# Cooperative Visual Manipulation of Music Notation

P. BELLINI, P. NESI, and M. B. SPINU
Universita degli Studi di Firenze

As computer technologies and their potential emerging applications spread out, new needs have been detected for computer-based applications of music; cooperative music notation editing both in orchestras and music schools is one of them. This article is the only public document describing the details of cooperative work on music notation of MOODS (Music Object Oriented Distributed System). MOODS is a synchronous real-time cooperative editor for music scores. Its architecture includes mechanisms for troubleshooting conflicts in real-time, managing histories of commands and versioning, and for performing selective undo. The system also includes specific solutions in order to control the editing on the account of editing permission profiles. The most important aspects of MOODS associated with cooperative work on music notation scores are reported herein. The article highlights the general problems of cooperative systems and provides rationales for the solutions, which were found to build MOODS. The MOODS system has been implemented and validated thanks to the endeavor of several musicians in orchestras, music schools, and project partners. A prototype has been demonstrated in public at the Scala Theatre in Milan, Italy.

Categories and Subject Descriptors: C.2.4 [**Computer-Communication Networks**]: Distributed Systems—*distributed applications*; D.2.2 [**Software Engineering**]: Tools and Techniques—*user interfaces*; H.1.2 [**Models and Principles**]: User/Machine Systems—*human factors*; H.5.3 [**Information Interface and Presentation**]: Group and Organization Interfaces—*computer supported cooperative work*; H.5.5 [**Information Interface and Presentation**]: Sounds and Music Computing—*methodologies and techniques*

General Terms: Design, Human Factors, Algorithms

Additional Key Words and Phrases: Collaboration of music notation editing, computer-supported cooperative work, user interface management systems, cooperative music, collaborative systems, distributed music, electronic lectern, consistency control, selective undo, neutral version, additional command list

Authors' address: Dipartimento di Sistemi e Informatica, Universita degli Studi di Firenze, Via S. Marta 3, 50139 Firenze, Italy; email: nesi@dsi.unifi.it, nesi@ingfi1.unifi.it, http://www.dsi.unifi.it/~nesi, http://www.dsi.unifi.it/~moods.
The MOODS project was partially funded by the European Commission in the HPCN (High Performance Computer Networking) domain of ESPRIT IV (http://www.dsi.unifi.it/~moods). Project partners were DSI (Department of Systems and Computer Science, University of Florence) as project coordinator; Teatro alla Scala (Milan); BMG Ricordi; Fiesole Music School; ELSEL srl; and Shylock Projects. Hewlett-Packard Italy is also partially supporting us in computer music projects.

## 1. INTRODUCTION

Traditional editors are not cooperative/collaborative. They are designed to be used by single users who may perform changes. Multiple invocations of the same editor can be used at the same time but this does not allow the cooperative manipulation of the same file: changes performed on a certain editor are not replicated on the others.

The final result holds merely the changes performed by only one editor that typically is the last editor that has been closed. In many fields, the realization of cooperative tools that allow people to work together is one of the most innovative challenges of the future.

Asynchronous applications are far from the concept of real-time collaboration where the same information is shared and cooperatively changed by multiple users. Tools such as CVS [Grune 1986] and RCS (which can be considered tools for the support of asynchronous work in development teams) allow preserving file integrity by tracking the versioning of file evolution. Typically, they are long-term collaborative systems where changes performed by one user are made visible to other people only some weeks or a few days later. Conflicting changes become visible only when different changes are merged. In such systems, the conflicts arising from the merging of different versions are managed sporadically and asynchronously with respect to the users' work. Conflict troubleshooting may bring about the elimination of some changes and give rise to the production of the hierarchical relationship of versions which may be discussed and managed by a version manager [Lee et al. 1998; Munson and Dewan 1994].

In synchronous applications, each performed change is made visible to all users and is somehow saved. Tools for cooperative work can be very beneficial when discussing common documents or graphs in real-time or quasireal-time. Interaction and collaboration among people are extremely relevant for many new emerging applications. The problems due to versioning management that are found in asynchronous applications may also be found in synchronous cooperative applications, due to the presence of communication delays and user communication protocols. The TimeWarp toolkit [Edwards 1997; Edwards and Mynatt 1997] proposes suitable solutions for detecting and managing conflicts in collaborative applications.

Cooperative techniques and solutions are not restricted to simple textual editors. The study and implementation of cooperative visual/graphic editors allow exploring new areas that satisfy new needs. Basically both textual and visual editors share several technical aspects that characterize the cooperative architecture and related solutions. These aspects are the fundamental issues of cooperative technology: synchronous and asynchronous collaboration, collisions, time granularity, working modalities, section size, undo support, history of changes, context modeling, real-time response, nonuniform user groups, and so on. Most of these aspects have a large impact on the user interface and on the operative functionalities of cooperative editors: What You See Is What I See, What You See Is What I May See [NewmanWolfe et al. 1992; Stefik et al. 1986]. For the former, editors of the cooperative environment are totally

joined; whereas for the latter, the updating of the changes performed by a user can be delayed according to specific protocols to interact with the cooperative tools.

A vast literature exists on techniques and examples of cooperative editors. CES (Collaborative Editing System) was one of the first collaborative editors [Greif and Sarin 1986; Sarin and Greif 1985]. Its granularity (segment dimension of the document) was of one section and no real-time updating was addressed. Other examples of non-real-time editors are Quilt [Fish et al. 1988; Leland et al. 1988] and PREP [Neuwirth et al. 1990, 1994]. GROVE is a fine-grained editor in which the grain is the textual line [Ellis and Gibbs 1989; Ellis et al. 1991]. These techniques have not been limited to simple textual editors; see, for example, Newman-Wolfe et al. [1992], Sun et al. [1998], Berlage [1994], and Myers and Kosbi [1996].

The study and implementation of cooperative visual editors allowed exploring new areas, thus fulfilling new needs. A solution for cooperative work on images was presented in the Ensemble graphic editor [Newman-Wolfe et al. 1992] as part of a distributed conferencing system.

A formal model was discussed by Sun et al. [1998]; the model dealt with the studying and preserving of consistency, causality, and intention of commands produced and received in real-time cooperative graphics editors. Cooperative solutions to draw graphic structures can be found in Greenberg and Marwood [1994], Karsenty and Beaudouin-Lafon [1993], and Moran et al. [1995]. In Hymes and Olson [1992], Stefik et al. [1987], Rhyne and Wolf [1992], Olson et al. [1992], and Edwards et al. [2000] cooperative supports for brainstorming or designing were presented. More complex cooperative graphic editors are those where the graphic screen depicted to the user is made of graphic elements representing data model relationships that can be manipulated. They are not simply sharing a common image; they follow a sort of cooperative blackboard paradigm. In Borghoff and Schlichter [1998], an interesting review of cooperative applications is reported.

In order to support the implementation of cooperative text editors, some tools have been proposed such as MACE [Pelimuhandiram 1991], and DistEdit [Knister and Prakash 1990, 1993]. In Dewan and Choudhary [1995] a formal model for coupling user interfaces of cooperative applications has been proposed. In Dourish [1998] the Prospero toolkit for the realization of cooperative applications has also been discussed. GINA [Berlage and Genau 1993; Berlage 1994], and Amulet [Myers and Kosbi 1996] are frameworks for building cooperative and multiuser applications capable of preserving consistency by using undo/redo mechanisms. These tools can be suitably used for both text- and graphics-based applications. In several cases, these tools support both synchronous and/or asynchronous approaches. Synchronous approaches are typically classified as loosely or tightly coupled. In loosely coupled systems the clients share the same data model, and have a separate user interface. In tightly coupled applications the cohesion is performed at the user interface level, which implies that each mouse movement or window scrolling performed by a client is replicated on the screen of all the other clients. Different types of conflicts have to be managed in these conditions.

Despite the large research activity on cooperative systems for visual/graphic environments, the cooperative manipulation of music notation has never been considered, even though it is one of the most ancient visual and cooperative activities. To give the reader an idea, it is enough to imagine work performed on music scores during performance rehearsals: the addition of details, the fixing of fingers, the highlighting of specific points, the changes of arrangements, and so on. Furthermore, the visual editing of music is one of the fastest growing graphic applications. More than 30 different music editors can be found on the market (among these are Finale, Sibelius, Igor, SCORE, Capella, etc.).

The cooperative manipulation of music notation presents several differences with respect to manipulation of other visual or graphical elements. The manipulation of graphic elements or components can range from applications where cooperation is simply reduced to graphic elements of insertion and deletion in a common area, for example, in brainstorming tools, up to applications where the graphic elements have specific semantics and can be placed according to certain given rules (e.g., flow charts, state charts). In music, such a problem is important since music notation consists of several hundred different symbols that have to be combined according to specific rules related to both semantics and visualization. The same symbols can be placed (associated with different symbols), ordered, and oriented in different ways. These aspects are relevant for understanding and interpretation of any music; some visual configurations have no meaning, and others are possible only in certain contexts. That is why the writing of music notation has to be assisted or automated in order to obtain consistent music notation constructs that can be correctly interpreted by both human beings and the MIDI player.

Before introducing our work on cooperative manipulation of music notation, a more detailed description of the needs related to this kind of application is required. The cooperative editing of music is traditionally employed in orchestras during rehearsals to produce fully annotated versions of music score sheets for the final performance. In this event, many time-consuming operations, together with versioning management, are typically performed, which can be strongly reduced thanks to a cooperative editor for music notation.

### 1.1 Reasons for Cooperative Working on Music

The amount of information managed by orchestras during rehearsals and final performances is huge: the main score and parts are typically maintained and prepared with the effort of several people. During rehearsals, the orchestra conductor works on a music score by reading and manipulating all music parts, whereas the musicians may read and manipulate only their specific parts according to their roles in the orchestra.

During rehearsals, some changes are decided by musicians and reported on parts; for instance, by adding dynamics, expression marks, string bowings, fingering, and so on. More complex changes, such as the deletion or addition of music sections, their movements along the music score, music arrangements, transposition, and the like, are decided by the conductor or composer and they should be shown in real-time to musicians. These modifications are very time

consuming, and are typically performed in operas, ballets, or new symphonic works. At present, the archivists cut and paste with glue and scissors to perform these changes, clipping pages together, writing passages to replace rests, and so on. This process can take a few hours, a couple of days, or sometimes weeks, thus delaying and fragmenting rehearsals. Relevant changes, which should be decided in advance, are often identified only during rehearsals, causing long and tedious pauses. Such major on-site decisions are especially commonplace with stage pieces like operas and ballets, where decisions on changes can arise from the various needs of singers or stage directors, from acoustical problems, and so forth. The pauses during rehearsal can be regarded as unwinding moments. They often turn out to be a break in the necessary concentration.

When different musicians use the same paper version of a music score, previous modifications must be deleted (if possible) and new changes made on the same score. If the marks are too heavy, the scores or parts must be replaced at significant cost. Both deleting old changes or eventually replacing scores are particularly time consuming. On the other hand, it is sometimes necessary to keep various versions of a set of scores and parts in the archive, because they bear the interpretation marks of important interpreters or because the same version may be needed some years later. The costs of storing, maintaining, or replacing multiple unique sets of parts can become exorbitant both for theaters, archives, and for publishers. As a result this situation culminates in theaters working with music scores borrowed from publishers. Besides, the logistics of the institutions borrowing music material or storing sets of music scores and parts (for opera, theater, or orchestra archives) make it difficult, if not impossible, in many cases to retain copies of all the various versions of a given work. At best, a copy of a modified score may be set aside, but not the full set of parts. When a specific version or production is repeated some seasons later, the modifications have to be regenerated, creating significant costs. A practical solution for the storage of multiple versions and in the meanwhile greatly reducing the amount of repetitive labor involved is an important goal for many performing organizations and publishers.

These problems can be managed by a cooperative system of computerized music lecterns/stands. In the past, the idea of an electronic lectern for music was a kind of daydream in the minds of many people. We prefer to use the term lectern. Our research group started in 1994 to study the problems of producing a network of computer-based music lecterns oriented to musicians. Such lecterns were conceived while paying attention to editing/showing music scores by means of a suitable interface. Several goals were considered in order to remove paper scores from musicians' lecterns during rehearsals and final performances, with the eventual target of solving or at least reducing the abovementioned problems. Computerized music lecterns can be used by musicians to save their work/changes, tracking and managing their versions. They can reduce the cost of rehearsals and improve the quality of the service offered by music publishers, and so on. This idea was deeply studied and a solution was found and implemented with the MOODS project of the European Commission (Music Object Oriented Distributed System) [Bellini et al. 1999]. The points of

Fig. 1. A first violin playing with a MOODS "electronic lectern" at the Scala Theature.

view of the potential end-users of MOODS (orchestras, music publishers, and music schools) were seriously taken into consideration.

In this article, details about the MOODS cooperative editor for music notation are presented and discussed. They include aspects related to:

(1) MOODS general architecture overview and solution (see Section 2);
(2) cooperative work on music and consistency preservation (see Section 3);
(3) orchestra configuration and subsequent network of computers/lecterns with mechanisms for regulating the authorizations to manipulate notation symbols in a cooperative way (see Section 4);
(4) mechanisms for undoing and tracing versions along the music evolution (see Section 5);
(5) experimental results and validation of MOODS solution with musicians (see Section 6); and
(6) *overview of the object-oriented model of MOODS (see http://www.dsi.unifi.it/~moods/moods/acm/oo-overview.html).*

Conclusions and future expected work are reported in Section 8.

