

Combining Software Components and Mobile Agents

M.Amor, M.Pinto, L. Fuentes and J.M.Troya

Depto. de Lenguajes y Ciencias de la Computación. Universidad de Málaga
Campus de Teatinos. E29071-Málaga (SPAIN)
{pinilla,pinto,lff,troya}@lcc.uma.es

Abstract. We present a first approach that combines the mobile agent and the compositional paradigms into a new agent-based compositional model. The aim of this work is to explore the capabilities of both paradigms in improving the design and development of open and distributed systems. Our goal is to add compositional characteristics to mobile agents by defining an agent-based compositional model. We take advantage of the mobility and the composition capability by applying this model to the problem of searching software components in partially-instantiated applications, both in design and run time. That is, the abstract components of an architectural design of an application can be fulfilled with concrete ones by retrieving them from the Internet.

1 Introduction

Nowadays, as a result of the growing use of the Web and networks, the complexity of software systems is increasing. Due to the heterogeneous nature of the distributed environments where they should run, these systems need to model new kinds of interactions, and also they may be able to be adapted to changing requirements and new trends. In order to cope with this complexity we need new paradigms that facilitate the design of distributed and open systems in a better way than the object-oriented paradigm.

Having into account actual running trends of the software engineering, two paradigms are applicant for this purpose, the component and agent-based paradigms. Both of them seem to represent an evolution of the object-oriented paradigm. However, each paradigm focuses on different aspects of distributed applications and so they might be complementary.

The mobile agent paradigm is especially attractive to perform complex, tedious or repetitive tasks in open and dynamic systems [SD98]. Agent-based applications are develop specially for dynamic, distributed, open and heterogeneous environments, such as the Internet. The basic entity for designing and building this kind of applications is the agent, which can be seen at some scale, like an active object. Therefore, the mobile agent paradigm may represent another step in the evolution of the object-oriented paradigm [Ven97].

* This research was funded in part by the CICYT under grant TIC99-1083-C02-01, and also by the telecommunication organization “*Fundación Retevisión*”.

Although, a mobile agent may be viewed as an object, because it has a state, behaviour and identity, and also it has location, that may change during its lifetime. Due to this feature, mobile agent paradigm represents a design pattern, which is considered useful to develop dynamic and distributed applications. Some of the main benefits that mobile agents offer in distributed environments are [LO99]: they improve system performance, because they reduce the network load, and they let to overcome network latency in critical real-time systems. Also they encapsulate protocols that can easily evolve to new requirements. Although current trends on Internet computing lead to the use of mobile agents, mainly to perform searching tasks, distributed information retrieval and personal assistance, some technical and non-technical hurdles remain yet, since in many aspects is still a novel paradigm [KG99]. Some of these hurdles might be clarified before mobile agents could be widely used.

The component-oriented paradigm is also considered as another step in the evolution of the object-oriented paradigm. Objects and components both make their services available through standard interfaces. But the software component definition covers also aspects such as semantic of composition, introspection, binary interfaces, and also, component market related aspects such as reutilization of third parties components (COTS) and the independent deployment issue [Syp98]. Component paradigm is gaining acceptance as the demand of assembling systems using components increase the productivity and decrease the cost of developing software, representing in some manner, the industrialisation of software development [Slo99].

Therefore, both these paradigms exhibit features that seem to make them appropriate for designing complex systems in an easy way. Mobile agents might be an effective choice as they provide a single, general framework in which distributed applications can be implemented efficiently and easily, but present some drawbacks [KG99]. Our goal is to define a new model taking advantage of the features of both paradigms for designing open distributed systems.

We have a wide experience in componentware developing MultiTEL [FT99], a compositional framework of the domain of multimedia and collaborative services. But we have found that some of the agents' characteristics such as the mobility and autonomy could endow more flexibility to the component-based design. The architecture of the MultiTEL framework follows a compositional model, which defines standard components and common patterns of user interactions as connectors from the multimedia services domain. Once the coordination patterns have been identified, these can be tailored to obtain new services also reusing components, thus decreasing the development time of multimedia services.

