Generation Components and Documentation for Prototype D4.5

Alexander Melengoglou, Ion Androutsopoulos, Jo Calder,
Charles Callaway, Rob Clark, Aggeliki Dimitromanolaki, Ian Hughson,
Amy Isard, Colin Matheson, Elena Not, Jon Oberlander,
Dimitris Spiliotopoulos, Sebastian Varges, Gerasimos Xydas

Distribution: Public

M-PIRO
Multilingual Personalised Information Objects
IST-1999-10982 Deliverable 1.5

December 2002

The deliverable identification sheet is to be found on the reverse of this page.

Project rf. no.	IST-1999-10982
Project acronym	M-PIRO
Project full title	Multilingual Personalised Information Objects

Security	Public
Contractual delivery date	M34 = Nov 2002
Actual date of delivery	December 2002
Deliverable number	1.5
Deliverable name	Generation Components and Documentation for Prototype D4.5
Type	Report
Status & version	Final 1.0
Number of pages	46 (excluding front matter)
Contributing WP	1
WP/Task responsible	IRST
Other contributors	All partners
Author(s)	Alexander Melengoglou, Ion Androutsopoulos, Jo Calder, Charles Callaway, Rob Clark, Aggeliki Dimitromanolaki, Ian Hughson, Amy Isard, Colin Matheson, Elena Not, Jon Oberlander, Dimitris Spiliotopoulos, Sebastian Varges, Gerasimos Xydas
EC Project Officer	Kimmo Rossi
Keywords	language generation, domain authoring, text planning
Abstract	This document describes the final version of Exprimo, the M-PIRO language generation engine.

The partners in M-PIRO are:		
	University of Edinburgh Language Technology Group	EDIN
	Istituto Trentino di Cultura	ITC--irst
	National Centre for Scientific Research "Demokritos"	NCSR
	National and Kapodistrian University of Athens	UoATH
	Foundation of the Hellenic World	FoHW
	System Simulation Ltd	SSL

For copies of reports, updates on project activities and other M-PIRO-related information, contact:

The M-PIRO Project Administrator
The Language Technology Group
Division of Informatics
University of Edinburgh
2 Buccleuch Place
Edinburgh
Scotland EH8 9LW

Copies of reports and other material can also be accessed via the project's administration homepage, `http://www.ltg.ed.ac.uk/mpiro/`

Generation Components and Documentation for Prototype D4.5

Alexander Melengoglou, Ion Androutsopoulos, Jo Calder,
Charles Callaway, Rob Clark, Aggeliki Dimitromanolaki, Ian Hughson,
Amy Isard, Colin Matheson, Elena Not, Jon Oberlander,
Dimitris Spiliotopoulos, Sebastian Varges, Gerasimos Xydas

December 2002

Contents

Executive summary

This deliverable describes the final version of Exprimo, the M-PIRO language generation engine. Exprimo is an extensive reimplementation of the ILEX system, generating descriptions of objects in a virtual museum. The main improvements on ILEX include the use of Java, the ability to generate in three languages, the use of generation resources authored using a custom-built tool, the use of a dedicated personalisation server to store user information, and the ability to alter the lexical choices and syntactic forms depending on the user type.

1 Introduction

The M-PIRO project's main goal was to produce tailored information describing objects, in this case in a virtual museum, taking into account previous interactions. To do this, information has to be stored in non-textual fashion and turned into a spoken or written message in the language of the user when required. The language generation engine produced in the project, called Exprimo, has fulfilled all the main objectives and has in many areas exceeded the initial project aims.

Various concerns have shaped Exprimo. The most important to note initially concerns a decision that was taken early in the project, mainly for stragic reasons, to produce a Java reimplementation of the Ilex system. To recap some of the points made in D1.2, the main justifications for this decision were:

1. the system should be readily deployed on a variety of platforms
2. the architecture of the system should be relatively explicit in the sense that the organization of the inherent components should be expressed directly within the system
3. interoperability with other programs and systems was desirable, if not crucial
4. documentation was required at appropriate levels, for both system developers and application programmers

All these points favoured the use of Java for the reimplementation, and for the most part the choice has been proved correct. Some practical difficulties were nevertheless encountered in the translation from the original Lisp, and as a result some aspects of the work had to be replanned. However, the final results are at least as good as originally anticipated, and the main indication of this is the actual quality of the generated texts in the three languages; to illustrate this, section 7 below provides a number of examples which highlight various aspects of the system.

The following section looks briefly at some general background issues in text generation which are useful for putting the following system description in context.

2 Text Generation

Natural Language Generation (NLG) is a branch of computational linguistics and artificial intelligence which aims to develop computer systems that can effectively apply linguistic knowledge such as semantics, morphology and syntax, as well as knowledge of the application domain, to produce meaningful texts in human languages, based on some non-linguistic representation of information as input (Reiter and Dale, 2000). The generation process in NLG systems typically consists of the following four main stages:

- **Content Selection** (Content Determination), in which the system selects those pieces of information from its database that are most appropriate for a particular communicative purpose, user model etc,
- **Text Planning** (Document Structuring), which specifies the structure of the text by ordering the facts to be conveyed to the user and by establishing the relations that hold between these facts,
- **Microplanning**, which involves the processes of lexicalisation (the choice of the particular words and word types to describe each fact, e.g. choosing the right verb, tense, voice, person etc.), aggregation (deciding how the facts can be combined in sentences to achieve a more fluent and concise text), and referring expression generation (selecting appropriate phrases to refer to particular entities in the domain), and
- **Surface Realisation**, which performs the mapping of the abstract specifications into natural text.

Exprimo largely conforms to this model, as does the ILEX system on which it is based.

3 Background to Exprimo

In general terms, Exprimo contains a number of important advances on previous generation systems. Among these are:

- the ability to generate in three languages using largely shared language resources
- the use of input from an authoring tool
- integration with user modelling
- the ability to change texts, both lexically and syntactically, depending on user type

In addition, we have researched and implemented innovative approaches to aggregation, comparisons, and the generation of forward pointers, and all of these are discussed below. For more background on Exprimo, please see M-PIRO deliverables D1.1, D1.2, and D1.4, particularly D1.2, which discusses the reasoning behind the design of the system. Some of these discussions are summarised very briefly in the following section. Note that the most detailed description is contained in the documentation which is available in the form of the JavaDoc material accompanying the release of the system.

This particular report focusses on Exprimo's Text Planning stage as it is here that the most significant changes have been made to previous versions of the system as reported in the deliverables above. In particular, the approach to aggregation which we have adopted can be seen as operating as a high-level text planner, deciding which facts to combine into single sentences, and it is useful to present the updates to the system from this point of view.

3.1 Ilex

As noted above, Exprimo is a reimplementation of the ILEX system which provides core generation capabilities together with facilities for connecting these capabilities with representation of the domain of interest and models of users. Much of the ILEX documentation is therefore relevant background information; in particular, the User's guide to ILEX3.2[1] describes the system developed for M-PIRO. Another source of useful material is the RAGS project[2]. RAGS proposed data representations for many of the component tasks of language generation, and M-PIRO has attempted to adhere to these proposals where appropriate.

3.2 The Setting of Exprimo

The Exprimo system was designed to fulfil a number of specific roles, including:

- construction and maintenance of resources
- testing
- on-line generation of text

It was assumed from the beginning that such activities involve people with very different levels of expertise in the areas of domain knowledge representation, language generation, Web design and delivery, and so on. Therefore, to facilitate the construction and maintenance of resources, it was further assumed that the resources should conform to explicit rules. As a result, all resources and information to do with configuration of the system are supplied in the form of XML documents which conform to some publicly available DTD.

From the implementational point of view, the design reflects the overall architecture of the system, and interactions between components are mediated by explicit interfaces. In Java terms, this amounts to

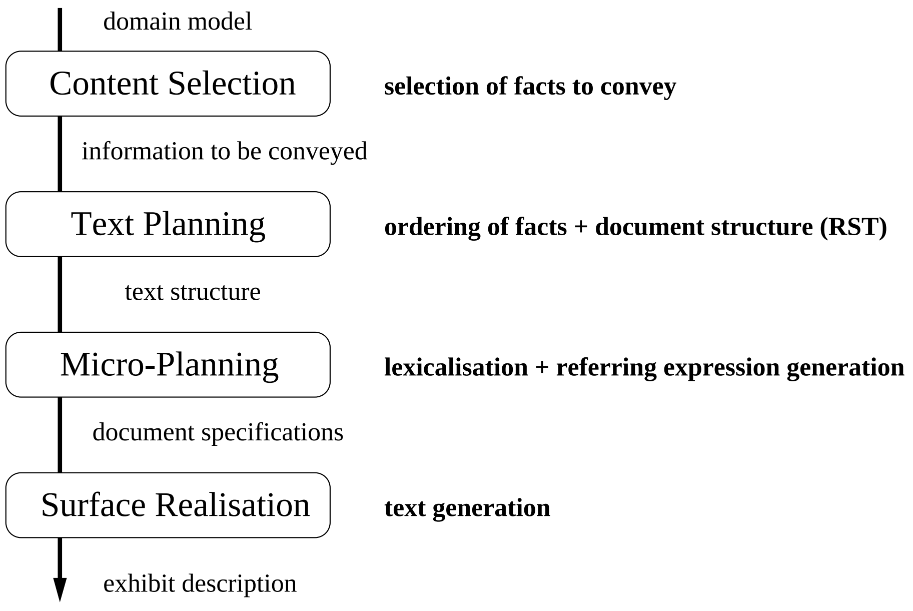
[1]`http://www.ltg.ed.ac.uk/mpiro/ilex`
[2]`http://www.itri.bton.ac.uk/research.html#RAGS`

saying that each component is specified to have certain functionalities, but the details of that functionality are private. The parts of the system that need to be visible externally are all accessed through explicit APIs. The system also includes a general debugging package which provides a general commentary on processing. The package can be instructed to display debugging information for selected subcomponents.

before looking in detail at the system components which are most important in terms of meeting the initial requirements, it is useful to review the pipeline architecture which the system assumes.

3.3 System Architecture

The general architecture of Exprimo follows a fairly standard pipeline approach to generation:


domain model
Content Selection
selection of facts to convey
information to be conveyed
Text Planning
ordering of facts + document structure (RST)
text structure
Micro-Planning
lexicalisation + referring expression generation
document specifications
Surface Realisation
text generation
exhibit description

The domain model consists of the domain database and the domain semantics. The domain database provides specifications for the basic classes of the entities present in the domain (like ”exhibit”, ”god”, ”historical-period”, and so on), as well as the entities themselves (such as ”exhibit1”, ”Apollo”, ”hellenistic-period”). Each of the entities is described by means of a number of predicates, which are either attributes (for example, ”exhibit-height” takes values like ”2.12.m”) or relations that can exist between the entity and members of other classes in the domain (for instance, the ”original-location” of an exhibit will be defined by another entity in the domain, such as ”Pergamos”, ”Amphipolis”, and so forth). Typically, each basic class defines the set of predicates that are relevant to its members.

