Adding Deductive Logic to a COTS Spreadsheet

Marcelo Tallis
Teknowledge Corp.
mtallis@teknowledge.com

Rand Waltzman
Teknowledge Corp.
rand@nada.kth.se

Robert Balzer
Teknowledge Corp.
bbalzer@teknowledge.com

Abstract

We exploit the spreadsheet metaphor to make deductive problem-solving methods available to the vast population of spreadsheet end users. In particular, we show how the function-based problem-solving capabilities of spreadsheets can be extended to include logical deductive methods in a way that is consistent with the existing spreadsheet "look and feel." The foundation of our approach is the integration of a standard deductive logic system into a successful Commercial-Off-The-Shelf (COTS) spreadsheet. We have demonstrated this by designing and implementing an extension to Excel that manages the integration of Excel and a deductive logic engine based on the World Wide Web Consortium (W3C) standard ontology language OWL + SWRL.

Introduction

End-user programmers (Nardi 1993) are people who write programs for their own use but are not employed as programmers and are usually not, in fact, trained programmers at all. They can be a teacher, engineer, physicist, secretary, accountant, or manager. They use computers as an essential tool for getting their job done but are not normally interested in programming per se. These end-user programmers currently outnumber professional programmers by more than an order of magnitude. In the U.S. alone, they are estimated to outnumber trained professional programmers by 55 million to 2.75 million in 2005 (Boehm et al. 2000). The spreadsheet is the programming language of choice for the overwhelming majority of these end-users. It is widely recognized as one of the few true success stories among systems for end-user programming.

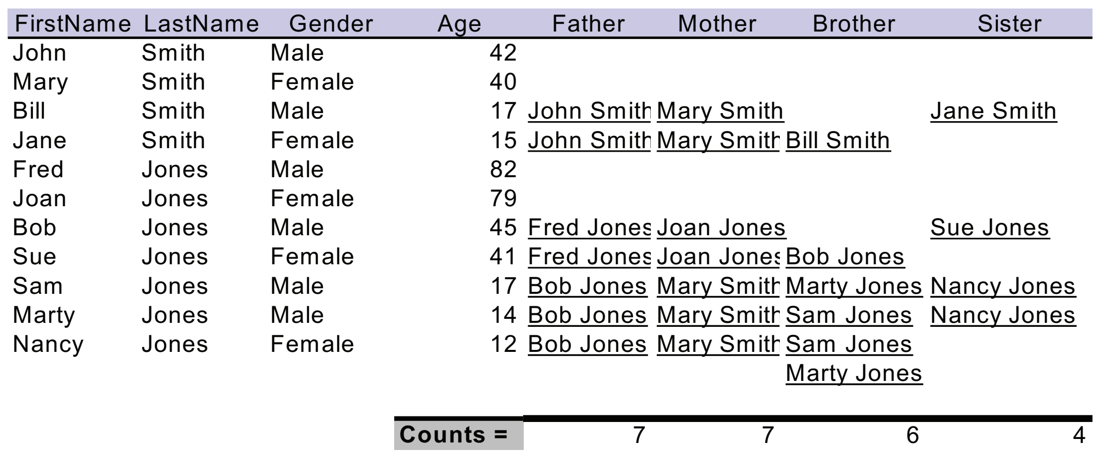
Studies have shown (Nardi 1993) that the spreadsheet's key to success is the combination of computational techniques that match the user's tasks (insulating them from the low-level details forced on professional programmers) and a table-oriented interface that serves as a model for users' applications. The structured visual format for data representation turns out to play a crucial role in an end-user's ability to formulate and critique models. We can summarize the spreadsheet's power by the expression computation + presentation. One of these two components alone is not sufficient.

Computationally speaking, a spreadsheet is essentially a visualization mechanism for a piecewise functional program that maintains state between computations. It has been shown (Nuñez 2000) that spreadsheets and dataflow diagrams are equivalent to one another. The spreadsheet can be thought of as consisting of two components. The first is logical (the Dataflow Logical Layer) and the second is physical (the Adjacency-based Editing Layer). The nodes of the dataflow diagram (i.e., the functional components of the program) are the formulae that are written by the end-users. Problem-solving with a spreadsheet amounts, therefore, to using a grid-based editor to construct a type of functional program by exploiting the dataflow concept.

However, many high-level problem-solving tasks are better expressed in logical deductive terms. Here we use "better" in the sense that logical deduction more directly expresses the way the end-user thinks about the problem. As we have mentioned above, the availability of computational techniques that closely match the user's tasks is one of the two key strengths of spreadsheet programming languages. The opportunity here is to exploit the spreadsheet metaphor in order to make deductive problem-solving methods available to the vast number of end-users already using spreadsheets with only a modest investment of their time.