In this first approach, we present an agent-based compositional model, which has components and compositional agents. We have defined the MultiTEL2 platform that provides a distributed environment for supporting the execution of applications based in the agent-based compositional model. By merging both paradigms, we aim to design more open application architectures, facilitating the construction of customisable applications over a distributed platform and providing an open market of components and agents-off-the-shell (like COTS).

One of the main contributions of our work is that we have applied this model to develop and bind partially-instantiated applications, where abstract components can be dynamically instantiated with real software components that can be obtained at

runtime from the net. Each abstract component has the ability of a search mobile agent encapsulating the searching criteria (e.g. incoming and outgoing messages defined as part of a component interface) for searching plug-ins in specific locations. An interesting and real example that can take advantage of this feature is retrieving a the software component that models a network camera connected to the Internet and managed from an unknown host. This kind of camera has a with built-in web server, and its own IP address, which means that it is not plugged to a host for the capture and transfer of images. Therefore, the component can be at any host inside the network. By this way, at runtime, a mobile agent obtains this component, and the network camera can be used inside the application like a local device, without a previous knowledge about where is the network camera or which is its IP address.

2 Components vs. Agents

At present, software engineer experts are discussing about the issue “agents versus objects”. However, we consider that this discussion should be moved to the component field, and confront agents and components, since we consider that both paradigms are at the same level of abstraction and also can be implemented using the object-oriented paradigm. Firstly, we are going to discuss what components may offer to agents.

Getting a look to the literature about components, we cannot find a consensus about what are the characteristics that a component might offer, but there are some of them that are present in almost every component definition. *Introspection* lets components to ‘say something’ about their behaviour and data. This property, which is not found on current agent systems, would allow agents to interoperate dynamically, and it would let them to afford, for example, a protocol based on introspection.

Dynamic composition means that components can be linked at runtime, and it is possible if components provide some information about them. This information must be included in components IDLs, which describe not only their incoming messages but also their outgoing messages [ÓB97]. Agents also show this property, as they can communicate with other agents and their environment, but in a predetermined way. An agent does not know who or when are they going to communicate, but they know how. For example, a search agent, which interacts to a predetermined kind of database, knows how to interact (the consult and the result are given in a well-known format). There is no flexibility. If the syntax changes or if it wants to dialogue with unknown and third party agents, the agent is not able to adapt its behaviour to it. Agent-based designs could be more flexible if the interaction is determined at runtime, when agents meet and after a negotiation about establish how they are going to interact.

Other property of components, is the *event-handling*. This mechanism affords a kind of dynamic composition via an anonymous interaction. The event sender does not need to know who could be the receiver of the event, which is established by composition. This mechanism offers flexibility to agent models, as an agent request can be handled like an event received from the environment. That is, if an agent needs

a service, it throws a request, which will be caught by other unknown agent. Some agent communication systems based on tuple spaces support a kind of anonymous interaction to coordinate mobile agents [OZ98]. However, at componentware field this has greater semantic load because the matching between input and output requests is established according to an architectural pattern.

In addition, components must have explicit context dependencies and have to provide binary interfaces, that is, a component might be delivered in a binary form to be ready for use without need of exploration of the implementation code. Furthermore, a component should be deployed independently, because software components will be sell inside a global component market. To carry out this issue, component needs to have explicit and well-documented context dependencies in order to be reused in any environment. Agents should also present these properties, as it would be possible to provide an open agent market, where software developers can make use of agent-based solution and reuse them. The benefits would be greater if agent's developers would come to an agreement about standards like agent communication language, agent mobility, and so on.

On the other hand, the agent paradigm has become an increasing and important area of software engineering research. Application domains in which agent-based solutions are being applied and investigated fill different disciplines, such as AI, Genetics, Robotics; but agents also are considered a promising technology for the design and development of distributed systems [Nwa96]. In distributed and open systems, executable code travels with mobile agents so an agent can be considered as a mobile object enriched with other features [C+94]. The goal of agents is to delegate and automate tasks, most of them tedious, in an asynchronous way. That is the reason why agents are being applied to the Web, for instance in searching and filtering information, intelligent electronic mail, electronic commerce, mobile computing and telecommunications [C+95]. We consider that the main characteristic that differentiates the agents from other design paradigms, included compositional paradigm, is its autonomy.

