Chapter 1

Introduction

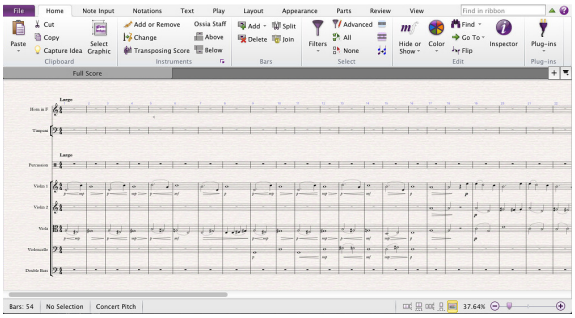
1.1 Motivation

There exist many programs for music notation and composition. Sibelius (fig 1.1 (a)) allows users to write scores using traditional western music notation, whilst music is produced in the live programming interface Sonic Pi (fig 1.1 (b)) by real-time editing of Ruby code [1]. These require users to gain familiarity with a new interface, often with a large threshold to creating simple musical ideas. There are four times more spreadsheet users than programmers [15], it being the preferred programming language for many people [13]. I believe that this ubiquituity, along with the affordances of the spreadsheet, would enable new ways to interact with musical notation that capitalise on existing familiarities with spreadsheets and their data handling capabilities.

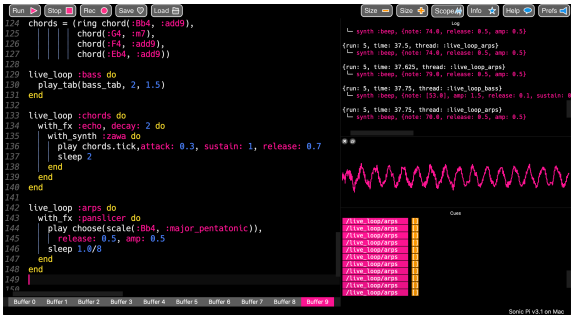
The use of grid structures is an established concept in music programs, with most sequencing software using one axis of the screen for time and the other for pitch or musical parts. Chris Nash's Manhattan [12] (fig 1.1 (c)) uses a grid structure where formulae can be defined in the cells to change the cell value, much like in a spreadsheet. However, it is limited to columns defining tracks and rows corresponding to different times. Advait Sarkar's SheetMusic [14] investigated how formulae with sound output can be included within the spreadsheet paradigm. This also introduced abstracting time away from the grid, in this case using an incrementing global `tick` variable which could be referred to in the formulae. Both axes can be used interchangeably for SheetMusic notation or markup that the user wishes to include which is not interpreted musically, a concept idiomatic to Excel usage. Simple formulae such as `if(tick%2==0) p('snare') else p('kick')` allow musical structures to be defined without advanced programming knowledge but quickly become unwieldy for defining larger pieces, especially if they are not highly repetitive. Whilst other spreadsheet music projects exist [5], these simply use the spreadsheet as the medium for conventional sequencing with an auxiliary script used to parse the grid and create musical output.

Excello (fig 1.1 (d)) is an Excel add-in for end-user music programming where users define music in the spreadsheet and can play it back from within Excel. It maintains the abstraction of time from the grid to keep the flexibility spreadsheets offer but was designed

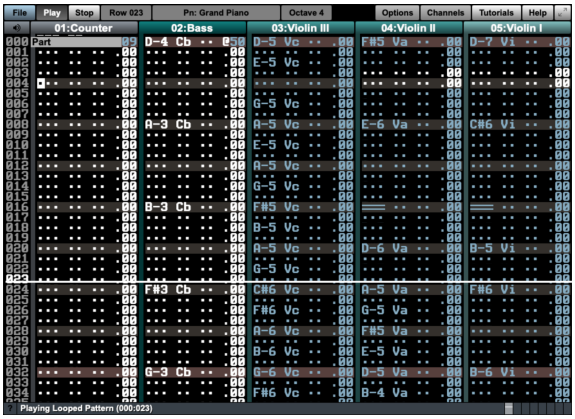
so individual cells would not become too complex. Existing functionality within Excel can be used, both accelerating the learning curve and increasing the available functionality.



