Information and Communication Technologies (ICT) Programme

Project N°: H2020-ICT-2016-1-732105

D3.5: Models of Computation

Lead Beneficiary: INSA
Workpackage: WP3
Date: 08/06/2018
Distribution - Confidentiality: [Public/Confidential]

Abstract:

This documents surveys state-of-the-art Models of Computation (MoC) used for the design of Cyber-Physical Systems. The MoCs used within the CEBRERO Project are specified and the planned innovations are presented in the last section of this document.

Disclaimer

The CERBERO Consortium is the following:

Num.	Beneficiary name	Acronym	Country
1 (Coord.)	IBM Israel – Science and Technology LTD	IBM	IL
2	Università degli Studi di Sassari	UniSS	IT
3	Thales Alenia Space Espana, SA	TASE	ES
4	Università degli Studi di Cagliari	UniCA	IT
5	Institut National des Sciences Appliquees de Rennes	INSA	FR
6	Universidad Politecnica de Madrid	UPM	ES
7	Università della Svizzera Italiana	USI	CH
8	Abinsula SRL	AI	IT
9	Ambiesense LTD	AS	UK
10	Nederlandse Organisatie Voor Toegepast Natuurwetenschappelijk Ondeerzoek TNO	TNO	NL
11	Science and Technology	S&T	NL
12	Centro Ricerche FIAT	CRF	IT

For the CERBERO Consortium, please see the http://cerbero-h2020.eu web-site.

Document Authors

The following list of authors reflects the major contribution to the writing of the document.

Name(s)	Organization Acronym
Karol Desnos	INSA
Maxime Pelcat	INSA
Julio Oliveira	TNO
Carlo Sau	UNICA
Luca Pulina	UNISS
Eduardo de la Torre	UPM
Eduardo Juarez	UPM
Pablo Muñoz	S&T
Rubén Salvador	UPM
Antoine Morvan	INSA
Francesca Palumbo	UNISS
Michael Masin	IBM

Document Revision History

Date	Ver.	Contributor (Beneficiary)	Summary of main changes
2017.12.18	0.1	Karol Desnos (INSA)	Table of content draft
2018.01.22	0.2	Karol Desnos (INSA)	Table of content update
2018.02.12	0.3	Karol Desnos (INSA) UPM, UNISS, UNICA	Plan list of surveyed MoC by UPM, UNISS, UNICA
2018.02.20	0.4	Karol Desnos (INSA) TNO	Plan list of surveyed MoC by TNO. Paragraph template for section 5.

2018.03.08	0.5	Karol Desnos (INSA) Maxime Pelcat (INSA) Julio Oliveira (TNO) Carlo Sau (UNICA) Luca Pulina (UNISS) Eduardo de la Torre (UPM) Eduardo Juarez (UPM) Pablo Muñoz (S&T)	Completion of section 4 & 5.
2018.03.21	0.5r	Rubén Salvador (UPM)	Review
2018.03.22	0.6	Antoine Morvan (INSA)	Process review comments; add links with D2.7 requirements; insert and complete MoC summary table.
2018.03.29	0.6r	Francesca Palumbo (UNISS)	Final review.
2018.03.29	0.7	Antoine Morvan (INSA)	Process review comments.
2018.04.09	1.0	Francesca Palumbo (UNISS) Antoine Morvan (INSA)	Final modifications.
2018.06.08	1.0r	Michael Masin (IBM) Antoine Morvan (INSA)	Revision from Michael; add reference on MoA.

Table of contents

1. Executive Summary

This documents surveys state-of-the-art Models of Computations (MoCs) used for the design of Cyber-Physical Systems (CPS), and it outlines the main characteristics of MoCs used for CPS design by presenting:

- the properties of their semantics (analyzability, decidability, reconfigurability, expressiveness, determinism, ...),
- the kind of algorithm it supports (data-driven, control-driven, …),
- the level of abstraction it captures (system-of-systems, system, component, …)
- the type of implementation it translates into (hardware, software, distributed, …).

The objective of this document is to give enough information to CPS designers to choose the MoC that best suit their needs.

As an example of this document utility, a study of most suitable MoCs for designing key features of the CERBERO use-cases is presented. Based on this study, we identify lacks in current MoCs semantics and we define a set of new MoC features needed to support the design of CERBERO use-cases, which will be developed during the project. Those features will advance state of the art and will allow these MoCs to be more effectively adopted in the CPS context.

1.1. Structure of Document

Section 2 of this document defines the notions of abstraction and models, which serve as a basis to the concept of *Models of Computation*. Section 3 introduces a set of properties of MoCs that are then used in Section 4 to characterize and compare state of the art MoCs commonly used for the design of CPSs. Finally, Section 5 presents the expected CERBERO innovations in the domain of MoCs for the modeling of CPS.

1.2. Related Documents

- D2.7 - CERBERO Technical Requirements
  - D3.5 contributes to satisfy D2.7 requirements. Details are given in Section 1.3.
- D3.4 - KPI Modeling
  - The KPIs can be used to represent the system properties, which can be verified and guaranteed with varying degrees of ease depending of the selection of the Model of Computation.
- D3.6 - Cross-layer Modelling Methodology for CPS
  - The models of computation described in this document are used to represent one aspect of the CPS, the behavior. This is a key foundation in the cross-layer modelling methodology.
- D5.6 - CERBERO Framework Components
  - D5.6 gives more details on the MoCs supported by the tools that are components of the CERBERO framework.

1.3. Related CERBERO Requirements

Deliverable D2.7 of the CERBERO project defines a list of CERBERO Technical Requirements (CTRs) the project should achieve. Each of them is referenced with a unique identifier ranging from 0001 to 0020. MoC exploration and innovation are carried out following the requirements in Table 1-1.

CTR id	**CTR Description**	**Link with the D3.5 document on *Models of Computation***
0001	CERBERO framework SHOULD increase the level of abstraction at least by one for HW/SW co-design and for System Level Design.	Innovations on MoCs help raising the abstraction level for the designer
0002	CERBERO framework SHOULD provide interoperability between cross-layer tools and semantics at the same level of abstraction.	Formalization of MoCs and homogeneity among partners foster tool interoperability
0007	CERBERO framework SHALL define methodology and SHOULD provide library of reusable functional and non-functional KPIs.	Non-functional KPIs can be influenced in the MoCs using proposed Moldable Parameters
0020	CERBERO framework SHALL provide methodology and tools for development of adaptive applications.	Proposed innovations on MoCs improve the expressiveness and specify the semantic of PiSDF for designing adaptive applications

Table 1-1: Links to CERBERO Technical Requirement

2. Models of computation

This first section briefly defines the core concepts of abstraction, model, and model of computations.

2.1. Abstraction

In general, abstraction is a tradeoff between the level of details and the complexity adopted when describing or representing a thing (e.g., an idea, a system, a place, an object, a phenomenon, etc.). Two distinct representations used to describe the same thing, each adopting a different abstraction tradeoff (i.e., amount of details conveyed about it), can be compared relatively to each other using so-called levels of abstraction.

- The lower level of abstraction gives a representation of the thing which is more detailed, thus giving a more precise and complete description.
- The higher level of abstraction gives a representation of the thing where some details are voluntarily omitted to decrease the complexity of the description. This higher complexity generally translates into a smaller and/or less dense representation of the thing.

2.2. Models

A model is a mathematically grounded representation capturing predictable characteristics of a system. More precisely, a model consists of a set of elements that can be assembled respecting a set of rules to describe a system. For a valid representation built with a model, mathematical equations associated to the elements of the model make it possible to predict some characteristics of the modeled system. Models are commonly used in all scientific fields to represent evolution of physical, computing, chemical, financial, or social systems.

