CARMA Eclipse plug-in: A tool supporting design and analysis of Collective Adaptive Systems

CARMA Eclipse plug-in: A tool supporting design and analysis of Collective Adaptive Systems*

Jane Hillston[1] and Michele Loreti[2]

[1] Laboratory for Foundations of Computer Science, University of Edinburgh
[2] Dipartimento di Statistica, Informatica, Applicazioni "G. Parenti", Università di Firenze

Abstract. Collective Adaptive Systems (CAS) are heterogeneous populations of autonomous task-oriented agents that cooperate on common goals forming a collective system. This class of systems is typically composed of a huge number of interacting agents that dynamically adjust and combine their behaviour to achieve specific goals. Existing tools and languages are typically not able to describe the complex interactions that underpin such systems, which operate in a highly dynamic environment. For this reason, recently, new formalisms have been proposed to model CAS. One such is CARMA, a process specification language that is equipped with linguistic constructs specifically developed for modelling and programming systems that can operate in open-ended and unpredictable environments. In this paper we present the CARMA Eclipse plug-in, a toolset integrated in Eclipse, developed to support the design and analysis of systems specified in CARMA.

1 Introduction

Collective adaptive systems (CAS) typically consist of very large numbers of components which exhibit autonomic behaviour depending on their properties, objectives and actions. Decision-making in such systems is complicated and interaction between their components may introduce new and sometimes unexpected behaviours. CAS are open, in the sense that components may enter or leave the collective at any time. Components can be highly heterogeneous (machines, humans, networks, etc.) each operating at different temporal and spatial scales, and having different (potentially conflicting) objectives. We are still far from being able to design and engineer real collective adaptive systems, or even specify the principles by which they should operate.

Existing tools and languages are challenged by the complex and evolving interaction patterns that occur within CAS. Nevertheless, the pervasive yet transparent nature of these applications makes it of paramount importance that their behaviour is thoroughly assessed during their design, prior to deployment, and throughout their lifetime.

Within the QUANTICOL project[3], the definition of a formal language to capture CAS has been investigated. Our objective was to develop a coherent, integrated set of linguistic primitives, methods and tools to build systems that can operate in open-ended, unpredictable environments. We named this language CARMA, Collective Adaptive Resource-sharing Markovian Agents. CARMA combines the lessons we learnt from

* This work is partially supported by the EU project QUANTICOL, 600708.
[3] http://www.quanticol.eu

other stochastic process algebras such as PEPA [8], EMPA [2], MTIPP [7] and MoDEST [3], with those learnt from languages specifically designed to model CAS, such as SCEL [5], the AbC calculus [1], PALOMA [6], and the Attributed Pi calculus [9], which feature attribute-based communication and explicit representation of locations.

To support analysis of CARMA models a prototype simulator has been also developed. This software tool, which has been written in Java, can be used to perform stochastic simulation and will also form the basis for implementing further analysis techniques in the future. An Eclipse plug-in, integrating an editor, static analysis tools and various views on a model, has also been developed. Using this plug-in, CARMA systems can be specified by means of an appropriate high-level language, which is mapped to the CARMA process algebra to enable qualitative and quantitive analysis of CAS.

In this paper we first briefly describe the basic ingredients of CARMA. After that an overview of the CARMA Eclipse plug-in and its features is provided.

2 CARMA in a nutshell

CARMA is a new stochastic process algebra for the representation of systems developed in the CAS paradigm [4]. The language offers a rich set of communication primitives, and exploits *attributes*, captured in a *store* associated with each component, to enable attribute-based communication. For example, for many CAS systems the location is likely to be one of the attributes. Thus it is straightforward to model systems in which, for example, there is limited scope of communication, or interaction is restricted to co-located components, or where there is spatial heterogeneity in the behaviour of agents.

A CARMA system consists of a *collective* operating in an *environment*. The collective is a multiset of components that models the behaviour of a system; it is used to describe a group of interacting *agents*. The environment models all those aspects which are intrinsic to the context where the agents are operating. The environment mediates agent interactions. This is one of the key features of CARMA. It is not a centralised controller but rather something more pervasive and diffusive — the physical context of the real system — which is abstracted within the model to be an entity which exercises influence and imposes constraints on the different agents in the system. The role of the environment is also related to the spatially distributed nature of CAS — we expect that the location *where* an agent is will have an effect on *what* an agent can do.

A CARMA component captures an *agent* operating in the system. It consists of a process, that describes the agent's behaviour, and of a store, that models its *knowledge*. A store is a function which maps *attribute names* to *basic values*.

Processes located within a CARMA component interact with other components via a rich set of communication primitives. Specifically, CARMA supports both unicast and broadcast communication, and permits locally synchronous, but globally asynchronous communication. Distinct predicates (boolean expressions over attributes), associated with senders and potential receivers are used to filter possible interactions. Thus, a component can receive a message only when its store satisfies the target predicate. Similarly, a receiver also uses a *predicate* to identify accepted sources. The execution of communicating actions takes time, which is assumed to be an exponentially distribution random variable whose parameter is determined by the environment.

