

Efficient Embedded Runtime Systems through Port Communication Optimization

Peter H. Feiler

Software Engineering Institute, Carnegie Mellon University
phf@sei.cmu.edu

Abstract

Traditionally shared data areas have been used to efficiently communicate between embedded system tasks, such as periodically executing control system tasks. Such implementations are highly sensitive to the execution order of tasks, i.e., they depend on a static timeline. SAE AADL supports a port communication model that ensures deterministic processing of signal streams. In this paper we discuss an analytical framework that allows us to optimize such port-based communication by generating a runtime executive that utilizes shared data areas where appropriate, while ensuring the timing semantic assumed by the control application.

1. Introduction

Traditionally embedded real-time software was executed by a cyclic executive. This executive was invoked periodically as result of a clock interrupt and made sequential calls to different application functions at the desired rate. Different application functions communicated with each other through shared data areas and the execution order of the functions determined the flow of data. The static timeline of the cyclic executive ensured that the signal stream of the control system was processed deterministically with minimal latency jitter as control systems are sensitive to latency jitter.

In recent years, real-time application development has used pre-emptive fixed priority or dynamic priority scheduling to improve resource utilization and architectural flexibility. Scheduling analysis such as Generalized Rate Monotonic Analysis (GRMA) [1] ensures that deadlines are met. However, when combined with shared data area communication latency jitter of multiples of task periods results in instability of the control algorithms.

Modeling tools such as MathWorks Simulink support port-based system modeling. These port connections are mapped into variables in a global data

area. This is illustrated in Figure 1. In the generated application code the variables are accessible through the structure rtB.

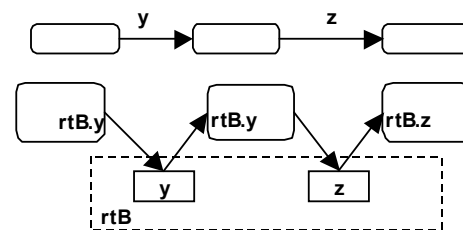


Figure 1 Shared buffer communication

Simulink blocks may operate at different sampling rates. A block executing at a slower sampling rate may down-sample the output from a higher rate block and vice versa. In order to get predictable execution behavior of code generated from Simulink models, application developers are required to insert rate transformation blocks.

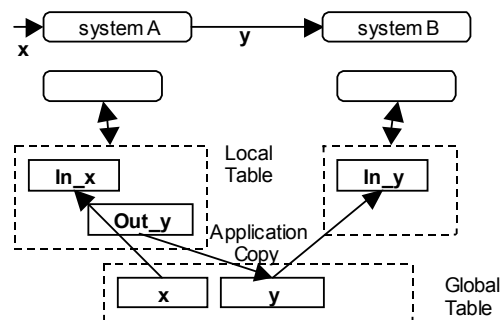


Figure 2 Local and global buffers

A port-based object (PBO) framework [2] was developed to support the development of configurable real-time software. This framework represents port automata with an algebra of concurrent processes [3] and has been shown to achieve stable control systems in the Robotics domain [4]. Application developers write application code as PBOs without concern of how it interacts with other application components. These ports are kept in state variable tables local to the PBO, thus, their content is not affected by other components. Only upon completion is the port content

transferred to and from a global state variable table. That transfer is executed atomically as part of the application task. This communication model is illustrated in Figure 2.

The SAE AADL [5] supports the PBO model through unqueued data ports and queued message ports. Data ports can be viewed as local port variables that are accessible to application functions. Application functions are dispatched by a runtime executive and the runtime executive is responsible for transferring the data between the port variables according to port connection specifications.

In AADL data port communication between periodic threads can be mid-frame or phase-delayed in order to ensure determinism. In AADL the communication code is part of the runtime executive together with task dispatch code. This ensures that the time of transfer between the ports is well-defined and deterministic with respect to sampling of the data stream. While the application code operates on the content of port variables, system buffers may have to be used to ensure that their content is not affected by other tasks, while a task performs its computations.

In the next section we introduce a message buffer lifespan model to determine the necessary number of message buffer copies and the impact of the different message communication implementations.

