

Remote Configuration of Agent-Based Component Systems

Rainer Weinreich and Reinhold Plösch, University of Linz, Austria

Abstract

The nearly omnipresence of the Internet and the steady increase of wireless computing and mobile devices require highly dynamic adaptable distributed system architectures. Building such architectures needs a combination of key concepts from component technology and distributed systems. Mobile agents provide this combination. We use mobile agents as the building blocks of a component-based system for remote supervision and control of both hard- and software in a distributed environment. In this paper we concentrate on the configuration of individual components and component relationships in our system. We identify requirements for remote configuration of agent-based component systems and discuss architectural and user interface related issues of our approaches. We use a code-on-demand approach for supporting elaborate user interfaces. We use a generative approach based on enhanced meta-information for reducing development effort. The presented approaches are applicable for remote configuration of component-based systems in general and consider additional requirements imposed through the use of mobile agent technology.

1 INTRODUCTION AND OVERVIEW

Distributed software architectures are currently increasingly influenced by two major technological movements—the Internet and pervasive computing (including wireless and mobile systems). In the last years, the Internet has mainly been used as the technological basis for creating the Web, a global hypertext and hypermedia network, enriched with simple interactive (HTML-based) services, like search engines, electronic shops, and electronic auctions. Currently, the Internet and its protocols are more and more becoming the infrastructural backbone for arbitrary services and systems. Nearly every distributed application is required to work in an Internet context or is based on standardized Internet protocols.

The Internet is also changing application deployment and maintenance. Internet-based deployment comprises not only the transfer and installation of software, but all activities from installation until deinstallation and removal of a software system at a consumer's site [1]. This includes tasks like remote activation, deactivation, configuration, reconfiguration, addition, removal, and update of software. All these

activities are not only performed for whole applications but also for individual components, and sometimes even at run-time. The result is the need for highly flexible and adaptable software architectures as well as the need for remote configuration and management tools.

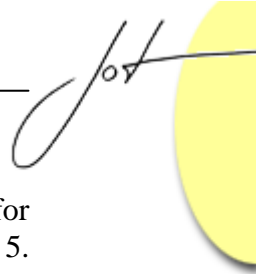
The second major technological movement is wireless and mobile computing, which makes further demands on distributed software architectures. Examples are adaptation to different environmental conditions, dynamic service discovery, scalability, robustness and security [2]. Remote configuration tasks may be performed using a whole range of potentially different end-user devices with dedicated user interfaces.

Many of the challenges stated above are addressed by component technology [3][4]. Component models [5] provide standards for component customization, communication, evolution and composition. Components are the basis for adaptable software architectures. Mobile agent technology has similar characteristics as component technology [6]. Nearly all distinguishing features of component systems that are standardized in general component models are equally important in mobile agent systems. However, mobile agent technology additionally emphasizes support for distribution, heterogeneity, adaptation to different environments, code mobility, and spontaneous computing. These features are especially important for the application domains outlined above. In fact, mobile agent platforms may be viewed as powerful and flexible component environments.

We use mobile agent technology as the basis for a flexible component system for remote diagnosis and monitoring of hard- and software resources in heterogeneous distributed environments. Currently the main usage areas are process automation systems though the system is not limited to this domain. A main characteristic of our system is its highly dynamic structure. Diagnosis and monitoring components may move within the network to their intended place of action, which is the hard- or software resource to be monitored or analyzed. This requires support for code mobility. Other features that are needed and supported by our system are dynamic service discovery, dynamic services, native-code management, multi-protocol remote access of various types of components, robustness, and security (see [7]).

A main feature of our system, which is also the topic of this paper, is remote configuration and management of monitoring and diagnosis components over Internet connections. Since the components of our system are mobile agents, we will use the terms component and agent interchangeably in the remainder of this paper. We have experimented with a number of approaches for remote configuration of individual agents and of system properties like agent relationships. While most of the explored techniques apply to remote configuration of components in general, some are specific to the characteristics of mobile agent systems.

The remainder of this paper is structured as follows: In Section 2 we describe requirements and solution options for the configuration of remote components including requirements that are specific for the configuration of mobile agents. In Sections 3 and 4 we present approaches for remote configuration that have been implemented in our system. The emphasis of this paper is on Section 3, which contains a discussion of



approaches for the configuration of individual agents. Section 4 outlines our approach for configuring system properties and structure. We describe related work in Section 5. Remote configuration over Internet connections needs security support. However, a discussion of security options and requirements is beyond the scope of this paper. A short overview of security support in our system can be found in [7].

2 CONFIGURATION REQUIREMENTS

In order to discuss typical requirements and approaches for configuring components and mobile agents, we first need to present different variants of system structures for remote configuration over the Internet. In its simplest form—depicted in Figure 1—the host for remote configuration is directly connected to a host where the components to be configured are installed and possibly activated.

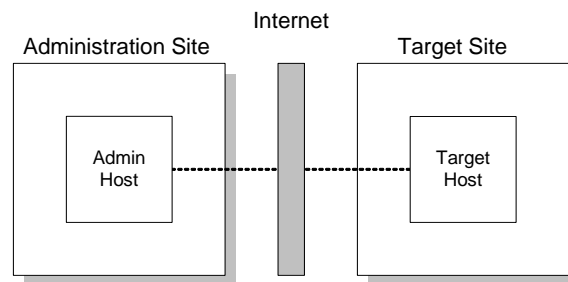


Fig 1: A simple system structure for remote configuration

We call the location where remote configuration tasks are performed by human operators the administration site. The administration site may be only one host or a network of hosts, which may all be used for configuration purposes. The location where the software is installed and running is called the target site. If the components to be configured are implementing the middle-tier of a three-tier application model, the target site might be a single computer with the application server hosting these components as shown in Figure 1.

