Mobile Agents and Software Deployment

Roberto Silveira Silva Filho,
ICS -Information and Computer Science, UCI - University of California Irvine
444 Computer Science Building, University of California
Irvine, CA 92697-3425, USA
e-mail: rsilvafi@ics.uci.edu

Abstract

Software deployment is a complex procedure that evolves the release, installation, adaptation, reconfiguration, update, activation, deactivation, retirement and remove of software in a set of sites. This paper presents and describes some ideas concerning the use of the mobile agents as a promising paradigm to support this important software engineering activity. The use of workflow is also presented as a coordination tool for expressing large-scale software deployment activities.

Key Words*: Mobile Agents, Software Deployment, Configuration Management, Workflow and Distributed Systems.*

1 Introduction

The objective of this paper is to propose some ideas concerned the use of mobile agents and workflow in the software deployment research area. Few publications addressed this problem using the mobile agents paradigm and none of them, according to the author knowledge, used the idea of workflow to express conditions and coordination in large-scale software deployment. This paper proposes some ideas and possible solutions in this area using the mobile agent paradigm and workflow.

This paper aims at answering the following questions:

How mobile agent paradigm can be used in the context of configuration management, runtime change and software deployment?

How workflow management systems can be used to express dependencies in large-scale software deployment?

1.1 Paper Description

The next section discusses, in a summarized way, the basic concepts related to software deployment, mobile agents and workflow Section 3 discusses the use of the mobile agent paradigm in the context of software deployment activities and its requirements. Section 4 proposes the use of workflow management as one solution of the coordination problem in software deployment. Section 5 presents the Software Dock as a related work. Finally, in section 6, some conclusions are posted.

2 Basic Concepts

In this section, there are introduces the basic concepts used in the comprehension of this paper. In special, we introduce the concepts of configuration management, software deployment and mobile agents and workflow.

2.1 Software Deployment

Software applications are not stand alone systems, they are implemented as a set of software components, data and executables, being developed to operate in a constantly changing environment. Additionally, in order to reach a broader consumer market, the software must be compatible with differ-

ent operating systems and its versions, besides of being able to run on different hardware platforms. In this heterogeneous environment, each host in a network can have different hardware and software configurations.

The software deployment process, for these systems, is becoming complex. These new applications demand new installation procedures and polices. Software producers distribute customizable and not complete applications. Before the installation, the system must be checked for the presence of required software component versions. In case they are not present, components from different vendors may need to be gathered and installed. This process usually evolves the query of different vendor sites distributed in the Internet.

Examples of such complex applications are modern Internet browsers. These applications are deployed with a minimum set of plug-ins. Whenever one of these components are required for a specific functionality, embedded in the Internet homepage, the consumer (client), in which the application is installed, usually needs to download this plug-in from its producer company home page.

Another example is the installation of operating systems in new workstations. It is usually the case that new drivers, not available in the distribution, are required.

This process introduces new requirements, which are discussed in section 2.1.3.

2.1.1 Definition

According to Carzaniga et al. [CFHHHW98], informally speaking, the term **software deployment** refers to all the activities that make software systems available for use. Software deployment comprehends the process and activities related to the release, installation, activation, adaptation, deactivation, update, removal and retirement of software components in a set of hosts. It requires interactions among **software producers** and **software consumers**. Once deployed, a software system is available for use in a **customer site**.

For now on, we will use the following terminology, described by Carzaniga et al, as follows.

A **site** may be a host or set of hosts that uses a set of resources. A **software system** is a coherent collection of artifacts, such as executable files, source code, data files and documentation. A **resource** is anything needed to enable the use of a software system at a site, for example, and IP port, memory, disk space and other system. A **software producer** is a company or site that creates and deploy new releases of the software to be installed. The **software consumer** is the host in which the software needs to be deployed to.

The software deployment process is composed by 8 main phases. In the next sections, these phases will be described and some software deployment requirements will be discussed.

2.1.2 Software Deployment Process

The deployment process consists of several interrelated activities that can be executed in the producer, consumer or both sites. These activities are release, installation, activation, adaptation, deactivation, update, removal and retirement. We will now describe these phases as presented by Carzaniga et al.

