Proceedings of the 2013 Federated Conference on
Computer Science and Information Systems pp. 1415–1422

Towards the Applicability of Alf to Model Cyber-Physical Systems

Alessandro Gerlinger Romero
Brazilian National Institute for Space Research, Avenida dos Astrounautas, 1758, 12227-010, São José dos Campos, São Paulo, Brazil.
Email: romgerale@yahoo.com.br

Klaus Schneider
University of Kaiserslautern Computer Science Department, Po box 3049, 67653, Kaiserslautern, Germany.
Email: klaus.schneider@cs.uni-kl.de

Maurício Gonçalves Vieira Ferreira
Brazilian National Institute for Space Research, Avenida dos Astrounautas, 1758, 12227-010, São José dos Campos, São Paulo, Brazil.
Email: mauricio@ccs.inpe.br

Abstract— Systems engineers use SysML as a vendor-independent language to model Cyber-Physical Systems. However, SysML does not provide an executable form to define behavior but this is needed to detect critical issues as soon as possible. Action Language for Foundational UML (Alf) integrated with SysML can offer some degree of precision. In this paper, we present an Alf specialization that introduces the synchronous-reactive model of computation to SysML, through definition of not explicitly constrained semantics: timing, concurrency, and inter-object communication. The proposed specialization is well-suited for safety-critical systems because it is deterministic. We study one example already modeled in the literature, to compare these approaches with our one. The initial results show that the proposed specialization helps to couple complexity, provides better composition, and enables deterministic behavior definition.

I. Introduction

CYBER-Physical Systems (CPSs) are an integration of computational and physical processes [14]. The difficulty in modeling cyber-physical systems comes from the diversity of these systems. The most promising approach to mitigate this problem is developing expressive and precise modeling languages [8].

Accordingly, the Object Management Group (OMG) and the International Council on Systems Engineering (INCOSE) developed Systems Modeling Language (SysML) [20]; a general-purpose modeling language for systems engineering applications. SysML has demonstrated a capability for top-down design refinement for large-scale systems [11]; therefore, SysML is expressive, but the lack of formal foundations in the SysML results in imprecise models.

A major current focus in systems engineering is how to introduce precision in the approaches based on SysML through formal methods. This introduction can be a legal requirement when dealing with safety-critical systems; e.g., the IEC 61508 (Functional Safety of Electrical/Electronic/Programmable Electronic Safety-related Systems) defines formal methods as highly recommended technique for the highest safety integrity level; moreover, DO-178C (Software Considerations in Airborne Systems and Equipment Certification) addresses formal methods as a complement to testing. There are languages with a formal semantics such as Esterel [5] or the B-language [7]; nonetheless, there are no modeling languages with widespread use in systems engineering community that have the attraction of SysML [10].

This paper focuses on the evaluation of a formal foundation in SysML engineering approaches concerning behavioural definitions. Behavior is defined using SysML, and also using Unified Modeling Language (UML) [18], mainly by Activity Diagrams, Sequence Diagrams, and State Machine Diagrams, which do not have precise semantics given by OMG; and, in general, are not executable.

Behavioural definition could evolve with the Semantics of a Foundational Subset for Executable UML Models (fUML) [19]; this specification defines a formal semantics for an executable subset of UML. Moreover, OMG Action Language for Foundational UML (Alf) is the textual language for fUML [21].

On the contrary, there are research papers [4][22] stating that fUML and Alf are not suitable for behavioural modeling the safety-critical systems yet. The reasons can be classified as follows: (1) nondeterminism in the execution model [4]; and, (2) current tools do not allow the use of model-checking or theorem proving [22]. Hereafter, we will explore the reason (1) in detail.

fUML standard execution model is based on a model of computation (MoC), which is nondeterministic (we consider this in Section III.A). On the other hand, there is one MoC that can provide determinism, and can simplify the modeling and verification tasks; it is called synchronous-reactive [14].

The synchronous-reactive MoC can provide determinism using the fundamental model of time as a sequence of discrete instants and parallel composition as a conjunction of behaviors [3]. This MoC has been established as a technology of choice for specifying, modeling, and verifying real-time embedded applications [3], e.g. Esterel [5], Lustre (as well as, Lustre-based commercial Scade tool) [3], Signal [3], and Quartz [26] are languages also based on this MoC.

This work was supported by the Brazilian Coordination for Enhancement of Higher Education Personnel (CAPES) and German Academic Exchange Service (DAAD).

The synchronous-reactive MoC means that most of the statements are executed in zero time (at least in the idealized model). Synchronous computations consist of a possibly infinite sequence of atomic reactions that are triggered by a global logical clock. In each reaction, all inputs are read and all outputs are computed by all components in parallel. In the synchronous-reactive MoC, the communication and computation of values is done in zero time. Consumption of time must be explicitly defined with special statements, as e.g. the pause statement in Esterel [5] and Quartz [26].