A typical agent-based system, however, is a distributed system where the components to be configured are distributed to a number of hosts at the target site (T1, T2, ...) as depicted in Figure 2. Configuration may be performed from different hosts at the administration site (A1, A2, ...), also shown in Figure 2.

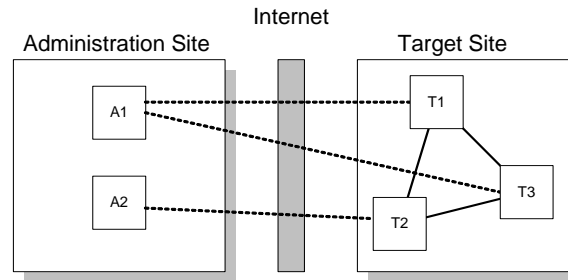


Fig 2: Configuring a distributed system from multiple hosts

Usually company networks are guarded by firewalls and not every host at the administration site may directly access the Internet. Likewise only selected hosts at the target site are visible to the Internet. Communication has to be routed through proxies (P) at the administration site and through dedicated entry points at the target site as depicted in Figure 3. In addition, the host acting as proxy in Figure 3 may also serve as administration server (AS) for centralized management of configuration tools and component repositories.

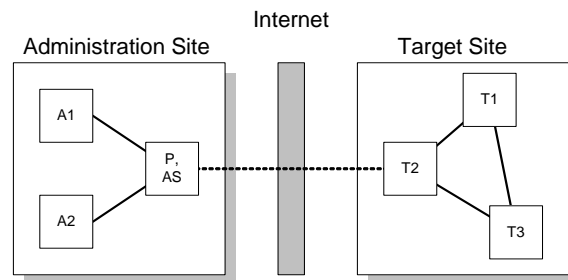
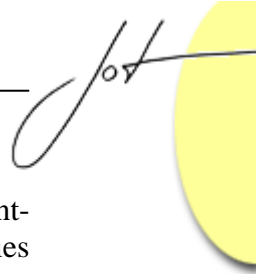


Fig 3: System structure with firewalls in mind

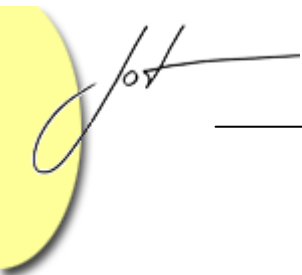
The presented system structures for remote configuration serve as the basis for the description of requirements on remote configuration systems in general and on our system in particular. Important requirements are:

- a) *Dynamic configuration of individual mobile agents and of the system structure.* We need to support the configuration of both individual mobile agents and general system properties and structure. System structure is defined through agent communication relationships. Parts of the structure may be defined through rather fixed relationships that can be changed manually. For example, our system allows the configuration of publish/subscriber relationships between agents. General system properties may be changed by configuring special agents that are responsible for distributing the information within the target site (see Section 4).

Dynamic configuration refers to the ability to configure the system while it is up and running. This requires a highly dynamic system architecture which allows

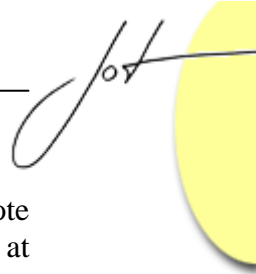


- adding and removing components at run-time – a natural feature of any agent-based system. However, it also requires special protocols to change the properties of individual agents. Mobile agents are active objects encapsulating their own thread of control. It is not possible to change a certain property at any time and sometimes it is not possible to change an agent's properties at all. This has to be taken into account when designing protocols for updating agent state at run-time.
- b) *Minimal administration of configuration tools at administration site:* This requirement refers to the administrative effort that is involved in managing the configuration tools and repositories at the administration site. Changes or updates of the tools itself should require no or only minimal activities at the configuration hosts (see A1, A2, ... in Figure 3). Pre-installing the configuration tools at each configuration host is not desirable. Centralized configuration can be achieved by loading the tools on demand from a central administration server (see AS in Figure 3). This requires a dedicated run-time environment at each host. In the ideal case such an environment is a standard equipment of the client host, like web browsers, which are able to host HTML-based user interfaces. If HTML-based user interfaces are not powerful enough, additional environments for hosting user interfaces based on other technologies have to be preinstalled at each configuration host. Examples are the Java Plug-In [8] and Java Web Start [9] technologies for Java-based user interfaces. This is still preferable to installing the application at each host, since update and other changes of the configuration tools require no management activity at the client hosts.
 - c) *Support for different types of configuration clients:* The rise of mobile and wireless computing is leading to a large number of different end-user devices with different display sizes and capabilities. The system structure at the administration site—as depicted in Figure 3—is also appropriate for supporting different kinds of configuration clients (A1, A2, ... in the figure). An administration server (AS) could provide different user interfaces depending on the end-user device used for configuration. For example, it might provide WML-pages for a WAP-enabled device [10].
 - d) *Loose coupling of tools at administration site and of components at target site:* Certain implementation decisions might lead to a tight coupling of the tools at the administration site and of the agents at the target site. Tight coupling may be the result of using a platform specific type system for configuration data, since this presumes that agents and tools are based on the same platform. For example, if configuration data is represented as Java objects both tools and agents need to be Java-based. Platform independent data formats and type systems (e.g., based on XML) are more flexible, since tools and target components may be implemented in any language. However, such type systems may not be as expressive as platform-specific ones, confining agent properties to simpler data types with no associated behavior. In the case of agent-based systems one might be tempted to install an agent platform not only at the hosts of the target site but also at the hosts of the administration site. However, this also leads to tight coupling of administration site and target site since it assumes that the configuration tools are



