Performance Evaluation of Mobility-based Software Architectures

Vincenzo Grassi * Vittorio Cortellessa * °
* Dip. di Informatica, Sistemi e Produzione, Università di Roma "Tor Vergata", Roma, Italy
° Computer Science and El. Eng., West Virginia University, Morgantown, WV
{grassi, cortelle}@info.uniroma2.it

1. INTRODUCTION

Software architectures (SA) describe software systems at an early design stage, in terms of components and interactions among them; decisions made at this early stage can have a deep impact on the overall quality of the final software product, since they can affect several quality attributes, like reusability, performance, reliability. For this reason, it is important to develop methodologies to evaluate the impact of different architectural choices on these attributes, at the time these decisions are made [1, 5, 7, 12]. One of such decisions concerns the adoption of an *architectural style*, that determines the components type and patterns of interactions that can be used [2]. Several styles have been identified, motivated by trends in software design and technologies. In recent years, it has emerged the idea of explicitly taking into account the notion of components *location*, to deal conveniently with the fact that it is becoming increasingly common for applications to operate in large scale computing environments (like the World Wide Web), with heterogeneous locations, both geographically and logically distributed, and that can also be physically mobile (portable devices).

This rises the question on which is a suitable architectural style for such distributed applications. Besides the traditional *client-server* style, another suggested style is based on the notion of *code mobility* [6], where components of an application may autonomously decide to move themselves to different locations, during the application lifetime. Arguments in favor of this new style concern the improvements of both qualitative attributes (like customizability) and quantitative ones (like network traffic). However, it is also recognized that such arguments are not valid in general, and hence the choice between the two styles should be performed on a case-by-case basis [6].

Our goal is to define a methodology that helps the designer to perform such a choice at the SA stage, by giving insights about the impact of the two styles on the final application. Given the large spectrum of quality attributes affected by such a choice, our methodology does not encompass all of them. Rather, it focuses on quantitative attributes that are significant for the class of applications that operate in a geographically distributed environment, with heterogeneous (and possibly physically moving) locations. In particular, we focus on interaction related performance indices (e.g. generated network traffic, possibly on some particular links, or consumed energy, of interest for portable devices [10]). For a more comprehensive evaluation of the merits of different styles, our methodology should be supported by further steps, addressing different attributes.

In the following, we outline the steps of our methodology, using as example a "context-aware" application [3].

WOSP 2000, Ontario, Canada
© ACM 2000 1-58113-195-X/00/09 ...$5.00

2. METHODOLOGY

2.1 SA description

The starting point of our methodology is the description of a software application in a suitable architecture description language (ADL) that includes the possibility of specifying components location. Languages of this kind have been proposed quite recently [4, 9, 11]. Our approach is not tied, in principle, to a particular language. However, for the sake of example, we adopt a modification of the COMMUNITY language [11]. In this ADL, each component is a set of named guarded actions, that can modify only local variables. A special local variable λ indicates component location. Each action is non-deterministically chosen for execution when its guard holds true. To model interactions and (possible) mobility, we introduce a parametric connector type (*Mob_Comm*), based on a slight modification of the COMMUNITY *Communicator* connector modelling synchronous message sending. *Mob_Comm* can also model code mobility; its prototype is as follows:

connector *Mob_Comm*(c1, c2: **component**; a1, a2 :
action; x1, x2: **any_type**; I, M: **condition**)

where a1 and a2 are the names of actions performed by c1 and c2 components, respectively, "synchronized" by the connector, while x1 and x2 are c1 and c2 variables, respectively, used to send and receive the exchanged value; I is a condition that controls the connector activation (action a1, that starts communication, is executed only when both its guard and I hold true, see [11] for further details). M is a condition that controls the mobility of component c1; if M holds false, then *Mob_Comm* has the same semantics of *Communicator* (the value of x1 is sent to x2), while if M holds true, then the connector semantics corresponds to changing the location of c1 to that of c2, and then transferring the value of x1 to x2. M may also be set to the undetermined value "**?**". In this case, the

connector semantics corresponds to the non deterministic execution of both the above options.

