A Simulation Framework for Design of Mixed Time/Event-Triggered Distributed Control Systems with SystemC/TLM

Zhenkai Zhang, Joseph Porter, Xenofon Koutsoukos, and Janos Sztipanovits
Institute for Software Integrated Systems (ISIS)
Department of Electrical Engineering and Computer Science
Vanderbilt University
Nashville, Tennessee, USA
Email: {zhenkai.zhang, joseph.porter, xenofon.koutsoukos, janos.sztipanovits}@vanderbilt.edu

***Abstract*—Mixed time/event-triggered (TT/ET) distributed control systems are complex systems which have emerged in many cyber-physical domains but have been difficult to evaluate at early design stages. In order to reveal design flaws as early as possible, this paper proposes a simulation framework based on an executable virtual platform model in SystemC/TLM. The executable platform is generated using a model-based approach from a system designed in the Embedded Systems Modeling Language (ESMoL). The virtual platform consists of three types of abstract models, the RTOS model, the communication system model, and the hardware model, to capture different behaviors of the mixed TT/ET distributed control systems. Preliminary results from a case study using a Quadrotor flight control system are used to illustrate the approach.**

***Keywords*-Mixed Time/Event-Triggered Distributed Control Systems; Virtual Platform; SystemC/TLM; Graphical Models;**

I. Introduction

Nowadays, most complex cyber-physical systems (CPSs), such as automotive vehicles, air planes and trains, use distributed control systems, in which several ECUs are connected by network(s)/bus(es) [22]. Typically, these control systems are hard real-time systems. Traditionally, these control systems are composed of event-triggered (ET) tasks and use event-triggered communication systems, such as CAN bus. As time-triggered architectures (TTA) offer advantages such as determinism, predictability, and composability [9], many new systems tend to be built using TTA. However, many sporadic events make the designs not fit into the strict periodic framework [13]. These CPS control systems often consist of both TT and ET tasks and use a mixed communication protocol (e.g. FlexRay and TTEthernet) to form mixed TT/ET distributed control systems [17].

When designing mixed TT/ET distributed control systems, many challenges arise due to a large design space and lack of tools to explore it. First, the hardware platform needs to be designed including a set of nodes connected by a communication system. Trade-offs between cost and performance drive the selection of the appropriate processors. The communication system bandwidth and topology also need to be considered in order to meet performance and redundancy requirements. Then, partitioning tasks into TT or ET needs to be considered. Moreover, mapping the tasks on the hardware platform is also important and affects timing [17]. After mapping, the mixed communication system needs to be configured using proper parameters. Thus, the space of possible design configurations is large and early evaluation is necessary to eliminate bad design decisions which may cause the system to fail to meet its requirements at a later design stage. However, most existing work focuses on specifying the control system [23], analyzing the system's schedulability [18], optimizing partitioning, mapping and bus cycle [17], and inter-task communication mechanism [21], but not evaluating the whole system.

For mixed TT/ET distributed control systems, both computation and communication should be captured to enable evaluation of the whole system with respect to functionality, timing, and performance. Both computation and communication concerns are coupled to the application software, system software and hardware platform. The ability to model, integrate, and simulate all parts together is essential for design space exploration during early development. A virtual platform including both hardware and embedded software can be used as a pivot in this evaluation framework, since it is available much earlier than the real system [7].

System-Level Design Languages (SLDLs), such as SystemC and SpecC, can be used to model both hardware platforms and embedded software. In addition, most SLDLs support the concept of transaction-level modeling (TLM) which separates the design of the computation and communication. A TLM communication structure abstracts away communication details to speed up simulation while keeping required accuracy. SystemC has been a *de facto* SLDL [3]. It also has a TLM library for modeling memory-mapped buses. SystemC/TLM-based virtual platforms on system-level can model the hardware behavior with good simulation efficiency and sufficient timing accuracy at early stages [4].

In order to evaluate a mixed TT/ET system, we propose a simulation framework based on a virtual platform in SystemC/TLM and is combined with a model-based design

environment in GME [12]. The virtual platform consists of a mixed TT/ET computation model and communication model. For the computation model, an abstract RTOS model and abstract hardware models such as processor and peripherals are needed. The abstract RTOS model is built in SystemC and takes charge of task scheduling, inter-task communication, synchronization and interrupt handling [5] [11] [24]. The behaviors of both TT and ET tasks should be captured by the abstract RTOS model, and abstract hardware models form the underlying computational platform. For the communication model, an abstract communication system model which can capture the behaviors of both TT and ET communication is needed. Since FlexRay has been widely used in many CPS domains, it is a good reference for modeling the mixed TT/ET communication [2]. The abstract communication system model in this framework is based on FlexRay.

The simulation framework is also integrated with a model-based design environment in GME called Embedded Systems Modeling Language (ESMoL) [19]. The mixed TT/ET distributed control systems in ESMoL can be transformed to the virtual platform models using automated model transformations. This integration makes the design and simulation of the mixed TT/ET systems effective and efficient. As the framework is still in progress, preliminary results from a case study using a Quadrotor flight control system are used to illustrate the approach. The bandwidth of two communication systems is evaluated for the designed mixed TT/ET system.

There have been efforts to establish simulation frameworks for design of distributed real-time control systems. In [15], a framework based on generated virtual execution platform is proposed, in which VaST tool is used to model a cycle-accurate hardware platform and μC/OS-II RTOS is ported to the modeled platform. Although the simulation can be very accurate, the very low abstraction level makes it not suitable for early design evaluation. Compared to [15], a framework named E-TTM is proposed on a very high abstraction level for the design of TTA-based real-time control systems in [16]. Too high level of abstraction also impedes the use of the framework due to inaccuracy. In [14], a UML-based design framework is proposed. A system is described in UML, and then the UML model can be converted into SystemC model for simulation. However, this framework is only for TT applications without considering mixed TT/ET systems.

