A Process Algebraic Approach to Decomposition of Communicating SysML Blocks

Jaco Jacobs and Andrew Simpson

***Abstract*—The block concept is a fundamental modeling construct in the Systems Modeling Language (SysML), a visual modeling 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 behavior 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 modeling 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 modeling notation suited to specify and design systems comprised of various components.

Modeling 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], 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 behavior without a detailed knowledge of the underlying mathematics of refinement.

Manuscript received January 13, 2013; revised March 27, 2013.

Jaco Jacobs is with the Department of Computer Science, University of Oxford (e-mail: jaco.jacobs@cs.ox.ac.uk).

Andrew Simpson was with the Software Engineering at the University of Oxfor.

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 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:

$$P := Stop \mid a \longrightarrow P \mid b \,\&\, P \mid P_1 \sim P_2 \mid P_1 \mid\sim\mid P_2 \mid P \setminus H \mid P_1 \,[\mid I \mid]\, P_2 \mid P_1 \,[\, I_{P_1} \mid\mid I_{P_2} \,]\, P_2$$

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, →, 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.

$P1 \sim P2$ models the deterministic choice between processes $P1$ and $P2$, where the choice is resolved by the environment. Conversely $P1 \mid\sim\mid P2$ represents non-deterministic choice where the choice is resolved internally by the process. The hiding operator, internalises or hides a set of events H from the alphabet of the process P.

Generalised parallel composition, of the form $P1 \,[\mid I \mid]\, P2$, requires synchronisation on the events of I. Alphabetised parallel composition, where we specify a synchronisation interface $IP \subseteq \alpha P$ for a process P, as in $P1 \,[IP1\ I\ IP2]\, P2$, requires synchronisation on the events of $IP1 \cap IP2$.

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* [P]. However, it is well understood that traces on their own are not enough to fully describe the behavior of a process. For a broader description of process behavior, we might

DOI: 10.7763/IJMO.2013.V3.257

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* [*P*]. 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* [*P*] — are the pairs of the form (*t*, *X*) such that, for all $t \in traces\ [P]$, $X = refusals\ [P/t]$ (where *P/t* represents the process *P* after the trace *t*).

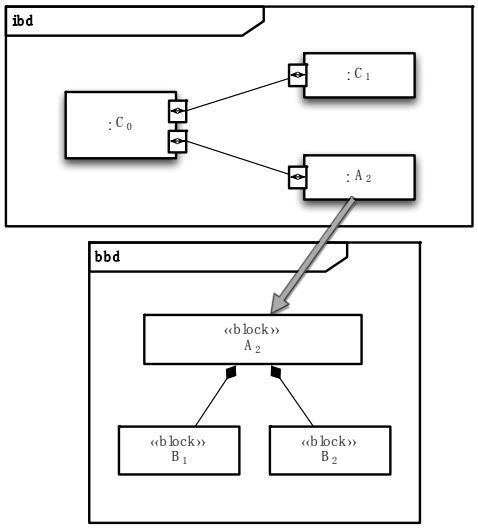

Fig. 1. The decomposition of an abstract block into concrete sub-blocks.

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

$$P_1 \sqsubseteq_T P_2 \Leftrightarrow traces\ [P_2\ ] \subseteq traces\ [P_1\ ]$$

Similarly, we can define failures refinement as

$$P_1 \sqsubseteq_F P_2 \Leftrightarrow$$
$$traces\ [P_2\ ] \subseteq traces\ [P_1\ ]$$
$$\cup$$
$$failures\ [P_2\ ] \subseteq failures\ [P_1\ ]$$

B. Systems Modeling Language

The diagrams currently present can broadly be classified as: those that enable the modeler to describe behavioral 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 modeling the behavioral aspects (and interactions) of blocks using state machines. Each *block* has associated classifier behavior, 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.

