Mobile Information Systems 4 (2008) 195–209
IOS Press

GAIML: A new language for verbal and graphical interaction in chatbots

Roberto Pirrone*, Giuseppe Russo, Vincenzo Cannella and Daniele Peri
DINFO, Universitá degli Studi Palermo, Palermo, Italy

Abstract. Natural and intuitive interaction between users and complex systems is a crucial research topic in human-computer interaction. A major direction is the definition and implementation of systems with natural language understanding capabilities. The interaction in natural language is often performed by means of systems called chatbots. A chatbot is a conversational agent with a proper knowledge base able to interact with users. Chatbots appearance can be very sophisticated with 3D avatars and speech processing modules. However the interaction between the system and the user is only performed through textual areas for inputs and replies. An interaction able to add to natural language also graphical widgets could be more effective. On the other side, a graphical interaction involving also the natural language can increase the comfort of the user instead of using only graphical widgets. In many applications multi-modal communication must be preferred when the user and the system have a tight and complex interaction. Typical examples are cultural heritages applications (intelligent museum guides, picture browsing) or systems providing the user with integrated information taken from different and heterogenous sources as in the case of the iGoogle™ interface. We propose to mix the two modalities (verbal and graphical) to build systems with a reconfigurable interface, which is able to change with respect to the particular application context. The result of this proposal is the Graphical Artificial Intelligence Markup Language (GAIML) an extension of AIML allowing merging both interaction modalities. In this context a suitable chatbot system called Graphbot is presented to support this language. With this language is possible to define personalized interface patterns that are the most suitable ones in relation to the data types exchanged between the user and the system according to the context of the dialogue.

Keywords: Dynamic interface generation, chatbot, pattern definition, GAIML

1. Introduction

The design of natural and intuitive interaction modalities is a major goal of the research in human-computer interaction. In particular, many efforts have been devoted in the development of dialogue systems. When the dialogue deals with abstract topics, it could be hardly represented by an arrangement of graphical widgets. An interaction involving also the natural language could increase the comfort of the user. However, in many cases words are not enough. As an example, a picture carries a lot of information that is strictly related to its content (characters, painting style, and so on). A textual description is not suitable to completely render the content of an image. As an example, often visitors in a museum are invited to listen to or to touch the items in the exhibition thus having a multi-sensorial experience. A multi-modal interactive museum guide could compare an artwork in the museum with another one that is far from the user position or even located in another gallery. In this case an effective exposition needs graphical supports.

*Corresponding author: Roberto Pirrone, Viale delle Scienze Ed., 6 P. 3, 90128 Palermo, Italy. Tel.: +39 0917028525; E-mail: pirrone@unipa.it.

A common way in HCI to have natural language interaction with the user is a chatbot system that is a conversational agent. The simplest chatbot is just a text field where the user inputs her sentences and a text area for the system replies. The interaction is not attractive for the user because the focus of the system is about the understanding process. The research on chatbots explores mainly the problems related to dialogue management. Natural language dialogue is related to several issues like using common sense knowledge, or managing the context. Another research field is related to graphical extensions. In particular, many efforts have been made to produce a virtual humanoid talker. Such systems have a voice based interface or a 3D anthropomorphic conversation partner (the "avatar"). This is a reply to the users' needs of systems able to somehow relate to them but more than the realistic aspect the system has to perform complex interaction processes. Another aspect is related to WIMP interface usage. For instance, when the user must make a choice from a long list of options, a list of check boxes or combo boxes is simpler and more expressive than a verbal request. These widgets allow the user to see the options carefully without a cognitive overload. The design of a dialogue must take care of all these aspects. Usually, the design of graphical interfaces is inspired to software building paradigms like the Model-View-Control (MVC) [1]. The fusion of these two aspects is a way to define a better HCI process. Furthermore, we claim that an integrated interface merging both linguistic and graphical interaction is a solution to reduce the overall complexity of the whole process. Pure verbal interaction can be enriched by the use of multi-modal widgets that can make the communication process simpler and more intuitive. As a consequence, dynamic interfaces that are suited to instantaneous user needs and are not biased towards architectural constraints could be obtained.

The present work presents the Graphical Artificial Intelligence Markup Language (GAIML), a language able to describe multi-modal interaction patterns. Moreover, GraphBot is described, which is a system implementation for automatic GUI generation using GAIML. In GraphBot, linguistic and graphical interaction modalities are combined in response to the evolution of the dialogue between the user and the machine. We instantiate or produce particular GUI patterns according to the evolution of the dialogue managed by a chatbot engine. Each GUI pattern is designed using the MVC approach. In this paradigm the designer takes care only of the data types exchanged between the user and the system. Another aspect is the direction of the information flow. The chatbot engine provides integration between different patterns, which are regarded as atomic moves inside the whole dialogue.

The rest of the paper is arranged as follows: Section 2 presents a state of the art for the automatic creation of user interfaces and for chatbots systems. Then, the AIML language is presented in Section 3. Section 4 describes GAIML. Section 5 introduces GraphBot. Finally, Section 6 draws conclusions and explains future work.

2. State of the art

Many tools for automatic interface building have been proposed in literature. A first example is Jade, presented in [17]. This tool can create and lay out automatically GUIs. The specification of interfaces does not depend on the data model of the application. The user defines directly the structure and the components of the interface. This can be only a partial solution. In this case, the developer has not to define entirely the interface, but she must instruct the system in the construction of the interface. Mastermind [20] supports the automatic construction of user interfaces from declarative models describing the components of the GUI or their behaviors, reserving no attention to the data structure. In other systems the interface generation is based mainly on the analysis of the data model. The Taellach system [18] and the Janus system [14] have been inspired to the object oriented paradigm.

