEQUENCE followed by BALANCETREE. As a result of RESEQUENCE, a shortened sequence may contain a vertex $v$ with increased memory requirement and hence a reduction in *vacancy(parent(v))*. This reduction in vacancy may prevent $v$ from folding into its parent. If RESEQUENCE would not have affected $v$, then BALANCETREE would have folded $v$ into its parent. Either case leads to a height reduction of 1. This argument is similar to the one used in claim 2 above.

### C. Minimally Altered Call-flows

Minimising the number of questions need not be the single motivating factor for reorganisation. Call-flow design entails accounting for numerous factors such as speech recognition accuracy and natural language processing. Such a well-designed call-flow can be used as a reference and altered *minimally* to meet memory constraints.

For quantifying minimality, it is necessary to define a notion of distance. We introduce a simple notion of distance based on two operations: *merge* and *split*. A single application of either of these operations on a call-flow $C$ (whether sequential or tree-type) increases the distance of the modified version from $C$ by 1. Let $C_a$ denote the atomic version of $C$. A *split* operation on a non-atomic call-flow $C$ can be simulated by

replacing a dialog in $C$ by its atomic components from $C$. Observe that no *split* operation was required for minimisi the number of dialogs in a call-flow because the initial ca flow was atomic. In this case, however, since the referen call-flow need not be atomic, we need to support the *sp* operation.

*1)* MASQ: Given a sequential, not necessarily atomic r erence call-flow $L^r$ and a memory constraint $M$, MAS constructs a call-flow $L^r_m$, a *minimally altered* version of that satisfies the memory constraint $M$. MASQ is simple. If a dialog can be accommodated within $M$, it remains unchang For the others, it has to be split.

MASQ

1) input: atomic sequential call-flow $L_a$.
2) input: reference sequential call-flow $L^r$.
3) output: minimally altered sequential call-flow $L^r_m$.
4) Construct a graph $G(V, E)$ as follows:
   a) Represent all dialogs by vertices labelled {1,...,n}
   b) Let $L^r_m = L^r$.
   c) for each vertex $i (1 \leq i \leq n)$
   d) if $(m(g_i) > M)$, then $split(i, L^r_m)$.
5) output $L^r_m$.
6) *split*$(v, L^r_m)$
   a) Find the set of atomic components $S_v = \{v_1, ..., v_\ell\}$ of $v$ from $L_a$.
   b) If $\exists v_i \in S_v, m(v_i) > M$, output "IMPOSSIBLE" and exit.
   c) Otherwise,
      i) $i = 1. SS_v = \emptyset$.
      ii) Find the largest $k$ such that $\Pi_i^k m(v_j) \leq M$.
      iii) $i = (k + 1)$.
      $SS_v = SS_v \cup \{i - k\}$.
      if $(k < \ell)$, repeat previous step.
   d) Replace $v$ by $SS_v$.

*Claim 4:* MASQ is correct and efficient.

*Proof:* The basic idea is to split only those dialogs that use memory larger than $M$. The *split* routine ensures the smallest number of splits. Each call to *split* involves a linear search of the corresponding atomic component set. This greedy method yields an optimal solution. ∎

*2)* MATREE: MATREE works on tree-type call-flows. In this case, the 'splitting' of a vertex is like an 'unfolding' (similar to the *fold* operation being analogous to the *merge*). The *unfolding* operation may cause the depth of the tree to increase, but the memory requirement at the vertex decreases.

MATREE

1) input: atomic tree-type call-flow $T_a$.
2) input: reference tree-type call-flow $T^r$.
3) output: minimally altered tree-type call-flow $T^r_m$.
4) initialise $T^r_m = T^r$;
5) do
   a) Traverse $T^r_m$ in preorder and if $(m(T^r_m) > M)$, then
      i) $unfold(T^r_m)$ ;
      ii) $matree(T_a, T^r_m)$ ;
6) output $T^r_m$.
7) *unfold*$(T)$
   a) Identify the corresponding subtree $T'$ of $T$ in $T_a$.

