

Part-Whole Statecharts, or How I Learned to Turn the Explicit Attitude of Composition Languages on My Side

Luca Pazzi

University of Modena and Reggio Emilia
Dipartimento di Ingegneria dell'Informazione
Strada Vignolese 905, Modena, Italy
email: pazzi@unimo.it

Abstract. Composition languages and paradigms may be framed into two categories, depending on whether the presence of explicit compound entities is enforced by the paradigm or not. This provides the basis for a general modeling principle, that can be enounced, rather informally, by stating that “*the explicit way of enforcing composition is better than the implicit one*”. The existence of the *explicit vs. implicit* modeling attitude of composition languages and paradigms as well as the soundness of the explicit modeling principle constitute the author’s main position statements at the WCL’02 workshop. The paper introduces first both the approaches, then presents the application of the related principle to the state-based modelling language Statecharts, which can be classified as an implicit composition language. A new composition language designed with the explicit principle in mind, called Part-Whole Statecharts, is presented in the second part of the paper. The software modeling advantages of Part-Whole Statecharts show the benefits of the explicit modeling approach and constitute thus an empirical confirmation of the soundness of the explicit approach.

1 Introduction

Most of my research work in the past years has been focused on the problem of composition languages and paradigms. I started working on a general *ontological* principle [2], which hypothesized that any *composition paradigm* (ranging from the object-oriented paradigm to process algebras like CCS and CSP) may be seen as being committed towards either an *implicit* or an *explicit* composition attitude. In the latter case the composition paradigm provides adequate tools for modeling explicitly the *associative knowledge* present in the domain. By associative knowledge I mean both static (**John**

is-husband-of Mary) as well as dynamic (Rescue12 tows Car1234) relationships among entities. The explicit composition principle can then be enounced by stating that *horizontal* (i.e., *peer to peer*) relationships in the domain call for some amount of structural knowledge, in the form of explicit entities — i.e., having the same dignity of John, Mary, Rescue12 or Car1234 — to be (explicitly) inserted in the domain. Such additional entities are bound to the original ones by part-whole relationships, which can be thought of as being *vertical* in contrast to the original, horizontal, associative relationships seen before. A language is committed towards the explicit approach if it provides both constructs and criteria for “transforming” associative knowledge into new entities, i.e. into *vertical relationships*. I will not discuss here directly the way a modelling language may be committed towards one of such modelling tendencies¹. I will sustain instead three related position statements:

1. there exists both an *explicit* and an *implicit* way to model *associative knowledge*: the modelling paradigms range over a spectrum between the two extremes;
2. the explicit modeling approach transforms *horizontal* (peer-to-peer) associative knowledge into *vertical* (part-to-whole) associative knowledge;
3. the explicit way of modeling associative knowledge brings benefits in terms of reusability, understandability and manutenability of the software abstraction (“*explicit is beautiful!*”);

Consider, for example, that the associative knowledge: “John is the husband of Mary” may be modeled by either the implicit approach, in which mutual references are embedded in the respective objects, or by the explicit approach, where an explicit entity (say **Marriage**, or **Family**) holds part-of relationships to John, in the role of husband, and Mary, in the role of wife.

The main advantage of the explicit approach consists essentially in achieving better overall **understandability**, since we make clear, at the conceptual level that a marriage entity, with its own attributes, exists in the domain given the mutual relationship between John and

¹ The reader may refer to [4] for a thorough discussion about the explicit modeling tendency of the Entity-Relationship model of data.

Mary. Another advantage regards the **extensibility** (hence in some sense the *manutenability*) of the whole design, since any children of the couple may be easily added to the family record in the role of “son” or “daughter”.

The advantages of the explicit composition point of view are however more evident in the dynamic modeling case. In the next Section I will present Part-Whole Statecharts, a composition language designed with the explicit composition approach in mind [3][5].

