Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 217 (2023) 835–845

Procedia Computer Science

www.elsevier.com/locate/procedia

4th International Conference on Industry 4.0 and Smart Manufacturing

Paradigms for database-centric application interfaces

Massimiliano Pirani[a,*], Alessandro Cucchiarelli[a], Luca Spalazzi[a]

[a]*DII Dept. of Information Engineering, Marche Polytechnic University, via Brecce Bianche 12, Ancona 60131, Italy*

Abstract

The database-centric approach for industrial applications in the fourth industrial revolution has been proposed as a viable possibility in view of new trends towards distributed, autonomic, and intelligent control systems. In particular, with the RMAS architecture and its compliance to the new directions envisioned by the IEC 61499 standard, a suitable advanced instance of the database-centric paradigm was achieved. In this work, the focus is on the aspects that concern the impact of the database-centric paradigm in the realm of design of human-machine and machine-to-machine interfaces. A discussion of the implications and an example are provided in order to let the industrial informatics community start with an assessment of the proposed vision.

© 2022 The Authors. Published by Elsevier B.V.
This is an open access article under the CC BY-NC-ND license (https://creativecommons.org/licenses/by-nc-nd/4.0)
Peer-review under responsibility of the scientific committee of the 4th International Conference on Industry 4.0 and Smart Manufacturing

Keywords: Active database management systems; machine interfaces; database-centric applications; relational-model multi agent system; human-machine interaction

1. Introduction

The use of a database-centric paradigm [1] can enable new visions in the concept of automated design, engineering, and deployment of software for application interfaces. It is achievable when the database-centric paradigm is associated with a MVC (model-view-controller) pattern in the context of intelligent and smart automation applications.

It is well-known that most of the software projects spend a significant proportion of their time developing UIs (user interfaces) [2]. In most of the cases, in particular in the industrial realm, the stylish rendering of UIs is not a priority, but even basic but professional UI development remains an unavoidable cost and barrier. In this case, an automation of UI development would be highly desirable, or at least a steep decrease in the effort for human UI developers.

* Corresponding author. Tel.: +39-071-220-4471 ; fax: +39-071-220-4224.
E-mail address: m.pirani@univpm.it

1877-0509 © 2022 The Authors. Published by Elsevier B.V.
This is an open access article under the CC BY-NC-ND license (https://creativecommons.org/licenses/by-nc-nd/4.0)
Peer-review under responsibility of the scientific committee of the 4th International Conference on Industry 4.0 and Smart Manufacturing
10.1016/j.procs.2022.12.280

Nomenclature

ADBMS	active database management system
API	application programming interface
DBMS	database management system
ECA	event-condition-action
MI	machine interface
MVC	model-view-controller
RM	relational model
RMAS	relational-model multi-agent system
SQL	structured query language
UI	user interface

The ability to adapt and speed up the design and development of user interfaces for applications will be increasingly required when new paradigms of distributed and adaptive interfaces are needed [3].

Interface development is still not a clear priority in industrial realm. Current shop-floor interfaces are mostly variants of SCADA (supervisory control and data acquisition) systems that, although robust and functional, are based on nearly 20-years old technologies, even though they are the flagship products of some of the leading suppliers. Industry 4.0 requires greater flexibility and richness and Industry 5.0 adds a sustainability dimension that has never been considered clearly in the past. The database-centric approach, along with recent standardization efforts like the IEC 61499, are keys for overcoming many technological silos and allow increased interoperability [4].

Relying on fundamental lessons learnt on software engineering is fundamental for future developments. F. Brooks's work has been a milestone for software engineering, and his principles need be reclaimed and looked in a new perspective: "*Show me your flowcharts and conceal your tables, and I shall continue to be mystified. Show me your tables, and I won't usually need your flowcharts; they'll be obvious.*" [5, p. 102]. This can be read here as: for an unknown black-box object, which architecture is database-centric (actually a grey box if some structure is known), all the informational content and behaviour of it can be in principle accessed exclusively by means of database queries.

This property holds naturally for an RMAS (Relational-Model multi-agent system) unit as defined in [6] (and references therein), but in general also for database-centric approaches [1]. In such a context, the automated or manual development of a machine interface (MI) to targeted digital objects or applications, consists in sequentially querying the object in order to discover and retrieve, from its reactions, what informational contents it holds (if not known) or what it can provide from a behavioural point of view. The MI concept subsumes the user interface (UI) concept and extends it also to artificial actors.

Our work heads for a future in which UI and MI design and self-adaptation will result from the automation and systematization of their development frameworks. This is proposed by means of a combination of new and well-known paradigms in order to attack the problem of quick and effective development of UIs and MIs in general.

In section 2, some related work and background for the discussion is provided. In section 3, the methodology and the paradigms that this work proposes are defined. A brief, but explicative example will be reported in section 4, in order to show a possible practice for the proposed methodology. Section 5 is dedicated to the conclusion of this work.

2. Background and related work

The developments here pursued belong to a long-lasting framework that initiated with a database-centric vision for industrial automation [1, 7]. The database-centric architecture follows an event-driven pattern. The insertion of data into a database table is an event that is captured with a database trigger. Thus, the database-centric paradigm uses the trigger mechanism of active database management systems (ADBMS) to implement distributed computations over a network of actors. Data flow is guaranteed by a flexible database replication infrastructure, along with a publish/subscribe paradigm [1].

