Gridlets: Reusing Spreadsheet Grids

Nima Joharizadeh
University of California, Davis

Advait Sarkar
Microsoft Research and
University of Cambridge

Andrew D. Gordon
Microsoft Research and
University of Edinburgh

Jack Williams
Microsoft Research

CHI '20 Extended Abstracts, April 25–30, 2020, Honolulu, HI, USA.

ACM ISBN 978-1-4503-6819-3/20/04.
http://dx.doi.org/10.1145/3334480.3382806

Abstract
Spreadsheets allow end users to blend calculations with arbitrary layout and formatting. However, when it comes to reusing groups of formulae along with layout and formatting, spreadsheets provide only limited support. Most users rely on copy and paste, which is easy to learn and use, but maintaining several copies can be tedious and error-prone. We present the concept of Gridlets, an abstraction over calculation and presentation applicable in common use case scenarios. Using the Cognitive Dimensions of Notations framework, we compare Gridlets to copy/paste and sheet-defined functions. We find that Gridlets are consistent with the spreadsheet paradigm, enable users to take advantage of secondary notation, and make common edit operations less viscous and less error-prone.

Author Keywords
Spreadsheets; Abstraction; End-User Programming

CCS Concepts
•**Human-centered computing** → **Heuristic evaluations;**
•**Applied computing** → **Spreadsheets;**

Introduction
Reuse in spreadsheets is overwhelmingly common. Users routinely duplicate template files or copy and paste regions of the grid. Despite the convenience, there is a cost.

	A	B
1	**Mortgage calculator**	
2		
3	**Mortgage details**	
4	Loan amount	£150,000.00
5	Deposit	£10,000.00
6	Term (Years)	30
7	Interest Rate	3.4%
8	Financed amount	£140,000.00
9	Monthly payment	£620.87
10	Total repaid	£223,514.54
11	Total interest	£83,514.54
12		
13	**Mortgage details**	
14	Loan amount	£300,000.00
15	Deposit	£40,000.00
16	Term (Years)	40
17	Interest Rate	2.7%
18	Financed amount	£260,000.00
19	Monthly payment	£886.37
20	Total repaid	£425,459.55
21	Total interest	£165,459.55
22		
23	**Mortgage details**	
24	Loan amount	£200,000.00
25	Deposit	£20,000.00
26	Term (Years)	25
27	Interest Rate	5.2%
28	Financed amount	£180,000.00
29	Monthly payment	£1,073.34
30	Total repaid	£322,002.82
31	Total interest	£142,002.82
32		
33	**Mortgage details**	
34	Loan amount	£102,000.00
35	Deposit	£43,000.00
36	Term (Years)	17
37	Interest Rate	6.2%
38	Financed amount	£59,000.00
39	Monthly payment	£468.61
40	Total repaid	£95,595.45
41	Total interest	£36,595.45

Figure 1: A spreadsheet for comparing mortgages. The range `A3:B11` has been copied and pasted 3 times, and the input values modified for each copy.

Spreadsheet edits do not propagate to every copy, therefore applying a change to a region and its copies—*a uniform edit*—will suffer from two significant issues:

1. High effort: When the source is edited, cognitive and physical effort is required to recall, locate, and edit every copy. This inflexibility of spreadsheets in allowing refinement of copied sources inhibits effective reuse.

2. High error-proneness: Since uniform edits are applied manually to every copy, there is the potential for error: the edit might be applied inconsistently, and not all copies might be recalled and edited. This leads to an inconsistency between source and (intended) copies, and errors that are difficult to detect and fix. This is a major and common problem for spreadsheets [6].

Consider the spreadsheet in Figure 1, in which the user is trying to compare four different mortgages. The user has set up the spreadsheet so that she can enter the loan amount, deposit, loan term, and interest rate as inputs (orange), and read out the financed amount, monthly payment, total repaid, and total interest as outputs (grey).

