On enforcing relational constraints in MatBase

Christian Mancas,Valentina Dorobantu

ABSTRACT

MatBase is a very powerful database management system based not only on the relational data model, but also on the elementary mathematical, entity-relationship, and Datalog logic ones. This paper briefly introduces MatBase and the elementary mathematical data model, after which is focused on the five relational constraint types and their enforcement in relational database management systems, as well as in MatBase.

Keywords: relational data model, domain constraint, not null constraint, unique key constraint, referential integrity constraint, tuple constraint, elementary mathematical data model, MatBase.

Classification: H.2.6 H.2

Language: English

London Journals Press

LJP Copyright ID: 206059
ISBN 10: 1537631683
ISBN 13: 978-1537631684

London Journal of Research in Computer Science and Technology

1 6 U K

Volume 17 | Issue 1 | Compilation 1.0

On Enforcing Relational Constraints in *MatBase*

Valentina Dorobantu[α] & Christian Mancas[σ]

I. ABSTRACT

MatBase is a very powerful database management system based not only on the relational data model, but also on the elementary mathematical, entity-relationship, and Datalog logic ones. This paper briefly introduces MatBase and the elementary mathematical data model, after which is focused on the five relational constraint types and their enforcement in relational database management systems, as well as in MatBase.

Keywords: relational data model, domain constraint, not null constraint, unique key constraint, referential integrity constraint, tuple constraint, elementary mathematical data model, MatBase.

Author α σ: Mathematics and Computer Science Dept., Ovidius University, Constanta, Romania.

II. INTRODUCTION

This first section briefly presents *MatBase*, the Elementary Mathematical Data Model (EMDM), the five relational constraint types ((co-) domain/ range, not null, unique keys, referential integrity/ foreign keys, and tuple/check) and their enforcement in the Relational Database Management Systems (RDBMSes), as well as related and further work.

The second section is the core of the paper, introducing the ways in which *MatBase* is enforcing and extending the five relational constraint types. The paper ends with conclusions and references.

2.1 MatBase

MatBase [1-5] is a powerful prototype multi-model, multi-user, and multi-language knowledge and data (KD) base management system (KDBMS) that currently has two versions: one in MS Access 2015 and the other in MS *SQL Server 2015* and C#. *MatBase* provides four data models: the (Elementary) Mathematical ((E) MDM) [1, 6-11], the Relational (RDM) [10, 12, 13], the Entity- Relationship (E-RDM) [10, 13, 14], and the *Datalog* ¬ [11, 13] based Logic (DL¬DM) ones.

Its main and most powerful interface is the (E) MDM one (which includes the DL¬DM one), where users manage sets, functions, and constraints that *MatBase* is automatically translating into corresponding tables, columns, and constraints (but users may also manage Datalog ¬inference rules and programs). For the relational constraints (see section 2.3), *MatBase* mainly uses the corresponding ones provided by MS Access and SQL Server. For the non-relational ones, it automatically generates embedded and/or extended SQL code into forms automatically built upon the corresponding tables.

MatBase also provides an RDM interface, where, dually, users manage tables, columns, and (only relational) constraints and it automatically generates corresponding sets, functions, and constraints.

Finally, *MatBase* also provides an E-RDM interface, where users manage E-R diagrams (E-RDs) and it automatically generates corresponding sets, functions, and constraints, as well as corresponding tables, columns, and constraints. Dually, users may ask in both (E)MDM and RDM interfaces generation of E-RDs for any set and its related ones on *n* nodes

distance from it, *n* being a natural parameter, or the full db scheme.

2.2 The Elementary Mathematical Data Model

(E)MDM schemes are quadruples made out of a finite nonempty collection of sets *S* partially ordered by inclusion, a finite nonempty set of mappings *M* defined on and taking values from sets of *S*, a finite nonempty set of constraints *C*, and a finite set of Datalog¬ programs *P* associated to the sets of *S* and mappings of *M*. Conventional database (db) schemas are triples <*S*, *M*, *C* >); when *P* is non- empty, the corresponding db is a deterministic deductive one, so a knowledge base.

S is partitioned into the following four blocks: object, value, system, and computed sets. *Object* ones are partitioned into *entity* (i.e. atomic) and *relationship* (i.e. non-functional math relations immune to their domain permutations). *Value* ones are subsets of (programming) data types. *System sets* include at least the data types, the empty set, and a distinguished countable set NULLS of *null values*. *Computed sets* are obtained from all other types of sets by using semi-naïve sets, functions, and relations algebra operators (e.g. union, difference, intersection, etc.).