- **Release**. It is the activity that interfaces between the software development and its deployment. It is performed in the producer side and encompasses all the operations needed to prepare a system for assembly and transfer to the consumer site. It collects and specifies all information necessary to carry the other activities of the deployment process. It is divided in the **packing** and **advertising** subphases.

  In the *packing* phase, all components necessary to the application are collected and organized, in order to be transferred to the consumer sites. Such information comprises the components, documentation, its installation procedures, dependencies and management properties.

In the *advertising phase*, meta-information about the characteristics of the system being deployed is collected and disseminated to interested parties.

- **Installation**. It covers the *transfer* of the application components form the producer site to the consumer site, followed by their *configuration*. It prepares the system to be activated.

- **Activation**. It is the activity of running the installed application in the customer site. For complex systems it might require the initialization of other services and process. An example is a network application that needs the appropriate network daemon to be running in the UNIX system. If the required applications are not properly installed in the system, this lack can trigger an installation process of these applications.

- **Deactivation**. Is the inverse of activation activity. It performs the shut down of the running application. It is also required before other deployment activities can take place, for example, during update operations.

- **Update**. It is a special case of installation. Represents the partial or total transfer of new component versions, in order to replace components of an existing installation. Before installation, most applications require the deactivation of the software. Some of them, however, allow this process to be performed at runtime.

- **Adaptation.** Like the update activity, the adaptation involves the modification of a software system that has been previously installed. Adaptation differs from update in that the update activity is initiated by remote events, such as software producer releasing a new component version, whereas adaptations are initiated by local events, such as a change in the environment of the consumer site. For example, the installation of a new graphic card may require the system do adapt to its new characteristics.

- **De-installation.** This activity consists in the removal (undo) of the application components from the system. As a result, the remove process must inspect the current state of the consumer site. This procedure must not affect other installed systems, and dependencies check must be performed in order to keep components that are shared with other applications.

- **Retirement or Derelease**. Consists in discontinuing the support for an application by the software producer. It usually requires that the withdrawn of the software by the producer be advertised to all known consumers of the system. It does not directly affect the consumers, which can continue to use the software.

In summary, the producer side is responsible for the release and retirement of the software, while the consumer side performs the activation, deactivation and adaptation of the software. The update and installation is a conjunct operation, performed by both sides.

2.1.3 Issues and Requirements

Carzaniga et al. also presents a list of issues concerned to the software deployment process. The most important ones for this work are described as follows.

Change Management. Activities such as software and hardware updates are natural and should be supported by the deployment process. The update and adaptation procedures must be able to deal with issues like new hardware and OS upgrades.

Component Dependencies. A non-trivial system is composed by many components exhibiting various interdependencies. A dependency is any kind of "use" relation among components. The deployment process must be able to deal with these dependencies, especially during installation and de-installation procedures. Installation may also depend on the existence of other applications and components. For example, a package may need a zip tool

in order to be unpacked. This tool needs to be previously installed in the system. De-installation should be carefully implemented in order not to remove shared components, used by other applications.

Coordination. In client-server distributed systems, it is usually the case that updates or deployment activities performed in the server, must be followed by updates in the clients. This process usually has to be coordinated. For example, it is usually the case that the clients need to shut down before the server update. Other example is the schedule of updates and adaptations to hours in which the system is less used. In order to cope with this requirement, mechanisms to provide these kinds of coordination should be provided.

Large-scale Content Delivery. The process of transferring software packages from the producers to the consumers is called *content delivery*. Mechanisms and polices that allow the efficient transfer of this data must be supported, especially in wide-area networks as the Internet, in which the low bandwidth can be a problem.

Heterogeneity. Current computer networks are composed of hosts executing different operating systems over a variety of hardware platforms. The deployment process must consider these differences, allowing the correct specification of component dependencies. Different software versions must be deployed to different hardware and software configurations, and the deployment process must be able to run on them.

Deployment Process Changeability. In general, most deployment activities execute at the consumer site, make use of system resources, and often require privileged access to system components. The system must be able to provide information about what is being executed and what is being changed during the deployment activities. This is usually implemented using logs and notification services.

New security polices, for instance, can be specified by the administrators in order to prevent improper changes in the system. The ability to change both, the polices associated to the software deployment and the procedures evolved in this process, must be supported.

