Using Summarization for Automatic Briefing Generation

Inderjeet Mani
Kristian Concepcion
Linda Van Guilder

The MITRE Corporation, W640
11493 Sunset Hills Road
Reston, VA 22090, USA
{imani,kjc9,lcvg}@mitre.org

Abstract

We describe a system which automatically generates multimedia briefings from high-level outlines. The system uses summarization in content selection and creation, and in helping form a coherent narrative for the briefing. The approach does not require a domain knowledge base.

1 Introduction

Document production is an important function in many organizations. In addition to instruction manuals, reports, courseware, system documentation, etc., *briefings* are a very common type of document product, often used in slide form as a visual accompaniment to a talk. Since so much time is spent by so many people in producing briefings, often under serious time constraints, any method to reduce the amount of time spent on briefing production could yield great gains in productivity.

Briefings involve a high degree of condensation of information (e.g., no more than a few points, perhaps bulleted, per slide), and they typically contain multimedia information. Many briefings have a stereotypical structure, dictated in part by the business rules of the organisation. For example, a commander may present a daily or weekly brief to her superiors, which is more in the nature of a routine update of activities since the last briefing; or she may provide an action brief, which is triggered by a particular situation, and which consists of a situation update followed by arguments recommending a particular course of action. Further, the process of constructing a briefing may involve certain stereotypical activities, including culling information from particular sources, such as messages, news, web pages, previous briefings, etc. Thus, while part of the briefing content may be created anew by the briefing author[1], other parts of the briefing may be constructed from existing information sources. However, information in those sources need not necessarily be in the same form as needed by the briefing.

All these characteristics of briefings make them attractive as an application of automatic summarization, which is aimed at producing a condensed, task-tailored representation of salient content in information sources. Often, the background information being used in a slide is quite considerable; the author needs to identify what's salient, presenting it in a succinct manner so as to fit on the slide, perhaps creating a graphic or other multimedia clip to do so. Automatic summarization, by definition, has a clear role to play here. A briefing usually involves a sequence of slides; as the summary becomes longer, it needs to form a coherent narrative, built around the prescribed structure. Finally, a briefing must strive, to the extent possible, to be persuasive and vivid, so that the point gets across. This in turn presents a further challenge for summarization: the ability to generate smoothly narrated, coherent summaries.

[1] The noun "author" is used throughout the paper to designate a human author.

It is therefore worthwhile investigating whether combining automatic summarization with intelligent multimedia presentation techniques can make the briefing generation amenable to full automation. In other words, the author should be able to use a computer program to generate an initial briefing, which she can then edit and revise as needed. The briefing can then be presented by the author if desired, or else directly by the computer (particularly useful if the briefing is being sent to someone else). The starting point for this process would be a high-level outline of the briefing on the part of the author. The outline would include references to particular information sources that had to be summarized in particular ways. If a program were able to take such outlines and generate briefings which didn't require extensive post-editing to massage into a state deemed acceptable for the task at hand, the program could be regarded as a worthwhile time saving tool.

2 Approach

Our work forms part of a larger DARPA-funded project aimed at improving analysis and decision-making in crisis situations by providing tools that allow analysts to collaborate to develop structured arguments in support of particular conclusions and to help predict likely future scenarios. These arguments, along with background evidence, are packaged together as briefings to high-level decision-makers. In leveraging automatic methods along the lines suggested above to generate briefings, our approach needs to allow the analyst to take on as much of the briefing authoring as she wants to (e.g., it may take time for her to adapt to or trust the machine, or she may want the machine to present just part of the briefing). The analyst's organisation usually will instantiate one of several templates dictating the high-level structure of a briefing; for example, a briefing may always have to begin with an executive summary. The summarization methods also need to be relatively domain-independent, given that the subject matter of crises are somewhat unpredictable; an analyst in a crisis situation is likely to be inundated with large numbers of crisis-related news and intelligence reports from many different sources. This means that we cannot require that a domain knowledge base be available to help the briefing generation process.

