International Journal of Computers, Communications & Control
Vol. II (2007), No. 4, pp. 303-313

DOMINO: Trivalent Logic Semantics in Bivalent Syntax Clothes

Boldur E. Bărbat

Abstract: The paper describes a rather general software mechanism developed primarily for decision making in dynamic and uncertain environments (typical application: managing overbooking). DOMINO (*D*ecision-*O*riented *M*echanism for "*I*F" as *N*on-deterministic *O*perator) is meant to deal with undecidability due to any kind of future contingents. Its description here is self-contained but, since a validation is underway within a much broader undertaking involving agent-oriented software, to impair redundancy, several aspects explained in very recent papers are here abridged. In essence, DOMINO acts as an "IF" with enhanced semantics: it can answer "YES", "NO" or "UNDECIDABLE in the time span given" (it renders control to an exception handler). Despite its trivalent logic semantics, it respects the rigours of structural programming and the syntax of bivalent logic (it is programmed in plain C++ to be applicable to legacy systems too). As for most novel approaches, expectations are high, involving a less algorithmic, less probabilistic, less difficult to understand method to treat undecidability in dynamic and uncertain environments, where *postponing decisions* means *keeping open alternatives* (to react better to rapid environment changes).
Keywords: undecidability; open, heterogeneous, dynamic and uncertain environments (OHDUE); decision-making; trivalent logic semantics; agent-oriented software engineering.

1 Introduction

Despite the major changes taking place in the Internet and globalization era - and their increasing speed because of the geometrically increasing computing power (due to Moore's law, that is expected to hold at least other ten years) -, basic software mechanisms advanced much too slowly. As regards uncertain knowledge processing, the hindrance is obvious: available software tools are either hardly affordable (because of high complexity - both cognitive and structural) or rather ineffective (designed for other environments, applied to ill-defined problems or lacking expected functionality). For instance, the very concept of "uncertainty" was treated inadequately - regardless of its growing relevance for application domains (mainly where real-time decision-making is involved [14]), environments (even more dynamic and uncertain [19]), end-user expectations (requesting anthropocentric interfaces [4]), IT paradigms (predominantly "computing as interaction" [1]) and so on. The main weaknesses: a) insufficient theoretical rigour (undecidability is considered primarily atemporal - keeping its initial mathematical meaning); b) poor practical effectiveness (confusing "unknown" with "unknowable" and applying sophisticated prediction methods in inappropriate contexts); c) unfit apparatus (algorithmic approaches implying determinism, decidability, and bivalence). Since the first two issues are investigated in very recent papers [8] [6] [9] [7], to reduce redundancy and to keep focusing on the very mechanisms, here the approach is an explicitly software engineering one: computer science aspects are dealt with only at the beginning (what for new mechanisms?) and at the end (what are the expectations?). In particular, the paper presents in detail DOMINO, a mechanism developed primarily for decision making in dynamic and uncertain environments (a typical example for potential application area comes from the overbooking policy of carrier companies). The *D*ecision-*O*riented *M*echanism for "*I*F" as *N*on-deterministic *O*perator (DOMINO) is meant to deal with undecidability due to any kind of future contingents. As a rather general software mechanism, its description here is self-contained. However, since a validation is underway within a much broader undertaking involving agent-oriented software (including non-algorithmic approaches to treat uncertainty), several aspects explained in the papers mentioned above are here abridged or skipped

over. As a result, the paper is structured as follows: Section 2 shows the *rationale* (i.e., *premises and diagnosis*) for developing new methods to handle uncertainty in decision support systems (DSS). Section 3 *delimits the problem* in both meanings: firstly it restricts it (narrowing the scope to undecidability due to future contingents) and secondly it *defines* it (as software engineering task). Section 4 outlines the architecture, basically *in search of the third value semantics*. On this groundwork, Section 5 explains the structure, in search of bivalent syntax clothes. Since conclusions are prohibited for a bottom-up project, before its validation, Section 6 lists the expectations, ranged in three time horizons (in fact, they are first conclusions).