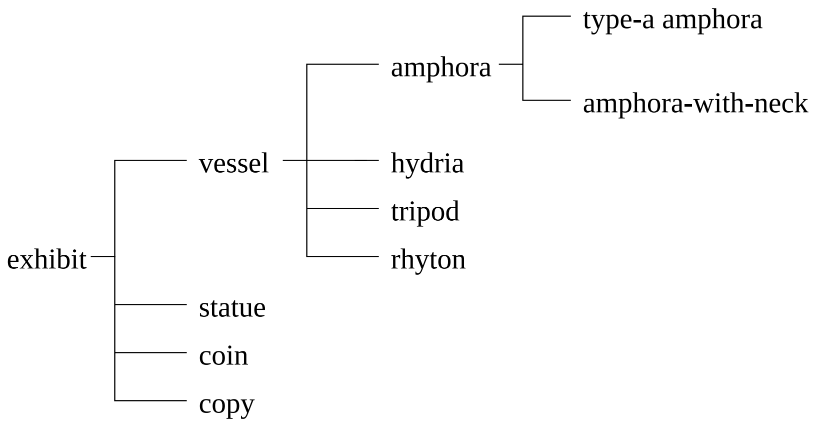
The domain semantics allows the system to interpret the domain database, by providing predicate definitions and the class taxonomy. The predicate definitions specify domain restrictions and the way in which the predicates should be expressed. For example, the definition of the predicate ”current-location” will specify that this predicate can only be used to relate an instance of the basic class ”exhibit” to an instance of the class ”museum”. In the realised clause, the former instance will occupy the `arg1` slot (that is, it will become the subject of the clause), while the latter will fill the `arg2` slot (i.e. will form the object):

```
<predicate  current-location
      arg1  exhibit
      arg2  museum>
```

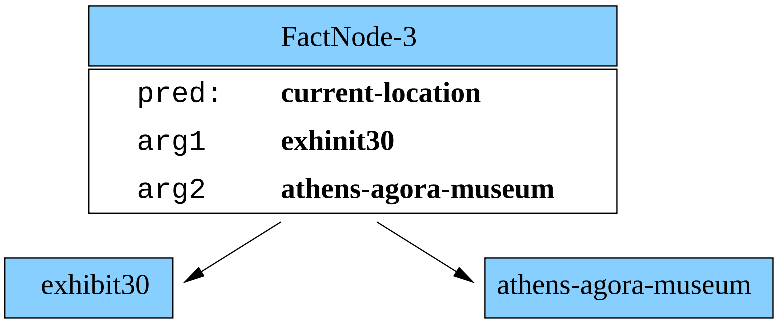
The predicate definition will also associate the predicate with a specific form of a verb:

```
<predicate  current-location
      verb  locate-verb
     voice  passive
     tense  present>
```

The taxonomy defines the hierarchy of classes in the domain. The table below shows a small part of the Exprimo taxonomy of classes, extending from the basic class ”exhibit” down to the instances of the class ”amphora”:



The Content Potential is an internal representation of the domain model in the form of a graph, reflecting the relations that exist between the entities in the database. Each instance of a relation in the domain (a fact) is a node of the graph (a fact-node), linked to the target domain objects (entity-nodes) by means of two arcs corresponding to the arguments of the predicate, Arg1 and Arg2:



The Content Potential represents the information that the system can communicate to the user, and the way in which the individual pieces of information are connected to each other. When the system receives a user request for an entity description, the system makes the particular entity the 'global focus' of the current page (Mellish et al., 1998). The facts that are connected to the entity in global focus (i.e. those satisfying the criterion of structural relevance) become candidates for selection by the system. The facts that will eventually be selected by the system are most likely to be those best matching the criterion of inherent relevance as well, which is evaluated on the grounds of the assumed interest of the content to the user, the importance of the fact in question relative to the pedagogical goal of the system, and the user's level of assimilation of the information Not et al., 2001.

The Text Planning module builds the structure of the text by ordering the facts and defining the relations that exist between them. The task of the text planner is to build the text-structure, which is a tree whose leaves are the selected propositions (in other words, each leaf is a mapping of a proposition onto a text-node). The internal nodes of the text structure specify the exact order in which the propositions should appear in the output text, and the discourse relations that hold between those propositions. The text planner of the Exprimo system expresses the dependencies that exist between the propositions by means of rhetorical relations.[3]

Text Planning builds the structure of the tree, a document plan that specifies the content and the ordering of the elements of the text that will result from the generation process. The microplanner takes this "rough" document plan and produces a detailed specification for the text, which is used by the realisation module to produce the surface document. Microplanning includes the language-dependent task of lexicalisation, which involves not only the choice of the words to use in the text, but also of the syntactic structures that will be used to express the content of the propositions.

Exprimo performs surface realisation using its computational grammar for the specific target language. The realisation component of Exprimo makes use of systemic grammars Halliday, 1985, which consist of choice systems, each offering a set of alternatives representing grammatical features. Realisation is carried out by means of a series of choices (traversal of the systemic network), where each choice is made at a different level in the hierarchy of ranks, starting from

[3]For details on the use of rhetorical relations in M-PIRO, see sections 4.2.3 and 4.3.

the clause-complex rank and ending at the word-rank, which specifies the exact grammatical and syntactic features at the lexical level.

The following sections look in more detail at the system components, occasionally discussing the implementation where this is appropriate.

3.4 Knowledge Representation

This component provides interfaces and classes for creating and accessing a model of a domain, from which the Content Potential is constructed. Classes for constructing and accessing the Content Potential are also provided. The purpose of all this is to decouple the domain model and Content Potential implementations both from each other and from the systemic functional formalism which is used for realisation.

The domain model embodies an important package in this respect as it provides facilities for modelling domain knowledge using an 'asymmetrical' variant of the Entity-Relationship-Attribute approach. In a standard E-R-A model, relationships are bi-directional and the entity types at either end are of equal status. In the variant implemented in Exprimo, relationships are uni-directional and have a primary, governing type at one end and a subsidiary one at the other. A relationship is conceived of as being primarily informative of the governing type, and there is thus a more primitive Subject-Predicate structure underlying both relationships and attributes.

Note that in the absence of exceptional circumstances, it is not desirable for a standard, symmetrical relationship to be represented explicitly in both directions as two asymmetrical ones, so an information provider must decide which is the dominant type and which the subsidiary in any particular relationship.

There are three intermediate interfaces which are used to insulate the domain model and content potential representations from the details of the grammatical representational formalism used, currently SFG.

3.5 Integration with the Authoring Tool

The final version of Exprimo is tightly integrated with the domain authoring tool. There are two main aspects to this; firstly, the domain descriptions produced using the tool can be stored as XML files which Exprimo can then use to generate, and secondly the authoring tool can use Exprimo to preview texts, giving immediate feedback to authors on the accuracy and appropriateness of new additions. The following two subsections look at these issues in turn.

3.5.1 Generating with Authoring Tool Output

Initially the Exprimo resources (domains, grammars, and lexicons) were stored and manipulated as ‘raw’ files. The authoring tool allows some of these resources, namely the domains and the lexicons, to be created automatically. It is possible to output both ILEX (Lisp) and Exprimo (XML) resources from the authoring tool, and the final version of Exprimo generates using the resulting files. Documentation of the first version of the tool can be found in D3.2, and full information on the updated version appears in D3.3. It is useful here to highlight a few points concerning the relationship between the tool and Exprimo, however.

The final version of the tool supports user modelling, allowing information on the user types and specifications such as the desired page length for each user type to be set. It is also used to set the interest, importance, and repetitions for each user type, and the repetitions parameter defines how many times the information must be conveyed before we can consider that the user has assimilated it. Setting `repetitions = 0` signals that the information must not be conveyed at all for that particular user type.

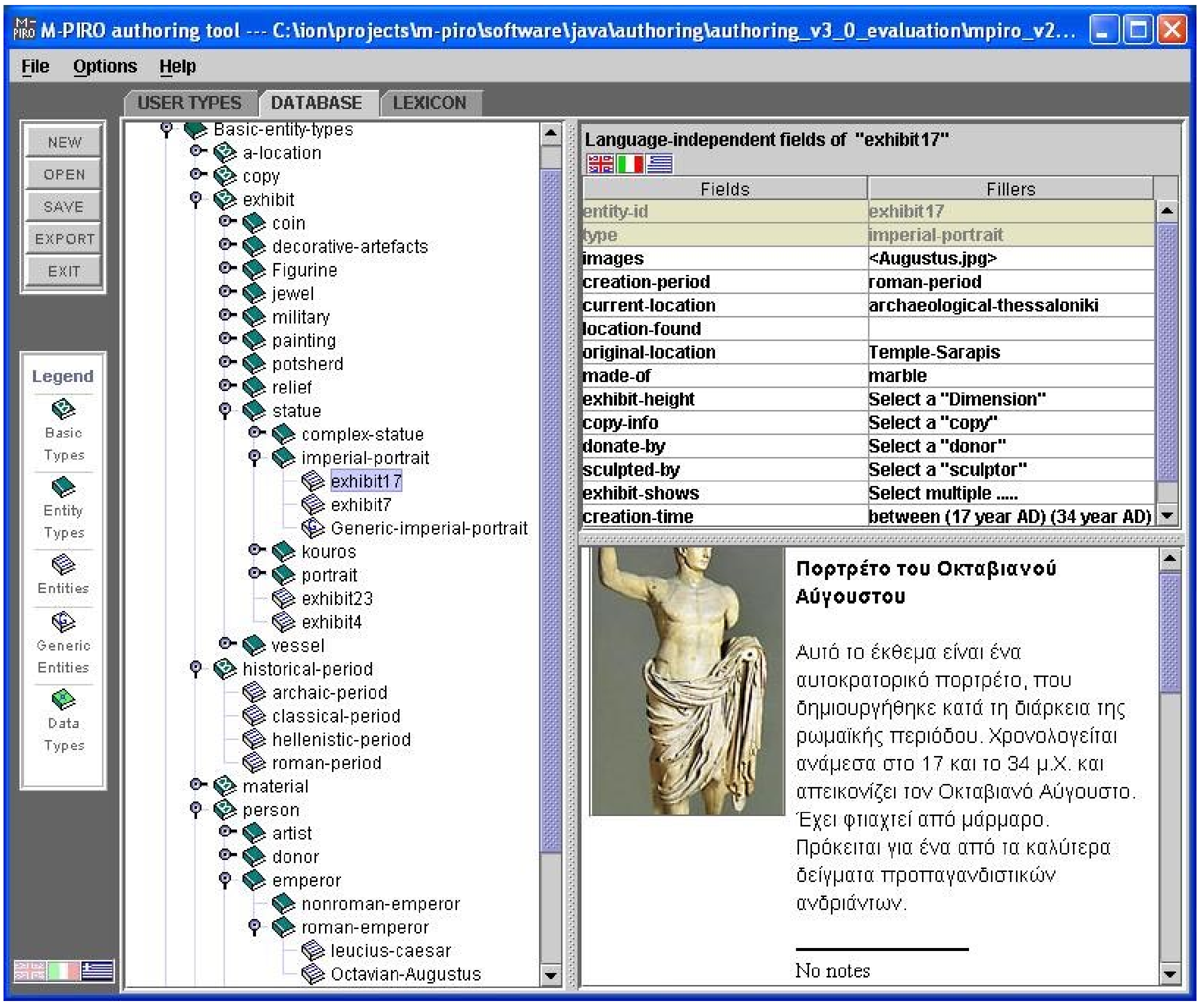
In general, the domain author can specify the interest, importance and repetitions of a field at the entity type that introduces it. In the current domain, for example, the interest of the `creation-period` field for adults is set to 3 at the ‘exhibit’ type, ensuring that all ‘exhibit’ entities inherit that value.

Micro-planning expressions have been included to allow for future versions of Exprimo to vary the expressions used to convey a particular fact. Currently this is not implemented. Positive values indicate appropriate words for a user type, with greater values indicating more appropriate words. Negative values indicate inappropriate words, with smaller values indicating more inappropriate words. The idea is that, for the sake of variation, Exprimo may not always choose the applicable word with the highest appropriateness score, but it will generally try to use words whose appropriateness is positive (unless there are none available), and from those, preference will be given to the words with the highest appropriateness scores. We plan to include similar appropriateness scores in future versions of the micro-planning tab.

The new version of the authoring tool can also download all the user modelling information to the personalisation server (which runs as a separate server), from where it becomes available to Exprimo.

3.5.2 Previewing

As noted above, the authoring tool can run Exprimo and preview texts as a means of checking that recent additions to the database have the desired affects. The example screenshot in figure 3.5.2 shows the authoring tool with a text in the preview window, alongside the underlying data



Language-independent fields of "exhibit17"

Fields	Fillers
entity-id	exhibit17
type	imperial-portrait
images	<Augustus.jpg>
creation-period	roman-period
current-location	archaeological-thessaloniki
location-found	
original-location	Temple-Sarapis
made-of	marble
exhibit-height	Select a "Dimension"
copy-info	Select a "copy"
donate-by	Select a "donor"
sculpted-by	Select a "sculptor"
exhibit-shows	Select multiple .....
creation-time	between (17 year AD) (34 year AD)

Πορτρέτο του Οκταβιανού Αύγουστου

Αυτό το έκθεμα είναι ένα αυτοκρατορικό πορτρέτο, που δημιουργήθηκε κατά τη διάρκεια της ρωμαϊκής περιόδου. Χρονολογείται ανάμεσα στο 17 και το 34 μ.Χ. και απεικονίζει τον Οκταβιανό Αύγουστο. Έχει φτιαχτεί από μάρμαρο. Πρόκειται για ένα από τα καλύτερα δείγματα προπαγανδιστικών ανδριάντων.

No notes

Figure 1: Previewing using the authoring tool

which was used to produce some parts of the text. One point to note about this is that there is no communication with the personalisation server when Exprimo is being run in this configuration; it does not, of course, make sense to be storing and/or retrieving user information in this context. To avoid having multiple versions of Exprimo, an emulation of the server was written which has exactly the same functionality as the full server except that its information is not persistent (so it discards all the information about its users when the authoring tool is exited. In all other respoects the emulator behaves like the real server, allowing Exprimo to operate as usual.

3.6 Macronode Authoring Tool

We have a working version of an authoring tool for macronodes, and we have tested this by authoring new exhibits using the macronodes in order to debug performance on unseen data. The first version of the tool has been distributed, and we currently have the first 15 exhibits converted completely to XML format. We hope to have macronodes authored for the complete proposed 50 exhibits by the end of the project.