2. A Message Buffer Lifespan Model

The lifespan model of unqueued message communication is used to determine how many message buffers and data transfers are necessary. Starting with a full message buffer model as shown in Figure 3 we use the message buffer lifespan concept to determine whether message buffers can be combined and data transfers eliminated without affecting data integrity and determinism of message communication. We also use this model to determine whether the runtime system produces the desired timing semantics of signal stream processing.

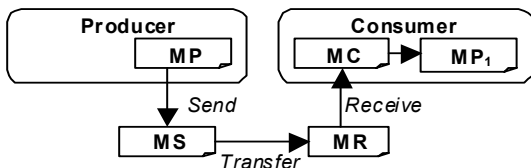


Figure 3 Full message buffer model

In a full message buffer model the producer of a message operates on a local copy (MP). A send operation copies the message into a system message send buffer (MS). The message is then transferred to a system message receive buffer (MR). A receive operation copies the message to a local message buffer of the consumer task (MC). The consumer task then

operates on MC and places the results of its processing in a local output message buffer MP_1 .

Each of these message buffers has a lifespan for each message instance. The producer generates a message instance by placing the message content into MP. It can do so anytime during its execution after a dispatch. The lifespan of message instance i in MP begins with the execution of P, i.e., $T_{P,M_i} \leq \alpha_{P,M_i} = B_{MP_i}$. The lifespan ends with the completion of the send operation S or with the next execution of P, whichever comes first, i.e., $T_{P,M_i} \leq B_{MP_i} \leq E_{MP_i} = S_{M_i} \leq T_{P,M_{i+1}}$. The lifespan for MC is similarly bounded by consumer deadline and the receive operation R, i.e., $T_{C,M_i} \leq R_{M_i} = B_{MC_i} \leq E_{MC_i} = \Omega_{C,M_i} \leq D_{C,M_i} \leq T_{C,M_{i+1}}$. The lifespan of MS is bounded by successive send operations and the completion of the transfer operation X, i.e., $S_{M_i} = B_{MS_i} \leq X_{M_i} \leq E_{MS_i} < S_{M_{i+1}}$. The lifespan of MR is bounded by successive transfer operations and the completion of the receive operation, i.e., $X_{M_i} \leq B_{MR_i} \leq E_{MR_i} = R_{M_i} < X_{M_{i+1}}$.

The lifespan of these message buffers is graphically illustrated in Figure 4. The figure shows the lifespan of each buffer for two message instances. The dashed boxes show the potential lifespan of MP and MC bounded by the dispatch time and deadline of the task. The dispatch times for the two tasks and for the transfer operation are shown as independent. The message send and receive operations are shown as being executed as part of the application code.

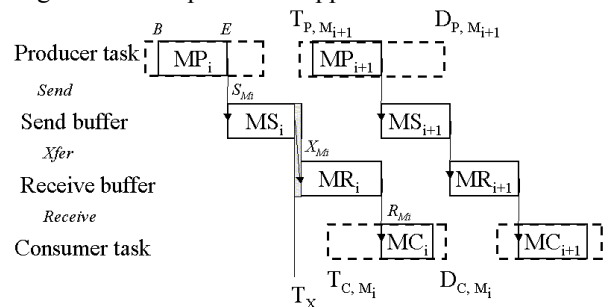


Figure 4 Message buffer lifespan model

The message buffer lifespan model supports representation of periodic message communication (PMT), immediate message transfer (IMT), and direct message transfer (DMT).

In case of PMT message transfer between the send buffer and the receive buffer is initiated independent of the producer and consumer tasks – supported by the four buffers, as shown in Figure 4.

In case of IMT the transfer is performed together with the send operation, i.e., we have a three buffer communication model, as shown in Figure 5.

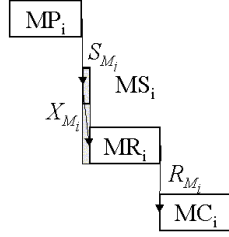


Figure 5 Immediate message transfer

In case of DMT upon completion of the producer task the message is transferred directly to the consumer task, i.e., the message content is copied directly from MP to MC resulting in a two buffer communication model.