Comparing a system described in the synchronous-reactive MoC against a system described following an asynchronous MoC for dual redundant flight guidance system, Miller et. al. [15] made the following observation: "the properties themselves are more difficult to state, were weaker than could be achieved in the synchronous case, and required considerable complexity to be added to the model to ensure that even the weakened properties were true". Furthermore, systems described by a synchronous-reactive MoC can be desynchronized [3] in a concrete solution that is then asynchronous, e.g. to generate Globally Asynchronous Locally Synchronous architectures (GALS) [15].

In this paper, we explore the causes of nondeterminism in fUML and Alf, and, present a deterministic specialization of Alf for CPSs modeling based on the synchronous-reactive MoC. This specialization removes deficiencies found by [2] [4] in fUML and Alf, and can be an alternative to define deterministic behaviors in SysML. The initial results show that the proposed specialization does not add complexity to the task of modeling CPSs using SysML, and enables a deterministic definition of the behavior.

The remainder of this paper is organized as follows: in Section II, related works are explored; in Section III, the relationships between Alf and other OMG specifications are explored; in Section IV, we present the initial approach; in Section V, we discuss the initial approach; finally, conclusions are shared in the last section.

II. Related Works

There is a large number of research papers about semantics for models defined using UML, and consequently, SysML. Hußmann [12] proposed the following classification for approaches concerning structural semantics: (1) naive set-theory, (2) meta-modeling, and (3) translation. This classification can also be used for the works focused on behavioural semantics.

Extending naive set-theory, Graves and Bijan [11] proposed one approach where behavior defined using SysML State Machine Diagrams is represented as a set of axioms in type theory. Graves [10] stated that SysML uses diagrams to model structure, and these diagrams can be encoded as axiom sets in OWL (Web Ontology language). The last work did not cover behavioural modeling, but it suggested that behavioural modeling should follow the same path of the structural modeling, i.e. behavior should be encoded as sets of axioms.

Alf [21], and the foundational subset for executable UML models (fUML) [19], combines the meta-modeling and an extension of set-theory, because the semantics of behavior is described operationally by fUML itself, and by a set of axioms (we consider this in Section III).

A broad set of researches adheres to translation through definition of a mapping between SysML and a formal language. Bousse et. al. [7] proposed a transformation from a subset of SysML into a subset of the B method; the selected subset of SysML covers behavioural definitions expressed by Alf. Afterwards, the resulting B method representation is proved by a specialized tool. Pétin et. al. [23] defined transformation from SysML requirements and SysML behavior (defined by State Machine Diagrams and Activity Diagrams, without use of fUML) into temporal logic and timed automata, respectively. Henceforth, the UPPAAL model checker is used to check safety requirements. Abdelhalim et. al. [1] defined a method that receiving State Machine Diagrams and Activity Diagrams (according to fUML) applies a transformation to Communicating Sequential Processes (CSP). Later, the method uses a model checker to verify the resulting CSP representation. This work focuses on maintaining the behavioural consistency between State Machine Diagrams and Activity Diagrams. Abdelhalim et. al. [2] refined their initial approach defining a subset of CSP to be used because difficulties emerge when non-trivial fUML inter-object communication mechanism is formalized. This work identifies patterns that are correct from the modeller's point of view and the system representation; however, when model checking the CSP representation of this model is performed, a state space explosion problem may occur. Perseil [22] suggested that a subset of Alf should be translated to PlusCal, which has precise semantics defined by a translation to TLA (Temporal Logic of Actions); later, the model checker from TLA would be used.

Some degree of semantics for models is a prerequisite for verification. Taking into account verification, there are a large number of research papers about the verification of UML, and consequently SysML, behavioural models, focusing on State Machine Diagrams, Sequence Diagrams and Activity Diagrams; nonetheless, a way to check the correctness of behavioural representations is still not agreed [24]. Planas et. al. [24] presented a method to verify correctness of behaviors defined using Alf through analysis of all possible execution paths. This method uses as input an UML model, and performs its checks directly on this model. This work states that translating UML behavioural models into other formalisms or languages could compromise scalability of these proposed methods.

However, few researches addressed the problem of nondeterminism, and its roots, in behavioural representations using fUML, and Alf.

Benyahia et. al. [4] showed that fUML, and also Alf, is not directly feasible to safety-critical systems because the MoC defined in the fUML execution model is nondeterministic. In spite of variation points provided by fUML, this work recognized that they are not powerful enough to change the MoC, and an alternative extension of the core execution model was presented to accommodate different MoCs.

III. OMG Specifications and MoCs

Execution and verification of models is the cornerstone of any Model-Driven Development (MDD). One prominent alternative for MDD is Model-Driven Architecture (MDA) [17] established by OMG. MDA defines three levels of abstraction: (1) Computational Independent Model (CIM) – focuses on the environment of the mission and mission's requirements; (2) Platform Independent Model (PIM) – defines requirements, structure and behavior for candidate abstract solutions; (3) PSM (Platform Specification Model) – describes concrete solutions.