In [7], the basic idea of using a DBMS (database management system) as the centre of all the information, all the communications, and control activities in an industrial process was outlined. The field experience and research conducted in two European research projects have laid the foundation for such a stance and approach (CAFE id. 212754, FRISBEE id. 245288).

From its inception, the database-centric vision constituted a completely reversed and anti-intuitive approach with respect to traditional and state-of-the-art schemes in industrial context. In usual and classical practice, databases are accessed and used as mere information storage systems, keeping logic, control, and information processing in separate layers of hardware and software. This is still a dominant approach used by major players in industrial process automation. Nevertheless, some database-centric methodology started to clearly appear on the market around 2011 [8]. Since then, many practitioners started to endow the (mostly SQL-based) DBMS systems with a more active role than in the past, in particular for process integration and interoperability.

Besides, the database-centric approach can count on a long research and practice history. Its developments are linked to the concept of active database management systems (ADBMS), which is a long-lasting line of research and vision.

Active database applications use active databases with event-condition-action (ECA) rules that specify the action to be executed by the database when a certain event occurs under a given condition [9].

In general, ADBMSs support mechanisms that enable them to respond automatically to events that are taking place either inside or outside the database system itself. Moreover, they allow a very rich and general definition of an event, which can be seen as a product of relational processing in its own right. When the relational model definition is extended in a most comprehensive way, an event is in itself a relation, but in general a special relation that creates a bridge between physical phenomena and computations [10]. An ADBMS provides mechanisms for users to: describe the reactive behavior or *knowledge model*; support for monitoring and reacting to relevant circumstances (*execution model*); and support for maintaining browsing, and debugging reactive behavior (*management tasks*) [9].

ADBMS mechanism is based on triggers that use rules having up to three components: an event, a condition, and a action (namely, the ECA rules). But most of all, an ADBMS can be a natural implementation for a reactive agent and a flexible and scalable framework for implementations of self-adaptive applications. However, as the limit between the external and the internal of an agency is blurred, the same reactivity can become a capability of proactivity for autonomic computing [6].

Since [7], it was remarked that more profound and extended use of ADBMS technology allows to organize a heterogeneous and scalable set of control systems. This is due to the possibility for an ADBMS to trigger all the actions in a distributed and networked control infrastructure for events coming both from hardware and software application layers. Such triggering events occur when an acquisition or a rewrite changes the value of a variable. The values of a variable are contained into the instance of a relation, which in practice is implemented as a database table [11].

The ADBMS can constitute a middleware and a common ground for many software solutions. An ADBMS can usually provide a set of APIs (application program interfaces) for different languages and architectures to access and manipulate it. As noted in [6, 12] and other references therein, a great part of what mainstream programming can accomplish can be done with an appropriate ADBMS tightly coupled with an external programming language, in particular in the context of pervasive environments. When an ADBMS is used as a hub for data and logic, the concern of the design of control programs, and their deployment, is detached from the details concerning connection to sensors and actuators. The capabilities of a DBMS provide abstractions with respect to the domain level. The DBMS language in general allows the programming to be lifted up to a descriptive level. The database objects and engines can be used as the main interface between different processes, distinct control layers, and legacy or alternative standard solutions.

In [1], the database-centric was introduced as an “information-conservative” approach. It was devised to be pervasive and interoperable across all the levels of the ISA-95 standard pyramid (or the newer IEC 62264), from sensing and actuation up to the management of a network of enterprises. The core of this concept is that a database contains the maximum amount of informational content available. This is due to the fact that an unspecified set of *views* can be gained from a certain database schema: the way in which a query is crafted provides a different informational content, depending also from the added knowledge of the agent (subject) that makes the inquiry (about the object).

Moreover, some existing database schemas can be enriched and changed in unanticipated ways during information gathering sessions of data mining. Data quantity is measurable, but information and data quality is in principle infinite

depending on the way in which one agent views them. A similar principle has today permeated also the Digital Twin theory. In [13], it is rightly noted that an information model over a reality is potentially infinite. However, only partial selective, and temporary views can be achieved, depending on the context and the specific observer.

A similar experience on the possibilities of active database systems and related discussion on database-centric approach can be found in [14]. There, the advanced capabilities of a full-blown ADBMS like PostgreSQL were exploited, aiming to some kind of publish-subscribe scheme enabled by the inter-process communications provision of the specific DBMS. The limits in their implementation, though very valuable to express the principles, are mainly:

- *scalability*. An ADBMS solution should be able to scale potentially down to the tiny devices of the Industrial Internet of Things, or it would not meet most of the needs of the current cyber-physical system of systems scenarios. At the same time ADBMS has to scale up to larger and distributed systems where a database technology can constitute a bottleneck.
- *distribution*. The new paradigms for modeling, programming and simulation in industrial control and communication are heavily and dynamically networked, and oriented to distributed intelligence and computing; for example the ABC (agent-based computing) model of computation [11].