All mappings in *M* are defined either on object sets or on computed sets based only on object ones. *M* is partitioned into the following four blocks: attributes, structural functions, system, and computed mappings. *Attributes* are taking values from value sets, while *structural functions* from object ones, both possibly combined with NULLS. *System mappings* include canonical projections, injections, and unity mappings. *Computed mappings* are obtained from all other types of mappings by using semi-naïve sets, functions, and relations algebra operators (e.g. composition, (Cartesian) product, etc.).

C is partitioned into the following four blocks: set, dyadic relation, mapping, and object constraints. (E)MDM has a rich panoply of constraint types: currently, it has 56, out of which 30 fundamental and 26 derived ones (see everyday life examples for all of them in [10, 11]). The main reason behind its introduction is that neither RDM nor E-RDM constraint types are not at all enough to guarantee db instances plausibility. For example, even such a simple constraint as "the capital city of any country should belong to that country" is not expressible in either RDM or E-RDM, whereas in (E)MDM it is, either as *Country* ^{o} *CapitalCity reflexive or, equivalently, as Country* ^{o} *Capital City* = 1_{CITIES}.

2.3 The 5 Relational Constraint Types and Their Enforcement in RDBMSes

RDM provides five types of constraints (i.e. closed first order logic calculus (FOLC) with equality formulae) that are embedded in all Relational Database Management Systems (RDBMSes) for enforcing these business rules, namely: domain (range), not null (function total definition), keys (uniqueness), typed inclusion (foreign keys, referential integrity), and tuple (check).

Tables storing fundamental data should make heavy use of them. Those storing temporary data should not have constraints (except for debugging purposes), as their enforcement costs both disk space and, especially, processing time. In what follows we consider only fundamental data tables.

Domain constraints restrict for columns the corresponding data types to some plausible subset. For example, to a function *BirthDate* : *EMPLOYEES* → [1/1/1900, *SysDate*() – 18 years] corresponds a DATE column *BirthDate* to which you have to add the domain constraint "between '1/1/1900' and SysDate() – 18 years". Obviously, without it users might store even highly implausible data, as DATE starts, for example, in MS Access with 1/1/100, in MS *SQL* Server and IBM DB/2 with 1/1/1, in Oracle with 1/1/4712 BC, and they all end on 31/12/9999: why letting users (and it doesn't matter whether by mistake or on purpose, to test your db design skills) entering data on employees born some 2000 years ago or that will be born only some 7000 years from now?

Note that, mathematically, this is, in fact, a co-domain definition, while in RDBMSes, as the corresponding SQL syntax is "CHECK BirthDate between '1/1/1900' and SysDate() – 18 years", it is generally called a check constraint.

A fundamental best practice rule is that for any column of type numeric (including DATE, which is also stored numerically) you should add a corresponding domain constraint. String character columns might also need such constraints. For example, to column *Sex* of table *EMPLOYEES* you should add the constraint "CHECK Sex in ('F', 'M')".

Not null constraints are, in fact, a particularization of the *existence constraints:* given any two columns f and g of a same table T, such a constraint is denoted by $f \vdash g$ and has the meaning "whenever f is not null, g should be not null too". For example, in a library db, in table *ITEMS* you should enforce both *BorrowDate* $\vdash$ *Borrower* (whenever an item was borrowed, you should also know to whom) and *Borrower* $\vdash$ *BorrowDate* (whenever somebody borrowed an item, you should also know when).

In the particular case when f is void, according to FOLC, the meaning of such a constraint is that g should never accept null values and this is why they are called *not null constraints*. Unfortunately, no RDBMS, except for *MatBase*, is providing existence, but only not null constraints. In SQL, they are enforced by declaring the corresponding column as NOT NULL.

A fundamental best practice rule is that any table should have at least one column, except for the surrogate key one, not accepting nulls.

Beware of notational confusions: Oracle, for example, considers NOT NULL constraints as being of type CHECK too.

Mathematically, functions are totally defined, so, for example, the correct definition of *CapitalCity*, if knowing capitals of all countries should not be compulsory in a db, would be *CapitalCity* : *COUNTRIES* → *CITIES* ∪ NULLS. In fact, as in dbs the vast majority of columns accept nulls, (E)MDM uses the dual notation that never explicitly uses NULLS, considering totality an optional constraint. For example, *CapitalCity* : *COUNTRIES* → *CITIES* and *CountryName* : *COUNTRIES* → ASCII(255), *total* mean that *CapitalCity* accepts nulls, whereas *CountryName* does not.