## 2. MOODS GENERAL OVERVIEW AND SOLUTION

The MOODS system consists of an integrated system of computer-based lecterns for the cooperative editing, visualization, and execution of music scores specifically oriented to musicians (see Figure 1) [Bellini et al. 1999]. MOODS is an innovative solution to automate and manage the large amount of information

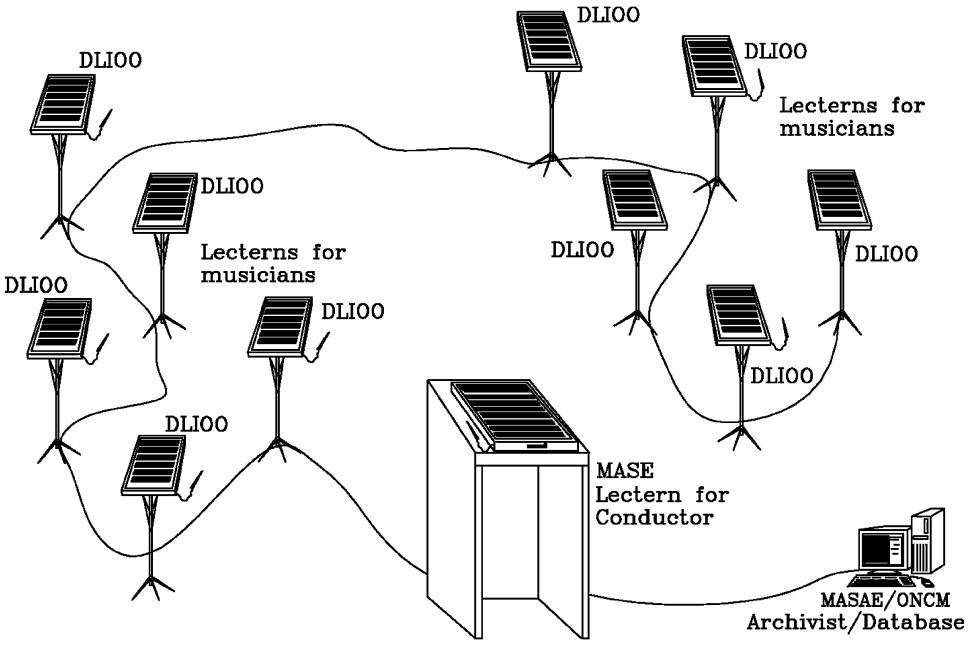


Fig. 2. MOODS General Architecture.

used by: (i) orchestras during rehearsals and public performances of concerts, operas, ballets, and so on (ii) music students during lessons in conservatories and music schools; and (iii) publishers executing massive amounts of music editing. The target end-users are theaters, itinerant orchestras, musicians' groups, music schools, television network orchestras, and music publishers.

The main features and advantages of MOODS are its ability to reduce the time needed to modify main scores and parts during rehearsals in a cooperative way; it can manage (load, modify, and save) instrumental and personal symbols on main scores and parts, thus saving artistic details never saved before. In a few minutes it can visualize scores from the theater archive on musicians' lecterns. With MOODS, it is possible to manipulate the main score and parts in real-time. Therefore it becomes possible to present the final version of the score to the musicians in real-time with the changes included.

Figure 2 shows the MOODS system including:

— a set of single part lecterns for musicians, called DLIOOs (distributed lectern interactive object-oriented) for editing and visualizing single score parts. Music for musicians is typically visualized in a different way with respect to the parts visualized in the main score observed by both conductors on MASEs and archivists on MASAE. Typically the parts present a more concise notation, for example, the adoption of generic rests and repeat symbols is quite usual on parts and not present in main scores;
— one or more lecterns called MASE (main score editor), used by directors, conductors, and directors of the chorus, in order to show, visualize, and modify the main score. In this case, the lectern displays the main score,

which is composed of all parts played by musicians, but formatted in a different manner;

— a general manager and music editor MASAE (main score auxiliary editor), to be used by the archivist, which can make global modifications on the main score during rehearsals or score revision. The archivist can also configure the orchestra structure by using the ONCM (orchestra network configurator and manager) tool. The changes on the main score may involve all parts simultaneously (e.g., a measure deletion), and/or may be so relevant that they have to be discussed with the conductor (the change of some notes in a traditional music piece);

— a database workstation for managing the music archive.

Hereafter, the term lectern is used for both MASEs and DLIOOs.

At first glance, the MOODS music editor may not appear much different from classical music editors (such as Finale, Score, Sibelius, etc.). Most of the music editors do not help musicians to produce correct music scores.

The automatic positioning of music symbols is not currently available in most music editors. They allow the free placement of music symbols in all positions on the staff or page, disregarding the formal/semantic relationships among them. The lack of support for formal relationships among music symbols is mainly due to the complexity of music notation, which includes many exceptions [Byrd 1984; Ross 1987; Rader 1996; Roush 1988; Blostein and Haken 1991; Gourlay 1986; Heussenstamm 1987; Wood 1989]. Therefore, most notation-oriented music editors allow the placement of several notation symbols in incorrect positions. This may result in producing strange, unreadable, and thus incorrect music scores. Commercial music editors for publishing permit the placement of music notation symbols on the staff (pentagram) or on the page to be printed rather than collecting relationships among symbols and organizing the visual information on that basis. This is easily verified, observing that for the position of markers (expressions, accents, fingerings, etc.) of notes the user has to select the marker and place it where it is needed without associating it with the note to be marked. Therefore, professional music editors are powerful, but become a difficult matter if used by nonexpert users in music notation. The automatic positioning of music notation symbols permits symbols to be placed in the right position without musicians mastering all the formatting rules. For example, when an accent has to be placed on a note, and this presents some other symbols (slurs, other accents, etc.), the latter and the new symbol have to assume specific relative positions. The automatic positioning allows such positioning work to be done for the user.

The cooperative editing of music has to be independent of the window size used for interacting with the music score. This means that the music notation model has to be mainly symbolic, which is something quite different from the music modeling typically used for traditional single-user music editors.

— Parts have to be considered as portions of the main score, and have to be shown as different views of the same main score. In classical music editors, it is possible to pass from the main score to parts but such a process is

not direct and reversible in all cases. This means that they do not have a unique representation of the music for the main score and parts. A score file representing a main score is physically different from the one modeling a single part.

— The editing of different music portions of the main score at the same time for both conductors and musicians has to be possible. The editing of the main score and parts as a unique model by using separate windows/views of the same data is not possible in classical music editors. They allow editing either the main score or the single parts but changes performed on one view are not replicated in the others.

— Visualization of the same music by using different zooming factors, formatting rules, justification rules, line breaking, and in some occurrences symbols, and the like, is not possible with classical music editors. On the contrary, in cooperative systems such a practice is strongly recommended since users, being several, may have different needs for visualization and interaction with the music score. In producing music scores, it often occurs that different formatting rules for music notation are used in the main score and in the parts (in parts, repeat symbols and generic rests, are used, whereas they are typically avoided in main scores).

For these reasons, visualization and manipulation rules for the main score and parts are different, justification is different, line breaking of view is different, and formatting rules have to be different [Bellini et al. 2001].

A solution for coping with these problems is defining a formal and unified model for representing music both in the main score and parts [Dannenberg 1993]. In Dannenberg [1990] a solution was given by suggesting the adoption of a unique model and distinct views in order to show music on the basis of each specific need and context. A similar solution has been adopted and improved in MOODS so as to maintain a unified model for the main score and parts [Bellini and Nesi 2001]. The music notation model adopted in cooperative systems has to be abstract enough to allow the interaction with music at a symbolic level: adding and deleting symbols, changing symbol features, and making these changes available regardless of visualization details and rules which can be those fitting either the main score or parts. This solution is really far from the solutions adopted by present music editors. In the case of cooperative editing, each notation element has to be unambiguously identifiable in any view in which it is manipulated, including the information concerning the author who manipulated the symbol.

MOODS is based on a specific music format that was presented and compared with several other notation models: SCORE, FINALE, NIFF, SMDL, and MIDI. For a comparison of music notation formats please see Selfridge-Field [1997]. As suggested by the acronym MOODS, the system is based on an object-oriented model of music. More recently the format of MOODS has been modeled in XML thanks to the results of the WEDELMUSIC project [Bellini and Nesi 2001].

Music on lecterns has to be well formatted, respecting specific relationships and proportions among notation symbols of the score in order to present an easily readable music score to the musicians. MOODS helps musicians produce

correctly formatted music scores in a easy way. To have correctly formatted music scores means to automatically generate relative positioning of notation symbols. This is possible by collecting during music editing the relationships among notation symbols and by using an automatic formatting tool for the music notation symbols. The modifications performed on a lectern have to be reproduced automatically on other lecterns since the graphic features of each and every lectern can be different. Changes performed on the main score via MASE or MASAE have to be replicated automatically on DLIOOs lecterns according to their visualization requirements, which is done by sending out symbolic commands.

In MOODS, the symbolic and unique representation of the main score and parts is managed by a specific engine for the automatic formatting of music according to specific formatting rules. The music formatting includes the definition of: stem direction, beaming groups, positioning markers associated with notes, avoiding collision among notation symbols, ordering markers, justifying measures, and so on. Formatting rules may differ according to user needs, as well as to different views of the same music score (e.g., the same main score can be formatted either to be viewed on the screen, using window size, or it can be printed using paper size). In MOODS music language and model, the solution chosen was the integration of a real-time inferential engine called MILLA (music intelligent language for automatic formatting) to arrange automatically symbols on the screen on the basis of rules [Bellini et al. 2001]. To change the visualization of music on their lecterns users can customize the MILLA rules of their lecterns.

For these reasons, we decided to implement two different types of clients (MASE and DLIOO) for conductors and musicians. They have different needs in term of user interface, even if they share the same music notation model. A more flexible solution could be to have only one kind of client (for both conductors and musicians) and this could be capable of working with different views. In that case, the user might decide to view either a part or the main score according to his or her needs. This could simplify the architecture at the expense of increasing the complexity of clients. To have a unique type of client can be more suitable for music teaching rather than for musicians in orchestras. According to a performed requirement analysis, musicians in orchestras typically do not have an interest in browsing the main score. When they have to begin to play in the middle of a piece, they have in their scores some small notes (cue notes) to guide the musicians in starting again to play. On the other hand, this solution could be more useful in working with music pieces where several different parts for the same instruments are present (e.g., main, first, and second violins). In such an occurrence, a violin may be interested in browsing the music of the others so as to understand the general desired effect. In this case, a specific real-time cooperative work/discussion of a subgroup of musicians could be activated. In the proposed solution, this functionality may be partially explored by activating more clients on the same computer/lectern.

Before starting to use a MOODS system, it has to be configured according to the orchestra organization in terms of parts/instruments and by the cooperative work rules determined by the relationships among musicians and parts,

including therefore the permissions assigned for the music notation editing. Orchestra network configuration details are discussed in the next sections.

Typically, cooperative editors are modal, meaning that they provide different functionalities in different operative conditions, such as scrolling/viewing, editing, and so on. In certain systems, different roles may be assigned to different users, and thus the modalities may be different on different terminals. In MOODS, only the MASAE station controls the switching of modalities. MOODS is a distributed heterogeneous modal editor that can operate in two ways: EDITING or EXECUTION. In EDITING, the music can be edited and viewed, and therefore it includes the classical modality of visualization. EXECUTION modality is used during music execution/playing and includes a mechanism for the automatic page turning of music on the lecterns. Please note that MOODS EXECUTION has nothing to do with audio generation from music notation, which is typically called MIDI playing.

## 3. COOPERATIVE WORK

When MOODS is in EDITING mode, the music can be cooperatively manipulated by musicians on DLIOOs, by conductors on MASEs, and by the archivist on the MASAE station in order to prepare the music for the final performance and eventually for the EXECUTION. In the EDITING mode, the changes are replicated in real-time to the other users working on the same data.

### 3.1 EXECUTION Mode of MOODS

During EXECUTION, MOODS distributes the music score to all lecterns. The page turning is automatic and this means that musicians do not have to worry about it. The system provides the right page at the correct time to both DLIOOs and MASEs. It should be observed that different lecterns might have page turning at different time instants. This is due to: (i) the different amount of written music that has to be executed by each instrument (more precisely, the screen space covered by such music), (ii) the dimensions of the lectern, and (iii) the different formatting rules applied [Bellini et al. 2001]. The velocity of music execution is initially set on the basis of the metronomic indication in terms of beats per minute. This ideal trend can be adjusted in real-time by increasing/decreasing the execution rate.

During execution, the next page is constructed on MASEs and MASAE starting on the left as sooner as the measures have been executed (see Figure 3, bottom). The screen displays two parts: the current page (on the right side of the right window shown in Figure 3), which is gradually reduced as measures and pentagrams are executed. On each DLIOO, there is a horizontal divider that drops across the screen as time passes. Above the divider the next page is being exposed, while below the divider the current page remains. The DLIOO shows the following page which is disclosed from top to bottom (see Figure 3, top). In this way, the modality which is generally adopted by musicians to read scores is left substantially unchanged. They can keep on passing from the bottom right corner, when leaving the current page, to the top left corner when observing the following page. In fact, musicians may always read several measures in advance with respect to the execution point. See Section 6 for further details.

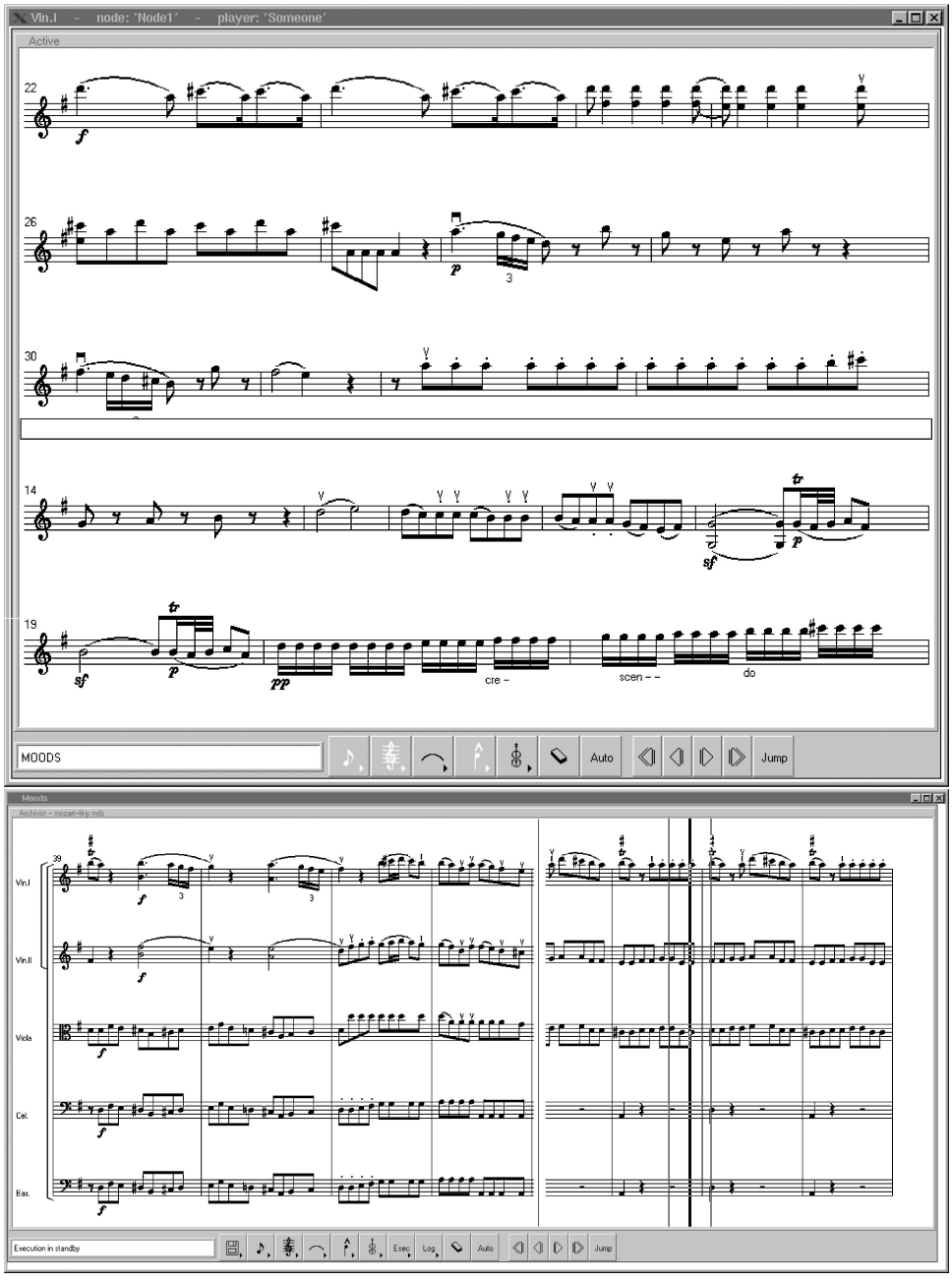


Fig. 3. DLIOO (left) and MASAE (right) during execution.