Compared to the related efforts, the main contributions of this proposed framework are: (1) it uses abstract computation and communication models to establish a universal simulation framework for mixed TT/ET systems, while the levels of abstraction are appropriate to keep the simulation efficient and accurate; (2) the framework is also integrated with a model-based design tool to improve its usability.

The rest of this paper is organized as follows: Section 2 introduces the virtual platform model in the proposed simulation framework which consists of three types of models; Section 3 describes how to transform an ESMoL design into an executable virtual platform; Section 4 gives a case study on the design of a Quadrotor flight control system and uses the simulation framework to evaluate the influence of the bandwidths of the communication systems; Section 5 concludes this paper and gives future work.

II. Virtual Platform Model

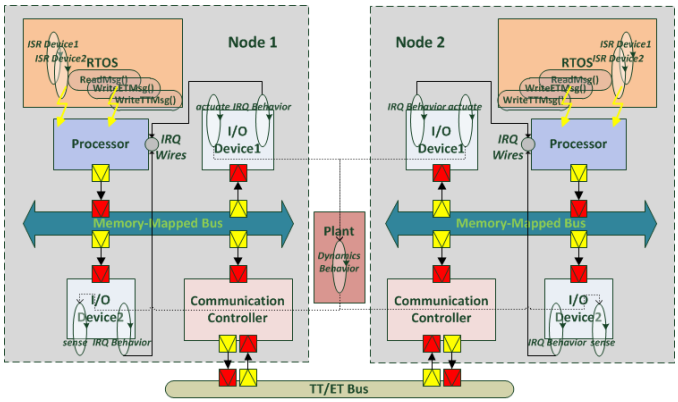

Fig. 1. Virtual platform with abstract RTOS model, abstract communication system model, and abstract hardware model.

The virtual platform, as shown in Fig. 1, consists of three types of models which are the abstract RTOS model, the abstract hardware model, and the abstract communication system model. The models are implemented in SystemC by inheriting from the *sc_module* class in which concurrent behaviors are modeled by a set of SystemC processes (*SC_THREAD* or *SC_METHOD*). These three models can be instantiated and integrated to be an executable model for simulating mixed TT/ET control systems.

A. RTOS Modeling

On a node of a distributed system, TT and ET tasks which realize the desired functionalities interact with an RTOS. The RTOS captures the dynamic behaviors of the tasks.

The abstract RTOS model has three *SC_THREAD* processes, which are the RTOS service process interacting with TT and ET tasks, the time-trigger process taking charge of triggering TT tasks according to a static schedule, and the interrupt handling process invoking an interrupt service routine (ISR) to handle the corresponding interrupt. Fig. 2 shows the fundamental services supported by the abstract RTOS model.

1) Task Management: In a mixed TT/ET system, tasks are divided into TT tasks and ET tasks. Time-triggered tasks are activated according to a predefined schedule. When a node's synchronized local clock reaches a predefined time instant, the corresponding TT task will be put into the ready queue. TT tasks can be non-preemptive or preemptive. ET tasks are activated dynamically depending on the occurrence of associated events. ET tasks can also be non-preemptive or preemptive.

Each task in the abstract RTOS model corresponds to a SystemC *SC_THREAD* process. In order to serialize the tasks and control their execution, each task pends on its own *sc_event* object. The RTOS scheduler controls the execution by notifying the task's *sc_event* object. A task's execution

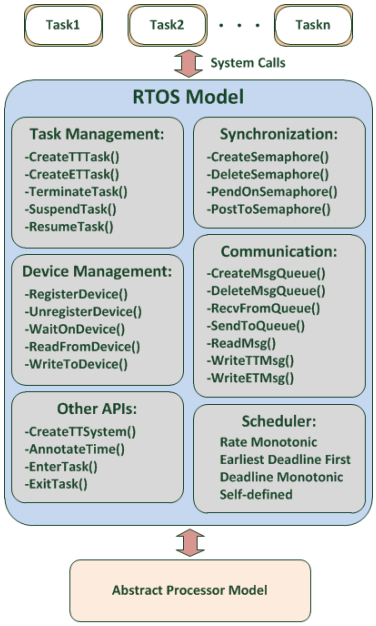

Fig. 2. Abstract RTOS model with supported primitives

information is stored in its Task Control Block (TCB). Since in SystemC the execution between two *wait()* statements is in zero simulation time, we need to advance time and model execution of tasks by using *wait()* statements. The execution time of a section of code is modeled by inserting timing annotations into the task. The annotation can be coarse-grained on the task level or fined-grained on the basic block or statement level.

There are a set of primitives provided by the RTOS model to manage a task's creation, termination, resumption and suspension. When created, each task needs a task name, a worst case execution time (WCET), and a deadline. In addition, each TT task also needs a predefined schedule passed as a parameter, and each event-triggered task needs a user-defined priority and/or a period depending on the scheduling policy.

2) Scheduling: The scheduler is the heart of the RTOS, which allocates CPU time to a selected task from the ready queue. The scheduler's behavior depends on a specific scheduling algorithm. In the abstract RTOS model, the scheduler has three common priority-based scheduling policies which are rate monotonic (RM), deadline monotonic (DM), or earliest deadline first (EDF). Other scheduling algorithms can also be easily added into the RTOS model. The scheduler's timing properties are RTOS- and hardware platform-specific. Basically, there are two main parameters of its timing properties: scheduling overhead and context switching overhead. There is some research work on how to accurately acquire these parameters [6] [8], which is not the focus of this paper, so we assume the parameters are already available for a particular system.

