Synchronous Specialization of Alf for Cyber-Physical Systems

Alessandro Gerlinger Romero¹, Klaus Schneider² and Maurício Gonçalves Vieira Ferreira³

¹ Brazilian National Institute for Space Research, São Paulo, Brazil.
romgerale@yahoo.com.br
² Department of Computer Science, University of Kaiserslautern, Germany.
schneider@cs.uni-kl.de
³ Brazilian National Institute for Space Research, São Paulo, Brazil.
mauricio@ccs.inpe.br

Abstract

Systems engineers often use SysML as a vendor-independent language to model cyber-physical systems. However, SysML does not provide an executable form of behaviors which is needed for simulation to detect critical issues as soon as possible. In this paper, we therefore present an action language for foundational UML (Alf) specialization that introduces the synchronous-reactive model of computation to SysML. This is done by definition of not explicitly constrained semantics of timing, concurrency, and inter-object communication. The smart parking system, a well-known cyber-physical system benchmark, was selected to evaluate this specialization. Our initial results show that the proposed specialization does not add complexity to the task of modeling using SysML, and leads to concise and precise behavioral definitions.

1 Introduction

Cyber-physical systems (CPSs) are obtained by integration of computational and physical processes: Embedded computers and networks monitor and control physical processes with feedback loops where physical processes affect computations and vice versa [17]. According to Cartwright et. al. [9], the difficulty in modeling CPSs comes from the diversity of these systems. Therefore, the most promising approach to mitigate this problem is to develop expressive and precise modeling languages.

As a result, a large number of languages and formalisms have been proposed to model CPSs [8]. One particular branch of these languages follows the synchronous-reactive model of computation (MoC) [4], which has several advantages for specifying, modeling, and verifying of reactive real-time systems [17].

The synchronous-reactive MoC provides a precise behavioral model using discrete reaction steps as the fundamental model of time, while computation and communication are executed in zero-time, and parallel composition is defined as a conjunction of behaviors [4]. There is a solid mathematical foundation that supports synchronous-reactive MoCs, which allows in particular formal analysis and verification. Languages based on this MoC, like Esterel [6], have therefore been developed and used for safety-critical systems [4][28]. Some of them, like Quartz [29], have been extended for CPSs [8] by special kinds of variables and statements to deal with continuous time.

Comparing an implementation based on the synchronous-reactive MoC with an alternative asynchronous implementation for a dual redundant flight guidance system, Miller et. al. [20] 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”.

Meanwhile, the Object Management Group (OMG) and the International Council on Systems Engineering (INCOSE) are developing the Systems Modeling Language (SysML) [25, 31]; a general-purpose modeling language for systems engineering applications based on the Unified Modeling Language (UML) [23]. SysML has demonstrated a capability for top-down design refinement, but the lack of formal foundations of SysML results in imprecise behavioral models.

In this paper, we present a specialization to the action language for foundational UML (Alf) [26] for behavioral modeling of CPSs. The hypothesis of this work is that a specialization of Alf according to the synchronous-reactive MoC can be sufficiently expressive to model the discrete behavior of CPSs using SysML. Consequently, adhering to the synchronous-reactive MoC, we will benefit from a solid mathematical foundation [4, 6, 29].

The remainder of this paper is organized as follows: in Section 2, related works are explored briefly; in Section 3, we present the initial approach; in Section 4, a case study is presented; in Section 5, we briefly discuss the initial approach and the case study; finally, conclusions are shared in the final section.

2 Related Work

There is a large number of research papers about formalizing the semantics of models using UML, and consequently, also about SysML. Hußmann [16] proposes the following classification for approaches concerning structural semantics: (a) naive set-theory, (b) meta-modeling, and (c) translation. This classification can be used for the works focused on behavioral semantics.

For example, Graves and Bijan [15] propose an approach where behaviors defined using SysML state machine diagrams are axiomatized using set/type theory. Alf [26], and the foundational subset for executable UML models (fUML) [24], follow the meta-modeling approach because the semantics of behaviors is described operationally using fUML itself. The circularity is broken by the base semantics of fUML which is specified using first order logic. However, Benyahia et. al. [5] show that fUML, and also Alf, are not directly feasible to safety-critical systems because the MoC defined in the fUML execution model (as it is) is nondeterministic and sequential.

Following (c), i.e., translation, Bousse et. al. [7] define a method to transform a subset of SysML in B method representations; the selected subset of SysML covers behavioral definitions expressed by Alf. Later, the B method representation is proved by a specialized tool. Abdelhalim et. al. [1] define a method that receiving state machine diagrams and activity diagrams (according to fUML) applies a transformation to communicating sequential process (CSP). Then, the CSP representation is verified by a specialized tool.