Given these task requirements, we have adopted an approach that is flexible about accommodating different degrees of author involvement, that is relatively neutral about the rhetorical theory underlying the briefing structure (since a template may be provided by others), and that is domain-independent. In our approach, the author creates the briefing outline, which is then fleshed out further by the system based on information in the outline. The system fills out some content by invoking specified summarizers; it also makes decisions, when needed, about output media type; it introduces narrative elements to improve the coherence of the briefing; and finally, it assembles the final presentation, making decisions about spatial layout in the process.

A briefing is represented as a tree. The structure of the tree represents the rhetorical structure of the briefing. Each node has a label, which offers a brief textual description of the node. Each leaf node has an associated goal, which, when realized, provides content for that node. There are two kinds of goals: *content-level* goals and *narrative-level* goals. Content-level goals are also of two kinds: *retrieve* goals, which retrieve existing media objects of a particular type (text, audio, image, audio, video) satisfying some description, and *create* goals, which create new media objects of these types using programs (called *summarization filters*). Narrative-level goals introduce descriptions of content at other nodes: they include captions and running text for media objects, and *segues*, which are rhetorical moves describing a transition to a node.

Ordering relations reflecting temporal and spatial layout are defined on nodes in the tree. Two coarse-grained relations, *seq* for precedence, and *par* for simultaneity, are used to specify a temporal ordering on the nodes in the tree. As an example, temporal constraints for a (tiny) tree of 9 nodes may be expressed as:

```
<ordering> <seq>
        <par>7</par>
        <par>8</par>
        <par>3</par>
        <par>4  5</par>
        <par>6</par>
```

<par>1 9</par>
<par>2</par>
</seq> </ordering>

The tree representation, along with the temporal constraints, can be rendered in text as XML; we refer to the XML representation as a *script*.

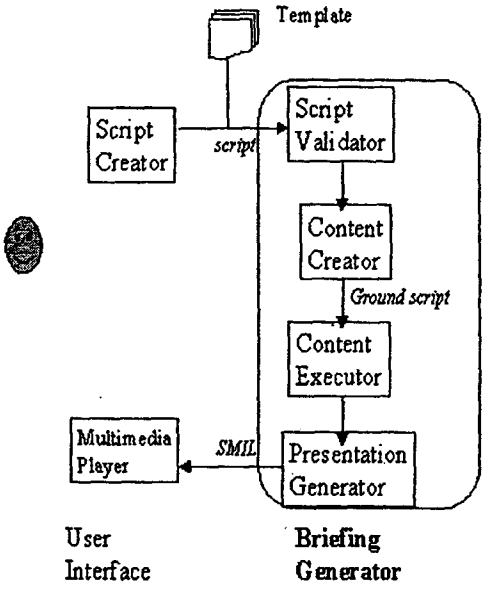


Figure 1: System Architecture

The overall architecture of our system is shown in Figure 1. The user creates the briefing outline in the form of a script, by using a GUI. The briefing generator takes the script as input. The *Script Validator* applies an XML parser to the script, to check for syntactic correctness. It then builds a tree representation for the script, which represents the briefing outline, with temporal constraints attached to the leaves of the tree.

Next, a *Content Creator* takes the input tree and expands it by introducing narrative-level goals including segues to content nodes, and running text and captions describing media objects at content nodes. Running text and short captions are generated from meta-information associated with media objects, by using shallow text generation methods (canned text). The end result of content selection (which has an XML representation called a *ground* script) is that the complete tree has been fully specified, with all the *create* and *retrieve* goals fully specified, with all the output media types decided. The Content Creator is thus responsible for both content selection and creation, in terms of tree structure and node content.

Then, a *Content Executor* executes all the *create* and *retrieve* goals. This is a very simple step, resulting in the generation of all the media objects in the presentation, except for the audio files for speech to be synthesized. Thus, this step results in realization of the content at the leaves of the tree.

