Virtualizing DDS middleware: performance challenges and measurements

Rosbel Serrano-Torres, Marisol Garcia-Valls, and Pablo Basanta-Val
Distributed Real-Time Systems Laboratory
Department of Telematics Engineering
Universidad Carlos III de Madrid
Leganés (Madrid) Spain
rosbel.serrano@gmail.com, {mvalls, pbasanta}@it.uc3m.es

Abstract—As new technology becomes available, systems increase in complexity, which in turn, raises the expectations of users for new applications that may presumably be more complex. This spiral process requires the usage of appropriate techniques to control complexity such as decoupled design and development paradigms, and communications middleware that facilitate the development of distributed applications. The highest exponent of them is, currently, DDS middleware (Data Distribution System for Distributed Real-Time Systems) that is specifically designed for applications that have timing requirements. Also, virtualization techniques follow the principle of complexity reduction, and they allow to customize the offered computational platforms and to achieve server consolidation in different domains ranging from industrial control systems, distributed surveillance, or enterprise resource planning applications. This paper describes some considerations for merging real-time middleware, such as DDS, and virtualization technology with the aim of suitting the cyber physical domain. This virtual integration tries to bypass the typical bottlenecks of performance. Performance results track the differences between the executions of DDS middleware in the bare machine compared to a virtualized environment.

Keywords— Cyber Physical Systems; middleware; DDS; virtualization; real-time; performance; distributed systems

I. INTRODUCTION

Software technology is well known for its constant and rapid evolution, which yields to a non-stopping increased complexity. In Cyber Physical Systems (CPS), applications are becoming embedded in the surrounding infrastructure that is tightly coupled to the physical environment, containing large numbers of nodes connected through different network segments, integrated with diverse software technologies and executing on heterogeneous hardware devices.

CPS are large scale distributed real-time embedded systems deeply integrated in the environment that they monitor. They need to integrate a number of new technologies, methodologies, and architectures that enable their creation. Some authors argument they will require a radical paradigm shift to construct them since they are highly complex large-scale systems [1]. The real-time side of CPS poses tremendous challenges in their development process that must reflect precisely the operation requirements with respect to the temporal constraints. It is, then, needed to employ design time tools to foresee the temporal behavior of the system. Temporal predictability restricts the computational cost and complexity of both the business logic (i.e., payload) and the operational logic (i.e., the system software such as the operating system, middleware, and drivers). The biggest threat and challenge to temporal unpredictability comes from the essence of CPS themselves. They are large scale open systems where all the possible situations, system states, transitions, required communication events, etc., are impossible to be known at design time. Therefore, most current approaches provide high performance execution in open systems as a substitute for delivering time-deterministic results. If the designed system is high performance and fast enough to process large amounts of data, the eventual effect can be as if it were real-time, although it does not guarantee time deadlines.

An example of such a CPS system is that of an enhanced intelligent system for firefighters search-and-rescue tasks. Firefighters carry personal devices to collect data from sensors as they enter affected zones. Each individual firefighter’s node has limited information from nearby sensors. To create a more global picture of the situation, all nodes have to collaborate by sharing their locally collected data in real-time. The fusion of data will produce vital information to save lives, including their own. At this point, it is essential to highlight that a CPS is not a distributed system that requires fast communication. A CPS is a computer-based system that can monitor and control very large scale physical environments with real-time constraints.

Therefore, the timing properties are essential in CPS.

The new trend of providing high performance versus real-time execution support points towards a more complex way to manage multiple systems and cross-connected controls. Future research requires providing new solutions probably yielding a paradigm shift including a broader view: open systems of open networked embedded systems [2].

CPS are driven by upcoming capabilities and ever-decreasing costs of computing and communication devices [3]. New technological architectures are coming from the initial distributed embedded systems (DES) and later large scale distributed systems; their theories and algorithms are being integrated, with special consideration for performance and security requirements, which heavily rely on network communication and data management activities. But CPS essential property is real-time, not networking.