For example, the symbol in Figure 1 – Bipolar Transistor Symbol and its associated equation in Figure 2 - Bipolar Transistor Equation are used to model and predict the voltage and current characteristics of a transistor within a model of an analog circuit.

Figure 1 – Bipolar Transistor Symbol

$$I_c = \beta I_s \left[1 + \frac{V_{ce}}{V_{EA}}\right] \exp\left(\frac{V_{be}}{V_{th}}\right)$$

Figure 2 - Bipolar Transistor Equation

In the context of cyber-physical systems (CPSs) engineering, several models adopting different levels of abstraction can be used to describe separated or nested aspects of a system. In particular the Models of Architecture (MoA) [Pelcat 2018] are used to describe the computing platform, often heterogeneous, including communication channels and memories. The application to be executed is modeled orthogonally using Models of Computation (MoC). More details on the use of heterogeneous models to describe a CPS are presented in D3.6.

2.3. Models of Computation

A Model of Computation (MoC) is a set of operational elements that can be composed to describe the behavior of an application. The set of operational elements of a MoC and the set of relations that can be used to link these elements are called the semantics of a MoC.

As presented in [Savage 1998], MoCs can be seen as an interface between the computer science and the mathematical domains. A MoC specifies a set of rules that control how systems described with the MoC are executed. Each element of the semantics of a MoC can be associated to a set of properties, such as timing properties or resource requirements. These rules and properties provide the theoretical framework that can be used to formally analyze the characteristics of applications described with a MoC. For example, using a mathematical analysis, it may be possible to prove that an application described with a given MoC will never get stuck in an unwanted state or that it will always run in a bounded execution time. Section 3 of this document describes a set of properties that are commonly supported by existing MoCs, which are themselves described in Section 4. A more extensive introduction to CPS modelling with MoCs can be found in [Lee 2017].

3. Characterization of Models of Computation

Since the introduction of modern computing systems in mid-1900s, a plethora of MoCs have been proposed by the scientific community. Very often, a new MoC is introduced to allow the specification of applications or systems that exhibit a set of characteristics whose specification was impossible or difficult to achieve with previously existing MoCs.

When designing a system, it is important to identify its required and desired properties. Once these have been identified, the designer can select the MoC whose semantics will make it easier to express, verify and guarantee those properties by construction.

The objective of this section is to give a definition of the properties used to characterize and compare the MoCs presented in Section 4.

3.1. Properties

This section lists a set of commonly used properties utilized to compare the system characteristics supported by different MoCs.

Analyzability

The analyzability of a MoC evaluates the availability of analysis and synthesis algorithms that can be used to characterize applications modeled with this MoC. For example, in the synchronous dataflow MoC, analysis algorithms can be applied at compile-time to compute the worst-case latency or the maximum memory requirements of a design.

Conciseness

The conciseness (or succinctness) of a MoC captures its ability to express complex system behaviors with a limited description size. This relative property is useful for comparing MoCs with equivalent expressiveness. Indeed, conciseness is often a desired feature for system developers as the design of an identical application with two MoCs (of identical expressiveness) will lead to a smaller design with the more concise MoC.

Compositionality

A modular MoC is compositional if the analyzable properties of a module described with this MoC are independent from the internal specification of the submodules that compose it [Ostroff 1995]. For example, in a compositional MoC, if each submodule used in the design is (independently) deadlock free, then the whole design combining these submodules will be deadlock-free by construction.

Decidability

A MoC is decidable if the schedulability of applications described with this model can be proved statically (i.e. at compile time) [Bhattacharyya 2006]. Hence, using a decidable MoC makes it possible to guarantee at compile-time that a system will never reach a deadlock state and that its execution will require a finite amount of memory. A non-decidable MoC does not mean that applications will not be schedulable, only that their schedulability can only be verified "on the fly" at runtime. Decidability is often obtained as a trade-off for a limited expressiveness of the MoC.

Determinism

A MoC is deterministic if the output of an algorithm only depends on its inputs, but not on external factors such as time or randomness. If determinism is a desired feature for most control and streaming applications, non-determinism may also be needed to describe applications reacting to unpredictable inputs.

Expressiveness

The expressiveness, of a MoC evaluates the complexity of application behaviors that can be described with this MoC. For example, the expressivity of the Dataflow Process Network (DPN) MoC has been proven to be equivalent to a Turing machine. The specialization of a MoC restricts the expressivity of this MoC to increase its analyzability, or to give it new properties such as determinism or decidability. Expressivity is often mistaken for conciseness. For example, the Cyclo-Static Dataflow (CSDF) MoC is often said to be more expressive than the Synchronous Dataflow MoC but meaning instead that it has a better conciseness.

Modularity

In a modular (or hierarchical) MoC, a system description can be broken into several independent modules. The modules that are combined to create a system can be (re-)used either in different systems specification or instantiated several times in the same. The modules themselves can be described using the same MoC as the top-level system description or can encapsulate other compatible MoCs.

Parallelism

In a parallel MoC, several independent elements of a system description may “activate” concurrently and independently from each other, each causing a change in the current state of the system. In a sequential (i.e. non-parallel MoC), all changes of the system state can be broken down to a sequence of actions triggered one after another, according to the system semantics.

Reconfigurability

A MoC is reconfigurable if the behavior of entire parts of a system description can be modified dynamically, to fulfill future execution goals for a foreseeable amount of time. Reconfiguration is used to dynamically adapt the behavior of a system to its environment, notably by enabling or disabling parts of the system, by modifying its functional behavior (e.g. its computations, QoS, …), or by modifying its non-functional properties (e.g. exposed parallelism, energy consumption, …).

Predictability

The predictability property is related to the reconfigurability property of a MoC. This property evaluates the amount of time between a reconfiguration of a part of the system, and the beginning of activity in the reconfigured part. The more predictable a MoC is, the more the time that can be used by a runtime manager to react and perform an optimization of the reconfigured part before using it.

3.2. Additional MoC Comparison Criteria

This section introduces a few other criteria that can be used to compare MoCs. These comparison criteria denote different classes of applications that a MoC can be used to represent. Unlike the properties presented in the Section 3.1, which capture properties supported (or not) by MoCs, this section introduces more subjective comparison criteria. Indeed, if some MoCs seem more suitable to implement a given class of applications, using them to implement another class may still be possible, but less practical or less common.

Algorithms Computation Classification

Algorithms described with a MoC can be classified into several classes depending on the type of involved computation:

- **Stream-based**: A continuous stream of data is steadily processed and produced by the described algorithm. The amount and nature of the computation do not vary depending on the data.
- **Data-driven:** The amount and nature of the computation do not vary depending on the data. Contrary to stream-based algorithms, data does not necessarily arrive continuously.
- **Control Driven**: The amount and nature of the computation depend on the processed data.
- **Event Driven**: Computations are triggered by events on the frontier of the system (i.e. by sensors, users, communication network, …).

Captured Algorithms Granularities

MoCs with different levels of abstractions are inherently suitable for representing behaviors of diverse granularities:

- **Function:** The modeled algorithm captures computations that are building blocks used to assemble an algorithm with a higher granularity.
- **Component:** The modeled algorithm serves a well-specified purpose with clear input and output interfaces and constraints.
- **System:** The modeled algorithm represents a collection of components with diverse objectives but running locally on a unique computing system.
- **System-of-systems:** The modeled algorithm consists of several independent "systems", each *existing and evolving* independently from the others but exchanging information among them through communication channels.

