Delay-based distribution and optimization of a simulation model

Clément Michel, Pierre Siron

▶ To cite this version:

Clément Michel, Pierre Siron. Delay-based distribution and optimization of a simulation model. 2018 IEEE/ACM International Symposium on Distributed Simulation and Real Time Applications (DS-RT), Oct 2018, Madrid, Spain. pp.21-28. hal-01916686

HAL Id: hal-01916686
https://hal.science/hal-01916686

Submitted on 8 Nov 2018

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Open Archive Toulouse Archive Ouverte (OATAO)

OATAO is an open access repository that collects the work of some Toulouse researchers and makes it freely available over the web where possible.

This is an author's version published in: https://oatao.univ-toulouse.fr/21037

To cite this version :

Michel, Clément and Siron, Pierre Delay-based distribution and optimization of a simulation model. (2018) In: 2018 IEEE/ACM International Symposium on Distributed Simulation and Real Time Applications (DS-RT), 15 October 2018 - 17 October 2018 (Madrid, Spain).

Any correspondence concerning this service should be sent to the repository administrator:
tech-oatao@listes-diff.inp-toulouse.fr

Delay-based distribution and optimization of a simulation model

Clément Michel
Pierre Siron
ISAE-SUPAERO, University of Toulouse
10 avenue Édouard Belin
BP 54032 - 31055 Toulouse CEDEX 4, France
Email: {firstname.lastname}@isae-supaero.fr

***Abstract*—The conception of Cyber-Physical Systems is a complex task: the multiple components making up those systems might not be fully known by the system architect, and putting those components together generates a new source of complexity. Study and validation of those systems is often done through simulation. Moreover, the CPS simulation is often studied through distributed simulation, as the CPS might itself be distributed or too complex.**
We present a methodology to distribute a simulation model in order to take full advantage of multiple processing units. We ensure that said distribution does not impact the simulation of our modeled system.

***Index Terms*—Embedded Simulation, Distributed simulation, Cyber-Physical Systems, HLA**

I. Introduction

CPSes models are "architecture models of a system that consist of software, hardware, and physical system components, their interactions, and the properties of these elements" [1].

The analysis of cyber-physical systems (CPS) is a complex task due to the heterogeneity of the parts involved, as they integrate physical components as well as computing resources. As such, CPSes integrate multiple models of computation; for instance a rotating motor could be represented in the continuous domain while the control chip regulating it might be represented in the discrete event domain. Analyzing such systems requires to handle the heterogeneity of their components and representations. Thus, they require specific tools for modeling and analysis.

In Section II, we discuss tools for modelization and simulation of cyber-physical systems before discussing distributed simulation using the High Level Architecture standard. Our distribution methodology is detailed in Section III, and examples are provided on Section IV.

II. Overview of CPS and distributed simulation

A. Modeling tools for cyber-physical systems

Synchronous languages fits these requirements, such as SCADE and its more specific subsets LUSTRE [2] and PRÉLUDE [3]. Those two languages specialize onto different types of systems: LUSTRE focuses onto the modelling of synchronous systems, while PRÉLUDE focuses onto validating the timing considerations of the modelled system.

Model-based engineering can also be used, with its focus on architecture definition: components are individually defined, and relations between said components are then specified. Some of the languages used in model-based engineering are SysML, UML or AADL [4], [5]. Simulink [6] is a graphical tool allowing conception and simulation using the principles of model-based engineering. PTIDES, a modeling tool implemented in the Ptolemy simulator, accounts for the differences in clock synchronization that can be encountered for actual systems with multiple computing resources requiring precise timing. This representation of time, close to how embedded systems interacts with time, also ensures more precisely that deadlines are enforced through both static analysis and simulation [7].

Thanks to this model-based approach, the complexity of the constitutive components is self-contained and the internals of a components can be as complex as needed without requiring the designer to be aware of it.

In this paper, we chose to focus on model-based techniques: the systems we want to study are composed of asynchronous parts that would require modification or encapsulation in order to be expressed in a synchronous language. Moreover, the component approach used in model-based engineering favors interoperability and reuse, a theme also present in HLA and discussed in Section II-C.

B. Simulation

CPSes are difficult to assess from a formal point of view [8]. The use of simulation allows to gain knowledge about CPSes without the need for formal analysis. However, the simulation needs to be representative of the actual system as well as being reproducible. In other words, fidelity is required both in modeling and simulating the modeled system.

Simulation of CPSes distinguishes itself from simulation of multiple computing devices because of the former's tight interaction with physical processes. In Cyber-Physical Systems and contrary to multicore/distributed computation, the passage of time is a central feature, and time coordination between the different system components need to represent the modeled system to a desired degree of faithfulness. The Ptolemy simulator [9] handles this complexity by using heterogeneous multimodeling [10], where the different Models of Computation

(MoC) can be nested within each other, and combined hierarchically.