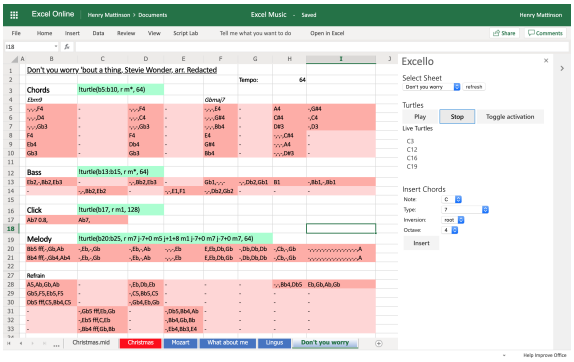
(a) Sibelius - an editor with playback for staff notation © Gorazd Bozic



(b) Sonic Pi - code is written on the left and output information on the right



(c) Manhattan - columns of parts played from top to bottom



(d) Excello - a spreadsheet of music notation with a window for playback on the right

Figure 1.1: The interfaces of a) Sibelius, b) Sonic Pi, c) Manhattan, d) Excello

1.2 Outline of work

1. I designed a system for musical expression and playback within Excel and built a prototype satisfying the all the success criteria for the system.
2. Participatory design began using this initial prototype. Following formative evaluation sessions with 21 participants, issues and feature requests were identified. Users continued to give feedback as they used the updated prototypes.
3. A series of extensions were implemented, solving problems identified by participants and adding requested feature.
4. A converter from MIDI to the Excello format was built to translate a large corpus of MIDI files to the Excello notation.
5. Summative evaluation was performed with the participants. Evaluating the features implemented during participatory design and Excello's usability using the Cognitive Dimensions of Notation (CDN) framework [4], using Sibelius as a comparison.

Chapter 2

Preparation

This chapter shall first address the refinements made to the project proposal. Then I shall explain the tests that were performed to establish Excel's suitability for musical development. Next, the design decisions for Excello shall be explained. The software engineering tools and techniques employed will then be introduced. Finally, the research that was conducted to decide to implement a converter from MIDI to Excello shall be summarised.

2.1 Proposal Refinements

The project designed a notation by which music can be defined within a spreadsheet along with a system for interpreting the notation in the spreadsheet to produce audio output. This continued to explore ways in which time can be abstracted away from the grid.

The aim was to implement an Excel add-in subject to successful initial testing. An add-in is a web application displayed within Excel that can read and write to the spreadsheet using the Office Javascript API [1]. Arbitrary additional data and markup can be included in the spreadsheet. Only information in the spreadsheet is required for playback with the add-in. Tests shall be carried out to verify that suitable audio output can be produced for music end-user music programming within Excel to be possible.

A sizeable addition to the project beyond the initial proposal was to perform participatory design [11] to advise on improvements that can be made beyond the initial prototype. Participants would be able to identify aspects of the current design that (don't) work well or add cognitive difficulty. The prototype was introduced to users and from this, new features and improvements implemented. A subset of these participants who gain sufficient familiarity with the project were used for more informed summative evaluation. As a result, the proposed extension of incorporating live-coding would only be implemented if deemed high priority by the participants.

[1] https://docs.microsoft.com/en-us/javascript/api/excel?view=office-js

MIDI was the file type for which a converter was implemented to translate to a CSV file for use in Excel. Additional explanation on the choice of MIDI is provided below. It was also motivated by participants who wished to be able to integrate Excello with their use of digital audio workstations such as Logic Pro, Ableton Live, and GarageBand.

2.2 Feasibility Analysis

The following section outlines the libraries I explored and the tests carried out to assess the feasibility of synthesising notes given data in a spreadsheet using an Excel add-in. All tests were carried out in Excel Online[2] using Script Lab[3], an add-in that allows users to create and test simple add-ins experimenting with the Office Javascript API. These add-ins have an HTML front end and can access libraries and data elsewhere online.

If add-ins were run in an older version of Internet Explorer, playing sound or using of the Web Audio API would be possible.An add-in that played a wav file verified that an add-in could create sound. [10].

2.2.1 Note Synthesis Library

The Web Audio API allows audio to be synthesised from the browser using Javascript [10]. Creating a program for users to define and play musical structures requires synthesising arbitrary length, pitch and volume notes. To avoid the lower-level audio components (e.g. oscillators), I researched libraries that would allow me to deal with higher level musical abstractions of the synthesised notes. Sarkar's SheetMusic used the library tones[4], a simple API where only the pitch and volume envelope[5] of all notes. Other limitations included no definition of volume and only including simple waveform synthesisers.