An important OMG specification for PIM is Alf [21]. Alf is a textual surface representation for UML modeling elements. It is an action language that includes primitive types (including real numbers), primitive actions (e.g. assignments), and control flow mechanisms, among others. It is object-oriented, and it is an imperative language (like C and Java). Further, Alf has the expressivity of OCL (Object Constraint Language) in the use and manipulation of sequences of values, enabling an OCL-like syntax.

The execution semantics of Alf is given by mapping the Alf concrete syntax to the abstract syntax of fUML [19]. fUML abstract syntax is a subset of UML with additional constraints, so a well-formed model is one that meets all constraints imposed on its syntactic elements by the UML abstract syntax as well as all additional constraints imposed on those elements by the fUML abstract syntax.

Moreover, the execution semantics of fUML is an executable model written in fUML. However, instead of using Activity Diagrams, activities are written as equivalent code in Java; to support that, a mapping from Java to Activity is defined for core elements of activities (Base UML - bUML). The circularity is broken by the base semantics for bUML, which is specified in first order logic based on Process Specification Language (PSL). PSL (ISO 18629) provides a way to disambiguate common flow modeling constructs in terms of constraints on runtime sequences of behavior execution; desired behavior is specified by constraining which of the possible executions is allowed [6]. Fig. 1 shows relationships between these OMG specifications. In the following, "fUML execution model" refers to fUML and Alf.

SysML reuses a subset of UML 2 and provides additional extensions to satisfy the necessities of systems engineering, e.g. Requirement Diagrams [20]. SysML and Alf integrate seamlessly because Alf can be used in context of models not limited to the fUML subset [19].

Concerning the MoC provided by UML, one basic premise from this modeling language is that all behaviors are ultimately caused by actions executed by active objects [18], which is an instance of an active class (executed concurrently).

This establishes concurrent processes (active objects) but does not define a specific MoC because all BehavioralFeatures (e.g., Operations and Receptions) in UML allow three types of concurrency: sequential, guarded, and concurrent. Therefore, the semantics is unconstrained, which supports heterogeneous MoCs; in fact, it is one of the goals of the specification.

fUML constrains the concurrency for all BehavioralFeatures to the sequential type; as a result, the sole mechanism for asynchronous invocation in fUML is sending signals (SendSignalAction) to other active objects [19]. Further, the sending action is not blocking, i.e., an object sends a signal and continues its execution; it does not wait for a response, or an acknowledgment (nonblocking write). In contrast, the reception action is blocking, i.e., one computation running is blocked when it expects to receive a determined signal (blocking read). Moreover, the received signals are stored in an unbounded event pool for each active object, which is a FIFO (first-in first-out) in the fUML standard execution model (this is a variation point [19]). Consequently, the fUML standard execution model is characterized by concurrent processes (active objects) communicating with each other through unidirectional unbounded FIFO event pools, where writes to the event pool are nonblocking, and reads are blocking.

These fUML's characteristics are what the Kahn process networks have [13]. However, fUML standard execution model defines that signals coming from different active objects should be stored in the same target event pool. Allowing more than one process to write to an event pool (channel), the resulting process network is neither deterministic [13] nor a Kahn Process Network (in the strict sense). Consequently, the resulting process network can be described by active objects that receive (input) and emit

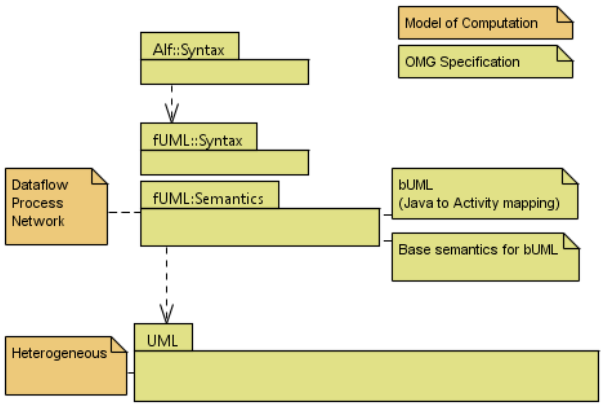

Fig. 1. Relationships between OMG specifications and MoCs.

(output) signals, and a set of firing rules (encoded in the behavior) defining when an active object should be fired; these characteristics are what the dataflow process networks have [13].

Nondeterminism can be a powerful modeling tool, but it should be used only when necessary [13]. Consequently, deterministic languages that allow nondeterminism remove it using precise techniques, e.g. the Quartz [26] compiler adds new control events to remove nondeterminism allowed by some statements.

Despite the nondeterminism of fUML MoC, it is designed to support a variety of different MoCs. This is pursued using two techniques: (1) defining explicit variation points, which are: event dispatching scheduling (used in the inter-object communication), and polymorphic operation dispatching; (2) leaving some semantics elements unconstrained that are: timing, concurrency, and inter-object communication.

IV. The Initial Approach

CPSs are often safety-critical systems [14]; hence executable models describing them must be deterministic: given a state $x(t_i)$ and an input $w(t_i)$ the system must generate the same output $u(t_i)$ for each reaction in state $x(t_n)=x(t_i)$ and input $w(t_n)=w(t_i)$.