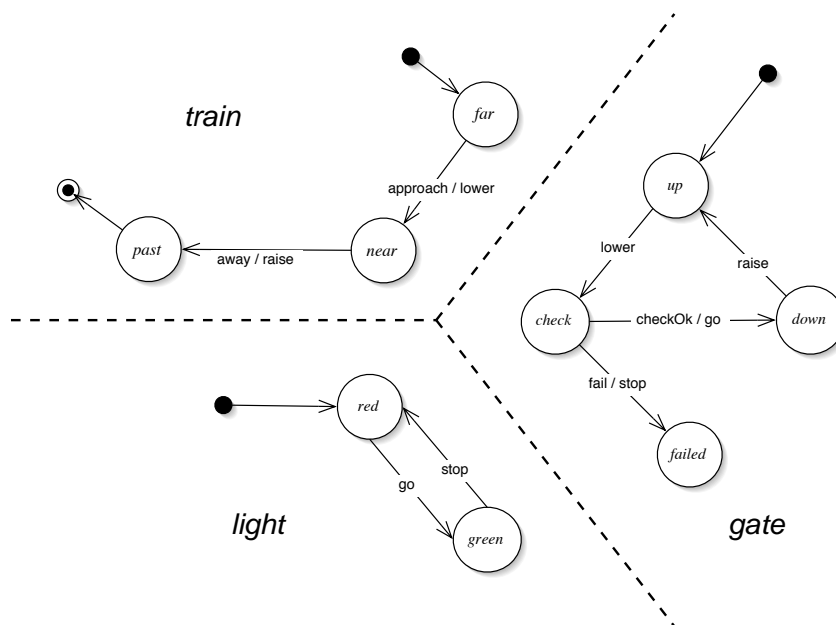


Fig. 1. The implicit composition of three Statecharts by AND (parallel) composition. The system is found each time in a state from each of the three behaviors.

2 Part-Whole Statecharts

In this Section I will give a brief overview of Part-Whole Statecharts by first presenting an example by means of ordinary Statecharts [1]. In this way, I will show the difference between the implicit way of composing behavioral entities and the explicit one. Consider a train approaching a gate: the train may be far, near or past: as soon as the

train is detected (say by an *approach* event emitted by the railway sensors) the gate has to be closed. In case the gate fails to close, a red light stops the train before it reaches the gate: if everything goes well, the gate is raised after an *away* position has been generated.

Traditional Statecharts compose by *horizontal peer to peer* message passing. Observe Figure 1: as long as the situation evolves, event messages are exchanged among the the three state machines, which evolve correspondingly: messages are free to travel (i.e., they are *broadcast*) from any entity to any other entity. It is indeed message broadcast (Figure 2) that realizes (horizontal) composition.

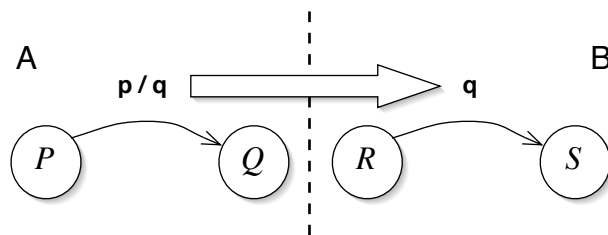


Fig. 2. Horizontal (part-part) Statecharts composition by event broadcast propagation.

We may observe different problems which are related to this way of achieving composition: in first place the behavioral abstractions are not *self-contained*, that is they need to host references to the other entities participating in the composition. This is primarily due to the need of specifying *action events* within the state transitions, that is events to be propagated when the transition is taken. In the example, the train abstraction is bound to the gate abstraction, which in turn is bound to the light abstraction. This implicit binding makes the single abstractions less reusable, since it is clear that they have been designed with the final composition in mind: this makes difficult their reuse in different contexts. It can be observed, finally, that *the whole behavior of the system is scattered*, that is we have to follow message passing from one abstraction to the another in order to grasp the whole behavior. This makes difficult, among other disadvantages, any further extension of the functionalities of the system.