C. Distributed simulation using HLA

Distributed simulation brings forth its own set of simulation problems, such as the loss of a total event ordering [11], proof of correctness (or lack thereof) and deadlock prevention [12].

Propositions such as the High Level Architecture (HLA) standard [13] allows developers to add a distributed layer on top of an existing simulator. HLA is suitable for hard real time constraints [14], thus appropriate for modeling our time-centric models. That distributed layer separates the distributed simulation into *federates* (simulations taking part in a distributed HLA simulation, called *federation*), that exchange data exclusively through the RTI, a middleware that implements the HLA rules and manages the communication between the different federates. This loose coupling of this distributed layer allows for interoperability and federate reuse. HLA is then used to couple simulators together. Some of those coupling implementations are PTII-HLA [15], or FMI-HLA [16]. A notable open-source HLA implementation is CERTI [17].

As outlined by previous works, the distribution of a simulation comes at a cost [18]: the conservative time management services allow for the coordinated and safe time advance of each federate through conservative simulation.

To ensure the conservative property of these time advances, a federate is forbidden to send messages to others federates before $t + lah$, with t being the current time for the federate and lah being a property of the federate called the lookahead. It might thus be necessary to introduce a delay equal or greater than lah in order to make the simulator compliant with the HLA requirement. HLA allows for $lah = 0$, however this zero lookahead will not be considered in this paper because of the different costs it carries, both in design cost and in simulation cost.

HLA's *Next Message Request* service (abbreviated as *NMR*) ensures that the logical time of the federate shall be advanced to the timestamp of the next timestamped message that will be delivered.

Let t be the time of a federate f_1 sending messages, and lah its lookahead. It sends its message to another federate f_2, and both federates uses the *NMR* service. The federate wishes to send a message timestamped $t + d$.

If $t + d < t + lah$, then the federate cannot send this message.

If $t + d \geq t + lah$, then the federate sends this message with timestamp $t+d$. The receiving *NMR* federate receives this message with the same timestamp $t + d$.

Because of this potential modification, observations and properties established in a (centralized) simulation does not necessarily carry over to the distributed version: the distributed model is then different from the centralized one.

D. Motivation

The rising complexity of CPSes can be traced down to a multitude of factors. For instance, more precise simulation with smaller simulation steps, or more precise simulation algorithms can induce more expensive computation. Moreover, the CPS itself can be complex. In this paper, we will focus on CPS simulation using heterogeneous multimodeling and a discrete event simulation at the top of our hierarchy.

To take full advantage of multicore architectures, it is mandatory for programs to be able to run on multiple cores at the same time, through the use of threads assigned to different physical or logical cores. This approach also naturally extends to multi-computer architectures through mapping to cores on distinct computers, thanks to the HLA framework that does not rely on shared memory.

For the remainder of this paper, we call a model able to execute onto different cores at the same time a *distributed model*, whereas a model that can only run onto a single core is called a *centralized model*.

Whereas the model can be created already distributed, as it is the case with systems that are distributed by nature, some systems are designed centralized from the ground up. Even a model created to simulate a centralized model can benefit from distribution, as its simulation can be sped up under the right conditions.

Our main motivation is to guide the user by proposing a model representation that will be both a modeling support and a cornerstone for our distribution process.

III. From centralized to distributed

We propose a methodology to transform centralized discrete event models into distributed discrete event ones. This methodology focuses heavily on making sure the resulting distributed model will yield the same results as the centralized one.

The process is as follows: we first split the model into clusters that will become components of the distributed simulation. We then modify the clusters to account for the inserted distributed mechanisms, and finally we determine the distributed parameters allowing for the fastest simulation possible.

A. Representation of models

We discuss here a representation graph aimed at abstracting our model. Our ideal representation is as minimalistic as possible, abstracting the studied models as much as possible but still faithfully representing both the structure and (some) timing elements of the modelled system.

This representation takes inspiration from both event graphs and dataflow diagrams. Event graphs [19] represent event scheduling using edges (representing state transitions), that connect nodes (representing events). These edges can integrate a time delta, representing a delay between two event $e_1(t_1)$ and $e_2(t_2)$. The existence of an edge means that event $e_2(t_2)$ can be generated during the processingof event $e_2(t_2)$.

Whereas the properties on edges are very interesting in our scenario, we would like to represent how events are circulating through the system rather than representing the possible sequences of events.

In this prospect, our representation is closer to a dataflow diagram, which describes processes and their relations [20]. However, a dataflow diagram does not have a notion of time. Since we want to capture time relations within the system, a dataflow diagram is not here the best match.

