

# Demo Proposal - dEVOLVe: Middleware Support for Application Deployment and Maintenance in Distributed EVOLVing Environments

Bart Elen, Sam Michiels, Pierre Verbaeten, Wouter Joosen  
IBBT-DistriNet, Department of Computer Science, K.U.Leuven  
Celestijnenlaan 200A, B-3001 Leuven, Belgium  
bart.elen@cs.kuleuven.be

## ABSTRACT

Deploying and maintaining non-distributed applications currently are easy tasks which often only require clicking a setup.exe icon. However, the deployment and maintenance of distributed applications in dynamic operational environments are significantly more complex.

First, application software needs to be deployed, configured and updated on all devices of the operational environment. Second, the application composition often needs to be adapted when devices are added or removed, when network connections are broken or re-established, and when devices are adapted. Third, we must check that the dependencies between the application software on the different devices are met at all times.

In this demo we present dEVOLVe, an extension for the OSGi framework which hides the complexities of dynamic distributed environments. Making distributed applications almost just as easy to deploy and maintain as non-distributed applications.

## Keywords

Application deployment, application maintenance, distributed applications, dynamic operational environments

## 1. INTRODUCTION

A large amount of devices with computing power (cars, VCRs, cell phones, ...) have been introduced on the market during the last decades. A new trend is that those devices are being equipped with network connectivity. Those newly formed distributed environments provide a promising platform for new distributed applications.

We focus on the 'deployment and maintenance challenges' that distributed applications are facing today. Non-distributed applications can be deployed in a relatively easy way. Often, the user can deploy or update the application by simply clicking a setup.exe icon. Deploying a distributed application is a lot more challenging:

1. **Distribution:** The application software needs to be installed and updated on multiple devices. Further, the application software needs to be configured to set up connections with the application software on the other devices. Both the installation and configuration of application software will require a large amount of work when hundreds of devices are involved.
2. **Environment dynamics:** The distributed operational environment is not static. It changes when devices are added to or removed from it, when network connections are broken or fixed and when devices are adapted (e.g. get an upgrade of their system software). All those adaptations may require a change of the application software or its configuration. In dynamic environments (e.g. with mobile devices), the maintenance task can become very hard.
3. **Remote dependencies:** Dependencies exist between the application software on the different devices. Remote dependencies can be broken, for instance, when the application software is updated on some devices, but not on all. The user has to make sure those dependencies are fulfilled at all times to allow a correct operation of the application. When lots of devices are used and the dynamics of the operational environment is high, manually checking those dependencies may no longer be feasible.

Typically, the end user has to take care of those challenges. We have build a middleware solution, called dEVOLVe, which frees the user from these tasks. It reduces the complexity gap between the deployment of distributed and non-distributed applications.

## 2. CHOSEN APPROACH

We reduce the complexity of application deployment and maintenance by using the following approach:

1. We want to adapt the application composition at runtime towards its changing distributed operational environment. To allow these dynamics, we choose to develop the distributed applications in a **component oriented way** with **service bindings** between the components.
2. Towards the user, we want to make the deployment of a distributed application as easy as the deployment of a non-distributed application. To make this possible,

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

*MINEMA '08*, March 31-April 1, 2008 Glasgow, Scotland  
Copyright 2008 ACM XXX-X-XXXXX-XXX-X/XX/XX ...\$5.00.

we choose to make the user only responsible for the deployment of a single, local, application component. We call this application component the **root-component** of the distributed application.

3. All other application components should be **automatically deployed** in the distributed operational environment. To realize this, we describe both local and remote dependencies between the application components in a **declarative** way. On the different devices, the middleware platform will select and deploy the application components best fitted to resolve the dependencies of the already selected application components.

We want to allow the user to install and update a complete distributed application by simply dragging and dropping application software and its updates towards the rREPOSITORY folder, which represents the distributed operational environment. Figure 1 shows such a rREPOSITORY folder with the Hello World and Distributed Movie Player applications. The user can uninstall the distributed application from the network by removing its application software from the rREPOSITORY folder. The rREPOSITORY folder also contains the `repository.xml` file used by OBR [1] and the `groupsConfiguration` file which will be discussed in section 3.2.



Figure 1: dEVOLVE rREPOSITORY folder

### 3. DEVOLVE

We implement our chosen approach in a middleware solution called dEVOLVE which is freely available on the project website: <http://www.cs.kuleuven.be/~barte/devolve>. dEVOLVE extends the OSGi framework with the following key functions:

#### 3.1 Remote declarative dependencies

The OSGi service platform has support for the description and wiring of service, package and bundle dependencies within a single VM. However, describing declarative dependencies with remote devices is not supported. Therefore, the dEVOLVE component adds support for the description of remote service, package and bundle dependencies in the bundle manifest file (Fig. 2). Remote variants are used of existing OSGi dependency headers. The 'group' argument of those manifest headers will be discussed in the next subsection.

dEVOLVE parses the manifest files of the used application components and tries to resolve their remote dependencies by installing the needed application components on the remote devices with the help of OBR [1].

```
Remote-Service:ServiceName; group="roleName(s)"
Remote-Package:PackageName; group="roleName(s)"
Remote-Bundle:BundleName; group="roleName(s)"
```

Figure 2: Remote dependency manifest headers

#### 3.2 Role assignment

Devices can be added and removed from the network environment. Because of this, application developers can not describe the devices themselves in the remote dependencies. To deal with this problem, we use roles to realize weak couplings with remote devices. The developers can describe dependencies with groups of remote devices fulfilling certain roles. They do not need to know which devices will fulfill these roles, and devices can be assigned and removed at runtime to each role. The assignment of roles to the devices is considered a task of the local network manager who may choose to automate this task with a role assignment algorithm [2]. dEVOLVE enables addressing the group of devices fulfilling a certain role. Our current implementation uses a `groupsConfiguration` file in the rREPOSITORY folder (fig. 1) of the deployment server to assign roles to devices. Figure 3 shows an example of a `groupsConfiguration` file.

```
group:advertisementControl:pc1.delijn.be
group:advertisementPlayer:bus1.delijn.be,
bus2.delijn.be, bus3.delijn.be
```

Figure 3: groupsConfiguration file

#### 3.3 Remote services

To make remote services locally available, dEVOLVE generates a proxy object for each remote service and registers it as a local service. A simple remote method invocation mechanism is used to handle the remote service calls. Conceptually, this is the same approach as used by R-OSGi [3].