In GENIUS [15] the interface is generated describing the data through an entity-relation model, while the dialogue is described through the “dialogue nets" that are particular Petri nets useful in describing the flow of the interaction. Sometimes the interface has to supply some data to the user. In this case, the problem is the arrangement of the content inside the window. Xadaptor [30] is a content adaption system that is able to define new interfaces by means of rule processing. In [13] a web service for interface composition and adaption is presented. Both systems are developed using XML manipulation for the interface generation and a set of on-the-fly rules for interface adaption. In other systems the interface generation is based mainly on the interaction task analysis. TRIDENT [16] generates the interface starting from the activity chaining graph, which describes the activity flow of the task. Each step of the task is associated to an abstract interaction object (AIO) that is an abstract description of a possible component of the interface. An AIO describes presentation and behavior of an interface component, and doesn't take care of the target environments. During the construction of the interface the system chooses the most suitable AIOs for the task. The selected AIOs are then translated into concrete interaction objects (CIOs) the tru implementation of the interface component in a programming language. Recently, GUI generation tools based on mark-up languages are gaining in interest. Such tools make use of the languages used in web applications, like XML and XSLT. An example of using XSLT to convert a XML data file into a Java Swing GUI is reported in [22]. Another example is offered by XUL [33], a cross-platform markup language allowing to describe the components and the structure of an interface. As in Jade, the specification of the interface does not depend on the data model. XIML [24] is also a XML-based language, but it enables a programmer to describe both GUI elements and data structure. Another example is UIML [34], a XML language for a device- and modality-independent user interface design. UIML source can be translated into a concrete interface implemented in a particular programming language such as Java Swing, HTML, VoiceXML. Some systems use a knowledge base describing the rules and the criteria that guide the interface building process. A first significant example is UIDE, presented in [35]. This system has a complex knowledge base describing manny different aspects, as the interface components, the actions of the interface, the parameters, the pre- and post-conditions of the different steps of the task managed by the interface. All these data are structured as frames, and are arranged in taxonomies. The system generates the interface following a collection of logical rules. Another example is MECANO [36]. It is a global project including the development of a formal language for interface description. Another example is the Set Description Language (SDL) that has been developed by some of the authors [25]. In the SDL framework, GUI generation is performed by means of a set of rules expressed in first order logic. At first, a GUI is described at the highest level of abstraction through the data it operates on, and it is generated in a way that enforces data correctness. The system has been implemented in Prolog and is based on the Model-View-Control (MVC) software design paradigm, which inspired the design of the system presented in this work.

As regards multi-modal interfaces, many markup languages have been proposed to define them. The main purpose is to build form based web applications. Nowadays, web applications provide as many functionalities as desktop applications, with additional remote capability and platform/device independency. As defined in [2] major goals to achieve are the efficiently input-output data passing, the utilization of different sets of clients and usability for users. As an example XForms [3] is a markup language that minimizes the need of scripting by allowing dynamic form data and events to be handled declaratively. It could be embedded in other languages and it is interpreted client side. These languages have been also employed for programming chatbots. A chatbot is a an intelligent system that is able to manage a natural language conversation with the user. It is often a stimulus-response system, and it's implemented as a finite state machine. One of the first and most famous multimodal interaction languages

```
<aiml:aiml version 1.01 xmlns:aiml = "http://alicebot.org/2001/AIML">
<aiml:topic name =".."> </topic>
<aiml category>
    <aiml pattern> ..
    </aiml pattern>
    <aiml template>..
    </aiml template>
</aiml category>
</aiml>
```

Fig. 1. An example of AIML structure.

was the Multimodal Presentation Markup Language (MPML) [4]. MPML is an XML dialect providing controls for the verbal and non-verbal behavior of affective 2D cartoon-style characters. Moreover MPML owns primitives to manage the presentation flow, and the integration of external objects, like Java applets. MPML supports the generation of multimodal presentations, including 2D graphics and spoken (synthetic) language, music, and video. Besides synthetic speech, animated characters may communicate information by using multiple modalities, such as facial displays, hand gestures, head movements, and body posture. A derivation of MPML was the Dynamic Web Markup Language (DWML). This language allows to define WIMP interfaces, but is no more supported by the group which has developed MPML.

3. AIML

A language to program chatbots is the Artificial Intelligence Markup Language (AIML) [5]. AIML is an XML-based language [6], and is used mainly for the definition of the knowledge base of chatbots. It allows to define single interactions. An interaction is composed by a stimulus produced by the user, and a corresponding response of the system. Both stimulus and response are coded in the `<category>` element (see Fig. 1). In turn, `<category>` contains two elements: `<pattern>` (stimulus) and `<template>` (response). Sometimes the `<template>` element is nested into the `<that>` element. This element allows to manage the context in a conversation.

An AIML interpreter can use directly the content of the `<template>` element to build the response. The content of a `<template>` element could refer to other categories. In this case, the response of the system is obtained combining the content of the involved categories together. One or more `<category>` elements can be grouped inside an optional `<topic>` element. When the user gives an input, the AIML interpreter tries to match this input with one pattern. The matching process consists of many steps. At first, the AIML interpreter tries to match the content of the `<category>` elements. A second match is tried on the pattern from the special `<category>` elements that are allowed in the language, if present. Finally, a match is tried with the `name` attribute in the `<topic>` element thus allowing to distinguish the topic of a conversation, if present.

The most famous AIML programmable chatbot is A.L.I.C.E. [7]. A.L.I.C.E. is an artificial intelligence natural language chat robot based on the famous test proposed by Alan M. Turing in 1950. In our work, we started from the A.L.I.C.E. software architecture to develop the GraphBot system.

