A Process Algebraic Approach to Decomposition of Communicating SysML Blocks

Jaco Jacobs & Andrew Simpson
Department of Computer Science, University of Oxford
Wolfson Building, Parks Road
Oxford OX1 3QD
{jaco.jacobs,andrew.simpson}@cs.ox.ac.uk

Abstract—The block concept is a fundamental modelling construct in the Systems Modeling Language (SysML), a visual modelling language for systems engineering applications. In a top-down systems engineering approach, an abstract block is decomposed into concrete communicating sub-blocks. However, the classifier behaviour of the abstract block must be exhibited by the composition of the concrete sub-blocks. We show how the process algebra Communicating Sequential Processes (CSP) and its associated refinement checker, Failures Divergence Refinement (FDR), may be used to ensure that such decompositions are valid. We introduce a small case study in order to validate the approach.

Index Terms—Systems Modeling Language, Communicating Sequential Processes, Composability

I. INTRODUCTION

The demand for aggregate systems — compositions of complex interconnected components — increases constantly; consequently, much research has been aimed at modelling approaches where the emphasis lies on treating a system as part of a larger composition. The Systems Modeling Language (SysML) [1], proposed by the Object Management Group (OMG)\(^1\), is a graphical modelling notation suited to specify and design systems comprised of various components.

Modelling a system with SysML relies on the concept of blocks — each with an associated set of states — that communicate via events, possibly resulting in a change of state for one or more of the communicating blocks. The architecture of these systems allows a top-down design, starting from an abstract level with high level concepts, down to levels with increasingly more details. These successive transformations allow replacing an abstract block with a composition of concrete sub-blocks. A major drawback of this decomposition, however, is that it is at best semi-formal and cannot guarantee consistency between an abstract block and its composing sub-blocks. In this paper, we propose a formal approach based upon Communicating Sequential Processes (CSP) [2], a well-established process algebra. In addition, the existence of Failures Divergence Refinement (FDR) [3], the associated refinement checker for CSP, makes translating SysML into CSP even more appealing: the translation provides us with a formal context in which we can evaluate and reason about behaviour without a detailed knowledge of the underlying mathematics of refinement.

The structure of the remainder of this paper is as follows. In Section II we provide brief introductions to CSP and SysML. Section III outlines our process algebraic approach to test for the validity of a decomposition. Then, in Section IV, we employ a small case study to validate our contribution. Section V draws comparisons with related work in this area. Finally, in Section VI, we conclude with a summary of the contribution of this paper, and identify potential areas of future research.

II. BACKGROUND

In this section we provide a necessarily brief introduction to CSP and SysML.

A. Communicating Sequential Processes

Events are at the heart of CSP descriptions: the set of all possible events within the context of a particular specification is denoted \(\Sigma\).

The process descriptions used in this paper can be characterised by the following grammar:

\[
P := \text{Stop} \mid a \rightarrow P \mid b \& P \mid P_1 \parallel P_2 \mid P_1 \parallel P_2 \mid P_1 \parallel P_2 \mid P_1 \parallel I \parallel P_2 \mid P_2 \mid P_1 \parallel I \parallel P_2 \mid P_2
\]

The alphabet of a CSP process \(P\), denoted \(\alpha P\), is the set of events that it is willing to communicate.

Stop is the simplest process: it is the process refuses to communicate anything at all.

The prefixing operator, \(\rightarrow\), allows us to prefix an event \(a \in \Sigma\) to a process \(P\).

\(b \& P\) is a guarded expression, where \(b\) is a Boolean condition that must be satisfied in order for a process to behave like \(P\).

\(^1\)http://www.omg.org
The focus of this paper is on modelling the behavioural aspects (and interactions) of blocks using state machines. Each block has an associated classifier behaviour, in our case described using a state machine. Each state machine has a set of states, and transitions between those states. The blocks communicate via events that act as stimuli for the associated state machines, causing the transitions to fire and resulting in a change of state. For the purposes of this paper, we assume that these events correspond to synchronous call events, as these allow us to demonstrate the relevant concepts more clearly.