Agents are autonomous, meaning that they act independently in order to perform their responsibilities, although an agent may interact with others. In addition, it can control how and when it answers to requests, and it has the ability to react to changing conditions in the environment without receiving a specific request. Others agent's attributes that could characterise an agent are: the mobility, whether it can move from a host to another; intelligence, if it can reason; adaptability, if it can learn. Also, agents can have knowledge about some domain; continuity, persisting over time. Unlike, components have a meaning inside the context of a domain-specific framework. The reutilization of a standalone component is not practical, it is desired to reuse a component inside a compositional framework, as part of a task. However, components are passive entities that only act when they receive input messages or events, and its behaviour depends on them. Agent's autonomy makes a component to be an active entity.

In the next section, we present a first approach that merge and integrate both paradigms, to obtain a new agent-based compositional model. Our aim is that applications derived from this model can take advantage of both paradigms.

3 The Agent-based Compositional Model

The proposed agent-based compositional model is based on the component model defined by MultiTEL [FT99]. Since we combine agents with components our agent-component model has two different types of entities: components and c-agents. Components model real entities of the system and encapsulate computation, while c-agents are active entities oriented to a task.

3.1 Components and C-Agents

Components are passive entities that model resources and real world entities like a camera, a printer, or a database. The IDL of a component defines the list of messages that can be received or sent by the component. The reception of a message causes changes in the internal state of components that are reported to the environment by sending one or more messages specified as part of its IDL. This means that an IDL does not only include information about the incoming messages as usual and also the outgoing ones.

Components have an identity independent of the different interactions in which they could engage. They do not know how the reception of a message and its computation influence the rest of the system. This characteristic contributes to have a component market from third parties (COTS) where components can be reused in new services without knowing their implementation, only knowing their interfaces, studying the services that they offers and the messages that they may send. A component may define private components for modularization purposes, and in this case must include in its IDL the subcomponents' incoming and outgoing messages. Incoming ones are delegated to the suitable subcomponent.

C-agents are autonomous active mobile entities, which are in charge of a task. C-agents can navigate through a set of places. In opposite with components, c-agents know how the reception of a message and its computation influence the rest of the system, and also indicate some information about the receiver of an emitted message. C-agents also provide an IDL that specifies the incoming and outgoing messages. Apart from the dialogue that may be established between c-agents and components, c-agents can communicate between them through the standard XML language.

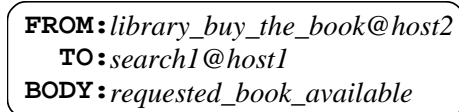
Both, components and c-agents, have a unique identity, like a URL, assigned on creation, which lets distinguish itself from others components and c-agents. Also, components and c-agents are characterised by a role. For components, the role is an identifier that represents the resource that models i.e. the component *camera* models a camera device. For c-agents, the role represents a set of abilities, i.e. *info-search* c-agent, it is a c-agent that can search information. The role of components and c-agents is useful in the context of application architecture, because by publishing the role, a component advertises the resource that models or represent, and a c-agent advertises that it can perform a collection of operations, tasks or services. Many components or c-agents with the same role can be running inside the application. In some way, the role of a component or of a c-agent is characterised by its IDL, that is, its outgoing and incoming messages. A component *printer* will contain in its incoming messages

something like *print_job*, or *ready*, related with a printer device. All components that play the same role or have the same functionality should implement IDLs containing the required input and output messages. Likewise, a c-agent with the role *search* will receive message to search for, and emit messages to return a search result.

3.2 Interaction: Message Delivery

Communication between components and c-agents is established by message passing. Since components are passive entities they only react to c-agents requests or they ask unknown c-agents for a service by a message in an asynchronous way.

As shown in figure 1, messages enclose the identifiers of the sender (from) and the recipient (to).



```
FROM: library_buy_the_book@host2
TO: search1@host1
BODY: requested_book_available
```

figure 1. Message format.