only used for configuring agents of a particular agent platform. This rules out systems like ours, where one administration site is used for configuring multiple target sites with possibly different agent systems installed. We will present our solution to this problem in Section 3.1. In addition, the notion of migrating an agent to an administration host, changing its configuration and sending it back to the target site is often not feasible. Two problems that come immediately into mind are security and agent activity. A firewall aware system structure as depicted in Figure 3 would need flexible agent platforms that allow control of message routing. However, agent platforms usually support peer-to-peer communication as depicted in Figure 2. Also, firewall settings at the configuration site might not allow an agent entering the site at his will; most of the time even callbacks are denied. A further problem is that an agent is an active entity. It is often not possible to stop an agent's activity just for changing some configuration settings.

- e) *Evolution support*: In dynamically adaptable systems components (mobile agents in our case) are added, removed and replaced by newer versions over time. Multiple versions of the same component may exist simultaneously in the system. This is supported by mobile-agent systems, since features like code mobility require flexible mechanisms for code management. Typically the agent system provides separate name spaces for different agents and a code loader which makes sure that the code of different versions of the same agent type can be loaded at the same time [11]. From the configuration viewpoint we have to make sure that we are able to configure an agent at any time during its life time. Even if some agent code has been removed from the repository at the administration site or if it has long been replaced by newer versions there may still exist some instances of older versions at the target system, which need to be configured. The most obvious solution to this problem is to store the user interface code for configuring the properties of a particular agent with the agent itself at the target site. If the agent is to be configured, the user interface code is requested from the agent and sent to the tools of the administration site (code on demand [12]). Otherwise the user interface code is integral part of the agent and is transferred along with agent state and code when the agent is roaming the network at the target site. However, the solution of storing user interface code with the agent itself also has drawbacks. It is a form of tight coupling of the target site with the tools at the administration site, since the user interface code needs a special execution environment at the administration site. In addition, multiple different end user devices for configuration are not supported. Still it may be useful for some kind of remote configuration systems and we will present a similar approach in Section 3.1. A better solution is to store a platform independent user interface description with the agent. This allows device independent user interface generation at the administration site while maintaining the ability of configuring each agent in the system. A further enhancement is generating the user interface by analyzing the agent itself. We present such an approach in Section 3.2.
- f) *Minimization of user interface development*: Development of user interfaces for remote configuration is a tedious task and component (agent) developers should



focus on developing the application logic instead of providing remote configuration support. In the ideal case, no user interface needs to be developed at all. One approach for supporting user interface development tasks is user interface frameworks. Component developers just need to adapt general framework classes providing generic functionality for setting new values, for reverting to old values and for performing consistency checks. As stated above this approach not only involves coding effort but also tightly couples configuration tools to the platform of the user interface framework. For example, a Swing-based user interface requires a Java runtime environment at the administration site. A better approach is to generate the user interface from some kind of User Interface Specification Language (UISL). This is platform independent but still the user interface has to be specified. The most preferable approach is generating the user interface by analyzing the agent itself. This approach is based on the availability of meta-data about components, a distinct feature of each component-based system (see [5] for the importance of meta-data). By using meta-data the user interface can be generated automatically and involves no development effort at all. Meta-data is usually extracted from component implementation and interfaces and is stored as part of the component. However, meta-data provided by component platforms like Java and .NET often lacks important information that is necessary for generating “well-formed” user interfaces and for providing sufficient validation of component property values. In Section 3.2 we present an approach for automatically generating user interfaces from enhanced agent meta-data.

- g) *Security*: Security is a dominant requirement for Internet-based remote configuration. Since security is a very broad topic to discuss in general and very dependent on the specific configuration of the system and on the organizations involved, we are not able to discuss it sufficiently in this paper. As discussed above and illustrated in Figure 3 security issues may influence the fundamental structure of a configuration system and needs to be taken into account from the beginning. We provide an overview of basic support for authentication and authorization in our system in [7].

We have outlined and discussed basic requirements and solutions for remote configuration of dynamic and adaptable component-based systems. Most of the presented requirements are typical for remote configuration of component-based systems in general. Some are imposed through the use of mobile agent technology. In the next section, we present two solutions for remote configuration, which have been realized in our system.

3 CONFIGURATION OF INDIVIDUAL AGENTS

We have implemented two different approaches for remote configuration of agent properties at run-time. Both solutions are based on a central administration server at the administration site (see Figure 3), which eases administration of configuration tools as

stated in Section 2b. In addition, the administration server acts as a proxy for the configuration hosts, thus supporting network configurations with firewalls. In the first approach, agents are configured from Java-based user interfaces that are loaded on demand from the target site. The second approach is a generative approach where user interfaces are generated on the fly based on enhanced meta-information about the agents to be configured. In the following two subsections we present both approaches in more detail and refer to the requirements presented in the previous section.