Downward scalability issues in database-centric implementations of distributed intelligence have been recently faced with examples in [12]. In [12], it is shown how a set of relational (also with just limiting the acceptation of relational model to SQL) transactions can solve by themselves a distributed negotiation for *holonic* agents in productive environment. The database technology is used as the key enabler for a performance improvement technique that deals with a fractal vision of the productive environment. Moreover, with a technological outlook, it was expressed the fact that DBMSs can eventually integrate and surrogate some layers of computational and functional stack for low-cost and low-resource devices. It was also shown that the ADBMS can take on the role of a middleware, like the Web, and at the same time can act as an implementation of a full-fledged operating system. In [12], a test of implementations on embedded systems have been performed, demonstrating that the footprint on memory resources can be limited to the order of hundred kilobytes, and the price of a device hosting the necessary functionality at around 5$.

The problem of distribution of computation of the ADBMS was also discussed in some detail in [11]. It was shown that it is possible to adopt a distributed computing strategy. The self-similarity of the schemas of a DBMS is leveraged in order to let the global conceptual schema be a mere union of self-similar local conceptual schemas. Nonetheless, even when this technological sophistication is not achievable, in [1] it is detailed how the selective replication of a global schema along with a publish-subscribe layer can allow an effective distribution of the computing for the distributed control of a cyber-physical system of systems.

An evolution of the database-centric approach and related technology proposals were eventually conveyed and integrated in the scalable and distributed RMAS (relational-model multi-agent system) architecture. In [6], the RMAS architecture in relation to autonomic computing was introduced. In this architecture, the database-centric paradigm constitutes the greatest part of its foundations. The RMAS unit is built around a relational computing core, which most effective implementation is achievable through a suitable ADBMS. In Figure 1, a basic scheme of an RMAS units is provided. In this figure, only the intermediate-level configuration of an RMAS unit is shown. Actually, in [11], it is shown and detailed how this level of RMAS unit is recursively an intermediate between possible parent and child levels. The multilevel RMAS architecture expresses at its best the distributed computing feature [11].

For the present scope, it is just recalled that any RMAS unit is defined by its input, output, core processing, and Oracular parts. These parts are associated to different partitions of the DBMS objects (denoted by different colors of tables in Figure 1. The *inputs* partition receives queries in the specific relational language (not necessarily SQL). These are processed from the input part and handled with a *publishers* scheme in which the associations to the message source is managed and used. Once an input is received, it triggers a cascade of computations that might involve all the other parts (the other database partitions). In particular, the output part will in turn handle a *subscribers* scheme in order to route purposefully the outcoming queries towards other RMAS units.

A summary list of some fundamental concepts and properties conveyed by RMAS are here recalled in order to render the following discussion self-consistent:

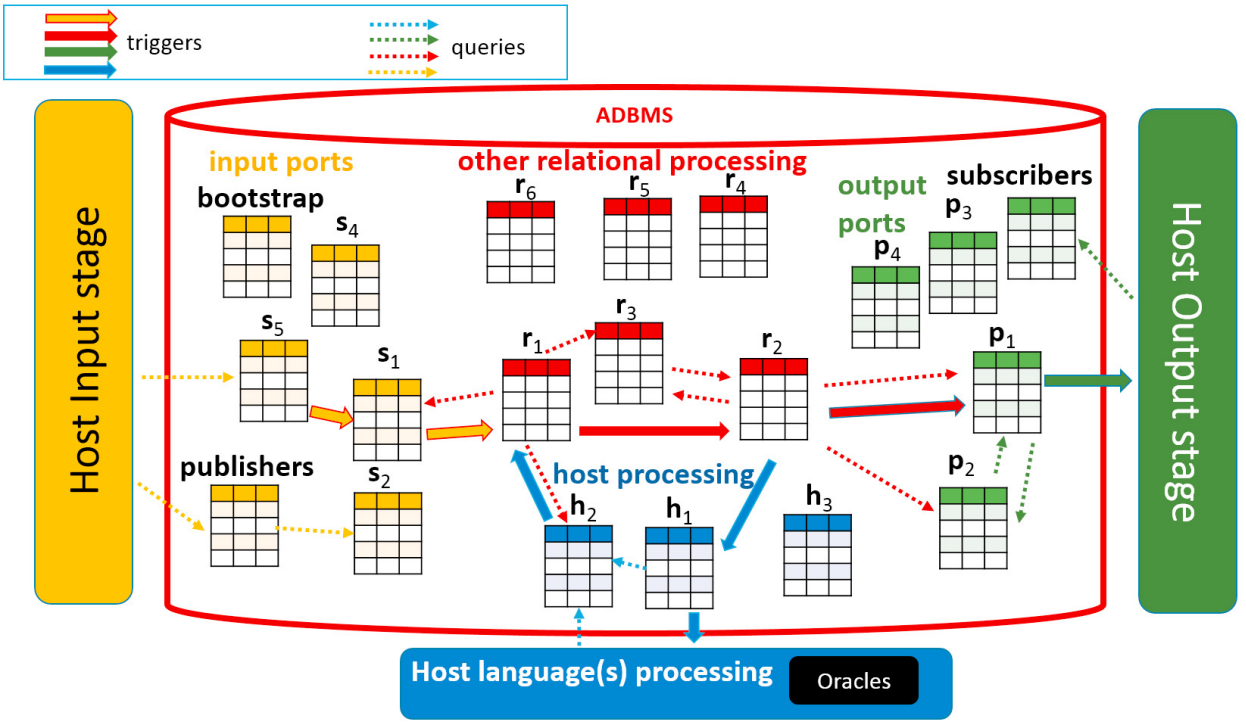

Fig. 1. Basic architecture and components of an RMAS unit.

