Team automata : a formal approach to the modeling of collaboration between system components

Beek, M.H. ter

Author: Beek, Maurice H. ter
Title: Team automata : a formal approach to the modeling of collaboration between system components
Issue Date: 2003-12-10

1. Introduction

This thesis studies formal aspects of *team automata*, a mathematical framework introduced in [Ell97] to model components of *groupware systems* and their interconnections. In particular, this thesis focuses on the flexibility team automata offer when modeling collaboration between system components.

We begin this Introduction by providing some background. Subsequently we introduce the model in an informal way, after which we discuss its main features in the context of several related models. Finally, we finish this Introduction with an overview of the contents of this thesis.

Background

A set of interacting, interrelated, or interdependent components forming a complex whole is what we mean by the frequently used, but seldom defined notion of a *system*. The human body and computers are thus examples of a system. A system is *distributed* if it consists of separate components but nevertheless appears to its users as a single coherent system. It does not have a single locus of control, but its components collaborate by way of interactions. The internet is one of the best known distributed systems.

A system is *reactive* if, in order for it to function, it has a continuous need to interact with its environment. Its functioning thus depends on the functioning of its environment. This contrasts with a system that is *transformational*, in which case its functioning (output) is merely a function of its input. Examples of reactive systems include computer operating systems and coffee vending machines, whereas a compiler is an example of a transformational system.

Computer Supported Cooperative Work

As the presence of computer-based systems in daily-life work situations continues to increase, the understanding of how people work together and ways in which computer technology can assist, has become more and more important.

This has resulted in the emergence of *Computer Supported Cooperative Work* (CSCW for short) as an inherently multi-disciplinary field of research (see, e.g., [Gru94]). By the nature of the field, part of the computer technology consists of multi-user software and hardware, called *groupware*.

Groupware systems are systems intended to support groups of people working together in collaborative projects. Such systems are often distributed and reactive, and conceived as consisting of components cooperating in a coordinated way. This leads to complex interactive behavior and, consequently, coordination policies and their effect on behavior are key issues within CSCW. At a conceptual level CSCW needs a precise, consistent, and unambiguous terminology, while at a lower, architectural level CSCW has been searching for a rigorous mathematical framework to specify and verify groupware systems.

Formal Methods

Mathematical techniques tailored for the specification and verification of systems are known as *formal methods* (see, e.g., [CW96]). This field of research cuts across many areas of computer science and comes with an impressive body of literature. A brief comparison of the main features of team automata with some of the best-known formalisms in this field follows later on in this Introduction, while a more detailed comparison with two such formalisms can be found in Chapter 7.

The model of *Input/Output automata* (I/O automata for short) was introduced in [Tut87] for the specification and verification of distributed reactive systems (see also, e.g., [LT89] and [Lyn96]). I/O automata served as the theoretical source of inspiration for the introduction of team automata in [Ell97] through the distinction of the model's actions into input, output, and internal actions. We come back to this shortly. A conceptual source of inspiration for team automata was [Smi94], which conjectures that well-structured groups (called teams) outperform individuals in certain tasks, but at the same time calls for models capturing concepts of group behavior.

Team automata were introduced explicitly for the specification and verification of groupware systems. They were shown to be promising at both the conceptual and the architectural level of groupware systems. In this thesis we elaborate on this. Our goal is furthermore to demonstrate that the usefulness of team automata is not limited to clarifying and capturing precisely notions related to collaboration between components of groupware sytems, but extends to other kinds of (reactive) systems.

The Model

We now provide an overview of the team automata framework. We begin with a brief sketch of the overall structure of team automata and subsequently we introduce them in more detail. Analogous to the setup of this thesis, we follow an incremental presentation of team automata.

A team automaton is composed of *component automata*, which are a special type of *automata*. The crux of composing a team automaton is to define the way in which those originally independent component automata interact. Their interactions are formulated in terms of *synchronizations* of shared actions, a method for modeling collaboration among system components well known from the literature.

Automata

Automata or *labeled transition systems* are a well-known model underlying formal specifications of systems. An automaton consists of a set of states, a set of actions, a set of labeled transitions between states, and a set of initial states. Labels represent actions and a transition's label indicates the action causing the transition from one state to another.

Assume that we have an automaton modeling a coffee vending machine. Then a possible event is a user inserting a coin, which when it occurs leads to a state change of the automaton. The user forms a part of the environment of the coffee vending machine. A coffee vending machine is thus an example of a reactive system, with the insertion of coins by a user as interactions with its environment.