In this scenario, the user has created a portion of grid in the range `A3:B11` consisting of data, textual labels, formatting, and labels. She has copied and pasted this range three times, to reuse most of these assets, but overwrites the input values for each different mortgage.

The problem of copy, paste, and uniform edit
Now suppose the user discovers an error in a formula or label, or wishes to update some aspect of the formatting. Having made the required edits, the user would once again *copy* the range `A3:B11`, *paste* it into the subsequent locations, and then *modify* each copy to restore the input values. This sequence of operations is effortful and error-prone.

Gridlets
We designed Gridlets to ameliorate the problems with uniform edit. In general, *a Gridlet is simply a copy of a range including formatting, with local modifications applied to the range, and a live link to the copied range.* We can create Gridlets through a graphical interface, or a formula. In this paper we illustrate a formula function as an interface for Gridlets. Implementing Gridlets as a formula requires no novel extensions to the spreadsheet user interface. With Gridlets, instead of making multiple manual copies of the source, for each intended copy, the user writes a formula such as `=GRIDLET(A3:B11, B4, 300000, B5, 40000, B6, 40, B7, 0.027)`, which reads:

"Take the source range `A3:B11`, replace `B4` with `300000`, `B5` with `40000`, `B6` with `40`, and `B7` with `0.027`, and then calculate the value of the range as an array."

Thus, the formula specifies a portion of the grid to copy, and a list of substitutions to make within that copy. Formulas can also be substituted. While the `GRIDLET` formula itself sits in a single cell, the value it returns is an array that "spills" into the grid. The idea of an array value "spilling" or "spreading" itself into adjacent cells is supported in some commercial spreadsheet applications,[1] and has been formalised [11]. A clearer illustration of how a call to the `GRIDLET` function would "spill" is given in Figure 2.

Because the copies are calculated through the `GRIDLET` function, the user needs to modify only the source range to correct a formula or update formatting across all copies, while preserving the substitutions local to each copy.

[1] E.g., https://aka.ms/excel-dynamic-arrays

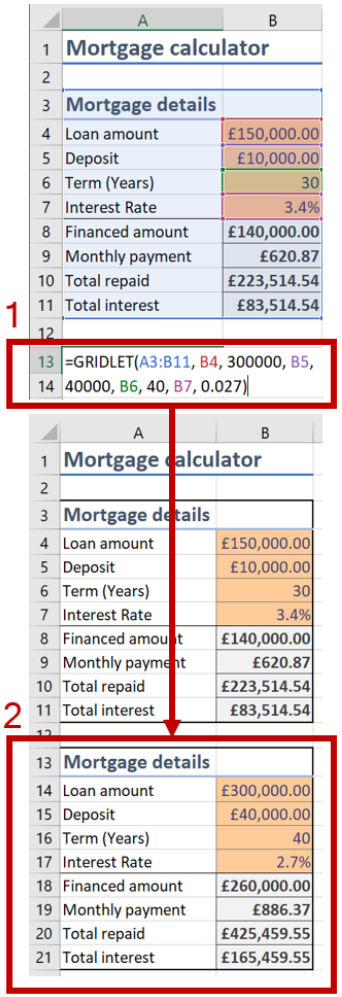

Figure 2: Constructing a copy using a Gridlet formula. The Gridlet formula returns an array that "spills" into adjacent cells. Changes in the source will automatically propagate to this copy, while preserving the local substitutions.

In this manner Gridlets provide a mechanism to abstract over the grid, carrying data, formatting, and formulas – everything the user would expect from copy/paste. However, unlike copy/paste, Gridlets maintain a link between copies and their sources. This reduces the complexity of uniform edit by alleviating the need to manually recall, locate, and update every copy.

Gridlets via UI

It should be possible to improve the usability of Gridlets and make them accessible to a wider audience (especially users without the inclination or expertise to write formulas) through graphical interfaces. Two examples are as follows:

Gridlet wizard and library A graphical wizard could lead the user through the creation and use of Gridlets. For example, the user could be prompted to select a portion of the grid to "save" as a Gridlet in a library. Gridlets thus created could be invoked by dragging and dropping onto the grid, as in Figure 3. The user would be able to modify values directly in the grid, but the functionality would be provided through a call to `GRIDLET` which could optionally be hidden from the user altogether.