3 Initial Approach

Execution and verification of models is the cornerstone of any model-driven development (MDD). One prominent alternative for MDD is model-driven architecture (MDA) established by OMG [22]. MDA defines three levels of abstraction:

(A) Computational Independent Model (CIM): to focus on the environment of the mission and mission’s requirements;
(B) Platform Independent Model (PIM): to define requirements, structure, and behavior for candidate abstract solutions;

(C) PSM (Platform Specification Model): to describe concrete solutions.

MDA established a large number of specifications, but for this paper, the most important one is Alf [26]. Alf is the concrete syntax for the abstract action language defined by fUML [24], i.e., a subset of UML [23]. The execution semantics for Alf is therefore given directly by fUML. According to INCOSE [27], fUML and Alf are MDA pillars for the definition of PIMs.

fUML [24], which defines the semantics for Alf, is designed to support more than one MoC. This is pursued with leaving the semantics of some elements unconstrained. These elements define aspects of concurrency and inter-object communication which work for simulation, whereas they are not suitable for formal verification. In particular, fUML does not define semantics for: (A) timing, (B) concurrency, and (C) inter-object communication.

Our initial approach is as follows: Given the semantics defined by fUML, we specialize the explicitly unconstrained elements with the purpose of precise definitions of models using Alf. In order to do this, we discuss proposed changes in the semantics of fUML. Further, we choose to discuss the semantics in an informal way, and to present a concrete additional language construct for the specialization of Alf. This additional language construct is defined using annotation, which is a way to identify a modification to the behavior of an annotated statement [26]. The applied approach allows early evaluation of the proposed specialization.

3.1 Timing

The timing semantics used divides the time scale in a discrete succession of instants. Similar to languages based on the synchronous-reactive MoC, each instant corresponds to one macro-step as defined in the next subsection.

3.2 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.

Active objects are the source of all behaviors in a system modeled with UML [22], SysML, fUML, and Alf. An active object is an instance of an active class. An active class must have a ClassifierBehavior that defines the class behavior. Each active object is executed independently, and the only way to communicate with other active objects is through signals [23].

One alternative to provide a combination of concurrency and synchrony (where computation and communication are instantaneous) is by using the synchronous-reactive MoC. 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 (i.e., the ordering of micro steps does not influence the semantics of a model). 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 with atomic actions, e.g., assignments of the model that implements these reactions [29].

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 of Alf. This annotation is designed to be used with loop constructs (while,
for, do while), and the semantics is as follows: 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 [26].

### 3.3 Inter-Object Communication

Inter-object communication in Alf is performed sending signals (SendSignalAction) to other active objects [23]. Further, this action is not blocking, i.e., an object sends a signal and continues with its execution (it does neither wait for a response nor for an acknowledgment). 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 [24].

Signals are based on the paradigm of message passing. Furthermore, fUML provides a point-to-point (also known as unicast) message pattern [24]. A signal is sent to a receiver (active object) using a reference to it. In contrast, multicasting is required in many safety-critical systems, e.g., fault-tolerance by active redundancy [21]. Multicasting also supports the non-intrusive observation of component interactions by an independent object. Moreover, it enables a better composition.

Multicasting was introduced by an active class called MessageDispatcher that provides the 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 active objects (receivers) is defined by the existence of the reception of that signal. All signals generated in the current macro-step are instantaneously available. Moreover, signals not used during a macro step are lost. 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 for it in the current macro-step.

### 4 Case Study

We evaluated our initial approach by a case study called SmartParking that is due to [13]. The points discussed in the previous section were applied to model a part of this system. The SmartParking benchmark has been chosen for three respective reasons: (1) it is a real-world cyber-physical system, (2) it can be modeled as a discrete system [13], and (3) Geng and Casandra [13] provide a detailed concrete solution. According to [11, 13], the case study is defined using MDA. The case study focuses on aspects related to computation and communication at the PIM abstraction level. A way to cover the control aspect is presented in [13].

#### 4.1 Mission Context and Requirements

Mission context and mission requirements were gathered and modeled in a SysML CIM Model. The mission is summarized as follows: A user inside a vehicle shall be able to request a parking space. The request for a parking space shall be evaluated considering two constraints given by the user: (a) maximum distance from current position, and (b) maximum cost that the user wants to pay. The user shall receive a response indicating the best parking space that satisfies
the imposed constraints. The user shall be able to accept or reject this response. The user shall be informed about where the parking space reserved for him/her is, as well as, about the availability of all other parking spaces up to 10 meters away from his/her current position. The vehicle shall be able to send its current position. The vehicle shall be detected when it arrives at a parking space, and when it leaves a parking space.