Our goal in this work has been twofold. First, we wished to demonstrate that we can, in fact, exploit the spreadsheet metaphor to make deductive problem-solving methods available to the spreadsheet end-user community. Second, we wished to make these methods available as efficiently as we could to the widest possible audience.

We have designed and implemented a demonstration prototype as an extension to Microsoft Excel, one of the most popular and successful spreadsheet program to date. Also, rather than develop a new deductive system or work with an idiosyncratic research prototype, we based the integration of the spreadsheet on well established deductive logic engines implementing the World Wide Web Consortium (W3C) standard ontology language OWL (McGuinness and van Harmelen 2004) including SWRL rules (Horrocks et all 2004).



FirstName	LastName	Gender	Age	Father	Mother	Brother	Sister
John	Smith	Male	42				
Mary	Smith	Female	40				
Bill	Smith	Male	17	John Smith	Mary Smith		Jane Smith
Jane	Smith	Female	15	John Smith	Mary Smith	Bill Smith	
Fred	Jones	Male	82				
Joan	Jones	Female	79				
Bob	Jones	Male	45	Fred Jones	Joan Jones		Sue Jones
Sue	Jones	Female	41	Fred Jones	Joan Jones	Bob Jones	
Sam	Jones	Male	17	Bob Jones	Mary Smith	Marty Jones	Nancy Jones
Marty	Jones	Male	14	Bob Jones	Mary Smith	Sam Jones	Nancy Jones
Nancy	Jones	Female	12	Bob Jones	Mary Smith	Sam Jones Marty Jones	
			Counts =	7	7	6	4

Figure 1 A Typical Spreadsheet

The prototype implements the critical subset of features that illustrates our approach. In particular, the Deductive Logic Mapping Manager maintains the consistency between the spreadsheet data and the deductive logic system's knowledgebase by mapping information between them as the spreadsheet data changes (by user action or by calculations performed by the spreadsheet) and as the deductive logic system infers or retracts derived data. Some cells act as inputs to the deductive logic system with their values mapped into OWL triples that are asserted into the reasoning engine. Others are specified as output cells that receive the result of mapping OWL triples, usually derived by the deduction engine, into spreadsheet values. We have extended Excel by enriching the contents of spreadsheet cells. Normally, cells can only hold values of type number, date, or string. We have enabled one additional type of value to be placed into a cell: Typed Objects (individual instances of a type such as person). We have also enabled the handling of multi-valued sets (for handling multi-valued relationships such as the siblings of a person).

To facilitate the adoption of the deductive capabilities we provide logic interpretations for some conventional spreadsheet structures. For example, tabular structures can often be logically interpreted with the rows representing the instances of a class (such as the divisions of a company) and the columns representing attributes of those instances (such as the name, size, revenue, and expenses of that division).

We believe our approach results in several important benefits to users:

- They will have a comprehensive, usable, and mature GUI (and underlying spreadsheet) with which they are already familiar. That will serve as an excellent foundation that they can build on to extend their already existing spreadsheet skills to include the tools of deductive reasoning.
- They will gain ready access to Semantic Web technology from within a familiar environment. This will facilitate the transition of this important technology from the research laboratory to widespread use and acceptance.
- They will be able to use the information derived by the deductive logic system just like any other computed spreadsheet value. This also means that Excel's huge library of built-in spreadsheet functions (such as present-value and moving averages) is available for use with the deductive logic data.
- They will be able to incrementally adopt the deductive logic capabilities in their existing spreadsheets rather than being forced to create completely new spreadsheets to use even a little deductive logic.

Each of these benefits also significantly contributes to the commercializabilty of the resulting product. Together they make an overwhelming argument for this approach.

A significant feature of our design is the clean separation and interface between the Deductive Logic Mapping Manager and the reasoning engine. We have demonstrated the power and flexibility of our design by implementing instances of this interface with three completely different reasoning engines – Jess, Prolog, and KAON2 (KAON2). The ease with which this was accomplished is partly due to our choice of OWL as the basis for spreadsheet/reasoning integration. Because OWL is a W3C standard for use in the Semantic Web, it is the subject of intensive research and development. Our architecture permits us to easily incorporate results of that development as they become available. We have demonstrated this by making use of two such efforts. We used the SweetRules framework developed by the SweetRules group at MIT to automatically generate both Jess and Prolog versions of our OWL ontology and rule base which we then used with well established Jess and Prolog reasoning engines respectively. More recently, we have also implemented an interface to another logic engine option based on the KAON2 framework (KAON2).