Finally, we would like to perform operations onto said graph in a manner that is close to a program dependence graph [21], albeit without representing explicitely both data and control dependencies.

The discussed representation takes the meaning of weighted edges from event graphs in order to represent the scheduling of events, and takes the meaning of nodes from dataflow diagrams in order to represent processes.

In the remainder of this paper, the following representations and notations will be used. Every reference to time is a reference to logical time, until stated otherwise.

Let A be a component in a model. A component is a "black box", represented by a rectangle in Figure 1 that takes in events and produces events. We have no interest in the component's inner workings: a component can be a submodel, a single function, etc... without affecting its interactions with the rest of the modelled system.

We consider that when component A is at time t, it ***can*** generate a new event e_A, of value v and of timestamp t'.

Events generated by a component may be sent to another component. This behavior is modeled by a weighted directed edge, starting from the component generating the event, and pointing towards the component receiving the event.

The weight on the edge represents the time between the emission of an event e_A by A, and the reception of e_A by B. For instance, with a weight of d on the edge linking A to B, the production of an event at time t by A means that B will receive an event $e_A(t^*, v)$ with $t^* = t + d$ and v the value of the event. This event will be received at time t^*.

An edge with an unspecified weight, such as the one on depicted Figure 1 is an edge of weight 0. The dotted edge means that A is part of a bigger model and receives events from this edge in an unspecified manner. This is the case for the initial events in the model.

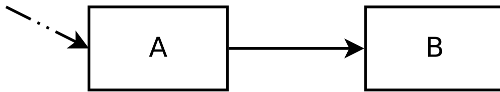


Figure 1: Component A sends data to component B.

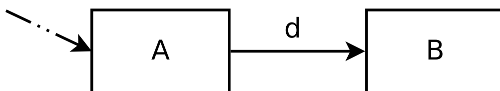


Figure 2: The same scenario, with a delay d between A and B.

Thanks to this representation, we are able to express delays in HLA the same way we express delays in a discrete event simulation: A delay d introduced by HLA has the same representation than a delay d existing within the discrete event simulation. For instance, a component A at t sending an event e_A at $t + d$ will be represented in the same way than a component A at t sending an event $e_A(t + d)$ to the *RTI*, with the *RTI* delivering the event at $t + d$. For HLA specialists, the services used are `updateAttributeValues` and `reflectAttributeValues` with a timestamp.

This abstraction level will allow us to change the origin of a delay at will without changing the behavior of the modelled system, thus allowing us to replace delays from the discrete event simulation with HLA-induced delays.

B. Delay implementation equivalence

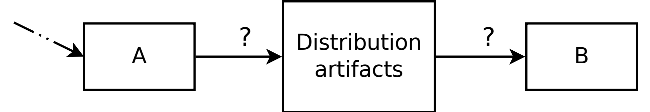


Figure 3: The model on Figure 2 now distributed.

We model the distribution mechanisms as an added component, called `Distribution artifacts` in Figure 3. In order to ensure that the distribution does not impact the output, events received by B in Figure 2 should be the exact same events than received by B in Figure 3. Let $e_A(t^*, v)$ be the event received by B in Figure 2 and $e'_A(t', v')$ the event received by B in Figure 3. We obtain the following conditions:

$$\begin{cases} \exists e_A \iff \exists e'_A & (1) \\ t^* = t' & (2) \\ v = v' & (3) \end{cases}$$

Condition 1 states that for every event e_A, another event e'_A must exist : in other words, the `Distribution artifacts` component in Figure 3 must transport events without deleting or creating new ones. The HLA standard defines different transport types. One of them is the *reliable* transport type, that ensures that delivery of one copy of the data is guaranteed [13]. Coupled with the fact that HLA isn't delivering data that isn't supposed to be delivered, using the *reliable* transport type is sufficient to ensure that Condition 1 is respected.

Moreover, HLA ensures integrity of the delivered message [13], thus validating Condition 3.

Recalling the properties of *NMR* discussed in Section II-C , if $d \geq lah$ with $lah > 0$, the message passing through the *RTI* behaves as if it have been delayed by an edge of weight d.

A lookahead

$$lah \leq d \qquad (4)$$

is thus a property sufficient to ensure Condition 2 for an edge of weight d. At this point, lah is undefined, and we only know that $lah > 0$. We will determine its value in a later step.

As a consequence of these properties, Figure 4 can represent/be implemented both by HLA in the simplified Listing 1 and Ptolemy in Figure 5.