Integration with the Internet. The Internet is a natural media for deployment, advertisement and integration of software components. The deployment process and technology must be tightly integrated with the Internet protocols and standards. For example, components can be deployed from different sites interconnected by the Internet. Systems can be configured to be automatically updated and use on-line support databases, which can be accessed using a browser interface.

Security: Privacy, Authentication and Integrity. With the increasing use of Internet as a deployment media, some security issues arise. Software components and sensitive customer's data are being transmitted through public network links. This data should be deployed in a secure way, using SSL for example, to ensure its integrity and privacy. Additionally, since the installation routines usually have access to special system resources, properties and databases, the execution of the deployment activities must be monitored. Routines to check the proper software execution can also be applied.

2.2 Mobile Agents

In this section, the mobile agent paradigm is introduced. Some definitions, examples and comparisons are also presented.

2.2.1 Definition

In computer science the term agent can be associated to many different concepts, as in artificial intelligence and user interface. In general terms, a software agent can be defined as a program that works on behalf of its owner [GHNCSE97].

In this paper, we are concerned with the mobile software agents. The mobile agents research is being concentrated in areas such as (telecommunication) network management, electronic commerce, load balancing, fault tolerance and mobile computing. In this context, Rus et al. [RGK97] defines a mobile agent as an object that migrates through many hosts in a heterogeneous network, under its own control, in order to perform tasks using resources of these hosts.

2.2.2 Mobile Agents versus Client-Server

Mobile agents are autonomous entities, independent from the applications that generated them. They can move along system hosts following a predefined plan (script), performing activities on behalf of its agent owner, or can use environment information, as network link load and a resource location, to guide their migration process and accomplish their mission. They carry their own state during migration and do not keep permanent connections with their home sites.

In the client-server paradigm, owners of a resource (servers) are usually physically distant from its users (clients). The communication between client-server is performed through message exchange (typically remote procedure calls) conveyed through a computer network. On the other hand, in the mobile agents paradigm, a software agent moves to the place where the resource(s) is(are) located in order to interact locally.

Compared to a client-server centralized system, the use of mobile agents carrying their own data does not reduce the overall traffic of data in the network. In both cases, data or part of the data must be copied locally, in the client hosts. The decentralized model, however, distributes the data traffic over the local network, unloading the central server backbone. The traffic is not client-server centric but peer-to-peer centric. The decentralization of data and control also distributes the server processing and communication among client hosts [SWME00].

2.2.3 Applications

The literature [CHK94; KT98, RGK97] describes many applications that can benefit from the use of the mobile agent paradigm. Some examples are described as follows:

- **Mobile computing**: In applications evolving mobility, the presence of a network connection is intermittent, or has variable bandwidth rates [RGK97]. Mobile agents can migrate to mobile hosts, perform their activities and move out when the network connection allows to.

- **Fault tolerance and Load Balancing**: Tasks and processes can be split in small subprocesses in order to perform their goal. These subtasks can be configured to migrate form host to host in order to distribute processing load or can be duplicated providing fault tolerance. The agent can also operate in the host independently form network connection, allowing temporary absence of it.

- **Electronic Commerce**: Mobile agents, acting as customers, can be configured to move through different nodes from a network in order to perform commercial transactions on behalf of its owner. The agents can search for certain kind of product or service, compare its prices and perform purchases and orders on behalf of its owner.

- **Distributed System Management**: Mobile agents can move through hosts in a distributed system, collecting management data (passive management) or reconfiguring nodes in order to implement different management polices (active management).

The use of mobile agent paradigm in configuration management, in special, software deployment, is a new field of study. An example of use of this paradigm in software deployment is described by Hall et. al [HHHW97], in the Software Dock system, described in section 5.1.

2.2.4 Mobile Agent Systems

A Mobile Agent Systems (MAS), or Agency, is a computational framework that implements the mobile agent paradigm, providing services and primitives that help in the implementation, communication and migration of these components. Some examples of such systems are ObjectSpace Voyager [ObjectSpace97] and IBM Aglets [KLO97]. A more detailed description and comparison among this ad other systems can be found in [KT98].

Due to the mobility requirement, the agent is generally implemented using interpretable programming languages. The Voyager framework, for example, uses the Java language. This characteristic also allows the implementation of security policies, in which the local access to resources can be limited. An example of this system is the Java sandbox, provided in the most popular Internet browsers. The sandbox is a restrict Java runtime environment that allows the execution of applets, mobile Java applications that are downloaded with web pages code.