The fUML specification states that there are a number of cases in which the UML indicates that the execution semantics in a certain area are nondeterministic [19]. In order to understand these nondeterministic areas, the next subsection discusses the roots of nondeterminism in the fUML execution model.

A. fUML and Nondeterminism

In order to analyze fUML's nondeterminism, behavior should be classified, which is done by UML [18] as: (1) intra-object behavior addresses the behavior occurring within classes; (2) inter-object behavior, which deals with how active classes communicate with each other.

The roots of nondeterminism in the fUML specification can be grouped as follows: (1) structural features manipulation – e.g. set one value to a property of an object; (2) conditions – fUML conditional clauses, e.g. defined using if or switch Alf statements; (3) token flow semantics – defines intra-object behavior semantics, e.g. how are tokens offered, and, consequently, in which sequence are nodes executed; (4) ObjectActivation – a key class responsible to bind inter-object behavior with intra-object behavior.

1) Structural feature manipulation

A property in a class, defined by a modeller, is a StructuralFeature in the meta-model of UML. Actions that write or remove values in a StructuralFeature can be nondeterministic. The nondeterminism occurs when a target property has multiplicity greater than 1, it is not ordered, and it does not have the uniqueness property; i.e., the property is a bag.

This nondeterminism can be a challenge for verification but it compromises neither the given definition of deterministic models nor the fUML MoC.

2) Conditions

Conditions are modeled in fUML using ConditionalNodes. ConditionalNode has an association with Clauses; each Clause can have an association with predecessor Clauses. The fUML execution model states that sequential evaluation is performed when the predecessor chain is defined.

Two statements in Alf map to ConditionalNodes in fUML: if, and switch. The statement if is mapped using predecessor clauses in fUML when the modeller uses the construct "if (*condition) else …"*, so the sequence of evaluation of clauses is deterministic; on the other hand, when the modeller uses the construct "if (condition) or *…"* the sequence of evaluation of clauses is nondeterministic. Finally, the statement switch is mapped without use of predecessor clauses in fUML so the evaluation of clauses is not deterministic.

As a result, the modeller has two options to produce a deterministic model, concerning conditions using Alf: (1) define conditions that are mutually-exclusive (assured by the modeller, or by an automated assistant); or, (2) use the Alf construct "*if (condition) else …"*. A nondeterministic model is defined otherwise.

This nondeterminism compromises the given definition of a deterministic model, but it does not affect the fUML MoC.

3) Token flow semantics

fUML states that different execution traces for the same inputs in an identical environment (including same state) are allowed to be different [19].

For example, given two actions that are not directly or indirectly ordered by their relationships, the order of execution is determined neither by UML semantics nor by fUML execution model, as recognized by [4]. Other example, a ForkNode enables race conditions. Therefore, nondeterminism is established in the intra-object behaviors.

Some basic nondeterminism (coming from UML), in the token flow semantics, are removed by semantic mapping from Alf to fUML, e.g., a naive modeller can, using fUML, connect an OutputPin at two InputPins without using a ForkNode (it copies tokens). However, that construction is not possible in Alf, which generates a ForkNode for each local name [21].

This nondeterminism (if these different traces lead to different outputs or signals sent to other active objects) compromises the given definition of deterministic models, and can contribute to the nondeterminism in the fUML MoC.

4) ObjectActivation

ObjectActivation is the class defined in the execution model to handle the active behavior of an active object. It is responsible to bind inter-object behavior with intra-object

behavior because it, together with EventAccepters, offers the blocking read feature for fUML MoC.

Two associations of this class are important for analysis of nondeterminism: (1) eventPool - the list, without upper bound, of pending signals sent to the object handled by this object activation; (2) waitingEventAccepters - the set of event accepters waiting for signals to be received by the object handled by this object activation.

For example, an execution sequence (ES) for two active objects communication can be explained as follows: (1) an active object (A) reaches an AcceptEventAction (statement accept defined by Alf), this is a blocking read for a signal; (2) the corresponding ObjectActivation object registers an EventAccepter in the waitingEventAccepter; (3) another active object sends a signal, that matches (A) receptions, and the registered accept statement; (4) the ObjectActivation object inserts this new signal at the end of eventPool; (5) considering that eventPool had no previous signals, this signal is removed from eventPool, dispatched to respective accept statement, and EventAccepter is unregistered.

The step (5) is one of two explicit variation points from fUML, called event dispatching scheduling. The standard execution model provides the implementation described above, where events are dispatched from the pool using a first-in first-out (FIFO) rule.

The ObjectActivation is the key to understand how nondeterminism in the fUML MoC and in the token flow semantics is combined. Exploring the execution model of fUML, Fig. 2 shows an Activity Diagram for an active class. Further, Fig. 3 shows an Alf representation for the Activity Diagram presented in Fig 2.