In a Ptolemy model, the components are called actors. A model is made of a director and multiple actors. The director dictates how time advance in the model, and how actors

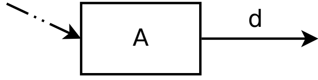


Figure 4: A component A with an edge of weight d.

interact together. In this article, we will focus onto Ptolemy models with a DE Director, implementing a discrete event MoC. A special actor called `TimeDelay` allows to modify the timestamp of its input events, simulating the latencies in the model. Without any `TimeDelay` actors, the model time would never advance.

Listing 1: Simplified HLA implementation of Figure 4.

```
nextMessageRequest(t_end)
reflectAttributesValues(object1, v1, t)
timeAdvanceGrant(t)
v2 = computeOutputValue(v2, t)
updateAttributeValues(object2, v2, t + d)
```

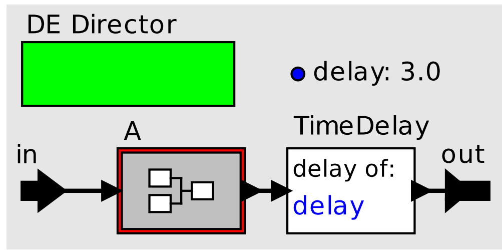


Figure 5: Ptolemy implementation of Figure 4.

C. Distribution methodology

Our approach for creating a distributed simulation is composed of two steps: first, we find the maximum number of federates, then we compute the lookahead of said federates in order to obtain the fastest simulation possible.

1) Clustering: In the first step of our methodology, we determine a way to split our centralized model into multiple federates.

On Figure 6, A and B cannot end up in two different federates, as a 0-weight edge links them together. Indeed, according to Equation 4, the lookahead would be 0, and as presented on Section III-B we only consider $lah > 0$. Let $sp(A, B)$ be the shortest path between A and B. Two components cannot end up being in two different federates if $sp(A, B) = 0 \lor sp(B, A) = 0$.

We thus obtain this sufficient rule for finding the maximum distribution obtainable: if there is a 0-weight edge between two components A and B (that is, if $sp(A, B) = 0 \lor sp(B, A) = 0$) then A and B must be in the same federate.

Our approach allows to find the maximum number of federates possible. However, this maximum may not be suitable or necessary. If maximum distribution is not desired, federates can be "fused" together by putting the contents of two federates in a single one.

2) Lookahead computation: Once the clustering step realized, we obtain a list of federates, each with a lookahead $lah > 0$. To make the simulation as fast as possible, we would like a value of lah as great as possible, as it means less mandatory waiting for other federates [13], thus

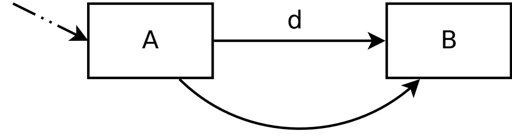


Figure 6: Weighted link that cannot be distributed.

less idle time and greater efficiency. However, we cannot set a lookahead greater than what an edge allows, as it would change the edge and thus change the model. Let an edge originating from a component in a federate f_1 and pointing towards a component in federate f_2. We call that edge an outgoing edge for f_1. Let E_d be the set of all outgoing edges for f_A. The lookahead for f_A will be:

$$lah = \min_{d \in E_d} d \tag{5}$$

In other words, the lookahead is the smallest weight on all outgoing edges.

On Figure 7, components A, C, E can all be reached from D through 0-weight edges and will be part of the same federate f_1. As such, f_1 will have three outgoing edges d_1, d_2, d_3. Thus, its lookahead will be $\min(d_1, d_2, d_3)$.

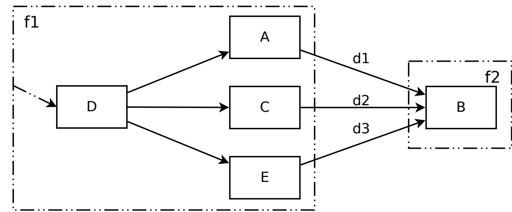


Figure 7: Future federate $f1$ with multiple outgoing edges.

3) Edge reweighting for Ptolemy-HLA implementations: We will use the Ptolemy-HLA framework [22] in order to implement our distributed smulation. This framework allows a Ptolemy model to interact with a RTI through the `HlaManager` that handles the high-level HLA rules and services, and two actors: the `HlaPublisher` that sends messages to the HLA federation, and the `HlaPublisher` that receives messages from the HLA federation.

A `HlaPublisher` executed at time t will send an event to the federation at time $t_{send} = t + lah$. This behavior is a property that have been chosen for the framework in order to let the system architect focus on the Ptolemy implementation more than on HLA fine-tuning. Since $lah \geq d$, we need to insert special delays in the Ptolemy model to accomodate the specifications of the Ptolemy-HLA framework.

We compute a value d' such as

$$d = lah + d' \tag{6}$$

In other words, the weight d is decomposed into two constitutive delays: the delay lah that will be implemented using HLA, and the d' delay, that will be implemented in-model, using `TimeDelay`-induced delays.

