Modeling, implementation, and analysis of XRCE-DDS applications in distributed multi-processor real-time embedded systems

Saeid Dehnavi, Dip Goswami, Martijn Koedam, Andrew Nelson, Kees Goossens
Department of Electrical Engineering, Eindhoven University of Technology, The Netherlands
{S.Dehnavi, D.Goswami, M.L.P.J.Koedam, A.T.Nelson, K.G.W.Goossens}@tue.nl

***Abstract*—The Publish-Subscribe paradigm is a design pattern for transparent communication in many recent distributed applications. Data Distribution Service (DDS) is a machine-to-machine communication standard that aims to provide reliable, high-performance, inter-operable, and real-time data exchange based on publish–subscribe paradigm. However, the high resource requirement of DDS limits its usage in low-cost embedded systems. XRCE-DDS is a Client-Agent based standard to enable resource-constrained small embedded systems to connect to the DDS global data space. Current XRCE-DDS implementations suffer from dependencies with host operating systems, target only single processing units, and lack performance analysis methods. In this paper, we present a bare-metal implementation of XRCE-DDS standard on the CompSOC platform as an instance of Multi-Processor System on Chip (MPSoC). The proposed framework includes a hard real-time side hosting the XRCE-DDS Client, and a soft real-time side hosting the XRCE-DDS Agent. A Scenario Aware Data Flow (SADF) model is proposed to capture the dynamism of the system behavior in terms of different execution scenarios. We analyze the long-term expected value for throughput by capturing the probabilistic scenario switching using a proposed Markov model which is experimentally validated.**

***Keywords*-Real-time Systems, Multi-processor, Data Distribution Service (DDS), XRCE-DDS, Distributed Embedded Systems**

I. Introduction

Data Distribution Service (DDS) [1] is a machine-to-machine communication standard that aims to provide reliable, high-performance and inter-operable data exchange based on the publish–subscribe paradigm. DDS is used as the core communication layer for Robot Operating System 2 (ROS2) [2]. Despite some lightweight implementations for DDS, the memory footprint is still too large to fit in resource-constrained embedded systems. This issue has been addressed by the Agent-Client based standard XRCE-DDS (DDS for eXtremely Resource-Constrained Environments) [3]. In this standard, the more powerful Agent nodes act as a bridge to connect resource constrained Client nodes to the DDS data space. Current lightweight DDS implementations such embeddedRTPS [4], and XRCE-DDS implementations such as eProsima version [5], suffer from dependencies with host operating systems, target only single-processor embedded systems, and ignore timing considerations. The proposed DDS performance analysis approaches such as [6] and [7] suffer from considering real-time properties, and lack a formal analysis approach respectively. Moreover, to the best of our knowledge, no modeling and performance analysis framework exists for XRCE-DDS applications in distributed real-time embedded systems.

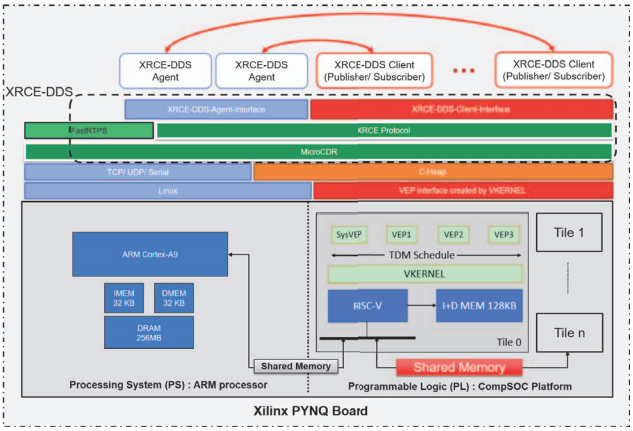

Fig. 1. XRCE-DDS implementation on PYNQ-Z2 Xlinix ZYNQ board

Contributions: Our contributions in this paper are:

1) A bare-metal implementation of the XRCE-DDS standard on the CompSOC [8] platform on a ZYNQ board. The Client is implemented on a hard real-time RISC-V processor, while the Agent runs on a soft real-time ARM processor. Multiple RISC-V processors run in parallel and each processor runs multiple Clients. (Section II)
2) Probabilistic performance modeling of XRCE-DDS applications with Scenario Aware Data Flow (SADF) [9]. A Markov model is proposed to analyze expected throughput in the long-term. (Section III)
3) Experimental Performance analysis of XRCE-DDS applications on a network of CompSOC platforms (Section V)

II. XRCE-DDS Implementation on CompSOC

There are various implementations of CompSOC on ASIC and on FPGA. In this paper we use the Verintec [10] implementation that follows the CompSOC concepts. We use a Xilinx PYNQ-Z2 board with a ZYNQ XC7Z020 that contains a processing system (PS) with an ARM Cortex-A9 (650MHz) running Ubuntu Linux, and programmable logic (PL). In this instance, the PL contains 3 predictable RISC-V cores (40MHz, 128KB memory) and a 16KB dual-port shared memory between each pair of (RISC-V, ARM) cores, as shown in Fig. 1. The VKERNEL [11] creates predictable and composable virtual execution platforms (VEP) [8] on the RISC-V cores through spatial and cycle-accurate temporal partitioning. Applications see a bare-metal interface and can use the C-HEAP FIFO library [12] to communicate on or between any pair of processors using the shared memories. The performance of C-HEAP is modelled using dataflow [13].

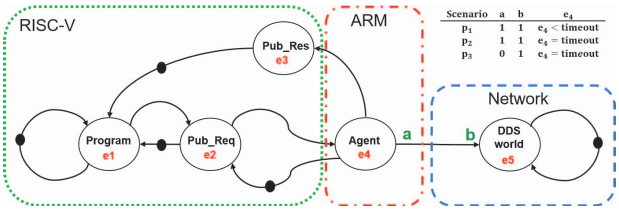

Fig. 2. SADF model of the publisher side

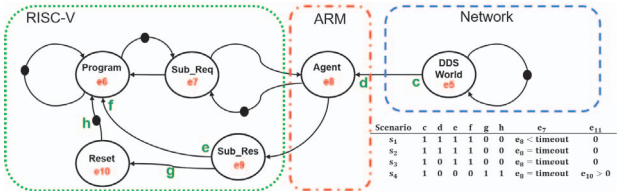

Fig. 3. SADF model of the subscriber side

A serialization layer (MicroCDR) is used to serialize/deserialize the messages between Clients and Agent. The main functionality of the XRCE-DDS standard is implemented in the XRCE Protocol layer that is responsible for message interpretation, error handling, synchronization, and sequence handling. While the XRCE-DDS Client Interface is only dependent to XRCE Protocol, XRCE-DDS Agent Interface is dependent on both XRCE Protocol and FastRTPS. This is because the Agent is responsible to forward messages between the Clients and other entities in the DDS world, which is done by FastRTPS as a light weight implementation of standard DDS. At the top layer, RT applications are developed through XRCE-DDS Client APIs on the MPSoC side, and the Agent has to implement XRCE-DDS Agent APIs. Each Client runs in its own VEP on a single RISC-V tile, without any interference by other (client) applications on the same or other tiles.

III. SADF Modeling for XRCE-DDS applications

The reason behind using SADF as the formal Model of Computation (MoC) in this paper is twofold: i) dynamic behaviour of the system can be modeled by different system scenarios which enable us to model variation in response time, message loss, and failure situations. ii) Efficient timing analysis by keeping as much as possible of the determinism in each scenario. By modeling and analysing the system behaviour, we want to show that: 1) The Clients (Publisher/Subscriber) on the RISC-V processor experience guaranteed Worst-Case Response Time (WCRT), 2) The Agents on SRT part experience a probabilistic behaviour in terms of response time and token loss while communicating through Non-RT network. A SADF Graph (SADFG) is defined by a tuple (Σ, M) where:

- $\sum = \{s_i | s_i = (A, C, e, r, i)\}$ is the set of system scenarios, each of which is modeled by a Synchronous Data Flow Graph $s_i = (A, C, e, r, i)$, where A is the set of actors, $C \subseteq A \times A$ is the set of channels between actors, $e : A \to \mathbb{R}_{\geqslant 0}$ is a function that assigns the WCRT of each actor in a specified scenario, $r : A \times C \to \mathbb{N}_{\geq 0}$ defines the production/consumption rate of the channels assigned to each actor and $i : C \to \mathbb{N}_{\geq 0}$ assigns the initial tokens on each channel. An actor immediately fires if its required tokens are available on the input channels. A number of tokens (specified by the production rate) is released to the output channel in each firing.
- A Sequence of scenario executions is modeled by a Markov chain $M = (S, p, \phi)$ in which S is the set of system scenarios, $p : S \times S \to [0, 1]$ returns the likelihood of switching between scenarios, and $\phi : S \to \mathbb{R}_{\geq 0}$ is a reward function on the scenarios that is used for the long-term performance analysis.

A. Publisher SADF Model

The SADF model of the publisher is shown in Fig. 2. In this model, Program is a RT program that uses the XRCE-DDS Client interface (Pub_Req and Pub_Res actors) to publish its messages to the DDS world at the end of each iteration. This is done by sending a request from XRCE-DDS Client (Pub_Req actor) to a specific XRCE-DDS Agent (Agent actor). After processing the XRCE-DDS request by the Agent, a response is returned to the Client (Pub_Res actor) to confirm if the message is successfully published to the DDS world. Since the Agent runs on the SRT Linux, it experiences variation in the response time. For this reason, we model different execution scenarios for the publisher by the following SADF scenarios:

- **P1 Common Case:** The DDS token is published quickly and much before the timeout (i.e $e_4 < timeout$).
- **P2 Slow Case:** The DDS token is published but in a longer time (we assume at the timeout), i.e $e_4 = timeout$.
- **P3 Failure to publish** If the successful response is not received from the Agent before $timeout$, Pub_Res lets the RT program to continue its work. In this scenario, no message is published to the DDS world. Therefore, the production rate (a) for the Agent actor is zero.

The throughput of the RT program in the publisher side is calculated by $\frac{1}{e_1+e_2+e_3+e_4}$ in different publisher scenarios.

B. Subscriber SADF Model

As shown in Fig. 3, Program is a RT program that executes its functionality on the messages that it receives from DDS world. It requests and receives DDS messages from a specific Agent (Agent actor) through a XRCE-DDS subscriber (Sub_Req and Sub_Res actors). We have 4 SADF scenarios:

- **S1 Common Case:** The DDS token is received quickly and much before the $timeout$ ($e_8 < timeout$).
- **S2 Slow Case:** The DDS token is received but in a slow time (we assume at the timeout), i.e $e_8 = timeout$.
- **S3 Tolerable Failure:** The response is not received after the $timeout$. Based on the failure management techniques for stream processing and the XRCE-DDS history parameter, we keep a buffer of the last n received messages to let the program continue its work in this situation. Since there is no message received from the DDS world, $d = 0$ and $c = 1$.
- **S4 Failure and Reset:** After n successive failures, the connection should be reset by a failure management task that takes e_{10} as the response time.

The throughput of the RT program in the subscriber side is calculated by the equation $\frac{1}{e_6+e_7+e_8+e_9+e_{10}}$ that changes by variation of e_8 and e_{10} in different modeled scenarios.