3.7 Integration with User Modelling

User modelling information is stored and retrieved over the internet from a personalisation server. The server records all the relevant modelling information, as instructed by Exprimo and the authoring tool, and sends the generation engine the required parameter values on request, thus tailoring the generated text. An example of the user modelling subsystem in operation is highlighted in section 7 below, and the following is a brief introduction to some of the main aspects of the approach assumed.

Basically, Exprimo's user modelling (UM) sub-system provides long-term personal models and stereotypes. It consists of a personalisation server (PS), which runs as a separate Web server, and additional Java code that can be used by other Exprimo components (the authoring tool or Exprimo) to interact with the PS. The additional Java code translates between Exprimo's user modelling API and the HTTP-based protocol that the PS uses as its native language.

An instance of the PS is currently running on an NCSR server. The PS is configured to work under Windows 2000 with MS Access or MySQL[4] as its underlying database system. It is written entirely in Java, and interacts with the underlying database system via ODBC, which means that it should be possible to port the PS to any other operating system and database system that provide the correct versions of Java and ODBC with minor modifications.

The additional Java code that translates between Exprimo's UM API and the native protocol of the PS is also written in Java. To clarify our assumptions about the architecture of Exprimo, the UM API was split into two parts. One specifies the API that the UM subsystem supports during the interaction with end-users (visitors), and the other specifies the API that the UM subsystem supports during the authoring process. The first makes no assumptions about the UM techniques that are used (such as whether or not user types are employed). This should make it possible to change some of the UM techniques at later stages without modifying anything in Exprimo. In contrast, the second part of the API makes certain assumptions about the UM techniques being used (for instance, it assumes that user types exist). Another distinction is that the first part of

[4]MySQL runs on both Unix and Windows and hence allows the whole system to be portable across these operating systems.

the API can be used concurrently by several users, whereas the second is currently designed to be used by a single user (author); this can change in future versions if necessary.

All the UM information that needs to be provided by the author (such as usertypes and stereotypical information) is entered using the authoring tool. As noted in Section 3.5 above, the tool provides an option to download this information to the PS. The information is also saved in the serialisable Java object file `mpiro`) which the authoring tool uses to save and load domain information. This is why there is currently no way for the authoring tool to retrieve UM information from the PS.

3.8 External APIs

The biggest change to the structure of the system in comparison to ILEX is the move towards decoupling the text generation modules from the preceding content potential level and the subsequent realisation levels. This allowed the introduction of a more general planner which combines the normal ILEX-style generation with the macronode system. To effect this decoupling, external APIs were defined and partly implemented at the content potential/planner interface and at the planner/realiser interface. The content potential interface is fully implemented, but the realisation stage is more complicated and more effort is planned in this area. Again, a detailed description of the existing implementations is available in the form of system documentation.

In general terms, the interfaces to the realisation stage provide an API for specifying and realizing arbitrarily complex grammatical structures. The model is of a constituent tree structure, with each node filling some role in its parent's structure and bearing grammatical features which further constrain its expression. The `RealizationFactory` interface should be implemented by the class providing the grammar(s) to which linguistic descriptions are meant to conform. Its basic purpose is to supply the nodes to act as the targets of descriptions. `RStructure` is the interface to be implemented by these nodes. It enables role and feature specifications to be passed to them and realized. It also allows any node's final realized form to be set.

Currently, `RealizationFactory` is implemented by the class `CGrammar`, which is able to return `RStructures` for any class which implements the `UnitWrapper` interface. `RStructure` is implemented by the `Unit` class and, as a result, is a very powerful construction tool. Indeed, although the primary intention of the interface is to provide sentence planning functionality, this implementation may be used to build structures at any level defined in the grammars supplied. Needless to say, this should only be attempted on the basis of a thorough familiarity with the grammars and a clear conception of the task. Returning to more standard usage, the constants provided reflect the emphasis on sentence planning. They cover all of the roles described in the ILEX documentation for the clause and NP grammars, to which the user is referred for further details.

It should be noted that the assignment of any role or feature to a node nearly always implies additional roles and features at that node, which can mean that the precise result of multiple assignments is dependent upon the order in which they are performed. In many cases, this has no practical impact and the same end result is achieved, but this cannot be guaranteed. In particular, the interface does not constrain the order in which specifications passed to the plan method are performed, nor any other aspect of their interpretation. A rule of thumb worth following, however, is that it is generally better to assign roles before features. Usability demands that multiple assertions of features at the same node be interpreted conjunctively and multiple assertions of roles at the same node be interpreted conflatively.

The current implementation of plan interprets its input as a series of role-value pairs. The reserved role name ”is”, which can be represented by the defined constant `FEAT`, signals a feature specification. This enables both features and roles to be combined in a single plan. Aside from adopting the above-mentioned rule of thumb, the implementation performs assignments in the order they are given. Multiple instances of the same role, including `FEAT`, at the same level are permitted and are interpreted as indicated above. A feature specification may consist either of a String, naming a feature, or a String array, naming zero or more features to be interpreted conjunctively. A role specification may consist of any one of the following:

- an `RStructure`, representing the node with which to fill the role.
- a String, being either the name of the `LexicalItem` with which to fill the role or a fixed string with which to fill it.
- an `Object` array, being itself a plan specification. By this means, arbirarily complex recursive structures may be described.

4 High Level Text Planning

As mentioned in the general introduction, the final text planning component contains most of the important updates on the previous versions of Exprimo and we can illustrate many of the central aspects of the system by looking in detail at how the aggregation module works. To recap the system architecture described in section 3.3 above, Exprimo works as a pipeline of content selection, text planning, microplanning, and realisation. The essential characteristics of everything apart from text planning are as described in previous deliverables, and the most important change to the text planner is in the way the aggregation module makes high-level decisions about the final structure of the text.

Much of the discussion of aggregation here is drawn from Alexander Melengoglou’s Edinburgh University MSC dissertation Melengoglou, 2002. Early versions of Exprimo were able to produce simple forms of aggregation, constructed at a late stage in the generation process by manipulating the actual generated texts. For many reasons, it was desirable to replace this with a module which acted at a deeper, and earlier, stage. The aggregation component reported here represents a significant advance on the previous versions, and also represents an improvement on the original ILEX aggregation module.

4.1 Background

An important point to emphasise here is that Exprimo is designed to generate personalised descriptions, in the sense that the system aims to adapt its educational goals to the current user, based on its model of potential user types. In this context, as well as being able to conjoin adjacent propositions using the appropriate operations, it is desirable that the aggregation module also possess the flexibility to adjust the syntactical complexity of the output text to the requirements of the target user type. The aggregation module of the Exprimo system constructs sentences of a maximum length dictated by the model of the target user, by making use of four standard aggregation operations, operation transformations to resolve conflicts within sequences of incompatible operations, and an algorithm that allows it to examine the length and the quality of the constructed sentences in parallel.

The Content Selection module packages information into ”verb-based, clause-sized propositions, each of which is realised as a single sentence” Shaw and McKeown, 1997. Texts composed exclusively of such single-fact sentences are very likely to contain repetitions and redundancies, and are almost certain to be considered boring and unnatural by human readers. To overcome this problem, NLG systems make use of aggregation algorithms which combine semantically related propositions in order to produce a more concise and coherent text. The effect of aggregation can be seen very clearly in the following example from (Jurafsky and Martin, 2000), in which two propositions with obvious common features were combined to produce a single sentence:

A. You’ve just compiled a simple C program
B. You’ve just run a simple C program

[A+B] $\rightarrow$ You’ve just compiled and run a simple C program

In the above example, assuming that the aggregated sentence was the output of some NLG system, aggregation made use of the system’s linguistic resources to combine the facts in sentences A and B in a manner that is perfectly grammatical in the target language (here, in English).

Aggregation can be thought of as a submodule of Microplanning or as a separate module between Text Planning and Microplanning. In any case, its input is the same, i.e. the output of the Text Planning module, which is a tree showing how the information to be presented is grouped. The structure of this tree reflects both the ordering of facts (each fact is a leaf in the tree -a text-node) and the dependencies that exist between the facts, often expressed by means of rhetorical (or, RST-) relations. Rhetorical relations can exist between either a dominant textnode , the ‘Nucleus’, and its dependent, the ‘Satellite’ node (‘Mono-Nuclear’ relations), or between textnodes of equal status (‘Multi-Nuclear’ relations) (see (Mann and Thompson, 1987)). To summarise, in the words of (Scott and de Souza, 1990), in text generation ”the message of the text is comprised of a set of propositions which form the leaves of a hierarchical rhetorical structure that expresses the writer’s intentions behind the inclusion of each proposition.”

Thus, in slightly more technical terms, we can define aggregation in NLG systems as the task of mapping the structures resulting from the process of Text Planning onto linguistic structures, such as clauses, sentences and paragraphs Reiter and Dale, 2000. Aggregation detects shared elements among neighbouring textnodes and combines them to remove redundancies and repetitions in the resulting text.

There are several types of aggregation, most of which are directly associated with the particular rst-relations holding between input propositions. For instance, the most common type of aggregation, simple conjunction, joins facts together by means of coordinating conjunctions, like ‘and’ and ‘but’. Coordinating conjunctions are used to put together clauses of equal importance, so we would expect to see this type of aggregation being applied on binuclear or other multinuclear relations. For example, we would expect to see two facts in a binuclear RST-CONJUNCTION joined with the conjunction ‘and’, while two others linked via an RST-CONTRAST would be aggregated with a ‘but’ or a ‘yet’. In such cases, aggregation has to make several decisions that influence the appearance of the text, such as whether to omit the subject in the second proposition (”he took her hands from her eyes and led her towards the house”[5]), the auxiliary verb (”someone may be killed or seriously injured”), etc. Another aggregation operation, syntactic embedding, is realised by means of one proposition being subordinated to another (”he was waving to the girl, who was running along the platform”). This type of aggregation is often called hypotactic aggregation and research indicates that it should only be used for mononuclear RST-ELABORATION relations Scott and de Souza, 1990.

It is evident that there are several types of aggregation and several ways in which each type of aggregation can be implemented. An NLG system must decide which types to use and in some cases whether aggregation should be performed at all. Aggregation results in shorter texts, but tends to make sentences more syntactically complex, which in certain cases may be undesirable (e.g. the case of a child reader of the output text). The choice of the particular aggregation operations seems to be highly domain-specific. According to (Reiter and Dale, 2000), a good approach, given uncertainties like the ones already mentioned, is to use ”corpus analysis to identify the kinds of aggregations that result in the human-authored texts in the target genre.”

[5]The examples in this paragraph are taken from the Collins Cobuild English Grammar, 1990.

4.2 Input to Aggregation

Aggregation receives as input a sequence of semantic representations of facts which were ordered and grouped into dependency structures, expressed in terms of rhetorical relations, during the Text Planning stage. In order to conjoin a sequence of facts successfully and in a manner preserving the meaning of the text, it is essential for aggregation to have a clear idea of the exact nature of the propositions in the sequence. The particular characteristics of each fact will define whether this fact can actually become a candidate for aggregation and the role that it will play in a potential sentence.

Propositions differ in terms of content, form and membership in rhetorical relations, and these distinctions are discussed in turn below.

4.2.1 Classifying Facts vs. Facts Presenting Attributes

Classifying facts define the class of the entity in question: *This exhibit is an amphora*, *Leucius Caesar was a Roman emperor*, and so on. No entity can have two classifying facts.

The specification for a classifying fact (used to generate the sentence *This exhibit is a black kantharos*) contains information like:

```
<FACT.1>:
        | ARG1: 2197
        @ filler: black-kantharos
        | PRED: 1
        @ filler: type
        @ verb: be-aux
        | ARG2: 2203
        @ filler: black-kantharos
        @ basic class: exhibit
```

In the above specification, the ARG1 role is occupied by the ”specific-thing” in focus, (here, referred to by the name of its basic class -”exhibit”) and the ARG2 slot by the ”generic-thing”, that is, the name of the subclass (”black-kantharos”) of the basic type (”exhibit”). The name of the filler of the predicate position in the above specification is ”type”, which is the name given to all classifying predicates in the Exprimo domain.

Facts which present attributes provide some information about an entity other than the entity’s class, used in sentences such as *It was created during the archaic period, it originates from Attica*, and suchlike. An entity can have several attributes, the number and nature of which are specified by the domain specification for the entity’s basic class (see the discussion of the domain model above).

For instance, a proposition realised as *It was painted with the west slope technique* would have a specification like:

```
<FACT.3>:
        | ARG1: 2197
        @ filler: black-kantharos
        | PRED: 36
        @ filler: painting-technique-used
        @ verb: paint-verb
        @ voice: passive
        @ tense: past
        | ARG2: 1005
        @ filler: west-sl-technique
        @ basic class: technique
        @ prep: with
```

Aggregation assigns less importance to facts presenting attributes than to facts specifying an identity, and in complex sentences the former are always subordinated to the latter.

