Mastering Heterogeneous Behavioural Models

J. Christian Attiogbé(✉)

LS2N - UMR CNRS 6004 - University of Nantes, Nantes, France
Christian.Attiogbe@univ-nantes.fr

Abstract. Heterogeneity is one important feature of complex systems, leading to the complexity of their construction and analysis. Moving the heterogeneity at model level helps in mastering the difficulty of composing heterogeneous models which constitute a large system. We propose a method made of an algebra and structure morphisms to deal with the interaction of behavioural models, provided that they are compatible. We prove that heterogeneous models can interact in a safe way, and therefore complex heterogeneous systems can be built and analysed incrementally. The Uppaal tool is targeted for experimentations.

AQ1

Keywords: Behavioural models · Heterogeneous systems · Interaction

1 Introduction

Mastering the composition of heterogeneous models contributes to settle the challenge of building and analyzing large systems. Models are used at different abstraction levels and for different purposes. Data models capture the structure of manipulated data, behavioural models often based on transition systems or event systems help to predict and reason about the behaviour of the software to be built. Other models such as timed models, security models, functional models are required according to the needs. A combination of these models is often necessary. In this article we focus on behavioural models used to capture the evolution and the interaction between processes of a more general heterogeneous system which can combine various components. An example of an heterogeneous system is an assembly of pieces of software and hardware communicating with a distributed architecture (using smart objects, sensors, actuators, mechanical parts driven by software). This kind of systems is spreading more and more. However mastering their design, proving their correctness and maintaining these systems are a challenge of first importance for the security of services and software, and especially for reducing time to market of smart objects.

There are several works and proposals related to heterogeneous issues; they embrace different abstraction levels and adopt various policies. There are a body of work around Interface Theories [7]. SysML [6] adresses system engineering at modeling levels. SystemC [4] adopts a rather low abstraction level by composing software or hardware modules which are classes containing processes modelling functionalities. The core of SystemC consists of an event-driven simulator working as a scheduler. The Ptolemy project [5,9] proposes one of the most advanced

Y. Ouhammou et al. (Eds.): MEDI 2017, LNCS 10563, pp. 1–9, 2017.
DOI: 10.1007/978-3-319-66854-3_22

framework, Ptolemy II [17] with which we share some concerns. But these are general purpose and heavy weight approaches which, from our point of view, constraint a lot the used components; they build a kind of a scheduler of the whole composition of components. Unlike approaches with strong coupling of formalisms inside a main one, we target a specific framework with a weak coupling of components described with different formalisms. We address systems with evolving adhoc structures, for small aperture net of components. The components can the be composed (plugged) or unplugged at any time.

This work is motivated by the necessity of light methods and tools to face the construction and the analysis of heterogeneous systems. The difficulties of heterogeneity arise not only at language level (data, property or behavioural), but also at the semantic level.

We propose a method supported by a tool (aZiZa), to make it easier the composition and the interaction between heterogeneous behavioural models. The main idea is that one can easily compose models described with different formalisms but having the same compatibility domain. Currently we focus on behavioural models.

The article is organized as follows. Section 2 introduces the materials we have used. Section 3 is devoted to the proposed method, an algebra to structure the composition of models. Section 4 reports on experimentations and evaluation supported by the developed tool. Section 5 concludes the article.

2 Materials: Models Compatibility and Composition

Heterogeneity is concerned with description formalisms and semantic models; but we focus on semantic models which we consider as *compatibility domains*. Several categories of compatibility domains can be considered, for instance labelled transition systems, event-based models, predicate transformer *á la Dijkstra*.

Compatible Models. Two models M_1 and M_2 (or more) are said *compatible* or not, with regard to at least three compatibility levels: syntactic compatibility, semantic compatibility and formal-reasoning compatibility. Transition Systems [2], Mealy Machines [16], with their various extensions, are widely used to handle complex dynamic systems and are at the heart of many methods analysis and verification tools. The underlying theories are well studied and, in the current state of the art a lot of effective systems are *compiled* into labelled transition systems (LTS). Process Algebra (such as CCS [14], CSP [15], LOTOS [12], π-calculus [13]) built on top of transition systems are recognized as powerful behavioural description models; they are also representative of many behavioural languages, hence their use as composition and interoperability basis.