For federates with a single outgoing weight, we have $lah = d$ as according to Equation 5, thus $d' = 0$. As a consequence, for federates with a single outgoing weight,

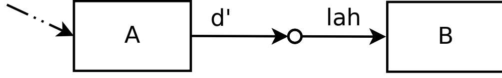


Figure 8: Weight from Figure 2 separated as HLA weight and non-HLA weight.

the delay d' does not exist, and the whole weight of the edge is implemented using HLA. We choose a lookahead as arbitrarily large as possible for it not to slow down the simulation.

D. `GRADE-II` implementation

A Python implementation of our algorithm has been written, called `GRADE-II` for *Graph Requirements for an Automatic Distribution Environment*. This implementation takes in a graph and returns the different federates that composes the now distributed model using the steps described in Section III-C, including the edge reweighting specific for Ptolemy-HLA implementations.

IV. Examples

A. A small example

We apply our distribution methodology to a Ptolemy model depicted in Figure 9 using the developped tool `GRADE-II`. In this model, the `PoissonClock` and `DiscreteClock` generate events that are sent to their respective `TimeDelay`. Its equivalent graph using the representation proposed in Section III-A is depicted in Figure 10.

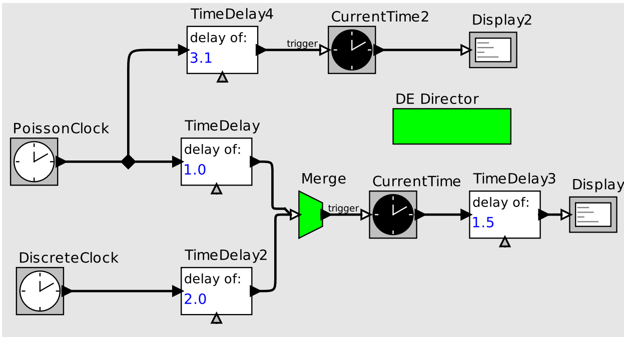


Figure 9: A Ptolemy model we wish to distribute.

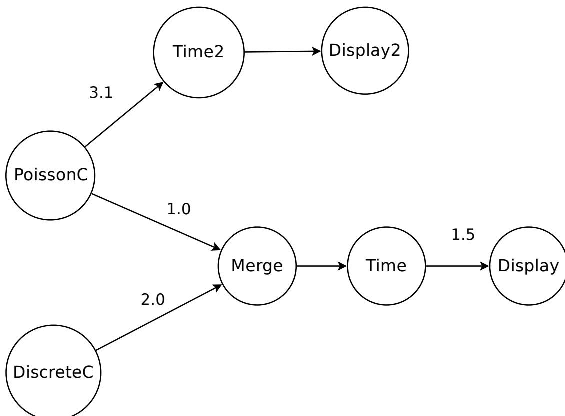


Figure 10: Figure 9 expressed as a graph.

Clustering the graph as described in Section III-C1 yields the 5 federates described in Figure 11.

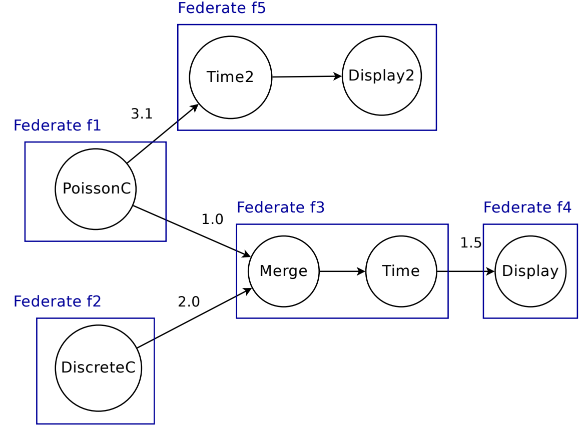


Figure 11: Figure 10 after clustering.

The next step is to compute the lookahead for each of the 5 federates. Applying Equation 5, we obtain $1.0, 2.0, 1.5$ for respectively federates f_1, f_2, f_3. Federates f_4, f_5 do not have outgoing edges, thus the lookahead can be arbitrarily large and is not specified.

Finally, the weight of the outgoing edges is reevaluated using Equation 6. Federates f_2, f_3 only have one outoging edge, and as such have $d' = 0$. Federate f_1 have two outgoing edges: one takes a new weight d' of 0 while the other takes a new weight of $3.1 - lah = 2.1$. The resulting federation is depicted in Figure 12. This federation was implemented using the Ptolemy-HLA framework and is represented on Figure 13. Each federate on Figure 12 is implemented as a Ptolemy model with a `HlaManager`. Outgoing edges start with a `HlaPublisher` and end with a `HlaSubscriber`.