Remote Configuration Based on Code on Demand

The code on demand approach supports dynamic configuration of agent properties (2a), central administration of configuration tools (2b) and component evolution (2e). Drawbacks are confinement to a particular kind of user interfaces (2c, 2d) and to Java based mobile-agent systems. Also, user interfaces need to be coded manually (2f), albeit based on a user interface framework which is part of our system.

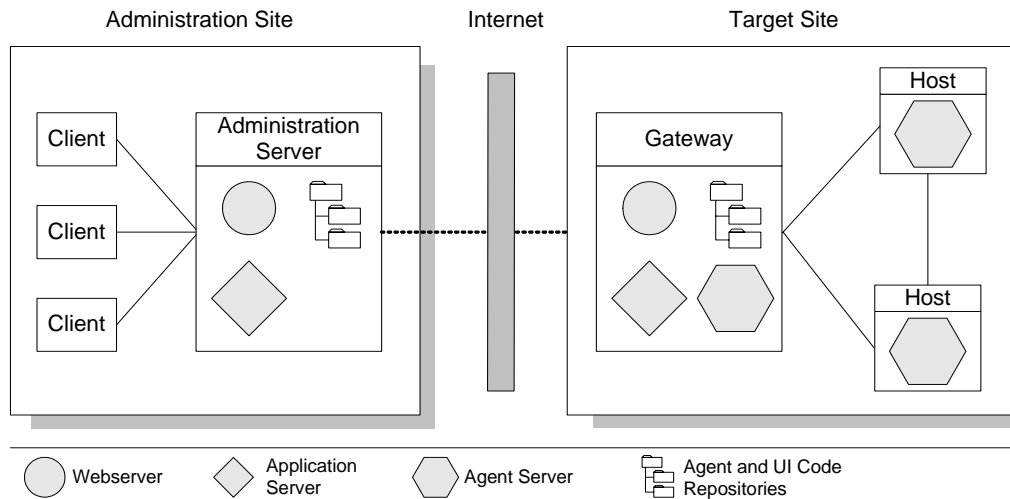
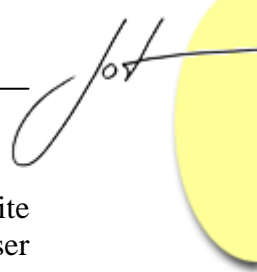


Fig 4: System Structure

The main system structure is equal to the structure shown in Figure 3 and is presented in more detail in Figure 4. Agents are installed and configured from configuration clients at the administration site. Upon installing an agent, its code, an initial configuration and its user interface code are transferred to the target site. The user interface code is not directly stored as part of the agent code. Instead, it is stored in a code repository at the target site. In principle, this would allow to implement the user interface for configuration based on other technology than the agent itself. In our system, however, both user interface and agent are implemented in Java. An agent does not store its user interface code directly but holds a unique ID that identifies the user interface code in the repository. If a configuration request is issued from one of the clients at the administration site, this ID is requested from the agent and used for identifying and transferring the user interface code to the configuration client.



Storing the user interface for configuring an agent in a repository at the target site ensures that for each agent that has been installed at the target site a configuration user interface can be found, no matter which administration site is used. Administration clients may even be placed within the target site since, from a logical perspective, the user interface that is needed for configuring an agent is always with the agent. From a technical perspective this solution enables code sharing. An installation tool might check whether an appropriate user interface for a newly installed agent is already available at the target site and assign its unique ID to the agent.

Storing the user interface code at the gateway server does not raise security problems, as only properly authenticated users are allowed to install or change mobile agents at a target system. Therefore the issue of malicious target sites tampering with the stored user interface code can be omitted.

We should note that we have also experimented with implementing the user interfaces themselves as agents and thus using agent mobility for transferring the user interface to the configuration clients at the administration site. This proved not feasible for mainly two reasons: (1) Configuring multiple target sites with different agent platforms is not possible and (2) agent platforms are not adaptable to the underlying network infrastructure.

First, we need to administrate multiple target sites based on different agent platforms from one administration site. As pointed out in Section 2d representing the user interface as agent would require an agent platform at the configuration client. Since target sites can use different agent platforms, a client would need multiple agent platforms for configuring agents from different target sites. Downloading the agent platform on demand is no feasible alternative. A common agent platform standard like FIPA [13] might help. However, standardization would need to include the underlying execution platform (e.g., the Java platform). We have defined an Agent Platform Abstraction Layer (APAL) specifying platform-independent abstractions for agent creation, disposal, communication and migration (see also [7]). This allows at least platform independent implementation and configuration of agents at the administration site and thus supports different agent platforms at different target sites (We should note that the implementation is still confined to Java-based agent platforms).

The second problem has also already been addressed in Section 2d and concerns network security based in firewalls. Corporate networks are usually secured by (multiple layers of) firewalls. Agent platforms need to be adaptable in terms of message routing and protocols to operate in such environments. However, typical agent systems are designed for operating in open environments based on peer-to-peer connections between agent servers. An additional problem for agent mobility is that accessing the administration site from the target site is prohibited by firewall settings.