Next assume that also the user is modeled by an automaton, with the insertion of a coin as one of its actions. Then we have two automata, both equipped with an action modeling the insertion of a coin. When composing these two automata into one system, inserting a coin into the coffee vending machine appears as a single synchronized action. In the composed system the occurrences of an action from the automaton modeling the user and the same action from the automaton modeling the coffee vending machine are identified, i.e. simultaneously executed by the two system components. The transitions of a thus composed automaton will be synchronized occurrences of transitions of its constituting automata that have the same action label.

Synchronized Automata

A *synchronized automaton* over a set of automata is an automaton, determined by the way in which its constituting automata cooperate by means

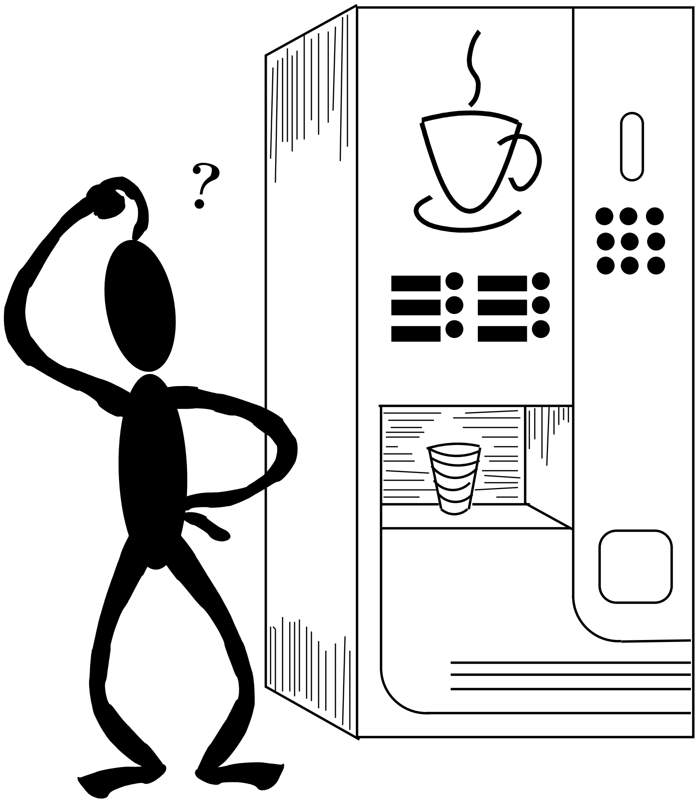

Fig. 1.1. A user in front of a coffee vending machine.

of synchronized transitions. Its (initial) states are combinations — a cartesian product — of (initial) states of its constituting automata. Its actions are the actions of its constituting automata. Its transitions, finally, are *synchronizations* of labeled transitions of its constituting automata modeling the simultaneous execution of the same single action by several (one or more) automata. The label of a transition is the action being simultaneously executed. When the synchronized automaton changes state by executing an action, all automata which participate simultaneously change state by executing that action, while all others remain idle.

An automaton does not necessarily participate in every synchronization of an action it shares. Hence there is no such thing as the unique synchronized automaton over a set of automata. Rather, a whole range of synchronized automata, distinguishable only by their transition relation, can be constructed from a given set of automata. It is this freedom to choose a transition relation that sets the team automata framework apart from most other models. Another distinguishing feature of this framework is the fact that the transitions of a synchronized automaton are labeled with one single action. We come back to this shortly.

From the way a synchronized automaton is constructed it is clear that it is itself an automaton again. Consequently, it can serve as a constituting automaton of a higher-level synchronized automaton, thus allowing hierarchical designs.

Within a synchronized automaton, three natural types of actions can be distinguished, based on the way they appear in synchronizations. Actions that are never executed simultaneously by more than one constituting automaton are *free*. Actions that are always executed as synchronizations in which all automata participate that have this action in their alphabet are called *action-indispensable*. *State-indispensable* actions, finally, require the participation of only those automata that are ready (in a suitable state) to execute that action.

Team Automata

A component automaton is an automaton in which *input*, *output*, and *internal actions* are distinguished. Input actions are not under the automaton's control, but instead are triggered by the environment including other component automata. Output and internal actions are under its control, but only the output actions are observable by other automata. Input and output actions together constitute the *external actions* and they form the interface between the automaton and its environment, whereas the internal actions are not available for interactions. This is formally achieved by requiring that the internal actions of each component automaton involved are unique to that automaton, which naturally prohibits synchronizations of internal actions with other automata.