Finally, the *Presentation Generator* takes the tree which is output from Content Execution, along with its temporal ordering constraints, and generates the spatial layout of the presentation. If no spatial layout constraints are specified (the default is to not specify these), the system allocates space using a simple method based on the temporal layout for nodes which have spatial manifestations. Speech synthesis is also carried out here. Once the tree is augmented with spatial layout constraints, it is translated by the Presentation Generator into SMIL[2] (Synchronized Multimedia Integration Language) (SMIL 99), a W3C-developed extension of HTML that can be played by standard multimedia players (such as Real[3] and Grins[4]. This step thus presents the realized content, synthesizing it into a multimedia presentation laid out spatially and temporally.

This particular architecture, driven by the above project requirements, does not use planning as an overall problem-solving strategy, as planning requires domain knowledge. It therefore differs from traditional intelligent multimedia presentation planners, e.g., (Wahlster et al. 93). Nevertheless, the system does make a number of intelligent decisions in organizing and coordinating presentation decisions. These are discussed next, after which we turn to the main point of the paper, namely the leveraging of summarization in automatic briefing generation.

[2] http://www.w3.org/AudioVideo/

[3] www.real.com

[4] www.oratrix.com

3 Intelligent Multimedia Presentation Generation

The author of a briefing may choose to flesh out as little of the tree as desired, with the caveat that the temporal ordering relations for non-narrative nodes need to be provided by her. When a media object is generated at a node by a *create* goal, the running text and captions are generated by the system. The motivation for this is obvious: when a summarization filter (which is a program under our control) is generating a media object, we can often provide sufficient meta-information about that object to generate a short caption and some running text. By default, all segues and spatial layout relations are also specified by the system, so the author does not have to know about these unless she wants to. Finally, the decision as to when to produce audio, when not specified by the author, is left to the system.

When summarization filters are used (for *create* goals), the media type of the output is specified as a parameter to the filter. This media type may be converted to some other type by the system, e.g., text to speech conversion using Festival (Taylor et al. 98). By default, all narrative nodes attempt to realize their goals as a speech media type, using rules based on text length and truncatability to less than 250 bytes to decide when to use text-to-speech. The truncation algorithm is based on dropping syntactic constituents, using a method similar to (Mani et al. 99). Captions are always realized, in addition, as text (i.e., they have a text realization and a possible audio realization).

Spatial layout is decided in the Presentation Generator, after all the individual media objects are created along with their temporal constraints by the Content Executor. The layout algorithm walks through the temporal ordering in sequence, allocating a segment to each set of objects that is designated to occur simultaneously (grouped by *par* in the temporal constraints). Each segment can have up to 4 frames, in each of which a media object is displayed (thus, no more than 4 media objects can be displayed at the same time). Since media objects declared to be simultaneous (using *par*) in the temporal constraints will go together in a separate segment, the temporal constraints determine what elements are grouped together in a segment. The layout within a segment handles two special cases. Captions are placed directly underneath their associated media object. Running text, when realized as text, is placed beside the media object being described, so that they are paired together visually. Thus, coherence of a segment is influenced mainly by the temporal constraints (which have been fleshed out by the Content Creator to include narrative nodes), with further handling of special cases. Of course, an individual summarization filter may choose to coordinate component multimedia objects in particular ways in the course of generating a composite multimedia object.

Details such as duration and onset of particular frames are specified in the translation to SMIL. Duration is determined by the number of frames present in a segment, unless there is an audio media object in the segment (this media object may have a spatial representation, e.g., as an audio icon, or it may not). If an audio media object occurs in a frame, the duration of all media objects in that frame is equal to the length of all the audio files in the segment. If there is no audio present in a segment, the duration is α seconds (α has a default value of 5) times the number of frames created.

4 Summarization Filters

As mentioned above, *create* goals are satisfied by summarization filters, which create new media objects summarizing information sources. These programs are called *summarization filters* because in the course of condensing information, they take input information and turn it into some more abstract and useful representation, filtering out unimportant information. Such filters provide a novel way of carrying out content selection and creation for automated presentation generation.