2.2.5 Advantages and Requirements

According to Harrison et al. [CHK94], the ability to migrate among distributed systems host provides many benefits to the mobile agent paradigm. Among them we can list:

- Local agent-host interaction, reducing the bandwidth use of the network;
- Support for thin clients, with short computational power, or with scarce resources;
- Facility to implement semantic routing, as the example of workflow applications;
- Support for scalable applications ; and
- Improvement of fault tolerance to network link failures.

On the other hand, the mobile agent paradigm has some disadvantages, which introduces some requirements as follows:

- Need for secure execution environments, with more severe access restrictions, in order to prevent malicious agents detection (virus);
- Performance limitations due to the use of security polices and interpreted languages;
- The communication and processing overhead associated to the migration of the agents.

Harrison et. al argues that, if considered all positive and negative points, the mobile agent paradigm provides an open and generic framework for distributed application development. Even though none of these characteristics are exclusive from the mobile agent paradigm, these aggregate set of benefits are hardly implemented by other paradigms as the client-server.

2.2.6 Current Software Deployment Tools

Currently, a wide variety of technologies exist to support various aspects and activities of the software deployment process. In this section a brief summary of these technologies will be described

Installation Tools. These tools generate a set of executable and artifact files that must be downloaded or provided via a distribution media like a CD, or via Internet. These files can be compressed in a single installation executable file. In order to install the system, the installation software needs to be executed in the customer site. One example of these tools is the InstallShield installation tool [InstallShield] that generates a self-extractor file that manages the installation and des-installation procedures of an application.

These tools are usually platform specific, and allow a minimal degree of configuration on what components to install.

Package Managers. These installation tools use the concept of packages. A package is an archive that contains the files that constitute a system together with some meta-data describing the system. The packages have to be copied to the customer site in order to be installed. Some package managers provide file transfer capabilities. Once in the system, the package is installed by the package manager.

These tools usually do not provide activation and deactivation capabilities and some of them allow a primitive police specification. An example of a package tool is the Red Hat RPM [BIJORD95].

Application Management Systems. These kinds of systems generally support all of the life cycle activities except the producer-side release activity. Their architecture is generally centralized. These systems were designed to manage the software deployment in large or medium organizations that both, produce and consume software. In these systems, a central server typically controls all management and deployment activities. An example of such application is the System View from IBM [IBM98].

2.3 Workflow and Workflow Management System

In this section, a brief introduction to workflow and workflow management system is presented.

2.3.1 Definition

Workflows are computer interpretable description of activities (or tasks), and their execution order. The workflow also describes the data available and generated by each activity, parallel activities, synchronization points and so on. This description may also express constrains and conditions such as when the activities should be executed, a specification of who can or should perform each activity, and which tools and programs are needed during the activity execution [JB96]. The basic workflow terminology is described by the Workflow Management Coalition [WFMC96].

Workflow Management Systems (WFMSs) are used to coordinate and sequence business processes, such as loan approval, insurance reimbursement, and other office procedures. These processes are expressed as workflows.

2.3.2 Example of WFMS

The WONDER (Workflow ON Distributed EnviRonment) architecture [SWME00] defines a WFMS that addresses, in special, the scalability and availability issues. The architecture is based on the mobile agent paradigm. the case is represented as a mobile agent that migrates from user host to user host, following the process definition. The case is implemented as a mobile In the WONDER architecture, the control, the storage of data, and the execution of the activities are all distributed over the hosts of an enterprise computer network.

The EVE Workflow [GT98] is a WFMS based on a persistent event middleware, the EVE. It was developed by Geppert et al. in the Zürich University. It is based in the Broker/Service model [TGD97], that allow the activation of services based on events. Brokers, distributed by hosts in a network. These brokers implement the workflow activities. The communication is performed using events, exchenged through distributed EVE servers.

3 Mobile Agent Paradigm and Software Deployment

The software deployment process requires the move and configuration of artifact files from the software producers to the software consumers. This deployment process must be able to properly adapt and configure the software to the current characteristic of a host in which the software will be installed. This process requires close interaction with the customer sites in order to adjust the system to it current configuration.