4.2 An Abstract Solution

Figure 1 shows the block definition diagram (BDD) for an abstract solution which is compatible with the concrete solution defined in [13]. The SmartParking system was decomposed in three main parts: SmartParkingEnablerDevice, SmartParkingAllocationCenter, and Spot. All of them are active classes.

The connections between these elements are not static. Therefore, they are not presented in Figure 1 as associations. The connections are showed in the internal block diagram (IBD) presented in Figure 2. In contrast to associations, which specify links between any instances of the associated classifiers, connectors specify links between instances playing the connected parts only [23]. The inter-object communication is provided by the multicast message exchange service (MessageDispatcher). Further, each active object has a reference to the same instance.
SmartParkingEnablerDevice models a device inside the vehicle. It receives Position from vehicle, and has a UserInterface (both interactions with the environment are depicted in the upper left corner in Figure 2). Each vehicle has a corresponding SmartParkingEnablerDevice active object. The abstraction used in this case study makes the internal structure of this component irrelevant. It, as well as other components, could be modeled later as software, hardware or a composition of both. For example, SmartParkingEnablerDevice could be implemented as software in a smartphone [13].

Each parking space managed by the system is an active object Spot, and each Spot has two interactions with the environment: (a) detecting that a vehicle arrived at a Spot (VehiclePresenceSensor) and (b) indicating the current state of the Spot to the user, and which one is reserved for him/her (LightsActuator). Spot and SmartParkingEnablerDevice (plant) are both managed by a block SmartParkingAllocationCenter (controller). In this case study, there is only one active object of this block, which is responsible in each macro step for: (a) gathering system state and events; and (b) determining the control output.

From the viewpoint of discrete event systems (DES) control, considering signals handled by SmartParkingAllocationCenter, the system can be described as follows:

\[ X(t) = \{D(t), P(t)\} \]  \hspace{1cm} (1)

\[ E = ECent \cup ED \cup ESpot \]  \hspace{1cm} (2)
\[ ECent = \{ RequestForSpotSP, AcceptSP, RejectSP \} \]  \hfill (3)

\[ ED = \{ SpotAvailableSP, SpotUnavailableSP, SpotTimeoutSP \} \]  \hfill (4)

\[ ESpot = \{ AllocatedSP, UnallocatedSP \} \]  \hfill (5)

\[ X(t + 1) = f(X(t), U(t), W(t)) \]  \hfill (6)

where:

Equation (1) defines the discrete state space \( X(t) \) composed of \( D(t) = \{ k \in \mathbb{N} \mid \text{SmartParkingEnablerDevice}_k \text{ in the system} \} \) (determined in each macro step by signal events of the signal DeviceStateSP) and \( P(t) = \{ k \in \mathbb{N} \mid \text{Spot}_k \text{ in the system} \} \) (determined in each macro step by signal events of the signal SpotStateSP). Equation (2) determines the discrete event set which is composed of signals \( ECent \), \( ED \), and \( ESpot \) as defined by the next equations: \( ECent \) (equation (3)) is received from \text{SmartParkingEnablerDevice}; \( ED \) (equation (4)) is sent to \text{SmartParkingEnablerDevice}, and \( ESpot \) (equation (5)) is sent to \text{Spot}. Equation (6) defines the evolution of the system over time, where the state \( X \) in the next macro step \((t + 1)\) is defined by the current state \( X(t) \), the current events \( W(t) \), and the current control signals \( U(t) \), where \( W(t) = \{ k \in \mathbb{N}, i \in ECent \mid \text{instance } i_k \} \) is determined by instances of signals defined in the set \( ECent \), and \( U(t) = \{ k \in \mathbb{N}, i \in ED \lor i \in ESpot \mid \text{instance } i_k \} \) is determined by instances of the signals defined in the sets \( ED \) and \( ESpot \).

Figure 3 shows that the Alf ClassifierBehavior of the \text{SmartParkingEnablerDevice} has two concurrent infinite loops. The first infinite loop depicted in Figure 3 is annotated with \@pausable, which means that it sends the current state of device. Thereupon, it waits for the next macro step (synchronization point, before next iteration). The current state is composed by the actual position and the state of current reservation, and is represented by an instance of the signal DeviceStateSP. Each active object sends this signal in each macro step using an instance of MessageDispatcher that is responsible for delivering a copy of these messages to every registered active object that has a reception for this signal.

The second infinite loop defines the expected reactions of the device for events received from UserInterface and from SmartParkingAllocationCenter. It starts with an accept statement, which blocks execution (possible during many macro steps) until the expected event occurs. Subsequently, it uses the same mechanisms described above to send signals for other active objects. Moreover, it uses a compound accept statement that determines which block will be activated based on the type of the signal received from UserInterface and from SmartParkingAllocationCenter.