Executing both the centralized model in Figure 9 and the distributed one in Figure 13, we obtain the same results for the two executions.

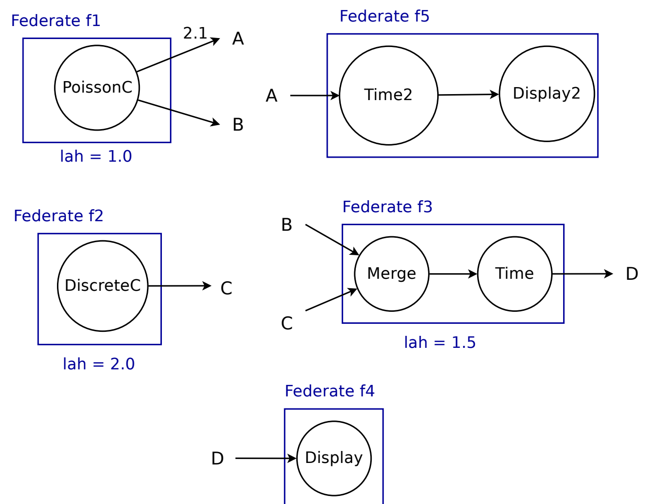
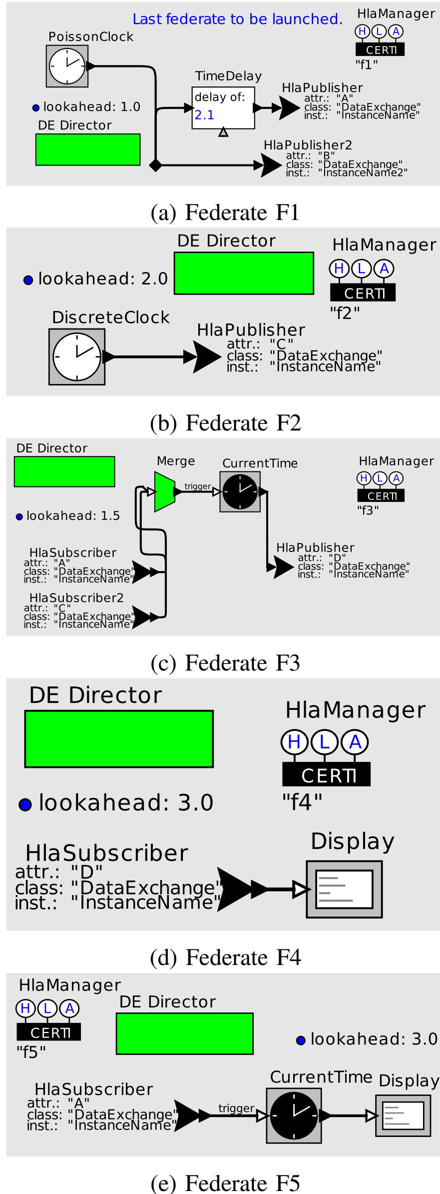


Figure 12: Lookahead and delays for Figure 11's federates.

V. The F-14 case study

In this section, we look at how variations of a model influence the distribution through study of an F-14 aircraft's



(a) Federate F1

(b) Federate F2

(c) Federate F3

(d) Federate F4

(e) Federate F5

Figure 13: Ptolemy models obtained from the Figure 12.

longitudinal control. This system was implemented both as a Simulink demo and later on, as a Ptolemy demo.

We study the autopilot system of an F-14 Tomcat twin-engine fighter aircraft. As such, we identify three major components for our model: the physical structure of the plane, the integrated controller handling the autopilot, and the pilot's inputs. Their interactions are shown on Figure 14.

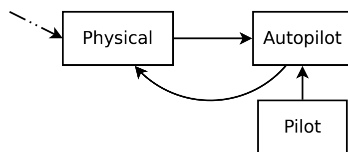
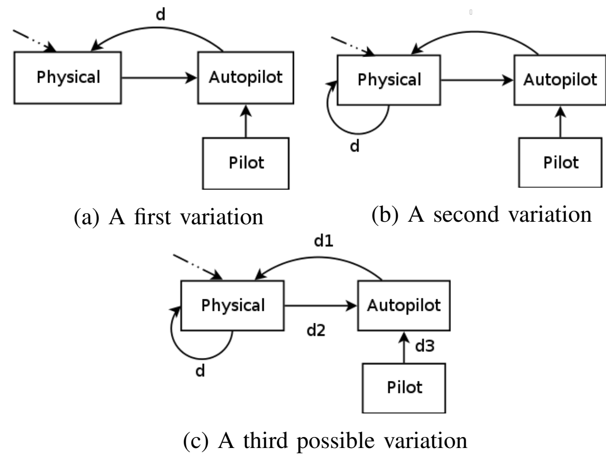


Figure 14: Interactions between the different f14 parts.