Tone.js[6] is a library built on top of the Web Audio API providing greater functionality than tones. An `Instrument` such as a `Synth` or `Sampler` is defined. The `triggerattackrelease` release method of these instruments allows a note of a given pitch, volume and duration to be triggered at a particular time. Notes are defined using scientific pitch notation (SPN) (e.g. `F#4`), name (`F#`) combined with octave (`4`). As Script Lab can reference libraries from the Node Package Manager (NPM), I tested playing notes with pitch defined in the add-in Javascript.

2.2.2 Office Javascript API

For users to produce music within Excel, the musical output must be informed by the data in the spreadsheet. Previous tests created notes defined in the add-in Javascript.

[2]https://office.live.com/start/Excel.aspx
[3]https://www.microsoft.com/en-us/garage/profiles/script-lab/
[4]https://github.com/bit101/tones
[5]How the note's volume changes over its duration
[6]https://tonejs.github.io/

To test the Excel API, I played a note with the Tone.js library, the pitch of which was defined in the spreadsheet. This was extended so note playing, not just the pitch, was defined within a cell, detected and executed.

Next, I was able to play a sequence of constant length notes defined in consecutive cells. The range of cells was accessed using the Excel API and the values were played using the `Tone.Sequence` object. Having performed these tests, I confirmed Tone.js combined with the Excel API had the functionality required to assist in the implementation of the project.

2.3 Excello Design and Language

2.3.1 Abstracting Time

Dave Griffith's Al-Jazari [8] takes place in a three-dimensional world where robotic agents navigate around a two-dimensional grid. The height and colour of the blocks over which the agents traverse determines the sound that they produce. The characteristics of the blocks are modified manually by users at run-time whilst the agents are moving. Whilst there are more complex conditional instructions, the basic instructions have the agents rotate and move forwards and backwards in the direction that they are facing. There exists a dual formalism in both the agent's instructions and the block state. This design is intended to make live coding more accessible, both when viewing performances and becoming a live coder.

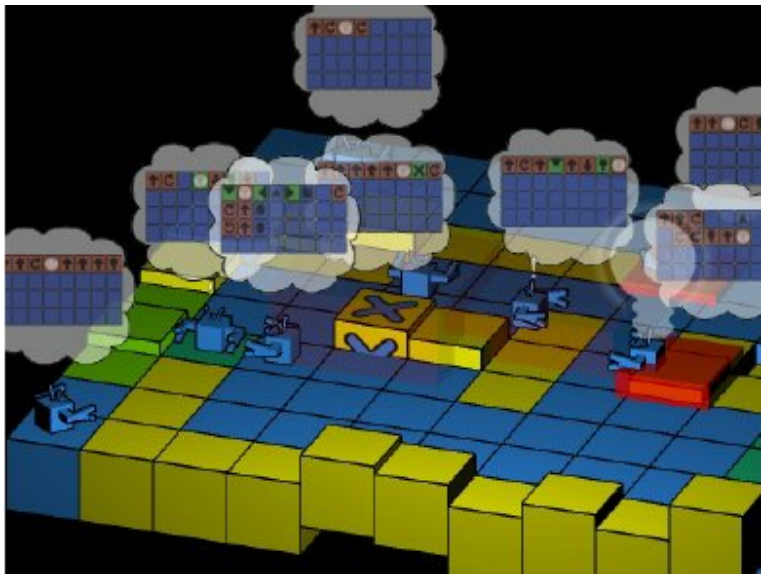


Figure 2.1: Programmed agents moving around a grid in Al-Jazari © Alex Mclean

In Al-Jazari, the agents are programmed by placing symbols corresponding to different movements in thought bubbles that appear above them. This is not suitable for programming within spreadsheets where all data must exist alphanumerically within cells. If an

agent was to continue moving forwards many times in a row, it would become tiresome to keep adding the move forward symbol. This is less of an issue in Al-Jazari where the grid within which the agents navigate only measures ten cells wide and long.