### III. Formal Approach to Decomposition

Due to limitations of space, we consider only a subset of the available state machine constructs. However, the fundamental principles presented extends equally well to more complex descriptions.

#### A. Modelling State Machines

A state machine $M$ is a 5-tuple $(Q^M, q_0^M, f^M, T^M, E^M)$, which consists of:

- the finite set of states, $Q^M$;
- the initial state $q_0^M \in Q^M$;
- the final state $f^M \in Q^M$;
- the finite set of transitions $T^M$, and
- the finite set of events $E^M$.

In addition, we define the following functions to return for a transition:

- the source state $source^M : T^M \rightarrow Q^M$;
- the target state $target^M : T^M \rightarrow Q^M$;
- the triggering event $trigger^M : T^M \rightarrow E^M$; and
- the effect $effect^M : T^M \rightarrow E^M$.

For example, $effect^M(t)$ returns the behaviour that executes on the transition $t$. An effect is optional; in the case where it is omitted, we define the function to return the empty effect $effect^M(t) = \epsilon$ for a transition $t$.

Intuitively, the formalisation maps every block’s classifier behaviour — described by a state machine, $M$ — to a CSP process, by mirroring the syntactical structure of the corresponding state machine diagram.

For a state machine, $M$, each state $q^M \in Q^M \setminus \{q_0^M, f^M\}$ is mapped to a local state in the corresponding CSP process, say $q$. A guarded expression is then used to make the events corresponding to the outgoing transitions of the current state available to the environment. For example, assume that the current state is $q_1^M$, and we have two transitions $\{t_1, t_2\}$ emanating from that state, with $effect^M(t_2) = \epsilon$. We can then model the behaviour using the following process:

$$M(q) \equiv (q = q_1^M) \& trigger^M(t_1) \rightarrow effect^M(t_1) \rightarrow M(target^M(t_1))$$

$$\square (q = q_1^M) \& trigger^M(t_2) \rightarrow M(target^M(t_2))$$

\[ ^2 \text{In this paper we assume that every effect is also a trigger.} \]
of $A_{N-1}$. The more refined process, \textsc{concrete}, is at least as good as $A_{N-1}$.

Stated formally, we require

$$A_{N-1} \sqsubseteq_M \textsc{concrete}$$

The notion of refinement chosen will depend on our purposes: if we wish the composition to not only communicate within the valid sequences of events, but also, for a given trace, not to refuse an event that the abstract block would allow, we require a failures refinement. Otherwise, if we only wish for the composition to communicate sequences that the abstract block would, a traces refinement would suffice.

IV. Case Study

The case study is based on a passenger safety system of a vehicle, inspired by the contribution of Carrillo et al. [4]. The passenger safety system consists of an \textsc{accelerationsensor}, a \textsc{seatbeltock}, and an \textsc{airbagcontrol} block. The \textsc{accelerationsensor} and \textsc{seatbeltock} are concrete; the \textsc{airbagcontrol} block is abstract. We propose to decompose \textsc{airbagcontrol} into concrete \textsc{airbag} and \textsc{occupantdetectionsensor} sub-blocks, in order to perform its function. This decomposition is natural, as the an \textsc{airbag} and occupant detection sensor would conceivably be two separate entities in a vehicle. The occupant detection sensor detects whether there is an occupant present in the vehicle, and, if this is the case, whether it is an adult or infant. If an infant is detected, the passenger is given the option to either enable or disable the passenger’s \textsc{airbag} (in case of a flawed detection).

The state machine diagrams for the \textsc{airbagcontrol}, \textsc{airbag} and \textsc{occupantdetectionsensor} blocks are shown in Figures 2, 3, and 4, respectively.