4.2.2 Facts Realised as Clauses vs. Templates

By default, Exprimo's systemic functional grammar generates indicative clauses in the declarative mood (so the subject noun phrase precedes the verb phrase). Indeed, the Subject-Verb-Object (S-V-O) constituent order is the most common in descriptive texts. However, not all facts are actually realised as clauses qualifying for aggregation. Several categories of facts presenting attributes are, either entirely or in part, expressed by means of non-aggregatable fixed strings. Such strings ("templates") are used when the information to be presented to the user requires syntactic structures that cannot be generated by the system. The fact that the attribute is conveyed with the use of a fixed string makes the manipulation (and even the detection) of one or more of the proposition's main features impossible for the aggregation module.

The text below consists of four facts that qualify for aggregation and a fact expressed by means of a template, which appears in the final sentence:

> This exhibit is a black kantharos, created during the hellenistic period. It was painted with the West slope technique and it originates from a city called Amphipolis. The ”West Slope” decoration is obvious on its neck.

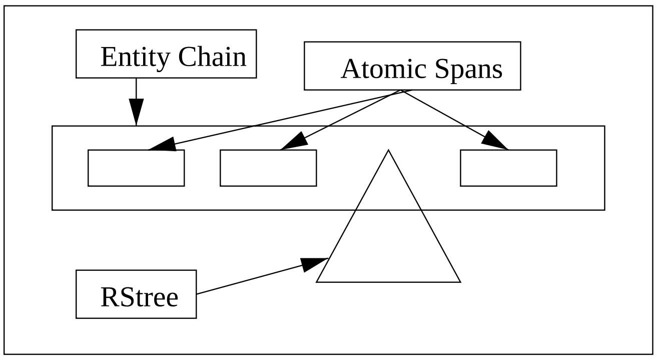
The specification for this template is the following:

```
<FACT.5>:
        | ARG1: 2197
        @ filler: black-kantharos
        | PRED: 13
        @ filler: exhibit-characteristics
        | ARG2: 2216
        @ TEMPLATE: the "West Slope" decoration is obvious on its neck.
```

Here we can see that the only information that aggregation was able to extract is the name of the entity to which it refers (”black-kantharos”) and the name of the predicate (”exhibit-characteristics”). No information about the verb, the voice, and so on, is visible to aggregation, or to any other part of the system. Thus, in a sequence of propositions, templates always function as actual aggregation barriers.

4.2.3 RST: Satellites vs. Nuclei

The subject of all the nuclei (either independent or part of a rhetorical relation tree – an *RStree*) in a description of a museum exhibit is the same: i.e. the exhibit in question, that is, the entity in ‘global focus’. Such an entity-chain (a sequence of text-spans about a specific entity), is illustrated in ILEX Knott, 1998 with the diagram below:



The text pages that are generated by ILEX are often composed of several entity-chains (each representing a paragraph) in a row, connected by means of resumption relations (where the focus of an entity-chain is an entity mentioned as the object in one of the propositions of a previous entity-chain). In Exprimo, however, resumption relations are not used, and each page simply consists of a single entity-chain, as above. The subject of the satellite propositions in such entity chains is always the object of the nucleus of the particular relation (the ‘local focus’). To illustrate this, note the following text, in which the focus only moves away from the exhibit in the final sentence:

[1] This exhibit is an imperial portrait. [2] It dates from the first century AD. [3] It is rendered with youthful features and an athletic bearing, similar to the poses that the classic sculptor Polykleitos was using in his statues. [4] It depicts Leucius Caesar. [5] Leucius Caesar was a roman emperor.

In this case the subject is the same in the classifying statement [1], the attribute proposition [2], the template [3] (all independent nuclei) and in the rst-bound nucleus [4] (the ”imperial portrait”), whereas in the embedded element [5] the subject is the object of the nucleus proposition [4].

The only relation currently employed in Exprimo is RST-ELABORATION, a ‘default’ mononuclear rhetorical relation holding if the nucleus makes a reference to an object and the satellite of this relation subsequently introduces an attribute of that object. According to (Knott et al., 2001), in this context, ”the precise meaning of ‘attribute’ is not clear, but any proposition which provides additional information about the object would seem to qualify”. Indeed, in Exprimo, we find both attribute propositions and classification statements in satellite positions, providing further information about the objects in the nuclei.

4.3 Relations Between Facts

The discourse relations literature contains many examples of sentences such as *Max fell. John pushed him* which can be interpreted, and therefore aggregated, in several different ways[6], depending on our guess of the rhetorical relation actually holding between the two. These examples show that the content of the propositions is not sufficient to represent meaning. They also illustrate the importance of a thorough specification of the rhetorical relations by the document planner: when the relations are properly defined, aggregation can combine linguistic structures to produce a meaning-preserving, more concise text.

The validity of the above conclusions is obvious in natural texts. In natural texts, it is very often the case that the content of two or more propositions can give rise to different interpretations of their meaning. The particular relations holding between the propositions must, therefore, necessarily be made explicit to the aggregation module, otherwise the actual meaning could potentially be lost in the resulting text.

However, in the descriptive texts produced by ILEX and Exprimo, each text-span in an entity-chain contributes a single piece of information to the description of the entity in focus, independent of the content of the other (neighbouring or distant) propositions. In such entity-chains,

[6]For instance:

a. Max fell because John pushed him (RST-EXPLANATION)

b. Max fell and John pushed him (RST-SEQUENCE).

rhetorical relations are only used to clarify the relations that hold within a complex text-span (an 'RStree' in figure 3.1), that is, between the propositions involved in a shift of the local focus within a particular entity-chain. The relations between the text-spans themselves (i.e. multinuclear relations) remain unspecified in Exprimo, because there is no reason for the system to enforce any specific method of realisation. Each text-span, only loosely related to its neighbours[7], can eventually be realised as either a stand-alone sentence, or as part of a sentence formed together with other spans -the particular selection by the aggregation module will not affect the meaning of the text. In this context, we can safely rely on the examination of the nature of the propositions to determine the relations that hold within a sequence. The relation between two adjacent text-spans will depend on the relative importance of their content. This is exactly the reason why the distinction between 'classifying' and 'attribute-defining' facts is so important to aggregation. The combination of the two types would result in a complex sentence, in which the fact specifying the identity is always realised as the main clause. For instance, in the complex sentence composed of the facts in tables 3.1 and 3.2 ”This exhibit is a black kantharos, painted with the West slope technique”, the main clause is a direct realisation of the classifying proposition, while the subordinate clause is formed by the fact expressing the attribute.

On the other hand, the aggregation of two attribute propositions creates a compound sentence. In the sentence *It was painted with the West slope technique and it originates from a city called Amphipolis*, the two coordinate clauses were produced by the realisation of two facts presenting attributes. In general, we can assume that there exists an inherent, default relation among the attribute propositions for the entity in global focus. We can then use our assumption of this default relation to conjoin these independent nuclei. According to Scott and de Souza, 1990, ”the paratactic marker and must only be applied to Sequence[8] and List” relations.

ILEX assumes a default RST-JOINT relation ”to signal the connection between two independent sentences” (Knott, 1998). In ILEX, a JOINT relation, ”which doesn't have much semantic content and is just used to fit all the substructures under the same roof” (Knott, 1998), is always realised through paratactic coordination. By making no distinction between classifying and property-defining propositions in aggregation, ILEX permits paratactic operations that result in sentences like *This jewel is a necklace and was made by a British designer called Edward Spencer*, in which the identity and the attribute are assigned equal status.[9]

4.4 Summary

To summarise, the input to the aggregation module is an ordered sequence of facts, presenting either the identity or an attribute of an entity. Certain categories of facts presenting attributes

[7]By means of the reference to the entity in global focus

[8]RST-SEQUENCE is a 'narrative' sequence, in which constraints are placed on the order of propositions.

[9]However, in Hua Cheng's description of the aggregation model developed for the ILEX-TS system (Cheng, 2001) we find complex sentences such as 'this jewel is a bracelet which is 0.6 in width'.

are expressed by means of fixed strings and consequently do not qualify for aggregation. Facts can either constitute atomic-spans in the entity-chain or participate in RStrees. The nucleus of the latter is always a fact presenting an attribute of the entity in global focus. The following table provides a synopsis of the possible realisations of each category of facts in the generated sentences:

Content	**Identity**		**Attribute**				
Form	aggregatable clause		aggregatable clause			template	
RST	ind-N	S	ind-N	N	S	ind-N	S
Realisation	stand-alone sentence or main clause	apposition	stand-alone sentence, subordinate clause, or coordinate clause	main clause	apposition, subordinate clause, or qualifier	stand-alone sentence	

ind-N: Independent Nucleus (Nucleus not present in an RST-ELABORATION)
N: Nucleus in RST-ELABORATION
S: Satellite in RST-ELABORATION

4.5 Domain-Specific Rules

We assume that a good set of aggregation operations is one that is able to:

- express the relations that hold between the segments in the resulting text
- clear the text of redundant and repetitive elements
- maintain consistency with the target genre

Exprimo generates descriptive texts, composed of relatively short, verb-centred sentences with declarative structure. The propositions describe either the identity or an attribute of an entity in focus. Classifying propositions have a higher status in the texts than facts presenting attributes, and, by default, we assume that there exists a sequence[10] relation connecting the attribute facts

[10]The ordering of propositions in Exprimo reflects both the importance placed on their content and the pedagogical goals of the human author. The selection of RST-SEQUENCE (rather than RST-LIST) indicates that the ordering of attribute propositions will not be changed by the aggregation module.

of the entity in bglobal focus. Finally, in RStrees, the satellites are always linked to the nucleus through an RST-ELABORATION.

Thus, we are looking for a set of aggregation rules capable not only of producing a text that is more concise and readable, but also of expressing:

- the dominance of classifying facts over attribute facts, when the two types are both present in a sentence,
- the equal status of attribute facts, and
- the subordination of satellite facts to the nuclei in RST-ELABORATIONS.

4.6 Aggregation Rules

The core of the aggregation module is the set of aggregation rules used to characterise the required options, plus the algorithm which determines the application of the rules. Full details on the aggregation capabilities can be found in (Melengoglou, 2002); here we summarise some of the main points, but we provide detail on some aspects as an illustration of how the module works.

4.6.1 Aggregating identity – attribute pairs

The following rules are used to aggregate two text-spans of unequal importance; namely, one classifying the entity in focus and another describing one of the entity's attributes:

Type-Comma:[11] This rule reduces the attribute proposition to a non-defining relative clause, subordinated to the proposition making the classification.

In English, the application of this rule depends on the voice of the verb in the subordinate clause. The rule is currently applied when the verb of the attribute proposition is in the passive voice, irrespective of whether it is in the present or in the past tense. The subject and the auxiliary verb of the subordinated proposition are omitted. The absence of auxiliaries changes the mood of the subordinate clause to non-finite. For example:

Nucleus1:	**This exhibit is a rhyton**	`pred-filler`: **type**
Nucleus2:	**It was painted by the painter of Sotades**	`pred-filler`: **painted-by**

[Nucleus1 + Nucleus2] $\rightarrow$ **This exhibit is a rhyton, painted by the painter of Sotades.**

[11]In the M-PIRO domain classifying predicates are called 'type' predicates, hence the name of the rule.

Nucleus1:	**This exhibit is a rhyton**	`pred-filler:` **type**
Nucleus2:	**It is painted with the red figure technique**	`pred-filler:` **painting-technique-used**

[Nucleus1 + Nucleus2] $\rightarrow$ **This exhibit is a rhyton, painted with the red figure technique**

In Italian texts, when the verb of the attribute proposition is in the passive voice, the subordination can still be expressed by changing the mood to non-finite:

Questo ritratto è una anfora, dipinta dal pittore di Cleofrade.

When the voice of the verb is active, the predicate is preceded by the pronoun ‘che’:

Questo ritratto è una anfora, che proviene dall’Attica.

In Greek texts, the type-comma rule is realised by means of a comma followed by the right choice of a gender-sensitive relative pronoun (among: ο οποίος (masculine), η οποία (feminine), το οποίο (neuter) or, alternatively, by means of the gender-insensitive pronoun που (equivalent to both the English pronouns ”who” and ”which”):

Αυτό το έκθεμα είναι μια κύλικα, η οποία δημιουργήθηκε κατά τη διάρκεια της κλασικής περιόδου.

Αυτό το έκθεμα είναι ένας μελαμβαφής κάνθαρος, που δημιουργήθηκε κατά τη διάρκεια της ελληνιστικής περιόδου.