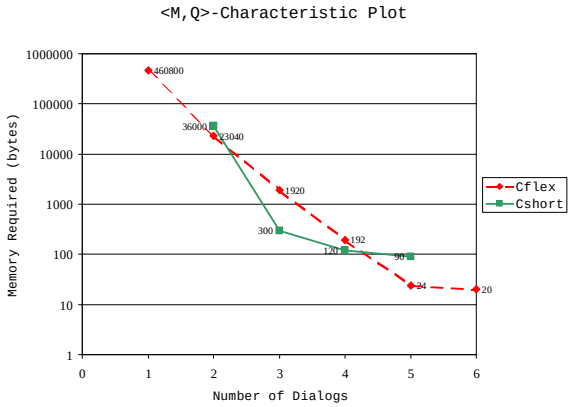


Fig. 4. Sample $\langle m, q \rangle$-*characterisation* plots for *Cflex* and *Cshort*. Due to reorganisational constraints, *Cshort* has a minimum of two questions. *Cflex* is more flexible and can support devices with lower memories.

   b) Traverse $T'$ bottom-up and for each vertex $v' \in T'$, $T'_m$ = *shorten*$(T', v)$.
   c) Replace $T$ with $T'_m$.

*Claim 5:* MATREE produces a tree with the minimum number of alterations.

*Proof:* Every vertex in $T^r_m$ can be created by merging several vertices in $T_a$. Each vertex in $T^r_m$ corresponds to a unique subtree in $T_a$. The *unfolding* operation identifies this subtree, and attempts to *shorten* it as much as possible to minimise alteration. Since each vertex corresponds to a unique subtree, the order of replacing the subtrees is inconsequential. ∎

### D. Device-independent Call-flow Characterisation

Given a call-flow $C$, the above algorithms can be run with various values of memory size $m_i, 1 \leq i \leq n$ and their corresponding minimum number of questions obtained. This gives us a *device-independent* characterisation of $C$. Since these $\langle m_i, q_i \rangle$-pairs are unique for a given call-flow, they can be thought of as a *reorganisational signature* of the call-flow. We call this signature an $\langle m, q \rangle$-*characterisation* of $C$. From a practical perspective, the $\langle m, q \rangle$-*characterisation* of $C$ provides a means for comparing two call-flows that essentially (semantically) perform the same task, that of doing airline reservation, for example, and traces the memory requirements of each. This is important in call-flow design.

The $\langle m, q \rangle$-*characterisation* function of $C$ is typically a decaying function – a composition of lines with negative, decreasing slopes. Consider a sequential call-flow $L$ of $n$ dialogs where each dialog $i$ requires memory $m_i$, a single question requires $\Pi_1^n m_i$. This is the largest value of the function. The smallest value is $max\ m_i$. When all the numbers are the same, this function reduces to an exponential function on $m_1$. In the most general case, this function is similar to the *falling factorial* function, except that the the numbers are not necessarily consecutive, so the slope of the curve continues to decrease faster than the *falling factorial* function. In the case of tree-type call-flows, since the numbers get added rather than multiplied, the effect is less pronounced.

Figure 4 shows a comparison of the $\langle m, q \rangle$-*characteristics* of two imaginary call-flows, *Cflex* and *Cshort*. Both call-

flows are semantically equivalent in that they perform the same task (for example, airline reservation) but were designed with different assumptions and considerations in mind. The choice of the call-flow could depend on a number of factors. For example, if the designer expects client devices (with less than 90 bytes) to access the application, *Cflex* is preferable. However, if the designer is concerned that he does not want to ask more than 3 questions, then *Cshort* accomplishes this with lesser memory.

## IV. SAMVAAD: Architecture, Implementation and Experiments

In this section, we detail the architecture and implementation of SAMVAAD, a dialog call-flow reorganisation system. We use a sample airline reservation system call-flow to illustrate the functioning of SAMVAAD and to analyse the behaviour of the algorithms for different memory constraints. We also present the outcomes of a user study we conducted to evaluate the usability of the reorganised call-flows, the output of SAMVAAD.