Key constraints are rejecting any attempt to duplicate data on corresponding columns of a table. For example, as there may not be two countries with a same name in the world, *CountryName* should be declared as UNIQUE in table *COUNTRIES*; similarly, as there may not be two states of a same country having same names the pair (*StateName*, *Country*) should also be declared as UNIQUE in table *STATES*.

RDBMSes enforce key constraints with unique indexes. Unfortunately, besides arity limitations (e.g. in current versions of MS Access a key may contain at most 10 columns, 16 in MS SQL Server, 32 in Oracle, 64 in IBM DB/2), RDBMSes do not accept in keys columns of some data types (e.g. long texts, OLE etc.) and some of them (e.g. MS SQL Server) do not accept columns having more than one null value.

Mathematically, a key is either a one-to-one function or a minimally one-to-one function product. A not minimally one-to-one function product is called *superkey* both in RDM and (E)MDM. For example, (*Population*, *StateName*, *Country*) is a superkey in *STATES*. Unfortunately (as not only conceptually superkeys are not minimal, but they also waste unneeded both disk space and enforcement time), all RDBMSes, again except for *MatBase*, accept enforcing both keys and superkeys.

Tables may have a *primary key*, which is a key not accepting nulls. Most RDBMSes provide a surrogate key data type (called COUNTER, GENERATED, AUTONUMBER etc.), i.e. fixed point numeric, generally with auto-numbering,

having no other semantics but unique row identification.

Other fundamental best practice rules are:

- Every table should have a surrogate primary key, with auto-numbering whenever it is not also a foreign key (which is needed for tables corresponding to subsets, e.g. for *DRIVERS* ⊆ *EMPLOYEES*);
- Every table should have all keys existing in the corresponding actual subuniverse.

A *referential integrity constraint* restricts the values that may be taken by a column or set of columns to those taken by another column or set of columns, respectively. For example, column *Country* from table *STATES* should only take values from those stored by the surrogate primary key column *ID* from table *COUNTRIES*. Columns to which such constraints are associated are called *foreign keys*; for example, *Country* above is a foreign key.

The SQL syntax for such constraints, exemplified for *Country*, is "FOREIGN KEY (Country) REFERENCES COUNTRIES", where whenever the foreign key references the primary key the latter needs not be explicitly mentioned.
Mathematically, if f references g, then $Im(f) \subseteq Im(g)$, where the image of a function, denoted *Im*, is the set of the values it is taking.

Another fundamental best practice rule is that every foreign key references a surrogate primary key.

Finally, *tuple constraints* have no generally accepted definition; generalizing corresponding RDBMS implementations, they are constraints relating several columns of a same table, e.g., in *EMPLOYEES*, *BirthDate* ≤ *HireDate* ≤ *Passed Away Date* (written by notational abuse as formulas of a propositional calculus, but being in fact FOLC closed formulas with only one variable occurrence universally quantified; e.g. the one above is a shortcut for $(\forall x \in EMPLOYEES)$ $(BirthDate(x) \leq HireDate(x) \leq PassedAway\ Date(x))$).

2.4 Related and Further Work

Oracle constraint enforcement is presented in Chapter 6 of [15]. Corresponding data for the MS SQL Server can be found in [16]. For MS Access, [17] presents the SQL CONSTRAINT clause, while [18] the ADO API.

A planned improvement of *MatBase* is to allow definition of existence constraints for computed functions too, as well as modifying existence constraints.

Last, but not least, further work will be done for providing *MatBase* versions for Oracle, IBM DB2, *MySQL*, and *PostgreSQL* in a next step.

III. ENFORCING RELATIONAL CONSTRAINTS IN *MATBASE*

MatBase not only enforces all five relational constraint types, but it does so more elegantly and powerful than any other DBMS. As most of the systems, it enforces constraints, regardless of type, only if the current db instance satisfies them.

3.1 Domain constraints

Both *MatBase* versions establish data types based on the corresponding co-domains. For example, both NAT (the naturals) and INT (the integers) are implemented as integers (unsigned for NAT). Corresponding subtypes are chosen based on the *DomCnstr* (Domain Constraint) values, if any. For example, a co-domain NAT with *DomCnstr* = 2 (i.e. NAT(2)) is implemented as the subtype Byte, as it can only store naturals less than 100. When no such value is specified, the largest corresponding subtype is selected; for example, INT with a null value in *DomCnstr* is implemented in Access as Long Integer.