A Deductive Logic Layer for Excel

Figure 1 shows a table that might appear in a typical spreadsheet. This table lists a group of persons where each row describes a person and each column indicates the values for one of the person's properties (e.g., gender). Some values in the table can be deduced from other values in the same table. For example, the brothers and sisters of a person can be deduced from the gender, the father, and the mother properties of that person and of other persons in the table. Our objective is to enrich Excel with deductive logic capabilities for automatically computing values like the brother and sister properties. We would like to provide this capability while preserving the mechanics of interacting with spreadsheets to which users are already accustomed: for example, entering data in some cells of a spreadsheet (e.g., the Gender column) and watching that data propagate to other cells of the spreadsheet (e.g., the Brother column). Empowering Excel to achieve this goal requires some deductive capability and the means for specifying the deduction rules particular to a problem. It also requires the specification of how to send the values from some spreadsheet cells to the deductive logic machinery as well as how to return values deduced by the inference engine back into spreadsheet cells. The following sections describe how our design satisfies these requirements.

Deductive Logic Spreadsheet Architecture

At the highest level, the integration of a deductive logic engine into a spreadsheet requires that information from the spreadsheet be available as inputs to the logic engine and that the deductions (or inferences) it draws from those inputs can be inserted as derived values in the spreadsheet.

Since the deductive logic engine and the spreadsheet have very different notions of the primitive data on which they operate (respectively triples and cells), a mapping must be established between the two so that data can be extracted from and placed into the spreadsheet (See Figure 2).

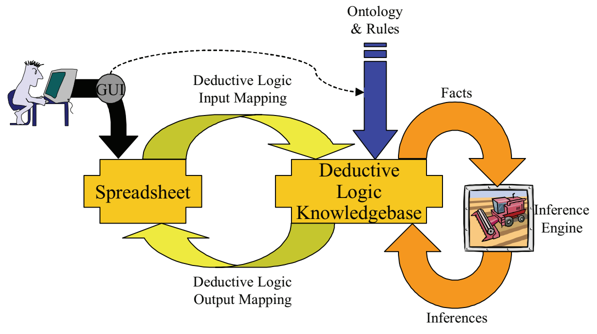

Ontology & Rules
GUI
Deductive Logic Input Mapping
Facts
Spreadsheet
Deductive Logic Knowledgebase
Inference Engine
Deductive Logic Output Mapping
Inferences

Figure 2: Deductive Logic Spreadsheet Architecture

The OWL deductive logic engine operates on <*subject*, *property*, *object*> triples indicating that "a *property* of *subject* is *object*." Thus, <A1, gender, male> or <A1, father, A8> respectively indicate that the gender of A1 is male and that the father of A1 is A8. The third element of a triple is the value contained in a spreadsheet cell (for inputs to the deductive logic engine) or the value placed in a spreadsheet cell (for outputs from the deductive logic engine). The subject and property for the triple are specified for each cell that participates as input for, or output from, the deductive logic engine and these declarations constitute the mapping between spreadsheet values and the deductive logic engine's knowledgebase.

These deductive logic mappings couple the inference engine into the spreadsheet by specifying where the facts the inference engine reasons across come from (through input mappings) and where the resulting inferences should be placed (through output mappings).

Their function is analogous to the formulas found in existing spreadsheets in that they specify where a result should be placed (in the cell containing the formula) and where the data needed to produce that result should be obtained (from the cells referenced by the formula).

However, formulas also specify the particular computation to be performed on the specified inputs to produce the result. For our deductive logic extension, the computation is defined by the set of rules loaded into the inference engine. These are specified independently of any particular result to be produced or of any particular input used to produce a result.

Thus the deductive logic coupling represents an aggregated data flow function ("Infer") from all of the input mapping cells to all of the output mapping cells and must be recomputed whenever any of those input mapping cells changes.

Deductive Logic Mapping Example

Figure 3 illustrates how the two way deductive logic mapping allows deductive logic to be integrated into a spreadsheet. The top layer, the Spreadsheet layer, is what the user sees. Underneath that and invisible to the user (at least without making a specific inspection request for which we will provide appropriate tools) is the Ontology and Rules layer that contains the ontology and the deductive logic knowledgebase. The spreadsheet shows a table of objects of type person with properties mother, child, gender and brother. Three person objects are shown in the table with ID's John, Jane and Jim respectively. The cells for the mother, child, and gender properties contain input mappings. The figure shows how the values contained in these cells are mapped onto triples (facts) in the deductive logic knowledgebase. The cells for the brother property contain an output mapping. The figure shows that the value to be mapped into the cell from the deductive logic knowledgebase is derived from the consequent of the Brother Rule that has three antecedents. We see that as the values from the mother, child, and gender property cells are mapped onto triples in the deductive logic knowledgebase, the triples are matched to the antecedents of the Brother Rule. Once the match of all antecedents is complete, the rule fires resulting in the assertion of the triple (John,Brother,Jim) into the knowledgebase and the transfer of the value Jim to the corresponding output cell in the spreadsheet.