The voice of the verb in the attribute proposition does not affect the implementation of this rule in Greek texts.

Type-Qualifier: In a sequence of two or more satellite propositions in an RST-ELABORATION, it is possible for an identity-attribute pair to follow syntactic embedding. In this case, the attribute proposition is attached to the preceding entity-classifying fact (now embedded) as a qualifier:

Nucleus:	**This portrait depicts Alexander the Great**	`pred-filler:` **person-country**
Satellite1:	**Alexander was a king**	`pred-filler:` **type**
Satellite2:	**Alexander was from Macedonia**	`pred-filler:` **person-country**

[Nucleus + Satellite1 + Satellite2] $\rightarrow$ **This portrait depicts Alexander the Great, a king from Macedonia**

Type-Semicolon: A semicolon may be used in place of the comma to remove the subordination of the right hand side of a relation whenever this side includes more than one proposition, two of which can be aggregated via simple conjunction:[12]

[12]Type-comma and simple conjunction are considered here to be incompatible aggregation operations – see the section on ‘restrictions’ in 4.6.2.

Nucleus1:	**This exhibit is a lekythos**	`pred-filler`: **type**
Nucleus2:	**It was painted with the black figure technique**	`pred-filler`: **painting-technique-used**
Nucleus3:	**It originates from Attica**	`pred-filler`: **original-location**

[Nucleus1 + Nucleus2 + Nucleus3] → **This exhibit is a lekythos; it was painted with the black figure technique and it originates from Attica**

A 'type-semicolon' is always followed by a simple conjunction. In some cases, the shared subject-predicate rule may fall between a type-semicolon and a simple conjunction:

> This exhibit is a rhyton; it was painted by the painter of Sotades with the red figure technique and today it is located in the Musee du Petit Palais.

The system treats the type-qualifier and the type-semicolon rules as transformations of the type-comma rule, which is assigned by default during the early stages of the aggregation process to any adjacent pair of identity-attribute propositions in the input sequence. Such rule transformations are used in later stages to resolve conflicts within sequences of aggregation operations in a sentence (see section 4.8).

4.6.2 Aggregating attribute pairs

The following rules are used to conjoin two facts of equal status, namely two facts which describe an attribute of the same entity:

Simple Conjunction: Simple conjunction forms a compound sentence. In its simplest form, simple conjunction will aggregate two propositions sharing the same subject, without omitting the subject of the second proposition:

Nucleus1:	**It was painted with the black figure technique**	`voice`: **passive**
Nucleus2:	**It originates from Attica**	`voice`: **active**

[Nucleus1 + Nucleus2] → **It was painted with the black figure technique and it originates from Attica**

When the voice of the verb is the same in both propositions, the subject on the right side of the conjunction can be omitted:

Nucleus1:	**It was painted with the black figure technique**	`voice`: **passive**
Nucleus2:	**It was made by a potter called Sotades**	`voice`: **passive**

[Nucleus1 + Nucleus2] → **It was painted with the black figure technique and was made by a potter called Sotades**

Dalianis, 1999 refers to the latter case as "subject aggregation". In Italian texts, simple conjunction is performed with the use of the coordinating conjunction *e*:

Questa anfora fu dipinta dal pittore di Cleofrade e proviene dall'Attica.

In Greek, simple coordination is carried out by means of the conjunction και. The subject in the second proposition is omitted in all cases, as in Greek null pronouns are a very common case:

Αυτή η υδρία ανήκει στον ερυθρόμορφο ρυθμό και προέρχεται από την Αθήνα.

Restrictions:

This rule cannot follow the type-comma rule in the same sentence. Even though the resulting sentences are grammatically correct, the existence of two predicates on the embedded side tends to assign more weight to the attributes, thus weakening the role of the identity statement:

This exhibit is a rhyton, created during the classical period and painted with the red figure technique.

Shared Subject-Predicate: This operation is used in cases where two adjacent propositions have identical subject and predicate positions.

Nucleus1:	**The rhyton was painted by the painter of Sotades**
Nucleus2:	**The rhyton was painted with the red figure technique**

[Nucleus1 + Nucleus2] $\rightarrow$ **The rhyton was painted by the painter of Sotades with the red figure technique**

This rule is used in Italian texts in the same manner as in the English texts:

Questa anfora fu dipinta dal pittore di Cleofrade con la tecnica a figure rosse.

Potentially, this rule can be used in Greek texts as well; however, the current Greek domain does not offer opportunities for shared subject aggregation.

4.6.3 Aggregating nucleus-satellite pairs

Syntactic Embedding: Syntactic embedding is an aggregation operation in which a proposition is realised as a constituent subordinated to a dominant proposition, typically by means of apposition or a relative clause. In M-PIRO, as in other NLG systems, it is used to express the subordination of a satellite textnode to the nucleus in an RST-ELABORATION. In the English texts generated by the system, the subordinate proposition is incorporated in the sentence:

- via apposition, when the verb of the satellite proposition is ‘to be’, or
- via a non-defining relative clause, in all other cases.

Apposition:

Nucleus:	**It depicts Leucius Caesar**	
Satellite1:	**Leucius Caesar was a Roman emperor**	`verb:` **be-aux**

[Nucleus + Satellite1] $\rightarrow$ **It depicts Leucius Caesar, a Roman emperor**

Non-defining relative clause:

Even though the current domain never presents opportunities for syntactic embedding that cannot be realised directly through apposition, the aggregation module is prepared to handle cases like the one below, in which the subordination is realised with the introduction of a non-defining relative clause:

Nucleus:	**It depicts Leucius Caesar**	
Satellite1:	**Leucius Caesar was murdered by a group of Republicans**	`verb:` **be-aux**

[Nucleus + Satellite1] $\rightarrow$ **It depicts Leucius Caesar, who was murdered by a group of Republicans**

Apposition, in general, is preferred to a relative clause as being more ”economical”, i.e. it only uses a noun phrase to convey the information in the embedded proposition. For example, compare the noun phrase ”a roman emperor” with the subordinate clause ”who was a roman emperor.”

In Italian texts, syntactic embedding is performed with the use of the pronoun *che*:

Questa ritratto imperiale rappresenta Leucius Cesare, che fu un imperatore romano.

In Greek texts, syntactic embedding is realised by introducing the subordinated proposition as a non-defining relative clause:

Απεικονίζει τον Λεύκιο Καίσαρα,ο οποίος ήταν ένας Ρωμαίος αυτοκράτορας.

4.6.4 Rule hierarchy

It is very often the case that the system cannot make use of all the opportunities for aggregation that are present in a sequence of propositions (see sections 4.7 and 4.8). When confronted with such a situation, the system needs to select which aggregation operations to apply and which ones to discard. This problem calls for a prioritisation of the aggregation rules. In general, we would like to give higher priority to rules which:

- *Result in less redundant, and thus, more readable, text.* Assuming that aggregation will not produce an undesirable loss of information, Dalianis, 1999 argues that "text length (i.e. shortest text) is the best measure of the readability of aggregated texts."
- *Help clarify the meaning*, by expressing the relations that hold between the elements in the text. Indeed, Reiter and Dale (Reiter and Dale, 2000) emphasise the significance of building more complex sentence structures, in order to utilise the "potential for giving clues as to the relative importance of different elements of the underlying content."

The last criterion favours aggregation operations that express subordination (for us, type-comma and syntactic embedding). This is because subordination is by no means the 'default' relation between any two adjacent textual elements, and readers, according to Scott and de Souza, 1990, "are unlikely to retrieve the rhetorical structure of a message unless it is stated explicitly." The importance of syntactic embedding, in particular, derives also from the fact that it is the only operation that can express the relation between a nucleus and a satellite in an RST-ELABORATION. Syntactic embedding is thus of great value to aggregation, as it is essential that the system manages to "keep the propositions of a rhetorical relation together in a text." (Scott and de Souza, 1990). On the other hand, simple conjunction is the aggregation operation that least meets both of the above criteria. Its application either does not reduce the length of the text at all, or, at best, removes the subject of the second proposition. Also, simple conjunction expresses the 'default' relation between the attributes of the entity in global focus, and thus adds little, or nothing, to the readers' understanding of the meaning of the text.

We therefore reach the conclusion that syntactic embedding is the most important rule in the set, while simple conjunction is clearly the least significant. We do not have a clear preference between type-comma and shared subject-predicate, as the former helps to explicitly define a relation of unequal status, while the latter results in shorter texts.

4.7 Rule Sequences

One of the most important user modelling parameters in Exprimo is `max_facts_per_sentence`, which as the name suggests specifies the maximum number of facts that the system should convey to a particular type of user in each sentence. Different values of this parameter will, in general, result in shorter or longer sentences, and its main purpose is to allow the system to adapt the output text complexity to target user types. Short sentences, according to (Reiter and Dale, 2000), ”may be well suited for people with limited reading ability, such as small children,” but for most readers ”the use of short sentences becomes irritating.” In essence, the `max_facts_per_sentence` parameter defines the maximum number of aggregation rules to use in a single sentence (which is equal to `max_facts_per_sentence` - 1). A value of ‘1’ effectively disables all aggregation operations. Any value greater or equal to ‘2’ will turn aggregation on and will greatly influence the module’s sentence-building strategies.

Given the current database, the set of aggregation rules used by the system, and the restrictions specified in the code, it is very often the case that the system returns sentences consisting of three facts joined together. Cases of four facts occur less frequently, but are not uncommon. The examples below illustrate the effect of different values of the `max_facts_per_sentence` parameter on a series of four propositions generated by the system:

> `max_facts_per_sentence` = 1
>
> This exhibit is a rhyton. It was painted by the painter of Sotades. It was painted with the red figure technique. Today it is located in the Musee du Petit Palais.
>
> `max_facts_per_sentence` = 2
>
> This exhibit is a rhyton , painted by the painter of Sotades. It was painted with the red figure technique and today it is located in the Musee du Petit Palais.
>
> *or*
>
> This exhibit is a rhyton. It was painted by the painter of Sotades with the red figure technique. Today it is located in the Musee du Petit Palais.
>
> `max_facts_per_sentence` = 3
>
> This exhibit is a rhyton , painted by the painter of Sotades with the red figure technique. Today it is located in the Musee du Petit Palais.
>
> `max_facts_per_sentence` = 4
>
> This exhibit is a rhyton ; it was painted by a painter called the painter of Sotades with the red figure technique and today it is located in the Musee du Petit Palais.

4.8 Sentence Quality

Each aggregation rule examines the local relations that exist between a pair of adjacent textnodes in the text structure. The question that arises when building complex sentences through the successive application of a number of rules is whether these local relations will continue to be in effect in a new environment in which certain propositions are no longer engaged in a single relation, but in two. As a sentence is formed, the status of certain propositions is altered. Inevitably, very often, conflicts between propositions that were previously found compatible emerge.

When there is a conflict in the application of two adjacent aggregation rules, the system must necessarily choose between adapting one aggregation rule to the new linguistic structure formed by the other (only possible in certain specific cases) and giving up the application of one of the rules. In some other cases, the most obvious of which is the case of two simple-conjunctions in a row, which would result in an entirely unattractive piece of text,[13] the system has no choice other than to abandon one of the rules.

We saw above that, as the number of successive aggregation opportunities rises, the aggregation of facts becomes an increasingly complicated task. The value of the `max_facts_per_sentence` parameter and the rules of grammar of the target language impose restrictions on the application of the candidate aggregation rules. Thus, there are two major types of restrictions that influence the system's sentence-building strategies:

- User Modelling Restrictions: imposed by the value of the `max_facts_per_sentence` parameter, and
- Text Quality Restrictions: those related to the overall readability of the generated text, in terms of grammatical correctness or otherwise.

What makes the creation of complex sentences in such a context a really challenging task is the fact that the system cannot deal with the two types of restrictions separately, i.e. it does not have the luxury of solving all the problems related to one of them first, and then attempt to find solutions for the other. For instance, let us suppose that we have four consecutive facts and that, after an examination of the nature of these facts, the aggregation module produces the following initial table of aggregation opportunities:

```
Initial Aggregation Table: Facts 1-2: type-comma
Initial Aggregation Table: Facts 2-3: simple-conjunction
Initial Aggregation Table: Facts 3-4: simple-conjunction
```

[13] For example, *This rhyton was created during the classical period and was made by a potter called Sotades and today it is located in the Musee du Petit Palais.*

If we apply the text-quality tests first, we will notice that no two neighbouring rules are readily compatible with one another. Type-comma cannot be followed by a simple conjunction, and this is also true for simple conjunction itself: it cannot be followed by another simple conjunction. Nonetheless, the first conflict can be resolved through a 'type-comma to type-semicolon' conversion. For reasons of variation, in similar cases, the system is very often making a random selection between alternative actions. For instance, in the present example, the module randomly selects between a punctuation transformation and the removal of the weakest rule (here, the simple conjunction between facts 2-3). Let us suppose that the system, indeed, makes the decision to introduce a semicolon. Having resolved the first conflict, the system examines the second one. As there is no satisfactory solution to a double conjunction[14] the system inevitably removes the second conjunction. Now we have:

```
Initial Aggregation Table: Facts 1-2: type-semicolon
Initial Aggregation Table: Facts 2-3: simple-conjunction
Initial Aggregation Table: Facts 3-4:
```

But what if the value of the user modelling parameter was actually set to 2? In this case, we cannot have two rules applied in a row, as this would result in a sentence of 3 facts. So, having finally considered the sentence-length restrictions as well, the module would return the following final aggregation table:

```
Final Aggregation Table: Facts 1-2: type-semicolon
Final Aggregation Table: Facts 2-3:
Final Aggregation Table: Facts 3-4:
```

This will produce highly undesirable results:

> This exhibit is a rhyton; it was made by a potter called Sotades. It was painted with the red figure technique. Today the rhyton is located in a museum called the Musee du Petit Palais.