2 Rationale. Premises and Diagnosis

Since in this section the paper is to some extent also a position paper, for the sake of simplicity, the premises, opinions, criteria, motives, and their corollaries are not separated in conceptual categories but asserted clustered around two "attractor words": *premises and diagnosis*. Thus, consensus is not mandatory to assess the research results. In other words, the utility of DOMINO can be evaluated even when some of the claims made here are rejected. Most of the assertions below are valid in any modern IT context but DSS are explicitly referred to because "IF" is the basic tool (again in both senses: *essential* and *simple*) for decision-making. Likewise, the emphasis is on the main changes affecting the essence of "IF". To preserve both text autonomy and shortness, for topics dealt with recently, details should be found in the paper the quotation comes from.

Premises. They reflect the general context, representing a very simplified input vector:

- *Environment*. "Present-day IT environments, except for some irrelevant applications, are open and heterogeneous (the resources involved are unalike and their availability is not warranted), dynamic (the pace of exogenous and endogenous changes is high) and uncertain (both information and its processing rules are revisable, fuzzy, uncertain and intrinsically non-deterministic" [6].

- *System*. Except trivial applications, the system exposed to decision-making is "man-machine system [...], highly complex (multi-distributed - mainly in space, but also in time and organization), under real-time control (the subsystems act on each other - at least, communicate intensely), almost always online" [7].

- *User expectations*. "Since most decision makers are already familiar with Google, most available general information is either known or easy accessed; now they need acceptable good answer, but very fast and with incomplete or even uncertain information" [7]. "Intelligent system behaviour - whatever that could mean - becomes a crucial user expectation" [8].

- *Task*. Most aim at "managing situations"; main attributes are: "high complexity, multicriterial, Pareto optimality, approximate, online, parallel, critical response time, high risk" [7].

- *DSS Architectonics*. The software process (not package) is "vaguely specified, validated against end-user expectations" and has "two new fundamental design-space dimensions: Time and Uncertainty" [7].

- *IT infrastructure and paradigms*. "The IT infrastructure is sufficiently advanced (in both facts and trends: nanoelectronics, broadband communication, semantic web, multimodal interfaces, etc.) to allow anthropocentrism for most IT applications" [8]. The leading paradigm is "computing as interaction" [1] [19]. Second echelon paradigms (prevalent in modern artificial intelligence) are assessed through an "affordability filter" in [6]. As regards software engineering, the paradigm that becomes dominant is agent-orientation.

Diagnosis. The main weaknesses of current IT systems are investigated in [7], where the focus is on conventional modelling. Since DSS weaknesses are very similar, they are stated here adapted and abridged from [7]: they stem from inappropriate conceptualising, based on rigid, algorithmic (i.e., deterministic, almost sequential, "computational", and atemporal processing), meant for decision making as "step by step solving of arising sub-problems", not for decision making as "continuous process of dealing with unexpected, potentially risky, fast changing situations requesting immediate - albeit not optimal - response". Sectorial aspects are:

- *Poorly reflected* (or absent) temporal dimension. Limited parallelism (if any), ineffective multithreading, poor reactivity (scarce interrupt handling impairing proper reaction to environment stimuli); no exception propagation; no dynamic priorities; no proper thread synchronization). Since the agent is a process - now acknowledged as such by a formal standard [13] - its temporality cannot be disregarded anymore.

- *Poor concurrent programming*. Often such situations are handled by resource wasting "active wait loops" or counterproductive "mutexes" instead of a simple Wait (for an event) with Timeout.

- *Misunderstood uncertainty*. Even if the fact that accurate numeric data are hard to get is accepted, the emphasis is on approximated, predicted, evaluated by rule of thumb, or even on intrinsically fuzzy data, rather than on missing ones (lacking sensor information, delayed previous decisions, server crash etc.).

- *Distorted prediction*. Bayesian inferences are considered unsuitable to decision-making because "Even if decision-makers could get all [Ĕ] answers in due time, would they believe them strongly enough to make critical decisions only on their basis? Humans are not "probabilistic beings" and are very prone to any sort of "gambler's fallacy"." [9].

In short, the real-world decision-making challenge is: the situations are such that you have no time to solve (accurately, complex) problems.

3 Delimiting the Problem

The multifaceted issue of handling uncertainty in DSS must be first restricted to arrive at manageable complexity and afterwards defined as software engineering task.