Our approach relies on component-based software composition, i.e., assembly of software units that have contractually specified interfaces that can be independently deployed and reused. The idea of assembling complex language processing programs out of simpler ones is

hardly new; however, by employing current industry standards to specify the interaction between the components, we simultaneously increase the robustness of the system, ensure the reusability of individual components and create a more fully plug-and-play capability. Among the core technology standards that support this plug-and-play component assembly capability are (a) Java interfaces, used to specify functions that all summarization components must implement in order to be used in the system, (b) the JavaBeans standard, which allows the parameters and methods of individual components to be inspected by the system and revealed to the users (c) the XML markup standard, which we have adopted as an inter-component communication language. Using these technologies, legacy or third-party summarizers are incorporated into the system by "wrapping" them so as to meet the interface specification of the system. These technologies also make possible a graphical environment to assemble and configure complex summarization filters from individual summarization components.

Among the most important wins over the traditional "piping" approach to filter assembly is the ability to impose build-time restrictions on the component assembly, disallowing "illegal" compositions, e.g. component X cannot provide input to component Y unless X's output type corresponds to Y's input type. Build-time restrictions such as these play a clear role in increasing the overall robustness of the run-time summarization system. Another build-time win lies in the ability of JavaBeans to be serialized, i.e., written to disk in such a way as to preserve the state of its parameters settings, ensuring that every component in the system can be configured and run at different times independently of whether the component provides a parameter file facility.

Establishing the standard functions required of a summarization filter is challenging on several fronts. One class of functions required by the interface is necessary to handle the technicalities of exchanging information between otherwise discrete components. This set includes functions for discovering a component's input and output types, for handling messages, exceptions and events passed between components and for interpreting XML based on one or more system-wide document type definitions (DTDs). The other, more interesting set of functions gets to the core of summarization functionality. Selecting these functions involves identifying parameters likely to be broadly applicable across most or all summarizers and finding ways to group them and/or to generalize them. This is desirable in order to reduce the burden on the end user of understanding the subtle differences between the various settings in the summarizers available to her.

An example of the difficulty inherent in this endeavor is provided by the *compression* (summary length divided by source length) vs. *reduction* (1's complement of compression) vs. *target length* paradigm. Different summarizers will implement one or more of these. The wrapper maps from the high-level interface function, where the application/user can specify either compression or target length, but not both, to the individual summarizer's representation. Thus, a user doesn't need to know which representation(s) a particular summarizer uses for reduction/compression.

A vanilla summarization Bean includes the following functionality, which every summarizer must be able to provide methods for:

> *source*: documents to be summarized (this can be a single document, or a collection)
> *reduction-rate*: either summary size/source size, or target length
> *audience*: user-focused or generic (user-focused requires the specification of a bag of terms, which can be of different types)
> *output-type*: specific data formats (specified by DTDs)

The above are parameters which we expect all summarizers to support. More specialized summarizer beans can be constructed to reflect groupings of summarizers. Among other parameters are *output-fluency*, which specifies whether a textual summary is to be made up of passages (sentences, paras, blocks), named entities, lists of words, phrases, or topics, etc. Given that definitions of summarization in more

theoretical terms have not been entirely satisfactory (Mani 2000), it is worth noting that the above vanilla Bean provides an *operational definition* of what a summarizer is.