In EXECUTION, the page-turning mechanism is driven by the MASAE, where the divider is vertical, as the score always has measures advancing simply from left to right without multiple grand staves per page. The page turning is synchronized according to the point where the orchestra is executing the music (marked with a mean line of a group of three, see Figure 3, bottom). The MASAE governs the whole communication process with the lecterns, these being totally enslaved to its control.

The rate of music execution is initially set on the basis of the metronomic indication in terms of beats per minute. This ideal trend can be adjusted in real-time by increasing/decreasing the execution rate by a given percentage. The adjustment can be easily made by the MASAE operator comparing the position of the line marking the ideal execution instant on the score (the median line in a group of three vertical lines) with the sound produced by the orchestra. This implies that the person devoted to this work has to be capable of reading music and following the score. For musicians, this is a trivial task. The archivist typically follows the music on the score during rehearsals and performances as a sort of quality control on musicians and on the prepared scores. In page turning, the precision up to a single note is not really needed; it is enough to visualize the current measure plus some measures in advance since musicians typically read the next three or four measures while executing the current one. This condition is abundantly met even when a musician plays the last measure page, since subsequent measures are available in the top left corner of the window shown.

Automatic page turning has several advantages such as: (i) musicians and conductor do not lose their concentration because of page turning, (ii) the prevention of any noise produced by turning pages, and (iii) the reduction of discontinuities in the sound due to the turning of pages. At present, trying to make sure that rests are placed in the last measure of the page reduces the latter problem. The first solution is not always feasible for all music parts. This is the reason why several pianists need someone turning pages in their place, and it is for the same reason that when two musicians read music from the same lectern, the one on the left has to turn the pages whereas the other goes on playing, thus producing some discontinuities in the music played.

In MOODS, the execution rates, both the ideal one and that adjusted by the archivist, can be saved for any further reuse. Therefore MOODS is able to regenerate the registered execution rate of each performance and rehearsal. This has a relevant impact on improving the quality of orchestra work and on studying the performances of important conductors and musicians in music schools.

### 3.2 EDITING Mode of MOODS

In the EDITING modality, MOODS is a fully cooperative distributed editor of music. Musicians and conductors use single part or main score lecterns to adjust the music score in a cooperative way, during rehearsals, in classrooms, and during composition. In EDITING, all musicians may perform changes simultaneously on the same music score. Each lectern can work on the assigned part along the whole composition regardless of the position of the other lecterns

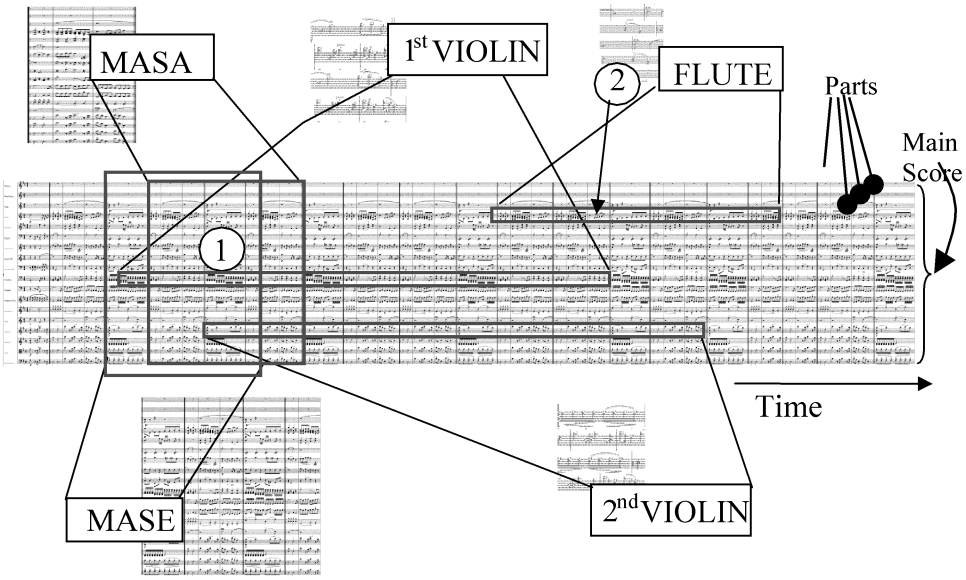


Fig. 4. Relationships among lecterns and the music model during EDITING.

while visualizing possible changes that others perform on the same part. Each musician may use the lectern to navigate on the assigned part independently, so as to visualize measures forward and backward. Therefore, each lectern may have a different vision of the same pieces of information, that is, the music score (see Figure 4). In MOODS, there is a unified model for the main score and parts; the main score is visualized as a set of parts. Formatting aspects are managed by different MILLA rules [Bellini et al. 2001]. Parts are distributed via the network to the DLIOOs (see Figure 4). The unique music modeling managed by the MASAE is based on a set of distinct parts, which are joined together in real-time to form the main score in the MASEs and MASAE editors and are singularly used by the DLIOOs.

The amount of music visualized by each lectern (in terms of number of measures) depends on the lectern's physical dimensions, on the graphical dimensions of the music symbols along the measures, and on the formatting rules. Modifications on music are performed in the unique model and are redistributed to the involved lecterns. Each change performed on a lectern (DLIOO or MASE) is sent in real-time to the MASAE and, from there, both to other lecterns requesting the same part (DLIOOs) and to the main score editors (MASEs). Changes on the score are visualized only on the lecterns involved in that change. For instance, in Figure 4, the change performed by MASAE in (1) on the first violin part will be seen in real-time by the first violins and by MASEs which are visualizing those measures; whereas change (2), performed on the flute lectern is communicated to the MASAE but is left unseen on the other lecterns, since no other lectern works on the same score piece in Figure 4.

Each modification performed on the current music score by using MASE/MASAE is sent in real-time to the corresponding lecterns thanks to

the association of each part with the corresponding physical lecterns connected through the network, according to the configuration (see Section 4). Modifications on the score may involve all music symbols.

### 3.3 Cooperative Architecture of MOODS

In MOODS, cooperative techniques have been used to visualize and manage the EDITING modality. According to the above-mentioned description, cooperative editing has to provide real-time response to distributed editors. In order to focus better on the relevant problems, MOODS has to be considered a synchronous cooperative editor of music notation. Music notation symbols share several relationships that are reflected in the data model. The relative positioning among music symbols is decided according to rules. Users define/modify merely the relationships performing typical operations such as insertion, delete, modify, and the like.

To better analyze the cooperative architecture of MOODS, problems of collisions, section size, and time granularity have to be considered.

A collision occurs when two or more users edit the same data/file section at the same time. Typically, cooperative editors adopt specific mechanisms to grant the resolution of collisions. Collisions can be resolved by buffering the requests for change and by applying them in the same order they arrived or by defining specific end-user working protocols. In the first case, the results can become semantically or syntactically invalid. At a lexical level, this problem can be avoided allowing the changes only at the level of the single word. In the syntax and semantics levels, the integrity can be preserved only by blocking the cooperative modification of sections or phrases, which can be achieved by using specific end-user working protocols. In such cases, the size of the sections is very important to define the operative functionalities and interaction protocols of the cooperative editor.

In many instances, methods to preserve sections are defined and the changes performed are updated answering to specific rules. Therefore, cooperative editors should be classified on the basis of two aspects: the dimension of the sections (i.e., the granularity), and the frequency of updating the sections by exporting the changes already made to other editors in the cooperative environment. In this view, if changes are immediately updated when performed, the editor is called real-time. Changes have to be updated in a short time otherwise the cooperative editor risks becoming similar to an asynchronous teamwork tool. The availability of very small sections allows managing small entities in real-time.

In some applications, users that cooperatively manipulate the data prefer to make their work visible to others only when the work is lexically, syntactically, and semantically consistent. During the manipulation of the information the section is often kept locked, preventing other users from manipulating the same piece of information. For this reason, the updating of the information can be automatic or user-driven.

Another important feature is the time needed to perform the information updating. Different combinations of section size and updating granularity/frequency can fit for different applications. The ideal solution is to allow

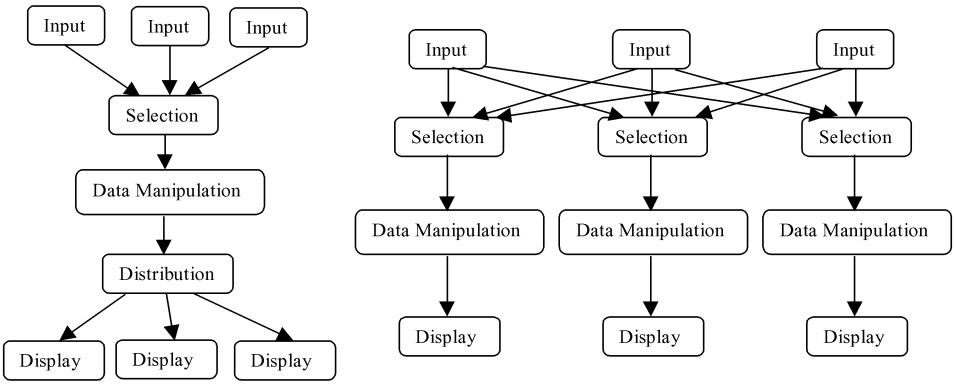


Fig. 5. Fully centralized (left) and distributed architectures (right).

real-time updating without the definition of the section size (minimal size taken), so that the cooperative work of each user towards user interfaces shared by others is made completely transparent. This type of solution is called unconstrained [Sun et al. 1998].

The concept of section can hardly be defined in the cooperative editing of music. There are several structures in music that can be considered as sections: single part, layer/voice with its measures, measure with its figures (chords, notes, rests, beams, etc.), beam with its figures, chord with its notes, notes with its markers, and single music notation symbols (as the single word in the text). In general terms, it is very difficult to agree on what can be considered a lockable section in the cooperative editing of music. That is why the chosen solution has been to consider the single section as the single music notation symbol. On the other hand, a mechanism to control the editing of music symbols has been defined (see Section 4).

These problems may have different solutions depending on the general architecture chosen [Newman-Wolfe et al. 1992]. While considering the architectures, the extreme cases are fully centralized or distributed (see Figure 5).

Centralized architectures present a centralization of the selection performed on the distributed user interface, a serial application to the data shared, and the corresponding distribution of produced results for the visualization. In the event of a central selection, when data manipulation and distribution are present, the distributed editors are only simple user interfaces to collect user interaction and visualization.

In fully distributed systems, the selection on the user interface, data manipulation, and visualization are replicated in each location of the distributed architecture. This means that each action performed by the user on the remote terminals has to be sent to the other terminals in order to keep the data integrity. In such instances problems of both synchronization and consistency maintenance can occur.

MOODS is a synchronous tightly coupled cooperative editor for music notation that has a hybrid architecture differing from that of DCS in Newman-Wolf et al. [1992] (see Figure 6). MOODS presents a distributed selection and a

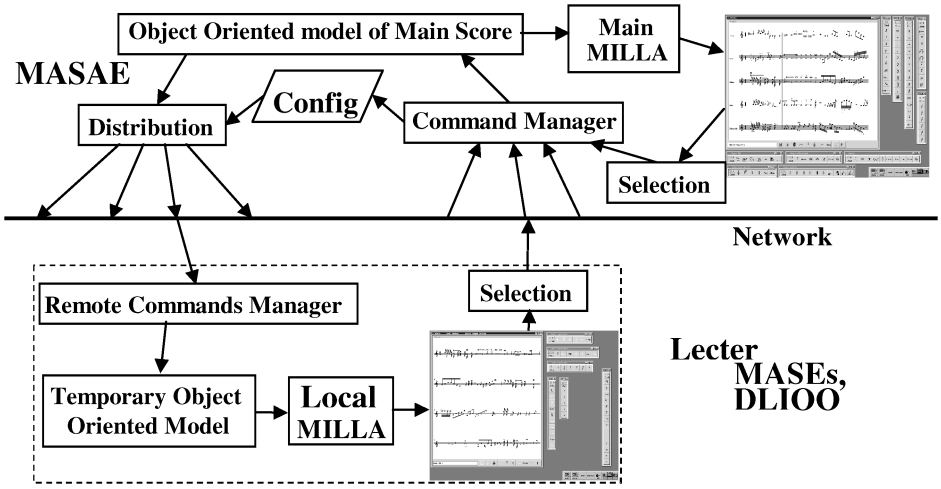


Fig. 6. MOODS hybrid cooperative architecture.

central coordinating process that maintains a complete master copy of the internal data representation: the full collection of music parts out of which both visualizations of the main score and parts are produced. Each single cooperative editor has its user interface for both selection and visualization of music. The music visualized/edited by remote terminals (Temporary Object-Oriented Model) is a copy or part of the whole data (Object-Oriented Model of Main Score) since:

— DLIOOs work on a single part and can visualize only the measures connected with that part; and
— MASEs work on the main score visualizing it.

The adopted architecture has both some problems and advantages. The main benefit is that clients can use different visualization mechanisms and rules on different terminals via a local MILLA interpreter (see Figure 6). However, in a distributed architecture some problems of inconsistency may occur as discussed below. In MOODS, each change (e.g., add or delete) performed on the lecterns (MASAE, MASEs, and DLIOOs) causes a selection that is coded in a command sent to the command manager process of MASAE. Generic commands may be those of insertion, deletion, addition of features, and modification of symbol features.