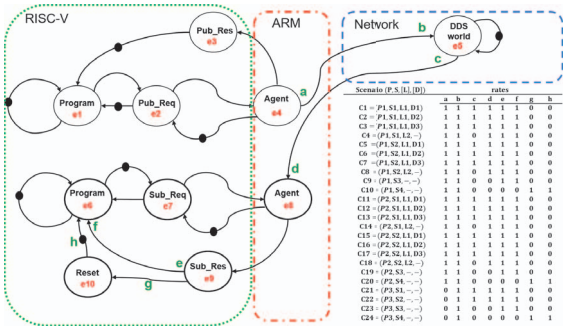

Fig. 4. Application SADF model

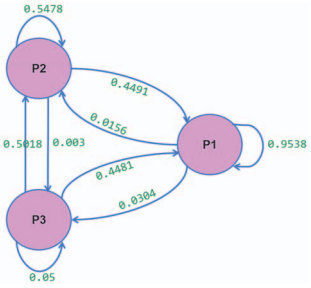

Fig. 5. Publisher Markov model

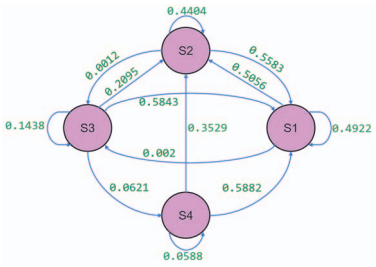

Fig. 6. Subscriber Markov model

C. Combined SADF Model

For end-to-end application-level performance analysis we now combine the RT publisher & subscriber client models with a model of the SRT Agents and their communication. Agents run on the SRT linux and use SRT [14] or Non-RT network protocols such as TCP and UDP. The actor DDSWorld models the FastRTPS overhead and the network delay for communication between the Agents. Based on our experiments on a network of Pynq boards, the response time (e_5) for DDSWorld experiences a variation of 3 values $[D1:1ms,\ D2:2ms,\ D3:(>3ms)]$ with probability $[D1:0.6,\ D2:0.3,\ D3:0.1]$. Data loss in the DDS world is modeled with SADF scenarios with a production/consumption rate of zero (b, c). The variation in data loss in DDS World is modeled by two scenarios $[L1(Lossless):0.98,\ L2(Lossy):0.02]$. The complete SADF of an application is the product of its publishers, subscribers, and their corresponding agents. For a single publisher and subscriber this results a model with 24 scenarios (Fig. 4).

IV. Long-term throughput analysis

As mentioned in section III, the switching between execution scenarios is modeled by a Markov chain $M=(S,p,\phi)$ shown in Fig. 5 and Fig. 6 for the publisher and subscriber respectively. Since the throughput of each scenario of the SADF model for publisher and subscriber can be calculated independently, we assign the calculated value as the reward function of the specified Markov models. Since our Markov models are Ergodic and Unichain we compute the long-term expected throughput with $\lim_{n\to\infty}E(r_x))=\pi^{\infty}r^T$ [15] where the long-term stationary distribution (π^{∞}) is calculated by:

$$\pi^{\infty}=\lim_{n\to\infty}\frac{1}{n}\sum_{k=0}^{n-1}\pi^0 p^k=\pi^0 p^{\infty} \qquad (1)$$

The expected long-term throughput of the application is calculated using a Markov model on different scenarios of the application SADF graph. The Markov model for the SADF in Fig. 4 contains 24 different nodes (scenarios) and the transition probability ($P(c_i,c_j)$) between two scenarios is:

$$P(c_i,c_j)=P(c_i^p,c_j^p)\times P(c_i^s,c_j^s)\times P(c_i^l,c_j^l)\times P(c_i^d,c_j^d) \qquad (2)$$

where, $c_i(p,s,l)$ is a scenario of the SADF model in Fig. 4, $P(c_i^p,c_j^p)$ and $P(c_i^s,c_j^s)$ are valued from the Markov models of the publisher and the subscriber in Fig. 5 and Fig. 6 (based on an experiment over 20 million iterations). $P(c_i^l,c_j^l)$ and $P(c_i^d,c_j^d)$ are based on the message loss probability and DDS-World response time variation presented in Section III-C. We calculate the throughput of each application scenario using SDF3 [16] tool and assign it as the reward function of the application Markov model. The expected long-term throughput for the application is then calculated by Eq. 1 similar to the publisher/subscriber expected throughput. For our analysis approach, we need the WCRT of each actor in the SADF models. We will measure the WCRT values in Section V.