Implementation Types

A MoC is a theoretical representation used to describe the behavior of an application. Implementing a MoC consists in translating this theoretical behavior into an "executable" description. Different types of implementations can be more or less suitable to implement each MoC:

- **Hardware**: Algorithms described with this type of MoC can be efficiently translated into logical gates, signals, and registers on an ASIC or an FPGA.

- **Software**: Algorithms described with this type of MoC can be efficiently translated into a sequence of instructions executed on a processor that manipulates data stored in a memory space.
- **Distributed:** Algorithms described with this type of MoC can be efficiently implemented by splitting them into several parts executed on separate Hardware or Software components, each storing a part of the system state and executing a part of the computations in parallel.
- **Heterogeneous:** Algorithms described with this type of MoC can be efficiently translated into a mix of hardware and software implementations.

4. Surveyed Models of Computation

Using key characteristics of MoCs defined in Section 3, this section briefly introduces state-of-the-art Models of Computations used to specify Cyber-Physical Systems. Starting from MoC for hardware description, we increase the level of abstraction and expressiveness of models up to the system requirements level.

4.1. Register Transfer Level

MoC brief description

The Register Transfer Level (RTL) models are intended for detailing the behavior and the structure of hardware. Hardware Description Languages (HDLs), such as VHDL or Verilog are mature and standardized languages that support this model of computation. They are parallel languages with modular representations (structure decomposition) and explicit parallelism at behavioral level. This, combined with the event-driven characteristics that are used to reflect the behavior of the system at clock cycle level, make these models very precise though too low-level for simulating large systems or systems of systems.

They can be used as design-entry level specification for relatively small to medium size hardware modules. Also, with the availability of High Level Synthesis (HLS) tools and the profusion of back-end tools to produce RTL from higher abstraction levels or other MoCs, this language is being relegated as a requirement for HW fabrication, but not as a conventional entry point.

RTL synthesizers are tools that transform RTL into netlists of logic gates. They are mature, commercially available, in cases customer dependent tools that take an RTL specification as an entry point and produce a netlist or, even further, a bitstream to be downloaded into reconfigurable devices such as FPGAs.

MoC properties

This model is the best representative for HW targets. They are *modular* and *composable* (with hierarchical description of components) due to the capability of modelling structure, as well as *analyzable* because of their property for describing behavior or functionality. The event-driven specification at clock-cycle level makes it *predictable* and *deterministic* (except for some rarely used constructs that are not common for synthesis—oriented products). RTL-level in HDLs contains the synthesizable constructs, while HDLs at higher levels of abstraction (not time-specific) are not considered RTL

Relationship with other MoCs

HLS tools provide transformations from C/C++/System-C/OpenCL specifications as well as for several dataflow-oriented MoCs, provided the availability of back-end tools that transform these models into RTL.

MoC Usage

RTL is clearly targeted for HW fabrics. These fabrics are useful to accelerate performance while providing reasonable energy consumption when dealing with data intensive applications. In the context of heterogeneous computing, more control-intensive

tasks will be more likely to be used in SW, while compute intensive tasks will be favored towards HW implementations.

MoC Support

HDLs supporting RTL are very mature and so they account with basic tools such as synthesizers, simulators, physical mapping tools (i.e. the layout for an ASIC or the bitstream for an FPGA). They also account with accurate models for estimating power consumption, as well as some indirect reliability indicators such as code coverage tools, assertion-based verification, etc. In CERBERO, HW-oriented targets ARTICo^3 and MDC will rely on RTL and conventional synthesis to obtain a suitable bitstream, while JIT composition will rely on pre-synthesized blocks (represented as bitstreams) that, by composing them dynamically will produce a module with a required new functionality.

4.2. Synchronous Dataflow

MoC brief description

The Synchronous Dataflow [Lee 1987] MoC models an application as a directed graph of computational entities, called actors, that exchange data through a network of First-In First-Out queues (FIFOs). Each time an actor is executed, or fired, it consumes and produces a fixed quantum of data, called data token, on the FIFOs to which it is connected. An example of SDF graph is given in Figure 3 - Example of Synchronous Dataflow Graph.

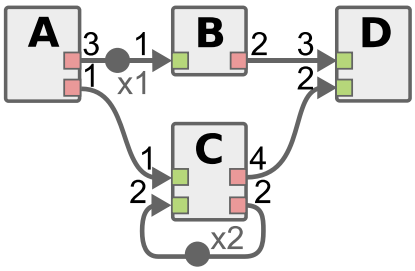

Figure 3 - Example of Synchronous Dataflow Graph

MoC properties

Synchronous Dataflow is a *parallel* and *decidable* MoC that exhibits one of the greatest degrees of *analyzability* among dataflow MoCs. Coupled with the *determinism* of the MoC, its *analyzability* makes it possible to prove algorithms deadlock freeness and boundedness at compile time and is often used to guarantee real-time properties (e.g. throughput, latency, worst-case execution time) of applications modeled with it. This great analyzability comes at the expense of a limited *expressiveness* of the MoC, because of the absence of any *reconfiguration* semantics in the MoC. The original MoC described in [Lee 1987] is *not modular*.

Relationship with other MoCs

The SDF MoC belongs to the family of dataflow models of computation. As one of the dataflow MoCs with the most restrictive semantics, SDF behavior can be expressed in most dataflow models.

As demonstrated in [Klikpo 2016], the MoC implemented in Labview® is equivalent to the SDF MoC.

There exist several dataflow MoCs with an equivalent expressiveness with the SDF MoC:

- The Cyclo-Static Dataflow [Bilsen 1996] and Affine Dataflow [Bouakaz 2012] MoCs which have a greater *conciseness* than the SDF MoC while retaining all its *analyzability*, by specifying sequences of production and consumption rates instead of scalar values.
- The Interface-Based SDF [Piat 2009] and Deterministic SDF with Shared FIFO [Tripakis 2013] MoCs which are two *modular* and *compositional* extensions of the SDF MoC.

MoC Usage

Synchronous Dataflow is mainly used to describe *stream-based* and *data-driven* algorithms, mostly at *function* and *component* levels. The SDF MoC is suitable for all kinds of implementations.

MoC Support

The SDF MoC is natively supported in the following tools: Ptolemy II [Davis 1999], SDF3 [Stuijk 2006], PREESM [Pelcat 2014], MDC [Palumbo 2017], LIDE [Shen 2011].

4.3. Parameterized and Interfaced Synchronous Dataflow

MoC brief description

The Parameterized and Interfaced Synchronous Dataflow (PiSDF) is the result of applying the Parameterized and Interfaced dataflow Meta-Modeling methodology [Desnos 2013] to the SDF MoC. PiSDF adds parameterization and interfaced hierarchy to the SDF MoC. The PiSDF MoC models an application as a directed graph. Besides actors and FIFOs (see section 4.1), parameters, hierarchical interfaces and parameter dependencies can also be vertices of the graph.

Parameters are employed to configure and modify dataflow specifications. Parameters can influence (1) the functionality of an actor, (2) the production/consumption rates of actor ports, (3) the value of another parameter and (4) a delay of a FIFO. Hierarchical interfaces convey data tokens or parameter values between levels of hierarchy. Hierarchical interfaced actors, or simply, hierarchical actors, are univocally linked to PiSDF subgraphs. Parameter dependencies propagate parameter values to other elements of the graph.

Actors, hierarchical or non-hierarchical, can have two types of ports: data ports and configuration ports. Data ports exchange data and configuration ports parameters. Parameters are connected to configuration ports through parameter dependencies. Both types of ports can be declared as input or output ports. An actor with an output configuration port is named a configuration actor. Firing of configuration actors dynamically produces values that set configurable parameters. The firing is only permitted at specific instants of time during a graph execution.