The command manager provides for the command interpretation and execution. Moreover, the commands are stored in a command list for further undo as described in Section 5. The configuration is maintained in the MASAE, where it is used for direct command redistribution. The configuration in the MASAE (Config) includes the relationships defined during the configuration. A local user, a part, or the main score and a permission table for editing music symbols are assigned to each and every lectern. The Temporary Object-Oriented Model of each lectern may contain a single part (in the case of DLIOOs), as well as a copy of the main score data (in the case of MASEs).

The execution of a command typically has an impact on the information included in the object-oriented model of the Main Score music of MASAE (i.e., a measure deletion, addition of music notation symbols, etc.). The general object-oriented music model of MOODS is inclusive of more than 300 classes of music symbols and their abstractions. The changes performed in the main score model have to be communicated to lecterns which are visualizing the music that has been changed. The list of lecterns involved in the changes is obtained on the basis of the configuration, considering the specific part visualized and copied in the Temporary Object-Oriented Model of each remote lectern. The communication of the performed changes is done by sending specific commands that are received and interpreted by the remote command manager of each remote lectern. In both cases (meaning on MASAE and on DLIOO) the music is visualized thanks to a MILLA interpreter adopting a specific set of rules. Rules can be different for each lectern in the cooperative system.

According to the classification proposed by Borghoff and Schlichter [1998], the architecture chosen is based on a pessimistic centralized control. In fact, in MOODS, all writing actions are serialized and synchronized. Such a solution was improved by replicating data in the local sites (the Temporary Object-Oriented Model). This allows having very fast access in reading and thus in paging music in order to navigate in the music score.

This type of architecture has a sort of bottleneck in the command manager of the MASAE. On the other hand, the access to the object-oriented model is very fast since there is an indexing mechanism, which permits reaching in a very short time the single music symbol where the command has to be executed. Such an architecture allows the prevention of communication problems in spite of 50 lecterns being present.

In addition, the solution proposed is partially fault tolerant. In fact, if a DLIOO lectern is turned off for some reason, its music part is saved into the MASAE station, in all MASE lecterns, and in the other DLIOOs that work on the same part. If a MASE lectern fails and is turned off for some reason, its data are saved into the MASAE station and in the other MASE lecterns. Finally, even if the MASAE station is turned off, the last version can be recovered from one of the other MASE lecterns. In this latter case, the history of the changes performed on MASAE is lost but the last resulting version is available.

### 3.4 Consistency Analysis

In this section, an analysis of consistency for the MOODS cooperative music editor is performed, taking into consideration the above-described architecture.

In MOODS, all operations performed on music are mainly operated considering music notation symbols as atomic (see Figure 7). This means that the insertion/deletion of more than one music notation symbol at the same time is not possible. For example, (i) the building of a chord made of several notes passes through the inserting of single notes in chord modality, (ii) the building of a set of beamed notes/chords passes through the previous inserting of notes/chords and then through the operation of beaming/grouping them. Only the operations of deletion could be performed at the level of a beam, chord,

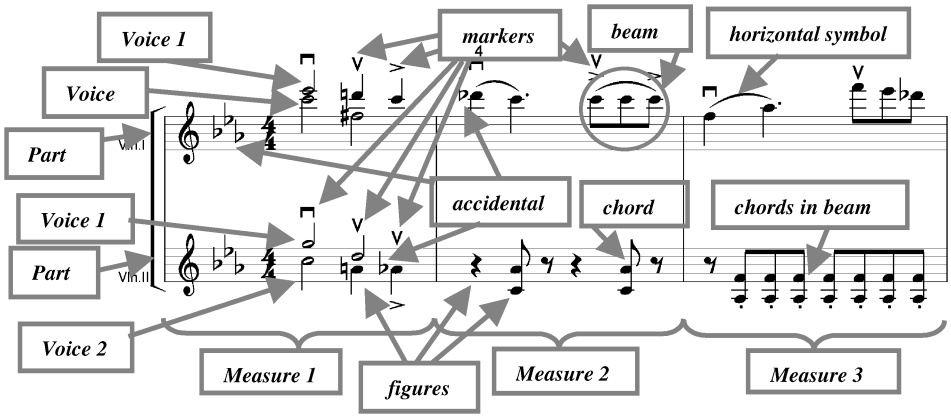


Fig. 7. Main structural aspects of music and notation used.

measure, part, or note with its markers, accidentals, and so on, permitting in any case the deletion of single notes included in chords and/or beams.

In Figure 7, two parts with three measures are depicted. Measures represent the time line ($x$-axis) along which the notes are drawn and have to be played. The first measures of both parts contain two voices/layers each. A voice is a sequence of figures along the part or measure. When two or more voices are present one or more notes are drawn at the same $x$-coordinate along the measure stating that they have to be played together. The second measure of the first part contains two notes and a beam, which in turn is inclusive of three notes. The second measure of the second part contains some rests (which are also figures) and two chords inclusive of two notes each. The third measure of the first part contains three notes and the last two are related by using a slur. This is a horizontal symbol such as the one present in measure two of the first part. In the third measure of part two, there are two beams of chords. Chords are not limited to two notes; they may be based on several notes. In the examples, markers such as fingering, bowing, accent, and accidentals such as flats and naturals are shown.

Music is a nonlinear structure where several nesting levels are present. In MOODS, the model of music notation and the visualization details are kept distinct. The MASAE communicates to clients only by using high-level commands. To this end, all music notation symbols have to be unequivocally identified on the basis of their position in the hierarchical music model. Therefore, elementary symbols are related to the single notes, for instance, accidentals, markers, accents, ornaments, and so on, and need the ID of the related note/rest for their insertion or deletion. The ID is a composite number based on a set of numbers to represent the levels and a final number that identifies the symbol in the level.

$$\textit{Complete ID} = \langle \textit{part number} \rangle \langle \textit{measure number} \rangle \langle \textit{voice number} \rangle \langle \textit{beam number} \rangle \langle \textit{chord number} \rangle \langle \textit{figure number} \rangle$$

In this environment, parts, measures, voices, beams, chords, and horizontal symbols also have IDs. Some of the component numbers of the ID assume a

value equal to 0 when the level is missing. For example, if a note is neither included in a beam nor in a chord these parts have value 0. For instance, in Figure 7, the first note in the second voice of the second part of the first measure is referred to as 2.1.2.0.0.23, stating that the number assigned to the figure is 23. The main operations allowed on music notation symbols are as follows.

— INS(<figure symbol>,<Complete ID>) for inserting the <figure symbol> such as notes, rests, and the like, after the figure is identified by the <Complete ID>. This implies the assignment of an ID to the new figure on the basis of the selection of the first part of the ID and assigning a new unique absolute number for the figure number.
— ADD(<figure feature>, <Complete ID>) for adding a <figure feature> such as accidentals, markers, ornaments, fingering, bowing, and the like, to the figure identified by the <Complete ID>. Features of figures are music notation symbols that make no sense without a reference figure. The ADD operation does not assign a unique Complete ID to the associated features of the figure; they are simple attributes. In some cases, <figure features> may have some detailed attributes, for example, the articulations of some ornament.
— ADDH(<horizontal symbol>, starting figure <Complete ID>, ending figure <Complete ID>) for adding a <horizontal symbol> such as slurs, crescendo, and the like, connected to a couple of figures identified by their <Complete ID>. The ADDH operation assigns a unique <Horizontal ID> to the associated symbol. In some cases, a < horizontal symbol > may have some detailed attributes. The <Horizontal ID> is defined as *<part number><voice number><beam number><chord number><horizontal number>*.
— DELF(<figure feature>,<Complete ID>) for deleting the <figure feature> of the figure identified with the <Complete ID>.
— DEL(<Complete ID>) for deleting a figure symbol having the <Complete ID>. This command leads to deleting all features associated with the figure symbol.
— DELH(<Horizontal ID>) for deleting a horizontal symbol <Horizontal ID>. This command leads to deleting all features associated with the symbol.

The Complete ID is also used in the following examples in its partial form. In most cases, only the figure number mentioned as figure ID is used in place of the Complete ID to facilitate the explanation without lack of generality.

Classical inconsistency problems are those related to lexical, syntactic, and semantic inconsistencies. In large-grain-based systems, large sections of the text can be locked by the user in order to reach a consistent version. On the other hand, each section moves from inconsistent versions before reaching the final and possibly consistent version. In fine-grain text-based cooperative editors (for instance, at the level of a single character), these problems cannot be solved since any change can lead to lexical, syntactic, and semantic inconsistencies even in noncooperative editors.

These kinds of problems are also present in visual/graphic editors if the visual composition under editing provides lexical and syntax rules and semantic meanings behind the composition of graphical elements.

In the case of music editors, lexical, syntactic, and semantic aspects of the manipulated model have to be considered.

— Lexical aspects have to deal with the use of elementary graphic elements for building basic music notation symbols, for example, hooks, lines, and notehead fonts to write notes, direction of the stems, positioning of the marker around the notehead, position of the augmentation dots, and so on.
— Syntactic aspects of the visual language of music have to deal with the sequence of music notation symbols, for example, the right beaming of notes, the correct association of markers with notes, the organization of notes for building chords (placing notes at the left or the right of the stem), the time consistency of a measure, the consistency of voices in the same measure, and so on. Please note that the music notation model may be syntactically correct but incomplete. For example, a nontime-consistent measure can be syntactically correct. In fact, during editing it is absolutely mandatory to pass from a time-inconsistent measure when a measure is produced from scratch. For the same reason, the time duration can be even temporarily abundant.
— Semantic aspects are related to the meaning of the written music, which is the execution rules related to music notation symbols and, on the other hand, the message that the music should communicate to the public. Therefore, the semantic consistency can neither be controlled nor automatically guaranteed such as in the editing of texts.

Such a classification is widely acknowledged for cooperative editing applications [Dourish 1998; Berlage and Genau 1993; Edwards 1997] even if the meaning of the terms can slightly differ from one author to another. For example, Edwards [1997] took into consideration only the semantics of commands and not the semantics of the data out of which the conflict is generated, as explained above, whereas the meaning reported above was used in Berlage and Genau [1993] and Sun et al. [1998]. In this article the above-mentioned meanings are adopted.

In MOODS, the above consistency problems are structurally partially solved. Lexical inconsistencies cannot be produced in MOODS since: (i) music is produced only by using music notation symbols and not by starting from graphical elements; (ii) the directions of stems and beams is automatically decided on the basis of MILLA rules; (iii) the positioning of note markers (expressions, accents, ornaments, etc.) is automatically computed on the basis of MILLA rules (that can be manually forced if needed); and (iv) the ordering among markers associated with notes is automatically decided on the basis of MILLA rules, and so on.

Most of the syntactic inconsistencies cannot be produced since MILLA rules also manage the automatic beaming (grouping decision and beam orientation estimation), as well as the generation of chords (left–right decision for placing noteheads with respect to the stem), and so on. The syntactic consistency of music also includes verification of the time consistency. This cannot be guaranteed without adding strong limitations to the music editing. In fact, during music writing, it is not always possible to have at each step time-consistent measures, such as in writing a text it is not always possible to have lexically and

syntactically correct phrases during their building. For these reasons the verification of time consistency is typically performed offline after the composition.

According to Edwards [1997], syntactic conflicts may be classified as spatial, structural, and temporal. Spatial problems are present when wrong spatial relationships are produced for the execution of commands coming from different users. This kind of conflict is not possible in MOODS since the MILLA formatting engine produces only spatially correct formatted music. Structural conflicts are produced when wrong structural relationships are generated by the execution of commands coming from different users. Temporal conflicts are produced for the arrival of incorrectly ordered commands such as when a command to change a symbol arrives after the symbol deletion. These last two types of conflicts can lead to the inconsistencies proposed in Sun et al. [1998]. In Timewarp [Edwards 1997], a solution to detect and solve these conflicts is given. The solution is based on the temporal ordering of commands, on the detection of conflicts, and thus on the reordering of commands when possible on the basis of visualization of histories. That approach was mainly defined for asynchronous loosely coupled applications.

The approach followed in MOODS aimed at defining a specific communicating protocol among the MASAE and the clients in order to prevent the production of any possible inconsistency in the music notation model.

According to Sun et al. [1998], there are three main types of inconsistencies: convergence, causality preservation, and intention preservation. These three problems are for the most part not completely solved by cooperative editing systems [Sun et al. 1998]. In the following a discussion of these properties in the context of the MOODS system is reported.

*Definition* 1. *Causality Preservation*. For any pair of operations, they must executed in all sites according to the same order in which they were generated on the original site. The problem of causality violation may occur when operations are executed out of their correct order due to latency of communication. Causality preservation ensures causality order for dependent operations performed on the same site.

The following two examples present operations that cannot be performed in a different order, the operations being insertion of a note and association of an accent to the note; and addition of an accidental to a note and its deletion. According to Ellis and Gibbs [1989] and Sun et al. [1998] causality preservation can be obtained by assigning a timestamp to each command delivered on the cooperative system and executing the commands in temporal order. This problem has to be tackled for commands received by the central stations of the cooperative systems and by the remote sites. Operatively, this means that, in order to have absolute certainty that no other earlier commands arrive after current command execution, it is necessary to wait at least a time equal to the maximum latency delay of the network support used. A total ordering allows the implementation of linear undo/do/redo mechanisms (see Section 5), and the acceptance of nonserialized commands may generate branched histories of commands that generate deferred conflict management. This is not acceptable in synchronous real-time collaborative systems.

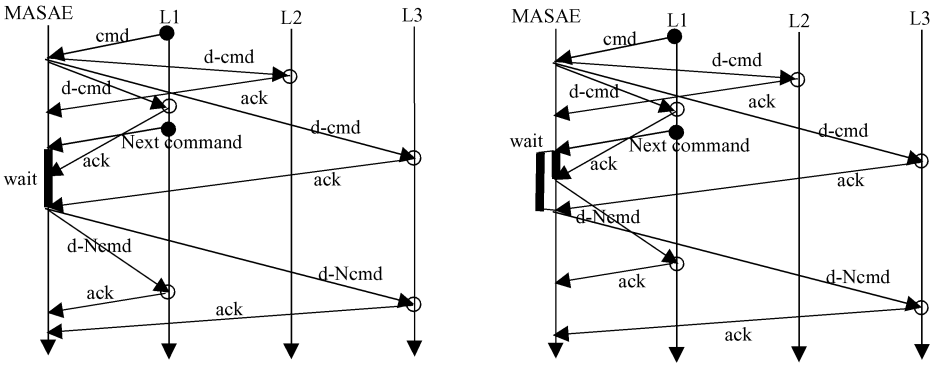


Fig. 8. Two different solutions for communicating changes to the remote sites; (right) the improved version.