The other last-moment alternative, of only keeping the simple conjunction in the middle, is considerably better, but still far from perfect:

> This exhibit is a rhyton. It was made by a potter called Sotades and was painted with the red figure technique. Today the rhyton is located in a museum called the Musee du Petit Palais.

[14] Sentences of the form *This rhyton was created during the classical period, it was made by a potter called Sotades and today it is located in a museum called the Musee du Petit Palais* are considered unattractive in this domain and are consequently not generated.

Even in this case, we have given up a rule of middle priority (type-comma) and a rule of lowest priority (simple conjunction) in exchange for a single rule of lowest priority (another simple conjunction).

It is evident that the best choice for the system to make is:

```
Final Aggregation Table: Facts 1-2: type-comma
Final Aggregation Table: Facts 2-3:
Final Aggregation Table: Facts 3-4: simple-conjunction
```

Which will produce:

> This exhibit is a rhyton, made by a potter called Sotades. It was painted with the red figure technique and today it is located in a museum called the Musee du Petit Palais.

However, the general solution does not lie in the reversal of the order of the two tests, i.e. defining the sentence boundaries first and then performing the text-quality checks. To demonstrate this, let us consider the case of the following initial aggregation table, knowing in advance that the user modelling parameter was set to the value of 3:

```
Initial Aggregation Table: Facts 1-2: subject-predicate
Initial Aggregation Table: Facts 2-3: simple-conjunction
Initial Aggregation Table: Facts 3-4: syntactic-embedding
```

We need to remove at least one of the rules from the table; otherwise we will end up with a sentence of four facts, which is a violation. Even if we knew at this point that these three rules are perfectly compatible with each other in that particular order (we do not know that yet, as we are applying the sentence-length test before testing for sentence quality), we would still need to remove one of the rules to reduce the size of our sentence to three facts instead of four. So, the natural choice seems to be the elimination of either the first or the last rule: in this case we will have a sentence of three facts (our ”goal”) and another, single-fact sentence.

However, the quality checks would seriously object to the sacrifice of a middle priority rule (subject-predicate) in favour of a simple conjunction. Moreover, based on the theory of rhetorical relations, these tests would definitely not allow the breaking of a nucleus-satellite rst-relation (syntactic embedding), regardless of the purpose of this breaking.

Indeed, the elimination of either the subject-predicate rule (*The rhyton was painted by the painter of Sotades. It was painted with the red figure technique and...*) or of the syntactic embedding rule

(*...and today it is located in the Musee du Petit Palais. The Musee du Petit Palais is in France*), would introduce repetitions in the text, for which we cannot be compensated by the fact that we have built a sentence of the ideal size.

The best choice, therefore, is the removal of the simple conjunction in the middle of the table and the formation of two sentences, each composed of two facts:

> The rhyton was painted by the painter of Sotades with the red figure technique. Today it is located in the Musee du Petit Palais, in France.

Only the text quality restrictions know that this is the right selection. But then again, these restrictions know nothing about the sentence boundaries set by the user modelling parameter. So, how does the system make the proper decision?

4.9 Placing Sentence Boundaries

We need to build sentences that are not only grammatical and readable, but also within the size limits set by the user modelling parameter. We saw in the previous section that it is very easy for the system to make inadequate selections while performing the two kinds of tests one after the other, irrespective of which test is executed first. It is obvious that we must find a way to combine the two procedures, so that every time one of the two tests (either the sentence-building or the sentence-quality one) is ready to make the final decision as to whether a candidate rule in question is to be employed or not, it knows that either

a. its decision is in agreement with a decision already made by the other, or that

b. it does not need to know what the decision of the other test would be on a particular case, because it has the right kind of information to make a decision on its own.

The aggregation rules are combined with other rules, and we saw that some rules are more important to the sentence and to the text as a whole than others, for a variety of reasons, such as the amount of repetitive or redundant information that they remove, their contribution to the naturalness and meaningfulness of the text and so on. We also saw that very often the selection of a rule inevitably results in the elimination of another, either for reasons of incompatibility or for the simple reason that the system very often has to select one of the rules and break it in order to form a sentence boundary. Therefore, before applying any rule, the system needs to find the answers to questions like: ”What would be the effect of applying this rule? Will we have to sacrifice another rule, and if yes, how important is this other rule?”

The module makes use of the following table, expressing the hierarchy of the aggregation rules in numbers:

Syntactic Embedding	2
Type-Comma	1
Shared Subject-Predicate	1
Simple Conjunction	0
No rule	-1

'No rule' receives a negative score as the system needs to know that the application of any rule, even of one of the lowest rank, is preferable to the application of no rule at all. An empty slot in the initial aggregation table has a negative value and may indicate an aggregation opportunity that was lost. An empty slot in the final aggregation table has no value at all and simply indicates a sentence boundary.

The sentence-building algorithm makes use of two parameters:

a. `max_facts_per_sentence`, which as we have seen is the user model parameter, which remains constant throughout the process, and

b. `number_of_facts_already_in_sentence`, which indicates the length of the sentence before the application of the candidate rule.

Whenever the system starts building a new sentence, `number_of_facts_already_in_sentence`, receives the initial value of 1. This value is increased as new facts join the sentence, but is never allowed to exceed the current value of `max_facts_per_sentence`.

The table in Figure 2 shows how the rules are used. Note that Steps 1 and 2 in Figure 2 are mutually exclusive: they cannot be both executed during the same iteration. Step 2 is always taken every time the rule examined by Step 1 in the last iteration has eventually succeeded in passing all the remaining tests and has actually become part of the sentence. It is interesting to note also that the sentence-boundary algorithms perform forward tests (looking at the next aggregation rule in the table), while the text-quality algorithms perform backward tests (examine compatibility with rules already present in the sentence).

The input of the sentence-building algorithm is the initial aggregation table. The output is the final aggregation table, which is then used by the module in its successive calls to the rule-implementing methods to carry out the actual aggregation of the propositions.

For examples illustrating in detail how the algorithm and the rules work, see Melengoglou, 2002.

STEP1:	IF (**number_of_facts_already_in_sentence = max_facts_per_sentence**) - 1) {
	  THEN
	    IF (THIS_RULE.**inferiorTo**(NEXT_RULE)) {
	      REJECT(THIS_RULE)
	      GOTO STEP4
	    }
	    ELSE IF (THIS_RULE.**superiorOrEqualTo**(NEXT_RULE)) {
	      GOTO STEP3
	    }
	}
	
	If the application of a new rule is about to fill the sentence with the maximum number of facts, check whether a more powerful rule immediately follows the rule in focus.
	Action: If a rule higher in the rule hierarchy comes next, remove the rule in focus and proceed with step number 4; otherwise proceed with step number 3.

STEP2:	IF (**number_of_facts_already_in_sentence** = **max_facts_per_sentence**) {
	  REJECT(THIS_RULE)
	}
	ELSE GOTO STEP3
	
	If the sentence has already reached its limit, then this rule is definitely redundant. It has already been tested (during the previous iteration at Step 1, as being the fact following the rule that was in focus) and rejected (but not deleted, in anticipation of the sentence quality results of the previous rule). It can neither be part of the previous sentence, which is now confirmed complete, nor start a new sentence, because the first proposition that it was meant to aggregate is already a part of the previous sentence.
	Action: If the sentence is complete, remove the rule in focus from the aggregation table; otherwise proceed with Step 3.

STEP3:	IF (THIS_RULE.**compatibleWith**(THIS_SENTENCE)) {
	  ACCEPT(THIS_RULE)
	}
	ELSE {
	  REJECT(THIS_RULE)
	}
	
	If the candidate rule has made it through Steps 1 and 2, test this rule for text quality restrictions.
	Action: If the new rule is compatible with the rules already in the sentence (with or without internal rule transformations) accept it; otherwise reject it.

STEP4:	IF (THIS_RULE.**partOf**(THIS_SENTENCE)) {
	  INCREASE **number_of_facts_already_in_sentence**
	}
	ELSE
	  SET **number_of_facts_already_in_sentence** = 1
	}
	
	Examine whether the rule is still present in the aggregation table after Steps 1, 2 and 3.
	Action: If the rule is still present in the table, increase the value of **number_of_facts_already_in_sentence** by 1. If not, reset its value to 1.

STEP5:	REPEAT UNTIL MORE_RULES = FALSE **Repeat** Steps 1, 2, 3 & 4 until there are no more rules left in the initial aggregation table.

Figure 2: The aggregation module’s sentence-building algorithm

5 Forward Pointers

An important aspect of the web-based presentation of objects is the generation of forward pointers, which suggest a number of alternative paths that the users can follow to examine other objects that are in some way related to the one in the current description.

Exprimo is equipped with a sophisticated module that generates lists of forward pointers in all three languages dynamically, with the use of the generation engine. This module builds lists about those pieces of information in the current description (attributes of the current exhibit) that are assumed to be of greatest interest to the user, and fills those lists with pointers (links) to exhibits which have not yet been visited by the user, which also possess the attribute in question.

To provide an illustration of the kind of pointer that Exprimo generates, here are typical examples of generated texts (in all 3 languages), followed by a list of the accompanying forward pointers:

> This exhibit is a black kantharos; it was created during the hellenistic period and it dates from the late 4th century B.C. Unlike the previous vessels, which were decorated with the red figure technique, it was painted with the West slope technique. In The West slope technique, the decoration, usually floral, is made of added clay. This is a technique typically used on vases of the Hellenistic period. Its name comes from the west slopes of the Acropolis, where several of these vases were found. This black kantharos was originally from Amphipolis. The ”West Slope” decoration is obvious on its neck. Today this black kantharos is located in the Archaeological Museum of Kavala.

- Other exhibits created during the hellenistic period:
  - A relief
  - A portrait made from marble
- Other exhibits originally from Amphipolis:
  - A figurine made from clay

> Αυτό το έκθεμα είναι ένας παναθηναϊκός αμφορέας,που δημιουργήθηκε κατά τη διάρκεια της ελληνιστικής περιόδου. Αυτός χρονολογείται στο 320 π.Χ. Προέρχεται από την Αττική και σήμερα βρίσκεται στο John Paul Getty, το οποίο είναι στην Καλιφόρνια.

- Παρόμοια εκθέματα στη συλλογή:
  - Ένας παναθηναϊκός αμφορέας που δημιουργήθηκε κατά τη διάρκεια της κλασικής περιόδου
- Άλλα εκθέματα που δημιουργήθηκαν κατά τη διάρκεια της ελληνιστικής περιόδου:
  - Μια υδρία του ρυθμού Hadra που προέρχεται από τη Ρόδο
  - Ένας μελαμβαφής κάνθαρος που ανήκει στο Ρυθμό Δυτικής Κλιτύος
  - Ένα ανάγλυφο
  - Ένα πορτρέτο που έχει φτιαχτεί από μάρμαρο

Questo reperto e una anfora, creata durante il periodo arcaico. Risale a gli inizi del 5 secolo a.C. Rappresenta un guerriero che esegue la splancnoscopia prima di partire per la battaglia. La splancnoscopia era l'analisi delle viscere animali, attraverso la quale la gente cercava di predire il futuro. Era uno dei metodi divinatori maggiormente impiegati nel periodo arcaico. E dipinta secondo la tecnica a figure rosse. Per ultiori informazioni, vedi Boardman J: Athenian red figure vases, the Archaic period, Thames Hudson, 1975.

- Altri oggetti che sono dipinti secondo la tecnica a figure rosse:
  - Un cratere da nozze che proviene da l'Attica
  - Un stamnos che e fatto in argilla
- Altri oggetti che furono creati durante il periodo arcaico:
  - Un stater che e fatto in argento
  - Una pittura che e fatta in legno
  - Un elmo

6 Comparisons

As discussed in section 3.3, the system stores the information that it can convey to the users about the entities in its database in the form of fillers (which are either values or pointers to other entities) of a number of predicates that are common to the members of a certain basic class. For instance, members of the basic class 'exhibit' can have predicates such as 'creation-period' and 'original-location', while members of the basic class 'person' different ones like 'person-country, and so on.