3.1 Mobile Agents Versus Installation Tools

The mobile agent paradigm allows both, the transportation and execution of the installation software in a more customizable way. Instead of providing all the installation files at once, in one single package, the mobile agent can be programmed to request the installation files according to the current configuration of the system, downloading only the necessary components. Furthermore, the mobile agent can be easily updated and managed, guaranteeing that the most recent installation procedure is executed.

3.2 Software Deployment Requirements

In this section, the use of mobile agent paradigm will be discussed with their benefits and drawbacks compared to some of the software deployment requirements described in section 2.1.3.

Large-scale Content Delivery. The agents can optimize the process of transferring the software application components. Based on the local description of the customer site, the agents can request only the necessary artifact components from the consumer host, preventing the transfer of already in-

stalled components.

Mobile agents are autonomous per-to-peer entities. During a large-scale delivery of software components that can communicate with each other, the agents can be configured to exchange artifacts between other agents in close hosts, avoiding the use of a (possible) slower link with the customer site. A peer-to-peer data transfer police was implemented in the WONDER project described in [SWME00].

Heterogeneity. The use of interpreted languages, as Java, by the MASs, allows these systems to cope with the hardware heterogeneity. The same installation software (the mobile agent) can be executed in different hardware/software platforms. After reading the local configuration, the mobile agent can request the specific component files, for the current operating system or hardware platform, to be transferred from the software producer(s).

Integration with the Internet. Current MASs, as the Voyager and Aglets, are implemented in Java and can communicate using RMI or CORBA. These protocols are implemented as middleware layers on top of standard Internet protocols.

Security and Deployment Process Changeability. The security restrictions and polices usually associated to the mobile agent paradigm are very similar to the ones of the software deployment. A MAS usually provides an agent execution environment, or agency, which hosts the execution of the mobile agents, and provides implements the interface between these agents and the host system. The deployment of software as a mobile agent allows the system to monitor installation procedures, using the agency resources, preventing illegal operations in a more reliable way than the execution of a standalone executable installation package.

For being mobile and generated over demand, the mobile agent script can be modified at deployment or at execution time. This change flexibility can be used to provide runtime change of deployment polices, for example, and to provide current changes in the deployment activities.

Push and Pull. The mobile agent paradigm allows the installation process to be either pull or pushed by the software consumer. In the pull approach, in response to an event, the agent can migrate from the producer site to the customer site, providing the installation data. On the other hand, an update in the consumer site, can result in the push of the agent to the server side, in order to provide the new components.

The **component dependencies** and **the change management** requirements are implemented as part of the deployment language. It is an internal aspect of the mobile agent script. The software Dock project addresses this problem defining a special language. This system will be described in section 5.1

3.3 Disadvantages

One disadvantage of the use of mobile agent paradigm is the need of a network connection, at least during the time necessary to the agent migrate to the customer site and collect all artifacts for the installation. After that, the mobile agent can be configured to perform the installation process without the network connection presence.

Another drawback related to the network use, if compared to the other delivery media, is the lower speed of current Internet connections. This problem, however, can be worked around with the use of a hybrid police. For example, a shared installation repository can be used in a local network, from where the installation takes place. The remote transfer is performed only once, after that, the installation of the software in the other hosts can use the local high speed LAN bandwidth. This approach is used by the Software Dock System described in section 5.1.

4 Coordination in Software Deployment

This section discusses the use of workflow and workflow management systems integrated with the mobile agent paradigm to provide the coordination requirement presented in section 2.1.3.

In order to express temporal and interdependence relations, the software deployment process can be expressed as a workflow. In such approach, the se-

quence of activities to be executed, for example, shut down of the client hosts, update of the server and restart of the system can be expressed as a set of consecutive and parallel activities.

An example of the use of a workflow in the mobile agent paradigm is the WONDER project, in this system, the mobile agents are represented as activities that coordinate their own execution and the creation of the subsequent activities. Each mobile agent follows a predefined plan, expressing the whole process.

In order to be used by the software deployment systems, these inter-activities dependencies must be expressed in the installation script language.

The Application Management Systems described in section 2.2.6, addresses this problem in a centralized fashion, using events to notify and control the software consumer hosts. The mobile agent approach, however, allows addressing of this problem in a decentralized way. Mobile agents are autonomous entities and can be programmed to coordinate other agents.