Gridlet as a paste option Modern spreadsheet applications allow copied items to be pasted in many different forms, such as paste values only, paste formatting only, paste transposed values, etc. An option to "paste as Gridlet" could be introduced, which pastes a portion of grid that appears identical to a regular paste, except that since the functionality is provided through a call to `GRIDLET`, updates from the source would propagate to every copy, while honouring local substitutions. This would introduce level 3 liveness [10] to copy/paste (in particular, to uniform edit). Level 3 liveness is achieved when "any edit operation by the user triggers [re-]computation". Thus, "paste as Gridlet" could alternatively be referred to as "*live* copy/paste". This option would make Gridlets much more competitive with copy/paste along the consistency dimension.

Sheet-defined functions: an alternative solution

We have presented Gridlets as an alternative to copy/paste in spreadsheets with a focus on improving uniform edit. Another proposed spreadsheet reuse mechanism, amenable to some cases of uniform edit, is the *Sheet-Defined Function* [4, 8, 9] (**SDF**). An SDF is a function that the user can define from an existing region of the grid. Users explicitly select the input and output cells of the function at the definition site, which are then used at every call site.

SDFs are a straightforward importation of the idea of functions or subroutines from traditional textual programming languages. The priority is *hiding implementation details*; the inner mechanisms of a subroutine add unnecessary complexity to the activity of programming, and so SDFs are appropriate for re-use when intermediate calculations, layout, and formatting are unimportant, and a compact 'black-box' calculation is required. SDFs are therefore envisioned as a way for users to extend the library of built-in formula functions by naming calculations defined in the grid. This is more easily understood by looking at Figure 4.

Gridlets, on the other hand, carry layout, formatting, and intermediate computations, and every call site to a gridlet can make substitutions in different locations. Rather than providing a 'black-box' abstraction, Gridlets provide a transparent abstraction for achieving the same effect as copy/paste, but with benefits for uniform edit.

SDFs, Gridlets, and Copy/paste are not in competition with each other as they provide different forms of abstraction. SDFs are most useful when a black-box abstraction is re-

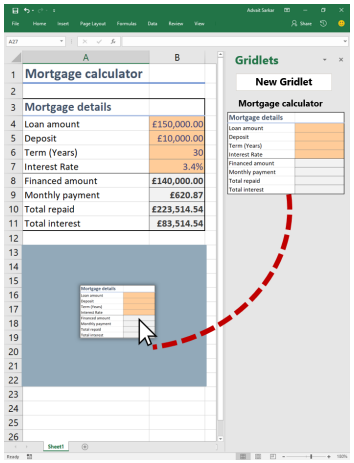

Figure 3: Dragging and dropping a Gridlet from a library to create a new copy. An illustration of an alternative graphical interface to the `GRIDLET` function.

quired. Gridlets are most useful when uniform edit is common. Copy/paste is useful for all other cases.

Related Work

Gradual structuring [5], the "Lish" data model [3], and Calculation View [7] all enhance the two dimensional layout of the grid by abstracting over a group of rows (or columns). Like SDFs, they reduce the viscosity and error-proneness of uniform edit, but at the cost of flexibility; they do not allow variation in local substitutions.

Similarity inheritance [1] is a model for inheritance in visual programming languages such as Forms/3. Similarity inheritance is an abstract notion and concrete realizations of it depend on the interaction model. In Forms/3, it is introduced by overloading the conventional copy/paste operation to establish inheritance relationships among cells and forms. This mechanism is similar to using Gridlets as a paste option. Furthermore, the semantics of similarity inheritance resemble substitution in Gridlets, albeit as applied to forms and not rectangular grids.

Evaluating Gridlets: what are they good for?