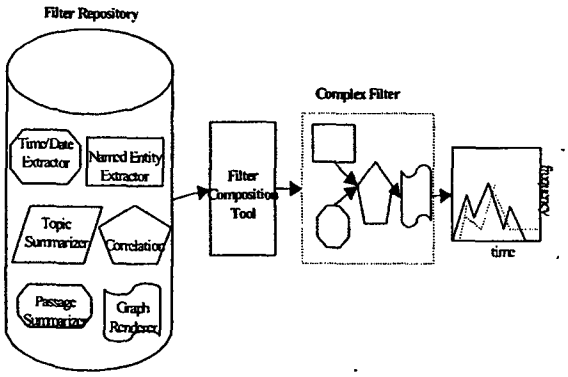


Figure 2: Summarization Filter Composition

In addition to its practical utility in the ability to assimilate, combine and reuse components in different combinations, and to do so within a GUI, this approach is interesting because it allows powerful summarization functions to be created by composing together simpler tools. (Note that this is different from automatically finding the best combination, which our system does not address). For example, Figure 2 illustrates a complex filter created by using a GUI to compose together a named entity extractor, a date extractor, a component which discovers significant associations between the two and writes the result to a table, and a visualizer which plots the results as a graph. The resulting summarizer takes in a large collection of documents, and produces as a summary a graph (a jpeg) of salient named entity mentions over time. Each of its components can be easily reused within the filter composition system to build other summarizers.

5 Narrative Summarization

As mentioned above, the system can construct a narrative to accompany the briefing. Narrative nodes are generated to cover captions, running text, and segues. The captions and running text, when not provided by the filters, are provided by the script input. In the case of *retrieve* goals, the objects may not have any meta-information, in which case a default caption and running-text is generated. Clearly, a system's explanatory narrative will be enhanced by the availability of rich meta-information.

The segues are provided by the system. For example, an item with a label "A biography of bin Laden" could result in a generated segue *"Here is a biography of bin Laden"*. The Content Creator, when providing content for narrative nodes, uses a variety of different canned text patterns. For the above example, the pattern would be *"Here is @6.label"*, where *6* is the number of a non-narrative node, with *label* being its label.

Peru Action Brief
1 Preamble
2 Situation Assessment
2.1 Chronology of Events
2.1.2 Latest document summary
create("*summarize -generic -compression .1 /peru/p32*")
2.2 Biographies
2.2.1 Biography of Victor Polay
2.2.1.1 Picture of *@2.2.2.person*
retrieve("D:\rawdata\polay.jpg")
2.2.1.2 Biography of *@2.2.2.person*
create("*summarize -bio -length 350 -span multi -person @2.2.2.person -out table /peru/**")
3 Coda
"This briefing has assessed aspects of the situation in Peru. Overall, the crisis appears to be worsening."

Figure 3: Input Script

```
Peru Action Brief
  1 Preamble
    audio = "In this briefing, I will go over
             the @2.label. This will cover
             @2.1.label and @2.3.1.label"
  2 Situation Assessment
    2.1 "An overview of the @2.2.label"
        (Meta-2.2)
    2.2 Chronology of Events
      2.2.1 audio = "Here is the @2.2.2.label"
            (Meta-2.2.2)
      2.2.2 text = "Latest document summary"
        audio = text =
        create("summarize -generic
                -compression .1 /peru/p32")
    2.3 Biographies
      2.3.1 audio =
            "A profile of @2.3.2.person"
            (Meta-2.3.2)
      2.3.2 Biography of Victor Polay
        2.3.2.1 audio = text =
                "A file photo of
                @2.3.2.person"
                (Meta-2.3.2.2)
        2.3.2.2 Picture of @2.3.2.person
          image =
          retrieve("D:\rawdata\polay.jpg")
        2.3.2.3 audio = text =
                "Profile of @2.3.2.person"
                (Meta-2.3.2.3)
        2.3.2.4 Biography of @2.3.2.person
          audio = text =
          create("summarize -bio -length 350
                  -span multi -person
                  @2.2.2.person -out table
                  /peru/*")

  3 Coda
    audio = "This briefing has assessed
    aspects of the situation in Peru. Overall,
    the crisis appears to be worsening."
```

```
<seq>
      <par>1</par>
      <par>2.2.1 2.2.2</par>
      <par>2.3.1</par>
      <par>2.3.2.1  2.3.2.2
           2.3.2.3  2.3.2.4</par>
      <par>3</par>
</seq>
```

Figure 4: Ground Script

All segue nodes are by default generated automatically by the system, based on node labels. We always introduce a segue node at the beginning of the presentation (called a preamble node), which provides a segue covering the "crown" of the tree, i.e., all nodes upto a particular depth d from the root (d=2) are marked with segue nodes. A segue node is also produced at the end (called a coda). (Both preamble and segue can of course be specified by the author if desired).

