Science of Computer Programming 133 (2017) 20–50

Contents lists available at ScienceDirect

Science of Computer Programming

www.elsevier.com/locate/scico

Checking global usage of resources handled with local policies ☆

Chiara Bodei *, Viet Dung Dinh, Gian-Luigi Ferrari

Dipartimento di Informatica, Università di Pisa, Italy

ARTICLE INFO

Article history:
Received 23 January 2014
Received in revised form 8 April 2016
Accepted 25 June 2016
Available online 4 July 2016

Keywords:
Network resources
Process calculi
Publish-subscribe systems
Formal methods
Control flow analysis

ABSTRACT

We present a methodology to reason about resource usage (acquisition, release, revision, and so on) and, in particular, to predict *bad* usage of resources. Keeping in mind the interplay between local and global information that occur in application-resource interactions, we model resources as entities with local policies and we study global properties that govern overall interactions. Formally, our model is an extension of π-calculus with primitives to manage resources. To predict possible bad usage of resources, we develop a Control Flow Analysis that computes a static over-approximation of process behaviour.

1. Introduction

We live in a world where mobility and distribution are part of the standard everyday (digital) devices, resources seem unlimited and always available. Scalable and elastic do not mean exactly this. Sooner or later people experiment that the available resources are not sufficient or that their existence can arise between their service requests and their satisfaction. One can add a 4 GB folder on her/his private data repository (e.g. Dropbox) and realises that the synchronisation with the Dropbox servers takes too long, especially for big amounts of data. The suggested solution is therefore to upload less big folders at a time, because the upload time is lower than the download one. Another one, let us say in Europe, wants to download some free software and suffers from the very slow downloading until realising that the chosen web-site is in USA and at that time most of the people have just woken up and began to surf on the web.

When designing a web-based distributed application the first focus is on the way of rendering the required functionalities into suitable tasks. This phase often abstracts away from the other side of the coin, i.e. the operational way of formalising the tasks and therefore of managing the assigned computational resources. Resource awareness, instead, should be part of the game from the very beginning, without being simply delegated to low-level supports. Standard programming metaphors consider resources as entities geographically distributed (typically available over the Internet) and with their own states, costs and access mechanisms. Furthermore, resources are not created nor destroyed by applications, but directly acquired on the fly when needed from suitable resource rental services, without investing in new infrastructures. Clearly, resource acquisition is subject to availability and requires the agreement between client requirements and service guarantees (Service Level Agreement – SLA).

☆ Research supported by the European FET Project "ASCENS", by the Italian MIUR PRIN project "Security Horizons" and by project *PRA*_2016_64 "Through the fog", funded by the University of Pisa.

* Corresponding author.
E-mail addresses: chiara@di.unipi.it (C. Bodei), dinh@di.unipi.it (V.D. Dinh), giangi@di.unipi.it (G.L. Ferrari).

http://dx.doi.org/10.1016/j.scico.2016.06.005
0167-6423/

The dynamic acquisition of resources increases the complexity of software since software capabilities strictly depends on resource availability. *Ubiquitous computing* [1] and *Cloud computing* [26,73,4] provide illustrative examples of applications where resources awareness is an essential concern. A further step towards ubiquitous computing is when software pervades the objects of our everyday life, e.g. webTV, cars, smartphones, ebook readers, etc. Often, these heterogeneous entities have a limited computational power, but are capable of connecting to the Internet, coordinating and interacting each other, in the so-called "plug&play" fashion. The real objects, as well as others of virtual nature (programs, services, etc.), which are connected in this way, form the Internet of Things (IoT) [5]. Similarly, cloud systems offer through the network a hardware and software infrastructure on which end-users can run their programs on-demand. In addition, a rich variety of dynamic resources, such as networks, servers, storage systems, applications and services are made available. The key point is that resources are "virtualised" so that they appear to their users as fully dedicated to them, and potentially unlimited.