4. The graphical artificial intelligence markup language

Usual chatbot interfaces are static, and the role of their widgets is fixed. Adaptive interfaces changing their arrangement on the basis of the dialogue context improve significantly the interaction and reduce the cognitive overload for the user. This is a well known finding in HCI because humans are very

clever in managing multi-modal information [8]. GAIML is aimed to enrich AIML with the ability to manage <pattern> and <template> elements, which contain information about GUI building. Three different situations have been devised. A GUI can be explicitly defined in the GAIML elements. In this case XHTML [9] and Javascript code can be used directly to generate GUI structure. A GUI can be built starting from predefined interface patterns to cope with different data. In this case patterns can be described in GAIML by means of XML schemata [10] to be transformed in GUI instances depending on the data structures they have to manage. Finally, a GUI is the result of a composition process because there are no patterns to describe a certain dialogue move. In this case GAIML can manage also a set of rules to match unknown interface patterns. GUI generation is guided by the data types to be exchanged between the user and the system.

4.1. Interface defined explicitly

The easiest way to use GAIML is when the programmer defines explicitly the interface. To this aim the language introduces a new tag: <xhtmlcode>. This tag contains the XHTML code of the new interface and the related Javascript code. The content of the tag is sent to the client and used to build the new interface. The new tag is inspired to the AJAX approach [11]. AJAX-based applications are able to load on the fly not only data but also stubs of event handlers and functions. The client side generation of the HTML code allows to load web pages almost quickly. This technique reduces significantly the bandwidth consumption. In this way the payload coming down has a very small size and it's possible to develop interactive web applications. Interface Design and generation is not a heavy task, but it could be necessary at each step of the conversation. This computational load has been assigned to the client to avoid the server to be overwhelmed. Figure 2 reports a short example of a GAIML interface defined explicitly and in Fig. 3 the related code is shown.

In the conversation process the user wants to get more information about the Italian art. The system replies asks for the user preferred artist and shows a list of the artists. The user replies that she prefers Leonardo Da Vinci. Until now the interaction between the user and the system has been in natural language. The system decides to list explicitly the works in his knowledge base and shows a Da Vinci paint, the Mona Lisa. In this situation, the system changes the interaction modality and proposes a graphical interface. The user gives her next reply through the GUI, which ensembles automatically the user's reply on the basis of the dialogue. Then, the system asks the user if she likes the picture and, after a comment about the reply, it asks something about the next topic to discuss. As it's shown in Fig. 3 the whole example uses just a few <category> elements.

4.2. Usage of predefined patterns

In the previous section the developer defines completely the interaction between the user and the system. This solution suffers many limitations because it is too static. Such a GUI cannot be adapted to the needs of the interaction. On the contrary, the interface should be generated automatically by the system as needed. In many cases, the data managed by the system during the conversation or the true nature of the dialogue with the user can change during the interaction. As a consequence the system has to be provided with the ability to follow different strategies in the interface generation. The choice of the strategy is determined by the interface purposes. Some interfaces are used for data exchange or retrieval, and they're defined following a data-driven approach. The programmer defines the correspondences between data and interface, and the system generates a particular interface when it must manage a certain data type. Correspondences can be defined in two ways. The developer can either describe explicitly

Fig. 2. An example of interaction in Graphbot where the interface has been defined explicitly.

the interface features for a specific data set or give a formal description of the correspondences through a set of generation rules. Such rules use first order logic and declarative programming to describe which interface can be associated to a data set. Interfaces can be described through the data they operate on, rather than be designed as a set of pre-assembled components. Moreover, interfaces exploit formalization of data structures to enforce data correctness. An interface is described through its state space, whose properties and constraints can be expressed using either logical or algebraic formalisms. In both cases, the correspondence is established with the schemata underlying the data set rather than with specific data instances. In this way the correspondence can be used for all the instances of a particular data structure. To this aim, we have defined the so called Dialogue Move Pattern (DMP). DMPs are predefined data structures of the data involved in a single step of the interaction. They are described in specific files loaded by the system. The programmer can define his own DMPs for frequently used patterns. The

```
<category>
<pattern>Leonardo</pattern>
<template>
    Ecco un'opera di Leonardo, ti piace?<xhtml:br/><xhtml:br/>
    <think><set name="pittore">Leonardo</set></think>
    <xhtml:script defer="true" type="text/javascript">
        document.getElementById("user-input-box").style.visibility="hidden";
        document.getElementById("user-input").setAttribute("disabled","disabled");
        function jsfun(){
            var form = document.getElementById("form");
            var lunghezza_form = form.length - 1;
            var stringa = "";
            for (i=0; i &lt; (lunghezza_form);i++){
            if (form.elements[i].checked==true){
                    stringa = stringa + form.elements[i].value;}}
            if(stringa != ""){
            DWRUtil.setValue('user-input',stringa);
            endToBot(stringa);
            document.getElementById("user-input-box").style.visibility="visible";
            document.getElementById("user-input").removeAttribute("disabled");}}
    </xhtml:script>
    <random>
    <li><xhtml:img src="resources/immagini/leonardo/leonardo_gioconda.jpg"
    width="340" height="240" align="top" hspace="20"/></li>
    <li><xhtml:img src="resources/immagini/leonardo/leonardo_ermellino.jpg"
    width="230" height="300" align="top" hspace="20"/></li>
    <li><xhtml:img src="resources/immagini/leonardo/ultimacena.JPG"
    width="230" height="300" align="top" hspace="20"/></li>
    </random>
    <xhtml:br/><xhtml:br/>
    <xhtml:form id="form">
        <xhtml:input type="radio" name="scelta" value="questa opera mi piace"/>
        si <xhtml:br/>
        <xhtml:input type="radio" name="scelta" value="questa opera non mi piace"/>
        no <xhtml:br/>
        <xhtml:input id="button" type="button" value= "invia" onclick= "jsfun()"/>
    </xhtml:form>
</template>
</category>
```

Fig. 3. GAIML code for the example in Fig. 2.

definition of a DMP is written using XML Schemata. The tags of these files are used to describe the structure and the nature of the data to be managed. The programmer can define any new tag inside the definition of a DMP. Metadata insert their correspondent data in a context and assign them a meaning.