We have implemented an adaptable communication infrastructure, which is used for sending agent properties from the configuration clients in Figure 4 to an agent at one of the agent servers at the target site. Communication is routed through a proxy at the administration server and through the host acting as entry point (gateway) at the target

site. Agent properties are not directly updated. Instead, the target agent first caches the configuration data and updates its properties only if it reaches a consistent state (2a).

Configuration data is encoded as Java objects. This might tightly couple user interface and agent code and imply that configuration user interfaces need to be Java based (see requirement 2d). However, this is not the case in our system. The target site can only be accessed through the gateway host (see Figure 4). Messages from external sources like configuration clients are routed through an application server at the gateway host which converts the protocol to the native protocol of the agent platform at the target site. Since multiple protocol converters for converting different protocols can be installed, configuration user interfaces that are not Java-based are possible. For example, we have implemented access from clients using HTTP for transport and XML for data encoding (see next section) and from CORBA via IIOP. Conceptually, accessing the system using other protocols like SOAP is possible.

A Generative Approach for Configuration UIs

As outlined in Section 2, generative approaches for configuration user interfaces are beneficial for supporting different types of configuration clients (2c) and for minimizing the effort involved in user interface development (2f). The approach presented in this section is particularly interesting for supporting these two requirements. However, other requirements like 2a, 2b, 2d and 2e are supported as well. The main idea is to use enhanced meta-data about an agent for automatically generating user interfaces and thus to eliminate the need for user interface development in the ideal case.

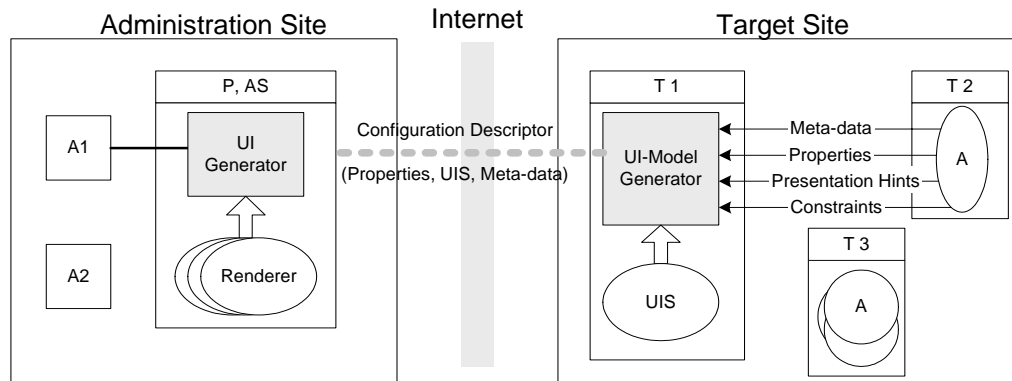
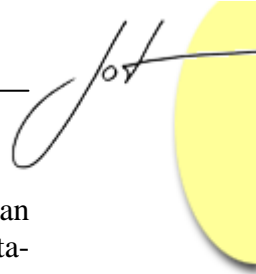


Fig 5: Generic User Interfaces

The basic system structure is similar to the one presented in the previous section and is shown in Figure 5. The figure also shows the main system components and the data needed for user interface generation.

In our system, agents are implemented in Java. Therefore meta-data about agents, like configurable properties, can be retrieved using introspection and reflection [16]. Meta-data and property values are transferred from an agent (A) to the UI-Model



Generator as shown in Figure 5. The UI-Model Generator automatically generates an XML-based User Interface Specification (UIS) and transfers it together with the meta-data and property values to a User Interface Generator that is located at the administration server at the administration site. We call the package consisting of UIS, meta-data about properties, and property values Configuration Descriptor (see Figure 5). Property meta-data and values are not represented as Java objects in the configuration descriptor, since this would lead to a tight coupling between the tools at the administration site and the components at the target site (see requirement 2d). Instead, we convert the retrieved meta-data (property names and types) as well as property values to a platform independent representation based on XML.

In principle, meta-data about property names and types is sufficient for automatically generating the user interface. However, the meta-data extracted from agents lacks important information like units of measurement and allowed ranges for property values. This information is needed for presenting and validating property values at the user interface. We enable an agent developer to provide such information either using an extended meta-data API or by providing XML-based constraint specifications for individual properties, which have to be deployed with the agent code. These constraints are sent to the user interface generator at the administration site as part of the configuration descriptor.

```
<dialog for="insight.agent.logfile.LogfileAgent" label="Logfile Agent Properties">
  <category label="General">
    ...
  </category>
  <category label="Task Schedule">
    ...
  </category>
  <category label="Protocol Files">
    <input name="rootDirectory" label="Root Directory:"> </input>
    <list name="files" label="Files: "> </list>
    <input name="expression" label="Expression:"> </input>
    <check name="sendFiles" label="Attach Files:"> </check>
  </category>
</dialog>
```

Fig 6: Presentation Hint Example

User interfaces based on constraint-enhanced meta-information are still rather crude in appearance. For example, field names that are derived from component properties are not verbose enough and all fields are just presented as one long list and lack semantic grouping (see left part of Figure 7). We allow agent developers to enhance the user interface layout and appearance by providing presentation hints as shown in Figure 6.