Figure 5 shows the state machine diagram for the \textsc{accelerationsensor}. The sensor is responsible for monitoring the acceleration of the vehicle. If a hard deceleration is detected, it activates the seatbelt lock; conversely, if a deceleration consistent with impact is detected, the airbag is deployed. The concrete sub-blocks of \textsc{airbagcontrol} needs to be a refinement if they are to replace the higher level block and function correctly in the complete system.

The description of the \textsc{airbag} process is shown below. The disable and enable signals toggle the state of the airbag. Once the airbag is deployed, its state machine does not respond to any other events.

\begin{align*}
\text{Airbag}(q) \equiv & \bigl( q = \text{enabled} \bigr) \& \text{activate} \rightarrow \text{Airbag(\text{deployed})}
\bigl( q = \text{enabled} \bigr) \& \text{disable} \rightarrow \text{Airbag(\text{disabled})}
\bigl( q = \text{disabled} \bigr) \& \text{enable} \rightarrow \text{Airbag(\text{enabled})}
\end{align*}
The process `AirbagControl` models the higher level behaviour of the airbag, depending on whether there is an infant, an adult, or no occupant detected in the passenger seat.

\[
(q = \text{deployed}) \& \text{Stop}
\]

The set of events are given by:

\[
X = \{\text{enable}, \text{disable}, \text{activate}\}
\]

\[
Y = \{\text{detectNone}, \text{detectInfant}, \text{detectAdult}, \text{getEnable}, \text{getDisable}, \text{activate}\}
\]

In the above, we hide all events but those of `\(\alpha\)AirbagControl`. Using FDR and the hiding operator, we test if the decomposition is valid:

\[
\text{AIRBAGCONTROL} \sqsubseteq M \text{ CONCRETE}
\]

Depending on the model of refinement:
- if \(M = T\), then the refinement holds.
- if \(M = F\), then the refinement does not hold and a counterexample is returned. FDR suggests that after performing \(\langle\text{detectNone}\rangle\), the concrete composition refuses all events, whereas the abstract process would allow (not refuse) events from the set \(\{\text{detectNone}, \text{detectInfant}, \text{detectAdult}\}\). This can be remedied by adding a \text{Disable} transition-to-self in the \text{disabled} state of the airbag state machine. The failures refinement now holds.

Having a formal model of the various interacting state machines would also allow us to better ensure that the requirements of the design are being met by the overall model. This would bestow a formal sense of requirements traceability.
V. RELATED WORK

Various types of UML diagrams have been given a formal behavioural semantics in CSP. For example, Ng et al. [5] translated state diagrams, while activity diagrams were translated by Dong et al. [6].

The application of formal methods to SysML in particular is extremely topical: Graves et al. [7] focused on the integration of SysML with type theory, within the context of aerospace engineering; CML [8] is a formal modelling language specifically targeted at the specification and design of systems of systems.


In recent component-based verification work, Carillo et al. [4] defined an approach using interface automata to check whether a proposed decomposition of an abstract block into concrete sub-blocks is valid.

VI. CONCLUSION

In this paper we have presented an approach to formalising and verifying the decomposition of communicating SysML blocks, in a refinement-based setting, utilising the process algebra CSP, along with its refinement checker FDR. In addition, we have illustrated the concepts using a small, but realistic, case study. Using FDR, we identified several inconsistencies between the abstract block and the proposed concrete decompositions.

The benefit of our approach is that there is already existing tool support available to assist in the refinement process (with FDR being a prime example). The automated translation from SysML diagrams to CSP is practically achievable through the use of a model-driven engineering framework.

Planned future work includes the development of a model-based framework to automate the process. The focus of the framework will be primarily the consistency checking between diagrams of the same kind, termed intra-diagram consistency; however, consistency checking between different diagram kinds, termed inter-diagram consistency, will also be a concern.

Ultimately, the fusion of formal and semi-formal methods — where the modelling activity is largely undertaken in the semi-formal domain, but with some underlying recourse to formal methods — will instil more confidence in the validity of the resulting design.

REFERENCES