Computer Languages, Systems & Structures 35 (2009) 395 – 405

Contents lists available at ScienceDirect

Computer Languages, Systems & Structures

journal homepage: www.elsevier.com/locate/cl

Software composition with Linda

Ana M. Roldan[a,*,1], Ernesto Pimentel[b,1], Antonio Brogi[c]

[a] *Department of Computer Science, University of Huelva, Spain*
[b] *Department of Computer Science, University of Málaga, Spain*
[c] *Department of Computer Science, University of Pisa, Italy*

ARTICLE INFO

Article history:
Received 5 October 2006
Received in revised form
21 July 2008
Accepted 27 August 2008

Keywords:
Coordination languages
Components
Software architecture
Compatibility
Interaction
Process algebras

ABSTRACT

Nowadays, tuple spaces have turned out to be one of the most fundamental abstractions for coordinating communicating agents. Some models, as Linda, were presented as a set of inter-agent communication primitives which can virtually be added to any programming language. These models have the advantage of capturing both communication and synchronization in a natural and simple way. In this paper, we analyze the use of Linda to specify the interactive behavior of software components. We first introduce a process algebra for Linda and we define a notion of process compatibility that ensures the safe composition of components. This definition of compatibility takes into account the state of a global store (tuple space), which gives relevant information about the current execution of the system. Indeed, a Linda-based computation is characterized by the store's evolution, so that the set of tuples included into the store governs each computation step. In particular, we prove that compatibility implies successful computation (i.e. without deadlock). We also argue that Linda features some advantages with respect to similar proposals in the context of dynamic compatibility checking. In this context, the success of the composition of a pair of agents in presence of a suitable store can be useful to condition the acceptance of a given component into an open running system. In order to extend our approach to complex systems, where constructing a system involves more than two components, we propose the use of distributed tuple spaces as the glue to join components.

1. Introduction

Component-Based Software Engineering (CBSE) is an emerging discipline in the field of software engineering. In spite of its recent birth, a lot of activities are being devoted to CBSE both in the academic and in the industrial world. The reason of this growing interest is the need of systematically developing open system and "plug-and-play" reusable applications, which has led to the concept of "commercial off-the-shelf" (COTS) components. The first component-oriented platforms were CORBA [1] and DCE [2], developed by Open Software Foundation (OSF) and Object Management Group (OMG). Several other platforms have been developed after them, like COM/DCOM [3], CCM [4], EJB [5], and the recent.NET [6].

Available component-oriented platforms address software interoperability by using Interface Description Languages (IDLs). Traditional IDLs are employed to describe the services that a component offers, rather than the services the component needs (from other components) or the relative order in which the component methods are to be invoked. IDL interfaces highlight signature mismatches between components in the perspective of adapting or wrapping them to overcome such differences.

* Corresponding author. Tel.: +34 959217387.
E-mail addresses: amroldan@dti.uhu.es (A.M. Roldan), ernesto@lcc.uma.es (E. Pimentel), brogi@di.unipi.it (A. Brogi).
[1] The work of Ana M. Roldán and Ernesto Pimentel has been partially supported by the Project TINC2004-07943-C04-01 funded by the Spanish Ministry of Education and Science.

1477-8424/$ - see front matter © 2008 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cl.2008.08.002

However, even if all signature problems may be overcome, there is no guarantee that the components will suitably interoperate. Indeed, mismatches may also occur at the protocol level, because of the ordering of exchanged messages and of blocking conditions, that is, because of differences in the component behaviors. To overcome such a limitation, several proposals have been put forward in order to enhance component interfaces [7]. Many of them are based on process algebras, and extend interfaces with a description of their concurrent behavior [8–13], such as behavioral types or role-based representations.