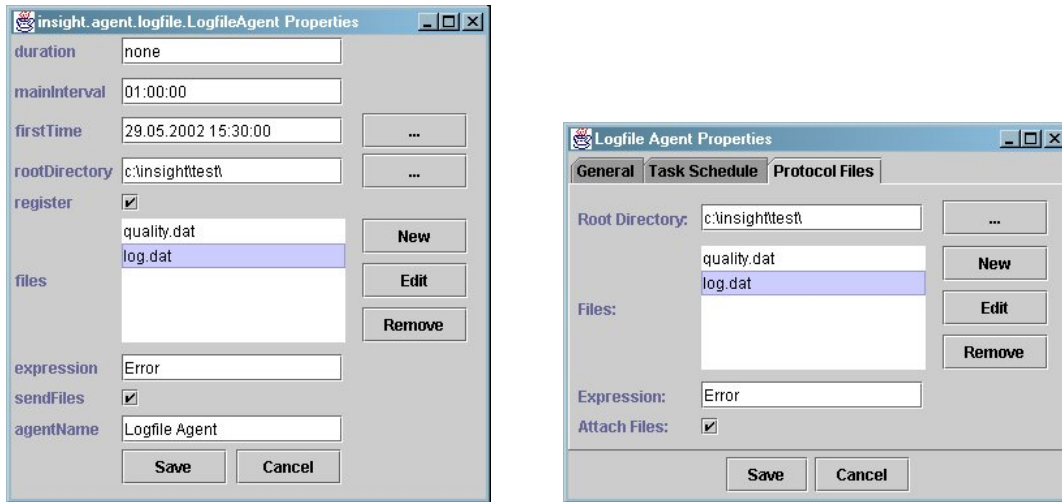


Fig 7: Use Interface without/with presentation hints

Using this information we are able to improve the appearance of the user interface as shown in Figure 7. The user interface shown in the right part of Figure 7 has been generated using the presentation hints presented in Figure 6. The main differences are verbose field labels and more clearly arranged user interface elements.

Summarizing, meta-data about properties, constraint specifications and presentation hints are extracted from an agent and sent to the user interface model generator at the target site. The model generator creates an XML-based user interface specification (UIS) which is transferred to the administration site along with property meta-data and property values in XML-format. The generated UIS is a hard- and software independent description of the layout of the agent properties and thus independent of any specific configuration client (see requirement 2c).

Instead of providing presentation hints, the complete UIS may be created manually and stored in a UI repository at the gateway host (see Figure 5). The main advantages of this approach are even more elaborate user interfaces, albeit the effort for user interface specification is increased, also.

The configuration descriptor (including the UIS) is transferred to the user interface generator at the administration site, which finally generates a client specific user interface. The UI generator uses pluggable renderers for generating different kinds of user interfaces. The user interfaces depicted in Figure 7 have been generated using a JFC/Swing renderer. Renderers for HTML, WML and other kinds of user interfaces may be provided as well.



4 CONFIGURATION OF SYSTEM PROPERTIES AND STRUCTURE

The previous section focused on approaches for configuring individual agents. In this section we outline our approach for configuring system properties and system structure. System properties are general data that is useful for all components within the system and data that is needed by system management tools. Examples are the hosts that are part of the system, the placement of agent run-time environments (agent servers), and information about the location of resources needed by agents. The system structure is defined by agent location and agent relationships. We support event relationships between agents which are used for modeling both control and data flow.

In principle, system properties could be provided by means of one central service at the target system. Alternatively they could be managed centrally by special agents at the target system. However, a central location for managing system properties, regardless of the implementation, suffers from two well known problems. Such a solution might become a performance bottleneck and it is not fault-tolerant. We provide different solutions for managing system properties—depending on the type of properties to be managed. Rather static properties that are unlikely to change are replicated throughout the whole system, which allows local access to these properties from all agents. Other system properties are stored in a system-specific trading and directory service. The trading service is fault tolerant using an enhanced multi-master replication mechanism. Storing information within the directory service may involve remote access for retrieving properties, but provides sophisticated synchronization mechanisms for changing and redistributing information within the system.

In both cases configuring system properties from a client perspective is achieved by configuring system property agents. Such an agent will either distribute the information within the system itself or store it in the system directory using the directory service.

The directory service is at the same time a trading service that maintains information about all agents (components) that are installed at the target site. Therefore, it can be used by configuration tools for determining which agents are located at a specific host or for retrieving a remote agent reference for accessing an agent. The trading service itself is based on agent technology and uses features like mobility for installing replicating instances at certain nodes within the system.

The environment is currently used as the basis for a system for remote diagnosis and monitoring of hard- and software resources in heterogeneous distributed environments. For setting up a working diagnosis and monitoring system, different types of agents need to be installed, configured and connected. The connections between agents make up the system structure. In our system these connections are event relationships, which can be changed either manually through configuration tools or automatically through program logic.

For example, the system provides worker agents performing measurement tasks, which can be connected to agents for filtering and condensing measurement values. These agents can in turn be connected to agents generating reports or storing the results in a data-base. In the described scenario, event connections are used for modeling data flow. Worker agents may also be connected to supervisors, which are used for monitoring and validating data. Supervisors may in turn be connected to other agents implementing the necessary actions that are to be performed in case of problems. In the latter case, event connections are used for specifying control flow.