- An RMAS unit is composed of relational database objects like tables, constraints, triggers, stored procedures, schemas, and libraries of procedural code that is invoked through the database triggers mechanism.
- All the computations and information processing in RMAS can be carried out in an event-driven fashion, where events are associated to update queries in the underlying active and distributed database management system.
- Some of the procedural code can be invoked by means of the host language. This allows to proceed in different computational realms beyond the relational computation. Such a capability is defined as Oracular computation. It is a computation that proceeds beyond the relational realm and is necessary for the contact between the database and the causal effects to and from the environment. Among many, this capability can cover for example the transduction from sensors and to actuators. An Oracle is a black-box object (with a well-defined set of input–output relations) that is queried from the algorithmic part of the program. Oracles can be any other model of computation beyond the relational model, being it any Turing machine, a huge distributed database, a neural network, an analog circuit, and others.

The RMAS architecture was initially introduced for a factual realization of networked and distributed automation, with event-driven and database-centric paradigms, towards advanced holonic MAS, basing on best available technologies [1]. Eventually, in [6], RMAS has been challenged against the expectations of autonomic computing, which required a minimum set of self-* capabilities that can be achieved through suitable technological support for autonomous program rewriting. The self-* capabilities, are essential requirement for the introduction of AI (artificial intelligence) in the actual industrial realm. Self-* can comprehend a vast set of capabilities, though major literature identifies the fundamental categories as self-configuration, self-healing, self-optimization, self-protection, self-regulation, self-learning, and self-awareness [6].

In addition, RMAS was proposed as a novel methodology and an overlay architecture over the computational semantic and runtime models of the IEC 61499 standard [4].

For further details on the RMAS architecture we refer the reader to [6, 11, 4] and references therein. In particular, a brief comparison between RMAS and other MAS (multi-agent systems) that expressed the agent-based computing in the recent manufacturing playground can be found in [11].

Another important paradigm for RMAS, beyond the database-centric, is the reliance on the *actor model* (see [11] and references therein). In the *actor model* the actors are concurrent units of computation, like the RMAS units, which communicate by asynchronous (and guaranteed) message passing, and the only way for an actor to communicate with another actor is by sending it messages. In response to an asynchronous message, an actor α_1 may perform one or more of the following actions:

- send a message to an acquaintance, an actor whose identity (address) is known to the actor α_1;
- create a new actor α_2, with a given behavior b_2 , or
- become ready to receive a new message carrying new behavior b_2.

The compliance of the RMAS to the *actor model* provides the involvement of RMAS in the ever-growing microservices and nanoservices architectures for distributed and decentralized applications [15], where actors (services) communicate each other through some form of message-oriented middleware. These architectures are a (relatively-)new big trend. The greatest digital business players use microservices to build large, complex and horizontally scalable applications composed of microservices that are small, independent and highly decoupled processes communicating with each other using language-agnostic application programming interfaces (API). In turn, nanoservices are a miniature version of microservices that tend to avoid container technology for ultimate downward scalability [15].

In this context, the RMAS architecture is at the same time an enabler for nanoservices – as it can scale to extremely low resource devices even without the use of operating system [12] – or can easily surrogate a typical microsystem's container. In fact, RMAS can provide (in principle) all the functionalities of a full-fledged operating environment (like a container) with suitable driver-level libraries and communication APIs and, most of all, a local DBMS [6, 11].

Differently from the cited literature, in this paper the focus will be on the Oracular part of RMAS. In particular, what will be interested here is the part that constitutes the interaction between RMAS units, and its database-centric paradigm, with the engineering of a user interface (UI). The part of RMAS that is more interested by interfaces is the generic host processing part (see Figure 1). This part wraps and control the contact with the Oracles that are any form of computation that can be accessed through APIs from the host language. A UI is just another Oracle for RMAS.

The overall effort in the development of a UI can be greatly affected by the architecture of RMAS and its capability to expose internal properties and mechanism in a unified way. In the RMAS approach, part of the technology stack used for programming user or machine interfaces would be incorporated in the ADBMS, as we will explain in the following. In the next section the database-centric and derived RMAS architecture are at the basis of the definition of a methodology for the efficient development of interfaces.

3. Methodology and paradigms

The main idea behind all this work is that the development and the adaptation of UI (user interface) should be the easiest and less effort consuming. In principle, the general concept of interface should involve also machine-to-machine (M2M) interfacing and interoperability (in a dynamical sense). Henceforth, we will refer MI (machine interface) to express both UI and M2M interfacing. This unification of terms is relevant if AI is considered in order for intelligent applications to be able to develop their own interfacing to an application, without major human intervention.

A unit that features the RMAS architecture is based on the general database-centric paradigm. It leverages ADBMS capabilities to make extensive use of ECA rules in order to implement complex constraints, consistency check procedures or, in general, procedures that produce articulated instructions as results of a certain database query.

These features allow the development of a MI that can be seen in essence as a particular *materialized view* of the underlying DBMS. Materialized views are persistent database views. The MI implements a set of database view objects that issue queries to the DBMS, and displays their results. These queries will be parametric, but mostly static in their structure. Typical database views allow to asynchronously poll new data and fetch them for their visual rendering. Furthermore, when an ADBMS is available, a materialized view is surely possible, or in the worst case being emulated, at least by means of triggers and stored procedures. Values of the materialized view can be updated

selectively and synchronously with an event-based push process, which usually makes the updating more efficient and real-time oriented.