4.2.1. Interface generation through direct schemata-interface correspondences

In this case, the system contains the collection of all possible DMPs used during the conversation. The system contains a set of predefined Dialogue Move GUIs (DM-GUIs) too. A DM-GUI is defined in XML Schema and describes the structure of an interface. Interfaces are described through a suitable language, whose terms refer to generic interaction elements in an interface. Each element can be bound to many different possible specific implementations. For example, we defined the `textCtrl` element that represent the generic text area where the user can write a free content. The `textCtrl` element can be translated into a form using XHTML and Javascript, or in a Java `JTextArea`. The programmer can extend the set of DM-GUIs. Each predefined DM-GUI can be associated to one or more DMPs, and each predefined DMP is associated to one or more DM-GUIs. For each association each element of the DM-GUI must have been associated to an element of the corresponding DMP. Not all the elements of the DMP must have been associated to an element of the DM-GUI. The system manages the modality and the context of the conversation. For each specified modality and context, there must be only one DM-GUI for a DMP. These associations are predefined and loaded by the system from a file. The programmer must associate each new DMP at least to an existing DM-GUI. The system generates the interface instance for the dialogue move instance on the basis of the relations between the elements of

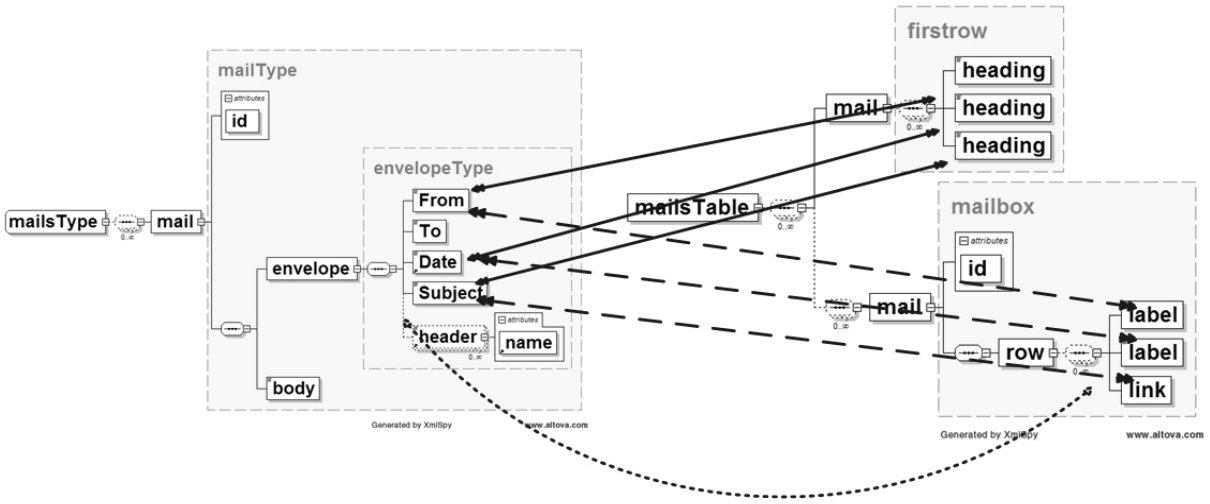

Fig. 4. An example of correspondence between a DMP and a DM-GUI.

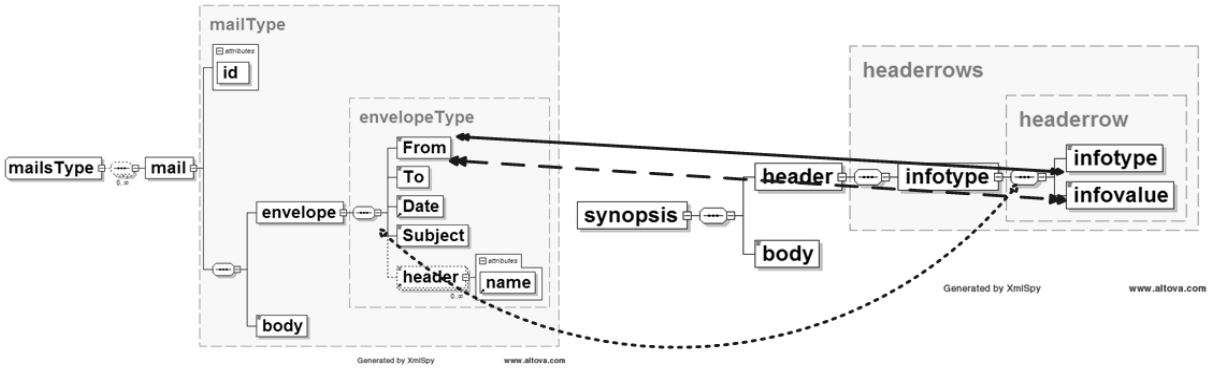

Fig. 5. Binding of the same DMP as in Fig. 4 with a different DM-GUI.

the DMP and the elements of the DM-GUI. A GAIML `<category>` can contain an instance of a DMP in the template field. In this case, the interface adopted in the conversation is generated on-line by the system.

In Fig. 4 an example is presented. In the left side there is the schema of a DMP. In particular, this schema describes the structure of a simple collection of mails in a mailbox. The right side of the image contains the generic schema of an interface that is a DM-GUI. Figure 5 shows the binding between the same DMP with a different DM-GUI. The system can switch from a correspondence to another one according to the specific context of the conversation. This goal is accomplished using different GAIML `<category>` elements containing different correspondences to manage different `<pattern>` elements.