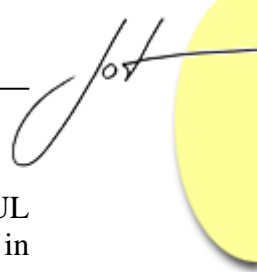
Event connections are maintained by an event service in our system, which also maintains the connection between agents, if agents change their location. The event service provides similar functionality as similar services for other distributed component models (e.g., CORBA notification service [14]). The API of the event service is similar to the API provided by the COM+ event service [15]. Like the trading service the event service is implemented based on agent technology. Event relationships can be changed dynamically using tools at the administration site, which are able to access the event service using the flexible communication infrastructure outlined in Section 3.1 and in [7].

5 RELATED WORK

In this section we identify approaches that are related to the work presented in this paper, specifically to our work on the configuration of individual agents, since this is the emphasis of this paper. We distinguish framework-oriented approaches (see Section 3.1) and generative approaches, which are further distinguished in specification-based and meta-data-based approaches (see Section 3.2).

The emphasis of framework-oriented approaches is on facilitating the development of complex user interfaces combined with support for code-on-demand (see Section 3.1). Examples of application frameworks for user interfaces are JFace [17] and JFC/Swing [18]. Although general user interface frameworks ease the construction of user interfaces, there is no explicit support for configuring distributed and potentially mobile components. User interfaces can be loaded on demand from remote servers using technologies like Applets and Java Web Start [9]. However, simple code on demand technologies are not enough. Configuring remote agents needs a more elaborate infrastructure for retrieving the user interface of a specific agent, wherever this agent is located. In addition, agent mobility, co-existing versions of agents and user interfaces as well as specific network topologies imposed by security requirements (see Figure 3 in Section 2) need to be considered. Further, code on demand technologies imply tight coupling of client and server since the code available on the server must be executable by the client.

Specification-based approaches aim at simplifying user interface development by generating user interfaces from user interface specification languages. This approach is interesting since it is independent from specific user interface toolkits and it also enables support for different hardware devices. The basic idea has been explored some time ago



(e.g., [19][20][21][22]). Newer approaches are based on XML like XForms [24], XUL [25], and PUC [26][27]. These approaches may be compared to the UIS mentioned in Section 3.2. However, the UIS is only one building block of a system for remote configuration of software components and it is not the main focus of this paper. The main idea of the approach presented in Section 3.2 is the combination of manual and automatic generation of the XML-based user interface specification and its integration in a remote configuration infrastructure.

As described in Section 3.2 automatic generation of user interface specifications is enabled by meta-data about the components to be configured. Meta-data-based approaches are particularly interesting for configuring component-based systems due to the self-descriptive nature of components. Automatic user interface generation based on meta-information is often used in user interface builders for configuring local user interface elements. Typical examples are development environments for Java and .NET. However, as discussed in Section 3.2 standard meta-data provided by components is not sufficient for creating well-structured user interfaces.

In terms of the remote configuration infrastructure our system can be compared to the Java Management Extension (JMX). JMX [28] is an approach for remote administration of hardware and software. The components that are installed for remote administration tasks are called MBeans (Managed Beans). MBeans may be compared to the agents in our system. An MBean represents a resource to be managed and may be accessed from remote clients using connectors and protocol adapters. Protocol adapters are installed at the target site and may use meta-data provided by MBeans for generating a configuration user interface. For example, the reference implementation of Sun Microsystems contains a general HTML-adapter which uses introspection and dynamically generates an HTML-based configuration user interface. Additional adapters for other kinds of user interface technologies can be provided.

Contrary to our approach, adapters in JMX are confined to the meta-data available in Java, which leads to less sophisticated automatically generated user interfaces. In addition, the complete user interface generator (a special adapter) is located at the target site. Supporting a new type of configuration client requires changes at the target site, since a new adapter has to be installed. In our approach, only a user interface specification is generated at the target site. The actual user interface is generated at the administration site (see Section 3.2). Only a new renderer has to be provided at the administration server in order to support a new kind of configuration client. In addition, the effort that is involved in creating new adapters for new kind of configuration clients in JMX is higher than in our system, since each adapter has to provide the entire functionality for meta-data analysis and user interface generation. In our approach, the functionality of the user interface specification generator at the target site need not be changed. It suffices to provide a new renderer for a new client type at the administration server. Finally, in our approach the coupling between configuration clients and the components at the target site is weak, as we use platform independent data formats and type systems, based on XML. In the JMX the coupling between client and server is defined by the implementation of a specific adaptor.

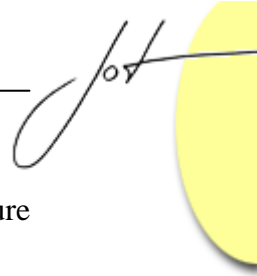
6 CONCLUSION

We have presented requirements and approaches for configuring remote and mobile components in a typical real world setting. Currently we use the system for configuring mobile agents performing monitoring and supervision tasks in process automation systems. Many of the presented requirements and solutions are important and useful for remote configuration of distributed components in general.

The use of mobile agent technology as the basis for the components at the target system imposes specific requirements on the configuration system like support for dynamically adaptable system structure and agent mobility. In terms of implementing the configuration system itself, we had to sacrifice seemingly obvious solutions for configuring remote agents (like migrating the agent and performing the configuration locally) in favor of other techniques like code on demand and automatic user interface generation.