The SmartParkingAllocationCenter behavior is shown in Figure 4. It has an infinite loop annotated with \@pausable that defines a synchronization point at the end of each execution of the loop body. The loop body starts with five concurrent accept statements, which means that it waits until no more signals of these types can be generated. Later, it applies the control law, and sends the response for other active objects (\text{SmartParkingEnablerDevice} and \text{Spot}) using the mechanism described above.

The Alf ClassifierBehavior of the \text{Spot} and the \text{SmartParkingEnablerDevice} are organized in the same way. There are two concurrent infinite loops: one sending signals about its state (with a synchronization point defined using \@pausable), and another one defining reactions for the received events from VehiclePresenceSensor and from SmartParkingAllocationCenter.
Figure 3: Alf ClassifierBehavior of SmartParkingEnablerDevice.
5 Discussion

The case study defines an abstract solution (PIM) for the mission that was modeled to explore concurrency, synchronization, and multicast messages. The solution is neither complete nor optimized, e.g., signals can be removed by a centralized version of the state of the system. A tradeoff could be evaluated taking into account an objective function defined at CIM level, e.g., considering the analysis of the messages (communication) during macro steps. In addition, the abstract solution has an important difference compared to the solution presented in [13]: there are no queues. This is a consequence of the synchronous-reactive MoC: All signals are received and processed in the same macro step. The SmartParkingEnablerDevice does not have the state “Waiting for Assignment” [13] because, given a macro step, the system state is gathered instantaneously. Afterwards, the control law is applied and all active objects in SmartParking
immediately receive an adequate response.

From the viewpoint of DES control [10], the case study satisfies the following key properties: (a) its state space is a discrete set, as defined in (1) and (b) the state transition mechanism is event-driven, which means that the state can only change as a result of asynchronously occurring instantaneous events over time [10]. Apart from that, the second property has a time window to occur during a macro step. In the case study, it is mandatory that many events occur in the same macro step, and the resulting state transition reflects the occurrence of all. However, some combinations of signals in the same macro step are not allowed, e.g., if a naive device sends in a given macro step one signal for requesting a spot, and one signal for acceptance, then the last one will be lost.

Concerning modeling, state machines and state machine diagrams are commonly used for modeling state-dependent behavior. A variation of these diagrams is used to express state-dependent behavior in [13]. However, UML, fUML, SysML, and Alf do not define precise semantics for state machines [12, 30]. This is ratified by Alf, which states that a normative semantic integration of state machines with Alf will be formalized later as a part of future standards [26]. Indeed, environments of synchronous languages offer tools to visualize the resulting automata from a given textual representation [6], e.g., Figure 3 can be automatically transformed in a state machine diagram. Languages have been developed to conciliate precise semantics and automata visual modeling as e.g., [2, 18].

The nondeterminism in the fUML MoC, which was recognized by Benyahia et. al. [5], can be removed using the proposed specialization. In fact, the proposed specialization adheres the idea of introducing the synchronous-reactive MoC during early stages of a system development [4]. It avoids asynchronous complexity in early stages of system modeling, analyzing, and verification. Furthermore, the synchronous-reactive MoC enables abstract solutions to be synthesized [28] in a concrete solution using globally asynchronous locally synchronous architectures (GALS) [20], or physically asynchronous locally synchronous architectures (PALS) [19].

The initial approach presented here provides rather a starting point than a complete result. It informally defines the semantics for two complementary constructs for Alf that together can transform Alf into a synchronous action language. However, the changes needed in the fUML execution model to support it must be defined, and the points about nondeterminism stated in [5] have to be addressed.

CPS is about the intersection of the computation, communication, and control [17]. The initial approach focuses on the computational and communicational aspects of CPSs, and it can be composed with control. The case study shows that our initial approach can transfer the solid mathematical foundation of synchronous languages to SysML executable models. We consider this step, as an intermediary step, before a formal verification of executable discrete SysML models.

6 Conclusions

This paper shows the initial results of our research that has the following basic hypothesis: A specialization of Alf according to the synchronous-reactive MoC can be sufficiently expressive to model the discrete behavior of CPSs systems using SysML. These results show that the proposed specialization does not add complexity to the task of modeling using SysML, and enables concise and precise behavior definition. We believe that specializing well-known vendor-independent specifications (Alf and SysML) can provide an understandable set of languages for modeling, analyzing and verification of CPSs. Moreover, such a set of languages can enable formal verification for discrete parts of CPSs.
Acknowledgments

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

References