As we would like to model these interactions in a Discrete Event simulation, we have to insert delays in our model in order to represent the passage of time. We propose here three ways of implementing said delays, represented on Figure 15. We would like to highlight the fact that there is no "best" variation: every variation produces slightly different results due to the emphasis onto different time architectures. For instance, the variation on Figure 15b puts the emphasis on an feedback loop on the physical part and the autopilot, that occurs once every d.

Recalling that in the described representation, an edge represents a *possibility* to send a message. It is thus implicit that the edge from `Autopilot` to `Physical` is triggered by the reception of an event going through the edge of weight d: in this representation, we can have a 0-weight loop between `Physical` and `Autopilot`. This representation does not fully reflect the knowledge we have of the system because the variation on Figure 15c takes into account the autopilot computation delay and the delay between the state of our physical process and the reading of said state through a sensor.



(a) A first variation

(b) A second variation

(c) A third possible variation

Figure 15: Three different ways of representing the F-14 autopilot loop.

The F-14 variation pictured in Figure 15b is one of the demonstration models that comes with the Ptolemy simulator. The heterogeneous multimodeling abilities of Ptolemy allows here to model the `Physical` component internals in a Continuous Model of Computation. Nonetheless, the whole `Physical` component behaves as if it is a traditional Discrete Event component. A `PeriodicSampler` actor within the Continuous MoC simulates a periodic sensor with an analog-to-digital converter. It can be seen as a self-edge on the Physical component, since new events are produced with a period D — in other words, events are produced with an interval d between them.

Attempting to distribute it with the methodology described in Section III-C does not yield a successful distribution, as there are 0-weight edges connecting the components.

Finally, the third variation, depicted in Figure 15c, can be distributed as there are no 0-weight edges linking the `Physical`, `Autopilot` and `Pilot` together.

With $d_1 = d_2 = d_3 = 0.005$, our methodology creates 3 federates, each with a lookahead $lah = 0.005$. Execution of our distributed simulation took 3.9 seconds.

Changing the lookahead to a smaller value and adding `TimeDelay` actors to keep the same event timestamps, we obtain significantly slower executions: an execution with $lah = 0.001$ took 6.2 seconds, and an execution with $lah = 0.0001$ took 25.5 seconds. This demonstrates the importance of a well-chosen lookahead in the simulation process.

The distributed version, as well as the centralized one, are downloadable with Ptolemy as demonstration models.

Other examples have been successfully studied, such as SDSE [23], but cannot be presented because of space constraints.

VI. Conclusion and perspectives

We have introduced a representation that requires very little assumptions onto the model. This representation is used to identify how a model can be sliced into submodels for distribution purposes.

We identified two use cases for our distribution methodology. The first one takes an already existing centralized model, expresses it in the discussed representation and find a distribution that produces the same results (if the distribution exists). The second one takes in a graph expressed in our representation and find a distribution compliant with the architecture and delays specified by the system architect.

We believe that this automatic distribution can be integrated as a feature of a larger framework. This framework would make extensive use of the representation introduced in this paper, extending it with new features and characteristics that would not impact the ones described in this paper. The hypothesized framework would allow a model architect to develop a model out of a library of existing components, test the soundness of the constructed model by expressing and testing constraints onto component connection (for instance to detect algebraic loops), and attempt to distribute it before proposing an implementation using a chosen simulator.

However, not all models can be distributed without altering them. In some usecases, we believe the model architect would obtain more from a slightly changed, fast and parallelized model than from a exact, slow and centralized model. We are convinced that providing the model architect with a way to distribute the studied model while inserting a minimum amount of alterations into said model allows to keep the distributed behavior as close as possible from the original.

Finally, HLA's zero lookahead option will be examinated, as well as compatibility between faithful distribution and HLA's *Time Advance Request* will be studied, to allow more diverse distributions and maybe better performance gains.

Acknowledgments

This research was partly supported by the French Ministry of Defense through financial support of the Direction Générale de l'Armement and by the Occitanie region.

References

[1] G. Yang and X. Zhou, *Cyber-Physical Systems*. 2013.

[2] N. Halbwachs, P. Caspi, P. Raymond, and D. Pilaud, "The synchronous data flow programming language LUSTRE," *Proceedings of the IEEE*, vol. 79, pp. 1305–1320, Sept. 1991.

[3] J. Forget, F. Boniol, D. Lesens, and C. Pagetti, "A Multi-Periodic Synchronous Data-Flow Language," in *2008 11th IEEE High Assurance Systems Engineering Symposium*, pp. 251–260, Dec. 2008.