The context-aware application that we consider uses as computing platform a portable device (PD) with wireless connections, and a server (SV) connected to a fixed network. In our example, the "context" simply consists of the physical position of PD; when PD is within a building, its position is detected by a device (AB) co-located with SV, while it is detected by a device (GPS) co-located with PD when PD is outdoor. The application consists of three components, namely AB, GPS and a monitor (MON) that periodically checks the current PD position; to this purpose, MON must communicate with AB, when PD is indoor, and with GPS when PD is outdoor. MON may be located on both PD or SV; the two solutions may have different impacts on the interactions cost. Its SA description is as follows; for the sake of brevity, we only report some components and corresponding connectors:

Components

program MON **at** λ
var sent, received : **bool**; oldpos, newpos: position;
init λ = PD **and** sent = **false and** received = **true and** newpos = **null**
do send_req: [sent = **false** ---> sent := **true** || oldpos := newpos]
[] get_pos: [**true** ---> received := **true** || sent := **false** || *Check*(oldpos, newpos)]

program GPS **at** λ
var pos: position; ready : **bool**;
init λ = PD **and** ready = **false**
do wait_req: [**true** ---> pos := *Current_pos*() || ready := **true**]
[] send_pos: [ready = **true** ---> ready := **false**]
…

Connections

MON-to-GPS: *Mob_Comm*(MON, GPS, MON.send_req, GPS.wait_req, **void**, **void**, *GPSactive*(), **?**)

GPS-to-MON: *Mob_Comm*(GPS, MON, GPS.send_pos, MON.get_pos, GPS.pos, MON.newpos, *GPSactive*(), **false**)
…

The **init** proposition states the initial variables value. MON-to-GPS and GPS-to-MON are instances of *Mob_Comm* that model, respectively, a request sending to GPS, and a reply from GPS. *GPSactive*() is a function that holds true when PD is outdoor, and false otherwise. Note that condition M in GPS-to-MON is set to false since our design choice is such that GPS can never change location, while it is set to **?** in MON-to-GPS.

2.2 Application dynamics

The goal of this step is to build a "representation" of the application dynamics. Interactions cost can change dynamically, due to location changes. Hence, the static description of a SA (its "box and line" diagram) does not contain enough information to perform an accurate evaluation of interaction related attributes. To this purpose, we derive from the SA description a *labelled transition system* (LTS), i.e. a graph where each node represents a given application state (e.g. components locations and internal state), and each arc represents the execution of a given action (e.g. the transfer of a value during an interaction). Each arc label contains information about the cost of the corresponding action (e.g. interactions between components that share the same location can be considered to have a negligible cost with respect to interactions involving communications through the network). The derivation of the LTS from the SA description may be performed automatically, using the "operational" semantics outlined in the previous section. Figure 1 shows the LTS for our example. In each state, we only report the location of MON (PD or SV) and of PD (IND for indoor or OUTD for outdoor). As cost measure we adopt the number of bytes transmitted over a wireless link (*r* is the size of a request from MON, *p* is the size of a reply from GPS or AB, and *m* is the size of MON when it changes location); where the cost is not shown it is equal to zero. Dashed arcs starting from a node represent the two possible actions (mobility or no-mobility) corresponding to the activation of the *Mob_Comm* connector with control condition set to "**?**". We call a node with these outgoing arcs a *decisional* node.

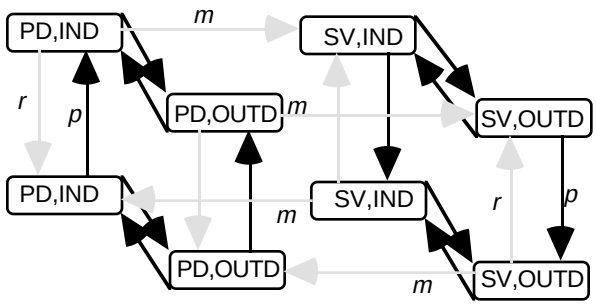

Figure 1 - LTS of the context aware application

2.3 Stochastic decisional model

The outgoing dashed arcs from a decisional node model a non deterministic choice between two alternatives (mobility/no-mobility). Our goal is to turn this non deterministic choice into a deterministic one, where one of the two alternatives is selected if it improves the considered performance index. To this purpose we derive from the LTS a Markov decisional process (MDP) [8]. *States* and *transitions* of MDP correspond to nodes and arcs of LTS, respectively; the *reward* to be associated to each MDP transition is given by the cost label of the corresponding LTS arc; with regard to MDP *actions*, only the MDP states obtained from decisional nodes have two alternative actions, that correspond to selecting the mobility or no-mobility option (with the corresponding cost). To complete the MDP construction, we only need to attach suitable probabilities to the process transitions. This is the only step that cannot be performed automatically.

2.4 MDP simplification

The goal of this step is to alleviate the computational problems caused by the state explosion. To this purpose, we drop all the non-decisional states whose outgoing arcs have cost equal to zero, and modify transitions among the remaining nodes so that the new process is equivalent to the original one (in the sense that the policy that optimizes the original MDP also optimizes the reduced one). This reduction can be performed automatically, but for space limits we do not report the algorithm. Fig. 2 shows the simplified MDP for our example (transition probabilities are not shown).