Restricting the scope. The target was narrowed to deal with undecidability due to future contingents because of five reasons:

- *Unaffordable complexity*. Uncertainty as epistemic concept, together with its species and degrees, was investigated in [9] starting from the 28 definitions found on the Web. Beside the overwhelming diversity of those definitions, ranging from "doubt" to "statistically defined discrepancy", the very meaning of uncertainty "depends on the professional background and on the task to carry out (better said, mostly on the time available to complete it)" [9]. Thus, "uncertain" means practically (mainly, subconsciously) for mathematicians, unknowable, for software developers, undependable, and for end users (decision-makers), undecidable. Recent related work attests the link between uncertainty and complexity: a terminology and typology of uncertainty "together with the role of uncertainty at different stages in the modelling processes. Brief reviews have been made of 14 different (partly complementary) methods commonly used in uncertainty assessment and characterisation" is presented in [17]. A rare case when undecidability is discussed outside mathematics is in a sociological context where the authors "suggest that as well as being able to consider organizational decision-making as an instance of (albeit bounded) rationality or calculability, it can

also be regarded as a process of choice amongst heterogeneous possibilities" [12]. "In this context could be found a common denominator for a general definition of uncertainty - at least, acceptable to the three categories mentioned above? Uncertainty, in its widest sense, comprises any unsure link in the chain of steps necessary to fulfil a task" [9]. Much too complex to cope with.

- *Avoidable complexity.* On the other hand, from the 28 definitions "only a few are interesting, since they are anthropocentric, mirroring the common user (mainly decision-maker) stance: a) "doubt" [...]; b) "the fundamental inability to make a deterministic prognosis" [...]; c) "lack of knowledge of future events"." [9].

- *Intrinsic importance*. "Real-world problems show that the most important and ill-treated kind of uncertainty is that due to future contingents: decisions are difficult to make because a relevant event not happened yet, not because a result is imprecise. Moreover, its pragmatic corollary highlights a key aspect in decision-making: since any statement about a future event is undecidable, how to proceed in this case? Should it be predicted, circumvented, waited?" [9].

- *Time Pressure*. The geometrically increasing computing power due to Moore's law (mentioned in Section 1) promotes factors tending to reduce radically the role of algorithms and (bivalent) logic in IT [7]. Two reasons are already manifest: "Since deterministic applications are vanishing (because of OHDUE), the conventional algorithm is not anymore program backbone. Even when still useful, the conventional algorithm is not anymore the main programming instrument (being hidden in procedures easily reached in a host of libraries or being generated by 4GL)" [8].

- *Risky Side Effects*. An algorithm is almost unable to feel time: no sense of future events, no such step as: "Warning: I don't know yet". Worse: often the algorithm cheats, confusing undecidability with negation.

Defining the task. "Since here the issue is to design a mechanism not a particular application, the cardinal concern, from a clear-cut software engineering perspective, is about reducing complexity, both structural (to make the mechanism useful to legacy systems too) and cognitive (to motivate system designers as well as to increase user acceptance)" [9]. Software engineering constraints imply: simple tools, with immediate applicability in current designs; no sophisticated concepts or instruments (such as agents, temporal logics, explicit uncertain information processing, computability theory and computational complexity theory, Bayesian methods, certainty factors, etc.); bottom-up design and testing; as much as possible conventional development (prevalent algorithmic reasoning, usual API functions, straightforward implementation, downward compatibility, etc.).

While most restrictions are comprehensible - albeit very tough -, prohibiting both temporal logics and explicit uncertain information processing seems counterproductive, since one of the premises claimed Time and Uncertainty as fundamental design-space dimensions. Unfortunately, even most responsive and appropriate approaches ([14] included) are less applicable because they are sectorial (e.g., treating time without uncertainty or vice versa). Other interesting temporal logics are too sophisticated: for instance, the "linear time" logic (used for specifying reactive systems) is based on the two modalities "Since" and "Until"; in [16], extensions of this logic are investigated from the perspective of undecidability for "X will happen within t unit of time" showing that the extension is undecidable, whenever t is irrational.