According to Figure 8, in MOODS a command (**cmd**) to perform a change in the object-oriented model of the main score is sent to the MASAE from client L1. Only after the application of the requested change on the MASAE a set of corresponding commands (**d-cmd**) is distributed to the interested lecterns (L1, L2, and L3) including that from which the command has been generated (L1). This centralization of the data model totally avoids the problems of noncausality at the expense of performance. After the distribution of commands to the lecterns for replicating the change performed on MASAE, the MASAE has to collect all the acknowledgments, **acks,** confirming the reception of commands sent. If the MASAE accepts the next command before receiving all the **acks,** a next distribution of commands may lead to executing the commands in the wrong order on some lecterns, thus violating the causality preservation property. Please note that the above-mentioned communication protocol is based on the low-level protocol implemented in MOODS with the PVM (Parallel Virtual Machine) [Geist et al. 1994].

To avoid this problem, the MASAE has to wait for the reception of all the **acks** to be ready for receiving the next command (see Figure 8, left). This approach provides the introduction of a delay, such delay depending on the maximum delay of the network and of the timeouts defined in the communication protocol. Please note that the MASAE sends the command according to the configuration of the orchestra to all MASEs and to the DLIOOs working on a specific part. This first solution constrained all clients to wait for the slowest client of the group.

To improve the mean reactivity of clients the solution has been to maintain a queue of commands for each lectern and to match the commands sent with the **ack** received. In this case, the command received is redistributed to each lectern as soon as the last corresponding **ack** is received (see right part of Figure 8). For example, in this case, L1 waits for a shorter time. This solution allowed decreasing the mean delay. The presence of a subsequent delay is not a real problem for such applications, as described hereafter. If an **ack** of a lectern is not received by the MASAE the connection is broken and the lectern has to open the connection again. In that case, the data to fill its Temporary Object-Oriented Model of Music have to be reloaded from the MASAE.

In editing music, as well as for any other visual language, waiting for the command processing before seeing the effect on the screen is not a problem since the delay is negligible with respect to the time needed to move the mouse. Typically, only symbolic commands are sent and the time needed is comparable to that of visualization and mouse moving. Text editors are much more critical since fast editing is required (even 350 chars per minute are needed). When it comes to music editing, for example, during rehearsal, the demand for a fast response is low, less than 40 mouse commands per minute. All commands are produced with the mouse (one or two clicks) and the mouse movements are needed in order to select the menu symbol and then place it in the music score on the screen; at that point the command is sent.

*Definition* 2. *Convergence.* When the same operations have been executed on all sites, then all copies used of the shared data are identical: the result is the same on all sites. A divergence can be due to the arrival/execution of operations in different order because of the latency of communication time. The convergence property guarantees the consistency at the end of the editing section but not in each instant.

In general, it is possible to have convergence and noncausality preservation; when the final result is coherent, the intermediate are inconsistent. The problem of nonconvergence can be easily avoided by serializing all commands/actions to obtain a global convergence. This is possible by extending the timestamp technique to all the sites. In that case, it is possible to have a global time ordering of all commands produced by all sites. Technically, this is only possible if all sites of the distributed system are synchronized or by asking the central station of the cooperative system for a timestamp or a code number for each command. For instance, if two users insert a symbol each at the same time, the expected results may differ from the obtained results; the first user would like to insert symbol B in the middle of sequence BAAB, and the second user would like to insert a C in the same place. The result can be BACBAB or BABCAB. If the global time ordering is imposed, the solution is that obtained by executing the commands in the same time order, thus the result is repeatable and independent of the latency of the communication.

In general, in systems for cooperative editing of music, the effects of deleting a note, A, in a lectern are reproduced to the other lecterns with a given delay; this could leave time for some user to try to associate a marker with note A, which has been deleted by another. In any case, the final result is correct: the note A with its markers is not present on the score. The intermediate result could be confusing. At the same time, some lecterns could see the note while others could have deleted the note and some could present the note with the marker for a few time instants. After that, the system converges to the final solution in which the note (with its features/marker) is deleted on all sites.

In MOODS, the problem of nonconvergence does not exist since the execution of a command A to a given figure (e.g., the addition of a feature/marker) can be performed only after the complete execution of command A. In the above example, this implies that the addition of a feature to A can be performed only when A is deleted. This means that no addition is performed. This avoids the

generation of confusion since the intermediate configurations described above are not produced and shown to the users.

*Definition* 3. *Intention preservation.* For any operation the effect of its execution on all sites is the same and does not depend on the context. The problem of nonpreservation of intention is due to the concurrent generation of operations and execution latency; the single actions could be performed on different conditions with respect to those that were present when generated.

Typically, intention preservation is the most complex property to be achieved. Initial intentions of users may not be satisfied even if the serialization of actions is imposed. For instance, if we have a sequence ABCDE, a first user would insert TS after the first symbol with command INS(TS,1), while a second user would perform the deletion of two elements after the second element, DEL(2,2). With this indexing mechanism the result depends on the ordering in which the actions are performed: ATCDE or ATSBE. In certain cases, results can be different on the corresponding two sites, and violate the users' intentions even if time ordering is imposed, since ATCDE does not satisfy the intention of both users.

In MOODS, the problem of *intention nonpreservation* is avoided by using the above-mentioned indexing mechanism to identify the single music notation elements. As said, each element of a sequence of symbols is marked with a unique Complete ID. Thus the above-mentioned sequence of symbols may be: A(9)B(3)C(4)D(2)E(23). The unique ID is automatically assigned to each symbol by the editor during the insertion so as to identify unequivocally the symbol in the entire representation of the music model.

With this approach the operations of insertion and deletion performed by the mouse become INS(T,9), INS(S,10), considering that ID = 10 has been assigned to T and DEL(4), DEL(2) for deleting symbols with ID equal to 4 and 2, respectively. In the insertion, the number included in the command allows finding the correct position for the symbol: it will be placed after the symbol with that ID. The delete command exactly identifies the symbol that has to be eliminated by using the ID. With this solution, the order in which the operations of insertion and deletion are performed does not influence the final result if the symbols are not related, that is, A(9)T(10)S(25)B(3)E(23), where IDs 10 and 25 have been assigned to the newly inserted symbols T and S, respectively.

The adopted approach still presents some minor problems. For example, when a command DEL(4) to delete symbol with ID = 4 is performed on a client and sent to the MASAE, and before the command distribution (from MASAE to the lecterns) the symbol with ID = 4 is used as a reference in an insertion command INS(R,4) (the selection has been possible due to the network latency). In this case, the second command is delayed by the MASAE since a command related to the same ID is not completed yet: the corresponding **ack**s are not yet received. Another example: when a user deletes a note, DEL(6), and a second user, before receiving the deletion command, adds a marker to the same note, ADD(i,6), then the command simply fails as stated above in the section discussing Definition 2, Convergence. If the commands are executed in the opposite order, the final result is the same: the note with its marker is deleted.

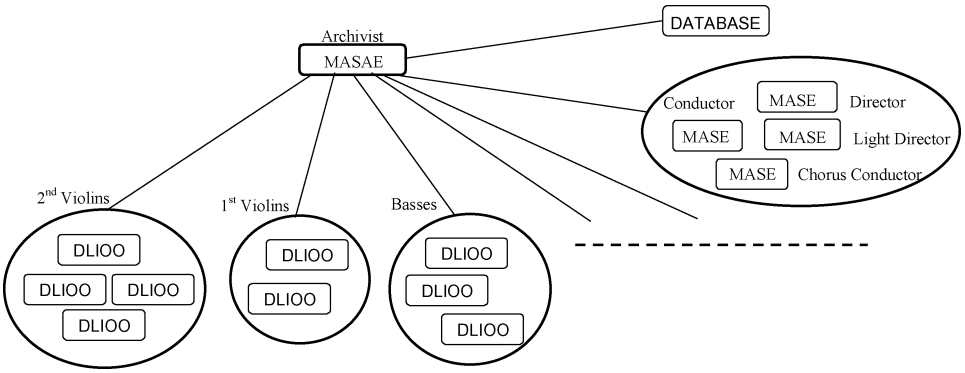


Fig. 9. Hierarchical relationships among lecterns in MOODS.

In music notation there are also horizontal symbols (such as slurs, ties, crescendo, decrescendo, etc.). They are related to two figures, the starting and the ending figure (see Figure 7). In some cases, they may start or end in points in the middle of figures: ADDH(SL,4,99). Defining special symbols called anchorages, which are simply figures without visualization and time duration, can solve this modeling problem.

## 4. ORCHESTRA CONFIGURATION

In several cases, the cooperative manipulation of information is performed by using a nonuniform group of people. They may have different preferences, experience, roles, and training [Teege 1996, 2000]. In cooperative editing of music, these problems are particularly relevant. In the management of music aimed at providing support for its cooperative manipulation, there are several problems of configuration, authorization access, and versioning. Problems dealing with authorization access and versioning are influenced by the orchestra configuration and organization in terms of director(s) (in some cases more than one director can be present: chorus, conductor, director, etc.), groups of musicians, and the archivist. In addition, cooperative music editing may lead to degenerative conditions if all musicians are enabled to deeply modify the music (e.g., removing/adding notes, rests, measures). To avoid such problems, specific flexible authorization mechanisms have to be provided. These aspects are also strictly related to the management of music score versions. Changes performed by musicians cannot be considered as belonging to the basic version provided by the editor. In the following sections, these aspects are discussed separately.

Before its adoption, the MOODS system has to be configured in order to assign parts to lecterns and to identify the physical characteristics of each lectern (screen and windows dimension, pointer type). The orchestra is a two-level organization, meaning that the MASAE station directly connects all lecterns while they are grouped: several groups of musicians and the group of directors (see Figure 9). In MOODS, each part is associated with a group. If two groups have to provide the same part for a different purpose—for example, two groups of violins located in different parts of the theatre—they have to refer to two copies

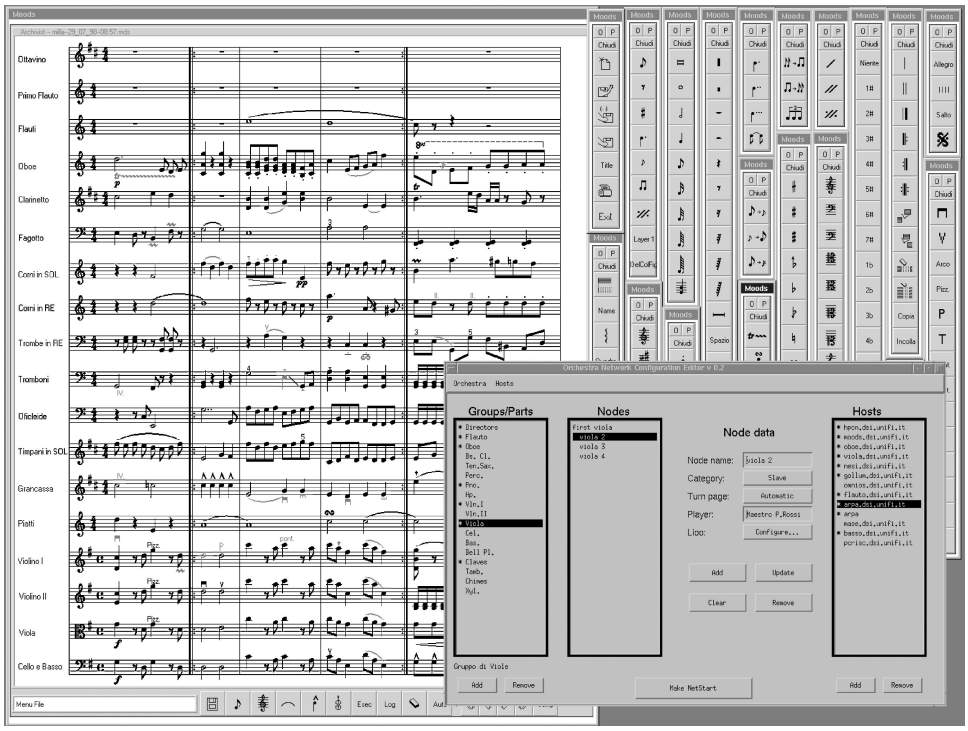


Fig. 10. MASAE workstation with active MOODS editor and ONCM (music reported is taken during the editing section; it may have no meaning or may be time inconsistent).

of the same part. Obviously, each electronic lectern may belong only to a Group. In orchestras, typically, each music lectern is read by a couple of musicians. This has also been replicated in MOODS without problems of readability by using lecterns based on a TFT LCD screen. The arrangement of musicians to lecterns depends on the instrument and on the orchestra organization.

The main entities in the configuration of a MOODS system are: the Parts/Groups (according to the loaded main score); (ii) the available physical lecterns (DLIOOs, MASEs); and (iii) the Groups of musicians and Directors. Typically, parts can be from 2 to 30, whereas DLIOOs can be from 2 to 50.

In MOODS, the configuration aspects can be stated by using the above-mentioned Orchestra Network Configurator Manager (ONCM). It helps the user (installation team with the archivist) to configure the orchestra network and the single lecterns via a graphical user interface. Figure 10 depicts the MASAE screen of the archivist workstation. It has both the MOODS music editor and the ONCM tool for orchestra configuration (in the bottom left corner). In the ONCM window, the list of possible Groups/Parts, the list of Nodes (the logical names of the lecterns, the name of the XWindow process that will be executed on the computer-based lectern), and the names/addresses (in symbols) of the lectern machines (MASEs and DLIOOs) are available. A list of Nodes can be associated with each part, and one of them plays the role of leader. A physical lectern has to be assigned to each Node. For the sake of flexibility and to meet

the needs of specific musicians, more than one logical lectern (and thus more than one part) can be associated with a single physical lectern. This allows musicians to see more than one part on their lectern, simultaneously or selecting each of them with a simple mouse click (during both EDITING or EXECUTION modalities). In general, several distinct processes (DLIOOs or MASEs) may run on the same computer (physical lectern). This is extremely useful for musicians who play more than one instrument in the same piece.

Moreover, load/save/editing of orchestra definitions are possible. To create a totally new orchestra configuration the ONCM user has to:

— add in the groups list box the names of parts that will be executed by the orchestra (these can be recovered from the loaded main score on MOODS according to the standard codification of part names); and
— select from the list of all the hosts (DLIOOs and MASEs) owned by the theatre the lecterns that have been installed for the orchestra; define the Nodes/Lecterns present in each group and eventually assign them a physical electronic lectern.