The task state transitions are modeled by two finite state machines as shown in Fig. 3, one for TT tasks and the other one for ET tasks. The *Created* state is for any new task. Depending on the task type (TT or ET), it transitions to the corresponding state. A TT task enters the *Idle* state, and an ET task enters the *Ready* state. A TT task enters the *Ready* state statically according to an *a priori* schedule table, while an ET task enters the *Ready* state dynamically when the event happens. There is only one ready queue, which contains both time-triggered and event-triggered "ready-to-run" tasks. The scheduler schedules this ready queue using the assigned scheduling policy. Only one task can be in the *Running* state at a time which is chosen by the scheduler.

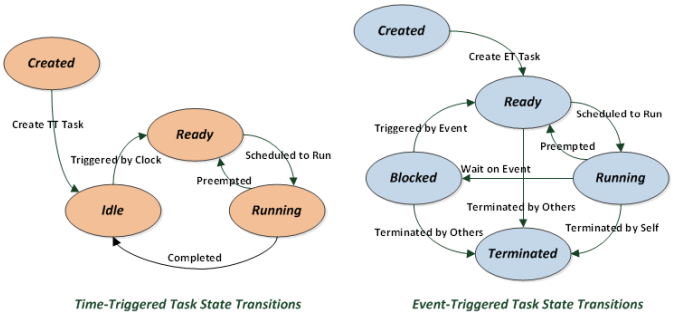

Fig. 3. Task state transitions of TT/ET tasks.

3) Inter-Task Communication: In a multi-tasking RTOS, tasks need to communicate with others synchronously or asynchronously using inter-task communication mechanisms. In the abstract RTOS model, inter-task communication on one node can be achieved by shared memory or message queue, and inter-task communication between different nodes can be achieved by message-passing. Shared memory is used between TT tasks, since it can be accessed without race-conditions. Semaphore synchronization is used by ET tasks to serialize access to shared memory and maintain task dependencies.

Communication between TT and ET tasks on a single node occurs through message queue. An agent is associated with a TT task in the message queue as a state message keeper. If the message queue is empty, when an ET task tries to poll a message from it, it will be blocked on it; whereas, for a TT task, the agent will give the task the message in the queue if it is available, and then update its state message as the latest dequeued message or give the task the state message if the queue is empty. So when a TT task accesses the message queue, it will never be blocked even if there is no message available in the queue.

Communication between two tasks on different nodes are through message passing. Two types of messages are supported in the model, one is TT and the other one is ET. TT messages are transmitted in the predefined static time slots of a communication cycle. ET messages are transmitted according to the combination of their priorities and the dynamic slots.

4) Interrupt Handling: SystemC has some disadvantages for RTOS modeling, which can be summarized as non-interruptible *wait-for-delay* time advance and non-preemptive simulation processes. When an interrupt happens, it requires the real-time system to react and handle it in a timely manner. Modeling an accurate interrupt handling mechanism plays an important role in RTOS modeling. We adopt the method from

[24] which makes task use *wait-for-event* other than *wait-for-delay* to advance its execution time. A system call of the RTOS model taking execution time as its argument makes the task wait on a *sc_event* object which will be notified after the given execution time elapses if no interrupt happens. When an interrupt happens and its corresponding ISR preempts the execution of the task, the notification of the *sc_event* object will be canceled and a new notification time will be calculated according to how much time the preemption took and how much execution time already passed.

B. Communication System Modeling

In a distributed control system, the timing behavior of the communication system has an important impact on system performance. There are a few communication protocols that provide predictable message delays [20]. The abstract communication system model in this framework is based on FlexRay protocol [2], which can handle both time-triggered and event-triggered communication. The data granularity of the model is at the message-level, since we only need to consider message delay rather than the detailed timing of underlying operations for evaluation of the system. The behavior of the communication system is modeled by a state chart as shown in Fig. 4. The abstract communication system model takes advantage of the global time in the SystemC simulation kernel, and uses it as its synchronized time base.

The communication controller model realizes the behavioral model of the communication system and takes charge of transmitting and receiving messages through the underlying medium. Its implementation class is also derived from the *sc_module* class of SystemC. It also utilizes the TLM-2.0 library in SystemC to realize the underlying transmission. The TLM-2.0 library is mainly for modeling memory-mapped buses, so we change some of its semantics to model our communication system. The controller acts as both an initiator and a target for TT/ET bus transactions, and the TT/ET bus is an interconnect component. The controller also acts as a target for memory-mapped bus transactions within a node. The write command of a transaction means to transmit the message included in the generic payload. For simplicity, only the blocking transport interface (*b_transport()* method) is used.

Bus communication is organized in cycles. Each cycle consists of three segments including a time-triggered static segment, an event-triggered dynamic segment and a waiting segment. The time-triggered part is based on a time-division multiple-access (TDMA) medium access protocol (MAC), and the event-triggered part is based on flexible TDMA as in FlexRay and Byteflight [2] [1]. For time-triggered communication, a predefined schedule is also needed and passed through a configuration file.

In the static segment superstate, if the current time slot is scheduled to receive a message from the bus, the communication controller goes into *RECV* state; if scheduled to send a message, it transitions to the *SEND* state to start a message transmission. Otherwise, it will stay in the *IDLE* state. When a static time slot is elapsed, the model checks whether the time-triggered communication part is finished by comparing if the slot counter has reached the allocated number of static slots. The schedule guarantees there are no transmission contentions, so dynamic arbitration is not necessary.