The middleware is a fundamental building block for enabling CPS development. Current middleware solutions will have to be enhanced to include extra intelligence that is
lightweight and that contains the essential enabling mechanisms to support run-time temporal predictability in a challenging environment where unexpected and unforeseen events may arrive and must be handled in a time predictable manner. Monitoring is an important means to ensure temporal predictability in these systems. At least, quality of service (QoS) and stability [4] have to be preserved. By QoS it is referred to trading off resources for the quality of the delivered output result [17][18][19][25][28]. Middleware technologies can be designed to embed QoS mechanisms for efficient resource management of real-time distributed systems. Among the most relevant communication middleware technologies, DDS has become the de facto standard for some domains as military. Some other approaches have used it and extended it to enable dynamic configuration of distributed applications with time requirements [13]. However, current technological trends are targeting at virtualized environments to obtain a number of important benefits such as reduced deployment costs, externalized maintenance, and increased server consolidation with the subsequent energy saving. Virtualization technology still threatens the real-time properties of systems, so it is being used typically in general purpose domains with non real-time hypervisors. It is, therefore, highly interesting to evaluate the performance issues of virtualization software to suit real-time systems and, specially, in the distributed domain where other interference sources are present such as the network communications.

This paper aims at integrating real-time support modules with DDS to virtualize it and assess its performance conditions. It is provided an approach to support virtualization resources enhancement from a real-time perspective.

The paper is structured as follows. Real-time systems, an overview of DDS middleware, and an introduction to virtualization are part of section II. It provides the necessary background into the proposal of this paper. Further CPS challenges and considerations are discussed in section III. Section IV presents an initial contribution for virtualizing DDS middleware. An implementation is provided to show results as well under certain performance indicators. Conclusions are provided in section V.

II. BACKGROUND AND STATE OF THE ART

CPS systems continue to require real-time resource management solutions. Those alternatives available for tasks (e.g. [17][26][27][28]) have now evolved for including dynamic execution (e.g. [5][18][16][19][20][23][29]), and have been integrated in real-time communication middleware (e.g. [21][15][14]) essentially for data-centric communication (e.g. [13][14][21][22]).

Data-centric paradigms [12] are gaining momentum and are being widely used for developing distributed real-time systems. The development based on data types is a flexible one allowing a decoupled construction of complex distributed environments. One of the de facto standards used in domains such as avionics is DDS. In it, data is communicated among nodes based on its content and passed across different threads of execution [30] in the different nodes. DDS changes the communication paradigm to permit flexibility, concurrent operation, and the creation, modification and delivery of events (i.e. messages, data) with QoS guarantees. DDS infrastructure is based on the publish/subscribe (P/S) model. In a P/S scheme, publishers generate information to event brokers. Subscribers do not make specific requests, but subscribe to a particular category of events. Event brokers deliver efficiently and reliably published events to interested subscribers [31].

RD DDS [6] presents a DDS approach that handles and shares information within a sensor network. It is a real-time version of DDS that provides the specific architectural features to deal with events requiring service with strict deadlines. Quality of Data (QoD) is an important precision bound introduced for data handling. Communication is improved by reducing the message exchange between publishers and subscribers; clients can run semantic-aware data models locally to compute current and future states.

A DDS architecture is composed of nodes that can work independently as publishers and/or subscribers. Several publishers read data from assigned (nearby) sensors to monitor and update the prediction model. Whenever a change in the model occurs, the new piece of information is transmitted to all subscribers; each will run and update results accordingly. Specific modules are integrated to improve efficiency such as load controller entities that send periodic events with alterations in buffer and CPU load; this helps in dynamically meeting desired QoS at run-time.

The concept of distributed applications is not new, not even for the hard real-time domain. Middleware solutions play a key role in hiding the complexity associated with sharing resources and delivering ever-increasing functionalities over a network, i.e. inclusion of mobility-oriented solutions, services moved to the cloud, in a secure and reliable (i.e., accurate and up-to-date) environment. Performance provisioning for data-intensive systems has higher demands that can be supplied with the appropriate architecture and monitoring design [32]. With virtualization, services are delivered to multiple users with the needed provisioning of execution environments and proper management of distributed resources [33].

Real-time systems are being confronted with large datasets requiring measurable and guaranteed computational processing with limited resources. Although distributed computing has been a natural move for developing systems, DDS is not taking yet full advantage of virtualization optimization services. Reasons for this may be (a) the challenge of understanding, managing, and bypassing hypervisors and operating system software, and (b) the required changes or add-ons impacting architecture, schedulability, recovery, and monitoring capacities. For efficiently managing storage resources and computing power that provides QoS warranty access, development modifications are required [35].