In Fig. 2, there are two concurrent AcceptEventAction waiting for the same type of signal; they are designed to execute two different tasks using received signals. The ForkNode, together with the fact that the next two actions wait for the same signal, defines a race condition, where the output depends on the sequence of tokens offered. Considering that a signal sent by another active object arrived after the two EventAccepters were registered, and the execution sequence (ES) presented above; during the event dispatching phase (5), there are two registered EventAccepters. In this case, the execution model chooses nondeterministically one of these [19], dispatches the event to it, and unregisters it.

This nondeterminism compromises the given definition of a deterministic model, and contributes to the nondeterminism in the fUML together with fUML MoC.

B. Proposed specialization of Alf and fUML

The initial approach is described as follows: given the semantics defined by fUML, we specialize the explicitly unconstrained elements with the purpose of deterministic behavioural definitions using SysML and Alf. We chose to discuss the semantics in an informal way, and to present concrete additional Alf constructs for the specialization.

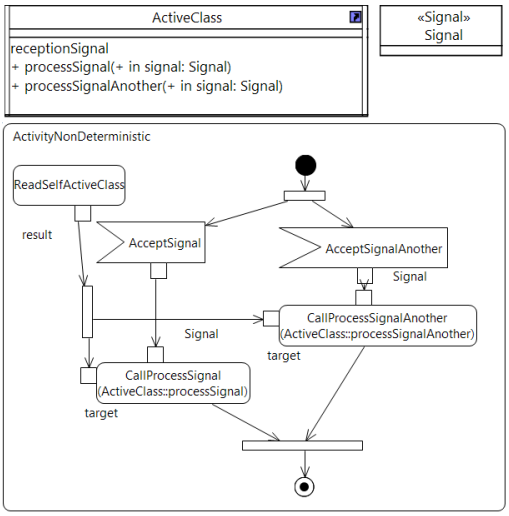

Fig. 2. fUML Activity diagram – nondeterministic.

These additional language constructs are defined using Annotation. According to Alf abstract syntax [26], annotation is a way to identify a modification to the behavior of an annotated statement. The applied approach allows us an early evaluation of the proposed specialization.

Therefore, a first concern is to introduce a synchronous-reactive MoC on fUML and Alf. A second concern is to specialize fUML and Alf, which means: do not change syntax parsing of Alf, but change its semantics.

The next three subsections explore the introduction of the synchronous-reactive MoC in Alf using unconstrained elements, and the variation points. The fourth subsection summarizes the proposal.

1) Timing

The behavioral semantics of UML only deals with discrete behaviors [18]. Accordingly, the timing semantics proposed divides the time scale in a discrete sequence of instants, each instant corresponds to one macro step as defined in the next subsection.

The annotation @delayed was introduced; it is the only way to assign new values to an already assigned variable in the current macro step. This annotation can be used in the assignments and in the SendSignalActions.

2) Concurrency

Concurrency can be achieved in Alf using two complementary techniques: (A) multiple active objects that, in general, imply the necessity of inter-object communication; or, (B) inside a given definition by the use of the annotation @parallel [21].

The alternative proposed is a combination of concurrency and synchrony (where computation and communication are instantaneous) through introducing the synchronous-reactive MoC to fUML and Alf. According to this MoC, a program can be defined by so-called micro and macro steps. Each macro step is divided into finitely many micro steps, which are all executed in zero time and within the same variable

environment. As a consequence, the values of the variables are uniquely defined for each macro step. Macro steps correspond to reactions of reactive systems, while micro steps correspond to atomic actions [26], e.g., assignments using Alf.

The demarcation of macro steps was introduced by the annotation @pausable; it is one of two ways to define demarcation between two macro steps. The second way is the use of the accept statement. This annotation is designed to be used with loop constructs (while, for, do while) but it can be also used with an empty statement of Alf. The semantics is: after each execution of the loop body, it waits for the next macro step. It follows that all concurrent behaviors run in lockstep: they execute the actions inside the loop in zero time, and synchronize before the next iteration.

The annotation @parallel can be used to define that all the statements in the block are executed concurrently. The block does not complete execution until all statements complete their execution; i.e., there is an implicit join of the concurrent executions of the statements [21].

Alf provides also an annotation called @isolated, it is defined in [21]: no object accessed as part of the execution of the statement or as the result of a synchronous invocation from the statement may be modified by any action that is not executed as part of the statement. Similar to this annotation, Alf provides the isolation expression through character $. Both options are not compatible with the synchronous-reactive MoC, where variables are uniquely defined for each macro step.

3) Inter-Object Communication and Event-Dispatching

Inter-object communication in Alf is performed by sending signals to other active objects. A signal is a specification of what can be carried; furthermore, a signal event represents the receipt of a signal instance in an active object [21]. A signal instance is identified by its contents.

Signals are based on the paradigm of message passing; furthermore, fUML provides a point-to-point (also known as unicast) message pattern. A signal is sent to a receiver (an active object) using a reference to it. In contrast, multicasting is required in many safety-critical systems, e.g., fault-tolerance by active redundancy [16]. Multicasting also supports the non-intrusive observation of component interactions by an independent object, and enables better composition [16].

```
                                                    //ACC0
//@parallel
{
    {
        accept( sig:Signal );                       //ACC1
        this.processSignal( sig );
    }
    {
        accept( sigAnother:Signal );                //ACC2
        this.processSignalAnother( sigAnother );
    }
}
```

Fig. 3. Alf representation for fUML Activity diagram – nondeterminitisc.