If the MI is conceived as a special materialized view, the architecture of the MI can follow common patterns like the Model-View-Controller (MVC) paradigm and the Observer Pattern. The MVC paradigm is involved as long as the MI must only deal with the visual rendering of statuses, events, and reactions (View) and the creation and submission of commands (Control) to the underlying Model.

The Model has to be engineered as a layer of software completely agnostic and independent from the specific programming technology that is used for the realization of the MI. Just as a prompt example, the Model shown in the following section 4 was written in Matlab and SQLite. Matlab is a language that is usually a bit more awkward in application interfaces with respect to state-of-the-art and web-based UI technologies. The possible weaknesses of Matlab in the UI part, are for sure balanced and traded off by its utmost strength in the computational capabilities for the Model and its business logic. Thus, the combination between Matlab and completely different purposes languages was considered a perfect instance to showcase the solidity of the paradigm here proposed. Besides, the SQLite and its provisions make the main role in the play as the major constituent of the ADBMS, and it appears to be the keystone language at the end of the day.

The MI, seen as a materialized view, might be developed in any language that can easily have access to the DBMS of the Model. The MI will communicate with the Model exclusively by means of relational queries. A note has to be made soon, in order to distinguish the concept of ”relational” from SQL technology: SQL does not realize the Relational Model in its full sense, where all the results of the operations from the relational language are *relation*s. A more profound discussion and details about the Relational Model and the relationships with SQL languages can be found in [11, 12] and references therein. For the present purposes, it should only be remarked that the coupling of SQL with some of its extensions (that unfortunately are not standardized) make at least the Relational Model capable of being emulated.

In this context, the Observer Pattern is called into question when some events are autonomously generated from the state evolution of the *Model*. These events affect the *View* that has to react to render the current state of affairs timely to one or more users.

A combination of the materialized view concept and of the Observer Pattern composes the paradigm here proposed: a methodology for the development of scalable and language-independent machine interfaces (MIs).

The combination of the two patterns can conveniently be accomplished by the technology and capabilities that an ADBMS can offer. However, the ADBMS definition can barely imply just the adoption of a DBMS plus some capability (more or less developed) of the database in activating procedures by means of triggers on arrival of new data or modifications of the database variables (reacting to an update action in general). In our particular releases and studies, the tiny SQLite embedded DBMS has been used for this purpose [12]. SQLite, among other nice features in resource footprint savings and portability on tiny embedded devices of edge computing, avoids the installation and the setup of a DBMS server or container.

If the database-centric paradigm is applied, the ADBMS ideally fulfils everything concerning the *Model* of the application. The adverb “ideally” is used to mean that the ADBMS is run just on fully relational language provisions as envisioned in [12]. In practice, such an ideality has never been reached, although a direction is already available [11]. In current state of best available technology the *Model* is a hybrid composition of host code and appropriate ADBMS with its own relational language. The host code has the role to extend the limits of SQL towards the Full Relational Model [12]. Host code is being invoked by the triggers of the ADBMS in order to produce unrestricted computing possibilities and whatever model of computation, in cascade to the relational computing core. Such an ADBMS under our proposal, has to constitute the “engine” of the application and the very flesh of it. Rather, the MI (machine interface) might remain a shallow piece of software that mostly constitutes the skin of the application.

Any MI implementing the View and the Control part of the application is then connected to this “engine” exclusively by means of relational queries. What is required to the MI is the capability to connect and manipulate the ADBMS, which in practice means accessing the ADBMS core application for the execution of retrieving queries (READ or SELECT) or updating queries (CREATE, UPDATE, DELETE, INSERT, etc.). In Figure 2, it is shown the schema that depicts the paradigm of the whole application through ADBMS and its MI. The schema in the figure is a reference architecture for applications that abide by the database-centric paradigm here recalled. The Model of the application is governed by a host language (in general even more than one). The host language can invoke (through

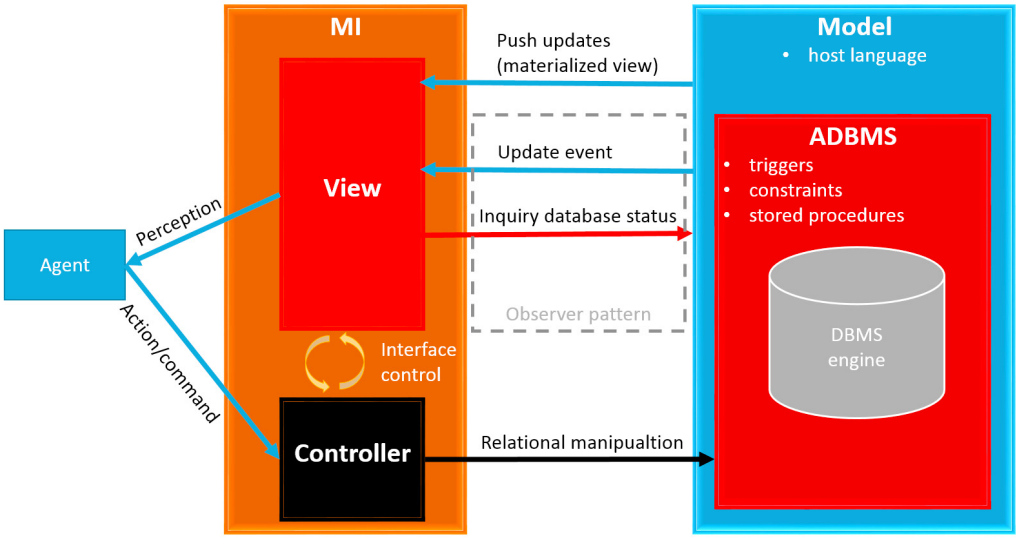

Fig. 2. The Model-View-Control and observer patterns in the database-centric reference architecture for machine interfaces.

