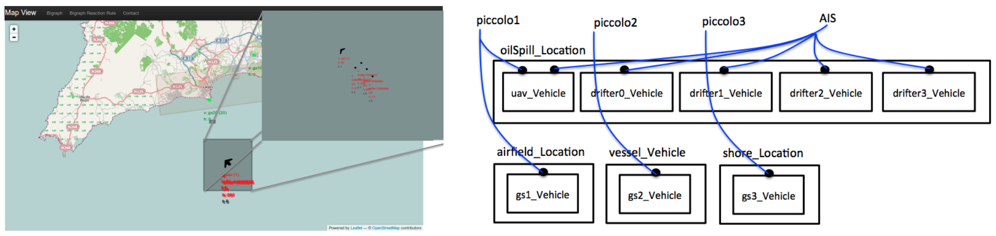
A Networked Robotic System and its Use in an Oil Spill Monitoring Exercise

Eloi Pereira[1,3], Pedro Marques da Silva[3], Clemens Krainer[2], Christoph M. Kirsch[2], Jose Morgado[3] and Raja Sengupta[1]

[1] Department of Civil and Environmental Engineering, University of California at Berkeley, USA
`eloi@berkeley.edu, sengupta@ce.berkeley.edu`
[2] Department of Computer Sciences, University of Salzburg, Austria
`ck@cs.uni-salzburg.at, clemens.krainer@cs.uni-salzburg.at`
[3] Research Center, Portuguese Air Force Academy, Portugal
`posilva@academiafa.edu.pt, japmorgado@gmail.com`

We have contributed with a bridging model, akin to the von Neumann model, to the cyber-physical systems literature [1]. This past contribution, named the BigActor model, is only a mathematical model. Here we describe a software system built to explore the value of the mathematics when integrating and controlling a network of robots sensing an environment. Figure 1(b) describes the "physical" components of the system as well as their relative location and connectivity. This particular robot network executes an oil spill monitoring exercise. There is one UAV, 3 ground control stations, 4 drifters that broadcast their position using Automatic Identification System (AIS), and one ship in the system. These are denoted as labeled boxes in Figure 1(b). Figure 1(a) shows the same information but in the physical space, i.e. the GPS coordinates of the vehicles.



(a) Mission Visualization Tool

(b) Bigraph Abstraction

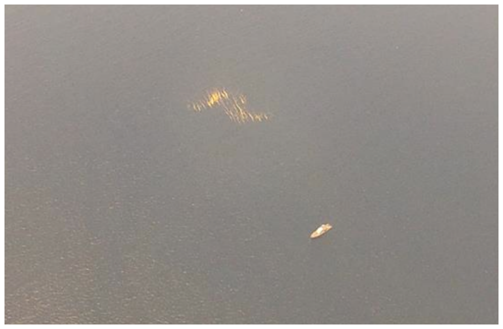
Figure 1: Oil spill mission visualization.

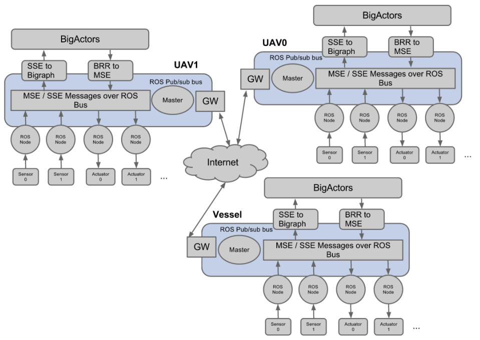
There are three communication channels. The AIS channel is used by the drifters, the *piccolo* channels by the UAV and ground stations. The other boxes in the picture are static locations representing areas like the take-off airfield, the oil-spill area, etc. The picture is formally a Bigraph as defined by Milner in [2]. We use it exactly as Milner intended to represent the ubiquitous computing machine, analogous to the von Neumann machine.

We control the system by commanding an entity such as the UAV to go from the airfield to the oil spill, or to disconnect from one ground station to another ground station, to represent logically the transfer of control of the asset from one jurisdiction to another (in one of our exercises the the ground station was onboard of a Navy vessel). These controls are formally Bigraph Reaction Rules (BRR) as in [2]. Thus the oil spill monitoring exercise is abstractly a sequence of pictures or Bigraphs, evolving in response to the controls or environmental disturbances like network disconnections due to limited communication range.

The "cyber" components controlling the system are BigActors. These are mathematically Actors as defined by Agha and Hewitt [3] bridged to the underlying Bigraph as defined in [1]. BigActors are actors equipped with the ability to observe and control the Bigraph. Figure 2(a) is an image captured from the UAV of the naval vessel and the emulated oil spill.

The Portuguese Navy emulated the oil spill by releasing 100kg of popcorn in the ocean (the yellow patch). The stills have been extracted from the UAV video to verify that the BigActor implementation is correctly able to control the UAV asset through take-off, three ground station


(a) Aerial imagery of the oil spill and vessel.



(b) System's architecture

Figure 2: Oil spill imagery and system's architecture.

handoffs, arrival in the oil spill area, imaging of the spill, connect with the AIS drifters, acquire their data, and deliver it back to base.

The creative engineering contributions in the implementation of the BigActor model are an approximation of the BigActor semantics enabling distributed control, control by logical space programming, as opposed to physical space programming, and a process to build networks out of different vehicles built by different teams. Unlike the von Neumann machine, the structure of our ubiquitous computing machine changes as the oil spill exercise executes. These changes of structure are first locally observed by each vehicle in the system. These local observations are then propagated throughout the overall system, flooding the information in a distributed manned. Our BigActor implementation has a distributed Bigraph synthesis protocol and semantics to enable coordinated control with decentralized information. The implementation supports control by logical space programming because the distributed Bigraph synthesis protocol provides each BigActor with a Bigraph abstraction of the whole system, enabling the actor to control at the Bigraph level. The Bigraph is a logical, not physical, space abstraction of the real world. Protocols in the implementation handle the binding of logical space to physical space, ensuring Bigraph Reaction Rules actually have a physical effect. This is performed using the Robot Operating System (ROS) as the underlying middleware [4]. We have created a ROS abstraction of each robot in the network called the `ros_vehicle` and defined a set of libraries that make a robot into a `ros_vehicle`. To put a new vehicle into the robot networks one should implement or install these libraries. They make the new vehicle into a `ros_vehicle`, and thereby a node in the network. The network control is at the BigActor level. This is implemented using the Scala Actor Library [5] running over ROS using ROSJava. The overall system's architecture is presented in Figure 2(b).

References

[1] E. Pereira, C. Kirsch, R. Sengupta, and J. Borges de Sousa, "Bigactors - a model for structure-aware computation," in *Proc. International Conference on Cyber-Physical Systems (ICCPS)*. ACM/IEEE, April 2013.

[2] R. Milner, *The Space and Motion of Communicating Agents.* Cambridge University Press, 2009.

[3] G. Agha, *Actors: a model of concurrent computation in distributed systems.* Cambridge, MA, USA: MIT Press, 1986.

[4] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "Ros: an open-source robot operating system," in *Proc. ICRA Workshop on Open Source Software*, vol. 3, no. 3.2, 2009.

[5] M. Odersky, P. Altherr, V. Cremet, B. Emir, S. Maneth, S. Micheloud, N. Mihaylov, M. Schinz, E. Stenman, and M. Zenger, "An overview of the scala programming language," Citeseer, Tech. Rep., 2004.