Definition 1. *(Compatibility Domain) A Compatibility Domain is defined as a category of models characteristics in such a way that, any two models considered within this domain, are comparable w.r.t. the considered characteristics. It is a model integration basis.*

Examples of *semantic compatibility domains* are: logics, labelled transition systems, trace semantics, temporal logics, weakest preconditions, Kripke model.

Proposition 1. *Within a compatibility domain, it is always possible to translate objects semantics (from one formalism and paradigm) into the domain, to compose or integrate them, to reason within the basis, and to possibly translate results in target formalisms.*

Semantic Models and Semantic Embedding. A direct application of the notion of compatibility is the construction of semantic bridges between models or the semantic embedding of one model into another one. We choose the LTS as a reference behavioural model, because it is widely used and equipped with various tools.

If two models M_1 and M_2 are in a compatible semantic domain (LTS in our case), it exists structure morphisms[1] ζ_1 and ζ_2 with related meaning matching such that $\zeta_1(M_1) = LTS_1$ and $\zeta_2(M_2) = LTS_2$. Accordingly, we are about to define some operators Φ which arguments are different but compatible behavioural models; these operators form an algebra that leads the interaction of behavioural models. The idea is that $\Phi(M_i, M_j)$ is semantically unfolded as $\phi(\zeta_i(M_j), \zeta_j(M_j))$ where ϕ is the domain-compatible equivalent of Φ.

If we consider a behavioural model M as a term of a given algebra $\mathcal{A}$, a sketch of the semantic embedding of models is as follows: we consider $\mathcal{A}_i$ as the source algebra to describe various but compatible models, then it exists a compatible domain denoted here by $(\mathcal{S}, L, \rightarrow)$, a LTS with the set of state, the set of labels and the transition relation.

It follows that when models are compatible, a semantic bridge can be used to relate them via the semantic structure induced by the compatibility domain (for instance their LTS). Consequently, a multilevel bridge can also be gradually built between compatibility domains to link two or more models.

3 Interaction of Hererogeneous Models: An Algebra

Interaction between behavioural models, whatever their description formalisms, is viewed as exchanges through common communicating channels. Typically the interaction is denoted by a flow of emission and reception statements. Process algebra models, as a compatibility domain, capture very well these interactions, where the unit of specifications is a *process* expressing an elementary sequential behaviour; more complex behaviours are expressed with the composition (sequential, parallel, etc.) of other processes, elementary or not.

Handling the heterogeneity is as simpler as if the LTS is the known user manual of each component. On the one hand, we extract the LTS from given components to compose them; on the other hand the LTS can be given by the

[1] like a function on elements, a structure morphism relates mathematical objects or structures.

component providers. Besides, an implementation can be built from a LTS used to tune a composition. We define a set of operators that impact the behaviour of composition:

– a process composition can be restructured through the renaming of channels;
– process communications can be broken through the modification of channels;
– the structure of a complete net of processes can evolve through channel restructuring, etc.

3.1 The Core Operators for Model Interaction

We are about to elaborate an algebra $\mathcal{A}$ to structure and analyse the composition of heterogeneous processes. The operators of the algebra are related to the two levels (Φ level and ϕ level) considered in Sect. 2, while the structure set of the algebra is the set of processes $\mathcal{P}$ to be composed. Our target is an algebra $\mathcal{A} = \langle \mathcal{P}, O_\Phi, O_\phi \rangle$; therefore we introduce these sets of operators. In the following, a *process* is denoted by the term:

$$\mathsf{Process\ procName\ [channel\ parameters](other\ parameters)}\ \{\mathsf{body}\}$$

where we consider its name, its channels and parameters, and a body. The body is an LTS which describes the behaviour of the process. Several named instances of a process can be defined using the process name as a type.