In its MS Access version, as there is no SQL CHECK clause, *MatBase* uses DAO, just like Access does when enforcing its Graphic User Interface (GUI) Validation Rule for columns. In its MS SQL Server version, *MatBase* uses the SQL CHECK clause.

In its GUI, namely in its *FUNCTIONS* form, you can specify domain constraints other than with *DomCnstr* by using either *MinValue* and/or *MaxValue* or user-defined value sets as codomains. Obviously, if, for example *MinValue* is *min* and *MaxValue* is *max*, it will enforce a BETWEEN *min* and *max* validation rule or CHECK constraint, respectively; if only *min* is not null whereas *max* is null a $\geq$ *min* one is enforced. You can define a value set by declaring it as a subset of a data type; for example, you can define *RAINBOW_COLORS* $\subseteq$ ASCII(6), which will create a corresponding table, and then fill it with data (e.g. 'red', 'orange', 'yellow', 'green', 'blue', 'indigo', 'violet'). If the corresponding set is declared as static, then in its MS SQL Server version a corresponding CHECK IN (...) constraint is enforced, whereas in its MS Access one a combo-box filled with corresponding field values is declared; if not, a referential integrity constraint is enforced between the corresponding column and the surrogate primary key with auto-numbering of that value set.

3.2 Existence constraints

MatBase enforces both the particular NOT NULL constraints and the general existence ones. NOT NULL is enforced in both versions with the corresponding SQL clause. According to the math definition, the *FUNCTIONS* form contains a *Total* column for it: when checked, NOT NULL is enforced for the corresponding column. For object identifiers (to which table primary keys correspond), canonical Cartesian projections (the so-called *roles* of underlying sets in relationships, e.g. *Country* and *Neighbor* in *NEIGHBORS*), canonical surjections (e.g. *Represented By* : *PEOPLE* $\rightarrow$ *MPS*, where *MPS* $\subseteq$ *PEOPLE* is the set of the members of the Parliament) totality cannot be removed.

As the general existence constraints (EC) imply each two functions, *MatBase* GUI provides an *EXISTENCE CONSTRAINTS* form with three columns: the corresponding primary key (that is also a foreign key referencing the primary one of the *CONSTRAINTS* table), the left, and the right side functions. You can only add or remove ECs. When adding a new one, first of all, *MatBase* enforces the meta-constraints stating that both functions have to be defined on a same set, not being total (i.e. accepting nulls), and, for the moment, not being calculated. If they are met and the db instance satisfies it, then corresponding code (VBA or C#, respectively, with embedded SQL) is automatically generated into the standard update form associated to the common functions domain. When an EC is deleted, this code is automatically deleted too.

3.3 Uniqueness (key) constraints

The unique feature of *MatBase* when it comes to key constraints is its *Keys Discovery Assistant* (KDA) [19]. You can invoke this wizard set by set, for both relationship and entity type ones. For relationships, in a first step, only structural keys are considered, i.e. keys made out only of the corresponding canonical Cartesian projections. In a second step, just like for the entity type sets, all prime mappings defined on that set are considered. When it opens, the wizard presents you with all keys already declared for that set, if any, as well as with all other possible key candidates, if any. You can delete existing keys or/and declare new ones out of the existing candidates. After each such operation, the wizard re-computes the possible keys. Moreover, KDA does the following other tasks:

- It is never including superkeys among the candidates.
- It stops when the maximum possible number of keys is reached.
- It does not allow enforcement of keys whenever the current db instance violates them.
- Generates and apply corresponding ALTER TABLE DROP/ADD CONSTRAINT SQL DDL UNIQUE statements whenever possible (i.e. in MS Access, for example, when the key arity is maximum 10 and all involved column data types are accepted in keys, and there are at most 6 keys per table, whereas in MS SQL Server the maximum is 32, and, moreover,

none of the columns has more than one null value).

- For all other keys (i.e. of greater arity or/and containing not accepted column data types), in order to enforce them it automatically generates code (VBA or C#, respectively, with embedded SQL) in the corresponding set/ table standard update form.