A team automaton over a set of component automata is defined in a way similar to the definition of synchronized automata. As before, its (initial) states are cartesian products of (initial) states of its constituting component automata. Its actions are the actions of its constituting component automata, now distributed over input, output, and internal actions. All internal (output) actions of the component automata remain internal (output) actions of the team automaton. The remaining actions are those input actions of the component automata that do not occur as an output action of any of the component automata, and they become the input actions of the team automaton. Its labeled transitions, finally, are — as before — synchronizations of labeled transitions of its constituting component automata.

Like in the case of synchronized automata, we do not require all constituting component automata sharing an action to participate in every synchronization of that action. Synchronizations of internal actions never involve more than one component automaton because every internal action uniquely belongs to one particular component automaton. Moreover, independently of the states of the other component automata, an internal action can always be executed as before the composition. Like in the case of synchronized automata, there is no unique team automaton. Rather a whole range of team automata, distinguishable only by their transition relation, can be constructed.

The reason given in [Ell97] for equipping team automata — like I/O automata — with a distinction of actions into input, output, and internal actions, is the explicit desire to model different types of synchronization. This is achieved by taking the different role (input, output, or internal) that actions can have in different component automata into account. External actions may be input to some component automata and output to other component automata. In *peer-to-peer* synchronizations, actions have the same role in each of the component automata involved. In such synchronizations, all component automata are on equal footing with respect to the action being synchronized. This differs from *master-slave* synchronizations, in which input actions ("slaves") are driven by output actions ("masters"), i.e. the slaves have to follow the masters.

Team automata form a very broad and generic framework. Component automata can cooperate in many possible ways through synchronizations of shared actions. The freedom of choosing the transition relation of a team automaton moreover offers the flexibility to distinguish even the smallest nuances in the meaning of one's design. Leaving the set of transitions of a team automaton as a modeling choice thereby becomes one of the most important features of team automata. One of the topics of this thesis is a systematic study of the role of free, action-indispensable, and state-indispensable actions — and to a lesser degree peer-to-peer and master-slave synchronizations — in the modeling of collaboration between system components.

Team Automata Versus Other Models

Team automata are not an isolated model but have several features which bear a close resemblance to characteristics of other models from the literature. We now discuss three such features in general terms.

First, the set of actions of a team automaton consists of input, output, and internal actions, thus allowing the classification of a broad range of often complex synchronizations in team autamata (cf. Sections 4.4 and 5.3). This distinction of input, output, and internal actions originates from two independently developed models: I/O automata (see, e.g., [Tut87], [LT89], and [Lyn96]) and *I/O systems* (see, e.g., [Jon87] and [Jon94]). Since the semantics of an I/O system — given in terms of automata — is essentially an I/O automaton, we will speak only of I/O automata in the sequel. Team automata are, in fact, an extension of I/O automata (cf. Section 7.1).

I/O automata are not the only model in the literature in which a distinction of actions is used. The same distinction can be found in the I/O automata-based *reactive transition systems* (see, e.g., [CC02] and [CCP02]) as well as in *interacting state machines* (see, e.g., [Ohe03] and [OL02]), which were introduced specifically for modeling reactive systems. A further example is the *Calculus of Communicating Systems* (CCS for short), an algebraic specification language introduced by Milner (see, e.g., [Mil80] and [Mil89]). In CCS, the internal or *silent action* τ is a distinguished element of the set of actions. It denotes the "perfect" action of a *handshake communication*, i.e. the synchronization of two *complementary (input and output) actions*.

Secondly, the transitions of a team automaton are synchronizations of transitions with the same label. The simultaneous execution of actions from a team automaton's constituting component automata is thus limited to common actions. We call such types of synchronization *uniform* in order to distinguish them from *pluriform synchronizations* in which distinct actions can be executed simultaneously.

Also this feature of allowing solely uniform synchronizations originates from the I/O automaton model. It is by far not the only model in the literature prohibiting pluriform synchronizations. Other examples include the *mixed product* over a set of automata introduced in [Dub86] and the *product automaton* introduced in [TH98]. A further example is the theory of *path expressions*, which was introduced in [CH74], consequently encompassed in the *COncurrent SYstems* (COSY for short) notation in [LTS79], and given a *vector firing sequence* semantics in [Shi79], which considers *vector actions* rather than ordinary actions (see also [JL92]). An entry of such a vector action is not empty if and only if the respective component participates.

