Awareness and Control in Adaptable Transition Systems*

Roberto Bruni[1], Andrea Corradini[1], Fabio Gadducci[1],
Alberto Lluch Lafuente[2], and Andrea Vandin[3]

[1] Department of Informatics, University of Pisa, IT
[2] DTU Compute, Technical University of Denmark, DK
[3] Electronics and Computer Science, University of Southampton, UK

The CoDa approach. Self-adaptive systems are advocated as a solution to the problem of mastering the complexity of modern software systems and the continuous evolution of the environment where they operate. Self-adaptation is considered a fundamental feature of autonomic systems, one that can specialise to several other self-* properties, like self-configuration and self-optimisation.

Should the analysis favour a *black-box* perspective, a software system is called "self-adaptive" if it can modify its behaviour in response to a change in its context. On the contrary, *white-box* adaptation focuses on how adaptation is realised in terms of architectural and linguistic mechanisms and usually promotes a clear separation of adaptation and application logics. Our own approach [2, 5] characterizes adaptivity on the basis of a precisely identified collection of *control data* (CoDa), deemed to be interpreted as those data whose manipulation triggers an adaptation. This view is agnostic with respect to the form of interaction with the environment, the level of context-awareness, the use of reflection for self-awareness. In fact, our definition applies equally well to most of the existing approaches for designing adaptive systems. Overall, it provides a satisfactory answer to the question "what is adaptation *conceptually*?".

But "what is adaptation *formally*?" and "which is the right way to reason about adaptation, *formally*?". We are aware of only a few works (e.g. [8]) that address the foundational aspects of adaptive systems, including their semantics and the use of formal reasoning methods, and often only generic analysis techniques are applied. An example of the possibilities of such technique is our approach [4] to adaptive self-assembly strategies using Maude (and following precisely both [8] and [2]), where we applied standard simulation and statistical model checking.

Adaptable Transition Systems. Building on the intuitions briefly discussed above and on some foundational models of component based systems (like *I/O automata* [7] and *interface automata* [1]), we proposed a simple formal model based on a new class of transition systems [3], and we sketched how this definition can be used to specify properties related to the adaptive behaviour of a system. A central role is again played by control data, as well as by the interaction among components and with the enviroment (not addressed explicitly in [2]).

* Research partially supported by the MIUR PRIN 2010LHT4KM CINA.

Let us recall that the steps of I/O and interface automata are labeled over three disjoint sets of actions, namely *input*, *output* and *internal* actions. The composition of two automata is defined only if certain disjointness constraints over the sets of actions are satisfied, and it is obtained conceptually as a synchronous composition on shared actions and asynchronous on the others, the differences between the two models not being relevant at this level of abstraction.

Adaptable Transition Systems (ATSs) combine these features on actions within an extended Kripke frame presentation, in order to capture the essence of adaptativity. An ATS is a tuple $\mathcal{A} = \langle S, A, T, \Phi, l, \Phi^c \rangle$ where S are the states, $A = \langle I, O, H \rangle$ is a triple of three disjoint sets of input, output and internal actions, and $T \subseteq S \times A \times S$ is a transition relation, where by A here we denote the union $I \uplus O \uplus H$. Furthermore, Φ is a set of atomic propositions, and $l : S \to 2^{\Phi}$ is a labeling function mapping states to sets of propositions. Finally, $\Phi^c \subseteq \Phi$ is a subset of *control propositions*, which play the role of the control data [2].

A transition $s \xrightarrow{a} s' \in T$ is called an *adaptation* if it changes the control data, i.e., if there exists a $\phi \in \Phi^c$ such that $\phi \in l(s) \iff \phi \notin l(s')$. Otherwise, it is called a *basic* transition. An action $a \in A$ is called a *control action* if it labels at least one adaptation, and the set of all control actions is denoted by C.

The relationship between the action set C and the alphabets I, O and H is arbitrary in general, but it could satisfy some pretty obvious constraints for specific classes of systems. For example, an ATS $\mathcal{A}$ is *self-adaptive* if $C \cap I = \emptyset$, i.e., if all adaptations are under the control of the system. If instead $C \subseteq I$ the system is *adaptable*; intuitively, adaptations cannot be executed locally but should be triggered by an external manager. Hybrid situations are possible as well, when a system has both input and local control actions.