However, the field *to* that contains the recipient of the message does not need to have a reference of a component or c-agent identifier. There are three way of specifying the component or c-agent target of the message. Depending on the information about the recipient attached to the outgoing messages, the model defines three types of message delivery:

- *Role-based*. The recipient of the message can specify receiver's role. For example, a c-agent with the address *search1@host1* is searching among digital libraries for a certain book. It has just arrived to a new host and wants to send a message to components *books_database*, asking for the book. In this case, the c-agent does not have to know the identifier of the local component, but it knows the role that characterises the component. Also, since more that one component *book_database* can be host at the same location, the message will be delivered to all.

- *Identity-based*. The receiver's identifier is specified when the sender emits the message. Component and c-agents can obtain the identifier of another component or c-agent from a previous incoming message, which allows answering a request. Following the previous example, the *books_database* component *library_buy_the_book@host2* can send an answer message to the c-agent *search1@host1* address, notifying that the book was available.

- *Content-based*. When any information about the recipient is given with the message, the recipient or recipients of the message will be all the components and c-agents whose IDL contain the message, that is, every component that offer the required service. For example, a component that models a printer can send a message to *anybody*, without putting an address, an identifier or a role in the field *TO* of the message, indicating that it is ready. This means that the target c-agents or components do not have to play the same role, only need to include the required message.

On the other hand, communication between c-agents is defined in a different manner, since it involves protocol specification and also c-agents can take part in

more complex interactions, like negotiation, complex queries, or exchange information. In order to model the complexity and the heterogeneity of message formats c-agents talk to one another by specifying their requests in XML, a common understood language. By XML we can describe a protocol, a data structure or a script. By this way a c-agent can define a protocol, or it may send a message not contained in the IDL of the target c-agent and interchange information represented in an independent format letting to describe any data content. In addition, XML lets to define new labels and attributes. For example, a c-agent has a search result that is stored in a data structure, and this result is requested by another c-agent. The interchange will be performed expressing the data structure and its content by XML, as the target c-agent does not have to know how it is the internal data structure of the other c-agent. Also, the c-agent can request the result, including the required format of the information that has to be returned.

4 MultiTEL2 Platform

MultiTEL2 is a middleware platform that offers the services required for the composition of applications based on the model defined in last section, and provide support for mobile agent characteristics of c-agents. Components and c-agents will run on a distributed platform that supports the communication mechanisms based on message delivery, defined by the model. MultiTEL2 provides common services for the distributed execution of applications and lets components and c-agents coexist and cooperate.

Figure 2 shows the MultiTEL2 platform, which provides a Component and C-Agent Creation Service, Naming & Locating Service, a Distributed Component Repository and a C-Agent Migration Facility that will be explained in the following sections.

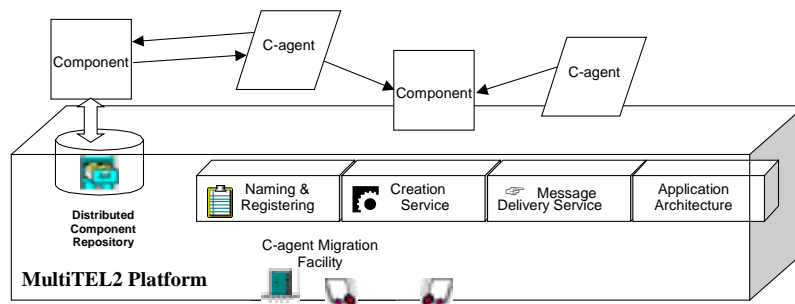


Figure 2. MultiTEL2 platform.

The platform and the agent-based compositional model that support migration is developed in Java, so components and c-agents are platform independent. In addition, the migration implemented, by transferring the bytecode with the Java class and the object state, removes the need to recompile the agent on arrival at a new host

4.1 Application Architecture

Standalone components and c-agents are useful when they take part of an application. In order to define the architecture of an application based on components and c-agents, it is needed to specify the requirements imposed to components and c-agents to take part of an application. These requirements are expressed in terms of the role and the incoming and outgoing messages. The data structure that holds the application architecture is maintained at the MultiTEL2 platform, and it can be consulted by the platform services.