There are also examples of automata-based models that do allow pluriform synchronizations, such as the *free product* and the *synchronous product* over a set of automata. Both were defined in [Arn94] as the culmination of a framework of process models proposed by Nivat and Arnold in a number of papers and course notes such as, e.g., [Niv79], [AN82], and [Arn82]. Another example is the framework of *Vector Controlled Concurrent Systems* (VCCSs for short) introduced in [KKR90] (cf. Sections 7.2.3 and 7.2.4). These systems, introduced as generalizations of the COSY theory, allow pluriform synchronizations of actions of its constituting components and execute vectors of actions rather than ordinary actions. In Section 7.2.1 we will switch to vector actions in order to visualize the (potential) concurrency within team automata actions, but such vector actions will still be uniform synchronizations.

Yet another type of synchronization is the handshake communication in CCS mentioned above. Many algebraic specification languages moreover contain specific parallel composition operators that allow processes to communicate through synchronizations (see, e.g., [BPS01]). Among the best known such examples are the *(Theoretical) Communicating Sequential Processes* ((T)CSP for short) originally introduced by Hoare (see, e.g., [Hoa78], [BHR84], and [Hoa85]).

Thirdly, the transition relation of a team automaton is not uniquely determined by its constituting component automata, which also distinguishes team automata from I/O automata. This freedom of choosing the transition relation of the automaton obtained when composing a set of automata, occurs in the literature as well. An example is the aforementioned synchronous product over a set of automata. Whereas the transition relation of the free product over a set of automata is the set of all possible pluriform synchronizations, that of the synchronous product over that set of automata is the restriction of the free product to the subset of all possible pluriform synchronization vectors defined by a specifically formulated synchronization constraint. This synchronization constraint is formulated in terms of the actions only and does not depend on the current states of the automata.

Most automata-based models, however, use a single and very strict method for choosing the transition relation of an automaton composed over a set of automata, in effect resulting in composite automata that are uniquely defined by their constituents. The choice prevalent in the literature is to include, for all actions, all and only those transitions in which all automata participate that have the action in their alphabet. Since this means that all actions will be action-indispensable, we call this the *ai* principle. Examples of automata-based models with composition based on the *ai* principle include the aforementioned mixed product and product automaton over a set of automata, as well as reactive transition systems, interacting state machines, and I/O automata (cf. Section 7.1). Other examples from the literature — without claiming completeness — include *cooperating (pushdown) automata* (see, e.g., [DH94] and [HH94]) and *timed cooperating automata* (see, e.g., [LMP00]). The *ai* principle furthermore appears in disguise in non-automata-based models like (T)CSP and *statecharts* (see, e.g., [Har87]).

In Section 5.4 we define team automata that are unique with respect to particular types of synchronization. Through the formulation of predicates of synchronization we moreover provide direct constructions for such team automata. Throughout the thesis we will see, though, that of all the resulting uniquely defined team automata, it is precisely the one based on the *ai* principle that possesses the at first sight most appealing characteristics. One of the contributions of this thesis is to put some order in the "chaos" obtained when refraining from the *ai* principle. More precisely, we present an overview of some interesting characteristics that hold for certain types of team automata, among which those based on the peer-to-peer and master-slave types of synchronization. Since these types of synchronization are introduced with a clear practical motivation in mind, it is worthwhile to notice that output peer-to-peer as well as master-slave synchronizations cannot be distinguished in I/O automata (cf. Section 7.1). In fact, in a team automaton constructed according to the *ai* principle, all synchronizations are by definition master-slave.

To the best of our knowledge, no automata-based model other than team automata unites the three features discussed above. I/O automata satisfy the first two features, viz. the distinction of input, output, and internal actions, and the prohibition of pluriform synchronizations. However — as already noted in [Tut87] — the single notion of automaton composition in I/O automata is rather restrictive and may hinder a realistic modeling of certain types of interactions. This is the main motivation given in [Ell97] for introducing team automata as a generalization of I/O automata. Another important reason for generalizing I/O automata is the fact that I/O automata are *input enabling*, i.e. in every state of the automaton every input action of that automaton can be executed. Though convenient when modeling reactive computer systems, this hinders a realistic modeling of interactions that involve humans (cf. Section 7.1). Team automata have thus been introduced with the motivation of creating a single model in which the above three features are united.

Origins of the Thesis

This thesis is a monograph which is partly based on papers that were published in various places. Below we list these papers in the order in which they were written.

In [BEKR03] we elaborated further on the concept of team automata, introduced in [Ell97] for modeling groupware systems, by defining team automata in a mathematically precise way. We showed how the formal setup allows one to distinguish between several types of synchronization and to classify team automata accordingly. Based on the observation that team automata can be used as components in higher-level teams, we showed also how the framework allows for the representation of hierarchical systems.