The concept of having a cursor navigate around a cartesian plane is the method used by turtle graphics. Just as this concept is used in Al-Jazari to play the cell the agents occupy rather than colour it, it is suitable for spreadsheets. The turtle abstraction is employed by Excello by having notes defined in cells and defining agents, known as turtles, to move through the spreadsheet activating them. In order to play a chord, multiple turtles must be defined to pass through multiple cells corresponding to the note of the chord. This method maintains high notational consistency but sacrifices the abstractions for musical structures that are available in languages like Sonic Pi - `chord('F#', 'maj7')`. By implementing methods in the add-in to add the notes of chords to the grid, the use of the abstractions is maintained whilst preserving consistency and cleanness in the spreadsheet itself.

The turtle is the crux of the Logo programming language [3]. In Logo, turtles are programmed entirely by text. for example, `repeat 4 [forward 50 right 90]` has a turtle move forwards 50 units and turn 90 degrees to the right. This is repeated four times to draw a square. A similar method is employed in Excello but the language is designed to be less verbose.

2.3.2 Initial Prototype Design

In Excello, notes are placed in the cells of the spreadsheet and pathways through the grid are defined using a language for programming turtle movement. The notes in the cells will be played when a turtle moves through that cell. When the program is run, the melodic lines produced by all turtles defined in the grid will be played concurrently. Turtles are defined with a start cell, movement instructions, the speed with which they move through the grid (cells per minute) and the number of times they repeat their path. As in Al-Jazari, distance in space maps to time [9], Excello extends upon this by allowing different turtles to navigate at different speeds. This allows parts with longer notes to be defined more concisely and for phase music[7] to be easily defined.

As in Logo, turtles begin facing north. The move command `m` moves the turtle forward one cell in the direction that it is facing. Just like in Logo, the turtle always moves in the direction it is facing. The commands `l` and `r` turn the turtle 90 degrees to the left and right respectively. Repeats are implemented in Logo with the command `repeat` followed by the number of repeats and the instructions to be repeated [3]. In order to create more concise instructions, single commands can be repeated in succession by placing a number immediately after it. For example, the command `m4` will have the turtle move forwards four cells in the direction that it is facing. The direction a turtle is facing can be defined absolutely using the commands `n`, `e`, `s` and `w` to face the turtle north, east, south

[7] Music where identical parts are played at different speeds

and west. This could have instead moved the turtle in that direction, but this would have lost the consistency that the turtle always moves in the direction it is facing. To change notes' volume, dynamics (`ppp`, `pp`, `p`, `mp`, `mf`, `f`, `ff`, `fff`) can be placed within the turtle instructions. Any notes played after this will be played at that dynamic. In the same way the dynamics in western notation are a property of the staff and not individual notes, dynamics were originally designed to be a property of the turtle. In order to repeat multiple instruction sequences, these are placed in brackets and the number of repeats put immediately after the bracket. For example, `(r m50)4` would define a path going clockwise around a fifty by fifty square. This 8 character example is equivalent to the Logo example above that requires 30 characters. The ability to repeat larger series of instruction is why the relative movements `l` and `r` are included in the language despite being less explicit than the compass based directions.

It may not be convenient for each melodic line to be defined by a single path of adjacent cells. Just as conventional score notation often spans across multiple lines, the splitting of parts is a useful form of secondary notation. This requires the turtle to navigate to non-adjacent cells and then proceed playing. For graphic drawing in Logo, the pen can be lifted, allowing the turtle to navigate without colouring the space beneath it. This is suitable for a graphical output where the number of steps the turtle takes has no effect on the output, only the cells it colours. However, the musical output is dependent on when the turtle is in certain cells, so this would not be convenient as it would introduce large rests. Analogous to lifting the pen for graphical turtles, one could set the turtle in a mode where it doesn't play the cells it navigates through and passes through them immediately until it is placed back in a playing mode. Here the actual path that the turtle takes is insignificant only the cell it ends up in. I have therefore added jumps to the language. This can be defined in absolute terms where the destination cell is given (e.g. `jA5`), or relatively (e.g. `j-7+1`), where the number of rows and columns jumped is given instead. An absolute jump may be more explicit to the human reader but defining jumps relatively allows them to be repeated, jumping to different cells in each repeat. for example, `r (m7 j-7+1)9 m7` plays 10 rows of 8 cells from top to bottom playing each row left to right.