Multicasting is provided by the introduction of an active class called MessageDispatcher; it provides a service for multicast message exchange. Instances of this class work as bus transferring instances of signals between previously registered active objects, which generate events in the target active object. Every signal handled by MessageDispatcher has a specific identifiable sender, and zero or more receivers.

The set of receivers (active objects) is defined by existence of the reception for the sent signal. All signals generated in the current macro step are available instantaneously in the synchronous-reactive MoC. Further, signals not consumed during a macro step are lost. Delayed SendSignalActions are available in the next macro step.

It is possible to receive signals individually or as a set. Receiving a set of signals is important for those active objects that need to process all signals sent in the current macro step. However, individual signal receiving is fundamental for those active objects that should only process one signal sent to them. For this case (individual signal), the annotation @nonblocking was introduced; it is the only way to receive signals without blocking (nonblocking read).

In a macro step just one signal value (a signal is identified by its contents) is allowed for a given signal type, and MessageDispatcher; therefore, values of the signals for a given MessageDispatcher are uniquely defined for each macro step.

4) Summary

Table I summarizes the annotations available in the specialization of Alf. All other annotations available in Alf now are just comments, as well as, isolation expressions.

Considering that execution model of fUML has changed to accommodate proposed specialization, the semantics of Alf representation in Fig. 3 changes. As just one signal value in a macro step is allowed for a given signal type, and MessageDispatcher; the same signal instance is dispatched for those two parallel accepts, and computation follows in the same macro step concurrently.

The specialized semantics removes the nondeterminism indicated in section “IV.A.4 Object Activation” as described earlier. Also, it removes the nondeterminism indicated in section “IV.A.3 Token flow semantics” because the ordering of micro steps does not influence the semantics of a model. However, the new semantics does not remove the nondeterminism indicated in section “IV.A.2 Conditions”,

TABLE I.
ANNOTATIONS IN THE SPECIALIZED ALF

Annotation	Informal semantics
@delayed	Delayed assignment or SendSignalAction
@pausable	Macro step demarcation
@parallel	Computations on each block are carried out concurrently
@nonblocking	AcceptEventAction read nonblocking, makes optional signals available

which should be rejected by an interpreter for proposed semantics (when conditions are not mutually-exclusive).

With the proposed specialization, Fig. 3 can be changed without modification of the semantics: the two accepts (ACC1 and ACC2) could be removed, and a new one (ACC0) could be inserted before the concurrent block. This is referential transparency, which means syntactically identical expressions have the same semantics regardless of their lexical position [13].

5) Example

We evaluate the example from [4] but a case study with well-known CPS is [25]. Fig. 4 shows the Block Definition Diagram (BDD) for it. A PingPongSystem is composed by one Player1 and one Player2; both players are active classes. These two active classes communicate by exchanging signals Ping and Pong. The respective Alf representation for the behavior of each player is presented using comments.

In a given macro step, Player1 sends a delayed Pong (P11), and awaits for Ping (P12). In the next macro step, Pong is received by Player2 (P21), who sends a delayed Ping (P22). The game continues forever as showed in Fig. 5.

Fig. 5 shows the Internal Block Diagram (IBD) for the system, and the Alf representation for the main behavior. Player1 (S3) and Player2 (S2) are created passing an object of MessageDispatcher (S3); later, an infinite loop annotated with @pausable (S5), containing an empty statement, is used to define the evolution of time.

In contrast to [4], which uses static Association between the players, it is used Connectors that specify links between instances playing the connected parts only [18] (decoupling Player1 from Player2). The communication is provided by the instance of MessageDispatcher. The Alf specialization makes the example different concerning evolution of time, signal events, and communication. Therefore, this model is deterministic while [4] is nondeterministic.

V. DISCUSSION

Activity Diagrams are used frequently [1][4][2][23][24]; however, for significant activities, these diagrams quickly become large, intractable to draw and hard to comprehend [19].

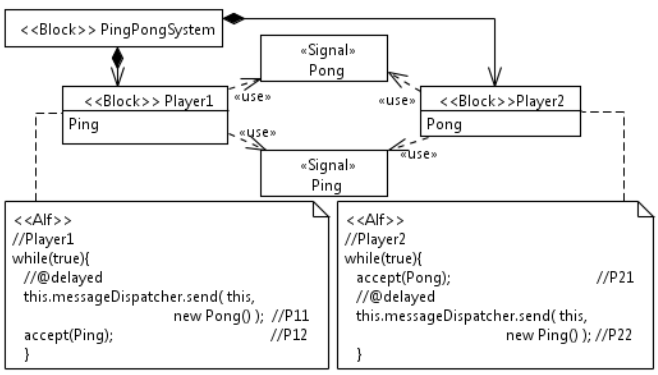

Fig. 4. BDD for the PingPongSystem.