Thus, each entity can be described by means of a finite set of predicates, and it is possible to group the members of a class in a number of ways, based on the fillers of their common predicates. For example, in the domain of museum exhibits, we may classify exhibits according to their origin (e.g. several exhibits originate from Attica, others from Argos etc.), according to the historical period during which they were created (archaic, classical, hellenistic etc.), and so on.

It is obviously not practical for applications like Exprimo to offer lists of similar (or different) entities within the body of a description for the entity in current focus (the forward pointers module partly performs this task separately from the descriptions, by providing lists of similar entities not yet examined by the current user). However, it is both desirable and feasible that such applications possess the ability to underline those similarities (or differences) that are assumed to be of particular interest to the users of the system. The generation of comparisons between the current and previously examined entities (”backward” comparisons) helps the users build a more complete picture of the concepts that are presented, make associations and reach valuable conclusions. By being the result of the user's own navigation, such comparisons result in texts that are more personalised and, thus, more interesting, attractive and intelligent.

6.1 Comparisons in Exprimo

An initial aim in generating comparisons was to avoid making individual, full-clause references to previously seen exhibits, such as:

> Like the amphora that was decorated by the painter of Kleofrades, this lekythos was created during the archaic period.

Such past references both distract the users from their focus of attention (the current exhibit) and tend to lead to confusion, as users are unlikely to remember which (of the possibly several) vessels that they have already seen is the one that actually matches a description. Thus, such comparisons tend to be boring, if not irritating, while their educational value is questionable. The comparison module attempts to overcome this problem by using the class hierarchy to group previously examined exhibits into broader categories and make either group references (references to the class, if the comparisons hold for all the previously mentioned members of the particular class) or short (name-only) individual references, when the system knows that the name of the exhibit is sufficient to make a unique reference.

Two types of comparisons are performed, based on the similarities and differences between exhibit (in terms of the resulting discourse relations: RST-SIMILARITY and RST-CONTRAST).

'Contrast' comparisons always take place between the current exhibit and a group of previously visited exhibits (never an individual exhibit). This group is always directly related to the current exhibit; it is either the class of the exhibit or one of the exhibit's superclasses. The module performs comparisons between exhibits of the same general category (i.e. between vessels, statues, coins, and so on). Thus, the system will try to compare a panathenaic amphora with a hydria or a black kantharos as they are all vessels, but never, say, with a drachma or with a suit of armour.

6.2 Locating target predicates

Let us suppose that the user has already been given descriptions of 9 ancient Greek vessels:

1 amphora
1 aryballos
1 lekythos
1 prochous
1 white lekythos
2 kylixes
2 panathenaic amphoras

The user is now requesting a description of another lekythos. The content planner selects the facts to convey to the user:

```
type
creation-period
creation-time
exhibit-depicts
painting-technique-used
original-location
exhibit-story
```

The system has a pre-set list of predicates which are assumed to be suitable for forming comparisons. While in commercial applications (like eCommerce) it would be meaningful to perform comparisons based on numeric values, such as price and dimensions (for instance 'Like the Alpha Romeo, this Rover costs 40,000 Euros'), in the M-PIRO domain it was considered more appropriate, mainly for pedagogical reasons, to base the comparisons on broader concepts, such as the period during which an exhibit was created ('Unlike the previous coins, which were created during the Hellenistic period, this tetradrachm was created during the Classical period') rather that on specific values, like dates and height. Comparisons such as 'Like the previous lekythos, this rhyton was created in 460 B.C.' are assumed to have little educational value in this particular context, and so we restrict the range of predicates that are appropriate targets. The system therefore examines the facts that were selected by the content planner and, in our example, finds three potential targets for comparisons:

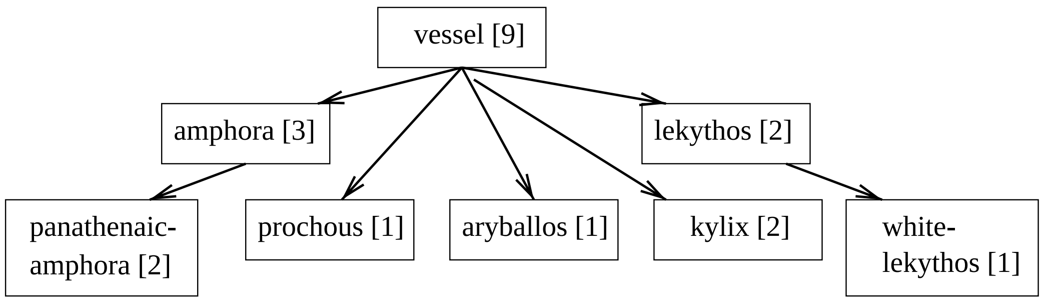
```
classical-period
red-figure-technique
attica
```

In the actual texts, such target predicates would appear as shown below in italics:

> This exhibit is a lekythos, *created during the classical period*. It dates from between 470 and 460 B.C. It has a picture of an athlete ready to throw his javelin. *It was painted with the red figure technique* and *it originates from Attica*. This lekythos is now exhibited in the National Archaeological Museum of Athens. In antiquity, javelin throwing was intimately bound up with the Greek way of life. Before it had become a feature of sporting life, the javelin was one of the weapons used by ancient Greeks in war and hunting. A javelin is a sharp, wooden spear about the height of a tall man.

6.3 The Class Hierarchy

As we saw, we are assuming that the user has already examined the nine vessels mentioned in the previous section. As a first step to forming a comparison, the system completes the domain class hierarchy tree for the previously examined exhibits that belong to the same exhibit subclass as the current exhibit (in our example, this subclass is vessel). The tree is created bottom-up, and can be represented as shown below:



The numbers in brackets represent the total number of instances of each type which have been seen so far by the current user. For instance, among other things, the above diagram is telling us is that three of the nine previously examined vessels are amphoras, and that two of these three amphoras are panathenaic. For the two instances of the panathenaic amphora that the user has already visited, the system extracts the following set of potential comparators :

```
panathenaic-amphora creation-period classical-period:  1
panathenaic-amphora creation-period hellenistic-period:  1
panathenaic-amphora original-location attica:  2
panathenaic-amphora painting-technique-used red-figure-technique:  1
```

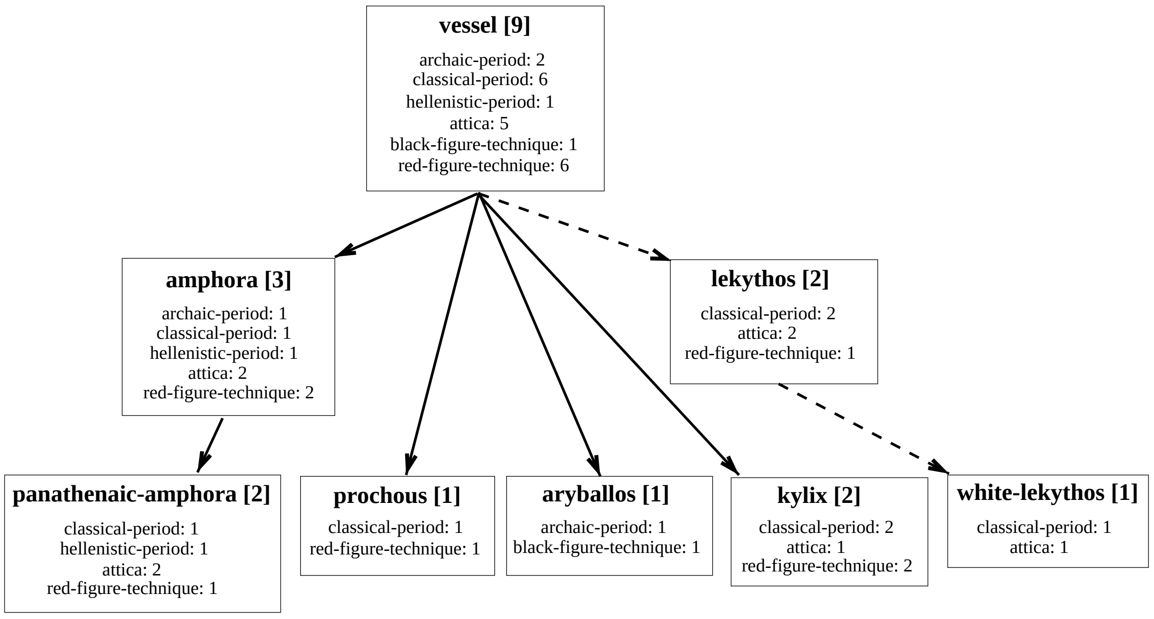
That is, both panathenaic amphoras originate from Attica, but each was created during a different historical period. Regarding the painting technique that was used, we only know that one of them was decorated with the red figure technique. For the third amphora, we have the information that it was created during the archaic period and that it was painted with the red figure technique. Thus, for the class amphora in general, the system will produce a different set of potential comparators:

```
amphora creation-period archaic-period:  1
amphora creation-period classical-period:  1
amphora creation-period hellenistic-period:  1
amphora original-location attica:  2
amphora painting-technique-used red-figure-technique:  2
```

Forwarding the properties of all nine previously visited vessels to the top of the hierarchy, we get the following list of potential comparators:

```
vessel creation-period archaic-period:  2
vessel creation-period classical-period:  6
vessel creation-period hellenistic-period:  1
vessel original-location attica:  5
vessel painting-technique-used black-figure-technique:  1
vessel painting-technique-used red-figure-technique:  6
```

Thus, we observe that of the nine Ancient Greek vessels that the user has examined, six were created during the classical period, two during the archaic period, a single vessel during the hellenistic, and so on. Including all the potential comparators collected by the system at each level in the diagram representing the hierarchy of the previously examined vessels, we can represent the possibilities as follows (each box in the diagram contains the potential comparators that are appropriate at each level in the hierarchy):



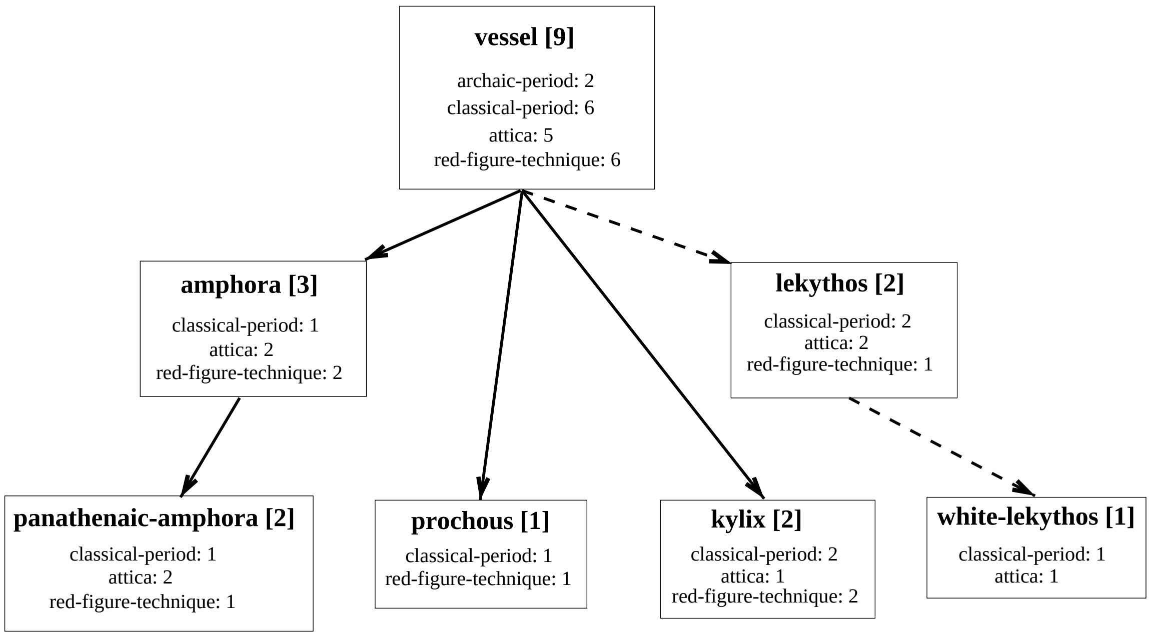
The relationship denoted by the dashed lines in the diagram, connecting the class of the exhibit in global focus (the ‘new’ object to be described, which in this example we are assuming to be another lekythos) with its superclasses and subclasses in the class hierarchy, is of particular importance to the comparisons module. In general, in choosing a comparator, we would like to give priority to past exhibits that are directly related to the current exhibit, i.e. to exhibits that are either similar to it (for example, comparing the current lekythos with all previously examined lekythoses), or which belong to a superclass (for instance, the current lekythos vs. all previously examined vessels), or which is a member of a subclass of the current exhibit (comparing the lekythos in the present description with a white lekythos visited earlier).