Distribution technology is now following a main principle: autonomy and decentralized control. Middleware must efficiently work as an independent component allowing embedded distributed and networked systems to execute preserving real-time guarantees. Middleware increases
modularity and flexibility at the cost of increasing resource consumption. Middleware designs must be mapped and deployed in resource-constrained environments offering the expected levels of performance. Networked systems suffer the effects of highly dynamic environments that require execution flexibility. In these environments, real-time requirements are especially hard to meet because of variable operational conditions. Thus, adaptation is needed and it must be managed in a non-conservative manner.

The next generation real-time systems, CPS, pose extremely challenging problems in which virtualization can play an important role. RDDS solutions can be extended into a distributed, virtualized solution. Currently and up-to-our knowledge, there are no other virtualized RDDS proposals. Accessing DDS virtually will provide a better resource exploitation to permit extensive data processing and transmission. With a slight modification to RDDS architecture, this paper presents an initial contribution to virtualize DDS communication middleware. The presented approach is in a very preliminary stage, and it is foreseen that it will provide high performance levels.

III. REAL-TIME CONSIDERATIONS ON VIRTUAL PLATFORMS

A middleware is a software layer, which physically resides between client, server applications and services. Context-awareness cruciality is integrated [11] with an application interface to decrease development time and effort. It also simplifies development by providing a uniform view of heterogeneous networks, protocols and OS features, among others [10]. Real-time systems need to use middleware that has essential properties of hiding network details and keeping stability, safety, and performance [7].

A. Differentiation of DES and CPS

CPS systems integrate computing and communication with monitoring, interaction, and/or control of entities in the physical world. Software execution details and timeliness are critical to the operation of CPS [8], whereas DES do not necessarily have temporal requirements. Real-time behavior is essential and inherent to CPS. DES, instead, function correctly with average execution times; that is to say, with no strict guarantees.

CPS encompass DES and go beyond them with a strong focus on stability and temporal guarantees. Technology advancements in network and mobility have currently shifted CPS into embedded systems, limited by the fact that DES are, by definition, highly resource constrained [3]. The typical challenges for development of CPS are:

- Architecture modeling and control. Communication bottlenecks, resource/time penalties, or performance drawbacks are present in certain architectural styles. These must be modeled and assessed partially before execution.
- Time and space scales. To preserve timing properties, temporal and spatial isolation between running entities (and even subsystems) must be considered.
- Uncertainty. Temporal unpredictability is inherited from temporal and spatial limitations and architecture-communication variable performance of large-scale open systems.

- Composition. Network, communication and scheduling of activities to meet temporal constraints.
- Resources. Processor cycles, memory, and the network must be handled in an integrated way.
- Complexity of heterogeneity. Technical problems for composition of heterogeneous nodes and different execution patterns of subsystems (autonomous versus controlled).
- Maintenance of continuous QoS execution levels. Mechanisms to keep the stability of QoS provision are needed.
- Safety, privacy, security. These are key elements for data management in an open networked environment.
- Precise data sending, processing, and intelligent prediction. It allows fast data management.

B. QoS and real-time properties

The different subsystems of a CPS may have different levels of time criticality, which guides the strict progression of time. QoS refers to the negotiation of requirements to build a real-time system fulfilling requested levels of performance and resources [9][19][18][17][20][24]. QoS service levels must be guaranteed under specified load and failure conditions. This entails reservation of resources and usage of admission control techniques to pre-reserve accordingly.

QoS parameters are bandwidth, latency, or jitter. When studying QoS in a data-driven application, this measure can be correlated to Quality-of-Data (QoD). Applications usually can specify network QoS in terms of delay, jitter, message loss, or bandwidth; these requirements are translated into network QoS [7].

Rate control is another open issue for autonomous systems. It defines the pace at which communication endpoints inject packets in the network. A major final limitation is the balance between resources and high-level services; meaning, middleware must be hosted on constrained devices (in terms of computation, memory, power and/or communication) and still must provide these high resource-consuming services.

C. Data Processing

Middleware data can be understood by dividing it into 2 functions: evaluate the system and transform retrieved data following defined specifications. Data belonging to the system itself gathers information to evaluate and monitor the system or application operation. This refers to either: (a) data obtained from system-monitoring activities, or (b) system analysis and testing information. In the latter, detecting faults is crucial, even more regarding real-time environments. Corruption and failures must be detected and tested thoroughly, including asynchronous events. Most testing errors can be categorized as: ordering, synchronization, or interleaving [1].