Although aspects of a primitive temporal dimension could be implemented using common API functions - e.g., Wait (for an event) with Timeout - as well as synchronization methods used in multithreading, no similar mechanism is available for uncertainty. Here lies the innovative core of the undertaking with all its potential, openings and risks: the only mechanism on hand in common operating systems, able to shift initiative from algorithm to environment, is exception handling. Thus, the algorithm gives up a morsel of its proactiveness - deterministic par excellence - to gain a touch of reactivity; this "small

amount of non-determinism" is mandatory to be able to mirror (in real time) its epistemic facet: uncertainty. (In DOMINO, when a conditional expression in IF is undecidable, an exception is raised and control is handed over to its handler.)

The idea to assign a new semantic value - i.e., enabling a software entity (labelled or not as agent) to react prompt to asynchronous environment stimuli - to a mechanism meant to increase robustness was advanced (and defended in detail) in [3] after testing it in expressing emotions of pathematic agents. Later, using exceptions to achieve agent reactivity was suggested in the context of sub-symbolic inferences [8] and of emergence as leverage [9]: "Even primeval animals move "algorithmically" ("if gap then avoid, else go on") only a few steps, in very hostile environments. Moreover, reaction to stimuli cannot mean perpetual looking for the stimulus. The cardinal hindrance stems [...] from the mechanisms employed: neither nature, nor technology can afford in the long run mechanisms involving large amount of testing because they are too time-consuming tools". (Currently exceptions are tested to help self-referencing agents to show some primitive form of self-awareness [5]. However, in recent related work exceptions are still regarded solely as a "mechanism for structuring error recovery in software systems so that errors can be more easily detected, signaled, and handled" [11]. One of the "fundamental motivations for employing exception handling in [...] robust applications" is to "improve the maintainability of the normal code and promote reuse of error handling code" [11]. In [10], the exception handling mechanism of a state-of-the-art industrial embedded software system is analysed and the fault-proneness of the return-code idiom - for dealing with exceptions - is evaluated.)

As a result, since a conditional expression in an ordinary IF has now a third exit variant (the exception), the very IF acquires a trivalent logic semantics. Then, why stay at the stage of "trivalent logic semantics" and not go further towards "trivalent logic"? Because such logics are far to meet the challenge, despite the huge effort to conceptualise, develop and implement them. A three-valued logic is a "logical structure which does not assume the law of the excluded middle. Three truth values are possible: true, false, or undecided. There are 3072 such logics" [18]. To avoid the 3073rd one[1] , "for the sake of simplicity, the trivalent semantics should be grounded on a usual bivalent software infrastructure" [9]. Indeed, the proposed solution for DOMINO considers the revisited concept of undecidability [7] [9] filtered through the double sieve of relevance to usual decision-making (Section 4) and design simplicity as vital software engineering request (Section 5).

4 In Search of the Third Value Semantics

Since the key aspect in decision-making is to handle "don't know"-like uncertainty [7] (e.g., when a relevant component of an IF condition is undecidable because an expected event not happened yet), it is appropriate to a third value meaning something like "Caveat: I don't know (yet)", "unknowable" or "unknown". In many-valued logics it "is general usage [...] to assume that there are two particular truth degrees, usually denoted by "0" and "1", respectively, which act like the traditional truth values "falsum" and "verum"." [15]. "Obviously, any decision-making needs those pillars of bivalence" [9]. Thus, the third value should be added to the two Boolean constants of IF. In short, the three outputs are: *Yes, No, Undecidable*, where it must still be decided what should "*Undecidable*" really mean.

The investigation starts with the first and more relevant three-valued logics and their respective motivations and meanings of the third truth value:

- *Łukasiewicz.* The first intention was to use the third value for "possible" (to model modalities) to deal with future contingents (first of all, the "paradox of the sea battle" posed by Diodorus Cronus). Ignoring the philosophical motivation and context (the ontological status of the future, is it determined or not, does free will actually exist, etc.), the third value - "**i**" for "indeterminate", interpreted

[1] At least (supposing that in this century no other trivalent logic has been concocted)

as "unknowable" or "problematical" - is semantically very close to the real-world decision-making problems. (On the contrary, the "1/2" notation - although elegant from a (meta)mathematical perspective - is totally unacceptable in software because, for both theoretical and practical reasons, interpreting it as intermediate value of "half true and half false" is at least confusing and useless; moreover, it could lead to computational disaster.)