appropriate APIs - application programming interfaces) relational language operations or be invoked by the ADBMS subsytem. The ADBMS can use the host language as a helper language for the enforcement of *constraints*, *triggers*, and *stored procedures*. The ADBMS can be seen definitely as a wrapper and shell for the DBMS engine that hosts relational language provisions.

The MI is in general composed of two subsystems, namely the View and the Controller, and a logic expressed in the language used for the MI (e.g., a Web-programming language like Javascript) that creates the interaction sometimes needed between the elements of the View and those of the Controller.

The View is updated in two major ways. The first, in case the View is implemented as a materialized view, can receive directly data as an asynchronous update event in a push fashion from the Model. In a second way, the Observer Pattern can be adopted. In this case, the View update, is obtained by means of an operation of information retrieval (typically a SELECT query in SQL), and has to be triggered by an event of update notification received from the Model. The collection of the event can be synchronous (with polling action) or asynchronous (with a push from the Model).

The Controller part is devoted to the operations that involve mostly an update of the Model's content and status directly or indirectly (through a command). This update operation can be a sequence of relational transactions that invoke SQL operators as CREATE, UPDATE, INSERT, DELETE, or even SELECT for sub-queries or further information retrieval. A rigorous definition for these operators and others in the context of the RMAS paradigm can be found in [11], but is avoided here for space and scope motivations.

Eventually, an entity interacts with the MI. In Figure 2, this entity is indicated as Agent. This term has been keenly adopted in order to express both a human and possibly artificial agent that had the capability to reason and react intentionally by means of commands and actions on the Controller, while receiving perceptions from the View. In case of artificial agents, the MI block should provide an API interface to the Agent. Since this possibility is left open, an artificial agent might be able to build its own interface from scratch by querying the Model, which is possible and viable in the database-centric approach, providing feedback and reactions that allow the agent to observe inside the Model.

A very important feature, in order to make the schema of Figure 2 being realistic and practically useful, is that the DBMS engine adopted has to sport concurrency in order to allow more that one process to keep open the connection to the underlying DBMS. While this is a feature that is natural in most of the DBMS having a server application to handle multiple connections, in tiny and embedded DBMS this feature is not granted. Nonetheless, SQLite has support for concurrency that behaves nicely for the most of the necessities.

4. An application example

Some prototypical and seminal versions of an application adopting the previous paradigms and related database-centric reference architecture have been successfully devised by author for more than a decade in private companies. Eventually, a first example of a similar implementation open to public was recently demonstrated in the developments of a EU project (ENCORE id. 820434).

In this project demonstration, the database-centric paradigm was used to develop two functionally equivalent versions of UI that accessed the same Model. The UI part, containing the View and Controller parts of the application have been obtained with two different technologies. A first version has been realized in Matlab language. A second version has been obtained in a typical Web-based development environment. Such an application demonstrated the implementation of a service for the management of a building renovation work based on a technique designated HMT (Holonic Management Tree) [16].

The application, has been developed by separating completely the part that embodies the Model and the part that is dedicated to the interaction with a human user, namely the UI. The Model part of the application has been developed using Matlab as the host language, as required in typical RMAS approach. The relational part of the Model, constituting an approximation of ADBMS, has been addressed using the SQLite DBMS by means of a basic Java library (JDBC) *sqlite-jdbc-3.8.7.jar*.

In Figure 3, a snapshot of the two UI prototypes are shown in the left part of the figure. In the left-upper part of the figure, the UI shown was obtained in Matlab environment, using the provisions of the *MATLAB App Designer* tool. In the left-lower part of the figure, the UI was developed in a typical Web development framework, namely the *Laravel* open-source PHP web framework,

Both UIs share the same Model. They express the same functionalities, with minor differences dictated by the technology and a slightly modified behaviour that depends on the resources and the features available in the two different development frameworks for the UI.

This experiment demonstrated how the use of a database-centric conception applied to the Model can conveniently be used to create multiple versions of UIs with reduced effort, by separating completely the concerns of the View and the Controller from the Model. The clear-cut separation between the Model and the other parts occurs not only in functional aspects but even in their technological implications. With the same approach, and minor effort, other kind of UIs can be easily created for the same Model. This property certainly allows a high degree of flexibility in satisfying users of very different natures. In Figure 3, are shown in schematic way also the main features that have been used in order to develop the example at hand. A remark has to be made, that the specific techniques or arrangements here used are only one specific instance of effective adoption of the RMAS and database-centric paradigms. However, the methodology here followed for the UI does not constitute any kind of directive on how to achieve it. Many other solutions could have been developed in alternative and possibly better ways, depending on the peculiar performance needs, requirements, and constraints.