The composition operations on I/O automata can be extended seamlessly to ATSs. They have been exploited to model the composition of an adaptable basic component $\mathcal{A}_B$ and an adaptation manager $\mathcal{A}_M$ that realizes the adaptation logics, for example a control loop in the style of the MAPE-K architecture [6]. In this case, natural well-formedness constraints could be expressed as relations among sets of actions. For example, the manager controls *completely* the adaptivity features of the basic component if $C_B \subseteq O_M$; and if the manager itself is at least partly adaptable (i.e., $C_M \cap I_M \neq \emptyset$), a natural requirement to avoid circularities would be that $O_B \cap C_M = \emptyset$, i.e. that the basic component cannot govern the adaptivity of the manager. Composition of ATSs will also be used to model different kinds of aggregation of adaptive systems, like *ensembles* and *swarms*.

Summing up the talk. ATSs are a concrete instance of a methodological approach to white-box adaptation for software systems. More precisely, the CoDa approach we sketched in the first section provides the designer with a criterion to specify where adaptation is located and, as a consequence, which parts of a system have to be adapted. It assumes the possibility to inspect, to some extent, the internal structure of a system, and requires to identify a set of control data, which can be changed to adapt the component's behaviour. Adaptation is the run-time modification of such data.

As described in the second section, ATSs extend interface automata by equipping them with a set of control propositions evaluated on states, which represent the formal counterpart of control data. As for control data, the choice of control propositions is arbitrary but it imposes a clear separation between the ordinary, functional behaviours and the adaptive ones. Control propositions can then be exploited in the specification and analysis of adaptive systems, formally recovering various notions proposed in the literature, such as adaptability, feedback control loops, and control synthesis.

The talk presents ATSs and some applications, and it introduces an explicit representation of *awareness data*, ideally intended as those "sensor" data that are exploited at the control level in order to possibly enforce an adaptation. Awareness and control data complement each other in answering the question regarding *where* and *when* adaptation takes places: A clear identification of awareness data helps selecting which artifacts indicate that it may be necessary to perform an adaptation, and precisely stating when that may occur.

References

1. de Alfaro, L., Henzinger, T.A.: Interface automata. In: ESEC/SIGSOFT FSE 2001. ACM SIGSOFT Software Engineering Notes, vol. 26(5), pp. 109–120. ACM (2001)
2. Bruni, R., Corradini, A., Gadducci, F., Lluch-Lafuente, A., Vandin, A.: A conceptual framework for adaptation. In: de Lara, J., Zisman, A. (eds.) FASE. LNCS, vol. 7212, pp. 240–254. Springer (2012)
3. Bruni, R., Corradini, A., Gadducci, F., Lluch-Lafuente, A., Vandin, A.: Adaptable transition systems. In: Martí-Oliet, N., Palomino, M. (eds.) WADT 2012. LNCS, vol. 7841, pp. 95–110. Springer (2013)
4. Bruni, R., Corradini, A., Gadducci, F., Lluch-Lafuente, A., Vandin, A.: Modelling and analyzing adaptive self-assembly strategies with maude. Science of Computer Programming 99, 75–94 (2015)
5. Bruni, R., Corradini, A., Gadducci, F., Lluch-Lafuente, A., Vandin, A.: A white box perspective on behavioural adaptation. In: Nicola, R.D., Hennicker, R. (eds.) Software, Services, and Systems. LNCS, vol. 8950, pp. 552–581. Springer (2015)
6. Horn, P.: Autonomic Computing: IBM's perspective on the State of Information Technology (2001)
7. Lynch, N.A., Tuttle, M.R.: Hierarchical correctness proofs for distributed algorithms. In: PODC 1987. pp. 137–151. ACM (1987)
8. Meseguer, J., Talcott, C.L.: Semantic models for distributed object reflection. In: Magnusson, B. (ed.) ECOOP. LNCS, vol. 2374, pp. 1–36. Springer (2002)