The send and receive operation may be invoked by the application code (ASR) or performed as part of task dispatching in a runtime executive (DSR).

In case of ASR, the time, at which the send operation from MP to MS and the receive operation from MR to MC occurs, varies depending on the priority of the producer and consumer task and possible preemption by other tasks. For send we have, given $T_P \leq \alpha_P \leq S \leq \Omega_P \leq D_P$ and for receive $T_C \leq \alpha_C \leq R \leq \Omega_C \leq D_C$. The lifespan of MP and MC are characterized by $\alpha_P - \Omega_P$ and $\alpha_C - \Omega_C$ respectively.

In case of DSR, the send operation to MS occurs at the deadline of the producer task, and the receive operation occurs at the dispatch of the consumer task, i.e., $S = D_P$ and $R = T_C$. The lifespan of MP and MC are characterized by $\alpha_P - D_P$ and $T_C - \Omega_C$ respectively.

Message communication is characterized by data integrity and by the communication semantics of message instances. Data integrity assures that the message content is communicated consistently, i.e., the message buffers contain the message content of a single message instance. Communication semantics of message instances can be deterministic, i.e., assure that consumers receive a sequence of message instances in a deterministic order. We will examine each in turn.

3. Data Integrity

Data integrity is affected by concurrent read/write or write/write access to message buffers. Producers and consumers can operate independently without affecting integrity of the message data as they operate on local copies. Read/write concurrency of MP is managed by the fact that the producer task performs write access to MP and issues the send operation, which performs read access. Read/write concurrency of MC is managed by the fact that the consumer task performs read access to MC and issues the receive

operation, which performs write access. Note that a consumer task may use MC as working memory and overwrite its content.

The system message buffer MS encounters concurrent write/write access if multiple producer tasks on the same processor send to the same port, and concurrent read/write access if the transfer operation is initiated independent of the send operation, as in case of periodic transfer. MR concurrent write/write access to MR is possible if the in port of a consumer is connected to multiple producer out ports, and concurrent read/write access is possible due to independence of transfer and receive operations.

Data integrity of MS and MR is achieved by atomic send, transfer, and receive operation. This atomicity can be assured through a variety of mutual exclusion mechanisms. In case of distributed transfer to another processor the duration of the transfer operation may be longer. In this case the mutually exclusive access time for MS and MP can be reduced by treating the transfer like a task, i.e., by copying the message data into a local buffer in the message transfer layer.

In case of IMT, we have three buffer communication, MP, MC, and a single system buffer MSR. Read/write concurrency of MSR is coordinated through atomic send and receive operations.

In case of DMT write/write and read/write concurrency may occur if the execution of the producer task and the consumer task overlaps. The send operation of the producer task may overwrite the content of MC, while the consumer uses it, i.e., data integrity is violated if $\alpha_C = B_{MC} \leq S = \Omega_P \leq E_{MC} = \Omega_C$.

Data integrity is affected by attempts to reduce message buffers.

- MR and MC can be combined in case of PMT if a transfer operation does not occur during the lifespan of MC while the consumer accesses MC, i.e., $\neg (B_{MC} \leq X \leq E_{MC})$.
- MSR and MC can be combined in case of IMT if a send operation does not occur during the lifespan of MC while the consumer accesses MC, i.e., $\neg (B_{MC} \leq S \leq E_{MC})$.
- MP can be combined with MS (in case of PMT) if the transfer operation does not occur during the lifespan of MP. The read access of transfer operation is not at the same time as the write access of the producer, i.e., $\neg (B_{MP} \leq X \leq E_{MP})$.
- MP can be combined with MSR (in case of IMT) if $\neg (B_{MP} \leq S \leq E_{MP})$.
- MP, MS, MR, and MC can be combined for PMT (MP, MSR, and MC for IMT, and MP, MC for

DMT) if the lifespan of MP and MC do not overlap, i.e., the producer and the consumer do not have write and read access at the same time, i.e., $B_{MP} - E_{MP} \cap B_{MC} - E_{MC} = \emptyset$.