When a middleware core function is data-centric, retrieval, transformation and visualization are crucial; for example, video or sensor data. Data processing solutions
include compression techniques, analysis algorithms (e.g. semantics), and data mining, among others.

IV. RDDS IN A VIRTUAL ENVIRONMENT

This section proposes to effectively provision resources for DDS in a virtualized domain by extending RDDS. The proposal considers heterogeneous environments and demands for processing and storage resources. It integrates virtualization to facilitate nodes movements, subscribers’ model enquiries, publishers’ updates, and required management of events and QoS levels. In a virtualized environment, the control of the middleware over the system resources is buried by the hypervisor and virtual machine context. Therefore, it is needed that the necessary hooks and modules are integrated to re-gain access to the platform resources dealing with: task management, scheduling of resources, temporal control of communications, timers, and access to system time.

The lower layer runs on the selected Host operating system that will execute the system’s operatives and the manager. This Virtual Machines Manager (VMM) is in charge of controlling all the operations: inputs, outputs, shared resources, availability, privileges, communication, and temporal management between the main system and the virtualized network. The VMM assigns, and possibly hard codes, the specific tasks and activities to the different machines on the virtualization layer. A system administrator determines the number of virtual machines in the system and its specifications concerning shared resources. Each virtual machine has logical system components of its own: memory and processing capabilities, access to the main system, and communication models to other optionally shareable networks. All these resources are dependent on hardware specifications. A virtual machine can manage its own system, communication, and policies integration.

All the virtual machines connect and interact with the controller, Physical to Virtual Controller (PVC). In this proposal, the controller is an extended version of DDS brokers. It has three main functions: (a) distribute tasks and assign activity queues to virtual machines, (b) process and control real-time properties (i.e. jitter, latency, deadlines, and priorities) from received/sent messages, and (c) schedule communications to either publishers or subscribers, or both with specified QoS levels. One of its main functions is QoS management, executed by adapting latency rates and deadlines according to communication exchange with the subscriber. In addition, it updates subscribers in a timely manner when new data from the model is computed or when inquired. The controller processes all messages and communications integrating temporal, transporting, and jitter priorities. It is constantly receiving liveness and communication metrics. Moreover, it schedules and buffers all incoming and outgoing messages prioritizing by predefined parameters, and lifespans/ deadlines.

One device may contain publishers and subscribers, or either one or the other. Publishers gather data from external devices, i.e. sensors, and process it to identify relevant data from the model. They run this model to update whenever new transmissions are received. They communicate with the controller to keep their liveness status and to update QoS requests.

Figure 2 shows a condensed architecture model to highlight communication flows, messages, parameters and incurred delays. New data is sensed and transmitted by publishers; it passes through all the layers until it reaches the base of the system, which should be secured and accessible by other allowed devices and services in the internal or external network. The message packet leaves the publisher with four attributes: the message with data (m), QoS specs (q), temporal requirements (t) and a priority (p). These attributes modify the flow scheme; they are processed at each level to guarantee the correct distribution of data. Communication follows the same path when it goes to the subscriber’s end. At such destination, final information is reviewed and incorporated, if needed.
Proper components’ task division specifies key sharing and communication points. These key points encourage smoother data transitions from layer to layer. Delays occur as messages go across layers. The diagram displays the delay levels acquired along the flow: \( \delta_1 \) is given for DDS handling, \( \delta_2 \) exists due to virtualization layer and \( \delta_3 \) represents the hardware’s delay for packet retrieving. \( \delta_2 \) would be bypassed if the communication skipped virtual machines and connected directly to the hardware queue (over UDP). \( \delta_2 \) appearance is an additional consideration when integrating virtualization in real-time environments. Mentioned communication points and delays should be evaluated to determine Virtual DDS convenience.

The proposed experiment 1) evaluates middleware gains and losses, and 2) contrasts its performance in local and virtual environments; all measures in microseconds (\( \mu s \)). The three data-centric applications retrieved round-trip times of different size messages transmitted between 2 parties, P/S. Implementations were tested under two profiles: network and CPU intensive. Message size increments and data processing tasks in DDS applications increase temporal requirements proportionally; even though, CPU-intensive executions degrade performance only marginally. The use of middleware solutions outperforms ICMP communications on half the studied categories (greater gains shown on achieved minimums); on the remaining ones, the difference is only \(~0.5x\) low-level service execution. Specific conditioning may benefit from these response times.