- *Kleene.* The third value in this logic is "**u**", interpreted as "undefined" or "undetermined" or "unknown" - with two connotations: "permanently unknown" or "temporary lack of knowledge". The second meaning is helpful for postponing decisions and is actually used in data base applications. For instance, "SQL implements ternary logic as a means of handling NULL field content. SQL uses NULL to represent missing data in a database. If a field contains no defined value, SQL assumes this means that an actual value exists, but that value is not currently recorded in the database. [...] Comparing anything to NULL - even another NULL - results in an UNKNOWN truth state. For example, the SQL expression "City = 'Paris'" resolves to FALSE for a record with "Chicago" in the City field, but it resolves to UNKNOWN for a record with a NULL City field. In other words, to SQL, an undefined field represents potentially any possible value: a missing city might or might not represent Paris." (en.wikipedia.org/wiki/ Ternary_logic). Moreover, the Kleene logic, as a natural sublogic of Belnap's four-valued logic [2], is an important framework for many paraconsistent logics - major feature for application in computer science.

- *Bochvar.* Inspired by the examination of semantic paradoxes, the third truth value "**m**" means "meaningless" or "paradoxical". This logic is useful when syntactic meaningfulness (rather than semantic one) is looked for (crucial for program verification but dispensable for decision-making).

Hence the semantics of the third value in a "three-output IF" should be based on a blend of a Łukasiewicz "**i**" interpreted as "unknowable" or "problematical"[2] and a Kleene "**u**" interpreted as "temporary lack of knowledge". Thus, the semantics of "Undecidable" is refined to "Undecidable in the time span given". In fact, it gives a chance to the "yet" in "I don't know (yet)", postponing the verdict of "Undecidable" as much as possible (depending heavily on both the problem to solve and the decision-making strategies applied). Of course, to be effective, such a procrastination strategy must be put to work only in a distributed environment (reflected in software through multithreading); otherwise, the user would be frustrated by the frequent wait periods.

5 In Search of Bivalent Syntax Clothes

DOMINO respects the rigours of structural programming and the syntax of bivalent logic, is programmed in plain C++ (to be applicable to legacy systems too), is based on a few functions of the Windows32 API, was tested only within the toy problems (regarding real-time decisions in industrial control or in medical informatics) the examples are taken from, and is currently tested in a real-world problem (a simplified version of implementing overbooking policy). Its flowchart is in Figure 1 (the only symbol that is not among the common flowchart notation is the one for raising exceptions).

The (Windows 2000) API functions involved are:

```
a)
BOOL SetEvent(hEvent)
HANDLE hEvent;     /* Event-object handle */
   Example: SetEvent(g_hEvent_pHLimit);
```