SAMVAAD is implemented in Java2(v1.5.0) and the generated dialogs were deployed on an IBM WebSphere Voice Response browser that uses the IBM WebSphere Voice Server for speech recognition and speech synthesis. We tested the system on several working VXML dialogs and note that it generates syntactically correct dialogs.

### A. Architecture and Implementation

Figure 5 shows the architecture of SAMVAAD. A VXML dialog file specifies the dialog call-flow, its prompts, and grammars. The VXML dialog file is first parsed by the VXML Parser. The Grammar Parser parses the SRGS grammars referred to in the VXML dialog. The Reorganisation Algorithms module processes the call-flow graph output by the VXML parser. This module outputs the reorganised call-flow graph and identifies the grammars that need to be merged. The Grammar Merger module merges the grammars and finally the VXML Generator reconverts the call-flow graph into VXML. We describe the five components in detail:

- VXML Parser: This parser parses VXML (version 2.0) dialogs and extracts the call-flow in a DOM tree representation. Each node of the tree corresponds to an input element in a VXML dialog and has an associated grammar. The number of children at a node is equal to the number of choices after the input <field> block. The VXML parser outputs a sequential or tree-type call-flow. It invokes the Grammar Parser, when required, by passing a <grammar-file> handle to it.
- Grammar Parser: It parses the grammar file associated with a node of the call-flow DOM tree. We preferred the SRGS-XML format so that we couls use a JAXP implementation of the DOM parser. We parse the grammar file to count the number of choices the grammar encapsulates: If the elements of a node are present in a <one-of> (an SRGS-XML tag element) block, all choices within each of these elements are *added*; otherwise, they are *multiplied*.
- Reorganisation Algorithms: This module contains the algorithms explained in Section III.
- Grammar Merger: Grammar merges are of two types: OR-type and the AND-type. For a OR-type merge, the final root will comprise of <one-of> block that contains references to the nodes of original grammars as its children. An AND-type grammar merge contains the rule references of the original grammars in its root node.
- VXML generator: The VXML generator takes the reorganised call-flow and generates the final VXML dialog that contains the merged grammars.

TABLE I
MEMORY REQUIREMENT FOR THE DIFFERENT GRAMMAR SIZES

| Grammar size | Memory required (bytes) | Grammar size | Memory required (bytes) |
|---|---|---|---|
| 1 | 47916 | 3000 | 53356 |
| 116 | 47960 | 4000 | 58712 |
| 280 | 48080 | 5000 | 60532 |
| 370 | 48052 | 7000 | 61888 |
| 960 | 48412 | 9600 | 62860 |
| 1440 | 48644 | 10670 | 63060 |

The VXML generator requires new prompts for the merged grammars. Our implementation of the system generates these prompts by concatenating the prompts corresponding to the original grammars. However these can be re-authored manually to provide a better correspondence with their associated grammars. The reorganisation constraints are specified in the input VXML dialog through a special reorganisation tag <must-separate> that is parsed by the VXML parser and is appropriately represented in the call-flow DOM tree structure.

### B. Experimenting with Grammar Memory

To demonstrate the varying memory requirements of grammars on a device, we calculated memory required by a speech recognition system for grammars of various sizes. We used a large vocabulary English speech recognition system to decode a speech utterance. The utterance comprised of a single word. The grammars used for decoding consisted of isolated words. The size of grammar therefore reflects the vocabulary of the recognition system. The decoding was performed for the same utterance, but with varying grammar sizes. Table I shows the memory required to perform decoding on a 450 MHz Quad processor AIX machine with 2GB of RAM. These numbers may vary depending upon the particular implementation of the speech recognition system and the hardware. The memory requirement of 47916 bytes for decoding the utterance against a one word grammar can be interpreted as the footprint that is required by the non-grammar specific portion of the speech recognition system. The additional memory requirement with the increase of grammar size is a reflection of the increased memory requirement for decoding the same speech utterance.