2.5 MDP solution

Solving a MDP means finding a *policy* that selects an action in each state, so that the total accumulated reward is optimized. In our perspective, if the obtained optimal policy selects in some states the mobility option, this can be considered as an indication that code mobility may represent an effective style for the considered application. In our example we get that code mobility is an effective option only if $\frac{r+p}{m} \geq max \left\{ \frac{2p_{io}(p_{io}+p_{oi})}{(1-p_{oi})(p_{io}+p_{oi}+2p_{io}p_{oi})}, \frac{2p_{oi}(p_{io}+p_{oi})}{(1-p_{oi})(p_{io}+p_{oi}+2p_{io}p_{oi})} \right\}$, where p_{io} and p_{oi} are the probabilities that PD moves from indoor to outdoor position, or from indoor to outdoor position, respectively. In general, a symbolic solution could not be feasible, and the solution must be performed numerically by instantiating appropriately the model parameters, where each instantiation defines a possible implementation scenario, so that we can compare the impact of code mobility in different scenarios.

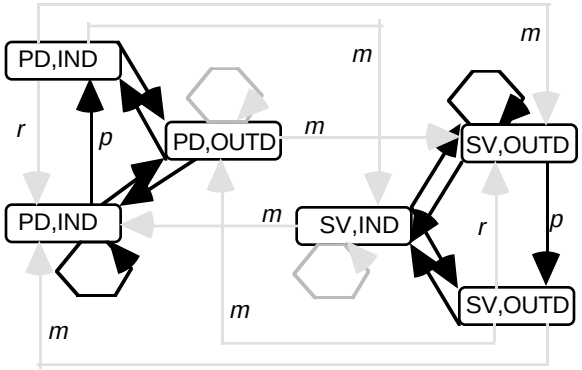

Figure 2 - Simplified MDP

Acknowledgements

Work partially supported by MURST project "Software architectures and languages to coordinate distributed mobile components".

References

[1] S. Balsamo, P. Inverardi, C. Mangano "An approach to performance evaluation of software architectures" in Proc. *Workshop on Software and Performance (WOSP '98)*, Santa Fe, New Mexico, Oct 12-16 1998

[2] L. Bass, P. Clements, R. Kazman, *Software Architectures in Practice*, Addison-Wesley, New York, NY, 1998

[3] P.J. Brown, J.D. Bovey, X. Chen "Context-aware applications: from the laboratory to the marketplace" *IEEE Personal Communications*, vol. 4, no. 5, Oct. 1997, pp. 58-64

[4] L. Cardelli, A.D. Gordon "Mobile ambients" *Foundations of Software Science and Computational Structures* (M. Nivat ed.), LNCS 1378, Springer-Verlag, 1998, pp. 140-155

[5] M.-H. Chen, M.-H. Tang, W.-L. Wang "Effect of architecture configuration on software reliability and performance estimation" in Proc. *IEEE Workshop on Application-specific Software Engineering and Technology (ASSET 98)*, Richardson, Texas, March 1998

[6] A. Fuggetta, G.P. Picco, G. Vigna "Understanding code mobility" *IEEE Trans. on Software Engineering*, vol. 24, no. 5, May 1998, pp. 342-361

[7] H. Grahn, J. Bosch "Some initial performance characteristics of three architectural styles" in Proc. *Workshop on Software and Performance (WOSP '98)*, Santa Fe, New Mexico, Oct 12-16 1998

[8] D.P. Heyman, M.J. Sobel, *Stochastic Models in Operations Research*, McGraw-Hill, 1984

[9] G.-C. Roman, P.J. McCann "An introduction to Mobile UNITY" in *Parallel and Distributed Processing* (J. Rolim ed.), LNCS 1388, Springer-Verlag, 1998, pp. 871-880

[10] M. Stemm, R.H. Katz "Measuring and reducing energy consumption of network interfaces in hand-held devices" *IEICE Trans. on Communications* (special issue on Mobile Computing), 1997

[11] M. Wermelinger, J.L. Fiadeiro "Connectors for mobile programs" *IEEE Trans. on Software Engineering*, vol. 24, no. 5, May 1998, pp. 331-341

[12] L.G. Williams, C.U. Smith "Performance evaluation of software architectures" in Proc. *Workshop on Software and Performance (WOSP '98)*, Santa Fe, New Mexico, Oct 12-16 1998