The language for turtle movement instructions can be summarised by the following context-free grammar, $(N, \Sigma, S, \mathcal{P})$. Where the non-terminal symbols $N = (\mathbf{S}, \mathbf{Y}, \mathbf{X}, \mathbf{I}, \mathbf{R}, \mathbf{A}, \mathbf{P}, \mathbf{C}, \mathbf{D})$, terminal symbols $\Sigma = (z{\in}\mathbb{Z}, n{\in}\mathbb{N}, c{\in}\texttt{[A-Za-z]}^+, \texttt{m}, \texttt{j}, \texttt{l}, \texttt{r}, \texttt{n}, \texttt{e}, \texttt{s}, \texttt{w}, \texttt{+}, \texttt{-}, \texttt{ppp}, \texttt{pp}, \texttt{p}, \texttt{mp}, \texttt{mf}, \texttt{f}, \texttt{ff}, \texttt{fff})$ and starting symbol S. The set of grammar rules are shown in table 2.1:

Notes are defined in the cells using SPN - the note name (with accidental[8] if required) followed by the octave number. Empty cells are interpreted as rests. In order to create notes longer than a single cell, the character `s` in will sustain the note that came before it. This is used to create notes longer than the duration of a single cell. A cell can be sub-divided time-wise into multiple notes by placing multiple notes separated by commas

[8]Sharp or flat symbol used to define a black note on a piano keyboard.

Table 2.1: Grammar rules for turtle movement instructions. $z \in \mathbb{Z}, n \in \mathbb{N}, c \in$ `[A-Za-z]`$^+$.

Grammar Rule	**Left Symbol Interpretation**
$\mathbf{S} \rightarrow \mathbf{Y}$	Starting symbol
$\mathbf{Y} \rightarrow \mathbf{X}\vert\mathbf{X}\ \mathbf{Y}$	A series of instructions
$\mathbf{X} \rightarrow (\mathbf{Y})z\vert\mathbf{I}$	A single command or bracketed series of instructions
$\mathbf{I} \rightarrow$ `m`$z\vert\mathbf{R}\vert\mathbf{R}z\vert\mathbf{A}\vert\mathbf{D}\vert$`j`$\mathbf{C}\vert$`j`$\mathbf{P}n\mathbf{P}n$	A single command
$\mathbf{R} \rightarrow$ `l`$\vert$`r`	Relative rotation
$\mathbf{A} \rightarrow$ `n`$\vert$`e`$\vert$`s`$\vert$`w`	Absolute rotation
$\mathbf{P} \rightarrow$ `+`$\vert$`-`	Sign
$\mathbf{C} \rightarrow cz$	Cell reference
$\mathbf{D} \rightarrow$ `ppp`$\vert$`pp`$\vert$`p`$\vert$`mp`$\vert$`mf`$\vert$`f`$\vert$`ff`$\vert$`fff`	Dynamic

into a cell. The motivation for this design decision was so the length each cell corresponds to is not bound by the length of the smallest note in the piece. For example, a piece defined primarily with crotchets (one unit) but with a single instance of a quaver (half a unit) and dotted crotchet (one and a half units) can define these two notes with `C4,C4` and `s` in two cells. Without this, representing this single quaver would require double the number of cells and introducing many additional `s` cells in the entire piece.

2.4 Software Engineering

2.4.1 Requirements

The success criteria of the project are as follows:

1. Implementation of an API for music playback within a spreadsheet where users can:
   - Play individual notes and chords and define their durations.
   - Define multiple parts.
   - Play loops.
   - Define sequences of notes and chords and be able to call these for playback.
   - Define the tempo of playback.
2. Performance of participatory design sessions.
3. Usability testing using participants who have gained familiarity with the system.
4. Implementation of a converter from MIDI to the spreadsheet representation.
5. In addition to these, the following extension work was completed::
   - Implement additional features that arise from participatory design.
   - Explore a compressive conversion from MIDI to the Excel system.

2.4.2 Tools and Technologies Used

Table following table outlines the tools, languages and libraries used.

Software	**Type**	**Usage**
Scriptlab	Add-in	Writing initial Excel add-in tests.
Typescript	Language	Writing Excello. Used for static type checking and Javascript libraries.
Yeoman[9]	Add-in Generator	Creating the blank Excel add-in project.
NodeJS	Javascript Notebook	Manage library dependecies and run local web servers.
Surge[10]	CDN	Hosting Excello online for participants' use.
Jupyter Notebook	Python Environment	Implementing the MIDI to Excello converter.
Tone.js	Library	Synthesising and scheduling sound via the Web Audio API.
tonal	Library	Generating the notes of chords.
Mido[11]	Library	Reading MIDI files in Python.