A *system* is made with the composition of at least two processes.

compose: ***Abstract Parallel Composition of Processes.*** Let P_1 and P_2 be two processes which use a shared communication channel nc.

$$\mathsf{Process\ Proc[nc]()\ P1;} \qquad \mathsf{Process\ Proc[nc]()\ P2.}$$

The expression $S = \mathsf{compose}(P_1, P_2)$ is the system made of the parallel composition of the processes P_1 and P_2 which interact via their common nc channel. Note that a channel can be hidden in a process by the renaming of the channel. The arity of the compose operator is not a strong constraint; a set of n processes can be composed either with the binary composition

$$\mathsf{compose}(\ldots(\mathsf{compose}(\mathsf{compose}(P_1, P_2), P_3), \cdots), P_n)$$

or directly with the list of processes as arguments: $\mathsf{compose}(P_1, P_2, \cdots, P_n)$.

The compose operator is an instance of the Φ operator. Typically, the embedding functions ζ_i compute the transition systems from the processes used as arguments of compose; then ϕ is the synchronous product [15] of the resulting transition systems. A component process of a system built by the compose operator may be **selected** with the projection operator denoted by $\upharpoonright$. Consequently an operation α can be applied to a process inside a composition by selecting it as follows: $\alpha(\mathsf{compose}(P_1, P_2, \cdots, P_n) \upharpoonright P_3)$.

rename: ***Renaming a Channel in a Process.*** The expression (P rename c as nc) denotes a process P where the channel c is renamed as nc.

Let P_3 be a process using nc as a channel: Process Proc[nc]() P3. The expression $S = \mathsf{compose}(P_1, P_2\ \mathsf{rename}\, nc\, \mathsf{as}\, c, P_3)$ results in a system where only P_1

and P_3 interact through nc. The behaviour of P_2 does not impact the behaviour of S since P_2 uses a local channel, thus the behaviour of P_2 is ignored in S.

replace: ***Substitution of Processes.*** Within a system, a given process is substituted by a given new one. The replace operator needs three arguments: a system S, a process oP already in S, a new process nP not in S. The process oP should share its channels with S. The process nP should have the same shared channels (for the substitution) but it can have more channels. The effect of the replacement is based on the shared channels; the shared channel oP is cut and replaced by the common channel in nP. The expression $sys = \mathsf{replace}(Sys, oP, nP)$ modifies Sys by replacing inside it, the behaviour of oP by the new behaviour expressed by nP.

Formally the channels shared by Sys and oP are renamed in oP with a new name unused in Sys and nP. Then Sys is composed with nP. Consequently if nc is the channel shared by the three processes, c a fresh channel, then we have:

$$\mathsf{replace}(Sys, oldP, newP) = \mathsf{compose}(Sys, (Sys \uparrow oP)\ \mathsf{rename}\ \mathrm{nc}\ \mathsf{as}\ \mathrm{c}, \mathrm{nP})$$

remove: ***Removing a Process from a Composition.*** A given process can be removed from a system. The remove operator (symbolically denoted by $\downarrow$) requires two arguments: a system S made at least with two processes, a process P already part of S. The process P will be removed from S; this is symbolically denoted by $S \downarrow P$. For instance the expression $sys = \mathsf{remove}(\mathsf{compose}(P_1, P_2, P_3), P_2)$ results in a system composed of the processes P_1 and P_3.

extractChan: ***Listing the Channels of a Process.*** This operator, when applied to a process, gives the list of channels used inside the process. The channels of processes can then be compared, reused, renamed, hidden.

These operators make our target algebra and practically a core language: $\langle \mathcal{P}, \{\mathsf{compose}, \mathsf{replace}, \mathsf{select}, \mathsf{remove}\}, \{\mathsf{rename}, \mathsf{extractChan}\}\rangle$. It is enough expressive, to describe the composition and the interaction between behavioural models as illustrated in the next section.

Semantics of Interaction. As far as the interaction between the behavioural models is considered, each process evolves according to the channels it uses. Interaction is based on communication via shared channels using emission and reception mechanisms (message passing). Synchronous channels involve handshake communications. A reception takes place when a process applies the appropriate reception primitive relatively to a channel and, there is a (abstract) data sent on the addressed channel by the emission primitive applied by another process. In the case of asynchronous channel, if there is nothing on the channel, the attempt of reception is aborted.

3.2 Illustration: A Heterogeneous Control System

We consider the interaction between a net of processes modelling a control system equipped with sensors, actuators and controlers, from various vendors.