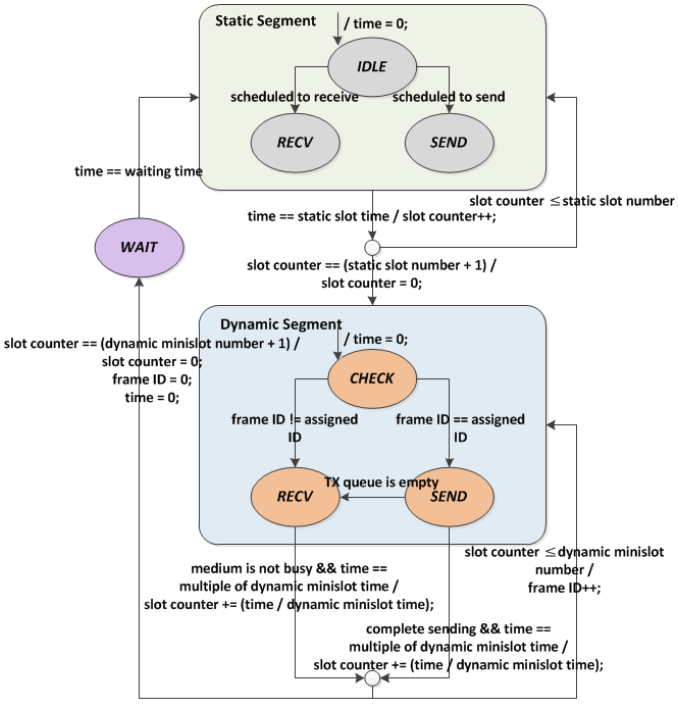

Fig. 4. Behavior of the abstract communication system model based on FlexRay protocol.

The dynamic segment superstate takes charge of event-triggered communication. The segment is divided into a set of minislots. A frame ID variable is updated synchronously by every node in the system. In order to solve the contentions between sending nodes, the rights of transmission are ordered by the frame ID assigned to each node. Different from FlexRay, in this model the dynamic messages are put in a single queue. The queue is sorted by the priorities assigned to the messages associated with the task's priorities, and the messages with the same priority are ordered by FIFO. Each dynamic time slot can have varying number of minislots. The length of a dynamic time slot depends on the size of the transmitted message. When the controller transmits the data in its queue, the controller needs to check whether this is allowed. First, it checks if it has the right to use the current frame ID by comparing with its assigned frame IDs. Then, it searches in the queue for the message with the highest priority which can fit into the remaining dynamic segment time. If there is such a message, the controller sends it on the bus by calling *b_transport()* method; otherwise, the controller defers sending and will try in the next cycle. When the dynamic communication phase is finished, the controller enters the *WAIT* state. In FlexRay, this time is mainly for clock synchronization. Since we use the global time in the SystemC simulation kernel, we do not need to do clock synchronization and we use this state to model a realistic timing behavior.

C. Hardware Modeling

The abstract RTOS model is running on an abstract processor model which communicates with other peripherals through a memory-mapped bus modeled in TLM-2.0, as shown in Fig. 1. Peripherals are divided into communication controllers and other I/O devices. I/O devices are used to model sensors and actuators that interact with the plant dynamics. Each I/O device has a corresponding ISR *SC_THREAD* process in the RTOS registered when calling *RegisterDevice()*. In the current state of the framework, the processor is modeled in a simplified way. The abstract RTOS model interacts with the processor model by: (1) the tasks invoke *ReadMsg()*, *WriteTTMsg()*, and *WriteETMsg()* primitives to make the processor initiate bus transactions with the communication controller; (2) the processor signals the RTOS model that a registered I/O device's ISR needs to be activated; (3) the ISR signals the processor to start bus transactions with the corresponding I/O devices. The processor model has a *sc_port* object which is a multi-port connected by each I/O device's interrupt request (IRQ) wire.

Each I/O device is derived from the *sc_module* class and has a *SC_THREAD* process to control its IRQ behavior. The behavior of the IRQ can be modeled in two modes, asynchronously periodic and sporadic. For example, an UART's IRQ can be modeled as sporadic if the interval between two interrupts has a minimum period, or it can be modeled as asynchronously periodic if the interrupts have a fixed period but are not synchronized with the clock of the processor. When an IRQ occurs, the I/O device will trigger the IRQ wire connected to the processor and corresponding ISR will become ready to handle the IRQ. The ISR will be put into the ready queue first usually with the highest priority, then it would preempt other tasks and run immediately. The order of interrupt handling is based on the IRQ priorities if there are more than one IRQs at the same time. When an ISR finishes, it will check if there is any ET task pending on it and put the corresponding blocked task into the ready queue. Each I/O device also has a *SC_THREAD* process to interact with the plant model. If the device is a sensor, a *sense* process will pull sensor data from the plant and wait for a read transaction. If the device is an actuator, an *actuate* process will wait for a write transaction and send the data to the plant.

III. Model-Based Approach

The front end of this simulation framework is a single multi-aspect embedded software design environment called Embedded Systems Modeling Language (ESMoL) [19]. The executable simulation model is generated from ESMoL. The model transformation process is shown in Fig. 5. Two interpreters are used to realize the model transformations.

An ESMoL model consists of different models used to capture different aspects of the designed system. The design entry of an ESMoL model is to specify the control system's functionality in the Simulink environment. The Simulink model will be imported into the ESMoL automatically to become the functional specification for instances of software components. A logical software architecture model is established to capture data dependencies between software component instances independent of their distribution over different processors. A hardware platform model is defined hierarchically as hardware units with ports for interconnections. By mapping software components to processing nodes and data messages to communication ports, a deployment model is created. By attaching timing parameter blocks to components and messages, a timing model is established. The whole design process is described in detail in [19].

