

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 Σ .

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

$$P := \text{Stop} \mid a \rightarrow P \mid b \& P \mid P_1 \square P_2 \mid P_1 \sqcap P_2 \mid P \setminus H \mid P_1 \parallel I \parallel P_2 \mid P_1 [I_{P_1} \parallel I_{P_2}] P_2$$

The *alphabet* of a CSP process P , denoted αP , 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>

$P_1 \square P_2$ models the *deterministic choice* between processes P_1 and P_2 , where the choice is resolved by the environment. Conversely $P_1 \sqcap P_2$ represents *non-deterministic choice* where the choice is resolved internally by the process.

The hiding operator, \backslash , internalises or hides a set of events H from the alphabet of the process P .

Generalised parallel composition, of the form $P_1 \llbracket I \rrbracket P_2$, requires synchronisation on the events of I . *Alphabetised parallel composition*, where we specify a synchronisation interface $I_P \subseteq \alpha P$ for a process P , as in $P_1 \llbracket I_{P_1} \parallel I_{P_2} \rrbracket P_2$, requires synchronisation on the events of $I_{P_1} \cap I_{P_2}$.

It is often useful to consider a trace of the events which a process can communicate: the set of all such possible finite paths of events which a process P can take is written $traces \llbracket P \rrbracket$. However, it is well understood that traces on their own are not enough to fully describe the behaviour of a process. For a broader description of process behaviour, we might consider what a process can refuse to do: the refusals set of a process — the set of events which it can initially choose not to communicate — is given by $refusals \llbracket P \rrbracket$. By comparing the refusals set with the traces of a process, we can see which events the process *may* perform: the *failures* of a process P — written $failures \llbracket P \rrbracket$ — are the pairs of the form (t, X) such that, for all $t \in traces \llbracket P \rrbracket$, $X = refusals \llbracket P/t \rrbracket$ (where P/t represents the process P after the trace t).

The FDR tool enables us to compare processes in terms of refinement. We write $P_1 \sqsubseteq_{\mathcal{M}} P_2$ when P_2 refines P_1 under the model \mathcal{M} . If we were only to consider traces, then

$$P_1 \sqsubseteq_T P_2 \Leftrightarrow traces \llbracket P_2 \rrbracket \subseteq traces \llbracket P_1 \rrbracket$$

Similarly, we can define failures refinement as

$$\begin{aligned} P_1 \sqsubseteq_F P_2 \Leftrightarrow & \\ & traces \llbracket P_2 \rrbracket \subseteq traces \llbracket P_1 \rrbracket \\ & \wedge \\ & failures \llbracket P_2 \rrbracket \subseteq failures \llbracket P_1 \rrbracket \end{aligned}$$

B. Systems Modelling Language

The diagrams currently present in SysML can broadly be classified as: those that enable the modeller to describe behavioural aspects of a design (*use case*, *activity*, *sequence* and *state machine*); and those that allow for the specification of structure (*package*, *block definition*, *internal block* and *parametric*). In addition, the *requirement* diagram is used to capture requirements that a design must adhere to.

The block definition diagram allows us to model the composition of a block: this can be an abstract block that is composed of concrete sub-blocks; alternatively, this can be a concrete block composed of other concrete blocks.

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 effect² $effect^M : T^M \rightarrow E^M$.

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_j^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

$$\begin{aligned} M(q) \hat{=} & \\ & (q = q_j^M) \& trigger^M(t_1) \rightarrow effect^M(t_1) \rightarrow \\ & \quad M(target^M(t_1)) \\ & \square \\ & (q = q_j^M) \& trigger^M(t_2) \rightarrow M(target^M(t_2)) \\ & \vdots \end{aligned}$$

²In this paper we assume that every effect is also a trigger.

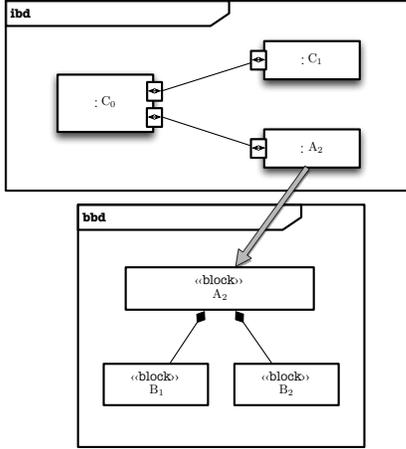


Fig. 1. The decomposition of an abstract block into concrete sub-blocks.

The CSP process is initialised to the initial state of the corresponding state machine by appropriately setting the state variable. The final state has no outgoing transitions and is modelled by the process *Stop*.