3.4 Referential integrities (foreign key) constraints

For any set, *MatBase* adds a corresponding object identifier mapping (named *x*), i.e. a totally defined, one-to-one function defined on that set and taking integer values (for which, in the corresponding table, a surrogate primary key is added, with auto- numbering if the table does not correspond to a subset of another set). For any structural function, i.e. one defined on and taking values from object sets, it automatically adds a corresponding column in the table corresponding to the domain set and declares it as being a foreign key referencing the primary key of the table corresponding to the co-domain set (for auto-functions, obviously, this is the primary key of the same table). Enforcement/ dropping of referential integrities is done through generation and execution of corresponding SQL DDL CONSTRAINT clauses of type FOREIGN KEY.

Consequently, all foreign keys generated by *MatBase* are single-columned and integer type ones referencing primary keys. For example, for the function *State* : *CITIES* → *STATES*, *total* in table *CITIES* an integer column *State* not accepting nulls is added and declared as a foreign key referencing the primary one of table *STATES*.

3.5 Tuple (check) constraints

MatBase enforces/drops these constraints exactly like the RDBMSes on which is built upon, either by generating and executing corresponding ALTER TABLE ADD/DROP CONSTRAINT SQL DDL CHECK statements in its MS SQL Server version or by calling VBA + embedded SQL methods using ADO with corresponding parameters in its MS Access one (as there is no CHECK clause in Access' SQL).

For example, in its MS SQL Server version, for the constraint *StartDate* ≤ *EndDate* ≤ *StartDate* + 30 attached to a table *PROJECTS MatBase* generates and runs the SQL DDL statement ALTER TABLE PROJECTS ADD CONSTRAINT PROJECTS_C_1 CHECK EndDate BETWEEN StartDate AND StartDate + 30.

IV. CONCLUSION

MatBase enforces all five types of relational constraints in both its versions. Whenever this is possible, it does so by using the underlying MS engines (Access and SQL Server, respectively).
Moreover, *MatBase* has several unique features that no other DBMS has, out of which the main ones are the following:

- Also enforces general existence constraints, not only their particular NOT NULL case.
- Provides users with the facility to declare functions (and corresponding columns) as being nonprime (i.e. never able to take part in a unique key), so as to minimize the time needed to discover all existing keys.
- Provides a Keys Discovery Assistant, which guides users and enforces/drops keys and only keys (i.e. never superkeys) in discovering and enforcing all existing keys in the least time possible.
- Enforces keys that are not enforceable by using the underlying MS engines (i.e. in Access, for example, those of greater than accepted arity or/and containing columns not accepted in keys or/and all those that surpass the maximum of 6 keys per table) by automatic code generation.
- Automatically generates optimal primary keys (i.e. single-columned and integer type ones).
- Automatically generates optimal foreign keys (i.e. single-columned and integer type ones referencing primary keys).
- Enforces dozens of other constraint types, which are non-relational and exist only in the

Elementary Mathematical Data Model, through automatic code generation.

Further work will include allowing definition of existence constraints for computed functions too, as well as modifying existence constraints. Moreover, *MatBase* versions for Oracle, IBM DB2, MySQL, and PostgreSQL are planned too.