[2] According to the ancient Stoic perspective, updated by Łukasiewicz

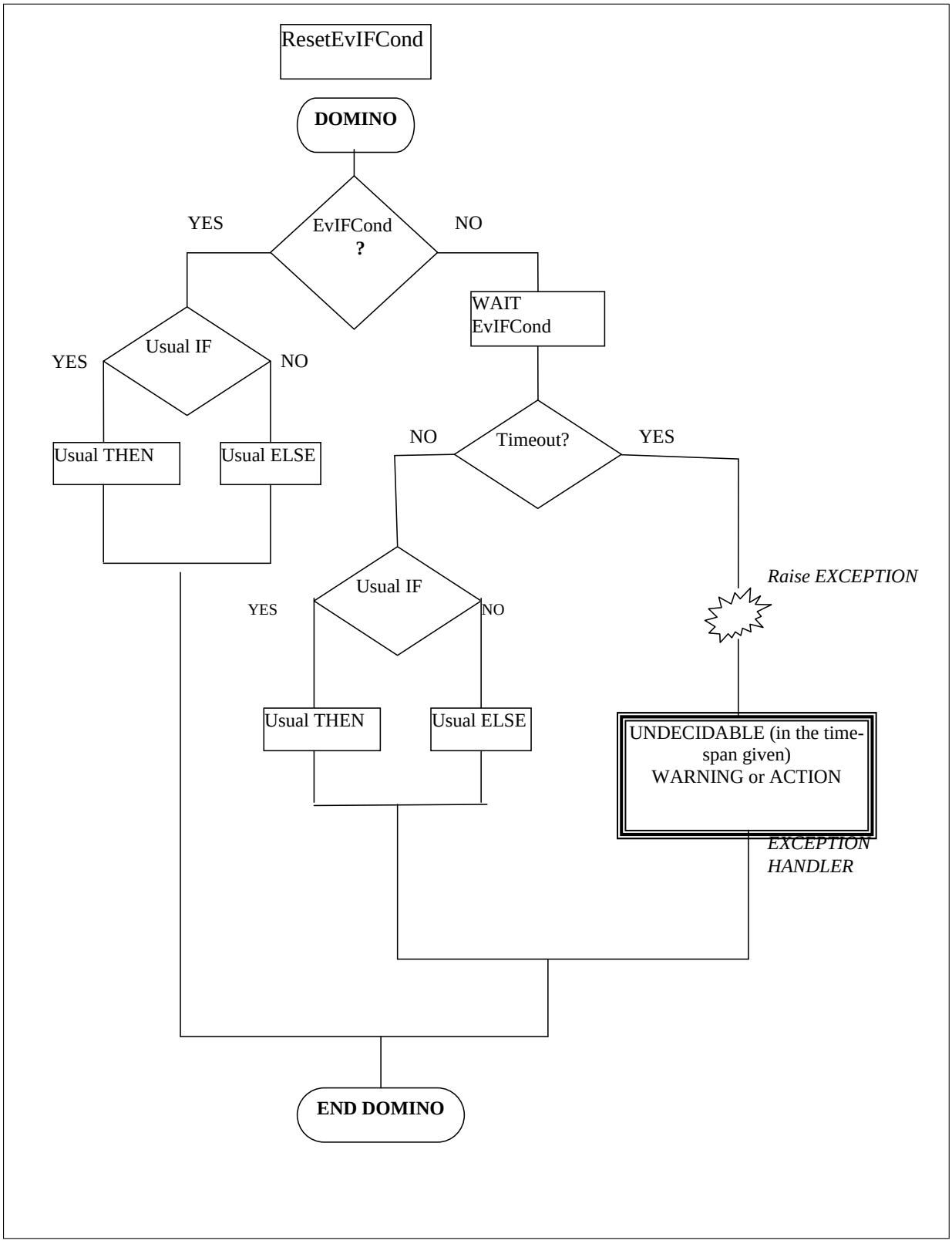

Figure 1: DOMINO: Architecture expressed in bivalent logic

```
b)
BOOL ResetEvent(hEvent)      /* likewise */

c)
DWORD WaitForSingleObject(hObject, dwTimeout);
HANDLE hObject;  /* handle of the Event waited for*/
DWORD dwTimeout; /* maximal Wait duration (ms)
(INFINITE: unlimited wait) (0: testing the Event state) */
    Example from an alarm-bell program:
dwResult = WaitForSingleObject(g_hEventOverheating, INFINITE);

d)
DWORD WaitForMultipleObjects(cObjects, lphObjects,
                                    fWaitAll, dwTimeout);
DWORD cObjects;  /* number of objects (maximum 64) */
CONST HANDLE *lphObjects;      /* handle table address */
BOOL fWaitAll;   /* flag for conjunction /disjunction
  (TRUE: all events are waited)
  (FALSE: only the first event is waited)*/
DWORD dwTimeout;         /* likewise*/
    Example from a domotics program
    (for 5 boilers; uses multiple events):
dwResult = WaitForMultipleObjects(5, hBoilerPressure, TRUE, 4000);
```

Remarks.

1. The exception handler can: a) give control to the human decision maker (after a warning); or b) act (for instance, propagating dynamically the exception from the callee to the caller).

2. Time is not just explicitly present in "Wait" but, more important, hidden in thread synchronization (SetEvent is employed in the thread providing information needed to make a decision).

3. Uncertainty is dealt with through the intrinsically uncertain interrupts generated by the environment stimuli, expressed at the program level through the exception handler.

4. If dwTimeout = 0, the mechanism is a conventional "IF" (however, a more robust one since evaluation is preceded by a real-time validation test).

6 Expectations

The three time horizons are roughly delimited by: a) validating DOMINO in the overbooking application; b) designing and validating other (similar) mechanisms of the AGORITHM toolbox [9]; c) API functions for DOMINO-like primitives.

Short range.

A) DOMINO addresses the main problem of current "IF" in applications running in OHDUE: "IF" is inadequate not because it cannot say "probably 80B) It solves this problem in a simple way (for both designer and user).