As shown in Fig. 4, each element of the generic interface is bound to a collection of items in the data structure. The binding describes how the items must be manipulated to generate the content of the interface. When generating the interface, the system follows these bindings to construct the description of the interface. Starting from this description, the system implements the concrete interface in a specific programming language. The bindings between the item of the schemas are depicted in the figures as lines connecting the items of the two schemata. In particular, each item of the interface schema is linked

with a subset of items in the data schema. These items contribute to define the content of a part of the interface. The description in the example has been implemented as a HTML/Javascript interface. In the example the data about the mails are used to fill a simple table, whose first row is the heading of the whole table and contains the data types stored in the subsequent ones. Information gathered from the data source can be related to the content or the structure of the data. In the example the first row of the table is filled with the name of the tag of the data structure (From, Date, and Subject) and the other ones are filled with the actual values of these fields for each message. The two cases are depicted respectively by continuous or dotted lines. Several relations can be defined between the items of the DMP and the DM-GUI. Such relations can be also parametric. For example, a relation can involve only the items in certain positions, or those containing data with certain values. A relation can involve not a single data item, but a whole set of data items. We refer to it as a collective relation. The properties of the items involved by a collective relation are expressed as constraints in first order logic. Collective relations can make use of XML Schema elements as `<xs:all>`, `<xs:any>`, `<xs:choice>`, and `<xs:list>`. Figure 5 shows a different correspondence between the same DMP and a different DM-GUI. In this case, the items are treated singularly. The interface is built from a single item, and shows the synopsis of a single mail. In this example, the items, are involved in a one-to-one relation.

4.2.2. Interface generation through rules

The definition of the relationships between the DMPs and the DM-GUIs can require a substantial effort in designing the interface generation process. In fact, the developer must bind all the items in the interface schema with the items of the data schema. A possible solution to this problem to make the system able to establish how to build the interface, choosing the most suitable widgets for the interaction. In this case, the system can be guided by a set of rules when executing this task, and the interface is defined through a declarative paradigm. The developer has to say only "what" she wants, and not "how" to get it. This way is more expressive and effective. Many systems able to generate interfaces through rules have been proposed. These rules are often inspired to generic design paradigms, or to specific guidelines. In this sense, they are inspired to more general criteria allowing to cover a wider range of cases. Usually the rules are defined in ad hoc formal languages. In this field, languages founded on the first order logic have proven to be useful to express program specifications as relationships between the input and the output data. This is a case of declarative programming. Statements in a formal specification language can be automatically analyzed to put in effect the specification themselves. A possible solution is offered by the Prolog language, which is a suitable tool to define generic formal rules. Some of the authors investigated this aspect in the past and realized a system able to generate interfaces from the formal description of their specifications. We're currently combining the past work with GAIML, to implement a very expressive interaction description language and a system able to interpret it. In particular, the last version of the language allows to define interface generation rules. These rules dictate the behavior of the system, on the basis of the state of the interaction, the managed data, the tasks to complete, the characteristics of the interface, and external events. Similar languages have been just developed. A possible example is XUL [33] that has a limited expressiveness. The developer must choose, however, the elements of the interface and their behaviors. The current version of our system is able to perform these tasks autonomously. The system analyzes the data managed by the interface, and the flow of the interaction. Then the interface components are assembled with their behaviors. Finally, they're inserted into the interface. We've developed only a collection of default rules in this version. In particular, the system chooses among a collection of simple graphical components according to the data types to be managed. The behavior of each component is specified in first order logic, through the constraints on the data. The insertion of the graphical elements into the interface is inspired to the arrangement of the newspaper pages.

4.3. Pattern composition

The complex scenario when specifying an interface in GAIML is the composition of an unknown DM-GUI for a certain DMP. According to their topological properties, both the DM-GUIs and the DMPs could be represented has labeled trees. The problem of finding similarity in tree structures is known in literature as the *tree edit distance problem*. The tree edit distance problem is solved in terms of an ordered sequence of steps (edit scripts). According to [26] an edit script S between two trees $T1$ and $T2$ is a sequence of operations turning $T1$ into $T2$. We are assuming that a cost function on each edit operation is defined. The cost of S is the sum of the costs of the single operations. An optimal edit script between $T1$ and $T2$ has the minimum cost and this cost is the tree edit distance. The tree edit distance problem is to compute the edit distance and the corresponding edit script. The definition of the proper cost function is related to the particular domain. In [12] an ad-hoc built cost function is defined for the problem of web information extraction from HTML documents. In [13] the starting problem is to group similar XML files to improve the storage mapping and indexing of documents. In our case we want to define a tree edit distance for a particular data type and to compute the distance from the data that we want to show. The second step is the composition of the new DM-GUI from the known DM-GUIs used to represent known data pattern. The tree pattern matching is based on simple tree edit operations consisting of *deleting*, *inserting*, and *relabeling* nodes. A survey is presented in [26].

Let T be a rooted tree. T is a labeled tree if each node is assigned a symbol from a finite alphabet α. In addiction, T is an ordered tree if is possible to establish a left-to-right order among siblings. If T is an ordered tree the tree edit operations are defined as follows:

- *delete:* Delete a non-root node n in T whose parent node is $n^{'}$, making the children of n become the children of $n^{'}$. The children are inserted in the place of n in the same left-to-right order of the children of $n^{'}$.
- *insert:* The complement of delete. Insert a node n as a child of $n^{'}$ in T making n the parent of the children of $n^{'}$
- *relabel:* Change the label of a node n in the tree T

For the ordered version of the problem polynomial algorithms in time and space exists. Given a rooted tree T the main parameters to define the complexity of the algorithm are the size of the tree $|T|$ (number of nodes) and the depth D of the tree that is defined as the maximum number of edges on the path connecting the root and a leaf l. Other parameters are the number of leafs L and the number of children C. In [27] an algorithm has been developed where the edit tree problem is solved with an order $O(|T1||T2| \min(D1, L1) \min(D2, L2))$ in time and $O(|T1||T2|)$ in space. In [28] has been proposed an improved algorithm for the worst case ($O(|T1|^2|T2|^2)$). The solution proposed in [27] is able to lower the complexity degree to $O(|T1|^2|T2| \log T2)$. The unordered problem is NP-complete. In our application scenario the order of the representation is a consequence of the logical structure of data. According to the level of confidence of the data that has to be shown, there are two possible scenarios. In the former, data are known while in the latter there is a level of uncertainty in the data due to the interaction with the user.