6.4 The Comparator Selection Process

Thus, in total we have a list of 27 potential comparators. We need to select one (or none, in case no single comparator is appropriate) for the current description. The selection process consists of 4 stages, 3 main and an initial pre-process, and we now examine these in turn.

6.4.1 The Pre-process

As it only makes sense in performing ‘contrast’ comparisons to compare the current exhibit with all the mentioned instances of a class (its own or one of its superclasses), this stage clears the list of those potential comparators having both an index of ‘1’ (i.e. their subject is a single entity) and an `arg2` filler which is different from that of the current exhibit for a given predicate in the list of targets (the rest still qualify for similarity-based comparisons). Thus, in our example, the candidate comparators are reduced to the 20 shown in the diagram below:



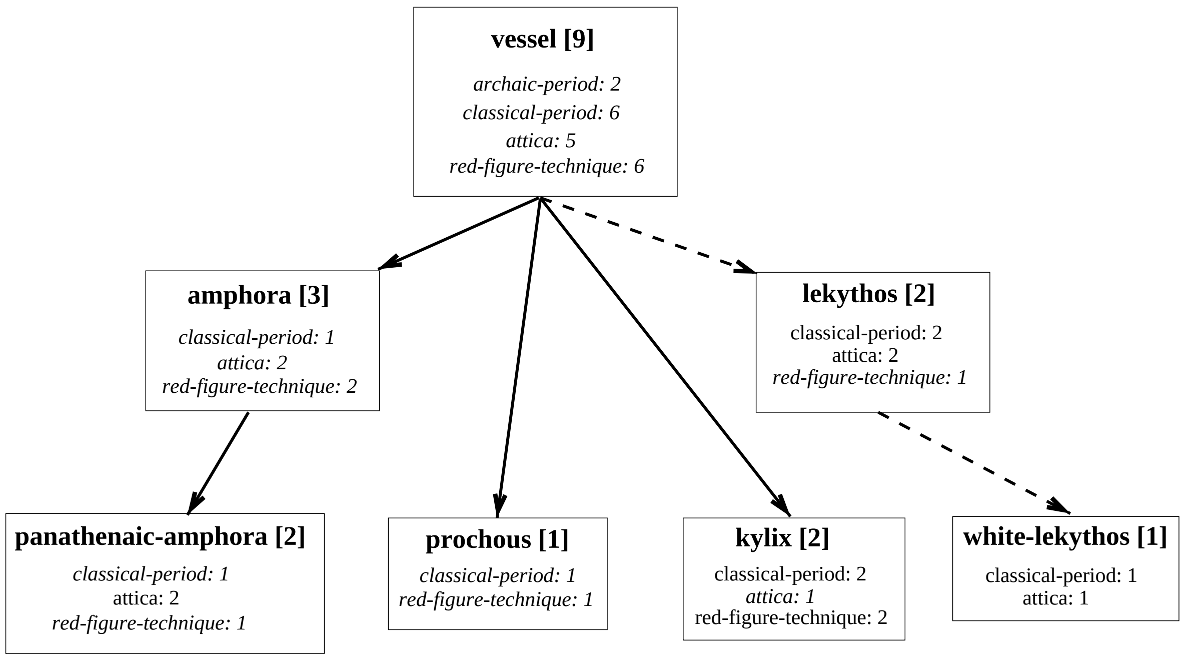
If all potential comparators fail to pass this check, no comparisons are performed.

6.4.2 Stage 1: Removing subsets and similar sparse classes

This stage consists of two steps:

a. Removing subsets

Since we would like to avoid making specific, full-clause references to past exhibits, the references that we will make in the comparisons must necessarily apply to all instances of any given class (or to the single instance, in case the set of mentioned members of that class consists of a single exhibit). Thus during the first step in this stage, the system removes all the comparators whose index is smaller that the size of their class (as shown in the numbers in brackets). Thus, the system will remove the potential comparators shown in the diagram below in italics:



b. Removing similar subclasses

For each of the comparators that was deleted during the first step, the system now identifies the comparators (if any exist) that:

(i) have an `arg1` entity which is a *subclass* of the deleted comparator,

(ii) have an *identical predicate and predicate filler* as the deleted comparator, and

(iii) are *not* directly related to the exhibit in current focus (i.e. are neither similar to it, nor belong to a superclass or a subclass of it).

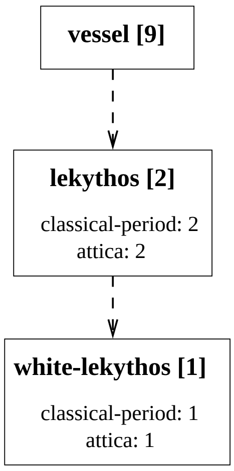
If any comparators match these criteria, the system will remove those whose index is smaller than the index of the deleted superclass; otherwise it will allow them to proceed to the second stage. The idea behind this second step is that we would like to refer to exhibits (or, again, to groups of previously visited exhibits) that are not directly related to the exhibit in current focus only when such exhibits exclusively possess a particular attribute of interest, i.e. when no other previously visited exhibit has the same filler for the predicate in question.

For instance, if the comparator `vessel original-location classical-period` had been deleted by the system earlier in this stage, it would now look for comparators such as:

```
amphora original-location classical-period
panathenaic-amphora original-location classical-period
kylix original-location classical-period
```

.. and so on. In our example, there are 6 vessels created during the classical period. Since two of those are the previously examined lekythoses, the system will discard all the other cases of comparators which relate to exhibits (which are not lekythoses) created during the classical period, as it makes more sense to compare the current lekythos with lekythoses, than, say, with kylixes or prochouses.

After the completion of this second step, there are only 4 potential comparators left for consideration:



Again, if all potential comparators fail to pass this stage, no comparisons are performed.

6.4.3 Stage 2: Removing weakest relatives

For each of the remaining potential comparators, the system checks if there are any identical comparators (i.e. with the same predicate and predicate filler) higher in the hierarchy list, and what their index is. For instance, for the comparator `white-lekythos creation-period classical-period`, the system will check the indexes of comparators of the following form:

```
lekythos creation-period classical-period
vessel creation-period classical-period
```

If such comparators do exist, then the system will either:

a. remove the subclass, if its index is lower than the index of the superclass, or else (if they are equal)

b. remove the superclass.

For instance if we have two lekythoses created during the classical period (and consequently at least two vessels created during that period), it would be meaningless to keep the vessel entry if we only have two vessels which were created during the classical period. On the other hand, since the superclass is confirmed (in stage 1) to consist entirely of members sharing this attribute, if the subclass contains fewer members than the class, then there is no reason for the system to maintain the subclass comparator.

The list now takes the following shape:

```
lekythos creation-period classical-period:  2
lekythos original-location attica:  2
```

Provided that the system actually reaches this point, this stage never returns an empty list.

6.4.4 Stage 3: Removing distant relatives

In this stage the system checks to see if there are any potential comparators that are direct relatives of the exhibit in current focus (i.e. comparators whose `arg1` entity is either the same as the `arg1` entity of the exhibit in global focus, a subclass or a superclass). Thus, in this case the system maintains both comparators, as they are both directly related to the current lekythos:

```
lekythos creation-period classical-period:  2
lekythos original-location attica:  2
```

If after this final stage there are still potential comparators left on the list, the system picks one at random. This stage is very likely to return an empty list, in which case the system uses a fall-back strategy, randomly selecting the comparator from the output list of the previous stage.

If the `arg2` filler of the comparator is the same as the `arg2` filler of the current exhibit, the system generates an RST-SIMILARITY, otherwise an RST-CONTRAST. In our example, assuming that the system randomly selects the comparator `lekythos original-location attica:  2` from the final list of potential comparators, the following text is generated:

> This exhibit is a lekythos, created during the classical period. It dates from between 470 and 460 B.C. It has a picture of an athlete ready to throw his javelin. It was painted with the red figure technique. *Like the previous lekythoses, it originates from Attica.* This lekythos is now exhibited in the National Archaeological Museum of Athens. In antiquity, javelin throwing was intimately bound up with the Greek way of life. Before it had become a feature of sporting life, the javelin was one of the weapons used by ancient Greeks in war and hunting. A javelin is a sharp, wooden spear about the height of a tall man.

6.5 Sample Navigation

The following texts illustrate a series of descriptions, indicating the comparisons that are generated:

> This exhibit is a panathenaic amphora, created during the classical period. It dates from 490 B.C. and it was painted with the red figure technique, the red figure technique is the opposite of the black figure technique. It originates from Attica.
>
> This exhibit is a kylix. *Like the panathenaic amphora, it was created during the classical period.* This kylix shows a young man who is seated and writes with a stylus. This kylix was decorated by Eucharides and was painted with the red figure technique. During the archaic period, the most poular method of writing must have been wooden tablets coated with wax, on which letters were written with the stylus and could easily be rubbed out and rewritten.
>
> This exhibit is an aryballos, created during the archaic period. It dates from the late 7th century B.C. and it belongs to the corinthian type. It is spherical in shape. *Unlike the previous vessels, which were painted with the red figure technique, this aryballos was painted with the black figure technique.*
>
> This exhibit is a panathenaic amphora, created during the hellenistic period. It dates from 320 B.C. *Like the previous panathenaic amphora, it originates from Attica.* The sport of running is so ancient that it is impossible to find out exactly when and how it started. Its importance was never in doubt, however; during antiquity it formed one of the key elements in a child's education. On the amphora you can see athletes running naked, as it was customary, except for the 'race in armour', in which athletes ran wearing their helmets, shields and greaves. There were many running sports, each characterised by the distance covered by the athletes. This panathenaic amphora is now exhibited in The John Paul Getty musem.
>
> This exhibit is *another* amphora. *Like the aryballos, it was created during the archaic period.* This amphora dates from the early 5th century B.C. It shows a warrior performing splachnoscopy before leaving for battle. Splachnoscopy is the study of animal entrails, through which people tried to predict the future. It was one of the most common divination methods used in the archaic period. This amphora was decorated by the painter of Kleofrades and was painted with the red figure technique.

7 Example session with Exprimo

8 References

Cheng, Hua, 2001. *Modelling aggregation motivated actions in descriptive text generation*. Ph.D. thesis, University of Edinburgh.

Dalianis, Hercules, 1999. Aggregation in natural language generation. *Journal of Computational Intelligence*, 15(4).

Halliday, Michael A. K., 1985. *An Introduction to Functional Grammar*. London: Edward Arnold.

Jurafsky, D. and J. Martin (eds.), 2000. *Speech and Language Processing: an introduction to speech recognition, computational linguistics and natural language processing*. New Jersey: Prentice-Hall.

Knott, A., J. Oberlander, M. O'Donnell, and C. Mellish, 2001. Beyond elaboration: the interaction of relations and focus in coherent text. In T. Sanders, J. Schilperoord, and W. Spooren (eds.), *Text representation: linguistic and psycholinguistic aspects*. Amsterdam: Benjamins, pages 181–196.

Knott, Alistair, 1998. The WAH/ILEX user manual. DAI report, University of Edinburgh.

Mann, William C. and Sandra A. Thompson, 1987. Rhetorical structure theory: A theory of text organization. Technical Report ISI/RS-87-190, USC Information Sciences Institute.

Melengoglou, Alexander, 2002. *Multilingual Aggregation in the M-PIRO System*. Master's thesis, University of Edinburgh.

Mellish, C., M. O'Donnell, J. Oberlander, and A. Knott, 1998. An architecture for opportunistic text generation. In *9th INLG*. Niagara-on-the-Lake, Ontario.

Not, Elena, Ion Androutsopoulos, Jo Calder, Aggeliki Dimitromanolaki, Vangelis Karkaletsis, and George Paliouras, 2001. Generation architecture and description of first M-PIRO prototype. M-PIRO Deliverable 1.2, University of Edinburgh.

Reiter, Ehud and Robert Dale, 2000. *Building Natural Language Generation Systems*. Cambridge, U.K.: Cambridge University Press.

Scott, Donia R. and Clarisse Sieckenius de Souza, 1990. Getting the message across in RST-based text generation. In Robert Dale, Christopher S. Mellish, and M. Zock (eds.), *Current Research in Natural Language Generation*. Academic Press. Paper presented at the 1989 European Natural Language Generation Workshop, Edinburgh, April.

Shaw, James and Kathleen McKeown, 1997. An architecture for aggregation in text generation. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI'97)*. Nagoya, Japan. Poster Session.