Each group of musicians has lecterns that work on the same part in a cooperative manner and do not have access to the other parts of the main score. A musician on a DLIOO needs to have the possibility of adding and manipulating a set of symbols depending on his or her instrument and role in the orchestra. For example, specific symbols for the strings (such as bowing up and down, Tallone, Ponte, etc.) are totally useless for the piano. The ONCM is used to establish the relationships between lecterns and parts and also to define a set of permissions for manipulating music symbols as discussed in the following section. Furthermore, a different set of MILLA rules can be associated with each distinct client, even when two different nodes/parts are located on the same machine.

Typically, a specific profile of permissions is imposed on the account of the instrument used and task in the cooperative work. Such a profile defines the symbols and the actions that can be performed on the music. The archivist may decide these details. Typically, the MASE lecterns have full access to music manipulation.

In each group, there is a leader who may decide specific instrumental/execution details (e.g., bow up, bow down, etc.) for the others. Group leaders are typically configured (at the permissions level) so as to perform changes on the current score (main and interpretation symbols), whereas others typically need less editing power. For instance, they could be limited to add fingering, attention markers, bowing, textual annotations, and the like. This organization allows the first instrument (e.g., the leader of the second violins) to perform deeper changes in the score with respect to other musicians of the same group. Sometimes, there can be two or more leaders per group: in this case, the other leaders can be enabled to perform fully cooperative work with the first leader. The changes performed by all leaders are reflected to all other lecterns of the group including those of any other leaders. In MOODS, several lecterns of the same group can have the same power of the leader and thus the distinction is

only formal and ruled by the profiles imposed in the co n figuration. At http://www.dsi.unifi.it/~moods/moods/acm/authorisation.html, details about the permission profiles are reported and discussed.

## 5. VERSIONING VIA ADDITIONAL COMMAND LISTS

The tracking of changes and thus the management of versions are mandatory features for cooperative tools, [Borghoff and Schlichter 1998; Lee et al. 1998]. These problems are strictly related to that of undo, which in cooperative editors can be particularly complex owing to the fact that consistent versions can be identified thanks to changes performed by several different users. Different versions may be managed by using different levels of granularity. At fine grain, each change in the data may produce a different version. At a larger grain, the versions are produced considering sessions of work that include several detailed changes. In MOODS, this second meaning of version was chosen to generate distinct manageable versions that can be generated and reproduced by using the MASAE station. Single musicians are not allowed to work with the version management.

For the management of music performed by orchestras, correctness regarding copyright problems is very important. Music used during performances is typically protected by copyright. Any change performed on the music score with the aim of producing a new printed version of that score should be authorized. Changes performed by pencil and pen on sheet music are not as important as those performed on a symbolic version of the music, since the latter may be used for producing a new version of the sheet music. The production of each new sheet music version has to be controlled by the publisher/owner of the rights. In MOODS it is possible to maintain the version provided by the publisher (referred to as the Neutral Version) distinct with respect to the detailed changes and versions produced by the orchestra and archivist during performance preparation (called in the following the additional command lists, ACLs). Each ACL contains the list of commands that once applied, allow passing from the Neutral Version to its corresponding version. The ACL contains both global and local changes performed in the orchestra. For instance, a local change can be the deletion of a symbol in a part and a global change can be the deletion of a measure (column) in the main score. According to previously presented mechanisms for managing conflicts, commands are serialized and thus, inside each ACL, the changes are linearly organized (i.e., commands coming from different users are interleaved). In this view, the single client is not allowed to see and manipulate the local history of changes. Local and global changes are collected in the MASAE station and managed only by the corresponding operator. The loading of ACLs permits keeping track of the different versions. For example, in Figure 11, Versions 1 and 2 (including Neutral Version plus ACL1 and ACL2, resp.) have been highlighted and Versions 1.1, 1.2, and 2.1 can be easily obtained. Version 1.1 is obtained by loading the Neutral Version and applying ACL1 and ACL1.1.

In MOODS, an ACL may be manipulated to generate different ACLs. The manipulation may be performed by undoing selected commands, applying a specific

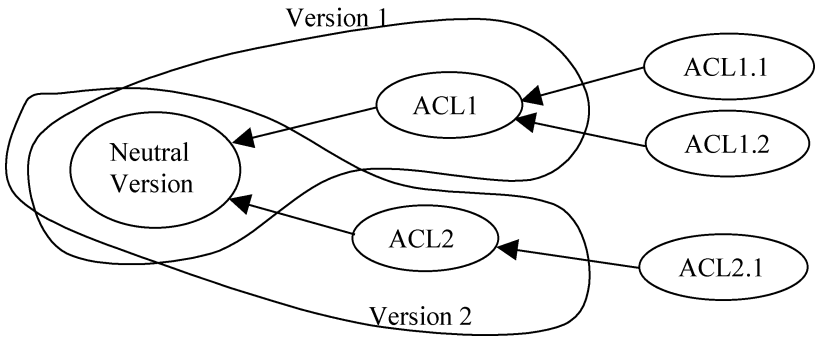


Fig. 11. Relationships among the Neutral Version, the ACLs, and the Versions.

filter to select families of commands, and/or adding other commands. The filtering procedure allows accepting changes from a version/ACL: performed by a client (MASEs or DLIOOs), related to a given family of symbols (only the slurs, only the bowing) decided by a conductor, performed by the archivist, or performed on each specific measure, part, and so on. The filtering process detects and automatically eliminates all eventual commands that create conflict, such as the change command on a deleted symbol. For example, starting from the condition reported in Figure 11, the manipulation of the ACL in Version 1 may produce during the saving a new ACL3 and thus Version 3. The production of a derived ACL (such as ACL1.1) is only possible when the previous ACL1 is totally accepted/validated. In MOODS, each client may manipulate only the allowed music notation symbols according to the assigned permission table. In addition, the commands related to some symbols may undergo validation from the MASAE operator that is responsible for music in the orchestra.

In order to manipulate the single ACL or to fuse hierarchically related ACLs, a specific tool has been set up. It allows the visualization of the loaded history of changes, filtering changes and the selection of only some of them, saving and loading separate lists of changes, keeping separate the original loaded version (Neutral Version) with respect to the changes performed (ACLs), and so on. The tracing of changes allows recovery of the operations performed by maintaining references to the authors.

In Figure 12, the Additional Command window for managing the versioning on MASAE is shown. By means of this window, the MASAE operator can validate, invalidate, or undo the commands issued from the lecterns (both MASEs and DLIOOs), and perform other more sophisticated operations. Such operations can be performed by means of the buttons in the dialog. In particular, in the list box of the dialog, all commands appear in the exact order in which they were performed in the system. For each command a description is reported. In the following paragraph, the fields of the single description command are commented on. The code on the left reports the status of the command: X means that the command is valid, O that it is invalid (validated or not validated by the MASAE operator); L means that the command has been sent by a DLIOO and D from a MASE; I means that the command is an insertion, C that it is a cancel/delete, and A is used for additional commands such as attach the

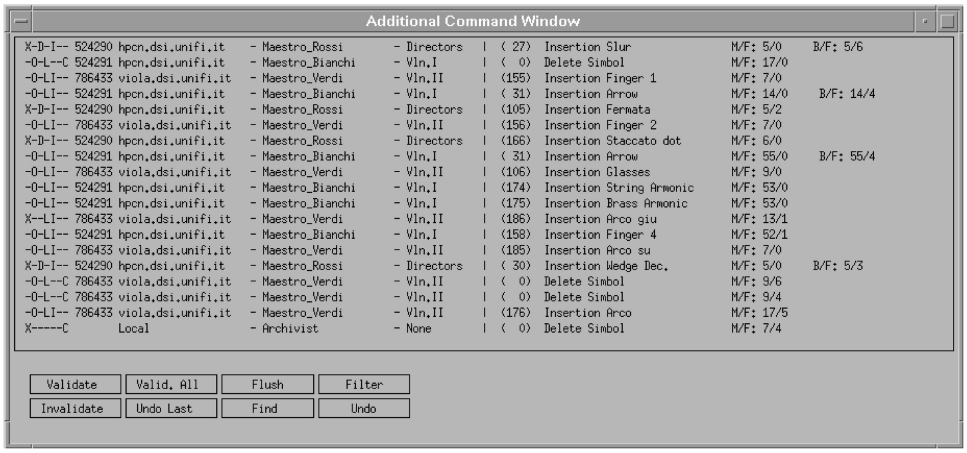


Fig. 12. Additional Command Window of MASAE with some command/changes to be validated.

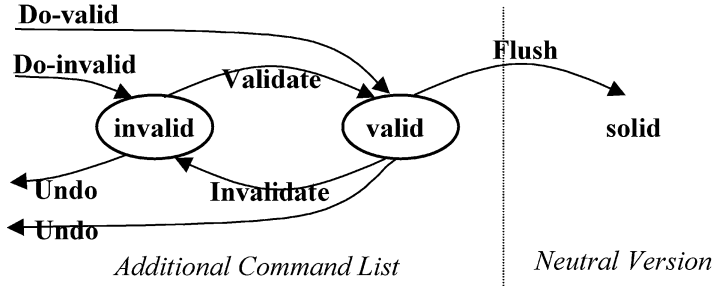


Fig. 13. Evolution of the state of each single command and change.

lectern and load its Temporary Object-Oriented Model, close the lectern, undo last command, and so on. In the command line, there is also (from left to right): the process ID of the PVM, symbolic IP address of the machine hosting the process, the musician name, the assigned part, command ID, command description, and starting and ending measure and figure. The last field is present only for horizontal commands such as slurs to identify the starting and ending figures.

According to the profile imposed on each lectern/client, the changes performed are initially considered valid or invalid (transitions Do-valid and Do-invalid in Figure 13 are associated with the command leading the command in the corresponding state). For example, a client can be entitled to perform directly valid changes in the fingering, whereas the changes related to slurs are too relevant and have to be validated by the MASAE operator.

Invalid changes are shown in a different color on the music score window of the MASAE station and must be validated by the archivist. The selection of a command in the ACL dialog list and the adoption of the Find button produces a jump in the main score window of the MASAE. This fact, joined to the knowledge of the part name and the presence of a different color, allows the operator to identify immediately the effect of the command in the music score. Once the invalid commands are selected and validated with a specific

command, the color of the related music symbol becomes black. The MASAE operator may invalidate commands that are currently valid. Only valid commands can be flushed to move them in the solid part of the music score, in the Neutral Version, to produce a consolidated version. This means that only flushed validated commands can be saved with the Neutral Version, whereas valid and invalid commands with their status can only be separately saved into the ACL file. Each ACL can only be loaded and thus reapplied to the Neutral Version from which it was created. In each ACL the referred Neutral Version is specified in order to avoid the production of inconsistencies.

In the lower part of the window, a set of buttons is present. Their functionalities are explained as follows.

—**Validate** validates a command passing it in the state of Valid.
—**Validate All** validates all the invalid commands present in the ACL, passing them in the state of Valid.
—**Invalidate** invalidates the selected commands. This is a selective invalidation mechanism that works according to the example discussed in the following.
—**Undo Last** undoes the last command. The undo of the last command (ADD, DEL, ADDH, etc.) is totally safe and inconsistencies cannot be created. The undo of the last command is the only undo operation lecterns/clients are allowed to carry out (see the following for details).
—**Flush** eliminates all valid commands from the ACL in order to make them correspond to the current Neutral Version. Once a command is flushed is not possible to separate it from the Neutral Version and it disappears from the ACL. This command is disabled when the Neutral Version is protected for copyright protection. When an unprotected Neutral Version is modified by a Flush command, it has to be saved otherwise the corresponding ACL is aborted because it risks being inconsistent.
—**Filter** allows visualizing all the commands in the ACL that are selected by the filtering statement. For example, it is possible to select all the commands issued by a specific lectern or group or part, all the changes performed and related to a measure, all the changes performed on a specific class of music notation symbol (e.g., slurs, bowing for strings, fingering, etc.), all the changes performed in a specific measure of a specific part or group, all the validated commands, and so on, and any combination of these constraints.
—**Find** highlights the symbol mentioned in the selected command with a different color in the windows containing the music score.
—**Undo** performs a selective global undo of a command contained in the ACL (see the following for details).

When nonlinear data structures, such as music, are cooperatively manipulated, the selective undo has to be allowed since strictly sequential undo can be too restrictive even if applied to the work of each user [Prakash and Knister 1994]. In cooperative systems, the enabling of local undo leads to the establishment of mechanisms of selective undo since, generally, the last command of a client may not be the last command of the system [Berlage 1994]. In Berlage

[1994], Berlage and Genau [1993], and Myers and Kosbi [1996], discussions about selective undo/redo mechanisms are reported. The discussion is mainly devoted to solving the problems related to the production of branches in the history of commands, nonlinear histories of commands that produce different versions. They are mainly generated when different users produce alternative commands in asynchronous systems or at the same time in synchronous systems. This approach fits for version management, whereas in MOODS the ACL has a linear structure, because commands are serialized. The adoption of a tree structure is not acceptable when changes have to be applied in real-time and are even performed by 50 people at the same time. In such conditions, conflicts have to be automatically managed in real-time, canceling one of the conflicting commands according to policies previously discussed. The user who saw his command canceled has time enough to replicate his command in the new context.

Both selective undo and invalidate commands (Undo and Invalidate buttons) work in a similar way so as to avoid the generation of inconsistencies. The only difference consists of the command being eliminated from the ACL and its effect removed from the music score in the case of Undo, whereas the invalidation simply implies a change of status in the ACL and a change of color in the music score shown on the MASAE (see Figure 13). This means that invalid commands are in a temporary condition to attract the attention of the MASAE operator.

We applied and customized some of the ideas already expressed in the literature about selective undo, according to the requirements of music cooperative editing. Generic graphic editors, the main goal of which being to put on the screen some graphic element, have to cope with conflicts about the possible inconsistencies that may arise when selective undo removes or changes elements deemed necessary for successive commands. In MOODS, as in other tools where graphic elements are related to each other, there is an additional problem dealing with the command deletion or its redo in reverted format to perform the undo. Commands are dependent on each other according to a nonlinear structure. The ACL has a linear structure whereas its commands are internally related according to nonlinear relationships. This is due to the nonlinear structure of music and to the relationships among the Complete IDs used in the commands. This kind of conflict may be a hindrance to the execution of the selective undo.

