Improving availability of Mobile Code Systems by decoupling Interaction from Mobility

Koukoumpetos K., Antonopoulos N., Zhang K., and Exarchakos G.

Abstract—Resource availability in pervasive environments is restricted by many either mobility- and/or security-related factors. Multi-agent systems deployed in such environments would have to rely on a potentially low number of hosts allowing and supporting the arrival and execution of foreign code. To address this issue, this paper proposes to decouple interaction of executing programs and services from the actual software mobility pattern used to realize this interaction. The proposed system (MoDeS – Mobility Decision System) dynamically decides on the best mobility method to implement a series of software interactions while satisfying the appropriate software constraints. The system takes as input an interaction plan and produces the corresponding mobility plan. A series of simulations were performed on single- and multi-hop scenarios which showed that MoDeS can significantly increase the availability of software interactions even in highly constraint environments.

Index Terms—cooperative agents, mobility patterns, pervasive computing, mobility policies.

I. INTRODUCTION

Pervasive and mobile computing environments have been proven to differ from the usual workstation internet-enabled environment that the average user is used to [1]. Mobile computers have limited available resources, since they are empowered with small processors, small storage space and are usually connected through a wireless network link. Accordingly, pervasive computing research is strongly coupled with mobile computing research. Challenges met in the mobile computing community can greatly enhance the evolution and realisation of Mark Weiser’s ubiquitous computing in our everyday life. According to M. Satyanarayanan one of the major challenges of pervasive computing is “cyber foraging” [2], the intention is to dynamically distribute computational demanding applications from a wireless mobile device to local compute servers that act as mediators or proxies and are willing to execute incoming applications, for the benefits of the user. The most advantageous way to achieve this is by using mobile code instead of the traditional client/server paradigm, since a typical pervasive environment is fully heterogeneous, thus the latter technique faces implementation difficulties [3].

A viable solution that finds support by many researchers [3, 4], is to use the Mobile Agent paradigm as a middleware for mobility-enabled devices. The concept is that mobile applications are represented by mobile agents that act as proxies on a fixed network with more processing power. However, MA systems [5, 6, 7] assume homogeneity across the environment, in that all the servers are capable of supporting the execution of any incoming mobile agent. Moreover, most systems leave it to the programmer to specify where the agents go and always assume that the best method of mobility is for the agent to move to the server which hosts the other interacting party. Besides, in a large scale environment it is impossible for the programmer to take all the relative information and requirements into consideration in order to implement mobility. For these reasons, the extensiveness of mobile code applications is limited, and mostly kept internally within research groups or organisations, because in a closed environment the security risks are limited and the execution platform is common.

Focusing on the above premises, it is unrealistic to assume that our applications will be supported at all nearby servers, or even that a server exists who is willing to accept both the application and the modifications needed to execute our request. Such an environment limits the availability, (i.e. the possibility that a server exists in which both a requester and a provider are able to execute in order to complete their interacting session).

The solution we propose is to decouple that interaction from mobility. So, we designed a system (MObility DECision System, MoDeS) to decide dynamically on the best mobility method (pattern) while satisfying the appropriate constraints. Thus, by taking these constraints into consideration prior to migration we believe that availability can be increased and the efficiency of a mobile-code-enabled environment improved. The system takes an interaction plan as input, and gives the mobility plan as output.

It is expected that in such distributed environment, system users and administrators introduce Mobility constraints to enforce specific mobility policies that conform to their application requirements. For example, a server that contains private and sensitive information can deny any incoming code; a server that provides and stores publicly available information can accept and execute mobile code; and applications of specific type (e.g. specific programming...
language) can check if the underlying execution environment exists at the destination.

II. SETTING THE SCENE

Current systems, upon request for interaction, will try to move the requester agent to the destination’s agent host. We call this mobility pattern Requester-to-Provider (R2P). Hence, a mobility pattern can be defined as a way or type of migration between two mobile entities, a requester and a provider. However, as discussed earlier, it is quite convoluted to assume that all mobile code applications can be executed in every host in the system. With current approaches, if the destination host is not able (e.g. due to lack of required platform) to accept or execute any incoming code then the potential of that interaction is lost. In order to improve the efficiency of such a system and increase the availability of interactions, there is a need to identify additional mobility patterns.

One solution is to consider the possibility of the provider agent to come to the requester (P2R). If, for instance, the provider agent represents a document that does not have interdependencies with other agents, then it is feasible to move a copy of that agent to the requester for the interaction to take place; assuming that the server where the requester resides is able to execute/facilitate the provider agent.