This structure shows how can be performed the composition between components and c-agents. For example, the application that buys a book has to be composed by c-agents *info-search* and components *book database*. Also, some of the incoming messages of the component need to appear as outgoing messages in the c-agents IDL and vice versa.

The architecture can be defined and stored in the data structure shown in Figure 3, where, for each component and c-agent that participates in the application, it is determined its role, the set of its incoming and outgoing messages, and its implementation. Also, for each c-agent the architecture specifies the components and c-agents that it may interact.

Role	Implementation	Incoming messages	Outgoing messages	Related to
Book_database	Oracle_library	Database_input	Database_output	
Inf_Search	Book_searcherv1	Search_request	Search_result	Book_database
Inf_Search	Book_searcherv2	Search_request	Search_result	Book_database

Figure 3. Application architecture.

4.2 Component and C-Agent Creation Service

This service attends requests from c-agents to create components and c-agents. The requests must specify the role of the component or the c-agent that have to be created, i.e. *create_component(camera)*, or *create_c-agent(search)*. The platform creates a component or a c-agent using the description given the application architecture, shown in Figure 2, where this service can map an implementation to the requested role. Components and c-agents are created dynamically along application execution.

4.3 Naming and Locating Service

The components and c-agents must have an own unique identifier to be univocally addressed. Once they are created, under a role inside an application, the platform assigns them an identifier. The service includes the host where it has been created as part of its identifier. The host where a c-agent is created is also known as the c-agent's home.

C-agents are mobile, which means that it can move through different hosts, and its location can change continuously. As we explained in last section, message delivery is determined by recipient, not by location, i.e. when a component sends a message to a c-agent, it only has to specify the c-agent's identifier, so the platform needs to maintain a data structure to register the last location of a c-agent. The actual location of a c-agent is updated at c-agent's home.

Figure 4 shows how the localisation of remote c-agents is performed. When the platform is requested to send a message (e.g. to "Search1@host1"), first it checks if the c-agent is running locally, consulting local context. If this checking is negative, the platform asks for the current location of the c-agent's home, in this case "host 1" (step 1). The c-agent's home consults a data structure (step 2) where actual location, which is "host 3", is registered, and returns that information (step 3). After receiving the actual, "host 2" will remit the message (step 4). Every time the c-agent moves to a new location, its home registers the new c-agent's location.

For every c-agent created on a host, the platform maintains a structure like it is shown in Figure 4 as *c-agent actual location info*, to maintain a c-agent migration registry for delivering messages to moving c-agents in a transparent way. This is not necessary for components, as they are not mobile.

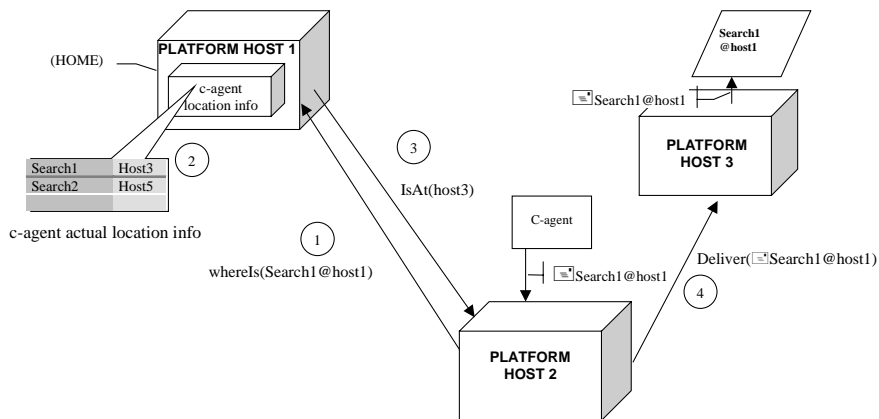


Figure 4. Localisation of mobile c-agents.