A typical piece of an ACL is shown in Figure 14. In the example, commands have been reported by using the Composite ID with <part number>.<measure number>.<voice number>.<figure number> and the Horizontal ID with <part number>.<voice number>.<horizontal number>. At line 4) main symbol X has been inserted in part 1, measure 23, voice first, after the figure with number equal to 65. Number 74 has been assigned to the new figure. In the example, the commands that are related to this insertion have been marked and arrows have been drawn to put in evidence the relationships that successive commands have established with the command at line 4). These dependencies have to be taken into account in successive actions of invalidate/validate, undo, save, and load of the ACL:

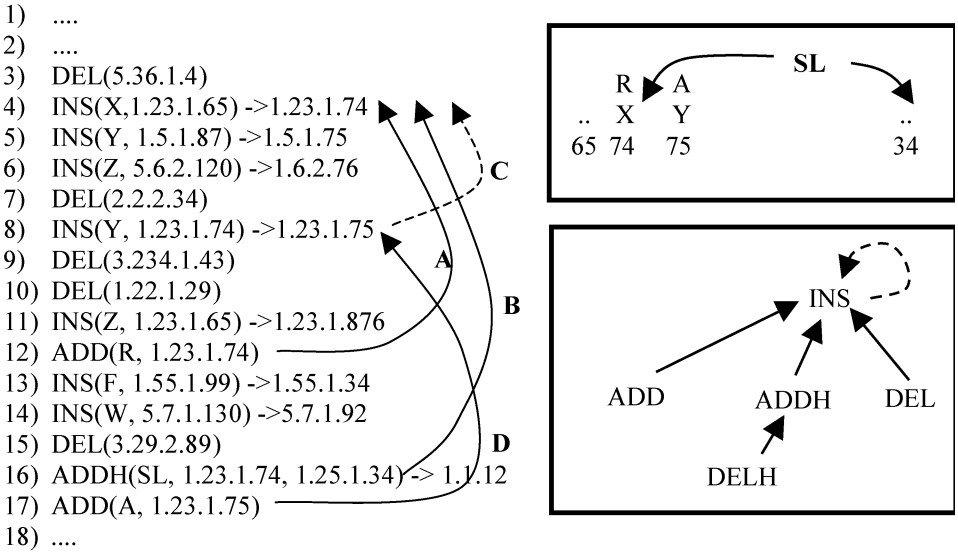


Fig. 14. An example of ACL: (upper right) graphic representation of related commands; (bottom) the general dependencies established among commands.

— **direct for insertion of** <**figure features**> (markers, ornaments, etc.). For example, command ADD at line 12) performs the addition of feature R at the figure X with Complete ID equal to 1.23.1.74, added with line 4);
— **direct for insertion of horizontal symbols**, such as that defined with command ADDH at line 16). This performs the addition of horizontal symbol SL (slur) between figures identified by 1.23.1.74 and 1.25.1.34;
— **direct for reference** (dashed line) during an insertion such as with command INS at line 8) in which another figure has been inserted after the reference figure 1.21.1.74 inserted at line 4);
— **indirect for insertion**, such as that realized with command ADD at line 17). This adds feature A at figure 1.23.1.75 inserted in line 8) which is turn references the figure inserted at line 4).

Please note that the above commands can be applicable only if the command at line 4 is present. This means that several dependencies are defined and have to be maintained in the saved ACL to prevent inconsistency when an ACL is loaded. In addition, relationships have to be maintained and navigated every time that selective invalidation/undo actions are selected. For example, if the command at line 4) is **invalidated or undone,** it also implies that depending commands of ADD at lines 12) and 16) have to be invalidated or undone, respectively. The invalidation of command is quite simple since it implies only a status change, whereas the implementation of the undo is performed by reverting the command. Since we have the mechanisms of validation we decided to avoid the study and the implementation and management of a redo history of commands.

In addition, in the case of Undo of command at line 4), the command at line 8) has to be updated in order to have a consistent ACL that can be saved and

loaded on the current Neutral Version. The updating of the command is simply performed by changing the <figure number> in the Complete ID, according to the identification contained in the referred command under undo. The reference figure of that command is passed on to the command that has to be updated, thus obtaining:

8) INS(Y, 1.23.1.65) →1.23.1.75

The invalidation and the undo of a DEL() command are always possible, since command DEL may be related only to other previous commands. This implies that the symbol becomes visible again with its previous ID. The IDs belonging to such figures are unique, which means that they are generated only once, so whenever a figure is deleted that number is no longer reused.

Clients are enabled to undo only the last command. The local undo can be performed along the list of local commands collected in the ACL until the candidate command to undo presents a successive command in the ACL that depends on the first. This condition detects commands based on the symbols added by the user performed by other users.

When a command is **validated** all previous commands related to the validated command have to be automatically validated. This avoids creating inconsistencies in the Neutral Version since the validated commands can be flushed in the Neutral Version, thus getting a current version that can be separately saved in two parts: the Neutral Version plus the Additional Command List. For example, the validation of the command at line 17) also validates commands at lines 8) and 4); if the command of line 12) is validated, then only the command at line 4) is automatically validated. A marker of a note cannot appear without the corresponding note in the music representation.

In Figure 14, on the right, a graphical representation of the dependent commands reported in the ACL example is presented. In the bottom box, the relationships among commands are shown. The insertion of figure symbols (INS command) can be done only referring to other figures and generating a Complete ID. The INS command has a self-referred arrow since other INS commands may refer to the figure added with an INS command but the command can be removed updating the used Complete ID as previously described for the dashed link at line 4). The addition of horizontal symbols (ADDH command) is performed by using two figures, two Complete IDs. The ADD command is used to insert figure features or their details. The delete can be directly applied only on figures and horizontal symbols, and the deletion of figure features is performed by referring to the figure Complete ID.

For these reasons, a specific command is related to other commands which are located before and after along the ACL. This aspect is highlighted (selecting the related commands) every time that a command line is selected with the right mouse button. When a filter is applied, the set of related commands can remain partially included in the shown list but the above-mentioned rules for validating, undoing, and invalidating commands are maintained to guarantee the consistency after any operation.

## 6. EXPERIMENTAL RESULTS AND VALIDATION

The MOODS system was implemented and validated during an ESPRIT project in which several partners were involved. The MOODS prototype has been validated by using several musicians belonging to the School of Music of Fiesole and of some local and Italian (national level) orchestras. The early experimental sessions were organized in a special room at the University of Florence [MOODS 1998]. The first validation phase was concluded with a final public demonstration at the Scala Theater in Milan, Italy. After that performance, the system was tested and assessed by several other musicians at the DSI laboratory with the support of the School of Music of Fiesole.

Before starting the final validation with end-users the system was tested and validated in a local network in order to stress the system's performance and workload. During the final validation period several musicians tested the solutions presented in this article by using a MOODS systems that included five DLIOO lecterns for musicians, a MASAE, and a lectern MASE for the conductor. This configuration was selected to specifically assess the responses of musicians playing in quartets and quintets. With this configuration, up to nine strings used the system at the same time (two first violins, two second violins, two cellos, two violas, and one contrabass). Alternately, some strings were substituted with flutes depending on the music chosen. We decided to have a group of strings because among the orchestral musical instruments, they perform the most intensive work on music scores, adding fingering, bowing, slurs, crescendos, and so on. The five DLIOOs presented different hardware pointers: trackball or touchscreen with pen.

In the validation period, 57 different skilled musicians used the system covering several roles. They were mainly conductors, musicians, small groups, and archivists who are in any case also musicians. Some of them performed more than one rehearsal session. The average age of musicians attending the rehearsals was close to 30. They were all Maestro (Music Master) for their instruments.

The main music scores used during rehearsals and executions were mostly provided by Casa Ricordi:

—*Linda di Chamonix* by Donizetti (symphony) for a group comprising two violins, a viola, a violoncello, and a contrabass;

—*Le 4 Stagioni* of Vivaldi ("La Primavera," first movement) for a group comprising one principal violin, two violins, two violas, two cellos, and a contrabass, or by using a different configuration (two first violins, two second violins, two cellos, two violas, and a bass);

—*La Traviata* of Verdi ("Il Preludio," the prelude of the first act) for a group comprising two violins, two violas, one or two cellos, and a contrabass;

—*La Traviata* of Verdi ("Il Preludio," the prelude of third act) for a group comprising two violins, one or two violas, one or two cellos, and a contrabass;

—"Eine kleine Nachtmusik" (*Serenate in G*) of Mozart (first movement) for a group comprising two violins, a viola, a cello, and a contrabass.

Table I. Mean Values of Votes Obtained During Validation and Evaluation

| Aspect | Average Votes During Rehearsals Value | Average Votes Obtained at Scala Theater Value |
|---|---|---|
| Readability of music on lecterns | 9.8 | — |
| Utility of turning page during execution | 9.2 | — |
| Usability in orchestra activities | 8.3 | 9.2 |
| Utility in orchestra | 9.6 | 7.8 |
| Interest in professional usage of MOODS | 9.7 | — |
| Usability for teaching purposes | 8.15 | 7.7 |
| Utility in music schools | 8.05 | 8.0 |
| Utility at home | 5.3 | 4.9 |
| Interest in using MOODS at home | 5.4 | 4.8 |

During the rehearsal several other configurations were tested for either the presence or the lack of other instruments or quantities of the same instruments. The above pieces of music were used every time restarting from the Neutral Version, without instrumental symbols, these were introduced during the rehearsal sessions generating several distinct ACLs. The above music scores were produced by converting the score from a FINALE version or by manually writing the music. All the work was performed respecting the rights of Casa Ricordi.

Each new musician was instructed on the system with a 15-minute presentation. We requested each musician to fillout a questionnaire in order to assess the system's usability, to get general reaction and suggestions, and to assess his or her knowledge of both the adopted music pieces in the session and the usage of computer systems and music editors. Among the music pieces selected we registered a large variation in knowledge level. This allowed us to verify the MOODS suitability to support a musician when he or she reads the music to be executed for the first time. The most well-known pieces were Mozart (with 9.5 on 10 votes) and "Primavera" (8.5), and the least known was the Donizetti with 2.2. Apart from lack of knowledge of the music piece, during rehearsals no additional problems were registered which could be related to the presence of the MOODS instead of paper sheet music.

Table I reports the results of the questionnaires filled in during sessions. These questionnaires were mainly prepared to get a general impression of the system in terms of usability and utility with respect to what is currently adopted (meaning the use of paper). On this ground, questions were quite simple, so as to just open the discussion on a given argument and try to quantify their general impressions with a vote from 0 to 10. The writing up of questionnaires was interactive in order to clarify to musicians the meaning of utility and usability. What follows are the examples used to establish an agreement with the musicians about the meaning of words and therefore about the votes expressed. We used a Likert scale in order to reduce misunderstanding about the usability terms that are typically performed in questionnaires in which it is requested to give a simple score from 1 to 10. Rensis Likert scales are typically used to measure attitudes and impressions of people as we have done; they can be centered or not centered.

— Usability. The comparison of the system usability (considering both Editing and Execution) with respect to the normal use of paper sheet music.
  — 0: not sustainable since it is too complex, inflexible, and far from real needs;
  — 3: interesting but with some obscure aspects;
  — 5: similar but not satisfactory; several functionalities are impossible or hard to reach;
  — 6: marginally satisfactory; the most useful functionalities are easily applied whereas others are complex;
  — 8: simple to use for the typical operations performed during rehearsals;
  — 10: much better since the functionalities significantly simplify the work and are easily performed.
— Utility. The innovations provided by the MOODS system can improve the quality of your work or save your time.
  — 0: absolutely no;
  — 3: for a limited amount;
  — 5: comparable to the work performed in traditional way; not enough benefits to be a good tool;
  — 6: marginally satisfactory; advantages are slightly greater than problems and complexity;
  — 8: satisfactory; new functionalities make it convenient with respect to paper;
  — 10: much better than working with paper.

The table has two columns of average scores. The first column reports the average votes expressed during rehearsals by musicians that used the system, and the second reports the impression of people attending the public demonstration at the Scala Theater. In this latter group, among attendees there were several experts, technicians, journalists, and commercial people, which is the reason why some questions were removed. In this case, the average age was 38.6 years.

The data obtained from the two sets of experts are quite comparable; this means that both the musicians who tested the MOODS system while trying to play music with it and simple attendees during presentations and demonstrations received the same impression of the system's utility. When it comes to usability, musicians who really experienced the system received a better impression than the audience that only saw the presentation.

The system has been evaluated as strongly useful for orchestras and music schools. The readability of music on lecterns has been considered very good, 9.8. This feature has been tested by using lecterns with different types of LCD devices and different numbers of people in front of them (1 or 2). The score is referred to the best solution, TFT 15", 5 to 6 lines of music per page. Typically, orchestras have two musicians sharing the same lectern; the visibility is also good in that condition. The most accepted pointer interface has been the touchscreen with pen. Attached trackballs (placed on the same plane of the screen or on the side of the lectern) did not get a large acceptance if compared with

the direct interaction with a pen. The adoption of a pen is very similar to the traditional interaction with pencil and paper.

The adoption of the MOODS at home has been considered not useful or too complex. 67% of those who filled in the questionnaire were computer users and among them only 32% were users with a certain experience on other music editors (mainly Finale and Encore). Musicians who were not familiar with computer music editors more frequently expressed higher votes. This is a very interesting result because it shows that the impact of the MOODS is very low and its adoption could be carried out quite easily.

As a general remark on the discussions performed during the questionnaire compilation, automatic page turning has been enthusiastically accepted, since musicians may always read several measures in advance with respect to the current measure under execution. Furthermore, musicians are used to passing from the right bottom corner to the left top corner when the page is finished. After the first instinctive reaction of turning the page before moving the eyes, they were able to work with the MOODS system. This training stage took only a few minutes since most of them were used to having someone else turning the page on their behalf. The solution found in the MOODS for presenting the succession of pages has been preferred by musicians to

1. a solution based on a scrolling page, since this shifts the fixation point of the reader;
2. one-shot substitution of the page, since this shares the same problems of classical paper pages where the end measure of the page must have a significant number of rests to give time to turn and read the next measures.

During the assessment, in editing mode, the profiles assigned to musicians limited their task to simple annotation symbols. This turned out to be satisfactory since musicians are seldom called upon to perform heavy changes on figures. It is the archivist who, owing to the conductor's requests, typically performs such changes in a preparatory phase. In certain contexts or orchestras some more permissive and collaborative behaviors can be established. For example, a musician may stand up to make changes on other lecterns or discuss the changes proposed by the conductor. This depends on the orchestra etiquette, on the conductor disposition, and on several unwritten social rules of the orchestra. In the MOODS, all these different organizations and policies can be managed by defining specific profiles in order to either limit or permit music notation manipulation. As limit cases, all musicians may have full control on their part or, to the extreme opposite; they could only read the music. To grant the system such flexibility was something decided from the collection of user requirements. In fact, whenever considering the several different occasions during which the system may be used, it was very difficult to adjust the collaborative work on music with fixed rules.