There are two types of parameters in a PiSDF MoC: configurable parameters and locally static parameters. Configurable parameters can be modified in each graph iteration, i.e. at run-time. Locally static parameters can only be modified at design-time. Parameter

values passed through input configuration interfaces of hierarchical actors always become locally static parameters of hierarchical (sub)graphs.

Output configuration ports are always connected to configurable parameters. A change in a configurable parameter is the result of a change in either an output configuration port of an actor or another configurable parameter the former depends upon.

MoC properties

PiSDF inherits the properties of SDF (see section 4.1) and adds the *modularity* and *reconfigurability* properties, with the advantage of keeping the *analyzability* of SDF. As the reconfiguration semantics is included into PiSDF, its *expressiveness* is greater than that of SDF. Besides modularity, reconfigurability is extremely handy in the context of cyber-physical systems, which is why in the CERBERO project we intend to use and extend PiSDF (see Section 5).

Relationship with other MoCs

The PiSDF MoC is related to the Interface-Based SDF [Piat 2009], from which it inherits the *compositional* hierarchy mechanism. The PiSDF MoC has the same *expressiveness*, but a better *conciseness*, as the Parameterized SDF MoC [Bhattacharya 2001]

MoC Usage

PiSDF is mainly used to describe stream-based, data-driven and control-driven algorithms (with a reduced number of configurable parameters in practice), mostly at functional and component levels. The PiSDF MoC is suitable for implementations in heterogeneous systems [Heulot 2014].

MoC Support

The SDF MoC is natively supported in the tool PREESM [Pelcat 2014], and the Spider runtime [Heulot 2014] is used to support the reconfiguration of graphs during execution. The tools MDC [Palumbo 2017] and ARTICo³ will support this MoC and integrate with PREESM and Spider. The objective is to offer new scheduling and mapping choices to the runtime manager when dealing with reconfigurable hardware, i.e. hardware and software implementations for an actor. The decisions will be driven by on-the-fly readings of performance indicators using the Performance API (PAPI).

4.4. Bulk Synchronous

MoC brief description

The Bulk Synchronous Parallel (BSP) MoC has been introduced by Valiant in [Valiant 1990]. This MoC is well suited to some types of highly parallel architectures such as GPU architectures, which makes it a very popular MoC. Figure 4 - Example of a Bulk Synchronous Model shows an example of an application representation using the BSP MoC.

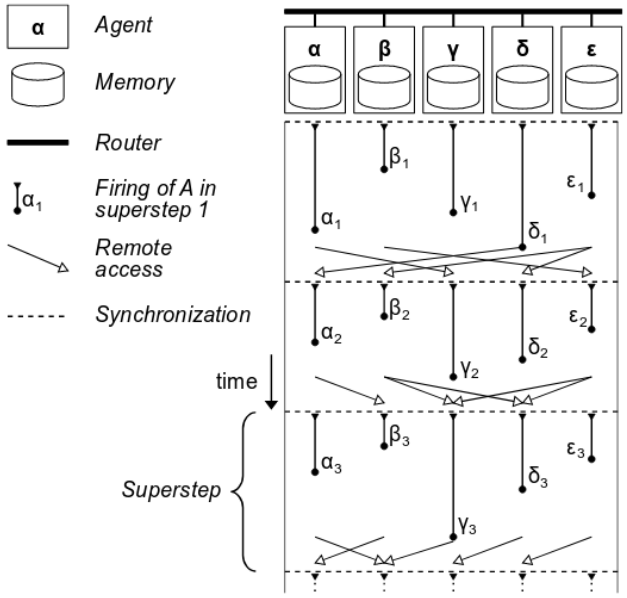

Figure 4 - Example of a Bulk Synchronous Model

BSP divides an application into several phases called supersteps. A BSP computation is composed of a set of components (we will call them agents). Each agent has its own memory. An agent can access the memory of another agent through a remote access (message) via a so-called router. The computation execution happens in a series of supersteps consisting of processing efforts, remote accesses and a global synchronization.

MoC properties

Bulk Synchronous Parallel is a *decidable* MoC which fosters execution *parallelism*. However, the processing of each core is modeled independently and statically divided into supersteps. Consequently, the *conciseness, expressiveness and reconfigurability* are limited. The size of supersteps offers a tradeoff between synchronization overhead and potential parallelism. BSP also provides a time performance evaluation for a superstep, giving the MoC some properties of a Model of Architecture (MoA) [Pelcat 2018].

Relationship with other MoCs

With respect to dataflow MoCs, including *modularity* and *compositionality* and well suited for application specification, BSP may be used as an intermediate representation for generating code for a parallel platform, limiting the backend complexity to a simpler support of a superstep at the cost of regular global synchronizations. As an example of a recent BSP study, Kapre et al. [Kapre 2017] discuss the pros and cons of using BSP versus SDF over OpenCL pipes on an FPGA.

MoC Usage

The BSP MoC can be used in PREESM or other SDF-based tools for both stream processing and batch processing. It needs a relatively large parallelism in the platform to be relevant.

MoC Support

- Bulk synchronous parallel ML [Loulergue 2005].

4.5. Kahn Process Networks

MoC brief description

Process Networks – also called Kahn Process Networks (KPN) after G. Kahn who first introduced them in his thesis [Kahn 1974] – is a MoC for describing signal processing systems where infinite streams of data are incrementally transformed by processes executing in sequence or parallel.

Process Networks are directed graphs where nodes represent computing processes and arcs are infinite message queues that connect these processes. Writing to a channel is non-blocking but reading is blocking. It cannot be waiting for data on one or another input channel.

It was proposed for modeling distributed systems but has proven its convenience for modeling signal processing systems as well. As pointed out by Edward Lee in [Lee 1995], this MoC does not require multitasking or parallelism and usually neither infinite queues; it is in fact usually more efficient than comparable methods in functional languages.

MoC properties

Processes in a KPN produce data elements that are placed in a communication channel and consumed by the destination process. Communication channels are the only way processes may exchange information. KPN systems are *deterministic* because the history of tokens produced/consumed does not depend on execution order. As discussed by [Parks 1995], it is not possible to tell in a finite time whether an arbitrary Process Network will halt in its streaming of data. Such behavior is related to two properties: *termination* and *boundness*. These properties are *undecidable* in finite time for the general case but, under some restrictions, we can study and classify PN before execution. Also, they are *compositional*.

Relationship with other MoCs

KPNs are a generalization of the Dataflow models described in section 4.1.

MoC Usage

Process Networks have found many applications in modeling embedded systems as it is typical for embedded systems to be designed to operate infinitely with limited resources.

MoC Support

Commercial systems like SPW from Alta Group of Cadence, COSSAP from Synopsys, the DSP Station from Mentor Graphics, Hypersignal from Hyperception or Simulink by Mathworks and research software tools like Khoros from the University of New Mexico and Ptolemy from the Univ. of California at Berkeley, are all based on variants of the PN model. Departing from the original Process Networks by Kahn, several more specific models have been derived.

In CERBERO, KPNs are the underlying semantics of the communication between tasks in the DynAA simulation tool of TNO.

4.6. Dataflow Process Network

MoC brief description

The Dataflow Process Network (DPN) [Lee 1995], also known as Dynamic Dataflow Model (DDF) is a MoC where data processing nodes, named actors, communicate through unidirectional unbounded FIFO channels. Actors are provided with a set of firing rules specifying the amount of data (tokens) required on the input channels to trigger the processing (fire). The firing of an actor consumes tokens from the input channels and produces tokens to the output ones. Figure 5 - Example of Dataflow Process Network Graph depicts an example of a DPN graph.