REFERENCES

1. Mancas, C. (1997) Conceptual data modeling. (in Romanian) Ph.D., Thesis: Politehnica University, Bucharest, Romania.
2. Mancas, C.; Dragomir S.; Crasovschi, L. (2003) On modeling First Order Predicate Calculus using the Elementary Mathematical Data Model in *MatBase* DBMS. In Proc. IASTED AI 2003 MIT Conf. on Applied Informatics, 1197-1202, Acta Press, Innsbruck, Austria.
3. Mancas, C.; Dragomir S. (2004) *MatBase* Datalog Subsystem Metacatalog Conceptual Design. In Proc. IASTED SEA 2004 MIT Conf. on Software Eng. and App., 34-41, Acta Press, Cambridge, MA.
4. Mancas, C.; Mancas, S. (2005) *MatBase* E-R Diagrams Subsystem Metacatalog Conceptual Design. In Proc. IASTED DBA 2005 Conf. on DB and App., 83-89, Acta Press, Innsbruck, Austria.
5. Mancas, C.; Mancas, S. (2006) *MatBase* Relational Import Subsystem. In Proc. IASTED DBA 2006 Conf. on DB and App., 123-128, Acta Press, Innsbruck, Austria.
6. Mancas, C. (1985) Introduction to a data model based on the elementary theory of sets, relations and functions (in Romanian). In Proc. of INFO IASI '85, 314-320, A.I.Cuza University, Iasi, Romania.
7. Mancas, C. (1990) A Deeper Insight into the Mathematical Data Model. Proc. 13th Intl. Seminar on DBMS, ISDBMS'90, 122-134, Mamaia, Romania.
8. Mancas, C. On Modeling Closed E-R Diagrams Using an Elementary Mathematical Data Model. Proc. 6th ADBIS 2002 Conf. on Adv. in DB and Inf. Syst., 165-174, Slovak Technology University Press, Bratislava, Slovakia, 2002.
9. Mancas, C. (2002) On Knowledge Representation Using an Elementary Mathematical Data Model. In Proc. IASTED IKS 2002 Conf. on Inf. and Knowledge Sharing, 206-211, Acta Press, St. Thomas, U.S. Virgin Islands, U.S.A.
10. Mancas, C. (2015) Conceptual Data Modeling and Database Design: A Completely Algorithmic Approach. Volume I: The Shortest Advisable Path. Waretown, NJ: Apple Academic Press / CRC Press / Francis & Taylor.
11. Mancas, C. (2017) Conceptual Data Modeling and Database Design: A Completely Algorithmic Approach. Volume II: Refinements for an Expert Path. Waretown, NJ: Apple Academic Press / CRC Press / Francis & Taylor (to be published).
12. Codd, E. F. (1970) A relational model for large shared data banks. CACM 13(6): 377-387.
13. Abiteboul, S.; Hull, R.; Vianu, V. (1995) Foundations of Databases; Addison-Wesley: Reading, MA.
14. Chen, P. P. (1976) The entity-relationship model: Toward a unified view of data. ACM TODS 1(1): 9-36.
15. Oracle Corp. (2005) Application Developer's Guide - Fundamentals. https://docs.oracle.com/cd/B19306_01/appdev.102/b14251.pdf
16. Microsoft Corp. (2016) SQL Server Constraints. https://technet.microsoft.com/en-us/library/ms1 89862(v=sql.105).aspx
17. Microsoft Corp. (2015) Access Constraint clause. https://msdn.microsoft.com/en-us/library/office/ff836971.aspx
18. Microsoft Corp. (2016) ADO API Reference. https://msdn.microsoft.com/en-us/library/ms678086(v=vs.85).aspx
19. Mancas, C. (2016) Algorithms for Database Keys Discovery Assistance. In Perspectives in Business Informatics Research, 322-338, LNCS 261, Springer International Publishing.

This page is intentionally left blank

London Journal Press Membership

For Authors, subscribers, Boards and organizations

London Journals Press membership is an elite community of scholars, researchers, scientists, professionals and institutions associated with all the major disciplines. London Journals Press memberships are for individuals, research institutions, and universities. Authors, subscribers, Editorial Board members, Advisory Board members, and organizations are all part of member network.

Read more and apply for membership here:
https://journalspress.com/journals/membership

For Authors

Author Membership provide access to scientific innovation, next generation tools, access to conferences/seminars /symposiums/webinars, networking opportunities, and privileged benefits.
Authors may submit research manuscript or paper without being an existing member of LJP. Once a non-member author submits a research paper he/she becomes a part of "Provisional Author Membership".

For Institutions

Society flourish when two institutions come together." Organizations, research institutes, and universities can join LJP Subscription membership or privileged "Fellow Membership" membership facilitating researchers to publish their work with us, become peer reviewers and join us on Advisory Board.

For Subscribers

Subscribe to distinguished STM (scientific, technical, and medical) publisher. Subscription membership is available for individuals universities and institutions (print & online). Subscribers can access journals from our libraries, published in different formats like Printed Hardcopy, Interactive PDFs, EPUBs, eBooks, indexable documents and the author managed dynamic live web page articles, LaTeX, PDFs etc.

GO GREEN AND HELP
SAVE THE ENVIRONMENT

JOURNAL AVAILABLE IN

PRINTED VERSION, INTERACTIVE PDFS, EPUBS, EBOOKS, INDEXABLE DOCUMENTS AND THE AUTHOR MANAGED DYNAMIC LIVE WEB PAGE ARTICLES, LATEX, PDFS, RESTRUCTURED TEXT, TEXTILE, HTML, DOCBOOK, MEDIAWIKI MARKUP, TWIKI MARKUP, OPML, EMACS ORG-MODE & OTHER

SCAN TO KNOW MORE

support@journalspress.com
www.journalspress.com

*THIS JOURNAL SUPPORT AUGMENTED REALITY APPS AND SOFTWARES