4.3.1. Pattern composition from known data

In this case the data to be shown are known. The system has to find the best match from a repository of known DMPs. After this first step the system starts to define the most suitable interface for the DMP. We define an unknown DM-GUI from the composition of the known patterns. The order for the involved DM-GUIs is produced by the composition usability rules. The relabeling operation consists of changing

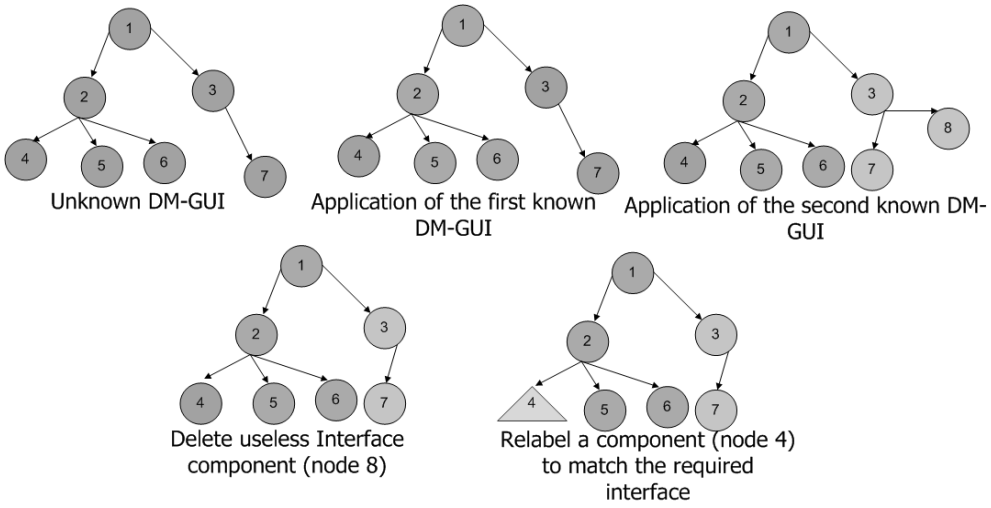

Fig. 6. A matching example.

a graphical element inside a pattern with another one. The delete operation gives a different interface by removing a component of a pattern and the insert operation consists of a component addition to the interface. Each component can also be a single widget. The matching is established through a set of matching rules. Figure 6 represents an example of the process to create a new pattern. The system has a repository of the previous matching. Firstly it tries to cover the pattern (first and second rule application). The third rule is a delete rule, while the forth one is a relabel.

The composition rules have a priority level. The most important rules are reported:

- maximum pattern covering: the rules try to cover the major portion (in number of elementary widgets) of an unknown pattern.
- minimum composition weight: this rule is a direct consequences of the tree edit distance problem.
- preference for insert and delete operations: the preferred operation for patterns are insertion and deletion for performance optimization reasons.

A side effect of using rules is that the application contexts can also be inherited. This is an important feature because the context definition provides the constraints on the interface usage and on the involved data types. The programmer could also define new rules to bind a pattern to another one.

4.3.2. Pattern composition from unknown data

Whenever the data to be presented are not known at all in advance, the system cannot start the process of interface composition. The creation of the interface is related to a syntactical matching between the possible data types (e.g. a text-box is used for strings). In this case, the programmer has to introduce an ad-hoc interface if she wants a perfect matching.

5. The GraphBot system

To prove our assumption the GraphBot system, a new *graphical chatbot*, has been developed. It integrates the two already defined interaction modalities. Graphbot can be programmed through GAIML and incorporates all the features described above.

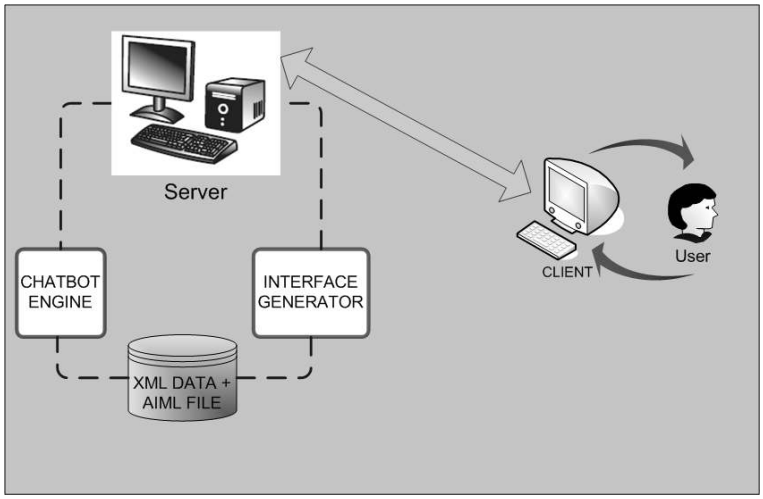

Fig. 7. The system architecture.

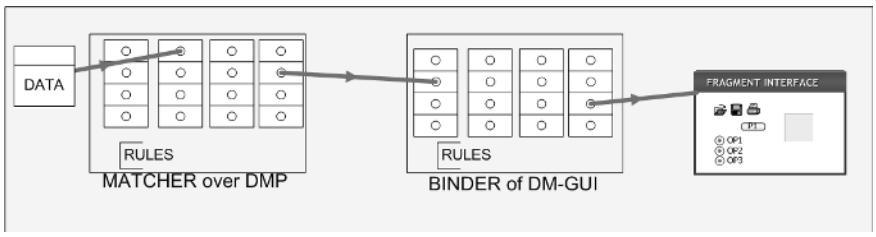

Fig. 8. The Graphbot server steps.