C) It is easy to implement due to its straightforward structure involving only common API functions: despite its enriched semantics the emulated "IF" does not need a modified compiler.

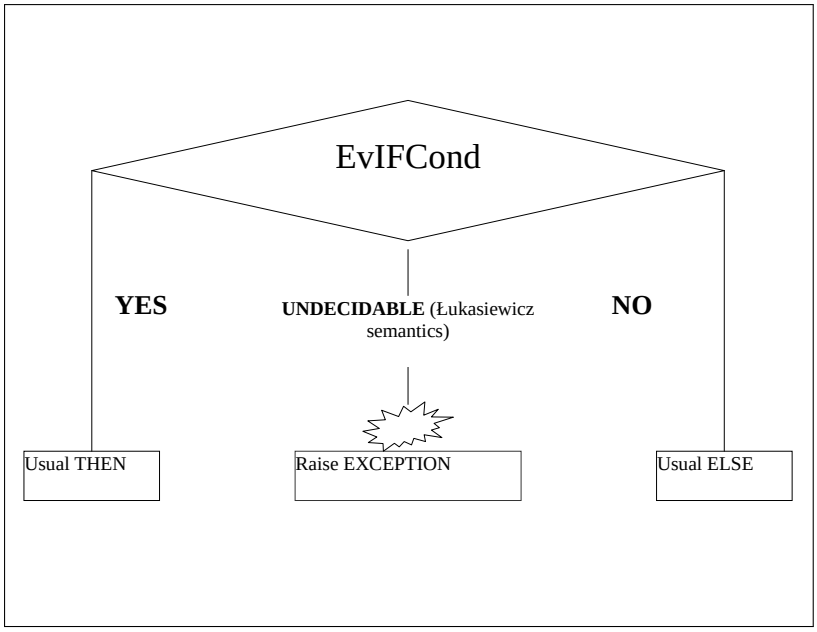

Figure 2: DOMINO: Semantics interpreted in trivalent logic

Middle range.
A) Corollary 1: being downward compatible, it should be useful for legacy systems.
B) Corollary 2: emulating a language primitive, it should be useful not only for DSS, but for any application in OHDUE.
C) Corollary 3: dealing with both time and uncertainty, it should be useful for any agent-oriented application: other AGORITHM mechanisms could be outlined similarly.
D) Hopefully it will start a "virtuous circle" able to adapt the DSS to the "Information Age" requirements changing both the decision-makers expectations and the programming paradigm.

Long range.
A) A new interpretation of DOMINO: a three-valued IF within the frame of bivalent logic
B) The badly needed shift in software engineering towards exception handling: the stimulus causes an interrupt that is treated as exception. (Real-world applications cannot afford large amount of "if temperature > n 0C then alarm" testing
C) Motivating applied research in AI logics, avoiding the gap between new logics (dealing at highly abstract mathematical levels with either time or non-determinism but rarely with both) and decision-support applications (either remaining at the level of rigid algorithmic approach or using approaches that ignore the fundamentals of human decision making).
D) A new, "procrastination logic": less algorithmic, less probabilistic, less difficult to understand and to apply in OHDUE, where postponing a decision means keeping open alternatives (to react better to rapid environment changes).

Bibliography

[1] AgentLink Roadmap: Overview and Consultation Report, AgentLink III. Agent based computing, University of Southampton, 2004.

[2] Anderson, A.R, N.D. Belnap, J.M. Dunn. *Entailment: The Logic of Relevance and Necessity*, Volume 2. Princeton University Press, Princeton, 1992.

[3] Barbat, B.E. *Agent-Oriented Intelligent Systems.* Romanian Academy Publishing House, Bucharest, 2002 (in Romanian, "Grigore Moisil" Prize of the Romanian Academy).

[4] Barbat, B.E. *The Impact of Broad-Band Communication upon HMI Language(s). (Chapter 7.) Communicating in the world of humans and ICTs. (Chapter 8.) in COST Action 269. e-Citizens in the Arena of Social and Political Communication* (L. Fortunati, Ed.), pp. 113-142, EUR21803, Office for Official Publications of the European Communities, Luxembourg, 2005.