#### 3.4 Environment evolution

The middleware platform must be able to deal with changes in the operational environment. A very robust technique to realize this is a periodical recomposition of the complete distributed application. dEVOLVE generates an application heartbeat signal for each locally installed application-root component, which triggers the recomposition. This application heartbeat is a periodic signal which starts from the application-root component and travels through the complete distributed application by following the dependencies between the application components. On each device where this heartbeat passes by, dEVOLVE reacts on this signal by checking if the current application composition is still the optimal one to satisfy the dependencies. When a better fitted application composition is discovered, the application will be adapted.

### 4. DEMONSTRATION

During this demonstration, we will show how well dEVOLVE is able to close the complexity gap between the deployment of distributed and non-distributed applications. We will demonstrate the capabilities of dEVOLVE with a distributed version of the Video Lan Client movie player (fig

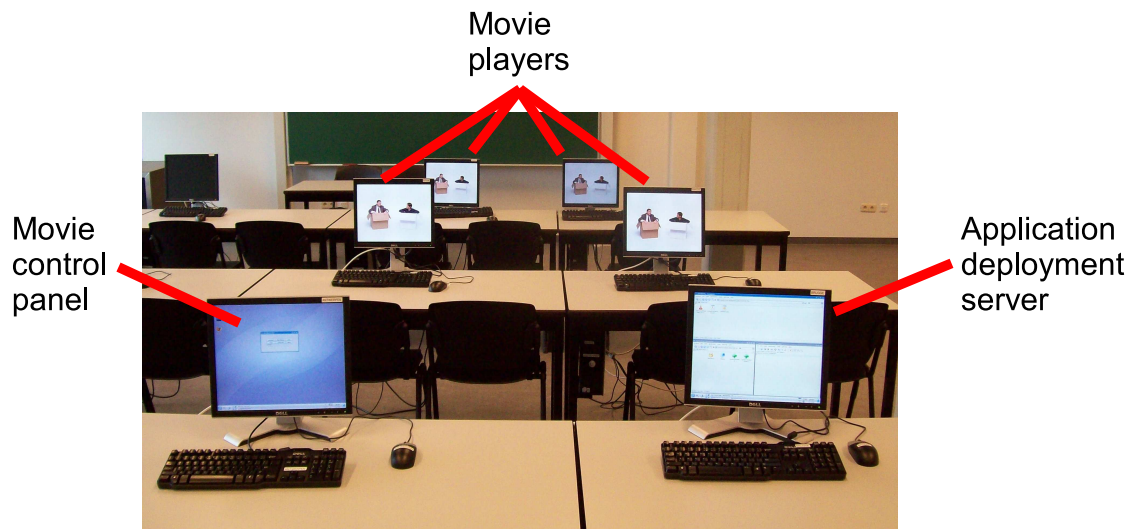


Figure 4: Distributed movie player deployed in PC lab

4), and a distributed version of the hello world application. Following capabilities will be demonstrated:

1. We will deploy distributed applications on multiple devices with a single user command. The same way, we will uninstall complete distributed applications with a single user command.
2. We will update parts of the application software with a single user command. The middleware will make sure the updates are only installed where needed. Further, the middleware will check if the dependencies between the application software on the different devices are still fulfilled and will try to fix them if needed.
3. We will demonstrate how dEVOLVE deals with the dynamics of its operational environment: We will add devices, change the roles fulfilled by the devices, and disconnect and re-connect network connections. The user is here freed from the maintenance tasks caused by these dynamics.
4. We will demonstrate that dEVOLVE is able to realize its task with a minimal impact on the application developer. dEVOLVE does NOT require the implementation of certain interfaces or the usage of a new XML language. The approach used by dEVOLVE only requires applications to be developed as a number of deployable units for which the bundle, package and service dependencies and capabilities are described in a declarative way. The components developed in the OSGi community do already fit those requirements. The only changes are that an application is no longer implemented in a single bundle, that the declarative description of the service dependencies is now mandatory instead of optional, and also that remote dependencies can be used.

## 5. CONCLUSION

We can conclude that this demo will demonstrate that by adding remote declarative dependencies and a mid-

dleware layer which automatically resolves those dependencies it becomes possible to build distributed applications which are almost just as easy to deploy and maintain as non-distributed applications. Even in dynamic network environments and with minimal additional effort for the application developer.

## 6. REFERENCES

- [1] Osgi alliance rfc-0112, osgi bundle repository, [http://www2.osgi.org/div/rfc-0112\\_bundlerepository.pdf](http://www2.osgi.org/div/rfc-0112_bundlerepository.pdf).
- [2] C. Frank and K. Römer. Algorithms for generic role assignment in wireless sensor networks. In *SenSys '05: Proceedings of the 3rd international conference on Embedded networked sensor systems*, pages 230–242, New York, NY, USA, 2005. ACM Press.
- [3] J. S. Rellermeier, G. Alonso, and T. Roscoe. R-osgi: Distributed applications through software modularization. In *Proceedings of the ACM/IFIP/USENIX 8th International Middleware Conference (Middleware 2007)*, November 2007.