How does the idea of Gridlets compare to other mechanisms for reuse in spreadsheets, and what are its relative strengths and weaknesses? The *Cognitive Dimensions of Notations* [2] (**CDN**) is a widely-used framework that provides a vocabulary for discussing the usability properties of programming language features or "notations". In our analysis, we focus on seven dimensions:

Consistency: When some of the language has been learnt, how much of the rest can be inferred? We desire our reuse mechanism to have *high* consistency with the spreadsheet paradigm, requiring the user to learn fewer concepts.

Viscosity: How much effort is required to make a change to a program expressed in the notation? The high manual effort of uniform edit means that this operation typically has high viscosity. We desire our reuse mechanism to have *low* viscosity with respect to uniform edit.

Error-proneness: To what extent does the notation influence the likelihood of the user making a mistake? As previously discussed, uniform edit typically has high error-proneness. We desire our reuse mechanism to have *low* error-proneness with respect to uniform edit.

Premature commitment: Are there constraints on the order of doing things? Is the user forced to make decisions before they might have all the available information for doing so, thereby creating future work to correct such decisions? We desire our reuse mechanism to have *low* premature commitment.

Hidden dependencies: Are dependencies between entities in the notation visible or hidden? Is every dependency indicated in both directions? Does a change in one area of the notation lead to unexpected consequences? We desire our reuse mechanism to be *low* on hidden dependencies, to minimise unexpected consequences.

Hard mental operations: Are there places where the user needs to resort to pencilled annotation to keep track of what is happening? As previously discussed, a successful uniform edit requires the user to recall, locate, and edit each copy, which can incur high cognitive costs, i.e., it is high on hard mental operations. We desire our reuse mechanism to be *low* on hard mental operations.

Secondary notation: Can the notation carry extra information by means not related to syntax, such as layout, color, or other cues? Spreadsheet users make liberal and effective

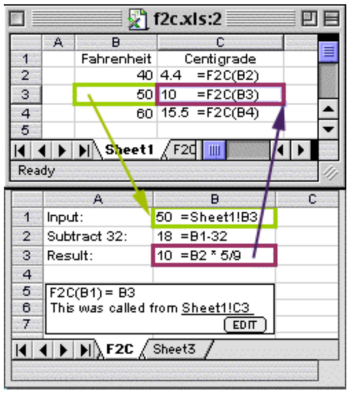

Figure 4: Sheet-defined functions (Reproduced from Peyton Jones et al. [4])

At the bottom we see a function instance sheet. The sheet "F2C" provides the definition of the `F2C` function (hence the function is *sheet-defined*), with cell `B1` receiving input, cell `B2` performing an intermediate calculation, and cell `B3` providing the output.

At the top we see a function invocation site. In sheet "Sheet1", cells `C2` to `C4` all make use of the function `F2C` as a black-box abstraction, supplying inputs as arguments and receiving a single output value.

use of secondary notation in the form of layout and formatting. We desire our reuse mechanism to have *high* support for secondary notation.

We now compare Gridlets, sheet-defined functions and copy/paste along the cognitive dimensions.

Consistency
The original exposition of SDFs cites consistency as a core motivation: "The implementation of a function must be defined by a spreadsheet, because that is the only computational paradigm understood by our target audience." [4] The same benefit applies to Gridlets as they exploit familiarity with ranges and formulas. Introducing a new notation that resembles existing notation, yet elevates the level of abstraction available to the user, has been shown to be a successful strategy in spreadsheets. One example is the *range assignment* notation of Calculation View [7].

Copy/paste, arguably, is the reuse mechanism having the highest consistency with the spreadsheet paradigm (and is consistent with reuse in much productivity software). SDFs and Gridlets have similar (good) levels of consistency to each other, requiring only an understanding of spreadsheet formulas, but since Gridlets allow the transmission of secondary notation in a manner very similar to copy/paste, it can be said to have higher consistency than SDFs.