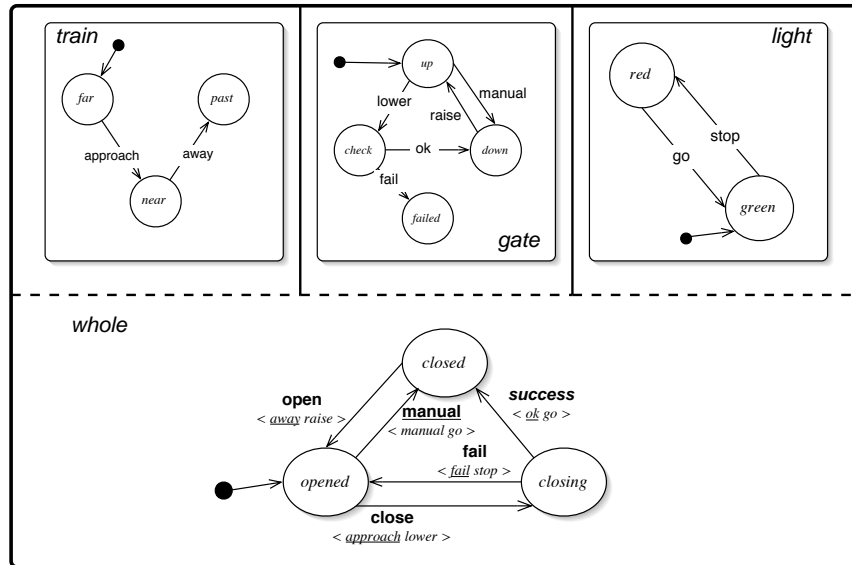


Fig. 3. The explicit composition of the three behaviors by a PW Statechart.

Part-Whole Statecharts (also referred to as either PW Statecharts or PWS) distinguish instead between a *part section*, which hosts the component behaviors and a *whole section*, where the entire behavior of the device is represented. It can be observed that the formalism does not allow message exchange among components: this is graphically suggested by solid lines which separate the components from each other. In this way self-containment is *enforced* at the component level: behavioral abstractions do not need to be designed with references to each other. A *dashed* line separates instead the component section from the whole section of a PWS: that suggests that the formalism allows message exchange between the components and the whole automaton, which represents indeed the composition semantics.

The railways gate example has been reformulated by a PWS in Figure 3: the upper part of the PWS hosts the three component behaviors which now lack, as observed, any reference to each other. The state transitions in the whole automaton are labeled by *compound events* having the form: $\mathbf{e} \langle e_1 e_2 \dots e_m \rangle$ where e_1, e_2, \dots, e_m are named *component events*, since they belong to the event alpha-

bets of the components, and **e** is called the *high level event*, since it must belong to the event alphabet of the whole and as such is exported to the outside. Each compound event specifies *exactly one* trigger, which may be either the high level event or one event belonging to the component list. In the example, the compound event **close** $\langle \underline{approach} \ lower \rangle$ denotes that the high level event **close** is composed by the *approach* and *lower* event from, respectively, the event alphabets of the train and the gate automaton. Trigger events are distinguished by underlining them.

For a denotational semantics of PWS the reader may refer to [5]: we give here a first account of the operational semantics, which is still under development. Observe Figure 4: as soon as a state transition is taken by a component, the event which labels the state transition is directed towards the whole. If such an event appears in the whole as a trigger for some state transition, then such state transition is taken and the remaining events in the event list are directed towards the respective components. In the example, suppose an *approach* event is generated by the train. Since *approach* appears underlined in the **close** state transition in the whole, such transition fires and the *lower* event is directed towards the gate.

A final remarks regards *extendibility* and *reusability* of the whole itself. Suppose that the bar of gate fails to lower: in that case a **fail** event is propagated to other more external contexts. Suppose for example that a railway control center monitors different gates along the railway line. We may think of such a control center as consisting of another whole automaton, having a state transition of the form, say **emergency** $\langle \underline{fail} \ manual \rangle$ In such case, a *manual* event is sent back to the signaling gate, and both a **manual** raise and a **go** signal are directed towards the gate and the light, thus moving the whole to the **closed** state. In this case the event propagation follows the schema reported in Figure 5. We may observe again that an explicit composition is easily extendible and reusable compared to the implicit case.