State Machine Diagrams are another commonly used form of diagrams, especially suited for modeling state-based behavior [1][2][4][11][23][24]. However, UML, fUML, SysML, and Alf do not define precise semantics for state machines [9]. This is ratified by the Alf specification itself, which states that a normative semantic integration of state machines with Alf will be formalized later [21]. Indeed, environments of synchronous languages offer tools to visualize the resulting automata [3], e.g. Fig. 3 can be automatically transformed in a State Machine Diagram.

Transformation from SysML to other languages or formalism could bring some serious problems [12], and could compromise scalability [24]. However, we consider the certification process [22] more challenging because it is needed to assess the original model, and the translated model (or even the transformation itself). Nevertheless, these transformations are powerful, and can provide feedback for the fUML specification MoC. For example, [2] defines a pattern suitable of optimization called "fUML-Opti-Rule(2): Detecting unacknowledged signals" - an unacknowledged signal is one that has been sent from an active object to another active object, and then it (source object) continues sending further signals without waiting for an acknowledgment signal. This pattern is detected through model checking executed over a CSP representation, which is the result of a transformation of a fUML model [2]. Based on this feedback, the modeller should evaluate acknowledging those signals to reduce the state space of the corresponding CSP model. Although, this is a rendezvous that is common in CSP MoC; considering this case, fUML MoC needs more design effort than CSP MoC.

Concerning [10][11] which propose to encode SysML structure as a set of axioms, fUML and PSL [6] are well suited, hence axioms about structure and behavior can be combined and evaluated together.

The evaluation presented corroborates [4] concerning two points about fUML (and also Alf) as it is: (1) the execution model is nondeterministic; (2) it is not suitable for safety-critical systems. Nonetheless, Alf should be specialized to allow safety-critical systems modeling [22].

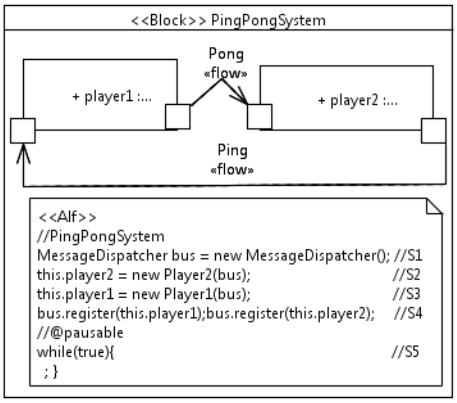

Fig. 5. IBD for the PingPongSystem (graphical view of flows).

The proposed specialization of Alf adheres the idea of introducing synchronous-reactive MoC during early stages of a system development [3]. The major drawback of this MoC is that the computer interpretation of the models is difficult [3]; further, polymorphism, reclassification, and dynamicity (actions: create, and destroy) can be even more challenging [3].

fUML states that every specialization must be defined using bUML; in fact, the initial approach presented here provides a complete description from the viewpoint of the modeller. It defines the semantics for three additional constructs for Alf that together with MessageDispatcher can transform Alf in a synchronous action language; however, the changes needed in the fUML execution model to support it must be defined.

VI. Conclusion

This paper shows the results of the proposed specialization of Alf, according to the synchronous-reactive MoC. It helps to couple complexity, provides better composition, and enables deterministic behavior definitions.

CPS is about the intersection of the computation, control and communication [14]. The initial approach focuses on the discrete computational and communicational aspects of CPSs. It can be composed with discrete control. A case study with a well-known CPS [25] shows that the initial approach can bring solid mathematical foundation from synchronous-reactive MoCs [3] to SysML executable models. We consider this as an intermediary step, located before a formal verification of executable discrete SysML models.

In summary, we believe that specializing well-known vendor-independent specifications (Alf and SysML) can provide an understandable and compact set of languages for modeling, analyzing and verifying of CPSs. Moreover, such a set of languages can enable formal verification for discrete parts of CPSs.

References

[1] Abdelhalim, I.; Schneider, S.; Treharne, H. (2011). Towards a practical approach to check UML/fUML models consistency using CSP. In Proc. ICFEM 2011 Proceeding of the 13th International Conference on Formal methods and software engineering, 2011, pg. 33-48.

[2] Abdelhalim, I.; Schneider, S.; Treharne, H. (2012). An Optimization Approach for Effective Formalized fUML Model Checking. In Proc. SEFM2012 Proceeding of the 10th International Conference on Software Engineering and Formal methods, 2012, pg. 248-262.

[3] Benveniste, A. ; Caspi, P.; Edwards, S.; Halbwachs, N.; Guernic, P.; Simone, R. (2003). The synchronous languages twelve years later. Proceedings of the IEEE, 2003, pg. 64–83.

[4] Benyahia, A.;Cuccuru, A.; Taha, S.; Terrier, F.; Boulanger, F.; Gérard, S. (2010). Extending the Standard Execution Model of UML for Real-Time Systems. In Proc. DIPES/BICC, 2010, pg. 43-54.