### C. Reorganisation Experiments

Figure 6(a) shows a sample airline reservation system call-flow. The reorganisation constraints, which are of the *must-*

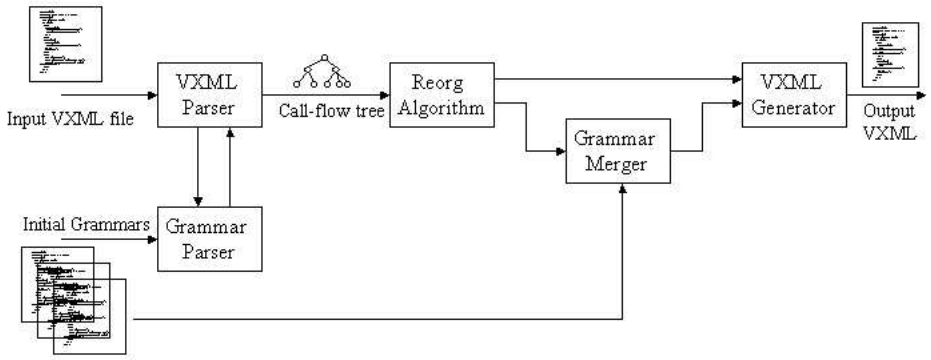


Fig. 5. System Architecture of SAMVAAD

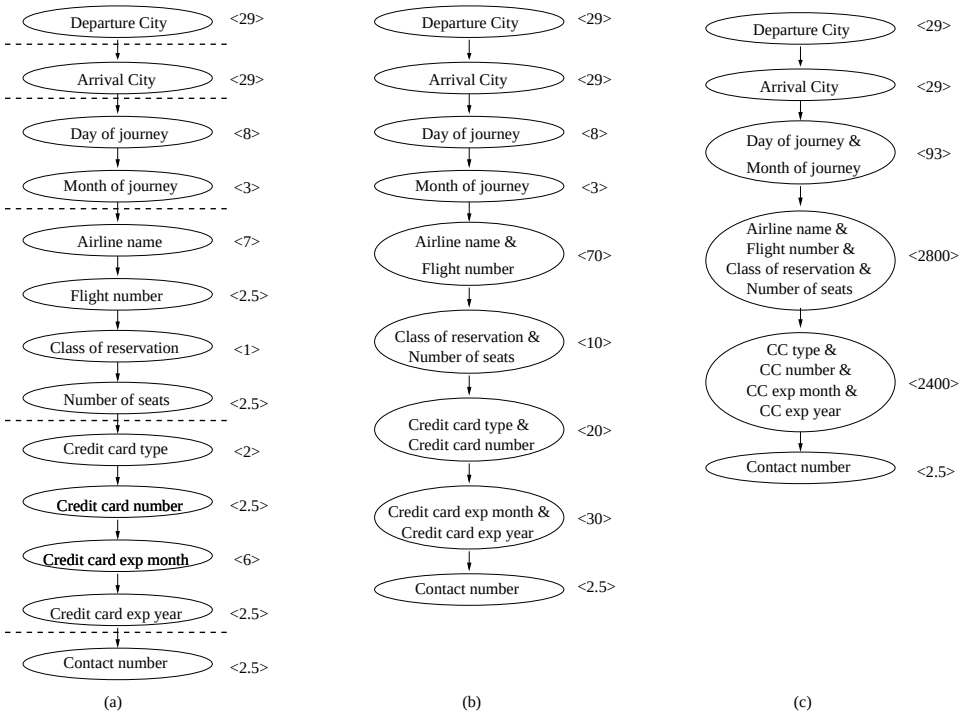


Fig. 6. (a) An airline reservation atomic call-fbw. (b) Output of the RESEQUENCE algorithm with m=70KB. (c) Output of the RESEQUENCE algorithm with m=3400KB. The memory (in KB) required by ASR to process the grammar is shown in $< >$ against each dialog.