4.4 Agent Migration Service

The platform is able to freeze an executing c-agent, and transfer its code and its state to another host. Moreover, the platform is able to unfreeze a c-agent transferred from another host and allow it to resume execution as shown in Figure 5. The c-agent migration can be easily implemented using Java, through object serialization and reflexive programming. Due to this strong migration, the target host does not need to have the Java class that model the c-agent.

When a c-agent wants to migrate, the platform deals with notifying c-agent's home that it is going to move (step 1 in Figure 5). Then, it takes care of packaging the agent,

including its actual state (step 2), and sends it through the network to the destination host (step 3), which is in charge of unpacking the c-agent, and registering it into the local context (step 4). Then it notifies c-agent's home the new location and resumes its execution from its last state (step 5).

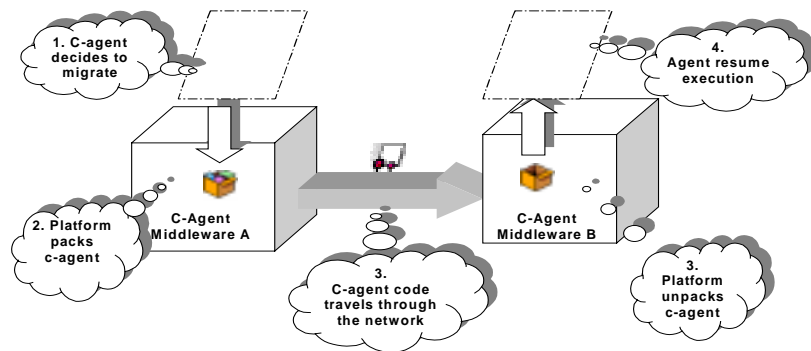


Figure 5. How Agent Migration is performed.

4.5 Message Delivery Service

Communication is performed by asynchronous message passing. The platform receives messages and delivers them to the right component or c-agent, like a post office. To deliver a role-based message, the platform first consults the local context, that is, the components and agents that are running locally and that fulfil the requested role. If there is no running component or c-agent fitting the role, it creates one.

Identity-based messages are delivered directly to the right component and c-agent. If they are not running in the local context, the platform deals with the remote delivery. In order to prevent message lost during migration, c-agent's home will queue messages. Once the migration is completed, the platform notifies it to its home host and then the home will dispatch the received messages to the c-agent at its new location.

Content-based messages are delivered to local components and c-agents. The platform only looks for components and c-agents that can receive the message, because it is contained in their incoming messages and also, are running locally. If the service does not find any applicant component or c-agent the message will not be delivered.

4.6 Distributed Component Repository

MultiTEL2 enforces to share and reuse components and c-agents. To make easier this task, the platform offers a Distributed Component Repository (DCR). In this way, application designers may reuse public software developed by others companies that conforms MultiTEL2 framework principles.

Designers may reuse a component or a c-agent in different contexts, so it is needed to provide information about its features and capabilities. This metadata, generally implemented in terms of IDL [KA98], shows component and c-agent's behaviour, and will help to classify their features easing their reuse in other contexts.

Following Web principle of information distribution, every MultiTEL2 machine has a Component Repository, organised in hierarchical domains. Each entry of the repository corresponds to an implementation of a component or a c-agent, and contains some information needed to know component and c-agent behaviour. The information available for a component is its IDL and a brief textual description of its behaviour. In the c-agents case, also include the task they can develop. For both we also include the assigned role that the component or the c-agent may perform in any service architecture.

5 An Agent-based Solution for Retrieving and Reusing Software Components

Classification and selection of reusable components are considered as a key success factor. Software component suppliers should provide standard documentation, to help customers in finding quickly the right component.