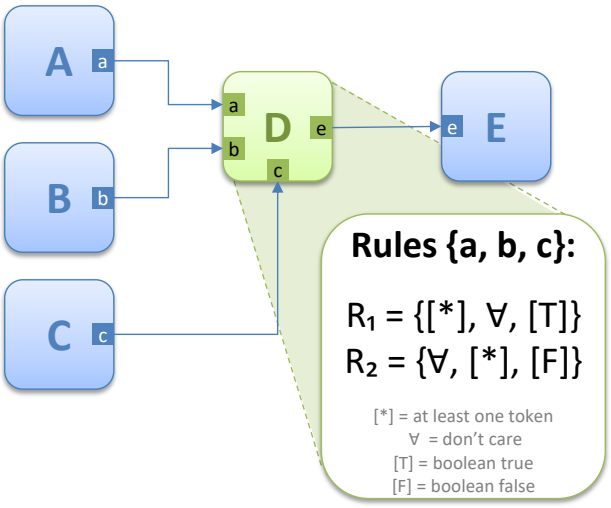

Figure 5 - Example of Dataflow Process Network Graph

MoC properties

DPN is the most *expressive* dataflow MoCs: it is Turing-complete, meaning that it can describe any *deterministic* or *non-deterministic* algorithm. This high degree of *expressiveness* comes at the price of *analyzability*, since depending on the specific case, a DPN could be very hard to analyze (e.g. for graphs modeling non-deterministic algorithms). Due to its *non-deterministic* nature, the DPN MoC exhibits also *non-decidability* and a restricted *parallelism* with respect to less expressive MoCs (such as SDF).

Relationship with other MoCs

Being the most expressive MoC among dataflow ones, a DPN can describe all other more restrictive dataflow MoCs, such as:

- SDF and PiSDF, obtained by limiting firing rules to one per actor and to fix its token rates;
- Kahn Process Network (KPN) by removing *non-determinism* behavior: action firings must be *deterministic* (output tokens depend only on input tokens without side effects) and the set of firing rules for each actor has to be *sequential* (they can be tested in a pre-defined order using only blocking reads [Lee 1995]).

DPNs can also be translated or expressed by means of other MoCs with the same or an enhanced expressiveness, such as a generalized PNs [Dimitrov 2017].

MoC Usage

DPNs are usually adopted for streaming applications with intensive computation, task parallelism and data locality, such as audio and video coding. The DPN MoC can describe any kind of application [Lee 1995].

MoC Support

The DPN MoC is supported by several frameworks and tools: Orcc [Yviquel 2013], CAPH [Serot 2014], MDC [Palumbo 2017], LIDE [Shen 2011].

4.7. Petri Networks

MoC brief description

Petri Nets (PNs) are one of the most important families of discrete event modeling formalisms. It was firstly introduced in the early 1960s by Carl Adam Petri as a bipartite weighted directed graph with two types of vertices called places (represented by circles) and transitions (represented by bars or rectangles). The 'execution' of a Petri Net can be seen as a game whose rules regulate the activation of transitions and transfer of information tokens between places. We refer to [Giua 2007] for a comprehensive system theory point of view on Petri Nets.

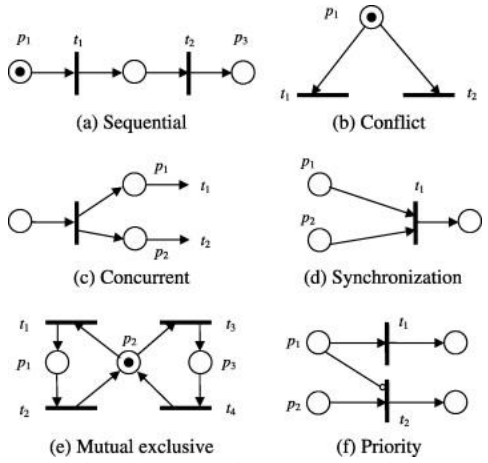

Figure 6 - Petri Nets semantics.

MoC properties

Petri nets are both a *graphical* and *mathematical* formalism, which provide a useful visual tool both in the design and analysis phase. They build on a *concise* representation of systems with a very large state space. Indeed, they do not require representing explicitly all states of a dynamical system but only an initial one – the rest of the state space can be determined from the rules that govern the net evolution.

Petri nets are *modular* and *parallel;* i.e., if a system is composed of several subsystems that interact among them, it is possible to represent each subsystem with a simple subnet and then combine the subnets to obtain a model of the whole system.

The execution of Petri nets is *non-deterministic*. If multiple transitions are enabled at the same time in a PN model, any one of them can fire. Also, it is not guaranteed that an enabled transition fires. An enabled transition can fire immediately or after any amount of

time (provided it remains enabled), or not fire at all. A comprehensive overview on the properties of PNs can be found in [Seatzu 2013].

Relationship with other MoCs

Many other models of computation may be derived from extending the rules of Petri Networks. For systems theory and the design of CPSs, the most important of these siblings are Hybrid Petri Nets, which represent both time- and event-driven components [Alla 1998]. Coloured Petri Nets [Kurt 1996] allows distinctions between the tokens and attribution of more sophisticated semantic rules to the firing of the transitions. Instead of extending the Petri net formalism, we can also look at restricting it. That yields another plethora of modeling formalisms that are important as well: state machines, marked graphs, and (extended) free choice networks**.** Each of which of these sub-classes modifies the basic properties of this MoC, for example, state machines *are not parallel*, and marked graphs are *parallel* and *deterministic.*

MoC Usage

Petri nets have been specifically designed to model systems with interacting components and as such are able to capture many characteristics of an event driven system, namely concurrency, asynchronous operations, deadlocks, conflicts, etc., [Cassandras 2008]. While they are not specifically designed for self-adaptive systems, Petri Nets can model self-adaptation on the system level.

MoC Support

PNs is the MoC rendered in the simulation engine of DynAA [Oliveira 2013], used in the CERBERO project. The University of Hamburg [Hamburg 2018] maintains a large database on academic and commercial tools that use PNs as a base.

4.8. Discrete Event System

MoC brief description

Discrete Event Systems (DES) is a model of computation mainly used for modelling and simulation [Zeig 2000]. The discrete event formalization provides the basis for orchestrating the occurrence of events in time during the simulation of a (cyber) system. The principles of a DES MoC are:

- The world is made of objects (things) and events;
- Only objects can generate events and only objects react to events.
- The state of the system can only be manipulated (modified) by objects upon the occurrence of an event, thus only in discrete points of time when an event occurs.

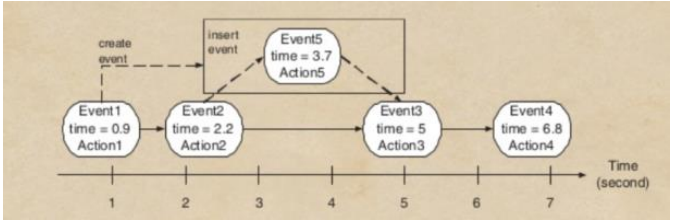

Figure 7 - Example of Discrete Event System

In discrete-event models, components communicate via signals consisting of events placed on a time line. Events are processed in a chronological order.

MoC properties

Discrete event models are very practical to model and understand and have a huge (practical) application range. Nevertheless, most of the properties for such models of computation are not guaranteed in a broad aspect and fail to exist in extreme cases. For example, discrete events are not compositional, because some discrete event sources may eventually block future events in the timeline to occur – for example, sources that lead to a *Zeno* condition.

Discrete event models often have to use a technique called *superdense* time that allows to model sequence of causally-related events (events that causes another events) to exist at the same time tag (instantaneous). This can lead to difficulties in proving or defining properties in the sense discussed earlier in this document. For example, the existence of real *concurrency and determinism* properties is dependent on how simultaneous events are chosen to fire. Choices that force a strict order become deterministic, but not concurrent (e.g. events concur on the same resource). On the other hand, if choices are random, the system becomes concurrent, but not deterministic.