Let a process controller C_1 interacting by reading a channel ic and writing on a control channel cc. Let a process Sensor S_1 interacting by sending data on the same channel ic. At the modelling level, we could simply write $\mathsf{compose}(C_1, S_1)$ so that C_1 and S_1 interact via ic. Considering A_1 as the actuator process interacting by reading the channel cc, then the description $sys = \mathsf{compose}(\mathsf{compose}(C_1, S_1), A_1)$ builds a new system where the three processes interact together through the channels ic and cc. The controler C_1 may send orders to the actuator A_1 depending on data read from S_1. Now, we define two actuator processes A_2 and A_3 and a hub of actuators HA which sends its data to A_1, A_2 and A_3:

```
Actuator A2; Actuator A3; HA = compose((A1 rename cc as sc), A2, A3)
```

The behaviour expressed by $\mathsf{replace}(sys, A_1, (HA\,\mathsf{rename}\,sc\,\mathsf{as}\,cc))$ results in a new system where the controler C_1 is not anymore directly connected to A_1 but to HA via the channel cc, a renaming of the previous sc channel of HA. In the same way we can easily add new sensors S_2, S_3 into an existing pool of sensors with the compose operator: $\mathsf{compose}(sys, S_2, S_3)$. The three sensors will write on the channel ic.

3.3 Extending the Core Operators

Consider that we have a system made of sensors, actuators, controlers and many other smart devices, making an adhoc network of communicating processes. We would like to plug a new device in the system; for instance a new plugged sensor detects the existing controlers and sends data to them, or a new plugged actuator joins the system and becomes ready to interact with the existing processes which send orders to the actuators. The system builder may evaluate the forthcoming system, decide if some components or operations are correct before performing them on the real system. This is profitable if the used models and operations on models are trustworthy. Consequently we would like to easily check the consistency of the new composition of processes prior to implementing it. For instance,

$$\mathsf{check}(compose(sys, newSensor))$$
$$\mathsf{check}(replace(sys, oldProcess, newProcess))$$

These scenario motivate the need to define the check operator which is not a process composition operator but an analysis one. Typically this kind of operators should implement at least the interaction compatibility, the absence of deadlock, liveness property. In the current stage of the work we reuse for this purpose, the existing tools of process algebra: UPPAAL [3] which has its own graphical input description formalism, SPIN [10] which has the Promela language as input process description language and CADP [8] which uses Lotos as input process description language.

4 Experimentations and Evaluation

We used the proposed operators to experiment with case studies, see for instance[2] for a detailed version of a case study of a distributed control system where various processes cooperate to control objects evolving within a given area.

The system consists of a set of robots which supervise a geographically widespread area and take actions with respect to events in the area: intruders or found unusual objects. Sub-components of the system are responsible of patrolling in different parts of the area and looking for preassigned objects; in case of detection of such objects a signal is sent to a supervisor. Other sub-components follow a specific object or a detected intruder and communicate its location to the supervisor. Several compatible processes initially described using DOT, Uppaal, promela were composed using our algebra. Several ζ_i morphisms were used to embed the described process models into LTS. The DOT formalism have been intensively used. The LTS are then embeded into Uppaal. For the current experiments the Uppaal tool have been used for the composition and the formal analysis of the composed system. We have been able to perform deadlock analysis and state reachability on the composed system.

Tool Support. To experiment with examples, we have developed the main modules of a prototype tool called aZiZa (see footnote 2), to support our method proposal. This is necessary to experiment and validate the proposed concepts and to improve the global composition method.

5 Conclusion

We have shown that under the hypothesis of semantic domain compatibility, the composition of heterogeneous behavioural models can be overcomed. We have used labelled transition systems as common semantics domain to found our method of composition. The method is based on an algebra of operators that focus on the manipulation of channels as the communication mechanism between the composed models. We have equipped the method with a tool in order to experiment with case studies that serve as a mean of assessment of the proposal.

One representative of the related works is Ptolemy II [11,17]. Ptolemy achieves the interaction between different actor-oriented models using an abstract semantics (namely the actor semantics). It also enables the use of finite state machines in place of actor-oriented models, but the interaction works rather as a global scheduler, controling a sequential execution flow of the FSM considered each as a global state linked via a port to another one. Moreover Ptolemy II is a general purpose heavy weight composition framework. Unlikely we target a specific, flexible and extensible framework dedicated to the composition and analysis of behavioural models dedicated to the growing small aperture nets of processes.