[5] Berry, G. (2000). The Esterel v5 Language Primer: version:5.91. France. Available at: <http://francois.touchard.perso.esil.univmed.fr/3/esterel/primer.pdf>. Access date: 14.Apr.2013.

[6] Bock, C.; Gruninger, M. (2005). PSL: A semantic domain for flow models. In Software & Systems Modeling, May 2005, Volume 4, Issue 2, pp 209-231, 2005. Springer.

[7] Bousse, E.; Mentré, D. Combemale, B.; Baudry, B.; Katsuragi, T. (2012). Aligning SysML with the B Method to Provide V&V for Systems Engineering. Proc. Of 12th Model-Driven Engineering, Verification, and Validation 2012.

[8] Cartwright, R.; Kelly, K.; Koushanfar, F.; Taha, W. (2006). Model-Centric Cyber-Physical Computing. In proceedings … NSF Workshop on Cyber-Physical Systems, 2006, Austin, Texas: USA.

[9] Fecher, H.; Schönborn, J.; Kyas, M.; Roever, W. (2005). 29 New Unclarities in the Semantics of UML 2.0 State Machines. In Proceedings of the Int. Conf. on Formal Engineering Methods, LNCS 3785, Berlin/Heidelberg, Germany, Springer-Verlag, 2005, pg. 52-65.

[10] Graves, H. (2012). Integrating Reasoning with SysML. In Proc. of 22th Annual INCOSE International Symposium. Rome, Italy, July, 2012.

[11] Graves, H.; Bijan, Y. (2011). Using formal methods with SysML in aerospace design and engineering. Journal Annals of Mathematics and Artifical Intelligence. Volume 63, Issue1, September, 2011. pg 53-102.

[12] Hußmann, H. (2002). Loose semantics for UML, OCL, in: Proceedings 6th World Conference on Integrated Design and Process Technology, IDPT 2002, June, Society for Design and Process Science, 2002.

[13] Lee, E.; Parks, T. (1995). Dataflow process networks. Proceedings of the IEEE, vol. 83, no. 5, May, 1995. pg. 773-801.

[14] Lee, E.; Seshia, S. (2011). Introduction to Embedded Systems - A Cyber-Physical Systems Approach. http://leeseshia.org/, 2011. ISBN 978-0-557-70857-4.

[15] Miller, P.; Whalen, M.; Obrien, D.; Heimdahl, M.; Joshi, A. (2005). A methodology for the design and verification of globally asynchronous/locally synchronous architectures. NASA Contractor Report NASA/CR-2005-213912.

[16] Obermaisser, R.; Kopetz, H. (2009). Genesys – A candidate for an ARTEMIS Cross-Domain Reference Architecture for Embedded Systems. 2009. Available at: <http://www.genesys-platform.eu/genesys_book.pdf> Access date: 17.May.2011.

[17] Object Management Group (OMG). (2003). Model-Driven Architecture. USA: OMG, 2003. Available at: <http://www.omg.org/mda>. Acesso em: 17 may. 2009.

[18] Object Management Group (OMG). (2011). Unified Modeling Language Superstructure: Version: 2.4.1. USA: OMG, 2011. Available at: <http://www.omg.org/spec/UML/2.4.1/>. Access date: 14.Apr.2013.

[19] Object Management Group (OMG). (2012). Semantics of a Foundational Subset for Executable UML Models: Version 1.1 RTF Beta. USA: OMG, 2012. Available at: <http://www.omg.org/spec/FUML/>. Access date: 24.Apr.2013.

[20] Object Management Group (OMG). (2012). Systems Modeling Language: Version: 1.3. USA: OMG, 2012. Available at: < http://www.omgsysml.org/>. Access date: 24.Apr.2013.

[21] Object Management Group (OMG). (2013). Concrete Syntax for UML Action Language (Action Language for Foundational UML - ALF): Version: 1.0.1 - Beta. USA: OMG, 2013. Available at: <http://www.omg.org/spec/ALF/>. Access date: 27.Apr.2013.

[22] Perseil, I. (2011). ALF Formal. Journal Innovations in Systems and Software Engineering, Volume 7, Issue4, December, 2011. pg. 325-326.

[23] Pétin, J.; Evrot, D.; Morel, G.; Lamy, P. (2010). Combining SysML and formal models for safety requirements verification. In 22nd International Conference on Software & Systems Engineering and their Applications, France, 2010.

[24] Planas, E.; Cabot, J.; Gomez, C. (2011). Lightweight Verification of Executable Models. In Proc. ER 2011 Proceedings of the 30th International Conference on Conceptual Modeling, 2011. pg. 467–475.

[25] Romero, A. G.; Schneider, K.; Ferreira, M. G. V. (2013). Synchronous Specialization of Alf for Cyber-Physical Systems. In First Open EIT ICT Labs Workshop on Cyber-Physical Systems Engineering, 2013, Trento, Italy.

[26] Schneider, K. (2009). The synchronous programming language Quartz. Internal Report 375, Department of Computer Science, University of Kaiserslautern, Kaiserslautern, Germany, December 2009.