Imposing further restrictions to the way events are triggered, combined, and handled yields other MoCs– see for example PNs.

Relationship with other MoCs

Everything modelled by computer can be represented by DES. Consequentially, other MoCs discussed in this document can be related to DES. PNs, Process Networks, SDF, etc., and all others herein can be described using discrete event theories. In a sense, the discrete event MoC focuses specially in attributing rules for the execution of the simulators, trying to establish a base for how race conditions are to be solved.

MoC Usage

This MoC is well suited for modeling digital circuits, communication networks, business processes, queue systems, etc.

MoC Support

There is a huge number of simulators based on discrete events. The most known ones are SimEvents [Mathworks], GoldSim [GoldSim], AnyLogic[AnyLogic], PtolemyII[Ptolemy].

In CERBERO, the tool DynAA developed by TNO is a modelling and analysis environment tightly coupled with a discrete event engine.

4.9. Situated Cognitive Engineering

MoC brief description

Situated Cognitive Engineering (sCE) is a knowledge-based multi-agent MoC developed to establish the required normative (policy-based) system behaviors concerning both planned (procedural) work and anomaly detection, re-planning and recovery processes. Using formal relations established between knowledge at distinct levels of abstraction,

reasoning rules are introduced that enable the system to navigate through these levels, providing the capability to inference high-level failures based on anomalous component status. Applications based on sCE are software implementations that can be easily distributed to assess CPSoS scenarios, typically in the form of an *ePartner* (see **Error! Reference source not found.**).

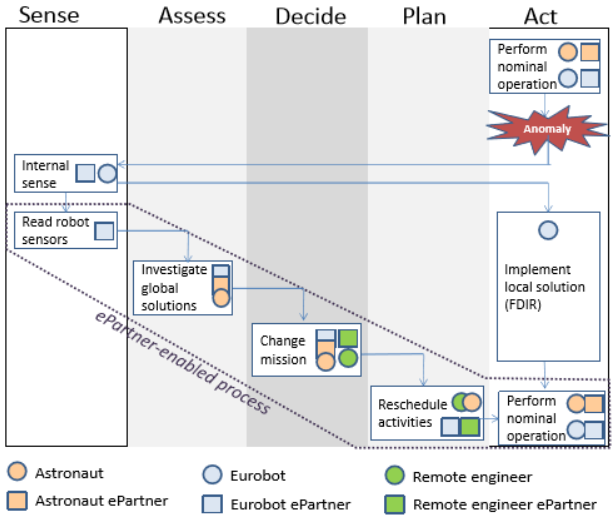

Figure 8 - sCE monitoring and decision process

MoC properties

sCE [Bosse 2017] is a *concise* way to represent the safety constraints that a system should exhibit during execution. Situated cognitive engineering provides a *deterministic* and *expressive* way to define the conditions to check and the required actions to return to a nominal state when unforeseen events occur. Each condition could be considered as a *module* that can be executed in *parallel* and interacts with the rest of the conditions by sharing information through a database. As well, each module can interact with external systems to provide *reconfigurability* and *modularity* (in an *event-driven* schema), at component system and system-of-systems levels by exploiting decision-making processes.

Relationship with other MoC

N/A

MoC Usage

sCE is used to define safety constraints that shall hold during the execution of a system, providing support to monitoring, diagnosis and decision-making to overcome anomalous states of the controlled system. It is possible to deploy reasoning rules from the component level to system-of-systems in distributed applications.

MoC Support

sCE is supported in the MECA-HEART tool [MECA 2015].

4.10. Transition System

MoC brief description

Transition systems are used as models to describe the behavior of systems. They are usually represented as directed graphs where nodes represent states (that describe some information about a system at a certain moment of its behavior), and edges model *transitions* (state changes). In particular, Kripke structures (see, e.g., [Clarke 1999]) are traditionally used for the interpretation of temporal logics such as the Linear Temporal Logic (LTL) in automata-based LTL model checking [Vardi 1986]. This approach is based on the fact that each LTL formula (representing a property to be checked) can be represented by a non-deterministic Buchi automaton and check the if such formula holds in the transition system.

MoC properties

The product transition systems obtained for automaton-based LTL model checking is a *decidable* (for finite systems) and *deterministic* MoC.

Relationship with other MoC

N/A

MoC Usage

This MoC is mainly used for control-driven and event-driven representations at the component and/or system level.

MoC Support

The most noticeable tool implementing this MoC is the model checker SPIN [Holzmann 1997] and it will be used within the CERBERO framework by the Verification Tool (VT).

4.11. Summary

Following is a table that summarizes the MoCs and their properties.

MoCs	Analyzability	Conciseness	Compositionality	Decidability	Determinism	Expressiveness	Modular	Parallelism	Predictability	Reconfigurability
RTL	+		++		+		++	++	+	
SDF	++	+		+	+	-	--	+		--
Cyclo-Static	++	++								
Interface-Based SDF	++	+	+							
Deterministic SDFwith Shared FIFO	++	+								
PiSDF	++	++	+	+	+	++	+	+		+
BSP		-	+	+		-	+	++		-
KPN			+	-	+			+		
DPN	-			-	+	++		-		
PN	+	+			-	++	+	+		+
DES	-		++	-		+++	++	++	-	+
SCE					+	+	+	+		+
TS				+	+					

Table 4-1: ***SDF***: Synchronous Dataflow; ***PiSDF***: Parameterized and Interfaced Synchronous Dataflow; ***BSP***: Bulk Synchronous Parallel; ***PN***: Petri Networks; ***DPN***: Dataflow Process Network; ***RTL***: Register Transfer Level; ***TS***: Transition System; ***KPN***: Kahn Process Networks; ***DES***: Discrete Event System; ***sCE***: Situated Cognitive Engineering**.** Non-bold MoCs are those inherited from their parent (first bold one above). Blank cells indicate a MoC has no specific traits for the property.

5. CERBERO Innovation on Models of Computation for CPS

Given the characteristic properties of the MoCs presented in Section 3 and the list of them surveyed in Section 4, this section presents the envisioned contributions of the CERBERO project to the model of computations domain. The main motivation behind these contributions is to support the specification of key aspects of CPSs.

5.1. Dataflow Extension for Persistent State Representation

Main contributors: INSA, UPM

Motivations

In synchronous dataflow MoCs, as for example in the SDF, PiSDF models presented in Section 4, the semantics is dedicated to the processing of infinite streams of data. To this purpose, the semantics of these dataflow MoCs has been tailored to capture in a concise form the data-parallelism and determinism of algorithms executed infinitely repeatedly, with numerous and entangled data dependencies.

Despite the many advantages of the semantics of SDF models, these cannot currently be used to represent concisely and unambiguously the persistence or the sporadic initialization of data within algorithms. In CPSs, where computing systems must continuously adapt their behavior to the physical environment enclosing them, these persistent data are needed to capture the adaptive state of algorithms, which may be sporadically updated to fit them to an evolution of their working environment. An example of such persistent data is the coefficients encoding the learning ability of the neurons in on-line machine learning algorithms.

Envisioned Contribution

Within CERBERO, an extension of the SDF models semantics is envisioned to support the specification of both persistent and sporadically updated data in described algorithms. This contribution will extend the existing semantics of dataflow by building on an existing element called *delay*. While improving the expressiveness and conciseness of the MoCs, this extended semantics will preserve the analyzability, the compositionality, and the data parallelism of the synchronous dataflow models.