V. Performance Evaluation

In this section we first measure the required parameters such as WCRT of each component. Then, we use the measured values for the long-term throughput analysis. We measure the WCRT of each component by the CompSOC global timer that is synchronized for all RISC-V tiles.

Publisher measurements: The measured WCRT of the components involved in the actor Pub_Req in the publisher SADF model (Fig. 2) for 20K publications with different message size is reported in Fig. 7. The WCRT time for the first two components are constant per message size, while there is some jitters on the *flashout* time that comes from different conditional statements in the source code. We also measured the WCRT for the Pub_Res actor that is constant per message size. We measure the Agent response time (Agent actor in Fig. 2) by the time different between sending the request and receiving the response in the PL side. Since the Agent runs in the PS side, the Agent response time experiences variation. This will be addressed by the *timeout* effect in the following. We use a histogram on the Agent response time to define the WCRT of the Agent actor in different SADF scenarios.

Subscriber measurements: The actor Sub_Req in the subscriber SADF model (Fig. 3) includes 2 components *Request Preparation* and *Request Flashout*. Moreover, the actor Sub_Res includes the component *deserialization* on the received message from the Agent. Unlike the publisher, in which *buffer preparation* is done per each iteration, in the subscriber side it is executed only once. Therefore, it can be ignored in the long-term experiments.

Timeout effect on WCRT: As it was mentioned in Section III, RT Program in both publisher and subscriber sides should continue its work if the Agent response is not received after a specified *timeout*. This helps to guarantee the WCRT of the RT program. To evaluate the validity of this parameter, we added

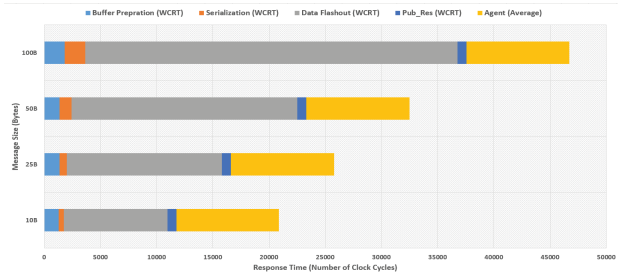

Fig. 7. Publisher measurements per message size

random bad requests in the publisher side to make the Agent take long time. As shown in Fig. 9, by hitting the *timeout*, the Client ignores the Agent response and the response time never goes higher than the defined *timeout* (2 milliseconds). The same experiment validates the *timeout* effect in the subscriber side. We believe that the measured WCRT for different actors are valid considering predictability and composability concepts guaranteed in the clock cycle level by the CompSOC platform.

A. Long-term analysis:

We compute the expected throughput in the long-term through the proposed Markov models in section IV. We need to find the transition probability of switching between different scenarios in the SADF models of the publisher and subscriber. For this purpose, we conducted an independent experiment of 2 million iterations on the publisher and the subscriber. Then, by writing a Matlab script that considers the probability distribution of the experimented values, we calculated the required transition probabilities that are reported in Fig. 2 and Fig. 3 for the publisher and the subscriber respectively. To check if the implementation conforms to the model, we created a dummy RT program with the WCRT of 25K clock cycles. We measured the throughput for the created RT program on publishing 2 million messages ranging from 10 Bytes to 100 Bytes. Moreover, we computed the throughput in each scenario of the publisher and assigned it as the reward function for the specified scenario in the publisher Markov model. Then we computed the expected throughput in the long-term using Eq. 1 on the publisher Markov model (Fig. 5). The same approach was applied to the subscriber side. We calculated the application expected long-term throughput using the approach discussed in Section IV. We also conducted some experiments to measure the experimental application throughout. As shown in Table I, the implementation throughput conforms to the expected throughput computed from the Markov models of the publisher, subscriber, and the application.