Spreadsheet / knowledgebase synchronization

The following procedure describes how the spreadsheet is kept synchronized with the deductive logic engine. For each spreadsheet cell defined as a deductive logic input by the mapping, the value in that cell (either input directly by the user or computed by some spreadsheet formula) is converted to a triple by combining it with its declared subject and property and asserted in the deductive logic knowledgebase. When the spreadsheet is first opened, this "Deductive Logic Input" initialization is performed as a batch operation on each spreadsheet cell declared to be a deductive logic input. Thereafter, the "Deductive Logic Input" update is performed whenever there is a change in the value of an input cell. Once the input operation is complete, the triples that have the subject and property declared in the mapping for each output cell are retrieved from the deductive logic knowledgebase and its third element (the object of the triple) is extracted and stored as a computed value in the spreadsheet cell.

This two way mapping (Figure 2) which extracts values from the spreadsheet cells for the deductive logic knowledgebase and then, after the deductive logic knowledgebase has quiesced, extracts values from the knowledgebase for the spreadsheet cells keeps the knowledgebase synchronized (and consistent with) the spreadsheet and allows the derived values from that knowledgebase to appear as computed values in the spreadsheet.

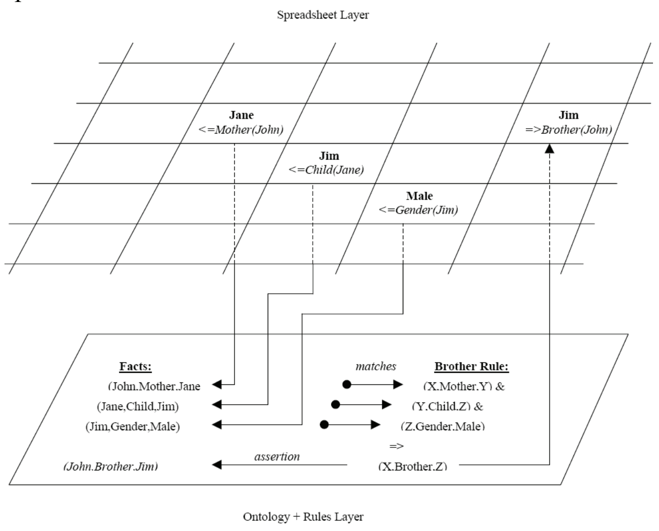


Figure 3: Mapping Data between Spreadsheet and Knowledgebase

The mapping into and out of the deductive logic database is triggered whenever the spreadsheet is changed. This can occur in two ways: when the user changes an input cell in the spreadsheet or when a new value is computed for an output cell. The latter occurs in response to the former and is triggered by the change in one of the inputs to the formula specified for that cell.

Excel provides events corresponding to these two types of cell update, allowing the mapping of data from the spreadsheet to the deductive logic database to be incremental (i.e. only the changed data is transferred to, or retracted from, the knowledgebase) and those transfers are synchronized with the changes to the spreadsheet.

Unfortunately, this incrementality doesn't exist in the opposite direction so that only changed derived information is propagated back to the spreadsheet. OWL doesn't define any mechanism for observing or responding to the changes in its derived knowledgebase. Our current implementation queries the knowledgebase with the output logic mapping rules at every change cycle. The effect of this policy in the performance of the system will depend on several factors, including the size and complexity of the knowledgebase, the number of output logic mapping rules in a spreadsheet, the frequency of spreadsheet updates, and the algorithms used by the underlying deductive logic engine (or the OWL reasoner). We acknowledge that our approach to deal with inference updates can compromise the scalability of the system and thus this issue will be subjected to further research.

As shown in the architecture diagram (Figure 2), an ontology and set of rules is also loaded into the deductive logic knowledgebase. The ontology defines the set of terms used to describe the types and properties used in the rules. OWL ontologies may also include an initial set of facts (triples) to be incorporated in the knowledgebase.

Once an ontology and its associated inference rules have been loaded, the deductive logic knowledgebase operates autonomously deriving inferences from the set of explicitly asserted facts. As new facts are asserted or removed it derives (a possibly empty set of) additional inferences or retracts (a possibly empty set of) previously inferred facts. Thus the set of inferred facts is synchronized with the explicitly asserted facts in the knowledgebase, ensuring that the update mapping (i.e. data going to the spreadsheet) is consistent with the input mapping (i.e. data coming from the spreadsheet).