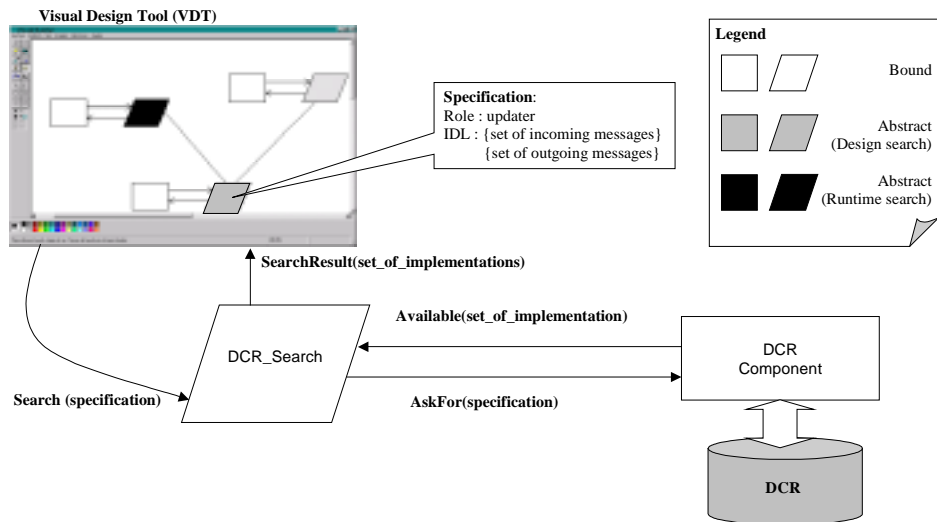


Figure 6. Agent-based search of a component at design time.

But, instead of performing an individual and manual search, we propose to delegate this task to c-agents. That is one of the purposes of c-agents: search and retrieval components and c-agents from DCR. The need of searching a component or a c-agent can arise at two stages of the software developing process, at design time and at run time. A kind of c-agents called *DCR_Search* will bring the desired software

component, directly to the MultiTEL2 platform. These are basic c-agents that perform a search through the repository and are included as part of the Visual Design Tool (VDT) for being used at design time (Figure 6).

As we explain in a previous section, application architecture is defined specifying components and c-agents' roles, and connections between them. The developer may design an application architecture by using a VDT, inserting components and c-agents, specifying the role they must perform inside the application and the connections between them. After that, the developer must choose an implementation that conforms all these requirements. Instead of implementing every component and c-agent, the developer can reuse other implementations, searching and consulting the components and c-agents available in the DCR.

But, instead of searching manually through the DCR, this task is performed automatically by a kind of c-agent called *DCR_Search*, shown in Figure 5. It carries out the requirement, that is, the role, the incoming and outgoing messages or a combination of both. The *grey hole* means that it corresponds to an abstract component or c-agent which implementation is not set yet, so the c-agent of the VDT, carrying the specification, searches for a suitable implementation to fill in the *hole*, saving developer's time. Once the c-agent finishes its search and presents its results, the developer will decide which component or c-agent will be used in the architecture by inspecting the information about the applicants retrieved by the c-agent.

We go further away, and we propose to delegate the searching of components or c-agents implementation until run time. Thus, the application will run over a partial-instantiated architecture, like a template, where the field implementation of some components or c-agents can be empty (in Figure 3) or a *black hole* of the VDT in Figure 6. When the application running on MultiTEL2 needs to instantiate a component or a c-agent, but its implementation is not specified in the application architecture, a kind of c-agent called *creator* is instantiated instead. This c-agent is a specialisation of the searcher c-agent used above, because it searches and also it takes the decision about the suitable implementation inside the result set. Once the c-agent finishes its search, it returns to the original host and creates an instance of the chosen component or c-agent. The user will not be aware of that different running instances of the application may have different component or c-agent implementations, since the c-agent search result can differ, but the restrictions imposed by the application architecture would be enough to guarantee that the implementation chosen by the *creator* c-agent at runtime will suit into the architecture requirements.

In addition, we can take another advantage from letting the application with an "incomplete" design. Let us take a provider of a product that has to be distributed to different vendors, which want to be notified about the last updates of the product. Instead of taking care of notifying the updates the provider puts the last version of every component and c-agent in its own DCR.