For introducing intervening segue nodes, we use the following algorithm based on the distance between nodes and the height in the tree. We traverse the non-narrative leaves of the tree in their temporal order, evaluating each pair of adjacent nodes A and B where A precedes B temporally. A segue is introduced between nodes A and B if either (a) the maximum of the 2 distances from A and B to their least common ancestor is greater than 3 nodes or (b) the sum of the 2 distances from A and B to the least common ancestor is greater than 4 nodes. This is less intrusive than introducing segues at random or between every pair of successive nodes, and appears to perform better than introducing a segue at each depth of the tree.

6 An Example

We currently have a working version of the system with a variety of different single and multi-document summarization filters. Figure 3 shows an input script created by an author (the scripts in Figure 3 and 4 are schematic representations of the scripts, rather than the raw XML). The script includes two create goals, one with a single-document generic summarization filter, the other with a multi-document user-focused summarization filter. Figure 4 shows the ground script which was created automatically by the Content Creator component. Note the addition of media type specifications, the introduction of narrative nodes, and the extension of the temporal constraints. The final presentation generated is shown in Figure 5. Here we show screen dumps of the six SMIL segments produced, with the audio if any for each segment indicated in this paper next to an audio icon.

7 Status

The summarization filters have incorporated several summarizers, including some that have been evaluated in the DARPA SUMMAC conference (Mani et al. 99-1). These carry out both single-document and multi-document summarization, and include a preliminary biographical summarizer we have developed. The running text for the biography table in the second-last segment of Figure 5 is produced from meta-information in the table XML generated by the biographical summarizer. The production method for running text uses canned text which should work for any input table conforming to that DTD.

The summarization filters are being tested as part of a DARPA situated test with end-users. The briefing generator itself has been used internally to generate numerous briefings, and has been demonstrated as part of the DARPA system. We also expect to carry out an evaluation to assess the extent to which the automation described here provides efficiency gains in briefing production.

8 Related Work

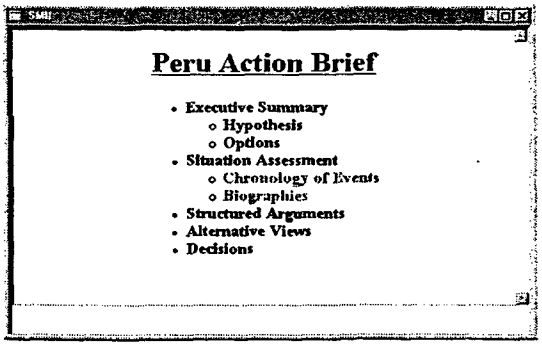
There is a fair amount of work on automatic authoring of multimedia presentations, e.g., (Wahlster et al. 93), (Dalal et al. 96), (Mittal et al. 95), (Andre and Rist 97)[5]. These efforts differ from ours in two ways: first, unlike us, they are not open-domain; and, second, they don't use summarization components. While such efforts are extremely sophisticated compared to us in multimedia presentation planning and fine-grained coordination and synchronization capabilities, many of the components used in those efforts are clearly applicable to our work. For example, (Andre and Rist 96) include methods for leveraging lifelike characters in this process; these characters can be leveraged in our work as well, to help personify the computer narrator. In addition, our captions, which are very short, rely on canned text based on node labels in the initial script, or based on shallow meta-information generated by the summarization filter (in XML) along with the created media object. (Mittal et al. 95) describe a variety of strategies for generation of longer, more explanatory captions, some of which may be exploited in our work by deepening the level of meta-information, at least for summarization components developed by us.