The Ontology selection control has been added to the Excel GUI to allow the user specify the ontology and associated rules to be used by the deductive logic engine. This selection is attached to the spreadsheet so that when it is saved and subsequently reopened this choice will be remembered and used to initialize the knowledgebase in the reopened spreadsheet. This is currently the only means by which the user can specify the deductive logic terms and rules to be used in spreadsheet and the deductive logic knowledgebase.

Defining the Deductive Logic Mapping

Deductive Logic Mappings are definable and viewable by the user through the Mapping Rule control in the Deductive Logic toolbar added to the Excel GUI. This control operates much like Excel's Formula control which displays the contents of the currently selected spreadsheet cell when that cell has data entered by the user or the formula used for computing the contents of that cell otherwise. Our Mapping control similarly displays the mapping function for the currently selected spreadsheet cell if one is defined or nothing if no mapping function exists for that cell.

Input mappings are entered and displayed as <=Property(Subject) and output mappings are entered and displayed as =>Property(Subject) where Property is the name of the property in the mapping triple for the cell and Subject is either a literal reference for the subject of that mapping triple, a reference to a cell containing such a literal reference for the subject of that mapping triple, or a reference to a cell range that corresponds to the definition of the object instance to be used as the subject of that mapping triple.

While the Mapping control makes it very simple for users to define the input or output mapping for an individual spreadsheet cell, having to do so for all the cells that participate in the deductive logic mapping would nevertheless in the aggregate be quite burdensome. Fortunately, many of the cells that contain a deductive logic mapping are part of a table whose rows represent objects of a specific type and each column represents a property of the type of object contained in the table (like the example table in Figure 1). This conventional way of representing data about objects is exploited to provide an aggregated mechanism to specify deductive logic mappings. In this alternative model mappings are entered and displayed only in the cells corresponding to the column headings of the objects table. Input mappings are entered as <=Property(SubjectType) and output mappings are entered as =>Property(SubjectType). All column mappings of a table should agree in the subject type. We call this kind of mapping definitions *aggregated mapping rules*.

Mappings declared in this way work as follows: 1) Each table row is equated to an instance of type SubjectType and generates a triple <id, hasType, SubjectType> where id is an identifier generated automatically by the Deductive Logic Mapping Manager and SubjectType is the type declared in the mapping specification associated to the table headings, and 2) each cell C in the table is mapped to a triple <S, P, O> where S is the instance represented by the row of C, P is the property declared in the mapping of the column of C, and O is the content of C.

To automate the creation of tables that define aggregated mappings the Deductive Logic toolbar includes a control that lets users select any type defined in the ontology to start a table of instances of that type in the spreadsheet. This control causes a table to be inserted with column headings for each of the properties defined in the ontology for the type the user selected. Columns can be reordered (through cut and paste operations) and any unneeded properties (i.e. columns) can be deleted. Each column comes with an associated property mapping specification. If that property is referred by a triple included in the consequent of deductive rule that column will define an output mapping by default. All other properties will define an input mapping. The user is allowed to override this default mapping specification.

Sometimes it is convenient for the columns of a table to describe not only the properties of the individuals represented by its rows but also the properties of other individuals that relate to them. For example, a table could describe the father of a person and the father's age in the same row. To allow this kind of descriptions we are currently working on extending the aggregated mapping scheme described above to include also mapping rules where the subject of a triple is the content of another column in the same row. Such a mapping scheme will enable the description of properties of properties of the individuals. This feature is often referred to as *drill-down*.

Handling Typed Objects in Spreadsheets

Beyond the triple versus cell differences between deductive logic and spreadsheets described in previous sections, a second major difference is that the Subject and Object elements of a triple can be a reference to an instance of a typed object (e.g., a person) while spreadsheets only support literal values (such as number or string). These typed object references introduce a new set of operations such as how they are rendered in the spreadsheet cells.

In effect, typed object instances normally appear in spreadsheets as rows within tables. For example, each row in a table describing persons can be said to define an instance of type Person. Thus, within the spreadsheet it is natural to equate an object instance with its defining row.