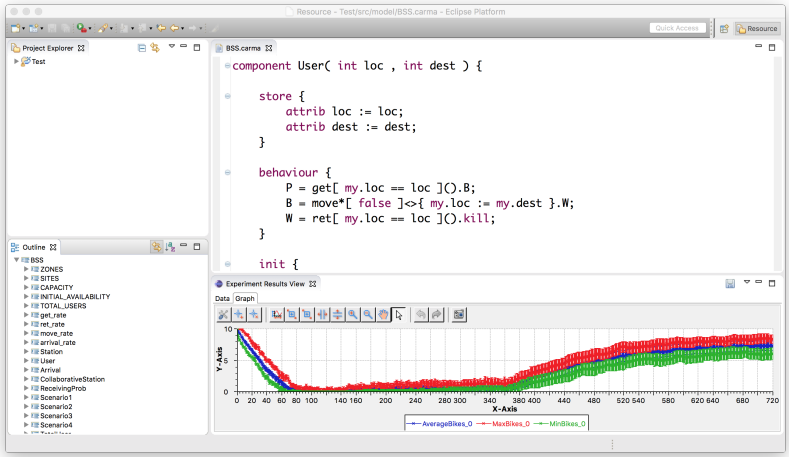

Fig. 1: A screenshot of the CARMA Eclipse plug-in.

3 CARMA Eclipse plug-in

An Eclipse plug-in for supporting the specification and analysis of CAS in CARMA has been developed. A screenshot of the plug-in is presented in Figure 1.

The CARMA Eclipse plug-in is available at http://quanticol.sourceforge.net/. At the same site detailed installation instructions can be found together with a set of case studies that shows how CAS can be modelled and verified with the provided tool.

The CARMA Eclipse plug-in provides a rich editor for CAS specification using an appropriate high-level language, called the CARMA *Specification Language* (CASL). This high-level language is not intended to add to the expressiveness of CARMA, which we believe to be well-suited to capturing the behaviour of CAS, but rather to ease the task of modelling for users who are unfamiliar with process algebra and similar formal notations. Each CARMA specification provides definitions for: structured *data types* and the relative *functions*; prototypes of *components* occurring in the system; *systems* composed by collective and environment; and the *measures*, that identify the relevant data to *measure* during simulation runs.

Given a CARMA specification, the CARMA Eclipse Plug-in automatically generates the Java classes needed to simulate the model. This generation procedure can be specialised to different kinds of simulators. Currently, a simple ad-hoc simulator is used. The simulator provides generic classes for representing *models* to be simulated. To perform the simulation each *model* provides a collection of *activities* each of which has its own *execution rate*. The simulation environment applies a standard *kinetic Monte-Carlo* algorithm to select the next activity to be executed and to compute the execution time. The execution of an *activity* triggers an update in the simulation model and the simulation process continues until a given simulation time is reached. From a CARMA specification, these activities correspond to the *actions* that can be executed by processes located in the system components. Indeed, each such activity mimics the execution of

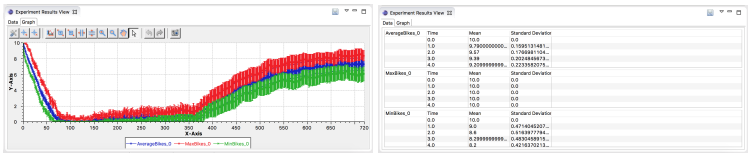

Fig. 2: CARMA Eclipse Plug-In: Experiment Results View.

a transition of the CARMA operational semantics. Specific *measure functions* can be passed to the simulation environment to collect simulation data at given intervals. To perform statistical analysis of collected data the *Statistics package* of *Apache Commons Math Library* is used[4].

The results are reported within the *Experiment Results View* (see Figure 2). Two possible representations are available. The former, on the left side of Figure 2, provides a graphical representation of collected data; the latter, on the right side of Figure 2, shows average and standard deviation of the collected values, which correspond to the *measures* selected during the simulation set-up, and are reported in a tabular form. These values can then be exported in CSV format.

References

1. Y. Abd Alrahman, R. De Nicola, M. Loreti, F. Tiezzi, and R. Vigo. A Calculus for Attribute-Based Communication. In *Proceedings of the 30th Annual ACM Symposium on Applied Computing, Salamanca, Spain, April 13-17, 2015*, pages 1840–1845, 2015.
2. M. Bernardo and R. Gorrieri. A Tutorial on EMPA: A Theory of Concurrent Processes with Nondeterminism, Priorities, Probabilities and Time. *Theoretical Computer Science*, 202(1-2):1–54, 1998.
3. H.C. Bohnenkamp, P.R. D'Argenio, H. Hermanns, and J-P. Katoen. MODEST: A compositional Modeling Formalism for Hard and Softly Timed Systems. *IEEE Trans. Software Eng.*, 32(10):812–830, 2006.
4. L. Bortolussi, R. De Nicola, V. Galpin, S. Gilmore, J. Hillston, D. Latella, M. Loreti, and M. Massink. CARMA: Collective Adaptive Resource-Sharing Markovian Agents. In *Proc. of the Workshop on Quantitative Analysis of Programming Languages 2015*, 2015.
5. R. De Nicola, M. Loreti, R. Pugliese, and F. Tiezzi. A Formal Approach to Autonomic Systems Programming: The SCEL Language. *TAAS*, 9(2):7, 2014.
6. C. Feng and J. Hillston. PALOMA: A Process Algebra for Located Markovian Agents. In *Quantitative Evaluation of Systems - 11th International Conference, QEST 2014, Florence, Italy, September 8-10, 2014.*, volume 8657 of *LNCS*, pages 265–280. Springer, 2014.
7. H. Hermanns and M. Rettelbach. Syntax, Semantics, Equivalences and Axioms for MTIPP. In U. Herzog and M. Rettelbach, editors, *Proc. of 2nd Process Algebra and Performance Modelling Workshop*, 1994.
8. J. Hillston. *A Compositional Approach to Performance Modelling*. CUP, 1995.
9. M. John, C. Lhoussaine, J. Niehren, and A.M. Uhrmacher. The Attributed Pi Calculus. In *Proc. of Computational Methods in Systems Biology*, volume 5307 of *LNBI*, pages 83–102, 2008.

[4] http://commons.apache.org