REFERENCES

- [1] R.S. Hall, D. Heimbigner, A.L. Wolf: "A Cooperative Approach to Support Software Deployment Using the Software Dock", *Proceedings of the International Conference on Software Engineering 1999 (ISCE '99)*, Los Angeles, California, USA, 1999.
- [2] Kindberg T., Amanda F.: "System Software for Ubiquitous Computing", *IEEE Pervasive Computing*, Vol. 1, No. 1, January-March 2002.
- [3] C. Szyperski: *Component Software – Beyond Object-Oriented Programming*, Addison-Wesley, 1998.
- [4] G.T. Heineman, W.T. Councill (eds.): *Component-Based Software Engineering*, Addison-Wesley, 2001.
- [5] R. Weinreich, J.Sametingner: "Component Models and Component Services - Concepts and Principles", in *Component-Based Software Engineering* (<http://www.cbseng.com>), G.T. Heineman, W.Councill (Ed.), Addison-Wesley 2001.
- [6] M.L. Griss: "Software Agents as Next Generation Software Components", in *Component-Based Software Engineering*, G.T. Heineman, W.Councill (ed.), Addison-Wesley 2001.
- [7] R. Weinreich, R. Plösch: "An Agent-Based Component Platform for Dynamically Adaptable Distributed Environments", *Informatica Journal*, Special Issue on Component Based Software Development, Vol. 25 Nr. 4, November 2001.
- [8] Java Plug-In home page, <http://java.sun.com/products/plugin/>
- [9] Java Web Start Home Page, <http://java.sun.com/products/javawebstart/>



- [10] Wireless Application Protocol Forum: "Wireless Application Protocol Architecture Specification, WAP-210-WAPArch-20010712", July 2001.
- [11] G.P. Picco: "Understanding, Evaluating, Formalizing, and Exploiting Code Mobility", *Ph.D. thesis, Politecnico di Torino, Dipartimento di Automatica e Informatica*, 1998.
- [12] A. Fugetta, J.P. Picco, G. Vigna.: "Understanding Code Mobility", *IEEE Transactions on Software Engineering*, Vol. 24, No. 5, May 1998.
- [13] Foundation for Intelligent Physical Agents (FIPA): Specifications are available at <http://www.fipa.org/repository/index.html>.
- [14] OMG: "Corba Notification Service Specification", Version 1.0, Document formal/00-06-20, <http://www.omg.org/>
- [15] D. S. Platt: *Understanding COM+*, Microsoft Press, 1999.
- [16] Sun Microsystems: "Java Core Reflection API and Specification", Sun Microsystems, 1999.
- [17] Object Technology International, web server of the Eclipse project, <http://www.eclipse.org>
- [18] A. Fowler: "The Swing Architecture Overview", see web server of the Swing Connection, <http://java.sun.com/products/jfc/>
- [19] D.R. Olsen: "A Programming Language Basis for User Interface Management", in *Human Factors in Computing Systems, Proceedings SIGCHI 89*, Austin, April 1989, pp 171-176.
- [20] P. Sukaviriya, J.D. Foley, T. Griffith: "A Second Generation User Interface Design Environment: The Model and The Runtime Architecture", in *Human Factors in Computing Systems, Proceedings INTERCHI 93*, Amsterdam, April 1993, pp 375-382.
- [21] P. Szekely, P. Luo, R. Neches: "Beyond Interface Builders: Model-Based Interface Tools", in *Human Factors in Computing Systems, Proceedings INTERCHI 93*, Amsterdam, April 1993, pp 383-390.
- [22] C. Wiecha, W. Bennett, S. Boies, J. Gould, S. Greene: "ITS - A Tool for Rapidly Developing Interactive Applications", in *ACM Transactions on Information Systems*, Vol. 8, No. 3, July 1990, pp 205-236.
- [23] C. Phanouriou, M. Abrams: "Transforming Command-Line Driven Systems to Web Applications", in *Computer Networks and ISDN Systems 29*, 1997, pp 1497-1505.
- [24] W3C: XForms 1.0, W3C working draft, <http://www.w3.org/TR/xforms/>.
- [25] The Mozilla Organization: XPTToolkit Project, <http://www.mozilla.org/xpfe/>.
- [26] B. A. Myers. "Using Hand-Held Devices and PCs Together", *Communications of the ACM*. Volume 44, Issue 11. November, 2001. pp. 34 – 41.
- [27] J. Nichols, B. A. Myers, M. Higgins, J. Hughes, T. K. Harris, R. Rosenfeld, M. Pignol. "Generating Remote Control Interfaces for Complex Appliances." Submitted for Publication, available at the web server of the Pebbles project, <http://www-2.cs.cmu.edu/~pebbles>.

- [28] Sun Microsystems: “Java Management Extensions Instrumentation and Agent Specification”, V 1.0, July 2000.

About the authors



Rainer Weinreich is an assistant professor at the Johannes Kepler University of Linz. His research interests include component-based and distributed software architectures and mobile agents. He can be reached at rainer.weinreich@jku.at.



Reinhold Plösch is an assistant professor at the Johannes Kepler University of Linz. His research interests include reliable components and mobile agents. He can be reached at reinhold.ploesch@jku.at.