The interface of GraphBot has been implemented with the AJAX (Asynchronous Javascript and XML) technology. AJAX is the result of the combination of a set of old technologies which are used according to a new philosophy. In AJAX contents structure and style are clearly separated from the methods used to manipulate them. XHTML is used for the mark-up, while CSSs are used to describe styles. AJAX applications resort to client-side scripting languages, such as Javascript or Jscript, to manipulate dynamically the DOM [29] of the documents. Raw data or contents are exchanged asynchronously in XML using the *XMLHttpRequest* object.

GraphBot system has been designed according to a multi-tier client-server architecture (see Fig. 7).

The system has two principal components: the *Graphbot server* and the *Graphbot client*. The server component is managed by a Tomcat HTTP server. It loads all data files containing data and schemata for the conversation at the boot phase. The server module replies to the requests coming from the client side. The server is composed by the usual chatbot module and the *Interface creator*. It is able to redesign and re-generate a new graphical interface for the client. The generation of the new interface is a due of the other module of the Interface creator.

At the boot phase the server reads and parses four files containing the description of the predefined data and DM-GUIs, the matching rules and the binding rules (see Fig. 8).

The matching rules define how a pattern can be matched to another one. A binding rule defines how a predefined pattern is bound to a predefined interface. The interface creator can have three different behaviors, according to the code in the `<template>` element corresponding to the `<category>`

element that has been activated. In the first case, the code of the new interface is explicitly inserted into the template section. The interface creator must only get this code and return it as output. In the second case, the code contains a predefined DMP. In this case the system owns a description of the interface, corresponding to the DM-GUI bound to the DMP. In the third case the DMP is not predefined, and the system must establish a matching between the DMP and a composition of other patterns. The procedure is re-executed until the unknown pattern is completely defined through known patterns. Finally, the system produces the whole interface corresponding to the composition of all the patterns associated to the selected DMPs.

6. Conclusions and future work

In this work a system able to use different and combined interaction modalities has been presented. Firstly a new language, the GAIML, has been proposed. This language allows to merge verbal and graphical interaction modalities. Another important feature of the language is the capability to rely data structure with interface. This binding process has been developed and could be performed in three different ways according to the level of complexity of data. To prove the benefits of the approach a system able to support this language called Graphbot has been developed too. The presented chatbot system is able to interact with users in a mixed mode. The system generates GUI patterns dynamically as a response to the dialogue context, and according to the data types to be exchanged with the user. Future work rely on the improvement of the process when data are unknown. We are also trying to embed the system in a knowledge discovery agent, which is able to assist the user for learning activities over a particular domain. The Graphbot has to be placed at the highest level, the presentation one and it will be used to define user requirements and needs to allow a more natural interaction.

References

[1] G. Krasmer and S. Pope, A description of the model view controller paradigm in the smaltalk 80 system, *Journal Of Object Oriented Programming Language* **1**(3) (1980), 26–49.
[2] R. Cardone, D. Soroker and A. Tiwari, Using Xforms to simplify web programming. *Proceedings of the 14th International Conference on World Wide Web* (*WWW '05*), 2005, 215–224.
[3] J.M. Boyer, M. Dubinko, L.L. Klotz, D. Landwehr, R. Merrick and T.V. Raman, Xforms 1.0 (third edition), *W3C Recommendation*, 2007. [Online]. Available: http://www.w3.org/TR/xforms/.
[4] MPML language. MPML Home Page. [Online]. Available: http://www.miv.t.u-tokyo.ac.jp/MPML/en/.
[5] AIML language. AIML Home Page. [Online]. Available: http://alicebot.org/TR/2001/WD-aiml/.
[6] T. Bray, J. Paoli, C. Sperberg-McQueen and E. Maler, eXtensible Markup Language (XML), 1.0 second edition. *W3C Recommendation*, 2000. [Online]. Available: http://www.w3.org/TR/REC-xml.
[7] A.L.I.C.E. Bot Home Page. [Online]. Available: http://www.alicebot.org/.
[8] A. Dix, J.E. Finlay, G.D. Abowd and R. Beale, *Human-Computer Interaction – third edition*, Pearson Education, Upper Saddle River, NJ, USA, ISBN: 978-0130461094, 2003.
[9] S. Pemberton, D. Austin, J. Axelsson, T. elik, D. Dominiak, H. Elenbaas, B. Epperson, M. Ishikawa, S. Matsui, S. McCarron, A. Navarro, S. Peruvemba, R. Relyea, S. Schnitzenbaumer and P. Stark, Xhtml1.0: The extensible hypertext markup language. *W3C Recommendation*, 2000. [Online]. Available: ttp://www.w3.org/TR/xhtml1.
[10] H. Thompson, D. Beech, M. Maloney and N. Mendelsohn, Xml schema part 1: Structures, 2nd edition. *W3C Recommendation*, 2004. [Online]. Available: http://www.w3.org/TR/xmlschema-1/.
[11] J.J. Garret, AJAX: A new approach to web applications. 2005. [Online]. Available: http://www.adaptivepath.com/publications/essays/archives/000385.php.
[12] Y. Kim, J. Park, T. Kim and J. Choi, Web Information Extraction by HTML Tree Edit Distance Matching. *2007 International Conference on Convergence Information Technology*, 2007, 2455–2460.
[13] D. Rafiei, D. L. Moise and D. Sun, Finding Syntactic Similarities Between XML Documents. *17th International Conference on Database and Expert Systems Applications* (*DEXA'06*), 2006.