Use in CERBERO

This extension of the dataflow model is suitable for the modelling of any CPS system. With an implementation of this contribution within PREESM & Spider, its use within CERBERO will be demonstrated with an implementation of a reinforcement learning algorithm that is applicable to any control system, such as the robotic arm of the Space Exploration Use-Case.

5.2. Non-Functional Properties Modelling in Dataflow

Main contributor: INSA, UPM

Motivations

Synchronous dataflow MoCs focus on the functional modelling of applications. To this purpose, the semantics of these dataflow models is dedicated to the representation of untimed computational and data transfer aspects of algorithms, independently from any implementation consideration. Although this implementation-independence principle grants the dataflow MoCs a great portability, it prevents them from being used for specifying non-functional properties of modeled algorithms, such as real-time properties that are critical for the design of constrained CPSs.

In state-of-the-art work on using synchronous dataflow MoCs for the specification of real-time systems, this issue is generally alleviated by breaking the architecture-agnostic principle of the model, as for example by associating an architecture-specific worst-case execution time to actors of a dataflow graph. Analyses based on these execution times are then used to predict the execution time of the application for the specific target.

Envisioned Contribution

An extension of the semantics of synchronous dataflow MoCs will be proposed within CERBERO to support the specification of non-functional properties in dataflow graphs. Unlike state-of-the-art work, the proposed semantics will be used to specify non-functional property as a design constraint of the algorithm, independently from the targeted architecture, thus retaining the portability of applications specified with the MoC. Consequently, it will be the responsibility of the design space exploration (DSE) tools and algorithms to simulate or profile the different components of the application, at compile-time and at runtime, and make sure that the specified constraints hold for the targeted system. Envisioned non-functional properties supported by the new semantics are real-time properties (throughput, latency, response time to events), but also other key performance indicators (KPIs) specified in D3.4, such as power and energy consumption.

Use in CERBERO

This extension of the dataflow MoC semantics will be implemented within the PREESM and Spider tools and connected to the KPI specification within the CERBERO Intermediate Format presented in D3.6. Papify implementation with MDC & ARTICo^3 will enable the application profiling and energy monitoring on heterogeneous platforms, thus providing a good support to enforce non-functional properties captured in the extended dataflow model.

5.3. Moldable Parameters in Dataflow for Extended Design-Space Exploration

Main contributors: INSA, IBM

Motivations

The DSE phase based on SDF MoCs mostly consists of mapping parallel actor executions on the heterogeneous computational hardware resources of the targeted

architecture, and the data transfers on the hardware means of storage and communication. In those cases where the number of parallel actors to map largely exceeds the available resources of the architecture, DSE optimization algorithms face an important increase in the complexity of the mapping problem. In such cases, developers will often manually update the model of their applications to adopt a coarser granularity of description. This coarser granularity translates into less numerous but 'larger' actors to execute. As illustrated in [Hascoet 2017], by carefully adjusting the granularity of the application description, enough elements will be exposed to permit a fair distribution of work on the available hardware resources, with a reasonable complexity exposed to the DSE algorithms.

Envisioned Contribution

The adaptation of the granularity of the application exposed to the DSE algorithm is generally left to the designer of the application. The objective of this contribution is to extend the semantics of SDF models to support the specification of so-called *moldable parameters*. A moldable parameter is a parameter associated to a range of acceptable values, thus leaving the responsibility to the DSE algorithm to select the most appropriate one in its optimization process. In general, moldable parameters are supposed to change only the 'organization' (e.g. like the exposed degree of parallelism) of computations, but not the output they produce. Hence, by specifying moldable parameters in SDFgraphs, it will be possible for the designer to let the DSE algorithms automatically control the parallelism and granularity of the application to obtain the best DSE solution in minimum time.

Use in CERBERO

The moldable parameters will be integrated within PREESM during the CERBERO Project. It is envisioned that DSE optimizations based on this extended semantics will be provided through a connection to AOW.

5.4. Extension of PiSDF MoCs through polyhedral transformations

Main contributors: UPM

Motivations

Polyhedral transformations are a solid set of well-known techniques to extract parallelism from nested loops. Despite the extensive literature available and research conducted, the constraints imposed on the code to be analyzed, make the application of polyhedral transformations difficult to be generalized. Briefly speaking, these limitations are mainly related to dynamic behaviors such as those expressed with PiSDF MoC. Currently, PiSDF related tools like PREESM consider each actor as a black box, so its contents are transparent to the tool. As a result, several parallelization opportunities could be missed, since it is up to the designer to explicitly consider this parallelization in the code describing the actor.

Envisioned Contribution

The main objective with this extension is to expand the polyhedral model to support the extraction of intra-actor available parallelism at run-time. This extraction will depend on model parameter changes and can be considered as an adaptation process.

Use in CERBERO

This extension will be first applied in PREESM, which is the PiSDF MoC used within the CERBERO project.

6. Conclusions

In this document we provided an overview on the state of the art (Section 4) and innovative activities (Section 5) that we are doing in CERBERO in terms of Model of Computations (MoCs). Table **4-1** provided a summary of the properties that the analyzed MoCs offer to the designer.

Since CERBERO is following a model-based approach and models are key ingredients to solve interoperability issues (as it will be deeply discussed in D3.6 that describe in detail CERBERO modeling activities), we provide in Table 6-1 a summary MoCs behind the tools of the CERBERO framework (where possible, for example the AOW tool does not refer to a specific MoC for computational system components). Please note that we indicate with (S) the already adopted/supported MoCs and with (P) models that are going to be supported by the end of the CERBERO project. In D5.6 (the deliverable that presents the tools composing the CERBERO framework) this mapping will be enriched to make clear which tools, and indirectly which models, are used in the different use cases.

Table 6-1 – CERBERO Tools to MoC Mapping. S: support, P: planned support within CERBERO duration

	SDF	**PiSDF**	**PN**	**KPN**	**DPN**	**RTL**	**DES**	**SCE**	**TS**
MECA								S	
VT									S
DynAA			S	S			S		
AOW									
PREESM	S	S							
SPIDER		S							
PAPIFY		P			S				
JIT HW						S			
ARTICo³		P				S			
MDC	S	P			S	S			

7. References

[Alla 1998] Alla, H., & David, R. (1998). Continuous and hybrid Petri nets. Journal of Circuits, Systems, and Computers, 8(01), 159-188.

[AnyLogic] Online, https://www.anylogic.com/company/timeline/

[Bhattacharyya 2006] S.S. Bhattacharyya and W.S. Levine. *Optimization of signal processing software for control system implementation.* In Computer Aided Control System Design, 2006 IEEE International Conference on Control Applications, 2006 IEEE International Symposium on Intelligent Control, 2006 IEEE, pages 1562-1567. IEEE, 2006

[Bosse 2017] T. Bosse, L. Breebaart, J. Van Diggelen, M.A. Neerincx, J. Rosa and N.J. Smets Developing ePartners for human-robot teams in space based on ontologies and formal abstraction hierarchies, Int. J. Agent-Oriented Software Engineering, Vol. 5, No. 4, 2017.

[Cassandras 2008] C. G. Cassandras and S. Lafortune. "Introduction to discrete event systems". Springer, 2008.

[CERBERO 2017] http://www.cerbero-h2020.eu

[Clarke 1999] Clarke, E. M., Grumberg, O., & Peled, D. (1999). *Model checking*. MIT press.

[Bilsen 1996] Bilsen, G., Engels, M., Lauwereins, R., & Peperstraete, J. (1996). *Cyclo-static dataflow*. *IEEE Transactions on signal processing*, *44*(2), 397-408.