The objective of this work is to explore the usability of the coordination language Linda [14] for specifying the interaction behavior of software components. Linda was originally presented as a set of inter-process communication primitives which allow processes to add, read, and delete data in a shared tuple space (store). Tuple spaces are considered as one of the most fundamental and successful abstractions for coordinating concurrent activities [14,15]. These are a number of reasons, both technical and pragmatic, to consider Linda as an appropriate alternative for specifying protocol behavior of software components. On the technical reasons, tuple spaces have the advantage of capturing both communication and synchronization in a simple and a natural way. Tuples themselves represent resources than can be communicated, shared and exchanged, without the need to use additional synchronization mechanisms. On the other hand, there are many kinds of problems that map naturally to the tuple space view of the system, namely, when there are many different sorts of concurrent agents that want to exchange information among them. In this sense, tuples represent the least common denominator of data structures and so, can be used to easily model almost any variety of stuff. Note that the use of Linda [16] can provide a unifying framework for interface definition protocols for interoperability, data exchange protocols for heterogenous databases, software components, etc. Linda's communication model features interesting properties, such as space and time uncoupling [14], as well as a great expressive power to specify concurrent and distributed systems [17].

The main contributions of this paper can be summarized as follows:

(i) We use Linda as the specification language to describe interface protocols. Syntactically, this corresponds to extending traditional IDL interfaces with a Linda description of component behaviors. The formal meaning of a Linda protocol is given by means of an extension of the process algebra presented in [18,28,29].

(ii) We define a notion of *store sensitive compatibility* to formalize the compatibility of two agents with respect to a given state of the store. The state of the store is particularly significant in Linda as it is the only means by which (all) Linda processes communicate. The store hence provides relevant information on the results of the current execution of the system, and it allows to contextualize the compatibility of agents in the perspective of dynamic compatibility checking. We show that the compatibility of two agents implies that their interaction will be successful, in the sense we will define later (Definition 1). The importance of the notion of compatibility relates to the possibility of performing *a priori* verification of complex interacting systems [19].

(iii) We present a software architecture as a collection of interconnected computational and data components. Indeed we consider software systems as compositions of specifications of their components.

The rest of the paper is organized as follows. Section 2 presents a process calculus for Linda. The use of Linda for specifying component protocols is also illustrated by means of a simple example. Section 3 is devoted to introduce the notion compatibility with respect to a store. In Section 4, we study the compatibility in a real example (*an electronic auction*). The next section shows software systems as architectural descriptions of their components and finally, some concluding remarks and future work are discussed.

2. Specifying component protocols in Linda

2.1. A Linda calculus

Linda [14] was the first coordination language [20], originally presented as a set of inter-agent communication primitives which can virtually be added to any programming language. Linda's communication primitives allow processes to add, delete, and test for the presence/absence of tuples in a shared *tuple space*. The tuple space is a multiset of data (tuples), shared by concurrently running processes. Delete and test operations are blocking and follow an associative naming scheme that operates like *select* in relational databases.

In this paper, following [18,28,29], we shall consider a process algebra $\mathscr{L}$ containing the communication primitives of Linda. These primitives permit to add a tuple (*out*), to remove a tuple (*in*), and to test the presence/absence of a tuple (*rd*, *nrd*) in the shared dataspace. The language $\mathscr{L}$ includes also the standard prefix, choice, and parallel composition operators in the style of CCS [21].

A *tuple space* is a finite multi-set of tuples, where a *tuple* is a finite sequence $\langle n_1, \ldots, n_h \rangle$ of names. The set of all tuples will be denoted by $\mathscr{T}$ and ranged over by t, u (possibly indexed). To express pattern matching, processes employ *tuple templates* inside *rd*(), *in*(), and *nrd*() operations. A tuple template is a finite sequence of names, some of which are possibly prefixed by "?" and are used as placeholders. A tuple template $\langle m_1, \ldots, m_k \rangle$ *matches* a tuple $\langle n_1, \ldots, n_h \rangle$ via a *name substitution* σ (denoted by $match(\langle m_1, \ldots, m_k \rangle, \langle n_1, \ldots, n_h \rangle, \sigma)$) iff:

(i) $k = h$, and
(ii) $\forall\, i \in [1..k]$ either $m_i = n_i$ or ($m_i = ?x_i$ and $\sigma(x_i) = n_i$).