The interpreter in stage 1 transforms the ESMoL model to an equivalent model in an intermediate language called ESMoL_Abstract. The model in this intermediate language is flattened and the relationships implied by structures in ESMoL are represented by explicit relation objects in ESMoL_Abstract. This translation is similar to the way a compiler translates concrete syntax first to an abstract tree, and then to intermediate semantic representations suitable for optimization. The interpreter in stage 2 uses the UDM model navigation API to generate the simulation model according to the corresponding templates. The generation of the simulation model consists of three parts.

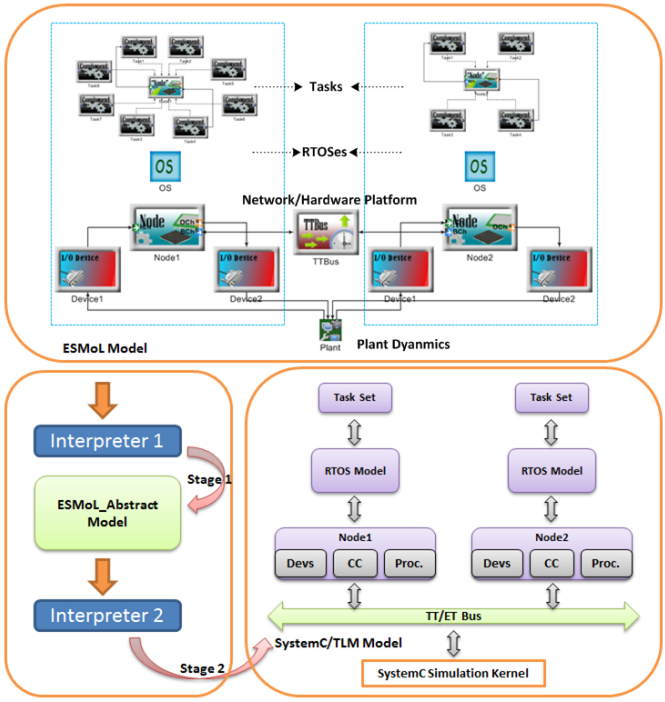

Fig. 5. ESMoL model and its corresponding SystemC model via model transformation using two interpreters.

The first part is to instantiate the hardware and software models according to the templates. Each processor, I/O device, communication controller, bus, and RTOS in the ESMoL_Abstract model is instantiated in the *sc_main()* function. All the instances belonging to the same node are assembled by binding the sockets or ports. The communication controller in each node is bound to the TT/ET bus instance. Each I/O device is registered into the RTOS by calling *RegisterDevice()*

method, which will register an ISR *SC_THREAD* process with its timing and type information (sensor/actuator) passed through the ESMoL model. ISR process pends on its own *sc_event* object, and has the address information of the device. Each task has a corresponding *SC_THREAD* process. The process pends on its own *sc_event* object which will be notified if the process is chosen to run by the scheduler. A task is time-triggered if its *ExecInfo* object in the ESMoL model is a *TTExecInfo* object, whereas it is event-triggered if its *ExecInfo* object is a *AsyncPeriodicExecInfo* or a *SporadicExecInfo* object. A TT task only waits on its own *sc_event* object. An ET task waits on the corresponding event by calling either *WaitOnDevice*, *WaitOnSemaphore* or *WaitOnMsgQueue* primitive. When the event occurs, the ET task goes to the ready queue. All the tasks are registered into the RTOS instances by calling either *CreateTTTask()* (time-triggered) or *CreateETTask()* (event-triggered) primitive. If the task is a sender of a message, it invokes *WriteTTMsg()* system call to send a TT message, or *WriteETMsg()* if the message is an ET one. Shared variables are used for inter-task communication of the same task type (TT/ET). Communication channel between a TT task and an ET task on the same node in the ESMoL model is translated to a message queue. The plant model also has a *SC_THREAD* process for time stepping its dynamics function and is instantiated in the *sc_main()* function. This process also exchanges data with sensors/actuators via shared memory.

The second part generates the configuration files for the model instances according to the specified attributes in the ESMoL model, such as the static schedule tables for RTOSes and the segment configurations for communication controllers. The third part is to generate the functional C code for the tasks and the plant dynamics using Real-Time Workshop (RTW), and integrate the functional code with the generated model in the first part. The generated codes can be easily wrapped into the corresponding *SC_THREAD* processes of the tasks and the plant model.

IV. Case Study

In this section, we employ the simulation framework for the design of a Quadrotor flight control system and present some preliminary results to illustrate the approach.

The controller for the Quadrotor is designed using two linear proportional derivative (PD) controllers, an inner loop and an outer loop, as shown in Fig. 6. The outer loop controller is

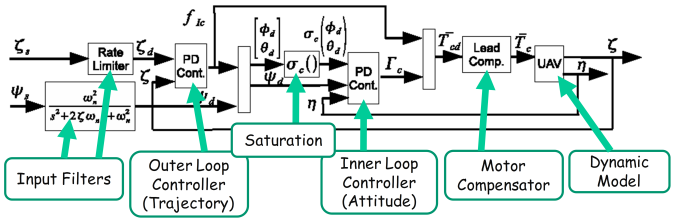

Fig. 6. Two PD controllers in the Quadrotor control system.

a "slow" PD inertial controller and the inner loop is a "fast" PD attitude controller. More details are described in [10]. The corresponding Simulink model (shown in Fig. 7) is built which has four blocks (*ReferenceHandler*, *DataHandler*, *InnerLoop* and *OuterLoop*). After validation of the Simulink model, the model is automatically imported into ESMoL.