2.4.3 Starting Point

Having used the Yeoman generator to create an empty Excel add-in, all of the code used to produce Excello and the MIDI converter is produced from scratch using the tools and technologies described above.

I had written simple Javascript for small web pages, but no experience using Node, libraries or building a larger project. I had never used any of the libraries before, therefore, reviewing the documentation was required before and during development. I had gained significant experience with Python and Jupyter Notebooks from a summer internship.

2.4.4 Evaluation Practices

To best tune the design of Excello to the needs of potential users, formative evaluation sessions were carried out with participants. A spiral development methodology [2] was used. The following steps were iterated: determining objectives, identify problems and solutions, develop and test, deploy and prepare next iteration. This was more suitable than the sequential document-driven process methods such as the waterfall method. Due to the number of participants and timeframe of the project, there were only two major development iterations with additional incremental updates. The first prototype following the design described above, and the second fixing issues and implementing requests brought up by the participants.

Summative evaluation was carried out with users involved in participatory design. Therefore, experienced users of Excello could be used for evaluation despite the product not yet being released in the public domain.

2.5 MIDI files

Musical Instrument Digital Interface (MIDI) details a communications protocol to connect electronic musical instruments with devices for playing, editing and recording music. A MIDI file consists of event messages describing on/off triggerings for a device or program to control audio [7]. MIDI files were designed to be produced by MIDI controllers such as an electric keyboard. As such, a MIDI file contains a lot of controller specific information that is not necessary for the creation of an Excello file. There exist musical formats such as MusicXML that specify the musical notation and as such may be more suitable for conversion to Excello.

Many musical programs support the importing and exporting of MIDI files. By allowing MIDI files to be converted to the Excello notation, Excello is more integrated into the environment of computer programs for playing, editing and composing music. Furthermore, there exist many datasets available for MIDI [6] which can immediately be played back for comparison.

Bibliography

[1] Samuel Aaron, Alan F. Blackwell, and Pamela Burnard. The development of sonic pi and its use in educational partnerships: Co-creating pedagogies for learning computer programming. *Journal of Music, Technology and Education*, 9:75–94, 05 2016.

[2] B Boehm. A spiral model of software development and enhancement. *SIGSOFT Softw. Eng. Notes*, 11(4):14–24, August 1986.

[3] Ron Goldman, Scott Schaefer, and Tao Ju. Turtle geometry in computer graphics and computer-aided design. *Computer-Aided Design*, 36:1471–1482, 2004.

[4] Thomas Green and Alan Blackwell. Cognitive dimensions of information artefacts: a tutorial. Technical Report Version 1.2, BCS HCI Conference, 1998.

[5] Sven Gregori. Never mind the sheet music, heres spreadsheet music, 2019.

[6] Allen Huang and Raymond Wu. Deep learning for music. *CoRR*, abs/1606.04930, 2016.

[7] D.M. Huber. *The MIDI Manual: A Practical Guide to MIDI in the Project Studio*. Taylor & Francis, 2012.

[8] Alex Mclean, Dave Griffiths, Foam Vzw, Dave@fo Am, Nick Collins, and Geraint Wiggins. Visualisation of live code. 01 2010.

[9] Alex Mclean and Geraint Wiggins. Texture: Visual notation for live coding of pattern. 01 2011.

[10] Mozilla. Web audio api. `https://developer.mozilla.org/en-US/docs/Web/API/Web_Audio_API`, 03 2019. Accessed: 2019-04-02).

[11] Michael Muller and Sarah Kuhn. Participatory design. *Communications of the ACM*, 36:24–28, 06 1993.

[12] Chris Nash. Manhattan: End-user programming for music. In *NIME*, 2014.

[13] Simon Peyton Jones, Margaret Burnett, and Alan Blackwell. A user-centred approach to functions in excel. June 2003.

[14] Advait Sarkar. Towards spreadsheet tools for end-user music programming. In *PPIG*, 2016.

[15] C. Scaffidi, M. Shaw, and B. Myers. Estimating the numbers of end users and end user programmers. In *2005 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC'05)*, pages 207–214, Sep. 2005.