11.1 Introduction

Wireless networks define a very challenging scenario for the application programmer. Indeed, the fluidity inherent in the wireless media cannot be entirely masked at the communication layer: issues such as disconnection and a continuously changing execution context most often must be dealt with according to the application logic. Appropriate abstractions, usually provided as part of a middleware, are therefore required to support and simplify the programming task.

Coordination [549] is a programming paradigm whose goal is to separate the definition of the individual behavior of application components from the mechanics of their interaction. This goal is usually achieved by using either message passing or data sharing as a model for interaction. Publish/subscribe, described in Chap. 10 is an example of the former, where coordination occurs only through the exchange of messages (events) among publishers and subscribers. While message passing, in its pure form, is inherently stateless, data sharing enables coordination among components by manipulating the (distributed) state of the system. The tuple space abstraction, the subject of this chapter, is a typical example of a data sharing approach. The two models, publish/subscribe and tuple spaces, have sometimes crossed paths in the scientific literature: their expressive power has been compared on formal grounds in [127]; the limits of an implementation of a stateful tuple space on top of a stateless publish/subscribe layer has been investigated in [150]; some extensions of publish/subscribe with stateful features exist (e.g., the ability to query over past events as in [511]). A thorough discussion of the relationship between the two is outside the scope of this chapter, and hereafter we focus solely on tuple spaces.

Linda [310] is generally credited with bringing the tuple space abstraction to the attention of the programming community. In Linda, components communicate through a shared tuple space, a globally accessible, persistent, content-addressable data structure containing elementary data structures called tuples. Each tuple is a sequence of typed fields, as in (“foo”, 9, 27.5), containing the information being communicated. A tuple $t$ is inserted in a tuple space through an out($t$) operation, and
can be withdrawn using \texttt{in}(p). Tuples are anonymous, their selection taking place through pattern matching on the tuple content. The argument \( p \) is often called a \textit{template} or \textit{pattern}, and its fields contain either \textit{actuals} or \textit{formals}. Actuals are values; the fields of the previous tuple are all actuals, while the last two fields of \( \langle \text{"foo"}, \ ?\text{integer}, \ ?\text{float} \rangle \) are formals. Formals act like “wild cards”, and are matched against actuals when selecting a tuple from the tuple space. For instance, the template above matches the tuple defined earlier. If multiple tuples match a template, the one returned by \texttt{in} is selected non-deterministically. Tuples can also be read from the tuple space using the non-destructive \texttt{rd}(p) operation. Both \texttt{in} and \texttt{rd} are blocking, i.e., if no matching tuple is available in the tuple space the process performing the operation is suspended until a matching tuple becomes available. The asynchronous alternatives \texttt{inp} and \texttt{rdp}, called \textit{probes}, have been later introduced to allow the control flow to return immediately to the caller with an empty result when a matching tuple is not found. Moreover, some Linda variants (e.g., [717]) also provide \textit{bulk operations}, \texttt{ing} and \texttt{rdg}, used to retrieve all matching tuples in one step.

The fact that only a small set of operations is necessary to manipulate the tuple space, and therefore to enable distributed component interaction, is per se a nice characteristic of the model. However, other features are particularly useful in a wireless environment. In particular, coordination among processes in Linda is decoupled in time and space, i.e., tuples can be exchanged among producers and consumers without being simultaneously available, and without mutual knowledge of their identity or location. This decoupling is fundamental in the presence of wireless connectivity, as the parties involved in communication change frequently due to migration or fluctuating connectivity patterns.\footnote{A rather abstract treatment of coordination and mobility can be found in [714].} Moreover, tuple spaces can be straightforwardly used to represent the context perceived by the coordinating components. On the other hand, this beneficial decoupling is achieved thanks to properties of the Linda tuple space—its global accessibility to all components and its persistence—difficult to maintain in a dynamic environment with only wireless links.

In the last decade, a number of approaches were proposed that leverage the beneficial decoupling provided by tuple spaces in a wireless setting, while addressing effectively the limitations of the original Linda model. Our group was among the first to recognize and seize the potential of tuple spaces in this respect, through the \textsc{Lime} model and middleware [670]. This chapter looks back at almost a decade of efforts in the research community, by concisely describing some of the most representative systems and analyzing them along some fundamental dimensions of comparison. In doing so, it considers two main classes of applications that rely on wireless communication. First, Sect. 11.2 considers \textit{mobile networks}, where the network topology is continuously redefined by the movement of mobile hosts. Then, Sect. 11.3 considers the more recent scenario defined by \textit{wireless sensor networks} (WSNs), networks of tiny, resource-scarce wireless devices equipped with sensors and/or actuators, enabling untethered monitoring and control.