On the other side, we can take advantage of this solution to the provision of customisable software products. Different vendors provide the same product but they could be able to customise the component that provides the user interface. With our approach, the developer does not need to worry about it, leaving the decision about what implementation will be used until runtime. Every time the component is required, it will be found at the nearest DCR, for instance, the local DCR of the

provider. By this way, the same application could display different graphic interfaces or whatever, depending on the vendor the user is subscribed.

From the given solution, the applications frameworks partially instantiated, we can find another utility, very fashionable nowadays. The run time retrieval of software can be applied to plug-in multimedia devices without worrying about download the needed drivers. When a multimedia application needs to create a component that manages the camera, only has to look for the best implementation for that camera model. This runtime search can be applied to any multimedia device component, but also to any system property that could enhance the performance of the application.

6 Conclusions and Future Works

In this first approach, we have presented an agent-based compositional model that combines mobile agent and compositional paradigms. We have obtained benefits from merging both paradigms that may improve the design of open and distributed systems, especially those that are implemented above the Web. For instance, by applying mobile agent characteristics we can improve distributed searching tasks, and with componentware applications we can improve independent software components. By endowing the agent-based design with compositional issues, we have tried to make it a more flexible and powerful paradigm applicable to any distributed system. One of the most important contributions of this work is that due to the compositional characteristics of the model, users can develop partially-instantiated applications based on components and c-agents, which could be dynamically instantiated at runtime. The benefits are reflected directly on the final software solution as it can provide customisable plug-ins, or to have, every time the application is execute, the latest version available.

Our future goals are to complete the definition of the agent-based compositional model presented in this paper, and to specify it formally. A further goal of our research is to address the c-agents interoperability issue, add the introspection property to them by using XML, and build a working prototype, regarding open distributed systems features like scalability, security issues and fault tolerance.

References

- [C+94] D. Chess, et al., "Mobile Agents: Are They a Good Idea". *IBM Research Report*, 1994.
- [C+95] D. Chess, et al., "Itinerant Agent for Mobile Computing". *IBM Research Report*, 1995.
- [FT99] L. Fuentes, J.M. Troya, "MultiTel: Multimedia Telecommunication Services Framework". A chapter of the book *Domain Specific Application Frameworks*. Wiley & Sons. October 1999.
- [KA98] D. Krieger, R. Adler, "The Emergence of Distributed Component Platforms". *IEEE Computer*. March 1998.

- [KB98] D. Kafura, J.P. Briot, "Actors and Agents". *IEEE Concurrency*, pp. 24-29. April-June 1998.
- [KG99] D. Kotz, R. Gray, "Mobile Code: The Future of the Internet". *MAC3 on Autonomous Agents'99*. <http://mobility.lboro.ac.uk/MAC3/>. May 1999.
- [LO99] D. Lange, M. Oshima, "Seven good reason for Mobile Agents". *Communications of the ACM*, Vol. 42, No.3 pp. 88-89. March 1999.
- [Nwa96] H.S. Nwana, "Software Agents: An Overview". *Knowledge Engineering Review*, Vol. II, N° 3, pp.1-40. Cambridge University Press. September 1996.
- [ÓB97] Á. Ólafsson, D. Bryan, "On the need for required interfaces of components". *Special Issues in Object-Oriented Programming*, pp 159-165. Verlag Heidelberg. 1997.
- [OZ98] A. Omicini, F. Zambonelli, "Coordination of Mobile Agents for Information Systems: the TuCSoN Model". *6th AI*IA Convention*. 1998.
- [SD98] A. Silva, J. Delgado, "The Agent Pattern for Mobile Agent Systems". *3rd European Conference on Pattern Languages of Programming and Computing*. 1998.
- [Slo99] F. Slootman, "A blueprint for Component-Based Applications". *Compuware Corp.* http://www.compuware.com/products/uniface/station/reading/ind_blue.htm.
- [Syp98] C. Syperski, "Component Software. Beyond Object-Oriented Programming". *Addison-Wesley*, Reading Mass. 1998.
- [Ven97] B. Venners, "Solver Real Problems with aglets, a type of mobile agents". *JavaWorld*. <http://www.javaworld.com/javaworld/jw-05-1997/jw-05-agents.html>. May 1997.