In our ability to leverage automatic summarization, our work should be clearly distinguished from work which attempts to format a summary (from an XML representation) into something akin to a Powerpoint briefing, e.g., (Nagao and Hasida 98). Our work, by contrast, is focused on using summarization in generating briefings from an abstract outline.

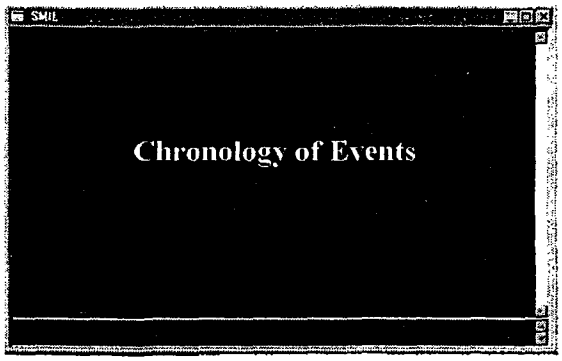
9 Conclusion

We have described methods for leveraging automatic summarization in the automatic generation of multimedia briefings. This work has taken an open-domain approach, in order to meet the requirements of the DARPA application we are involved with. We believe there is a stronger role that NL generation can play in the narrative aspects of our briefings, which currently rely for the most part on canned text. Our future work on description merging in biographical summaries, and on introducing referring expressions into the narrative nodes, would in effect take advantage of more powerful generation methods, without sacrificing open-domain capabilities. This may require much richer meta-information specifications than the ones we currently use.

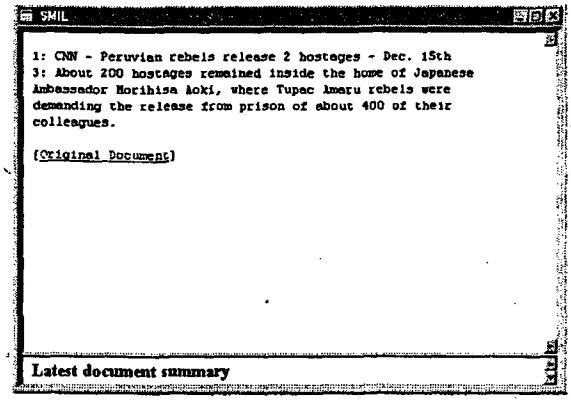
Finally, we have begun the design of the Script Creator GUI (the only component in Figure 1 remaining to be built). This will allow the author to create scripts for the briefing generator (instead of editing templates by hand), by laying out icons for media objects in temporal order. A user will be able to select a "standard" briefing template from a menu, and then view it in a briefing/template structure editor. The user can then provide content by adding annotations to any node in the briefing template. The user will have a choice of saving the edit version in template form, or in SMIL or possibly Microsoft Powerpoint format.



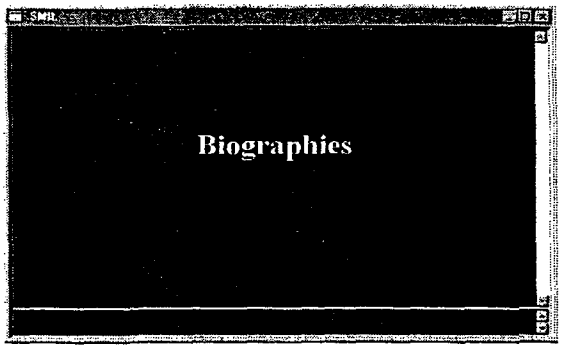
In this briefing I will go over the situation assessment. This will cover an overview of the chronology of events and a profile of Victor Polay.