B. Proposed Approach

Assume that the SysML model is comprised of a set Ω of N communicating blocks, say $\Omega = \{C_0 \dots C_{N-2}\} \cup \{A_{N-1}\}$, where the blocks $C_0 \dots C_{N-2}$ are concrete, and A_{N-1} is abstract. Provided A_{N-1} is decomposed into K concrete sub-blocks, $B_0 \dots B_{K-1}$, the problem description can be stated thus: is the suggested decomposition valid, such that $B_0 \dots B_{K-1}$ can be substituted for A_{N-1} in Ω ?

A top-down systems engineering approach allows for an abstract block, modelling behaviour at a higher level, to be refined and decomposed into concrete realisations that are closer to the implementation. However, the behavioural aspects of the abstract block needs to be preserved if we are to replace it with the concrete sub-blocks: the behaviour of the composition should not be able to perform a sequence of events not permitted by the higher level abstraction (block A_{N-1}). Clearly, we are excluding events introduced at the lower level (blocks $B_0 \dots B_{K-1}$) from this observation.

Assuming we have a CSP process (with an appropriate synchronisation set) for each sub-block $B_0 \dots B_{K-1}$, we can form the composition using the alphabetised parallel operator. We define the process *CONCRETE* as

$$CONCRETE \triangleq \parallel_{j=0}^{K-1} [I_{B_j}]B_j$$

If we consider all the possible sequences of events that A_{N-1} communicate as valid, then as long as the process *CONCRETE* does not communicate a sequence outside of these valid sequences, we can use *CONCRETE* in the place

of A_{N-1} . The more refined process, *CONCRETE*, is *at least as good as* A_{N-1} .

Stated formally, we require

$$A_{N-1} \sqsubseteq_{\mathcal{M}} CONCRETE$$

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 AccelerationSensor, a SeatbeltLock, and an AirbagControl block. The AccelerationSensor and SeatbeltLock are concrete; the AirbagControl block is abstract. We propose to decompose AirbagControl into concrete Airbag and OccupantDetectionSensor sub-blocks, in order to perform its function. This decomposition is natural, as the an 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 airbag (in case of a flawed detection).

The state machine diagrams for the AirbagControl, Airbag and OccupantDetectionSensor blocks are shown in Figures 2, 3, and 4, respectively.

Figure 5 shows the state machine diagram for the 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 *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 *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.

$$\begin{aligned} Airbag(q) &\triangleq \\ &(q = enabled) \ \& \ activate \rightarrow Airbag(deployed) \\ &\square \\ &(q = enabled) \ \& \ disable \rightarrow Airbag(disabled) \\ &\square \\ &(q = disabled) \ \& \ enable \rightarrow Airbag(enabled) \\ &\square \end{aligned}$$

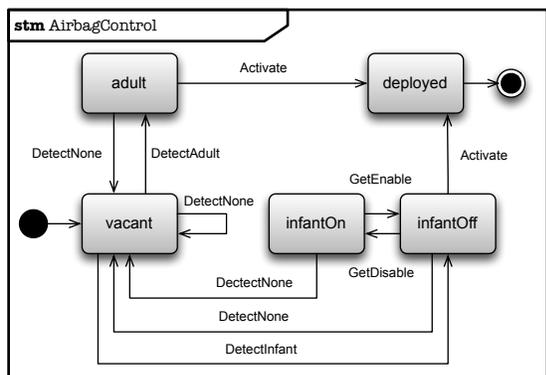


Fig. 2. The state machine for the abstract AirbagControl block.

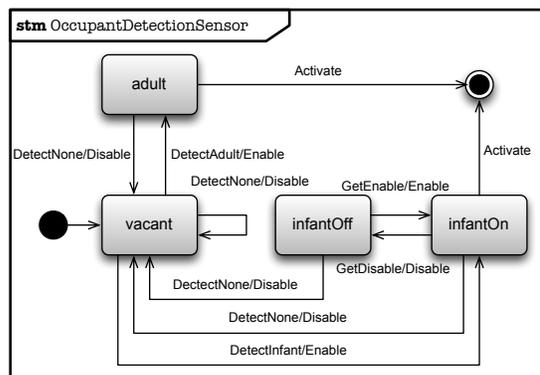


Fig. 4. The state machine for the OccupantDetectionSensor block.

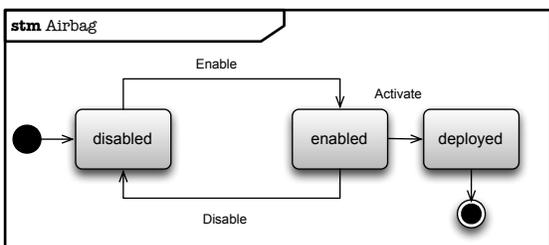


Fig. 3. The state machine for the Airbag block.