Viscosity and Error-proneness
In our context, the *viscosity* and *error-proneness* dimensions are highly related. Copy/Paste increases the cost of uniform edit because the user must manually recall, locate, and edit every copy. This process is also error-prone. Gridlets, by design, reduce the viscosity and error-proneness of uniform edit in most cases. In the subset of Copy/paste scenarios where a black-box abstraction is appropriate, SDFs similarly reduce viscosity and error-proneness.

With Gridlets and SDFs, some uniform edit scenarios are still viscous and error-prone. Consider what must happen when a re-used calculation that notionally takes two arguments is altered to take three. Copy/paste requires each copy to be edited. SDFs require each call site to be edited to introduce a new argument. Gridlets require each use site to be edited to introduce a new argument. In general this happens whenever there is a change to the location or number of arguments, but this is no worse than Copy/paste.

One workaround to this problem is use of names instead of cell addresses. Some commercial spreadsheets include features that would enable this. For example, in Excel, the *Name Manager* allows users to assign human-readable names to cell and range addresses, and use these in formulas. This provides the additional flexibility to change the location of inputs or dimensions of the Gridlet while eliminating the need to update each invocation of `GRIDLET`.

Premature commitment
SDFs require the highest levels of premature commitment. When an SDF is created the user must nominate input and output cells, as well as a single calculation. The choice applies to every invocation therefore the user must commit to a particular usage pattern for that function. Gridlets require less premature commitment as every Gridlet invocation can have a unique set of substituted values and formulas that is suitable for the particular instance.

Copy/paste requires the least premature commitment. Both Gridlets and SDFs have a notion of precedence; a specific source area of the grid must be modified for uniform edit to take place. In copy/paste, the user can perform a uniform edit starting with any copy; any copy can become the new source. Copy/paste also allows certain kinds of edit that are not achievable through the substitution mechanic of Gridlets, such as such as inserting a row.

Rank by desirability	Copy/Paste	Gridlet	SDF
Consistency	1	2	3
Viscosity	3	1	1
Error-proneness	3	1	1
Premature commitment	1	2	3
Hidden dependencies	3	1	1
Hard mental operations	3	1	1
Secondary notation	1	2	3

Table 1: Summary of CDN analysis. Note: systems are ranked higher for having high Consistency and Secondary Notation, but for other dimensions, they are ranked higher for being lower on those dimensions.

Hidden dependencies
Since Gridlet invocations blend seamlessly with the rest of the grid, it is possible that a Gridlet invocation uses an area on the grid that itself is the output of another Gridlet invocation. This might introduce a chain of hidden dependencies that is difficult to track. This same limitation applies to SDFs. One mitigating strategy is to notify the end-user programmer about forward and backward dependencies of a cell as the user modifies the cell.

Superficially, it might appear that copy/paste, because it does not maintain a link between source and copy, does not introduce any hidden dependencies. That is true if the user does not wish to maintain a link. However, if they do, the dependency between source and copy still exists, but the burden of maintaining it has been shifted to the user! For this reason copy/paste can be worse than both Gridlets and SDFs in terms of hidden dependencies.

Hard mental operations
For copy/paste, the operation of uniform edit requires the user to manually *recall*, *locate*, and *edit* each copy. Each of these is a hard mental operation. Recalling every instance of reuse is especially challenging in large spreadsheets, old spreadsheets, spreadsheets that have been authored by other people, spreadsheets that have been poorly annotated, and so on. Locating an instance of reuse involves navigating to the correct region of the correct sheet in the correct workbook. Moreover, the user must keep track of the intended edit and local modifications for each copy while performing uniform edit; these might be tracked either in the user's own working memory, or the user might resort to using temporary scratch space in the spreadsheet. Both Gridlets and SDFs mitigate these hard mental operations.

However, it is far from clear whether Gridlets incur lower cognitive costs overall than copy/paste. Both Gridlets and SDFs depend on a more abstract mental model than copy/paste, and require expertise in writing formulas. The acquisition of these mental model incurs significant upfront cognitive costs which are a high barrier for many users. However, once the correct mental model is acquired, this is not a recurring hard mental operation. In future work, we also intend to explore graphical interfaces for Gridlet creation and use, that further lower the cognitive barriers.