We find it convenient to represent references to instances of typed objects as hyperlinks. A hyperlink in Excel includes two pieces of information: 1) a visible string that describes the linked object and 2) an invisible string that refers to the linked object. Thus a reference to an object instance is represented as a hyperlink displaying a mnemonic that identifies the object instance (such as the name of a company or first and last name of a person) and a reference to the named range that corresponds to the row in the table that defines that instance. When the user clicks on the cell containing the hyperlink the cells in the named range pointed by the hyperlink are selected (this highlights those cells and moves the spreadsheet view to show those cells). As all data exchanged between the spreadsheet and the deductive logic knowledgebase occurs through the deductive logic mappings, the mappings will translate these hyperlinks into an object reference that can be used by the deductive logic system and translate these object references back into the corresponding hyperlink for use within the spreadsheet.

We have provided a simple mechanism that allows the user to specify how these hyperlinks should be mnemonically displayed. The mnemonic is defined by marking (with an asterisk) the mapping formula specifications of one or more columns in the table that defines the typed object instances. The mnemonic is then automatically formed for each object instance in the table by concatenating the column values corresponding to the marked columns. For example, a table containing instances of type Person might have columns labeled FirstName and LastName (in that order). When a cell in the FirstName column is selected, the formula <=FirstName(Person) appears in the Mapping control on the Deductive Logic toolbar. The user then adds an asterisk resulting in the formula <= FirstName(Person)*. The user repeats this process for the LastName column. If the user creates a new Person instance (by adding a new row to the table) and enters John in the FirstName column and Smith in the LastName column, the mnemonic John Smith will be automatically created and assigned to that Person instance.

Object instance references (i.e., hyperlinks) can be created and stored into spreadsheet cells through the Insert Instance Hyperlink control on the Deductive Logic toolbar. The Insert Instance Hyperlink control displays a menu with all the instances created in the spreadsheet and the user selects the one that he wants to store in the current cell. For example, suppose the table containing instances of type Person has a column labeled Father and that the user has created a Person instance whose FirstName is Mary and whose LastName is Smith. If the user wants to specify that Mary Smith´s father is John Smith, he would select the Father cell in the row corresponding to Mary Smith and select the John Smith hyperlink from the Insert Instance Hyperlink control menu. The hyperlink John Smith will then appear in the corresponding cell. Hyperlinks can also be copied from one cell to another through the normal copying mechanism provided by Excel.

Handling Multi-Valued Properties in Spreadsheets

Deductive Logic objects can have multiple values for a given property: a person can have multiple brothers and a company can have multiple divisions. Since spreadsheet cells can only hold a single value, the mapping established between spreadsheet cells and deductive logic triples must be expanded to account for these multiple attribute values.

Our current prototype is able to handle multi-valued properties only when the Deductive Logic Mapping is aggregately specified on the column headings (described in Section *Defining the Deductive Logic Mapping* above). In this kind of mapping subjects are represented as rows and properties are represented as columns. A multivalued property is represented by inserting below the subject row as many detail rows as required to allocate all the values of the multivalued property. Detail rows are blank except for the columns representing the multivalued properties. As with single valued properties these values can be either a data value or an object reference.

These additional rows are defined within the spreadsheet as detail rows (using Excel's Outline capability) and are normally displayed collapsed so that only the main row containing the subject (and all its single valued attributes) is visible.

Using the Outline control provided (a + sign) at the left edge of the row, the user can expand the detail rows to make them visible (thus displaying the N-1 values for the multi-valued properties that didn't fit into the main row). The insertion and deletion of the necessary detail rows is automatic.

One limitation of the current scheme for automatically generating detail rows is that the added rows only contain values for the columns that correspond to the multivalued properties. Very often Excel tables contain columns whose content is derived from other columns through Excel formulae. For example, in our table containing instances of type Person we could have an Age column whose content is obtained by subtracting a DateOfBirth column from the current date. In Excel a derived columns is specified on a cell by cell basis by replicating a formula on every cell (or row) contained in that column. A very useful extension to our detail generation mechanism described above would be to transfer to the added rows the formulae that depended on the multivalued properties. Such a mechanism should be intelligent enough to adjust the references to single valued properties and other cells contained in the main row that were not replicated to the detail rows. We plan to enhance our current detail generation mechanism to provide this capability.

Prototype Implementation

Our design implementation (Figure 4) includes a clean separation between the Deductive Logic Mapping Manager front-end and the Logic Server back-end that allows us to change the logic engine with minimal effort. We accomplish this by providing a logic server with a well-defined interface to the Deductive Logic Mapping Manager that in turn has access to a wrapped logic engine of choice. One advantage of this design is that it allows the implementation language of the Mapping Manager to be independent of that of the logic server and engine. For example, the Mapping Manager of the prototype is implemented as an Excel COM add-in using Microsoft Visual Basic .Net while the logic server and logic engine wrapper are implemented in Java. The logic server implements a standard web service interface to support the interoperation between the .Net and the java realms.