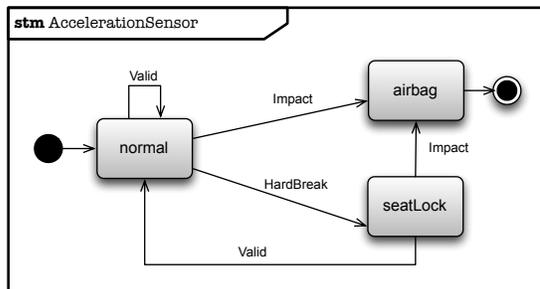


Fig. 5. The state machine for the AccelerationSensor block.

$$(q = \text{deployed}) \ \& \ \text{Stop}$$

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.

$$\begin{aligned}
 \text{AirbagControl}(q) \hat{=} & \\
 & (q = \text{adult}) \ \& \ \text{detectNone} \rightarrow \text{AirbagControl}(\text{vacant}) \\
 & \square \\
 & (q = \text{adult}) \ \& \ \text{activate} \rightarrow \text{AirbagControl}(\text{deployed}) \\
 & \vdots \\
 & (q = \text{deployed}) \ \& \ \text{Stop}
 \end{aligned}$$

The processes of our case study are initialised in accordance with their initial state:

$$\begin{aligned}
 \text{AIRBAGCONTROL} \hat{=} & \text{AirbagControl}(\text{vacant}) \\
 \text{AIRBAG} \hat{=} & \text{Airbag}(\text{disabled}) \\
 \text{ODS} \hat{=} & \text{OccupantDetectionSensor}(\text{vacant})
 \end{aligned}$$

We combine the sub-block processes (using *generalised* parallel composition to aid readability) and define *CONCRETE* thus:

$$\text{CONCRETE} \hat{=} \text{AIRBAG} \parallel X \parallel \text{ODS} \setminus (\Sigma \setminus Y)$$

The set of events are given by:

$$\begin{aligned}
 X &= \{\text{enable}, \text{disable}, \text{activate}\} \\
 Y &= \{\text{detectNone}, \text{detectInfant}, \text{detectAdult}, \\
 &\quad \text{getEnable}, \text{getDisable}, \text{activate}\}
 \end{aligned}$$

In the above, we hide all events but those of $\alpha\text{AirbagControl}$. Using FDR and the hiding operator, we test if the decomposition is valid:

$$\text{AIRBAGCONTROL} \sqsubseteq_{\mathcal{M}} \text{CONCRETE}$$

Depending on the model of refinement:

- if $\mathcal{M} = T$, then the refinement holds.
- if $\mathcal{M} = F$, 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 *Disable* transition-to-self in the *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] focussed 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.

Moisan *et al.* [9] introduced an approach based on the formalisation and subsequent verification of component protocols using the NuSMV [10] model checker.

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

- [1] Object Management Group, *Systems Modelling Language Specification, version 1.3*, 2012.
- [2] C. A. R. Hoare, *Communicating Sequential Processes*. Prentice Hall, 1985.
- [3] Department of Computer Science, University of Oxford & Formal Systems Europe, *Failures Divergence Refinement User Manual, version 2.94*, 2012.
- [4] O. Carrillo, S. Chouali, and H. Mountassir, “Formalizing and verifying compatibility and consistency of SysML blocks,” *ACM SIGSOFT Software Engineering Notes*, vol. 37, no. 4, pp. 1–8, 2012.
- [5] M. Y. Ng and M. Butler, “Towards formalizing UML state diagrams in CSP,” in *Proceedings of the 1st International Conference on Software Engineering and Formal Methods (SEFM 2003)*, pp. 138–147, IEEE, 2003.
- [6] X. Dong, N. Philbert, L. Zongtian, and L. Wei, “Towards formalizing UML activity diagrams in CSP,” in *Proceedings of the International Symposium on Computer Science and Computational Technology (ISCST 2008)*, pp. 450–453, IEEE, 2008.
- [7] H. Graves and Y. Bijan, “Using formal methods with SysML in aerospace design and engineering,” *Annals of Mathematics and Artificial Intelligence*, vol. 63, no. 1, pp. 53–102, 2011.
- [8] J. C. P. Woodcock, A. L. C. Cavalcanti, J. S. Fitzgerald, P. G. Larsen, A. Miyazawa, and S. Perry, “Features of CML: a formal modelling language for systems of systems,” in *Proceedings of the 7th International Conference on System of System Engineering (SOSE 2012)*, IEEE, 2012.
- [9] S. Moisan, A. Ressouche, and J.-P. Rigault, “Behavioural substitutability in component frameworks: a formal approach,” in *Proceedings of the Workshop on Specification and Verification of Component-Based Systems (SAVCBS 2003)*, pp. 22–28, ACM, 2003.
- [10] A. Cimatti, E. Clarke, F. Giunchiglia, and M. Roveri, “NuSMV: a new symbolic model checker,” in *Proceedings of the 11th International Conference on Computer Aided Verification (CAV 1999)*, vol. 1633 of *Lecture Notes in Computer Science*, pp. 495–499, Springer, 1999.