*separate* type, are represented by a dashed line in the call-flow and are extracted from the <must-separate> tag in the VXML file.

RESEQUENCE: For the input call-flow of figure 6(a), 6(b) and 6(c) show the output of RESEQUENCE for memory sizes $m = 70KB$ and $m = 2400KB$ respectively. While the first call-flow requires 9 questions, the second call-flow requires 6 questions to gather the same information. The decrease in the number of questions is a result of merging the corresponding grammars.

The $\langle m, q \rangle$ characteristics of the above call-flow is shown in Figure 7. Each bar in the chart corresponds to a unique call-flow. If the call-flow has only one question, its memory requirement is too big to be accommodated by any device. On the other extreme, the call-flow that has six questions can run on all devices. The value on the y-axis refers to the memory required to execute the largest grammar in the respective call-flow. The plot has been generated by running the RESEQUENCE algorithm on the call-flow mentioned in Figure 6(a) by varying $m$ and finding the corresponding $q$ values.

MASQ: Figure 8(a) shows an ideal call-flow corresponding to the call-flow shown in figure 6(a). The ideal call-flow, atomic call-flow, reorganisation constraints and the device's memory resources form an input to MASQ. The output of the algorithm is the optimal call-flow with minimum distance between the output and the ideal call-flow. Figure 8(b) shows a call-flow for $m = 29KB$. It requires 12 questions to be answered during the course of the call-flow execution.

The implication of the distance between two call-flows can be observed by comparing the output of RESEQUENCE and MASQ. With reference to the atomic call-flow described in figure 6(a) for $m = 29$, the output of the two algorithms would be different. RESEQUENCE can output a merged

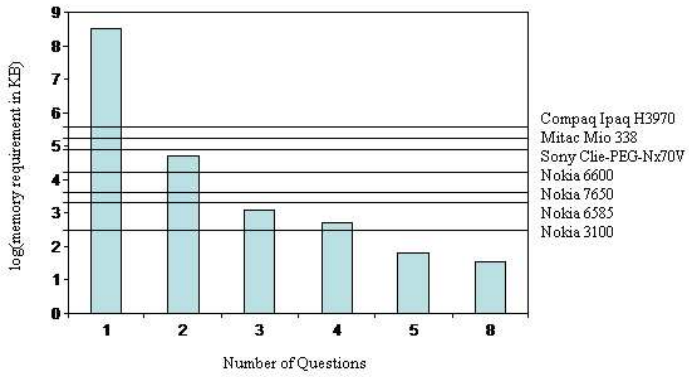


Fig. 7. The $\langle m, q \rangle$-characterisation of the dialog call-flow shown in Figure 6.

grammar corresponding to either 'Flight number' with 'Class of reservation' or 'Class of reservation' with 'Number of seats'. The output is such because the memory requirement for both the merged grammars is same. So RESEQUENCE algorithm can arbitrarily pick one of them. However in MASQ, the output call-flow would have a merged grammar of 'Class of reservation' and 'Number of seats'. This is because the resulting call-flow is at the least distance from the ideal reference call-flow 8(b).

### D. User Experiments

We ran an experiment using the reorganised airline reservation dialog call-flow with 14 users. The original call-flow had 9 questions and the reorganised call-flow had 6. The users were asked to rate the reorganised call-flow on a scale of 1–5 (5 being most satisfactory) for the number of questions, recognition accuracy and dialog completion time. Of the three merges in the reorganised call-flow, one of the merges resulted in poor recognition accuracy and led to decreased user satisfaction. The users were otherwise satisfied with the reorganised dialog.

## V. Summary and Future Work

We introduce, formulate and analyse device-specific adaptation of dialog call-flows and realise it in the form of SAMVAAD. We believe that the concept of $\langle m, q \rangle$*-characterisation* captures the essence of the adaptability of a call-flow and needs to be probed further for a clearer and quantifiable interpretation.