[4] P. H. Feiler and D. P. Gluch, *Model-Based Engineering with AADL: An Introduction to the SAE Architecture Analysis & Design Language*. The SEI series in software engineering, Upper Saddle River, N.J: Addison-Wesley, 2013. OCLC: 820515268.

[5] F. Cottet and E. Grolleau, *Systèmes temps réel embarqués: Conception et implémentation*. 2014. http://sbiproxy.uqac.ca/login?url=http://international.scholarvox.com/book/88825521.

[6] MathWorks, "Verification, Validation, and Test - MATLAB & Simulink Solutions." https://fr.mathworks.com/solutions/verification-validation.html.

[7] J. C. Eidson, E. A. Lee, S. Matic, S. A. Seshia, and J. Zou, "Distributed Real-Time Software for Cyber-Physical Systems," *Proceedings of the IEEE*, vol. 100, pp. 45–59, Jan. 2012.

[8] E. A. Lee, "CPS foundations," in *Design Automation Conference*, pp. 737–742, June 2010.

[9] C. Ptolemaeus, ed., *System Design, Modeling, and Simulation Using Ptolemy II*. Ptolemy.org, 2014. http://ptolemy.org/books/Systems.

[10] C. Brooks, C. P. Cheng, T. H. Feng, E. A. Lee, and R. Von Hanxleden, "Model engineering using multimodeling," tech. rep., California Univ. of Berkeley, Dept. of Electrical Engineering and Computer Science, 2008.

[11] L. Lamport, "Time, Clocks, and the Ordering of Events in a Distributed System," *Commun. ACM*, vol. 21, pp. 558–565, July 1978. http://doi.acm.org/10.1145/359545.359563.

[12] K. M. Chandy and J. Misra, "Distributed Simulation: A Case Study in Design and Verification of Distributed Programs," *IEEE Transactions on Software Engineering*, vol. SE-5, pp. 440–452, Sept. 1979.

[13] IEEE-SA Standards Board, *IEEE Standard for Modeling and Simulation (M & S) High Level Architecture (HLA): Federate Interface Specification*. New York: Institute of Electrical and Electronics Engineers, 2010. http://ieeexplore.ieee.org/servlet/opac?punumber=5557726.

[14] C. Gervais, J. B. Chaudron, P. Siron, R. Leconte, and D. Saussié, "Real-Time Distributed Aircraft Simulation through HLA," in *2012 IEEE/ACM 16th International Symposium on Distributed Simulation and Real Time Applications*, pp. 251–254, Oct. 2012.

[15] G. Lasnier, "Towards a Distributed and Deterministic Framework to Design Cyber-Physical Systems," tech. rep., ISAE-Supaero, Toulouse, France, Oct. 2013.

[16] M. U. Awais, P. Palensky, A. Elsheikh, E. Widl, and S. Matthias, "The high level architecture RTI as a master to the functional mock-up interface components," in *2013 International Conference on Computing, Networking and Communications (ICNC)*, pp. 315–320, Jan. 2013.

[17] "CERTI Home page." http://savannah.nongnu.org/projects/certi, July 2002.

[18] C. Michel, J. Cardoso, and P. Siron, "Time Management of Heterogeneous Distributed Simulation," in *Proceedings of the 31st European Simulation and Modelling Conference (ESM) - Modelling and Simulation*, (Lisbon, Portugal), pp. 343–349, EUROSIS-ETI, Oct. 2017.

[19] A. Buss, "Basic Event Graph Modeling," 2001. https://calhoun.nps.edu/handle/10945/45519.

[20] P. D. Bruza and T. P. van der Weide, "The Semantics of Data Flow Diagrams," in *In Proceedings of the International Conference on Management of Data*, pp. 66–78, McGraw-Hill Publishing Company, 1993.

[21] J. Ferrante, "The Program Dependence Graph and Its Use in Optimization," *ACM Transactions on Programming Languages and Systems*, vol. 9, p. 31, July 1987.

[22] G. Lasnier, J. Cardoso, P. Siron, C. Pagetti, and P. Derler, "Distributed Simulation of Heterogeneous and Real-Time Systems," in *17th IEEE/ACM International Symposium on Distributed Simulation and Real Time Applications*, (Delft, Netherlands), pp. 55–62, IEEE, Oct. 2013. http://ieeexplore.ieee.org/document/6690494/.

[23] J.-B. Chaudron, D. Saucciè, P. Siron, and M. Adelantado, "How to solve ODEs in real-time HLA distributed simulation," in *Simulation Innovation Workshop (SIW)*, (Orlando, United States), p. 13, Sept. 2016.