Different permission profiles and configurations (see Sections 4 and 4.1) are useful for adapting the tool to different contexts where the cooperative work on music notation is needed:

—music manipulation during music lessons,
—music composition,
—music rearrangement,
—analysis for critical review of music pieces,
—music encoding and formatting for printing sheet music, and
—music annotation in the orchestra during rehearsals for performance preparation and execution, and so on.

## 7. CONCLUSIONS

In this article we presented MOODS, a synchronous real-time cooperative editor for music scores. Its architecture includes mechanisms for troubleshooting conflicts in real-time, managing histories of commands and versioning, and performing selective undo. The system also includes specific solutions in order to control the editing of permission profiles.

The study and implementation of the MOODS cooperative editor for music have highlighted several problems. Most of them are common to other cooperative editors of nonlinear information. Among these the most interesting aspects are the indexing mechanism used, the configuration, the permission management, the separation of the Neutral Version and the Additional Command List, and the selective undo integrated with aspects of validation and invalidation for versioning management. As far as such aspects are concerned, the mechanisms for avoiding conflicts and the management of the ACLs for synchronous real-time cooperative systems such as those in MOODS are quite innovative. The real innovation is the integration of these techniques for the implementation of a system used for the cooperative editing of music notation. In this context, several detailed aspects of the music model have been considered. We hope that the identified solutions will be useful for any other similar application.

The MOODS system has been validated by several musicians who worked with the tool during several rehearsal sessions. The validation highlighted high levels of usability and utility in music schools as well as in orchestras. Low interest in its domestic use was raised. The MOODS may bring forth a significant saving of time and money while preparing final performances in theatres. It also allows the storing of precious information concerning the music customized for a specific conductor with a specific orchestra, and the like. The relevance towards both the orchestra and the publisher has been discussed; the MOODS is also extremely relevant from a cultural point of view and for music schools. Students could examine different interpretations of the same score. Different scores and versions could be viewed in a few seconds on students' lecterns, allowing swift comparison of music scores and interpretations.

This article is the first public document describing the details of the cooperative work on music notation. According to the first end-users' tests, the MOODS was considered a useful tool for orchestras and music schools. MOODS' functionality releases composers and conductors from mundane, time-consuming tasks, allowing more time for experimenting (instantaneously) with new solutions or

effects. It also means an enormously increased level of creative feedback between a conductor and the performing group/orchestra with the common goal of effective interpretation of the music score. The MOODS opens the path to what, in the next few years, could be a real revolution in the world of music publishing and in musicians' approach to music scores.

## 8. FUTURE WORK

The first evolution of the MOODS has been to experiment with the integration of symbolic music with the image of sheets music. This integration makes the MOODS also fit for saving the cultural heritage, thus revitalizing old sheet music/pieces that could be too expensive to convert into symbolic formats. The MOODS model and language have been used as a basis for further research work performed for WEDELMUSIC (Web delivering of music scores, http://www.wedelmusic.org), an IST project of the European Commission. The project partners are DSI of the University of Florence (coordinator), ARTEC, IRCAM (France), RICORDI (Italy), SUVINI ZERBONI (Italy), SVB (The Netherlands), ILSP (Greece), CESVIT (Italy), FHG-IGD (Germany), and SMF Music School (Italy). The WEDELMUSIC project defined an XML version of the MOODS format that is currently used by the MOODS system. Moreover, thanks to WEDELMUSIC, distributing music scores via the Internet while respecting the rights of the music owner becomes possible. Future research activities and evolution of the MOODS system are in the areas of integration of the system with the management of the theatrical special effects that have to be synchronized with the music in real performances, the cooperative work on music notation for virtual orchestras, and cooperative learning of music and distance learning. The activity of the research group continues on music modeling and tools within the MUSICNETWORK project: www.multimediamusicnetwork.org.

## ACKNOWLEDGMENTS

The authors would like to thank all the members of projects LIOO, $O^3MR$ and MOODS partners: Maestro Carlo Tabarelli (Teatro alla Scala), Maestro Gabriele Dotto, Maestro Marco Mazzolini (BMG Ricordi, CASA Ricordi), Maestro Nicola Mitolo (Scuola di Musica di Fiesole), Francesco Cicillini (ELSEL), and Sandro Moro (Shylock). A sincere thanks to the reviewers for their suggestions and comments which contributed to improve the article. A warm thanks to Simonetta Ceglia for his support in polishing the article and for day-by-day activity.

## REFERENCES

BELLINI, P. AND NESI, P. 2001. WEDELMUSIC format: An XML music notation format for emerging applications. In *Proceedings of the International Conference of Web Delivering of Music*, (Florence, Nov. 23–24), IEEE Press, Los Alamitos, Calif., 79–86.

BELLINI, P., DELLA SANTA, R., AND NESI, P. 2001. Automatic formatting of music sheets. In *Proceedings of the International Conference of Web Delivering of Music* (Florence, Nov. 23–24) IEEE Press, Los Alamitos, Calif., 170–177.

BELLINI, P., FIORAVANTI, F., AND NESI, P. 1999. Managing music in orchestras. *IEEE Computer*, 26–34.

BELLINI, P., FIORAVANTI, F., NESI, P., AND SPINU, M. B. 2000. MOODS: A music format for new Applications. Tech. Rep. Department of Systems and Informatics.

BERLAGE, T. 1994. A selective undo mechanism for graphical user interfaces based on command objects. *ACM Trans. Comput. Hum. Interact. 1*, 3, 269–294.

BERLAGE, T. AND GENAU A. 1993. A framework for shared applications with a replicated architecture. In *Proceedings of the Sixth Annual ACM Symposium on User Interface Software and Technology* (Atlanta), 249–257.

BLOSTEIN D. AND HAKEN L. 1991. Justification of printed music. *Commun. ACM 34*, 3, 88–99.

BORGHOFF, U. M. AND SCHLICHTER J. H. 1998. *Computer Supported Cooperative Work*. Springer, Berlin.

BYRD D. A. 1984. Music notation by computer. Department of Computer Science, Indiana University, UMI, Dissertation Service. Available at http://www.umi.com.

DANNENBERG, R. B. 1990. A structure for efficient update, incremental redisplay and undo in graphical editors. *Softw. Pract. Exper. 20*, 2, 109–132.

DANNENBERG, R. B. 1993. A brief survey of music representation issues, techniques, and systems. *Comput. Music J. 17*, 3, 20–30.

DEWAN, P. AND CHOUDHARY, R. 1995. Coupling the user interface of a multiuser program. *ACM Trans. Comput. Hum. Interact. 2*, 1, 1–39.

DOURISH, P. 1998. Using metalevel techniques in a flexible toolkit for CSCW applications. *ACM Trans. Comput. Hum. Interact. 5*, 2, 109–155.

EDWARDS, W. K. 1997. Flexible conflict detection and management in collaborative applications. In *Proceedings of the Tenth Annual ACM Symposium on User Interface Software and Technology* (Banff, Alberta, Canada) (UIST'97), 139–148.

EDWARDS, W. K. AND MYNATT, E. D. 1997. Timewarp: Techniques for autonomous collaboration Conference on Human Factors and Computing Systems. In *Proceedings of the Conference on Human Factors in Computing Systems*, (Atlanta) 218–225.

EDWARDS, W. K., IGARASHI, T., LAMARCA, A., AND MYNATT, E. D. 2000. A temporal model for multilevel undo and redo. In *Proceedings of the Thirteenth Annual ACM Symposium on User Interface Software and Technology* (San Diego) 31–40.

ELLIS, C. A. AND GIBBS, S. J. 1989. Concurrency control in group-ware systems. In *Proceedings of SIGMOD'89 Management of Data* (Seattle), 399–407.

ELLIS, C. A., GIBBS, S. J., AND REIN, G. L. 1991. Groupware—Some issues and experiences. *Commun. ACM 34*, 1, 38–58.

FISH, R. S., KRAUT, R. E., LELAND, M. D. P., AND COHEN, M. 1988. Quilt: A collaborative tool for cooperative writing. In *Proceedings of the Conference on Office Information Systems*, ACM SIGOIS (Palo Alto, CA), 30–37.

GEIST, A., BEGUELIN, A., DONGARRA, J., JIANG, W., MANCHEK, R., AND SUNDERAM, V. 1994. *PVM 3 User's Guide and Reference Manual*, Oak Ridge National Laboratory, Oak Ridge, Tennessee.

GOURLAY, J. S. 1986. A language for music printing. *Commun. ACM 29*, 5, 388–401.

GREENBERG, S. AND MARWOOD, D. 1994. Real time group-ware as a distributed system: Concurrency control and its effect on the interface. In *Proceedings of the ACM Conference on Computer Supported Cooperative Work*, 207–217.

GREIF, I. AND SARIN, S. 1986. Data sharing in group work. In *Proceedings of the Conference on Computer Supported Cooperative Work* (Austin), 175–183.

GRUNE, D. 1986. Concurrent version system, a method for independent cooperation. Report IR-114, Vrije University, Amsterdam.

HEUSSENSTAMM, G. 1987. *The Norton Manual of Music Notation.* Norton, New York.

HYMES, C. M. AND OLSON, G. M. 1992. Unblocking brainstorming through the use of a simple group editor. In *Proceedings of the ACM Conference on Computer Supported Cooperative Work*, 99–106.

KARSENTY, A. AND BEAUDOUIN-LAFON, M. 1993. An algorithm for distributed group-ware applications. In *Proceedings of the Thirteenth International Conference on Distributed Computing Systems*, 195–202.

KNISTER, M. J. AND PRAKASH, A. 1990. DistEdit: A distributed toolkit for supporting multiple group editors. In *Proceedings of the Conference on Computer Supported Cooperative Work*, (Los Angeles), 343–355.

KNISTER, M. J. AND PRAKASH, A. 1993. Issues in the design of a toolkit for supporting multiple group editors. *J. Usenix Associ. 6*, 2, 135–166.

LEE, B. G., CHANG, K. H., NARAYANAN, N. H. 1998. An integrated approach to version control management in computer supported collaborative writing. In *Proceedings of the 36th Annual Conference on Southeast Regional Conference*, Marietta, Ga, April 1–3, 34–43.

LELAND, M. D. P., FISH, R. S., AND KRAUT, R. E. 1988. Collaborative document production using Quilt. In *Proceedings of the Conference on Computer Supported Cooperative Work* (Portland), 206–215.

MOODS 1998. Music object oriented distributed system, HW-SW final validation, with examples. An activity within the TETRApc-TTN, MOODS HPCN ESPRIT Project Deliverable, DE7.1.

MORAN, T., MCCALL, K., VAN MELLE, B., PEDERSEN, E., AND HALASZ, F. 1995. Some design principles for sharing in Tivoli, a whiteboard meeting-support tool. In *Groupware for Real-time Drawings: A Designer's Guide*, S. Greenberg ed. Mc Graw-Hill, New York, 24–36.

MUNSON, J. P. AND DEWAN, P. 1994. A flexible object merging framework. In *Proceedings of the Conference on Computer Supported Cooperative Work* (Chapel Hill, N.C. Oct. 22–26), 231–242.

MYERS, B. A. AND KOSBI, D. S. 1996. Reusable hierarchical command objects. In *Proceedings of the Conference on Human Factors in Computing Systems*, (Vancouver, Canada), 260–267.

NEUWIRTH, C. M., KAUFER, D. S., CHANDHOUK, R., AND MORRIS, J. H. 1990. Issues in the design of computer support for co-authoring and commenting. In *Proceedings of the Conference on Computer Supported Cooperative Work* (Los Angeles), 183–195.

NEUWIRTH, C. M., KAUFER, D. S., CHANDHOUK, R., AND MORRIS, J. H. 1994. Computer support for distributed collaborative writing: Defining parameters of interaction. In *Proceedings of the Conference on Computer Supported Cooperative Work* (Chapel Hill, N.C.) 145–152.

NEWMAN-WOLFE, R. E., WEBB, M. L., AND MONTES, M. 1992. Implicit locking in the Ensemble concurrent object-oriented graphic editor. In *Proceedings of the Conference on Computer Supported Cooperative Work*, 265–272.

OLSON, J., STRORROSTEN, M., AND CARTER, M. 1992. How group-editor changes the character of a design meeting as well as its outcome. In *Proceedings of the ACM Conference on Computer Supported Cooperative Work* (Toronto), 91–98.

PELIMUHANDIRAM, H. K. 1991. MACE: The mother of all concurrent editors. Master's thesis, University of Florida CIS Department.

PRAKASH, A. AND KNISTER, M. J. 1994. A framework for undoing actions in collaborative systems. *ACM Trans. Comput. Hum. Interact. 1*, 4, 295–330.

RADER, G. M. 1996. Creating printed music automatically. *IEEE Computer 61*, 68.

RHYNE, J. R. AND WOLF, C. G. 1992. Tools for supporting the collaborative process: Symposium on user interface software and technology. In *Proceedings of the Fifth Annual ACM Symposium on User Interface Software and Technology*, (Monterey, Calif., Nov. 15–18), 161–170.

ROSS, T. 1987. *Teach Yourself: The Art of Music Engraving.* Hansen, Miami.

ROUSH D. 1988. Music formatting guidelines. The Ohio State University, Computer and Information Science Research Center, Columbus. OSU-CISRC-3/88-TR10.

SARIN, S. AND GREIF, I. 1985. Computer-based real-time conferencing systems, *Computer*, (Oct.) 33–45.

SELFRIDGE-FIELD, E. 1997. *Beyond MIDI—The Handbook of Musical Codes*. MIT Press, London.

STEFIK, M., BOBROW, D., LANNING, S., AND TATAR, D. 1986. WYSIWIS revisited: Early experiences with multi-user interfaces. In *Proceedings of the Conference on Computer Supported Cooperative Work* (Austin), 276–290.

STEFIK, M., FOSTER, G., BOBROW, D., KAHN, K., LANNING, S., AND SUCHMAN, L. 1987. Beyond the chalkboard: Computer support for collaboration and problem solving in meetings. *Communi. ACM 3*, 1, 32–47.

SUN, C., JIA, X., ZHANG, Y., YANG, Y., AND CHEN, D. 1998. Achieving convergence, causality preservation, and intention preservation in real-time cooperative editing systems. *ACM Trans. Comput. Hum. Interact. 5*, 1, 63–108.

TEEGE, G. 1996. Object oriented activity support: A model for integrated CSCW systems. *Comput. Support. Coop. Work 5*, 1, 93–124.

TEEGE, G. 2000. Users as composers: Parts and features as a basis for tailorability in CSCW systems. *Comput. Support. Coop. Work 9*, 101–122.

WOOD, D. 1989. *Hemidemisemiquavers . . . and Other Such Things. A Concise Guide to Music Notation.* Heritage Music, Dayton, Ohio.

Received December 2000; revised February 2002; accepted May 2002