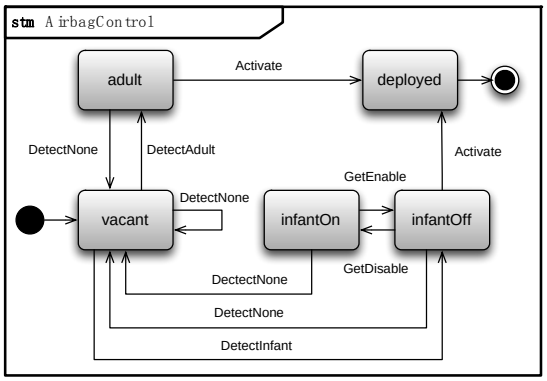

Fig. 2. The state machine for the abstract Airbag Control block.

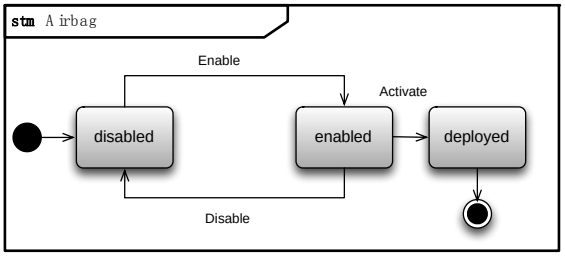

Fig. 3. The state machine for the Airbag block.

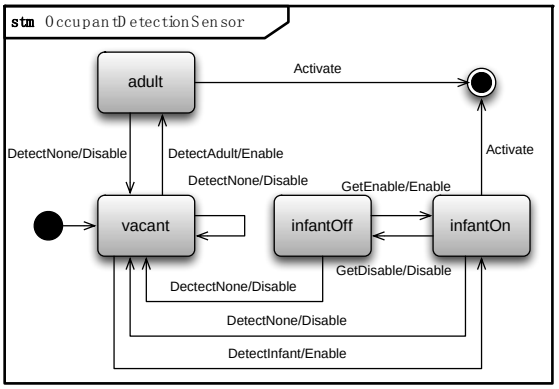

Fig. 4. The state machine for the Occupant Detection Sensor block.

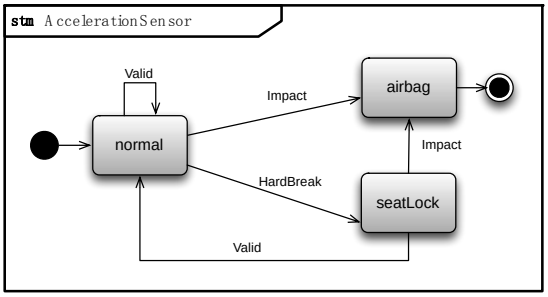

Fig. 5. The state machine for the Acceleration Sensor block.

A. Modeling State Machines

A state machine M is 5 tuples (Q^M, $q0^M$, f^M, T^M, E^M) which consists of :

- the finite set of states, Q^M ;
- the initial state, $q0^M \in Q^M$;
- 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, $source^M : T^M \longrightarrow Q^M$;
- the target state, $target^M : T^M \longrightarrow Q^M$
- the triggering event $trigger^M : T^M \longrightarrow E^M$; and
- the effect[1] $effect^M : T^M \longrightarrow E^M$.

For example, *effect* (t) returns the behavior 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* (t) = θ for a transition t.

Intuitively, the formalisation maps every block's classifier behavior — 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 $q^M \in Q^M \setminus \{q0^M, f^M\}$ 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^M, and we have two transitions $\{t_1, t_2\}$ emanating from that state, with $effect^M(t_2) = \theta$. We can then model the behaviour using the following process

$M(q)=$

$$(q=q^M) \,\&\, trigger^M(t_1) \longrightarrow effect^M(t_1)$$
$$\longrightarrow M(target^M(t_1))$$
$$[]$$
$$(q=q^M) \,\&\, trigger^M(t_2) \longrightarrow M(target^M(t_2))$$
$$\ldots$$

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

B. Proposed Approach

Assume that the SysML model is comprised of a set Ω of N communicating blocks, say $\Omega = \{C_0 .. C_{N-2}\} \cup \{A_{N-1}\}$ where the blocks $C_0 \ldots C_{N-2}$ are concrete, and A_{N-1} is abstract. Provided A_{N-1} is decomposed into K concrete

Sub-blocks, $B_0 . . B_{K-1}$, the problem description can be stated thus: is the suggested decomposition valid, such that $B_0 . . B_{K-1}$ can be substituted for A_{N-1} in Ω?