In [HB00] we sketched how team automata can be employed to model collaboration between teams (of humans) engaged in team-based development of (software) configuration management models.

In [BEKR01b] we demonstrated the model usage and utility for capturing information security and protection structures, and critical coordinations between these structures. On the basis of a spatial access metaphor, various known access control strategies were given a rigorous formal description in terms of synchronizations in team automata.

In [BEKR01a] we presented a survey of [BEKR03] and [BEKR01b], augmented with the introduction of team automata with vectors as actions, and a preliminary comparison of team automata with I/O automata and models based on Petri nets.

In [BK03] we presented an initial investigation of the conditions under which team automata satisfy compositionality, in the sense that their behavior can be described in terms of that of their constituting component automata.

Outline of the Thesis

Although this is a theoretical thesis written for theoretical computer scientists interested in formal models with a clear practical motivation, we hope that it is also accessible for practical computer scientists well motivated to look for formalizations of models that can aid in the early design phase of complex systems. In order to achieve this we have generously accompanied our formal definitions and results by explanations and examples, providing the motivation for and the interpretation of these definitions and results.

After this Introduction, we fix most basic notation and terminology used throughout this thesis in Chapter 2. In Chapters 3 and 4 we introduce automata and synchronized automata, respectively. On top of this foundation we then build our team automata framework in Chapter 5. In Chapter 6 we study the behavior of team automata, while Chapter 7 provides a comparison with other models. Before finishing the thesis with a Discussion, we show some of the fields of application of team automata in Chapter 8. We now provide a more detailed description of each of these chapters and, where appropriate, mention the published papers used in that chapter.

In Chapter 3 we define the automata as used in this thesis and we review some notions from automata theory.

In Chapter 4 we define how to combine a set of automata in order to form a synchronized automaton. We also define how to obtain a subautomaton from a synchronized automaton as a subset of its constituting automata, and we study the relation between synchronized automata and their subautomata in terms of computations. Consequently, we show how to compose synchronized automata in an iterative way. Within synchronized automata we then characterize three basic and very natural ways of synchronizing on shared actions of their constituting automata, which form the basis of the more complex types of synchronization we introduce later. Finally, we define unique synchronized automata being maximal with respect to a given type of synchronization. Through the formulation of predicates of synchronization we moreover provide direct constructions of such synchronized automata. Some of the material in this chapter is based on [BEKR03].

In Chapter 5 we define team automata as compositions of component automata, i.e. from now on we distinguish input, output, and internal actions. To this aim we use the foundation laid in the preceding chapters and build team automata and component automata on top of (synchronized) automata. We then build subteams on top of subautomata, and we study the relation between team automata and their subteams. Also in the case of team automata, we show how to compose them in an iterative way. We then build several complex types of synchronization on top of those introduced in the previous chapter, by using the different roles that an action may have in the various component automata. Similar to synchronized automata, we define unique team automata being maximal with respect to particular types of synchronization. Through the formulation of predicates of synchronization we furthermore provide direct constructions for such team automata. Most of the material in this chapter is based on [BEKR03].

In Chapter 6 we study the computations and behavior of team automata in relation to those of their constituting component automata. Therefore we study (synchronized) shuffles and their properties. We prove that the behavior of certain types of team automata can be described in terms of certain (synchronized) shuffles of the behavior of their constituting component automata. Some of this material is based on [BK03].

In Chapter 7 we provide a comparison of team automata with two other models. The first is I/O automata, of which team automata are an extension. The second is a model based on Petri nets, for which we define team automata with vector actions as an extension of team automata. A small part of this material is based on [BEKR03], but most of it is based on [BEKR01a].

In Chapter 8 we present three examples demonstrating the usefulness of team automata in practical settings. Based on [BEKR03], we first show how to model a specific groupware architecture by team automata. Secondly, based on [HB00], we show how team automata can be employed to model collaboration between teams of developers engaged in the development of models of complex (software) systems. Thirdly, based on [BEKR01b], we show how various known access control strategies can be given a rigorous formal description in terms of synchronizations in team automata.

In the Discussion, finally, we recall the main contributions of this thesis and point out some topics worth further investigation. Furthermore, we indicate how — in theory — team automata can be used for system design and where — in practice — they have actually been used.

It is worth mentioning that at the end of this thesis one can find — in addition to the Bibliography and the Index — a List of Figures and a List of Symbols, which should allow one to quickly find the page on which a figure or symbol first appeared.