	Publisher TP		**Subscriber TP**		**Application TP**	
Message	Exp.	Exptl.	Exp.	Exptl.	Exp.	Exptl.
10B	0.8242	0.8464	0.6066	0.6284	0.4629	0.5288
25B	0.7446	0.7779	0.5253	0.5691	0.4291	0.4417
50B	0.6591	0.7079	0.4686	0.5148	0.3989	0.4362
100B	0.5321	0.5641	0.4268	0.4526	0.3626	0.4167

TABLE I
EXPECTED (EXP.) AND EXPERIMENTAL (EXPTL.) THROUGHPUT (TP) IN NUMBER OF MESSAGES PER MILLISECONDS

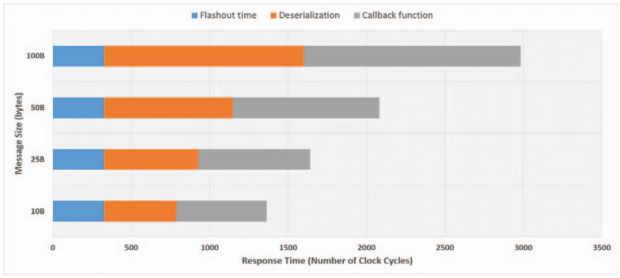

Fig. 8. Subscriber Measurements per Message size

VI. CONCLUSION

In this paper, we proposed a bare-metal implementation of the XRCE-DDS standard on the CompSOC platform. The dynamic behaviour of the system was modeled by SADF. We analyzed the expected long-term throughput by a proposed Markov model, and the experimental results on a network of Pynq-Z2 boards shows that our implementation conforms to the proposed analysis approach. ECSEL JU grant agreement No 826610 (COMP4DRONES) supported this work.

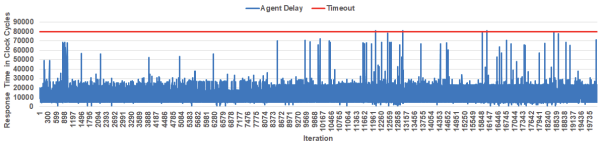

Fig. 9. Timeout effect in the publisher per number of clock cycles

REFERENCES

[1] G. Pardo-Castellote, "OMG Data Distribution Service: Architectural overview," in *ICDC*, 2003.
[2] M. Quigley *et al.*, "ROS: an open-source Robot Operating System," in *ICRA*, 2009.
[3] Object Management Group, "DDS for eXtreamly Resource-Constrained Environments (XRCE-DDS)," 2019.
[4] A. Kampmann *et al.*, "A portable implementation of the real-time publish-subscribe protocol for microcontrollers in distributed robotic applications," in *ITSC*, 2019.
[5] J.Bermúdez and B.Outerelo and others, "Micro-XRCE-DDS," in *https://github.com/eProsima/Micro-XRCE-DDS*, 2019.
[6] Y. Liu *et al.*, "Formal analysis and verification of dds in ros2," in *MEMOCODE*, 2018.
[7] K. Krinkin *et al.*, "Data distribution services performance evaluation framework," in *FRUCT*, 2018.
[8] K. Goossens *et al.*, "NoC-Based Multiprocessor Architecture for Mixed-Time-Criticality Applications," in *Springer*, 2017.
[9] T. Basten *et al.*, "Scenarios in the design of flexible manufacturing systems," in *System-Scenario-based Design Principles and Applications*, Springer, 2020.
[10] In, *https://verintec.com*.
[11] A. Nelson *et al.*, "Comik: A predictable and cycle-accurately composable real-time microkernel," in *DATE*, 2014.
[12] A. Nieuwland *et al.*, "C-HEAP: A heterogeneous multi-processor architecture template and scalable and flexible protocol for the design of embedded signal processing systems," *DAES*, 2002.
[13] A. Nelson *et al.*, "Dataflow formalisation of real-time streaming applications on a composable and predictable multi-processor SOC," *Journal of Systems Architecture*, 2015.
[14] H.-Y. Choi *et al.*, "Making DDS really real-time with OpenFlow," in *EMSOFT*, 2016.
[15] N. Privault, "Understanding markov chains," *Examples and Applications, Springer*, 2013.
[16] S. Stuijk *et al.*, "SDF^ 3: SDF for free," in *ACSD*, 2006.