Logic Server

The Logic Server manages the logic processing of the loaded ontologies, rules and facts (i.e., triples) represented in the spreadsheet. Through its interface it provides several services including:

- Loading ontologies and rules.
- Accessing ontology elements (e.g., all defined types as well as the properties of each type) that are used by the Mapping Manager to support the definition of Deductive Logic Mappings.
- The interchange of triples with the Mapping Manager.

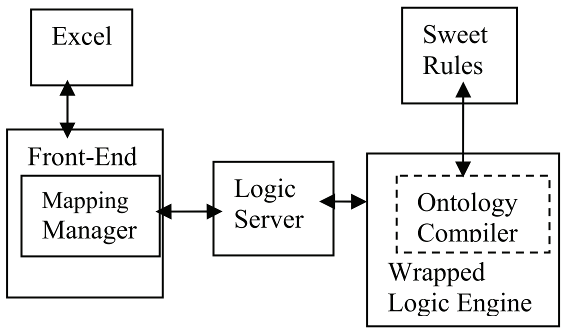


Figure 4 Functional Modules

The Deductive Logic Engine has to support OWL (McGuinness and van Harmelen 2004). To gain in expressiveness we opted for logic engines that support rules like SWRL (Horrocks et all 2004) in addition to OWL. However, this requirement is not strictly necessary. The current version of the system doesn't support the editing of ontology concepts or rules but we plan to add this capability in the future.

At the time of the writing of this article only two logic engines that support OWL + rules were available to us: SweetRules (SweetRules) and KAON2 (KAON2). We implemented a version of the logic server for each one of them. The first one was based on SweetRules and the second one was based on KAON2.

In the SweetRules version, the logic server interface also provides access to some specific SweetRules services like the ontology/rule base compiler. The compiler interface allows the user to specify an OWL ontology and a corresponding rule base written in the Semantic Web rule language RuleML (RuleML)[1]. The compiler, using the SweetRules (SweetRules) framework, translates and combines the OWL ontology and RuleML rule base into a single knowledgebase file of the appropriate type (e.g., Jess) which is then loaded into an instance of the rule engine and is ready for use. The SweetRules framework is being developed by the SweetRules group at the MIT Sloan School. The compilation process is completely automated using the application programmers interfaces (APIs) provided by SweetRules. Using SweetRules functionality an OWL file is first translated into RuleML. The resulting RuleML file is merged with the RuleML rule base into a single RuleML file that is translated into a complete knowledgebase file like Jess or Prolog. The resulting knowledgebase file is loaded into an instance of the rule engine that is then ready to reason over facts that we assert.

This compiled knowledgebase file is saved. The next time the user opens the Excel spreadsheet he or she can decide to load the already compiled ontology/rule base. Our design makes it straightforward to change, replace, or even remove the compiler with no impact on the spreadsheet manager or the rule engine. For example, it would be a simple matter to produce a distribution version of the system that did not contain the compiler. The users would only be allowed to load precompiled ontology/rule bases.

Because the knowledgebase has been converted to some rule engine specific language (e.g. Jess) the logic engine wrapper needs to translates OWL facts from the spreadsheet into rule engine specific facts, assert them into the rule engine, run the engine over the asserted facts, retrieve the output facts, and translate them back into OWL facts for return to the spreadsheet.

The KAON2 based version of the logic server is able to load ontologies with embedded SWRL rules. KAON2 is an infrastructure for managing OWL-DL, SWRL, and F-Logic ontologies. It was produced by the Information Process Engineering (IPE) at the Research Center for Information Technologies (FZI), the Institute of Applied Informatics and Formal Description Methods (AIFB) at the University of Karlsruhe, and the Information Management Group (IMG) at the University of Manchester.

Deductive Logic Mapping Manager Front-End

The Deductive Logic Mapping Manager front-end controls the transfer of data between the spreadsheet and the deductive logic server. It also provides a specific GUI to handle the authoring of Deductive Logic Mapping declarations and other related functions. Some of the important features of the Deductive Logic Mapping Manager front-end include:

- The ability to automatically generate typed object tables based on information from the ontology.
- The ability to save a spreadsheet including the Deductive Logic Mapping specifications in a persistent form that can be reloaded the next time the user opens the application.
- Integration with the logic server back-end. The logic server accepts facts in the form of triples and can respond to triple queries.
- Interpretation of a generic Typed Object Table description. Each typed object table describes instances belonging to one ontology class: the rows correspond to class instances, and the columns correspond to its property values. The content of this table is transformed into OWL triples that are transferred back and forth between the Excel worksheet and the logic server as the user make changes in the Excel worksheet and new facts are inferred by the logic engine.
- Implementation of instance references as hyperlinks to the table rows that define those instances.
- Implementation of multi-valued properties as detail rows in tables.