[14] H. Balzert, F. Hofmann, V. Kruschinski and C. Niemann, The Janus application development environment / Generating more than the user interface, *Computer-Aided Design of User Interfaces,* **1461** (1996), 183–205, Namur: Namur University Press.
[15] C. Janssen, A. Weisbecker and J. Ziegler, Generating user interfaces from data models and dialogue net specifications, in: *Bridges between Worlds. Proceedings InterCHI93*, S. Ashlund et al., eds, New York: ACM Press, 1993, pp. 418–423.
[16] F. Bodart, A. Hennebert and J. Leheureux, A systematic building of software architectures: the TRIDENT methodological guide, in: *Design, Specification and Verification of Interactive Systems*, P. Palanque and R. Bastide, eds, Wien: Springer, 1995, pp. 262–278.
[17] F. Bellifemine, A. Poggi and G. Rimassa, JADE – A FIPA2000 Compliant Agent Development Environment, *Agents Fifth International Conference on Autonomous Agents*.
[18] T. Griffiths, P.J. Barclay, J. McKirdy, N.W. Paton, P. Gray, J.B. Kennedy, R. Cooper, C.A. Goble, A. West and M. Smyth, Teallach: A Model-Based User Interface Development Environment for Object Databases, *User Interfaces to Data Intensive Systems* (1999).
[19] P. Lay and S. Lüttringhaus-Kappel, Transforming XML Schemas into Java Swing GUIs. *GI Jahrestagung (1), INFORMATIK 2004 - Informatik verbindet, Band 1, Beiträge der 34. Jahrestagung der Gesellschaft für Informatik e.V.* (*GI*), vol. P-50, 2004, pp. 271–276.
[20] T.P. Browne, Using declarative descriptions to model user interfaces with MASTERMIND. *Formal Methods in Human Computer Interaction*, 1997.
[21] B. Gooder, I. Hickson, D. Hyatt and C. Waterson, XML User Interface Language (XUL) 1.0. [Online]. Available: http://www.mozilla.org/projects/xul/xul.html.
[22] A. Puerta and A Eisenstein, XIML: A Common Representation for Interaction Data, *Proceedings of ACM IUI '01*), 2001, 214–215.
[23] F. Alonge, E. Ardizzone and R. Pirrone, Neural Classification of Multiple Sclerosis Lesions in MR Images, *International Journal of Knowledge-Based Intelligent Engineering Systems* **5**(4) (2001), 228–233.
[24] E. Ardizzone, R. Pirrone, O. Gambino and D. Peri, Two Channels Fuzzy C-Means Detection of Multiple Sclerosis Lesions in Multispectral MR Images. *Proc of IEEE International Conference on Image Processing 2002* (*ICDM-GUI 2002*), vol. 3, 2002, 345–348.
[25] E. Ardizzone, V. Cannella, D. Peri and R. Pirrone, Automatic Generation of User Interfaces using the Set Description Language, *WSCG* **12** (2004), 209–212.
[26] P. Bille, A survey on tree edit distance and related problems, *Theor Comput Sci* **337**(1–3) (2005), 217–239.
[27] K. Zhang and D. Shasha, Simple fast algorithms for the editing distance between trees and related problems, *SIAM J Comput* **18** (1989), 1245–1262.
[28] P. Klein, Computing the edit-distance between unrooted ordered trees. In *Proceedings of the 6th Annual European Symposium on Algorithms, Lecture Notes in Computer Science*, vol. 1461, 1998, 91–1002.
[29] A. L. Hors, P.L. Hégaret, L. Wood, J. Robie, M. Champion and S. Byrne, Document Object Model (DOM) level 3 core specification, *W3C Recommendation*, 2004. [Online]. Available: http://www.w3.org/TR/DOM-Level-3-Core/.
[30] J. He, T. Gao, W. Hao, I.-L. Yen and F. Bastani, A Flexible Content Adaptation System Using a Rule-Based Approach, *IEEE Transactions on Knowledge and Data Engineering* **19**(1) (2007), 127–140.
[31] J. He and I.-L. Yen, Adaptive User Interface Generation for Web Services, *2007 IEEE International Conference on e-Business Engineering*, 2007, 536–539.
[32] B.V. Zanden and B.A. Myers, Automatic, look-and-feel independent dialog creation for graphical user interfaces, *Proceedings of the SIGCHI conference on Human factors in computing systems*, 1990, 27–34.
[33] B. Gooder, I. Hickson, D. Hyatt and C. Waterson, XML User Interface Language (XUL) 1.0. [Online]. Available: http://www.mozilla.org/projects/xul/xul.html.
[34] User Interface Markup Language Specification Working Draft 3.1 [Online]. Available: http://www.oasisopen.org/ committees/documents.php?wg_abbrev=uiml.
[35] J. Foley, C. Gibbs, W.C. Kim and S. Kovacevic, A knowledge-based user interface management system. *Readings in intelligent user interfaces*, Morgan Kaufmann Publishers Inc., 1998, 474–480.
[36] A.R. Puerta, The MECANO Project: Comprehensive and Integrated Support for Model-Based Interface Development. *CADUI*, 1996, 19–36.

Roberto Pirrone received his M.D. in Electronic Engineering in 1991, and the Ph.D. in Computer Science and Engineering in 1995. Since 1999 he has been Associate Professor at the Computer Science and Engineering Department – University of Palermo. His main research interests deal with: medical image processing; knowledge management for e-learning and medical systems; intelligent web GUIs.

G. Russo received his M.D. in Computer Science and Engineering in 2005, At the moment he is a Ph.D. candidate in Computer Science at the Computer Science and Engineering Department - University of Palermo. His main research interests are about the knowledge management and ontology engineering.

V. Cannella received his M.D. in Computer Science and Engineering in 2003, and the Ph.D. in Computer Science and Engineering in 2008. Actually he has a postdoctoral fellowship in intelligent web GUIs for knowledge management and e-learning.