[Bouakaz 2012] Bouakaz, A., Talpin, J. P., & Vitek, J. (2012, June). Affine data-flow graphs for the synthesis of hard real-time applications. In *Application of Concurrency to System Design (ACSD), 2012 12th International Conference on* (pp. 183-192). IEEE.

[Davis 1999] Davis II, J., Goel, M., Hylands, C., Kienhuis, B., Lee, E. A., Liu, J., ... & Smyth, N. (1999). *Overview of the Ptolemy project* (Vol. 99). ERL Technical Report UCB/ERL.

[Desnos 2013] K. Desnos, M. Pelcat, J. F. Nezan, S. S. Bhattacharyya and S. Aridhi, "PiMM: Parameterized and Interfaced dataflow Meta-Model for MPSoCs runtime reconfiguration," *2013 International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS)*, Agios Konstantinos, 2013, pp. 41-48.

[Dimitrov 2017] D.G. Dimitrov, Generalized Net Representation of Dataflow Process Networks. In Recent Contributions in Intelligent Systems, pp. 23-31, 2017.

[Goldsim] Online, https://www.goldsim.com/Web/Home/

[Giua 2007] Giua, Alessandro and Seatzu, Carla. "A systems theory view of petri nets". Advances in Control Theory and Applications, pages 99-127, 2007.

[Hamburg 2018] Universität Hamburg, *Petri Nets Tools Database, online.* *https://www.informatik.uni-hamburg.de/TGI/PetriNets/tools/quick.html*.

[Heulot 2014] Heulot, J., Pelcat, M., Desnos, K., Nezan, J. F., & Aridhi, S. (2014, September). Spider: A synchronous parameterized and interfaced dataflow-based rtos for multicore dsps. In *Education and Research Conference (EDERC), 2014 6th European Embedded Design in* (pp. 167-171). IEEE.

[Holzmann 1997] Holzmann, G. J. (1997). The model checker SPIN. *IEEE Transactions on software engineering*, *23*(5), 279-295.

[Kahn 1974] Kahn, G. (1974). The semantics of a simple language for parallel programming. Information Processing, pages 471-475.

[Kapre 2017] Kapre, N., & Patel, H. (2017, May). Applying Models of Computation to OpenCL Pipes for FPGA Computing. In Proceedings of the 5th International Workshop on OpenCL (p. 9). ACM.

[Kurt 1996] Jensen, Kurt (1996). Coloured Petri Nets (2 ed.). Berlin: Heidelberg. p. 234.

[Klikpo 2016] Klikpo, E. C., Khatib, J., & Munier-Kordon, A. (2016, April). Modeling multi-periodic simulink systems by synchronous dataflow graphs. In *Real-Time and Embedded Technology and Applications Symposium (RTAS), 2016 IEEE* (pp. 1-10). IEEE

[Lee 1987] E.A. Lee and D.G. Messerschmitt. *Synchronous dataflow*. Proceedings of the IEEE, 75(9):1235-1245, sept. 1987.

[Lee 1995] Lee, E. and Park, T. (1995). Dataflow Process Networks. In Proceedings of the IEEE, volume 83, pages 773-799.

[Lee 2017] Edward A. Lee and Sanjit A. Seshia, *"Introduction to Embedded Systems, A Cyber-Physical Systems Approach"*, Second Edition, MIT Press, ISBN 978-0-262-53381-2, 2017.

[Loulergue 2005] Loulergue, F., Gava, F., & Billiet, D. (2005, May). Bulk synchronous parallel ML: modular implementation and performance prediction. In International Conference on Computational Science (pp. 1046-1054). Springer, Berlin, Heidelberg.

[Mathworks] Online, www.mathworks.com

[MECA 2015] The MECA Consortium, MECA-HEART Design Document, Deliverable WP2-D3a, MECA-HEART WP2-D3a – Design Document, 20 Feb 2015.

[Oliveira 2013] J. Oliveira et al. , Model-based design of self-adapting networked signal processing systems, International Conference on Self-Adaptive and Self-Organizing systems, 2013

[Ostroff 1995] J.S. Ostroff. *Abstraction and composition of discrete real-time systems*. Proc. of CASE, pp 370-380, 1995.

[Palumbo 2017] F. Palumbo, C. Sau, T. Fanni, P. Meloni and L. Raffo, SS-design: Dataflow-based design of coarse-grained: Reconfigurable platforms reconfigurable platform composer tool project. In proceedings of the IEEE International Workshop on Signal Processing Systems, 2016.

[Parks 1995] Parks, T. M. (1995). *Bounded Schedule of Process Networks*. PhD thesis, University of California at Berkeley.

[Pelcat 2014] Pelcat, M., Desnos, K., Heulot, J., Guy, C., Nezan, J. F., & Aridhi, S. (2014, September). *Preesm: A dataflow-based rapid prototyping framework for simplifying multicore dsp programming.* In Education and Research Conference (EDERC), 2014 6th European Embedded Design in (pp. 36-40). IEEE.

[Pelcat 2018] Pelcat, M., Models of Architecture for DSP Systems. Springer. *Handbook of Signal Processing Systems*, Third Edition, In press. 〈hal-01660620〉

[Piat 2009] Piat, J., Bhattacharyya, S. S., & Raulet, M. (2009, October). Interface-based hierarchy for synchronous data-flow graphs. In *Signal Processing Systems, 2009. SiPS 2009. IEEE Workshop on* (pp. 145-150). IEEE.

[PtolemyII] Online, http://ptolemy.eecs.berkeley.edu/ptolemyII/index.htm

[Savage 1998] Savage, J.E. *Models of Computation*, Volume 136, Addison-Wesley Readings,

MA, 1998

[Seatzu 2013] Seatzu, Carla and Silva, Manuel and van Schuppen, Jan H, "Control of discrete-event systems: automata and petri net perspectives" Lecture Notes in Control and Information Sciences, Vol. 433, Springer, 2013.

[Serot 2014] J. Serot & F. Berry, High-Level Dataflow Programming for Reconfigurable Computing. In Proceedings of the IEEE 26th International Symposium on Computer Architecture and High Performance Computing Workshops, 2014.

[Shen 2011] Shen, C. C., Wang, L. H., Cho, I., Kim, S., Won, S., Plishker, W., & Bhattacharyya, S. S. (2011). *The DSPCAD lightweight dataflow environment: Introduction to LIDE version 0.1*.

[Stuijk 2006] Stuijk, S., Geilen, M., & Basten, T. (2006, June). Sdf^ 3: Sdf for free. In *Application of Concurrency to System Design, 2006. ACSD 2006. Sixth International Conference on* (pp. 276-278). IEEE.

[Tripakis 2013] Tripakis, S., Bui, D., Geilen, M., Rodiers, B., & Lee, E. A. (2013). Compositionality in synchronous data flow: Modular code generation from hierarchical sdf graphs. *ACM Transactions on Embedded Computing Systems (TECS)*, *12*(3), 83.

[Valiant 1990] Leslie G Valiant, "A bridging model for parallel computation,", Communications of the ACM, vol. 33, no. 8, pp. 103–111, 1990

[Vardi 1986] Vardi, M. Y., & Wolper, P. (1986). An automata-theoretic approach to automatic program verification. In *Proceedings of the First Symposium on Logic in Computer Science* (pp. 322-331). IEEE Computer Society.

[Yviquel 2013] H. Yviquel, A. Lorence, K. Jerbi, A. Sanchez, G. Cocherel, and M. Raulet, Orcc: Multimedia development made easy. In Proceedings of the 21st ACM international conference on Multimedia, 2013.

[Zeig 2000] B. P. Zeigler, H. Praehofer, T. G. Kim. Theory of Modeling and Simulation, Second Edition. Academic Press, 2000.