Secondary notation
Layout and formatting are an important and frequently used form of secondary notation. Copy/paste and Gridlets both carry this form of secondary notation, while SDFs do not.

In summary, we anticipate that Gridlets improve on copy/paste in terms of viscosity, error-proneness, hidden dependencies and hard mental operations, at the cost of slightly lower consistency and support for secondary notation. However, Gridlets are still an improvement over SDFs in terms of consistency, premature commitment, and secondary notation. An illustrated summary is given in Table 1. It is important to note that our analysis is a form of heuristic evaluation that gives us a preliminary indication of the potential tradeoffs, but which should be further substantiated with an empirical user study, that we intend to do in the future.

Conclusion

We present the design and preliminary evaluation of Gridlets, an abstraction for uniformly propagating edits to copied regions of a spreadsheet. The Cognitive Dimensions of Notations framework anticipates that using Gridlets can reduce the labor and error-proneness associated with copy and paste. In future work, we intend to evaluate these predictions with an empirical study, and also explore Gridlets as a paste option to further lower their cognitive costs.

REFERENCES

[1] Rebecca Walpole Djang and Margaret M Burnett. 1998. Similarity inheritance: a new model of inheritance for spreadsheet VPLs. In *Proceedings. 1998 IEEE Symposium on Visual Languages (Cat. No. 98TB100254)*. IEEE, 134–141.

[2] T. R. G. Green. 1989. Cognitive Dimensions of Notations. In *Proceedings of the Fifth Conference of the British Computer Society, Human-Computer Interaction Specialist Group on People and Computers V*. Cambridge University Press, New York, NY, USA, 443–460. http://dl.acm.org/citation.cfm?id=92968.93015

[3] Alan Hall, Michel Wermelinger, Tony Hirst, and Santi Phithakkitnukoon. 2018. Structuring Spreadsheets with the "Lish" Data Model. *arXiv preprint arXiv:1801.08603* (2018).

[4] Simon Peyton Jones, Alan Blackwell, and Margaret Burnett. 2003. A user-centred approach to functions in Excel. *ACM SIGPLAN Notices* 38, 9 (2003), 165–176.

[5] Gary Miller, Felienne Hermans, and Robin Braun. 2016. Gradual structuring: Evolving the spreadsheet paradigm for expressiveness and learnability. In *2016 15th International Conference on Information Technology Based Higher Education and Training (ITHET)*. IEEE, 1–8.

[6] Ray Panko. 2016. What we don't know about spreadsheet errors today: The facts, why we don't believe them, and what we need to do. *arXiv preprint arXiv:1602.02601* (2016).

[7] Advait Sarkar, Andrew D. Gordon, Simon Peyton Jones, and Neil Toronto. 2018. Calculation View: multiple-representation editing in spreadsheets. In *2018 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. 85–93. DOI: http://dx.doi.org/10.1109/VLHCC.2018.8506584

[8] Peter Sestoft. 2008. Implementing function spreadsheets. In *Proceedings of the 4th international workshop on End-user software engineering*. ACM, 91–94.

[9] Peter Sestoft and Jens Zeilund Sørensen. 2013. Sheet-Defined Functions: Implementation and Initial Evaluation. In *End-User Development*, Yvonne Dittrich, Margaret Burnett, Anders Mørch, and David Redmiles (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 88–103.

[10] Steven L Tanimoto. 1990. VIVA: A visual language for image processing. *Journal of Visual Languages & Computing* 1, 2 (1990), 127–139.

[11] Jack Williams, Nima Joharizadeh, Andrew D. Gordon, and Advait Sarkar. 2020. Higher-Order Spreadsheets with Spilled Arrays. (2020). To appear in Proceedings of 29th European Symposium on Programming (ESOP).