For example, in a server update procedure, a set of parallel agents could be deployed in the network, they would migrate, each one, to a host in the system and deactivate the software that could be affected by the server modification. A synchronization activity is defined in order to receive notifications from the clients, informing about the program deactivation. After that synchronization (and join), the server update agent is executed. When the update is complete, new parallel agents are deployed in order to activate the client programs. These agents can be generated by the agent, which executed the update.

Once created, the agents can follow a predefined script in order to be independent from the software producer site. Once in a client host, the agent requests the appropriate software components from the software producer, which, in this case, works as a software artifact repository.

5 Related Work

There are few approaches in the research literature that addresses the problem of software deployment using the mobile agent paradigm. During the writing of this paper, the only system that provided such facility, and was known by the author, was the Software Dock project, described as follows.

This section describes the software Dock System, providing some comments and suggestions at the end.

5.1 Software Dock

The Software Dock, developed by Hall et al. [HHHW97], uses the mobile agent paradigm for software deployment. It defines two main components: the *release dock*, representing the software producer, and the *field dock*, representing the software consumer. The mobile agents perform specific software deployment activities between these two components.

5.1.1 Main Components

The **release dock** works as a release repository for the software systems provided by the software producer. There is one release dock per software producing organization. The release dock provides a web interface in which users can select software versions and components to update.

Within the release dock, each software release is described using standard deployment schema. Each software release is accompanied with generic agents that perform the software deployment activities by interpreting the description of the software release.

The release dock advertises the field docks when new updates and changes are performed in the software release. The notifications are conveyed using an event notification service. In order to be notified the field docks must subscribe within this event service.

The **field dock** is a server executing in each consumer host. It provides information about the consumer side resources and configuration. The agents interact with the local host through the software

dock interface. This interface provides capabilities to query and examine the resources and configuration of the customer site in a standard way.

Both the release and the field docks have a registry database. In the release dock, the registry provides a list of available software releases, while in the field docks, the registry provides access to consumer side information. The registry follows a standard structure, providing a name space of attributes used by the software dock deployment scripts.

5.1.2 Software Deployment

Agents use information in deployable software descriptions (DSDs) in order to perform their activities. The agents are generated in the corresponding release dock (home site). There is one agent for each of the deployment activities described in section 2.1.2, these agents are specially designed for each deployment activity, performing generic procedures based on the DSD models. The agents migrate from the release dock server to the field dock of the host in which the activity will be performed. Once in the field dock, the agent performs its corresponding activity according to his script. After deployed, the agent can subscribe to receive events from its corresponding home site (the release dock). Agents can also communicate with each other using Internet standard protocols. Once in the dock field, the agent can request other agents according to the necessary activities to be performed.

For example, once the installation agent moves to the field dock, it performs the necessary configuration in the local system and requests, according to its script, a set of artifacts from his respective release dock. It cans also requests for other agents, in order to perform the other deployment activities.

5.1.3 Deployable Software Description Format

The DSD (Deployable Software Description) format is a format used by the software dock to describe the software system to be deployed. It models a software system based on properties and the proper configuration of those properties. It was specially designed for software dock project and allows the specification of:

- **Configurations**, allowing the description of component revisions and variants in terms of software system families;
- **Logical assertions** that have to be true in order to allow the installation of the system, for example, the presence of an specific hardware or operating system;
- **Dependencies** among software components (artifacts) and
- **Activities,** external applications to be executed during the deployment process.

The applicable schema elements for a software release are computable via guard conditions that are dispersed through the DSD specification.

5.1.4 Comments

Agents could use a generic script, and be responsible for the whole deployment process. This approach could result in the economy of the customer site resources, as less memory could be used. The delays associated to the agent transfer could be also avoided.

The software dock could allow the remote reconfiguration of the agents as a way to allow the dynamic change of deployment polices and procedures.

The DSD language does not allow the specification of coordination and of scheduled activities described in the section 2.1.3. On the other hand, it can be implemented using the event structure used by the software dock, in a way similar to the EVE workflow. In order to do so, the DSD language should be extended to express interdependency constraints among activities, as well as synchronization activities.

6 Conclusions

This paper presented the software deployment process, its phases, requirements and issues. In this context, the mobile agent paradigm was introduced, and presented as a possible approach to the software

deployment problem. Some examples, issues and problems related to the use of this paradigm were discussed, with some possible implementations suggested.