The reorganisation algorithms require an *atomic* call-flow as input. It would be interesting to know how these atomic call-flows can be automatically derived. One potential approach could be to *split* a grammar by exposing the intermediate non-terminals in the grammar. Another objective is the automatic generation of prompts for the merged/split grammars. This might require the use of Natural Language Processing techniques. Our user experiments have shown that we need to take into account the recognition accuracy before merging grammars. It would be nice to have a method for estimating the recognition accuracy of merged grammars.

More generally, the idea of extending call-flow adaptation for systems that use language models (rather than small

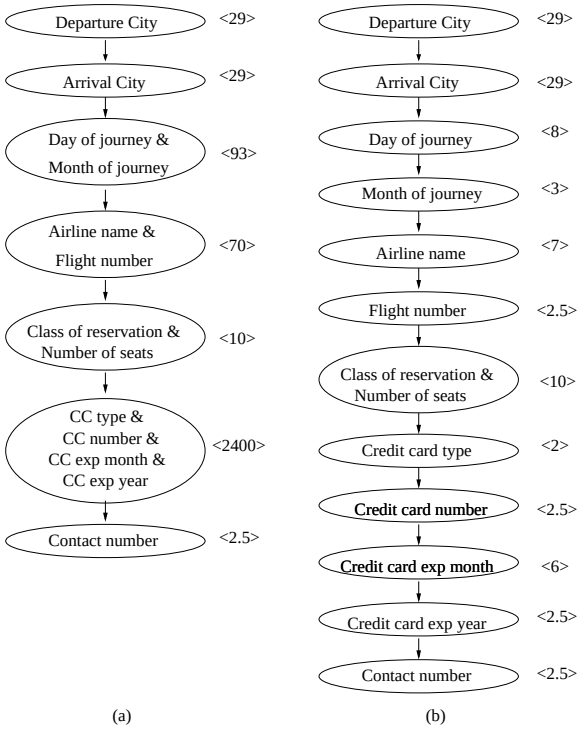


Fig. 8. (a) An ideal reference call-flow, (b) Output of MASQ for $m$=29KB.

"enumerated" grammars) coupled with an NLU engine would be interesting. Building a mechanism for adaptation in the absence of grammar operations appears to be a very challenging problem.

As speech applications become available on more and more devices, various interesting usability issues are likely to surface. Meeting the user expectation without having to manually customise a conversation for every person on every device is a worthy goal for the speech research community.

## References

[1] G. Banavar et.al., "Tooling and System Support for Authoring Multi-Device Applications," *Journal of Systems and Software*, 69(3), Jan 2004, pp. 227-242.
[2] E. Braun et. al., "Single Authoring for Multi-Device Interfaces," *Adjunct Proceedings of the 8th ERCIM Workshop: User Interfaces For All*, 2004.
[3] A. Jameson, "Adapting to the user's time and working memory limitations: New directions of research," *ABIS-98*, FORWISS.
[4] D. Litman and S. Pan, "Emperically Evaluating an Adaptable Spoken Dialogue System," *International Conference on User Modeling*, Banff, Canada,1999.
[5] P. Heisterkamp et.al., "Intelligent Dialog Overcomes Speech Technology Limitations: The SENECa Example," *ICIUI*, Miami, Florida, Jan 2003, pp. 267–269.
[6] E. Levin et.al., "A Stochastic Model of Human-Machine Interaction for Learning Dialog Strategies," *IEEE Transactions on Speech and Audio Processing*, 8(1), January 2000.
[7] G. N. Ramaswamy and P. S. Gopalakrishnan, "Compression of acoustic features for speech recognition in network environments," *ICASSP98*, Vol 2, pp 977–980.
[8] SRGS, W3C Recommendation, http://www.w3.org/TR/speech-grammar/
[9] VXML, W3C Recommendation, http://www.w3.org/TR/voicexml20/