Related Work

There have been various attempts to use spreadsheets as front ends to deductive systems. Interestingly,

[1] We used RuleML because SweetRules did not include support for SWRL in the context we needed.

Teknowledge (where two out of the three authors of this paper are currently employed) itself accounts for two of these. Our first effort was over 20 years ago when we tried to construct a symbolic spreadsheet using Turbo Pascal to build a spreadsheet front end to Prolog. Our second effort was in the early 90's where, working together with FMC, we embedded our reasoning tool M.4 into Excel in order to implement a version of Michael Porter's Competitive Strategy techniques. Kriwaszek (1987) tried to incorporate a limited form of interactive logic into a spreadsheet. It made use of a call to an external prolog workspace. Spenke and Beilken (1989) used the idea of programming-by-example to create user-defined functions and logic rules in a uniform manner without the need for a new formal language. An interesting recent commercial attempt by Amzi! (the company that produces Amzi! Prolog) to enhance Microsoft Excel with deductive logic capabilities is their ARulesXL product.

Future Work

We have demonstrated the feasibility of extending a traditional spreadsheet with deductive logic capabilities. A major direction of future work will be to make these capabilities accessible to the end-user with a minimum of effort. For example, the spreadsheet should be able to automatically recognize the existence of tables of typed objects within the spreadsheet and define the deductive logic mappings needed to integrate that typed data into the deductive logic knowledgebase. This Typed-Object-Table-Recognition facility would utilize and extend Excel's ability to infer data entry forms from "lists." Taking this one step further, we would extend the Typed-Object-Table-Recognition facility to work even in the absence of a loaded OWL ontology. It would infer the ontology implied by the structure of the table and transfer this ontology to the deductive logic system in addition to defining the deductive logic mappings that couple the data in this table with the deductive logic knowledgebase. Together with the ability to dynamically define deductive rules through the GUI, this would allow spreadsheet users to utilize the deductive logic system entirely within Excel – without any reliance on external resources (such as an OWL ontology). The implied ontology, and any dynamically defined deduction rules would be saved with the spreadsheet so that it is totally self-contained when reloaded. An additional useful extension to our current design would be to allow mapping rules to be used as multi-directional constraints, i.e., in both input and output modes simultaneously.

Conclusion

It has been a long time goal of industrial artificial intelligence (AI) developers to make AI technology readily accessible to a wide audience. Success in this regard has been limited up to now. However, we believe that our effort shows great promise in helping to fulfill that goal by making deductive logic part of the everyday problem solving toolkit of the large and ever expanding spreadsheet end-user programming community.

Acknowledgments

Our thanks to all anonymous reviwers. This work was supported by DARPA under agreement W31P4Q-05-C-R069.

References

Boehm, B.; Horowitz, E.; Madachy, R.; Reifer, D.; Clark, B. K.; Steece, B.; Winsor Brown, A.; Chulani, S.; and Abts, C. 2000, *Software Cost Estimation with COCOMO II*. Prentice Hall PTR.

Horrocks, I.; Patel-Schneider, P. F.; Boley, H.; Tabet, S.; Grosof, B.; Dean, M. 2004. SWRL: A Semantic Web Rule Language Combining OWL and RuleML. W3C Member Submission. At http://www.w3.org/Submission/SWRL/

KAON2 -- Ontology Management for the Semantic Web Project homepage. At http://kaon2.semanticweb.org/

Kriwaszek, F. 1987. LogiCalc – A PROLOG Spreadsheet. in D. Michie and J. Hayes, *Machine Intelligence* 11, 1987.

McGuinness, D. L. and van Harmelen, F (ed). 2004. OWL Web Ontology Language Overview. W3C Recommendation 10 February 2004. At http://www.w3.org/TR/2004/REC-owl-features-20040210/

Nardi, B. 1993. A Small Matter of Programming: Perspectives on End User Computing. Cambridge, Mass: The MIT Press.

Nuñez, F. 2000. An Extended Spreadsheet Paradigm for Data Visualization Systems and its Implementation, Masters Thesis, Dept. of Computer Science, University

of Cape Town.

RuleML Hompeage. At http://www.ruleml.org/

Spenke, M. and Beilken, C. 1989. A Spreadsheet Interface for Logic Programming.. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems: Wings for the Mind, p.75-80.

SweetRules Project Home Page. At http://sweetrules.projects.semwebcentral.org