The use of workflow was proposes as a possible solution for the large-scale software deployment coordination. Some ideas and examples of its integration with the mobile agent paradigm and the software deployment issues were discussed.

This paper presents some general ideas, without providing further implementation and testing. However, the ideas presented here can foster some further research in the software deployment area.

The integration of software deployment, runtime change and configuration management can also be theme for further research [HHW98], specially the use of mobile agent paradigm as a support for these systems.

7 References

[BIJORD95] L. A. Barroso, S. Iman, J. Jeong, K. Öner, K. Ramamurthy and M. Dubois. RPM: A Rapid Prototyping Engine for Multiprocessor Systems. IEEE Computer, pp. 26-34, February 1995.

[CD99] J.E. Cook and J.A. Dage. Highly Reliable Upgrading of Components. In Proceedings of the 1999 International Conference on Software Engineering, May 1999.

[CFHHHW98] A. Carzaniga, A. Fuggetta, R.S. Hall, A. van der Hoek, D. Heimbigner, A.L. Wolf. A Characterization Framework for Software Deployment Technologies. Technical Report CU-CS-857-98, Dept. of Computer Science, University of Colorado, April 1998.

[CHK94] D. Chess, C. Harrison, and A. Kershenbaum. Mobile Agents: are they a good idea?. IBM Research Report, IBM T. J. Watson Research Center, Yourktown Heights, N.Y. RC 19887, December 1994.

[GHNCSE97] S. Green, L. Hurst , B. Nangle, P. Cunningham, F. Somers, and R. Evans. Software Agents: A review. Trinity College Dublin. May 1997.

[GT98] A. Geppert, D. Tombros. Event-based Distributed Workflow Execution with EVE, Proceedings Middleware'98, The Lake District, England, September 1998, pp. 427-442.

[HHHW97] R.S. Hall, D.M. Heimbigner, A. van der Hoek, and A.L. Wolf. The Software Dock: A Distributed, Agent-Based Software Deployment System. Technical Report CU-CS-832-97, Department of Computer Science, University of Colorado, Boulder, Colorado, February 1997

[HHW98] A. van der Hoek, D. Heimbigner, and A. L. Wolf. Investigating the Applicability of Architecture Description in Configuration Management and Software Deployment. Technical Report CU-CS-862-98, Department of Computer Science, University of Colorado, Boulder, Colorado, September 1998.

[IBM98] RS/6000 System Management, 1998. http://www.austin.ibm.com/resource/aix_resource/Pubs/redbooks/ookscl16.html

[InstallShield] http://www.installshield.com/

[JB96] S. Jablonski, C. Bussler. Workflow Management - Modeling Concepts, Architecture and Implementation. International Thomson Computer Press, 1996.

[KLO97] G. Karjoth, D. Lange, and M. Oshima. A Security Model for Aglets. IEEE Internet Computing, July-August 1997, pp. 68 - 77.

[KT98] N. M. Karnik and A. R. Tripathi. Design Issues in Mobile-Agent Programming Systems. IEEE Concurrency, July-September 1998.

[ObjectSpace97] ObjectSpace: ObjectSpace Voyager Core Package Technical Overview. Tech. Repot. ObjectSpace Inc. Dallas, 1997. http://www.objectspace.com.

[RGK97] D. Rus, R. Gray, and D. Kotz. Transportable Information Agents. Proceedings of the first ACM international conference on Autonomous agents, 1997, pp. 228 – 236.

[SWME00] Silva Filho R. S., Wainer J., E. R. M. Madeira, Ellis, C. - CORBA Based Architecture for Large Scale Workflow: Special Issue on Autonomous Decentralized Systems of the IEICE Transactions on Communications, Tokyo, Japan, Vol. E83-B, No. 5.

May 2000, pp.988-998.

[TGD97] D. Tombros, A. Geppert, and K. Dittrich. The broker/service model for the design of cooperative process-oriented environments. Technical report. University of Zürich, 1997.
ftp://ftp.ifi.unizh.ch/pub/techreports/TR-97/ifi-97.o6.os.gz

[WFMC96] Workflow Management Colatition. Terminology & Glossary, Version 2.0. WFMC-TC-1011, Jun. 1996.