[5] Barbat, B.E., A. Moiceanu, I. Pah. *Gödelian Self-Reference in Agent-Oriented Software.* Proc. of the 11th WSEAS International Conference on COMPUTERS (ICCOMP '07) (N.E. Mastorakis et al, Eds.), 92-97, Agios Nikolaos, Crete, 2007.

[6] Barbat, B.E, A. Moiceanu, S. Plesca, S.C. Negulescu. *Affordability and Paradigms in Agent-Based Systems.* Computer Science Journal of Moldova, 2007. (In print.)

[7] Barbat, B.E., R.S. Muntean, R. Fabian. *Approximation versus Undecidability in Economic Modelling.* Proc. of the International Workshop New approaches, Algorithms and Advanced Computational Techniques in Approximation Theory and its Applications (D. Simian, Ed.), 2007. (In print.)

[8] Barbat, B.E., S.C. Negulescu. *From Algorithms to (Sub-)Symbolic Inferences in Multi-Agent Systems.* International Journal of Computers, Communications & Control, 1, 3, 5-12, 2006. (Paper selected from the Proc. of ICCCC 2006.)

[9] Barbat, B.E., S.C. Negulescu, S. Plesca. Emergence as Leverage and Non-Algorithmic Approaches in Agent-Oriented Software. Studies in Informatics and Control Journal, 16, 4, 2007. (In print.)

[10] Bruntink, M., A. van Deursen, T. Tourwé. *Discovering faults in idiom-based exception handling.* Proc. of the 28th international conference on Software engineering, 242 - 251, ACM Press, New York, 2006.

[11] Castor Filho, F., A. Garcia, C.M.F. Rubira. *Error Handling as an Aspect.* Proceedings of the 2nd workshop on Best practices in applying aspect-oriented software development, ACM Press, New York, 2007.

[12] Clegg, S., Kornberger, M., Rhodes, C. *Organizational ethics, decision making, undecidability.* The Sociological Review, 55, 2, 393-409(17), Blackwell Publishing, 2007.

[13] FIPA TC Agent Management. *FIPA Agent Management Specification. Standard SC00023K (2004/18/03).* http://www.fipa.org/specs/fipa00023/SC00023K.pdf, 2004.

[14] Fisher, M. *Temporal Development Methods for Agent-Based Systems. Autonomous Agents and Multi-Agent Systems,* 10, 41-66, Springer Science + Business Media Inc., 2005.

[15] Gottwald, S. *Many-valued Logic. In Stanford Encyclopedia of Philosophy* (E.N. Zalta, Ed.). http://plato. stanford.edu/entries/logic-manyvalued/, 2004.

[16] Rabinovich, A. *Temporal logics with incommensurable distances are undecidable.* Information and Computation, 205, 5, 707-715, Elsevier, 2007.

[17] Refsgaard, J.C. et al. *Uncertainty in the environmental modelling process-A framework and guidance.* Environmental Modelling & Software, 1543-1556, Elsevier, 2007.

[18] Weisstein, E.W. Three-Valued Logic. *From MathWorld–A Wolfram Web Resource.* http://mathworld. Wolfram.com/Three-ValuedLogic.html. CRC Press LLC, Wolfram Research, 1999.

[19] Zambonelli, F., A. Omicini. *Challenges and Research Directions in Agent-Oriented Software Engineering.* Autonomous Agents and Multi-Agent Systems, 9, 253-283, Kluwer Academic Publishers, 2004.

Boldur E. Bărbat
"Lucian Blaga" University of Sibiu
Department of Computer Science
5-7 Ion Raţiu St., 550012, Sibiu, ROMANIA
Email: bbarbat@gmail.com

Received: March, 24, 2007

Boldur Barbat M.Sc. in Electronic Engineering, postgraduate specialising in Programming, Ph.D. in Digital Computers ("Politehnica" University Bucharest). He is with "Lucian Blaga" University Sibiu, Faculty of Sciences (full professor) and "Politehnica" University Timisoara, Faculty of Automation and Computers (advisor for doctoral studies in Computer Science). Over 20 books (Romanian Academy IT Prize, 2002). About 50 papers/articles in English in the last five years. Current research interests: emergence in agent-based systems (self-awareness, stigmergy), human-agent interaction (transdisciplinary), agent-oriented software engineering (non-algorithmic real-time software, uncertainty).