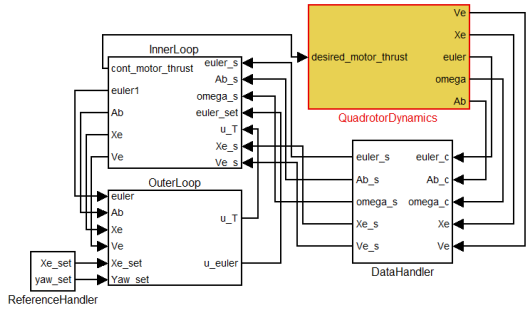

Fig. 7. Simulink model of the Quadrotor control system.

Four ESMoL models are established to capture different aspects of the design. As shown in Fig. 8, the logical software architecture in the ESMoL model gives the data dependencies of the tasks in the Quadrotor control system. Fig. 9 shows the platform of the Quadrotor which has two nodes, one is based on PXA255 processor and the other one is based on ATmega128, and they are connected by a TT/ET bus. Each node has its I/O devices to interact with the outside plant. For task deployment as shown in Fig. 10, two tasks (*ReferenceHandler* and *OuterLoop*) are assigned to the PXA255 node, and the other two tasks (*DataHandler* and *InnerLoop*) are assigned to the ATmega128 node. A timing model (Fig. 11) is also established by attaching timing parameter blocks to components and messages. As shown in Fig. 11, the *ReferenceHandler* task and *DataHandler* task are assigned as event-triggered tasks, and the other two tasks are time-triggered tasks.

In this design, we constrain the design space by using the above hardware platform, task type (TT/ET) assignment, and task deployment, and only focus on the influence of the bandwidth of the TT/ET bus.

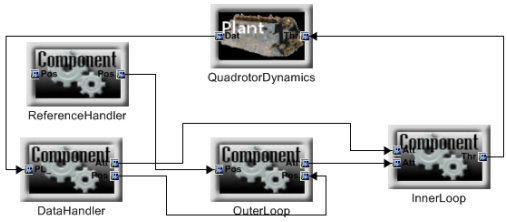

Fig. 8. Software data dependencies of the Quadrotor control system.

The sampling period of the system is 20ms. For tasks on each node, their WCETs are measured empirically. On the PXA255 processor, *ReferenceHandler* needs 50μs with relative deadline 5ms, and *OuterLoop* needs 1.6ms with relative deadline 2ms. On the ATmega128 processor, *DataHandler* needs 200μs with relative deadline 4ms, and *InnerLoop* needs 600μs with relative deadline 1ms. The ISR of the Ethernet on

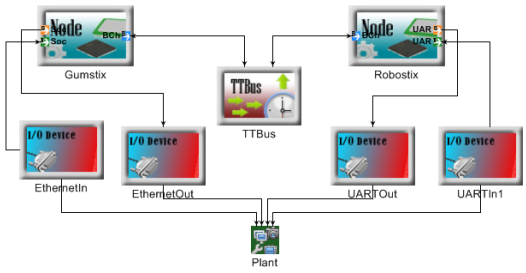

Fig. 9. Hardware platform of the Quadrotor control system.

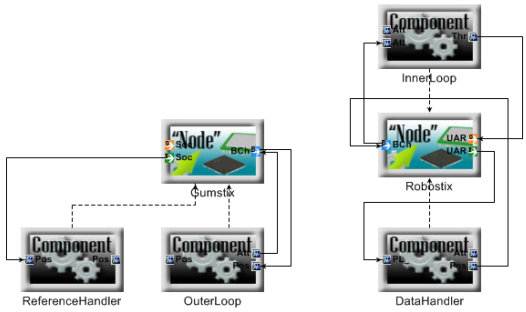

Fig. 10. Task deployment of the Quadrotor control system.

PXA255 needs 5μs, and the ISR of the UART on ATmega128 needs 2μs. In each cycle, *OuterLoop* is time-triggered at 12ms and sends a message to *InnerLoop* with the attitude control data, *InnerLoop* is time-triggered at 10ms, and *DataHandler* sends a messge to *OuterLoop* with the position data.

Since there are a few tasks contending for computation resources, the scheduling algorithm will not make a big difference for control performance. On PXA255 processor RM is used and on ATmega128 processor EDF is used. However, the Quadrotor control system is sensitive to communication delays, which will make us choose the appropriate communication system. Suppose there are only two options for the communication system, one has the maximum bandwidth of 400Kbit/s but cheaper and the other has the maximum bandwidth of 2Mbit/s but more expensive.

First, the 400Kbit/s bandwidth TT/ET bus is tried and the static time slot is set as 2ms, since the largest message (60 bytes) needs 1.2ms for transmission. PXA255 uses the seventh static slot to transmit the attitude control data and ATmega128 uses the sixth static slot to transmit the position data. The simulated control performance is shown in Fig. 12. The left figure shows the simulated trajectory of the Quadrotor compared to its reference, in which the solid lines give the trajectory and the dotted lines give the reference: red lines are positions along X-axis, black lines are positions along Y-axis, and blue lines are height positions, respectively. The right figure shows the error between the simulated trajectory and reference. The timing behaviors of the tasks is shown in Fig. 13. In the timing diagram, each red dotted line represents the deadline of the task. From the timing diagram we can see every task meets its deadline. However, the control performance begins more and more unstable as time passes.