Yet, if this pattern is not possible the notion of meeting point servers can be used. All or specific servers of the system can act as meeting points where two agents can meet to interact (RP2MP). If the system knows the Mobility constraints of the servers and of both the interacting agents, then it is possible to find out whether a server exists which can accommodate the needs for the execution of the interaction.

Furthermore, for a more refined analysis [8, 9, 10], we can benefit by using mobile agents that are a collection of different components. Some existing systems [11, 12] include this concept in their computation for migration, whereby fine-grain techniques decide which parts of the agent are needed at the destination to complete an interaction in order to minimise data traffic. By visualising the agent as a set of components, various improvements can be introduced. A major one is that components can be used among different users or agents. In terms of mobility though, this separation can introduce more ways of migration which can affect availability and data traffic savings. In [13] we presented the benefits of using component-based agents to minimise the used bandwidth in a mobile system, compared to the standard technique of R2P; an issue which affects pervasive computing environments as well. In terms of availability the use of components introduces five more mobility patterns. If it is feasible, instead of moving the whole agent, the components of the interacting agents can migrate. Thus, Requester Component-to-Provider (Rc2P) and Provider Component-to-Requester (Pc2R) are two more possibilities. In the case of components migrating to meeting points it follows like: Requester Component and Provider Agent move to the Meeting Point (ReP2MP), Requester Agent and Provider Component moves to the Meeting Point (RPc2MP), or both Requester and Provider Components move to the Meeting Point (RcPc2MP).

Putting it all together, we assume a mobile computing environment which consists of the following entities: agent servers, components, and mobile agents. An agent server is a system entity which provides the necessary infrastructure for the agents to be created, live and execute, communicate, interact and migrate. Components can be considered like functions or methods of the agent; small programs that have a specific input and specific output. Agents are instantiated at an agent server. This agent server is called home server and is responsible to create a globally unique ID (GUID) to label each agent during their life. Every agent is a set of different components and has an itinerary (interaction plan). A monolithic agent is an agent with one component. In the system, there are eight (8) mobility patterns (R2P, P2R, RP2MP, Rc2P, Pc2R, RpP2MP, RPe2MP, RPePc2MP) available for the agents to use to accomplish their task and all its entities are described in XML. Agents and components in the system are grouped into different types/roles depending on various characteristics. Such characteristics are:

- **Type (Agents & Components):** Since agents and components have different roles in the system, restrictions can be imposed on them according to their types. For example if the type of an agent is ‘security’ or ‘system’ agent, and represents local security policies, then it is suitable for that agent not to leave the server.

- **Owner (Agents & Components):** It is possible that some administrators need to block specific agents or components from either leaving or visiting the system depending on their creators or originators.

- **Language (Agents & Components):** Since agents and components can be written in different programming languages in order to accommodate greater functionality of the system, if the required environment is not available at the destination, the relevant Mobility constraints should be implemented.

- **Home Server (Agents & Components):** Constraints can also be enforced by the location of the home server.

- **GUID (Agents Only):** Constraints can be enforced by direct reference to the Global Unique ID of an agent.

- **Input or Output (Components Only):** Since components are the methods of an agent they have to specify the format of the input they accept and the format of the output they produce. That can be in the form of files or attributes or parameters. If, for example, a component gives output in a specific file format, and that format cannot be displayed at the server, then that component cannot execute.

According to this role-based scheme, users and administrators can create sets of rules for filtering and applying the various constraints onto groups, instead of individual agents or components.

There are various pragmatic reasons why constraints are introduced in such a system. A possible advantage will be to know in advance any lack of a supporting environment; in this case the system will try to find other servers that are able to execute specific applications, prior to a redundant migration. Moreover, some servers will be able to enforce certain constraints on a component-by-component basis.
constraints depending on their processing load (load balancing). Also, servers that specialise in specific domain can impose constraints to accept agents relevant to that domain and reject any other type. Finally, trust related constraint may need to be enforced by the servers since it has been considered as an increasingly important factor in network management [14].

There is a distinction between the user constraints and the system constraints. The owner of the mobile agent, upon creation, can specify which locations his/her agent is not allowed to visit. On the other hand, servers not only enforce location constraints, but can also block specific agents, components or groups of them from migrating to that server. Servers do that by either explicitly defining Unique IDs of the agents or components or by using more complex filtering notations depending on their characteristics. So, in total there are three tables of constraints:

a) one that is created by the user(s) and specifies locations (destinations) where owned agents cannot visit,

b) one that is created by the administrators which specifies which agents or components are not allowed to come to the system, and

c) one that is created by the administrators and specifies where local agents are not allowed to go.