In our programming model processes and resources are distinguished entities. Resources are computational entities with their own life cycle. They can range from computational infrastructures, storage systems and data services to special-purpose devices. Processes are instead thin entities that can dynamically acquire the required resources when available, but that cannot create any resource. This programming model abstracts some of the features of the systems discussed above. Indeed, our challenge consists in the metaphor of designing applications on top of heterogeneous sets of resources, by ensuring interoperability among them and providing transparency with respect to the actual resource implementation. As an example, let us consider a cloud system offering computing resources. The available resources are the CPU units of a given power and processes can only acquire the CPU time, when available, to run some specialised code. Similar considerations apply to storage services, where client processes can only acquire slots of the available storage. In our programming model, the deployed resources can be dynamically *reconfigured* to deal with resource upgrade, resource unavailability, security intrusion and failures. However, the reconfiguration steps that update the structure of the accessible resources are not under the control of client processes. Therefore, clients establish SLAs for having the necessary guarantees on the availability of the required resources.

This paper introduces the formal basis of our programming model. Specifically, we introduce the G-Local π-calculus, a process calculus with explicit primitives for the distributed ownership of resources, which can be either virtual or physical. In our calculus, resources are not statically granted to processes, but they are dynamically acquired on the fly, when required. We start from the π-calculus [62] and we extend it with primitives to represent resources and with the operations for acquiring and releasing resources on demand. Central to our approach is the identification of an abstract notion of resource. In our model, resources are *stateful* entities available in an active and dynamic environment, and processes continuously interact with them. A resource is described through the declaration of its interaction endpoint, its *local* state and its *global* properties. Global properties establish and enforce the SLA to be satisfied by any interaction the resource engages with its client processes. We do not address here the precise nature of these properties. The definition of the global interaction properties may involve several kinds of trade-offs. We assume that the global interaction properties can be expressed by means of a suitable resource-aware logic in the style of [11], or contract-based logic as in [30,13]. The interplay between the local operations over the state and the global SLA enforcement that occur in the process-resource interactions motivates the adjective *G-Local* given to our extension of the π-calculus.

Since we start from π-calculus, name-passing is the basic communication mechanism among processes. Beyond exchanging channel names, processes can pass resource identifiers as well. Resource acquisition is instead based on a different abstraction. To acquire the ownership of a certain resource, a process issues a suitable request. Such request is routed in the network environment to the resource. The resource is granted only if it is available. Conceptually, the process-resource interaction paradigm adheres to the *publish-subscribe* model: resources act as publishers, while processes act as subscribers. Actually, the publish-subscribe paradigm not only is a natural choice to represent distributed resources, but it also emphasises the fact that resources have to be published by external parties and therefore have to be available to everyone through appropriate requests. Notice that processes issue their requests without being aware of resource availability. When they have completed their task on the resource, acquired in an exclusive but limited usage, they release it and make it available for new requests. Furthermore, whenever the usage of the acquired resource does not respect the global SLA policy at hand, the release operation is forced. We argue that this approach relaxes the inter-dependencies among computational components thus achieving a high degree of loose coupling among processes and resources. Under this regard, our model resembles coordination models based on the notion of tuple space [40] and seems to be particularly suitable to manage distributed systems in which the set of published resources is subject to frequent changes and dynamic reconfigurations.

In summary, our approach combines the basic features of π-calculus with the distributed acquisition of stateful resources equipped with global SLA policies. This is our first contribution and also constitutes an original feature of the proposal since it covers both aspects of process-resource interactions.

A second contribution consists in the development of a *Control Flow Analysis* (CFA) for our calculus. The analysis computes a safe approximation of resource usage. Hence, it can be used to statically check whether or not the global properties of resource usage are respected by process interactions. In particular, it helps detecting possible *bad usage* of resources, due to policy violations. The analysis identifies the sensible points in the code that need dynamic checks in order to avoid policy violations. Our analysis also manages iterative behaviour of processes acting over resources. The analysis constructs a *finite-state* model that approximates the executable behaviour of processes capable of performing unbounded iterative actions over shared resources.