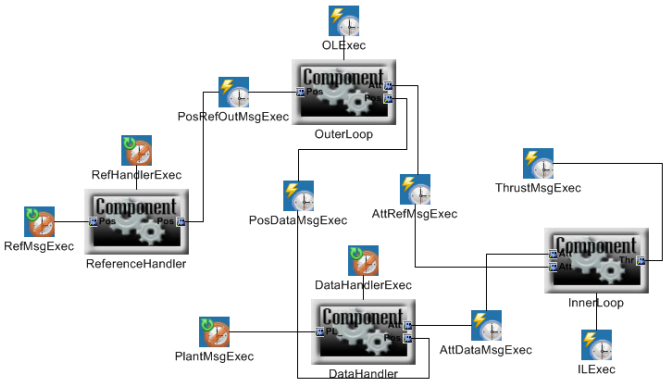

Fig. 11. Timing model of the Quadrotor control system.

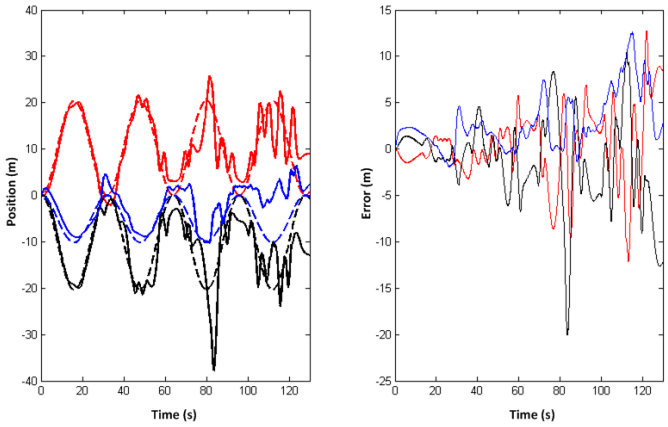

Fig. 12. Control performance with bus bandwidth of 400Kbit/s.

The second case uses the 2Mbit/s bandwidth TT/ET bus and the static slot is set as 1ms, as the largest message (60 bytes) needs 240μs for transmission. PXA255 uses the fourteenth static slot to transmit the attitude control data and ATmega128 uses the eleventh static slot to transmit the position data. Fig. 15 depicts the timing behaviors of the tasks which are similar to the timing diagram in Fig. 13 due to the modified communication configuration which does not affect the computation part. The message from *DataHandler* to *OuterLoop* is transmitted during the eleventh static slot, which can be used as the latest state message by *OuterLoop*. The improvement of the bandwidth increases the cost of the system but stabilizes the control performance which is shown in Fig. 14.

V. Conclusion

In this paper, a simulation framework for design of mixed TT/ET distributed control system is introduced. The framework consists of a virtual platform model in SystemC/TLM and a model transformation approach to generate the virtual platform for a designed system. The virtual platform model has three different models, which are abstract RTOS model, abstract communication system model, and abstract hardware model. The RTOS model is used to capture the dynamic behaviors of TT/ET tasks, and the communication system

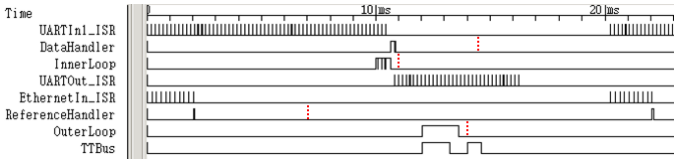

Fig. 13. Timing diagram of the control system with bus bandwidth of 400Kbit/s.

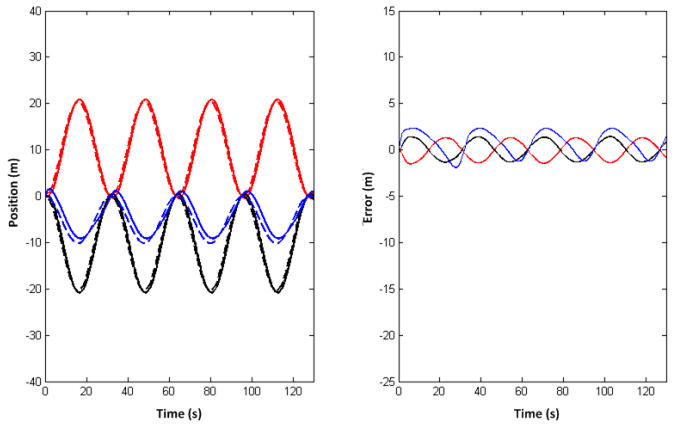

Fig. 14. Control performance with bus bandwidth of 2Mbit/s.

model is used to capture the behaviors of TT/ET communication. The hardware model integrates the RTOS model and the communication system model together. Two model transformations translate the ESMoL model to the corresponding virtual platform model for simulation. We present a case study on a Quadrotor flight control using this framework, and give the simulation results for different bandwidths of the TT/ET buses.

The future work includes introducing a more realistic communication system model with startup, restart, and clock synchronization services and more realistic hardware models that give more realistic timing behaviors.

REFERENCES

[1] Byteflight Homepage. http://www.byteflight.com.

[2] FlexRay Homepage. http://www.flexray.com.

[3] IEEE Standard SystemC Language Reference Manual, 2005.

[4] D. D. Gajski, S. Abdi, A. Gerstlauer, and G. Schirner. *Embedded System Design: Modeling, Synthesis and Verification*. Springer Publishing Company, Incorporated, 1st edition, 2009.

[5] A. Gerstlauer, H. Yu, and D. D. Gajski. RTOS Modeling for System Level Design. In *Proceedings of the conference on Design, Automation and Test in Europe - Volume 1*, DATE '03, 2003.