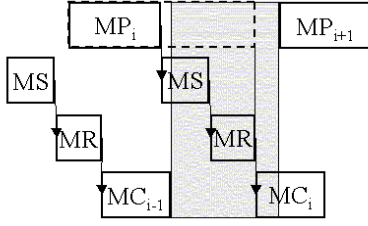


Figure 6 Required message copy operation

The minimum number of message buffers is determined by how many message instances are actively being processed by the producer and consumer. In terms of lifespan of message instances, this is the smallest k st. $E_{MC_i} < B_{MP_{i+k}}$.

We can also determine which message operation must be performed a message content copy. This is determined by the overlap of a message instance lifespan with the lifespan of specific message buffers, as illustrated in Figure 6. In the example two message buffers are required ($E_{MC_{i-1}} < B_{MP_{i+1}}$) and the copy operation for message instance i must occur after $E_{MC_{i-1}}$ and before $B_{MP_{i+1}}$, i.e., either be performed as part of transfer or receive.

4. Communication Semantics

Determinism of port communication is affected by the order in which send, transfer, and receive operations are executed in time. As noted earlier task dispatch and deadline determines send and receive execution under DSR, while actual task execution affects the time of send and receive execution under ASR. In the former case the alignment of task dispatches and task deadlines and the transfer rate for PMT affect the deterministic nature of message communication. In the latter case task execution order and concurrency affects the determinism of actual send, transfer and receive order. Task ordering (τ_p ; τ_c or τ_c ; τ_p) implies that $\Omega_{P_j} \leq \alpha_{C_j}$ or $\Omega_{C_j} \leq \alpha_{P_j}$ and that $\alpha_P - \Omega_P \cap \alpha_C - \Omega_C = \emptyset$. Task ordering can be assured through precedence ordering and result in deterministic communication. Precedence ordering may be established by the dispatch order for tasks with the same priority, by the priority for tasks with the same dispatch time, by alignment of dispatch offset of the consumer with the deadline of the producer, or by control flow events resulting in the dispatch of the consumer task upon completion of the producer task.

Non-preemption of tasks ($\tau_c \neq \tau_p$) assures that

only one task completely executes at a time on a single processor, i.e., $\alpha_P - \Omega_P \cap \alpha_C - \Omega_C = \emptyset$, but does not guarantee task ordering.

Concurrent task execution ($\tau_c | \tau_p$) occurs if tasks execute simultaneously on shared processors, or preemptively on the same processor, i.e., $\alpha_P - \Omega_P \cap \alpha_C - \Omega_C \neq \emptyset$.

Finally, task preemption through static priority assures when task completion of a lower priority task occurs, no higher priority task is active, i.e., $\neg(\alpha_H \leq \Omega_L \leq \Omega_H)$. We denote preemption by producer with ($\tau_p > \tau_c$) and by consumer with ($\tau_p < \tau_c$).

The order of send, transfer, and receive operations also affects whether a message instance is communicated immediately, or is delayed by a message instance. The former is known as mid-frame communication, while the latter is known as period (or phase) delay. Mid-frame communication occurs for PMT if $S_{M_i} = S_j < X_{M_i} < R_{M_i} < T_{P_{j+1}}$ and for IMT if $S_{M_i} = S_j < R_{M_i} = R_k < T_{P_{j+1}}$. Phase-delayed communication occurs if the message instance is not received until after the next producer dispatch and the send occurs after the receive, i.e., $S_{M_i} = S_j < T_{P_{j+1}} < R_{M_i} = R_k < S_{j+1}$.

5. Port Buffers for Periodic Tasks

In this section we apply the lifespan concept to periodic task communication to determine the number of buffers necessary to ensure deterministic port communication. We take into consideration the choices of ASR and DSR, DMT, IMT and PMT, and task execution patterns. We identify whether communication is mid-frame (MF), phase-delayed (PD), non-deterministic (ND), or lacks data integrity (NDI).

Periodic task with the same period have the same task dispatch time, i.e., $T_{P_j} = T_{C_j} = T_j$ for all task dispatches j . Tasks are assumed to have pre-period deadlines, i.e., $D_j \leq T_{j+1}$. For embedded control systems this assumption typically holds and RTOS typically do not support queuing of dispatch request or multiple simultaneous dispatch execution of the same task.