In the special case at hand and for the requirements of efficiency and functionality, in Figure 3 are indicated the more relevant structures and relational features devised for the implementation of the Model:

- *tasks_queue*. This is a table in the database where computational actions are triggered by flagging a certain registered *event* as 'PENDING'. By following a typical ECA (event-condition-action) rule, the event is processed and a detailed return status can be accessed. Such a table constitutes the entry point for actions started by the UI. In order for the UI to start an action, it has only to submit a query to the SQLite database (using the appropriate API) and so trigger the event. An example is the following SQLite query to start the *app* session: "`UPDATE tasks_queue_HMI SET status='PENDING' WHERE event == 'start_session'`".
- *Relational processing*. In cascade to the triggers of the *tasks_queue*, an avalanche of relational processing is performed in order to complete the task invoked by *tasks_queue* mechanism. This mechanism produces a result as a reaction to the behaviour of the Model, which responds to the action started by the UI. This phase is denoted in Figure 3 with the interaction between tables like $r_1, r_2, \cdots, r_n$.
- *observer_queue*. This is a table in the database that contains a sequence (typically used as FIFO queue) of reaction commands (or methods) that the UI should express even beyond those directly connected to an UI action, as mentioned above. Due to the Model state evolution (for example the arrival of new sensors acquisition,

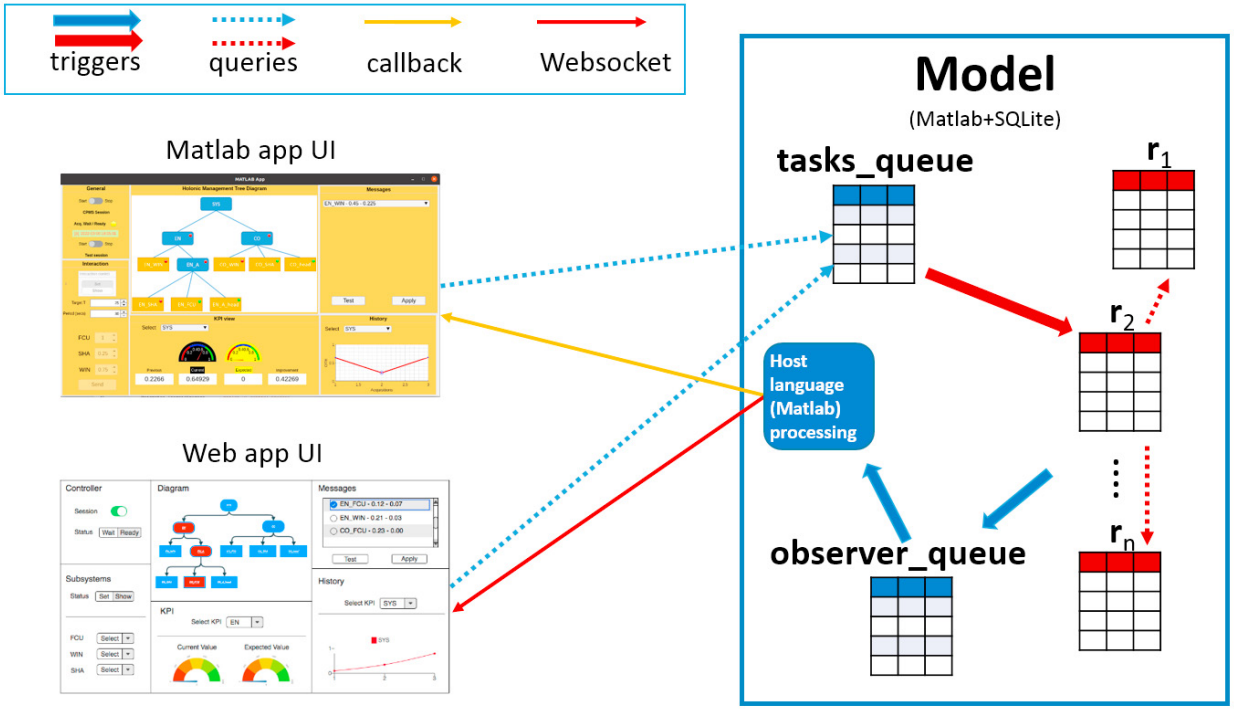

Fig. 3. Essential features that enabled the database-centic paradigm for the example UIs.

which is not triggered by the UI) a reaction command is inserted in the queue along with the other reactions to the actions triggered by means of the *tasks_queue*. The *observer_queue* is used to receive the strings that bear the invocation of a method for the UI. The semantics of the content of these strings depends on the UI context and technology. Thus the actual rendering and effect of the reaction command relies on "the eye of the beholder".

- *Host language processing* of the *observer queue*. This processing is triggered by the *observer_queue* in order to transmit and invoke the method of reaction to the UI. The kind of processing depends from the capabilities and nature of the UI. In the developed example, in the first case of Matlab UI, the processing consisted of an mere invocation of a Matlab function callback. In the second case, of Web UI, a technology like Websockets is exploited in order to push such reaction events to the UI. Note that this reaction event transmission is the first step in an implementation of the Observer Pattern. The second step is the retrieval, from the database, of the information needed for the processing and visual rendering of the reaction effect (e.g., a certain status or data for graphs), which is naturally achieved through the same API to the database used for the *tasks_queue* mechanism.

New research and developments on this kind of interaction between UIs and database-centric applications are currently under further development towards a key enabler technology for the aims and scope of the EU project (ENOUGH id. 101036588).