[2] http://aziza.ls2n.fr.

Yet we have considered some experiments where components deal locally with time constraints but dealing with time constraints at the global level is a challenge. As future work, we have planned experimentations with the CADP framework and especially its `exp.open` composition tool. We plan some improvements among which the propagation of global properties inside local components and vice versa. For this purpose we are investigating the Property Specification Language [1], an IEEE standard, as a pivotal for property passing through the components and through the various tools.

References

1. IEC 62531 Ed. 1 (2007–2011) (IEEE Std 1850–2005): Standard for Property Specification Language (PSL). IEC 62531:2007 (E), pp. 1–152, December 2007
2. Arnold, A.: Verification and comparison of transition systems. In: Gaudel, M.-C., Jouannaud, J.-P. (eds.) CAAP 1993. LNCS, vol. 668, pp. 121–135. Springer, Heidelberg (1993). doi:10.1007/3-540-56610-4_60
3. Behrmann, G., David, A., Larsen, K.G.: A tutorial on Uppaal. In: Bernardo, M., Corradini, F. (eds.) SFM-RT 2004. LNCS, vol. 3185, pp. 200–236. Springer, Heidelberg (2004). doi:10.1007/978-3-540-30080-9_7
4. Black, D.C., Donovan, J., Keist, A., Bunton, B. (eds.): SystemC: From the Ground Up, 2nd edn. Springer, Heidelberg (2010)
5. Eker, J., Janneck, J.W., Lee, E.A., Liu, J., Liu, X., Ludvig, J., Neuendorffer, S., Sachs, S., Xiong, Y.: Taming heterogeneity - the ptolemy approach. Proc. IEEE **91**(1), 127–144 (2003)
6. Friedenthal, S., Moore, A., Steiner, R.: A Practical Guide to SysML. The MK/OMG Press, Morgan Kaufmann, Boston (2015)
7. de Alfaro, L., Henzinger, T.A.: Interface theories for component-based design. In: Henzinger, T.A., Kirsch, C.M. (eds.) EMSOFT 2001. LNCS, vol. 2211, pp. 148–165. Springer, Heidelberg (2001). doi:10.1007/3-540-45449-7_11
8. Garavel, H., Lang, F., Mateescu, R., Serwe, W.: CADP 2011: a toolbox for the construction and analysis of distributed processes. STTT **15**(2), 89–107 (2013)
9. Goderis, A., Brooks, C.X., Altintas, I., Lee, E.A., Goble, C.A.: Heterogeneous composition of models of computation. Future Gener. Comp. Syst. **25**(5), 552–560 (2009)
10. Holzmann, G.J.: The spin model checker. IEEE Trans. Softw. Eng. **23**(5), 279–295 (1997)
11. Lee, E.A.: Disciplined heterogeneous modeling. In: Petriu, D.C., Rouquette, N., Haugen, Ø. (eds.) MODELS 2010. LNCS, vol. 6395, pp. 273–287. Springer, Heidelberg (2010). doi:10.1007/978-3-642-16129-2_20
12. LOTOS: a formal description technique based on the temporal ordering of observational behaviour. International Standard 8807, IOS - OSI, Geneva (1988)
13. Milner, R., Parrow, J., Walker, D.: A calculus of mobile processes. J. Inf. Comput. **100**, 1–40 (1992) AQ2
14. Milner, R.: Communication and Concurrency. Prentice-Hall, Upper Saddle River (1989)

15. Roscoe, A.W., Davies, J.: CSP (communicating sequential processes). In: Padua, D.A. (ed.) Encyclopedia of Parallel Computing. Springer, Heidelberg (2011). doi:10.1007/978-0-387-09766-4_186
16. Roth, C.H., Kinney, L.L.: Fundamentals of Logic Design. Thomson, Luton (2004)
17. Tripakis, S., Stergiou, C., Shaver, C., Lee, E.A.: A modular formal semantics for ptolemy. Math. Struct. Comput. Sci. **23**(4), 834–881 (2013)