For PMT the transfer period is assumed to be the same, i.e., $X_j = T_j$. In case of DSR we assume the transfer is immediately followed by the receive, i.e., X_j ; $R_j = T_j$, and for $D_{P_{j-1}} = T_{P_j}$ immediately preceded by the send, i.e., S_{j-1} ; X_j ; $R_j = T_j$.

For ASR the lifespan of MP is characterized as $B_{MP_i} = \alpha_P$ and $E_{MP_i} = \Omega_P$, same for MC. For ASR and IMT we have $\alpha_P - \Omega_P \cap \alpha_C - \Omega_C = \emptyset$ for the task

ordered and non-preemptive case, i.e., a single buffer solution suffices.

In case of $\tau_p ; \tau_c$ we get $T_j \leq \alpha_{p,j} < S_j < \Omega_{p,j} \leq \alpha_{c,j} < R_j < \Omega_{c,j} \leq T_{j+1}$. For a message instance M_i produced during dispatch j we get MF communication, i.e., $S_j = S_{M_i} \Rightarrow R_j = R_{M_i}$.

For $\tau_c ; \tau_p$ we get $T_j \leq \alpha_{c,j} < R_j < \Omega_{c,j} \leq \alpha_{p,j} < S_j < \Omega_{p,j} \leq T_{j+1} \leq \alpha_{c,j+1} < R_{j+1}$. In this case we get PD communication, i.e., $S_j = S_{M_i} \Rightarrow R_j = R_{M_i-1}$.

For $\tau_p \neq \tau_c$ we have non-deterministic execution order, i.e., non-deterministic send and receive order.

For $\tau_p | \tau_c$ we have $\alpha_p - \Omega_p \cap \alpha_c - \Omega_c \neq \emptyset$ order

First, we consider a task order of producer followed by consumer, i.e., $\tau_p ; \tau_c$ or $\Omega_{p,j} \leq \alpha_{c,j}$.

Applied to IMT, i.e., $S_j = X_j$, we get $T_j < S/X_j < R_j < T_{j+1}$.

Applied to PMT, we get $T_j = X_j < S_j < R_j < T_{j+1} = X_{j+1}$. For a message instance M_i produced during dispatch j

For DSR communication the receive occurs before the send, i.e., $S_{j-1} = D_{p,j-1} \leq T_j = R_j < S_j = D_{p,j}$. For message instance M_i produced during dispatch j we get PD communication, i.e., $S_j = S_{M_i} \Rightarrow R_j = R_{m_i-1}$. This applies to both IMT and PMT as the transfer is either combined with send, or immediately succeeded by receive.

For DMT communication ($S_j = X_j = R_j$), we have MF communication, i.e., $S_j = S_{M_i} \Rightarrow X_j = X_{M_i} \wedge R_j = R_{M_i}$. The lifespan of message instance M_i produced during dispatch j can be determined as follows. MF communication of ASR/IMT and DMT is characterized by $S_j = S_{M_i} \wedge R_j = R_{M_i}$, i.e., $E_{MC_i} = \Omega_{c,j} \leq T_{p,j+1} \leq B_{MP_{i+1}} = \alpha_{p,j+1}$. This results in a single message buffer solution. PD communication of ASR/PMT, DSR/PMT, and DSR/IMT is characterized by $S_j = S_{M_i} \wedge R_j = R_{M_i-1}$, i.e., $\alpha_{p,j} = B_{MP_i} \leq \Omega_{c,j} = E_{MC_{i-1}} \leq T_{j+1} \leq \alpha_{p,j+1} = B_{MP_{i+1}}$. This results in a two message buffer solution.

For DSR/IMT the data copy between the two message buffers must occur through the receive operation, i.e., $\Omega_{c,j} = E_{MC_{i-1}} \leq D_{p,j} \leq T_{j+1} = R_{j+1} \leq \alpha_{p,j+1} = B_{MP_{i+1}}$.

In case of DSR/PMT and ASR/PMT the data copy must be performed by the transfer at task dispatch time.