A top-down systems engineering approach allows for an abstract block, modeling behavior at a higher level, to be refined and decomposed into concrete realisations that are closer to the implementation. However, the behavioral aspects of the abstract block need to be preserved if we are to replace it with the concrete sub-blocks: the behavior 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 $B_0 . . B_{K-1}$) from this observation.

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

$$CONCRETE = \quad ||^{K-1}_{j=0} [I_{B_j}]B_j$$

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 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

$$A_{N-1} \sqsubseteq_M CONCRETE$$

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 Acceleration Sensor, a Seat belt Lock, and an Airbag Control block. The Acceleration Sensor and Seatbelt Lock are concrete; the Airbag Control block is abstract. We propose to decompose Airbag Control into concrete Airbag and Occupant Detection Sensor sub-blocks, in order to perform its function. This decomposition is natural, as 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 Airbag Control, Airbag and Occupant Detection Sensor blocks are shown in Fig. 2, 3, and 4, respectively.

Fig. 5 shows the state machine diagram for the Acceleration Sensor. 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 Airbag Control need to be a refinement if they are to replace the higher level block and function correctly in the complete

[1] In this paper we assume that every effect is also a trigger.

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.

Airbag (q) =

$(q = enabled) \,\&\, activate \rightarrow Airbag\ (deployed)$
[]
$(q = enabled) \,\&\, disable \rightarrow Airbag\ (disabled)$
[]
$(q = disabled) \,\&\, enable \rightarrow Airbag\ (enabled)$
[]
$(q = deployed) \,\&\, Stop$

The process Airbag Control models the higher level behavior of the airbag, depending on whether there is an infant, an adult, or no occupant detected in the passenger seat.

Airbag Control (q)=

$(q = adult) \,\&\, detect\ None \rightarrow Airbag\ Control\ (vacant)$
[]
$(q = adult) \,\&\, activate \rightarrow Airbag\ Control\ (deployed)$
…
$(q = deployed) \,\&\, Stop$

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

AIRBAGCONTROL = Airbag Control (*vacant*)
AIRBAG = Airbag (*disabled*)
ODS = Occupant Detection Sensor (*vacant*)

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

$CONCRETE = AIRBAG\ [|X|]\ ODS\ \backslash(\Sigma \backslash Y)$

The set of events are given by:

$X = \{enable, disable, activate\}$
$Y = \{detect\ None, detect\ Infant, detect\ Adult, get\ Enable, get\ Disable, activate\}$

In the above, we hide all events but those of α Airbag Control. Using FDR and the hiding operator, we test if the decomposition is valid:

$AIRBAGCONTROL \sqsubseteq_M CONCRETE$

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 *(detect None)*, the concrete composition refuses all events, whereas the abstract process would allow (not refuse) events from the set *{detect None, detect Infant, detect Adult}*. 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 behavioral 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 modeling 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 modeling activity is largely undertaken in the semi-formal domain, but with some underlying recourse to formal methods — will instill more confidence in the validity of the resulting design.

References

[1] *Object Management Group, Systems Modeling 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 and 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 *Proc. 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 *Proc. the International Symposium on Computer Science and Computational Technology (ISCSCT 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 modeling language for systems of systems," in *Proc. 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 *Proc. 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 *Proc. the 11th International Conference on Computer Aided Verification (CAV 1999)*, vol. 1633 of *Lecture Notes in Computer Science*, pp. 495–499, Springer, 1999.

Jaco Jacobs gained an honours degree in Engineering Sciences from the University of Stellenbosch, South Africa. After a few years in industry he enrolled on the part-time Software Engineering Programme at the Department of Computer Science, University of Oxford and gained an MSc with distinction in 2010. He is now pursuing his DPhil working with Andrew Simpson. He is interested the application of formal methods to graphical modeling notations.

Andrew Simpson gained a first class honours degree in Computer Science from the University of Wales, Swansea. Later, he received an MSc and a DPhil from the University of Oxford. He is currently a University Lecturer in Software Engineering at the University of Oxford, teaching on the Software Engineering Programme. His research interests include, amongst others, the application of formal description techniques (Z, CSP, Alloy) to the modeling and analysis of critical systems.