Adding a virtualization layer will, in general, consume more resources: time in the presented scenarios (as shown in Table I results). Virtualizing communications moves average times to a range of 1 – 6 ms for 512B messages. This round-trip performance may limit its usage in hard real-time applications, while still offering an alternative for soft real-time. Under certain conditions, ICMP messages provide better behaviors when virtualized; this is also true for the average performance of one DDS middleware (RTI). The observed worst-case cost for virtualization, 3.5x, offers an acceptable solution. It’s also noticeable that, even if slight, there are some time benefits when running on virtual environments. Hence, the presented approach can be accepted as well as the remaining challenges and open items pointing towards a deeper understanding of virtualization effects to identify the most beneficial circumstances.

### Table I. Additional Resource Consumption in DDS and Low-Level ICMP Implementations When Adding a Virtualization Layer

<table>
<thead>
<tr>
<th></th>
<th>Min. times</th>
<th>Average times</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>CPU – intensive</td>
<td>CPU – intensive</td>
</tr>
<tr>
<td>Network</td>
<td>Network</td>
<td>Network</td>
</tr>
<tr>
<td>OSPL</td>
<td>+9.21x</td>
<td>+2.1x</td>
</tr>
<tr>
<td>RTI</td>
<td>+1.84x</td>
<td>-0.25x</td>
</tr>
<tr>
<td>ICMP packets</td>
<td>+1.01x</td>
<td>+1.53x</td>
</tr>
</tbody>
</table>

The use of middleware solutions outperforms ICMP communications on half the studied categories (greater gains shown on achieved minimums); on the remaining ones, the difference is only \(~0.5x\) low-level service execution. Specific conditioning may benefit from these response times.

Adding a virtualization layer will, in general, consume more resources: time in the presented scenarios (as shown in Table I results). Virtualizing communications moves average times to a range of 1 – 6 ms for 512B messages. This round-trip performance may limit its usage in hard real-time applications, while still offering an alternative for soft real-time. Under certain conditions, ICMP messages provide better behaviors when virtualized; this is also true for the average performance of one DDS middleware (RTI). The observed worst-case cost for virtualization, 3.5x, offers an acceptable solution. It’s also noticeable that, even if slight, there are some time benefits when running on virtual environments. Hence, the presented approach can be accepted as well as the remaining challenges and open items pointing towards a deeper understanding of virtualization effects to identify the most beneficial circumstances.

### A. Initial DDS virtualization approach

Major concerns towards the feasibility of the proposed Virtual DDS mechanism in real-time systems can be appraised with a data-latency evaluation. This approach measures the performance of virtualized DDS: OpenSplice, RTI and a low-level service application using ICMP (Internet Control Message Protocol). The setup of this experiment, as seen in Figure 3, involved two physical machines; each configured with a virtual machine (VM) managed by VirtualBox. Each of the totaling 4 machines had a unique IP address and had installed the data distribution service systems. The communication exchange occurred between 2 machines executing at OS-level (bare machine), sized up to the execution of 2 virtual machines (at different hosts). On average, middleware virtualization can reduce temporal costs by \(~20\%) or can represent a temporal increase of up to 3.5x (as explained below & seen in Table I).

### Table I. Additional Resource Consumption in DDS and Low-Level ICMP Implementations When Adding a Virtualization Layer

<table>
<thead>
<tr>
<th></th>
<th>Min. times</th>
<th>Average times</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>CPU – intensive</td>
<td>CPU – intensive</td>
</tr>
<tr>
<td>Network</td>
<td>Network</td>
<td>Network</td>
</tr>
<tr>
<td>OSPL</td>
<td>+9.21x</td>
<td>+2.1x</td>
</tr>
<tr>
<td>RTI</td>
<td>+1.84x</td>
<td>-0.25x</td>
</tr>
<tr>
<td>ICMP packets</td>
<td>+1.01x</td>
<td>+1.53x</td>
</tr>
</tbody>
</table>
controlled and integrated structure simplifies the identification of processes and components. The proposed virtual integration to DDS systems can be achieved with minimum modifications to existing applications. This mechanism can be further studied and leaves several open points to be extended.

ACKNOWLEDGEMENT

This work has been partly supported by the Spanish national project REM4VSS (TIN 2011-28339), Madariaga Grant Programme for International Research Stays from the Spanish Ministry of Education (PRX12/00252), and eMadrid (S2009/TIC-1650).

REFERENCES