The same analysis can be performed for a task order of $\tau_c ; \tau_p$. Different from the previous task order, ASR/IMT and DMT implies $R_j < S_j$, i.e., period delay. The task order implies $\Omega_{c,j} = E_{MC_{i-1}} \leq \alpha_{p,j} = B_{MP_i}$, i.e.,

a single buffer solution.

A similar analysis can be done for $\tau_c \neq \tau_p$ and $\tau_c | \tau_p$. The results of these analyses are summarized in Figure 7.

Periodic Same period	ASR		DSR		DMT
	IMT	PMT	IMT	PMT	
$\tau_p ; \tau_c$	MF:1B	PD:2B S-X-R	PD:2B R	PD:2B S-X/R	MF:1B
$\tau_c ; \tau_p$	PD:1B	PD:1B	PD:1B	PD:1B	PD:1B
$\tau_p \neq \tau_c$	ND:1B	PD:2B X	PD:2B R	PD:2B X/R	ND:1B
$\tau_p \tau_c$	ND:3B S/X _c R _c	PD:2B X	PD:2B R	PD:2B X/R	NDI:2B S/X/R _c

MF: Mid-Frame
 PD: Period Delay
 ND: Non-Deterministic
 NDI: No Data Integrity

1B: Single buffer
 2B: Two buffers
 3B: Three buffers
 4B: Four buffers

S, X, R : data copy
 S/X : IMT combined send/xfer
 S/X/R : DMT combined S, X, R
 X/R : DSR/PMT combined X, R
 o1vo2 : One operation copy

Figure 7 Summary of same period task communication

Several observations can be made from the summary table.

- Mid-frame communication is only possible if task order can be assured.
- In case of consumer/producer order the lifespan of a message instance spans two periods, thus PD, but the lifespan of successive instances do not overlap, thus a single message buffer.
- Application send and receive operation execution (ASR) can result in non-deterministic message communication due to non-deterministic send and receive execution order. This is due to non-deterministic task order, i.e., $\tau_c \neq \tau_p \vee \tau_c | \tau_p$. PMT is necessary for deterministic message communication, i.e., message transfer must perform data copying. Note that typically periodic message transfer is not supported on a single processor.
- For ASR using IMT non-preemption of producer and consumer assures data integrity by non-concurrent task execution including non-concurrent send and receive operation execution. However, it does not alleviate non-deterministic message communication.
- For ASR using IMT concurrent producer and consumer execution requires the 3 buffer solution to achieve data integrity, but has non-deterministic message communication.
- Dispatch aligned send and receive (DSR) always achieves deterministic message communication through alignment of send and receive with task dispatching – even for non-deterministic task order.
- For DSR IMT and PMT communication produce the same result as the message transfer and receive are successively executed at task dispatch in both cases.

Deterministic message transfer for non-deterministic task order is achieved through data copying by the receive operation, i.e., DSR is not dependent on local message transfer support.

- For DSR communication always introduces period delay. If mid-frame communication is desired DSR has to be complemented with DMT, i.e., direct message communication at producer completion.

6. The AADL Runtime Executive

In AADL data ports are represented by port variables that correspond to MC, and out data ports correspond to MP. Port variables of incoming ports are assumed to be non-destructive, i.e., the application only reads from the port.

AADL places the restriction on data ports that a data port can only have one incoming connection. In other words, a consumer of data stream through a data port can only have one producer, although the producer can execute at a higher or lower rate than the consumer. The output of a producer can be made available to multiple consumers, and each consumer may be executing at a different rate.

AADL supports sampling of data ports in that the consumer samples at rates independent of the producer rates. By default consumers sample their input at dispatch time, i.e., their MC is updated at dispatch time. This is accomplished through sampling connections.

In addition, AADL support the specification of immediate (mid-frame) and delayed (phase-delayed) connections between periodic tasks. The timing semantics of these connection are defined such that they ensure deterministic sampled processing to maintain stability of control systems [7].

In case of immediate connections when the dispatch time for the producer and consumer are the same, the execution of the producer and consumer are aligned such that the producer completes before its output is made available to the consumer and the consumer starts execution. As a result, for threads with the same period the lifespan of MP and MC do not overlap. This will result in a single variable buffer between the two tasks.