Mobility constraints are described with their respective XML Schemas. In particular, the server constraint XML Schema is split in two parts representing the afore-mentioned two tables. One for applying constraints on incoming agents and the other for restricting local agents to leave the server. Both the incoming and the outgoing parts specify which agents or components cannot come or leave by using filtering on their respective characteristics. The only difference between the incoming and outgoing server constraints is that the outgoing constraints should also confer the prohibited locations that agents leaving the system cannot visit.

III. DESCRIBING AND ANALYSING THE ALGORITHMS

When a user creates a new agent, its Mobility constraints are saved within the agent’s XML description. There are also certain constraints that can be enforced upon an agent from its home server, as described in the previous section. Server Mobility constraints have higher precedence over agent constraints to ensure server integrity, and are applied in a cascading manner. These constraints have to be included into the agent, to avoid future interactions with non-permitted servers. For example, if an agent changes its interaction plan after an interaction that has moved him to a foreign server; MoDeS has to re-calculate its mobility plan and for that it will have to take into consideration both the agent’s and home server’s constraints. The algorithm iterates through the outgoing server constraints and looks for any possible matches on the agent fields. If it finds a match it inserts the respective locations in the agent’s XML file. It continues the same process for each of its component constraints as well; and if a match is found it updates the agent file. When the process finishes it removes the duplicate and initialises the agent.

For the system to be able to devise the possible Mobility Patterns the agent’s (requester or provider) constraints must be known as well as the constraints of their servers and possible meeting point servers. And although this may seem to produce great overhead in general, there are examples of areas that could use this technique. In this system there are two approaches that can be used for updating. One approach is periodic updating and shares the concept of link-state routing method which is used in computer communications. It is applied by requiring each server to pass on details about its closest neighbours to every other server in the network. The second is the ‘per hop’ method and works by requesting the relevant information at each step of the travel plan. This can be possible by using an updating protocol, which returns the relevant information from the server that hosts the provider agent. When each server receives the above information, it stores the relevant XML files for each server but also builds a list of the locations of all the agents of the system. Thus, when a local agent requests interaction with a foreign agent, the system will be able to retrieve the relevant file of the provider, in order to load the relevant constraints.

The most crucial algorithm, though, is the one that considers the Mobility constraints in order to find a possible method for the interacting agents or components to meet. For each interaction, the system has to check which of the mobility patterns are available depending on the constraints. First the system has to verify that the agent about to migrate can leave the server by checking the agent’s constraint table and its home server’s outgoing table. Then the system has to verify if that agent is allowed to arrive at the destination, by checking the destination server’s incoming table.

IV. RUNNING AND EVALUATING THE SIMULATIONS

A simulator was set-up in order to compare availability of interaction in different mobility patterns when certain Mobility constraints are applied in a mobile agent system. The simulator creates agents with random characteristics, i.e. number of components, number of constraints, home server, number of hops, etc. These agents are allocated on servers which have random constraint characteristics of their own. The selected parameters to run the simulation were the following:

- The number of agents was picked from a set of 10, 50, 100, 500, 1000,
- The number of servers where the agents were distributed was picked from a set of 2, 10, 50, 100, 500
- The number of hops, which also specifies if the agents were single-hop or multi-hop was picked from a set of 1, 2, 3, 4, 5, 10, 25. The selected number acts as an upper bound. Thus in a simulation there is the possibility that agents have less number of hops than this upper bound.
Finally the percentage of constraints each agent and server has. This number is selected from the following set (25%, 50%, 75%) and represents again the upper bound of Mobility constraints. When the simulator generates constraints for an agent, this number will be randomised as the upper limit and rounded, and the outcome will be the number of the prohibited locations for that agent. For example, if the simulation runs with 1000 agents, 100 servers and 50% constraints, for each agent the maximum outgoing constraints will be 50 (100 servers * 50%). The same corresponds to the server incoming constraints, but there this number does not map only to locations but also to agents or components of agents. The possibility of any is equally distributed at 33%.

The results of the simulations confirmed our expectations. By having different mobility patterns in a mobile computing environment, even with high number of restrictions or constraints applied, the overall availability is increased. Compared to the current approach of most mobile code systems which rely purely on the R2P pattern (labelled as R2P in the simulation graphs) the availability is increased around 25% with the MoDeS approach. According to the results some of which are provided in the charts attached (see Figures 1 to 4), we can observe that MoDeS performs better in all situations.