[6] Z. He, A. Mok, and C. Peng. Timed RTOS Modeling for Embedded System Design. In *Proceedings of the 11th IEEE Real Time on Embedded Technology and Applications Symposium*, RTAS '05, pages 448–457, 2005.

[7] S. Hong, S. Yoo, S. Lee, S. Lee, H. J. Nam, B.-S. Yoo, J. Hwang, D. Song, J. Kim, J. Kim, H. Jin, K.-M. Choi, J.-T. Kong, and S. Eo. Creation and Utilization of A Virtual Platform for Embedded Software Optimization: An Industrial Case Study. In *Proceedings of the 4th international conference on Hardware/software codesign and system synthesis*, CODES+ISSS '06, pages 235–240, 2006.

[8] Y. Hwang, G. Schirner, S. Abdi, and D. G. Gajski. Accurate Timed RTOS Model for Transaction Level Modeling. In *Proceedings of the Conference on Design, Automation and Test in Europe*, DATE '10, pages 1333–1336, 2010.

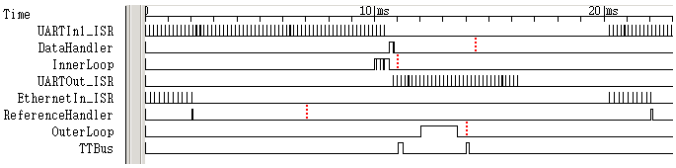

Fig. 15. Timing diagram of the system with bus bandwidth of 2Mbit/s.

[9] H. Kopetz and G. Bauer. The Time-Triggered Architecture. *Proceedings of the IEEE*, 91(1):112–126, 2003.

[10] N. Kottenstette and J. Porter. Digital Passive Attitude and Altitude Control Schemes for Quadrotor Aircraft. In *Proceedings of the 7th IEEE Intl. Conf. on Control and Automation*, ICCA '09, ChristChurch, New Zealand, 2009.

[11] R. Le Moigne, O. Pasquier, and J.-P. Calvez. A Generic RTOS Model for Real-time Systems Simulation with SystemC. In *Proceedings of the conference on Design, automation and test in Europe - Volume 3*, DATE '04, pages 30082–, 2004.

[12] A. Ledeczi, M. Maroti, A. Bakay, G. Karsai, J. Garrett, C. Thomason, G. Nordstrom, J. Sprinkle, and P. Volgyesi. The generic modeling environment. In *Workshop on Intelligent Signal Processing*, 2001.

[13] A. Metzner. Analyzing Mixed Event Triggered/Time Triggered Systems.

[14] K. D. Nguyen, P. S. Thiagarajan, and W.-F. Wong. A UML-Based Design Framework for Time-Triggered Applications. In *Proceedings of the 28th IEEE International Real-Time Systems Symposium*, RTSS '07, pages 39–48, 2007.

[15] S. Park, W. Olds, K. G. Shin, and S. Wang. Integrating Virtual Execution Platform for Accurate Analysis in Distributed Real-Time Control System Development. In *Proceedings of the 28th IEEE International Real-Time Systems Symposium*, RTSS '07, pages 61–72, 2007.

[16] J. Perez, A. Perez, and R. Obermaisser. Executable Time-Triggered Model (E-TTM) for Real-Time Control Systems. In *Proceedings of the 2010 13th IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing*, ISORC '10, pages 42–49, 2010.

[17] T. Pop, P. Eles, and Z. Peng. Design Optimization of Mixed Time/Event-Triggered Distributed Embedded Systems. In *Proceedings of the 1st IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, CODES+ISSS '03, pages 83–89, 2003.

[18] T. Pop, P. Eles, and Z. Peng. Schedulability Analysis for Distributed Heterogeneous Time/Event Triggered Real-Time Systems. In *Proceedings of the 15th EuroMicro Conf. on Real-Time Systems*, ECRTS '03, pages 257–266, 2003.

[19] J. Porter and G. Hemingway et al. The ESMoL Language and Tools for High-Confidence Distributed Control Systems Design. Part 1: Language, Framework, and Analysis. Technical Report ISIS-10-109, ISIS, Vanderbilt Univ., 2010.

[20] J. M. Rushby. Bus Architectures for Safety-Critical Embedded Systems. In *Proceedings of the First International Workshop on Embedded Software*, EMSOFT '01, pages 306–323, 2001.

[21] N. Scaife and P. Caspi. Integrating Model-Based Design and Preemptive Scheduling in Mixed Time- and Event-Triggered Systems. In *Proceedings of the 16th Euromicro Conference on Real-Time Systems*, ECRTS '04, pages 119–126, 2004.

[22] J. Sztipanovits, X. D. Koutsoukos, G. Karsai, N. Kottenstette, P. J. Antsaklis, V. Gupta, B. Goodwine, J. S. Baras, and S. Wang. Toward a science of cyber-physical system integration. *Proceedings of the IEEE*, 100(1):29–44, 2012.

[23] T. Yokoyama. An Aspect-Oriented Development Method for Embedded Control Systems with Time-Triggered and Event-Triggered Processing. In *Proceedings of the 11th IEEE Real Time on Embedded Technology and Applications Symposium*, RTAS '05, pages 302–311, 2005.

[24] H. Zabel, W. Müller, and A. Gerstlauer. Accurate RTOS Modeling and Analysis with SystemC. In W. Ecker, W. Müller, and R. Dömer, editors, *Hardware-dependent Software*, chapter 9, pages 233–260. Springer Netherlands, 2009.