3 Conclusions

This paper reports my position regarding two ways of achieving composition which strongly impact on the overall modeling quality, by

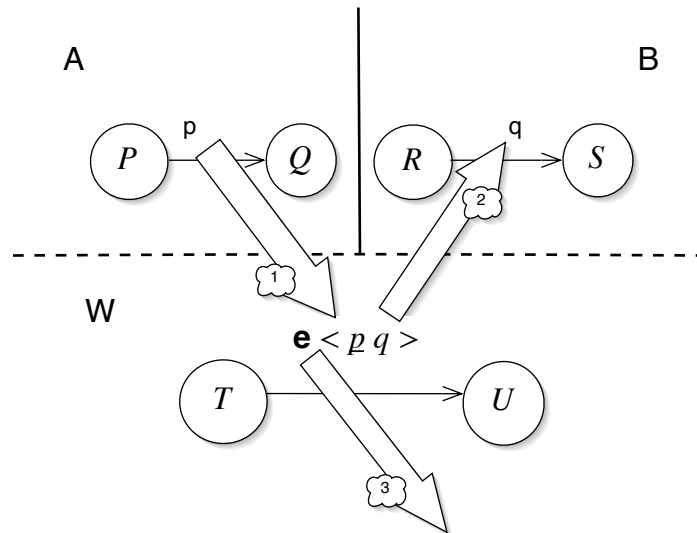


Fig. 4. A first sketch of the PWS operational semantics: part to whole propagation. Event p is emitted by state machine A and triggers the state transitions in the whole (1). Event q is directed towards the state machine B (2), and finally event e is directed towards any contexts of which it is part.

either allowing to exploit or not the associative knowledge which binds different parts into complex wholes. The two approaches characterize any modeling paradigm: I have shown, for example, how the state modeling paradigm can be committed to both, with different results. Being aware of the differences helps in building modeling paradigms which are more expressive and suitable to produce self-contained as well as highly reusable modeling artifacts.

References

1. D. Harel. Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, 8:231–274, 1987.
2. L. Pazzi. Dynamically-based complex objects: From process synchronization to entity aggregation. In *Proceedings of ECAI-94 Workshop on Parts and Wholes: Conceptual Part-Whole Relationships and Formal Mereology*, 1994.
3. L. Pazzi. Extending statecharts for representing parts and wholes. In *Proceedings of the EuroMicro-97 Conference, Budapest, Hungary*, 1997.
4. L. Pazzi. Implicit versus explicit characterization of complex entities and events. *Data & Knowledge Engineering*, 31:115–134, 1999.
5. Luca Pazzi. Part-whole statecharts for the explicit representation of compound behaviors. In Andy Evans, Stuart Kent, and Bran Selic, editors, *UML 2000 - The*

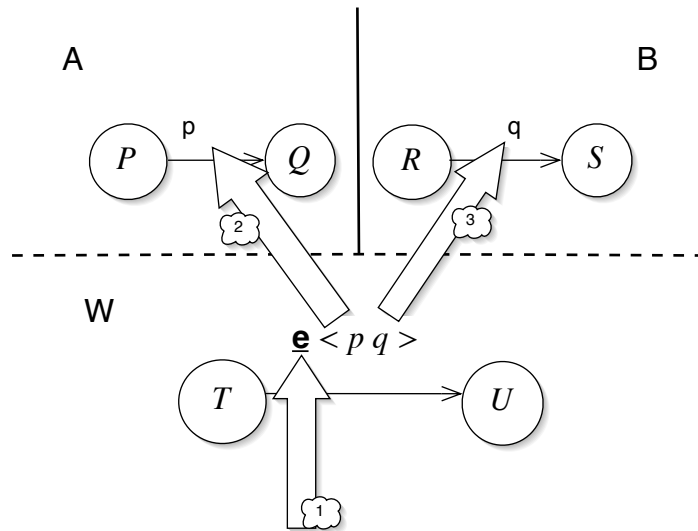


Fig. 5. A first sketch of the PWS operational semantics: whole to parts propagation. Event e is directed towards the whole by one of the contexts of which it is part (1). Subsequently, both the component events p and q are directed towards the respective state machines (2 and 3).

Unified Modeling Language. Advancing the Standard. Third International Conference, York, UK, October 2000, Proceedings, volume 1939 of LNCS, pages 541–555. Springer, 2000.