5. Conclusion

In this work the database-centric paradigm an the related RMAS architecture have been leveraged in order to provide a technological path towards fast and efficient development of machine interfaces for applications in industrial context.

With a simple and practically developed example it has been shown how the proposed database-centric paradigm can be promisingly adopted for new concepts of graphical interface design and development that will possibly be key enablers for a new generation of applications, pervasive, liquid, and scalable enough to face the needs of cyber-

physical system of systems. The proposed paradigm provides at the same time an agnostic stance with respect to the choice of programming languages for interfaces, and so independence and resilience to technological evolution and obsolescence.

Being the database-centric paradigm at the core of the RMAS architecture, the example here provided showed how the elements of the RMAS are effective key enablers for the simplified design of UIs that are destined to impact the introduction of distributed computing and multi agents in the industrial context, in particular as envisioned by the RMAS concept in relation to the IEC 61499 and the autonomic computing framework. The theme of adaptive interfaces is also implicitly introduced and involved, as the simplification in the process and the capabilities of active database systems are also an important enabler for new techniques in this direction.

Nonetheless, more insights in this matter are left for future works due to space and scope reasons.

The proposed example is limited to a simple solution that still does not directly confront with automatic development of machine interfaces but poses a foundational basis for future work in this field.

Acknowledgements

Financial support in this work has been partially achieved through two EU Horizon 2020 projects. The first is the ENCORE project g.a. 820434. The second is the ENOUGH project g.a. 101036588.

References

[1] Bonci, Andrea, Massimiliano Pirani, and Sauro Longhi (2018) "A database-centric framework for the modeling, simulation, and control of cyber-physical systems in the factory of the future" *Journal of Intelligent Systems* **27** (4): 659–679.

[2] Kennard, Richard and John Leaney (2010) "Towards a general purpose architecture for UI generation" *Journal of Systems and Software* **83** (10): 1896–1906.

[3] Razzaq, Mirza Abdur, Kashif Hussain Memon, Muhammad Ali Qureshi, and Saleem Ullah (2017) "A survey on user interfaces for interaction with human and machines" *International Journal of Advanced Computer Science and Applications* **8** (7): 462–467.

[4] Bonci, Andrea, Sauro Longhi, and Massimiliano Pirani (2021) "IEC 61499 device management model through the lenses of RMAS" *Procedia Computer Science* **180**: 656–665.

[5] Brooks Jr, Frederick P (1995) *The mythical man-month: essays on software engineering (anniversary ed.)* Addison-Wesley Longman Publishing Co., Inc. anniversary edition

[6] Bonci, Andrea, Sauro Longhi, and Massimiliano Pirani (2019) "RMAS architecture for autonomic computing in cyber-physical systems" in "IECON 2019-45th Annual Conference of the IEEE Industrial Electronics Society", volume 1, IEEE, pages 2996–3003.

[7] Bonci, Andrea, Simone Imbrescia, Massimiliano Pirani, and Paolo Ratini (2014) "Rapid prototyping of open source ordinary differential equations solver in distributed embedded control application" in "2014 IEEE/ASME 10th International Conference on Mechatronic and Embedded Systems and Applications (MESA)", IEEE, pages 1–6.

[8] Inductive Automation (2012) "SQL: The next big thing in SCADA. how SQL is redefining SCADA" Available at: https://inductiveautomation.com/resources/article/sql-the-next-big-thing-in-scada

[9] Paton, Norman W (2012) *Active rules in database systems* Springer Science & Business Media

[10] Pirani, Massimiliano, Aldo Franco Dragoni, and Sauro Longhi (2021) "Towards sustainable models of computation for artificial intelligence in cyber-physical systems" in "IECON 2021–47th Annual Conference of the IEEE Industrial Electronics Society", IEEE, pages 1–8.

[11] Pirani, Massimiliano, Andrea Bonci, and Sauro Longhi (2022) "Towards a formal model of computation for RMAS" *Procedia Computer Science* **200**: 865–877.

[12] Bonci, Andrea, Massimiliano Pirani, Aldo Franco Dragoni, Alessandro Cucchiarelli, and Sauro Longhi (2017) "The relational model: in search for lean and mean CPS technology" in "2017 IEEE 15th International Conference on Industrial Informatics (INDIN)", IEEE, pages 127–132.

[13] Imort, Sebastian and Andreas Deuter (2019) "The digital twin theory: designing a test infrastructure using SysML" *Direct Digital Manufacturing in the Context of Industry* **4**: 253–263.

[14] De Morais, Wagner O, Jens Lundström, and Nicholas Wickström (2014) "Active in-database processing to support ambient assisted living systems" *Sensors* **14** (8): 14765–14785.

[15] Harjula, Erkki, Pekka Karhula, Johirul Islam, Teemu Leppänen, Ahsan Manzoor, Madhusanka Liyanage, Jagmohan Chauhan, Tanesh Kumar, Ijaz Ahmad, and Mika Ylianttila (2019) "Decentralized IoT edge nanoservice architecture for future gadget-free computing" *IEEE Access* **7**: 119856–119872.

[16] Pirani, Massimiliano, Andrea Bonci, Alice Cervellieri, and Sauro Longhi (2021) "On the synthesis of Holonic Management Trees" in "2021 26th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)", IEEE, pages 1–4.