As constraints increase though, the benefits of using a system like MoDeS increase dramatically up to the point where more than 50% of interactions that were not possible to execute with R2P are now possible. The improvement achieved depends on how constrained the environment is. In general, MoDeS improves availability more significantly in not very severely constrained environments because the MoDeS system can easily bypass few constraints whereas a standard algorithm is blocked straight away. The charts show the number of interactions that could not complete due to Mobility constraints. MoDeS is compared against the standard agent mobility approach whereby the requester always migrates to the provider agent. Thus “NO R2D” labelled graphs show the number of failed interactions in mobile agent systems that use R2P as the only available pattern. In single hop charts the total number of interactions is the same as the number of agents. In multi-hop we have included an environment where requester agents have five different

![Fig. 1: Availability in Single-Hop with 10 Servers with (a) 25%, (b) 50% and (c) 75% Constraints](image1)

![Fig. 2: Availability in Single-Hop with 100 Servers with (a) 25%, (b) 50% and (c) 75% Constraints](image2)
provider agents to interact with, so the total number of interactions is the number of agents multiplied by the number of hops (5 in our case).

It must be noted that three assumptions were made for this system to work:

i. Incoming Server constraints block agents or components according to constant characteristics and not according to previous visited locations. This way the application of constraints becomes more scalable since the updating of constraints does not need to occur frequently.

ii. We assume that the system has global knowledge of the Mobility constraints and that updating of these Mobility constraints is not often. This assumption is realistic if we realise that Mobility constraints differ from security policies. Mobility constraints appear at the execution and implementation layer of mobile entities and can emerge in situations where that execution becomes problematic.

iii. We assume that at specific instances in time some agents act as client agents and some others act as server agents. This is a sensible approach since in a typical agent environment we have agents that play the role of a server by providing specific services and other agents that use these services as clients.

It has to be emphasised that although the analysis of the above is based on a mobile multi-agent environment, it does not bind the benefits to that environment only. The techniques that are explained can be applied in all mobile code architectures as long as the required modifications are made. But the mobile agent approach presents a more detailed basis for analysis of the concepts.

MoDeS uses the following protocols:

1. The constraint sharing protocol: This protocol is used by the servers to disseminate any changes on their own or their agents’ constraints. The packet of this protocol contains the names of the servers whose constraints it carries, the last date of update and the table of constraints for each server.

2. R2P migration protocols: When MoDeS decides R2P as the appropriate mobility pattern, then the system ‘packs’ the agent’s XML file with any other pertinent files and sends them to the destination server. The destination host, after receiving the agent, creates a configuration file for that specific agent. This file must include information...
about the host, the way in which the provider agent can be contacted (e.g. local path), as well as the method of communication the relocated agent can use (i.e. message queue ID, shared memory path). Then the destination server is responsible to re-initiate the agent. The agent before migration has to be set to ‘migration mode’. When it is initiated again at the destination server, it has to acquire the relevant information from the configuration files, and then confirm its location.

3. P2R migration protocol: In the case of P2R, MoDeS has to inform the provider agent’s host for the required migration, and request the agent or component to be sent. The ‘packing’ of the provider agent is done in the same way as in the previous case.

Meeting Point protocol: Furthermore, in the case of meeting points, MoDeS has to ‘pack’ and send the agent to the meeting point, inform and request the provider agent to move to the meeting point and also inform the meeting point server that it will be used as such. This last information is crucial for the meeting point concept to work. If it does not happen, we may have a situation where one agent arrives, is initialised and tries to communicate with the other party, which still has not arrived, thus does not have a local path or access point.

V. CONCLUSIONS

The current landscape of software mobility is characterised by the strong coupling of interactions to a specific mobility method (e.g. client program mobility in mobile agent systems; server program mobility in Java applet applications). Furthermore, the mobile code programmer needs to specify not just the interactions of its program with other programs or services but also the location where these interactions will take place. Mobility thus has become yet another programmer-defined parameter. Although that approach can work well in small to medium scale, homogeneous, closed environments, it does increase the complexity of developing mobile code. It also makes its applicability rather restricted in pervasive computing environments whose sheer scale and heterogeneity makes the assumption that the programmer knows a priori the location and constraints of each service or network quite unrealistic.

Our proposed solution first decouples interaction from mobility. It then finds all the ways that each interaction can actually be implemented taking into account server and agent constraints. These Mobility constraints provide a single but powerful method for modeling the heterogeneity of pervasive computing infrastructures. They can represent hardware limitations (i.e. low processing power) lack of support for certain software (i.e. absence of certain language interpreters) and load balancing / specialisation restrictions (i.e. servers dedicated to supporting specific applications such as simulations or desktop publishing).

We are planning to investigate interesting variations to the system we described in this paper. For example, rather than allowing only two possibilities with respect to whether an agent is allowed to go to a particular server (allow or forbid), we could assign percentage values. In this way we could setup policies whereby specific agents could use specific servers preferentially or otherwise. So, predefined types of processing could be directed towards specified sets of servers and therefore an administrator could effectively perform processing driven server partitioning.

VI. REFERENCES