In the case of delayed connections the output of the producer is not made available until the deadline of the producer. Since the default input sampling time is the dispatch of the consumer, the sampling of the consumer input occurs deterministically, i.e., in a deterministic write/read order. However, at least two buffers are required as the lifespan of MP and MC overlap.

In data ports AADL allows users to specify data ports as in/out data ports. This indicates that the same

port variable is used for both input and output. When combined with immediate connections we get the effect of a single shared variable across multiple tasks.

As we can see from the analysis framework in Section 2 assurance of deterministic communication and sampling requires that the communication be performed as part of the runtime executive rather than placing the communication calls into the application code. The runtime executive generator for MetaH [8] has done exactly that and produces highly efficient task dispatch and communication code. For example, data port communication between tasks on the same processor consists of a single memory move – expressed as an assignment statement of one port variable to another. MetaH was used as the starting point for SAE AADL and included data ports and immediate/delayed connections.

Since the desired timing semantics of task execution and task communication is explicitly specified in an AADL model, a runtime executive generator can produce efficient code by minimizing the number of buffers involved in the communication. At the same time the generator can ensure that when necessary the appropriate number of buffer variables are used in order to ensure data integrity and determinism of data communication. By doing so it will maintain the latency and age assumptions made by control algorithms as these algorithms are implemented in software and deployed as a set of tasks on a shared set of compute hardware resources.

7. Conclusions

In this paper we have defined an analysis framework that allows us to determine the impact of different implementation choices for communication on the latency characteristics of a data stream being communicated. It establishes criteria for data integrity and for deterministic data communication. Non-determinism in data communication, i.e., non-determinism in the send and receive order between producers and consumers, leads to frame-level jitter, i.e., runtime variation of latency in multiples of periods at which data is being sampled and processed.

The analysis framework has identified the degree of double buffering that is necessary in order to ensure determinism in this communication. This information can be used by a generator for AADL runtime systems that on one hand minimizes the amount of buffering necessary to possibly a shared variable solution, but at the same time guarantees the desired timing semantics through double buffering for the relevant port connections between tasks.

The framework can also be used to validate existing implementations of task communication that

may involve application code performing the data transfer – either by invoking a message system, or by utilizing common data areas. When implementing task communication on top of a real-time operating system several communication mechanisms may be available to the application developer. The framework presented here can provide guidance in choosing the appropriate communication mechanism for each of the task interactions and for understanding the consequence of the choice.

8. References

- [1] M. Klein, et.al., "A Practitioner's Handbook for Real-time Analysis: Guide to Rate Monotonic Analysis for Real-Time Systems". Boston: Kluwer, 1993.
- [2] D.B. Stewart, R.A. Volpe, and P.K. Khosla, "Design of dynamically reconfigurable real-time software using port-based objects," IEEE Trans. on Software Engineering, v.23, n.12, Dec. 1997.
- [3] Steenstrup, M., Arbib, M.A., and Manes, E.G., "Port Automata and the Algebra of Concurrent Processes" Journal of Computer and System Sciences, Vol. 27, No. 1, Aug. 1983, pp.29-50.
- [4] D. M. Lyons and M. A. Arbib, "A formal model of computation for sensory-based robotics," IEEE Transactions on Robotics and Automation, v.5, n.3, pp. 280-293, June 1989.
- [5] SAE-AS5506. "SAE Architecture Analysis and Design Language (AADL)." International Society of Automotive Engineers. Warrendale, USA, November 2004.
- [6] Feiler, P. H., Gluch, D., Hudak, J., "The Architecture Analysis & Design Language (AADL): An Introduction", Software Engineering Institute, Technical Note, CMU/SEI-2006-TN-011, Feb 2006.
- [7] Feiler, P. H., Hansson, J., "Impact of Runtime Architectures on Control System Stability", 4th International Embedded Real-Time Systems Conference (ERTS), Toulouse, France, Jan 2008.
- [8] Vestal, S. & Binns, P. "Scheduling and Communication in MetaH," 194-200. Proceedings on Real-Time Systems Symposium. Raleigh-Durham, NC, Dec. 1993. New York. NY: IEEE, 1993.