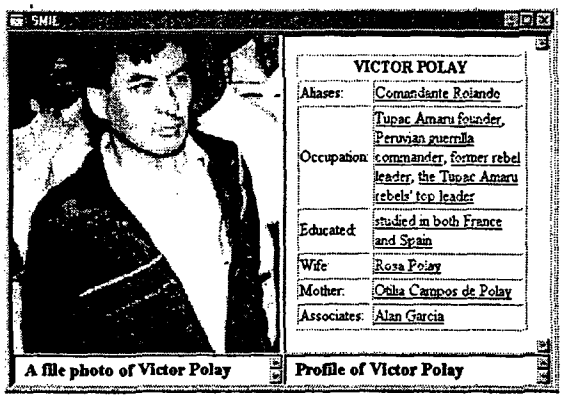
Here is an overview of the chronology of events.



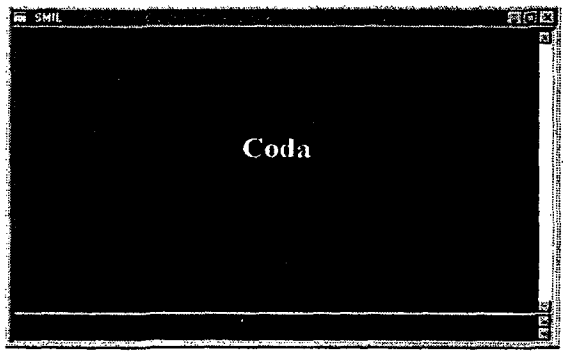
Here is the latest document summary.



Next, a biography of Victor Polay.



Victor Polay, also known as Comandante Rolando, is the Tupac Amaru founder, a Peruvian guerrilla commander, a former rebel leader, and the Tupac Amaru rebels' top leader. He studied in both France and Spain. His wife is Rosa Polay and his mother is Otilia Campos de Polay. His associates include Alan Garcia.



This briefing has assessed aspects of the situation in Peru. Overall, the crisis appears to be worsening.

Figure 5: Presentation

References

Andre, E. and Rist, T. (1997) *Towards a New Generation of Hypermedia Systems: Extending Automated Presentation Design for Hypermedia.* L. Dybkjaer, ed., Proceedings of the Third Spoken Dialogue and Discourse Workshop, Topics in Natural Interactive Systems 1. The Maersk Mc-Kinney Moller Institute for Production Technology, Odense University, Denmark, pp. 10-27.

Dalal, M., Feiner, S., McKeown, K., Pan, S., Zhou, M., Hollerer, T., Shaw, J., Feng, Y., and Fromer, J. (1996) *Negotiation for Automated Generation of Temporal MultimediaPresentations.* Proceedings of ACM Multimedia '96.

Mani, I., Gates, B., and Bloedorn, E. (1999) *Improving Summaries by Revising Them.* Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics, College Park, MD, pp. 558-565.

Mani, I., Firmin, T., House, D., Klein, G., Sundheim, B., and Hirschman, L. (1999) *The TIPSTER SUMMAC Text Summarization Evaluation.* Proceedings of EACL'99, Bergen, Norway, pp. 77-85.

Mani, I. (2000) *Automatic Text Summarization.* John Benjamins Publishing Company. To appear.

Mittal, V., Roth, S., Moore, J., Mattis, J., and Carenini, G. (1995) *Generating Explanatory Captions for Information Graphics.* Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI'95), pp. 1276-1283.

Nagao, K. and K. Hasida, K. (1998) *Automatic Text Summarization Based on the Global Document Annotation.* Proceedings of COLING'98, Montreal, pp. 917-921.

Power, R. and Scott, D. (1998) *Multilingual Authoring using Feedback Texts.* Proceedings of COLING'98, Montreal, pp. 1053-1059.

Taylor, P., Black, A., and Caley, R. (1998) *The architecture of the Festival Speech Synthesis System.* Proceedings of the Third ESCA Workshop on Speech Synthesis, Jenolan Caves, Australia, pp. 147-151.

Wahlster, W., Andre, E., Finkler, W., Profitlich, H.-J., and Rist, T. (1993) *Plan-Based Integration of Natural Language and Graphics Generation.* AI Journal, 63.