

Let's try the "I'm Feeling Lucky" Approach

[Position Paper]

Giovanna Di Marzo Serugendo
Department of Computer Science
University of Geneva
Switzerland
Giovanna.Dimarzo@cui.unige.ch

Manuel Oriol
Department of Computer Science
University of Geneva
Switzerland
Manuel.Oriol@cui.unige.ch

ABSTRACT

Even though modern software entities are able to engage dynamic interactions with other entities at run-time, communications still rely on standards, pre-established protocols, or common languages. In a world where communicating entities will be more and more present, due to the advent of wireless and embedded devices, we can anticipate that communication of the future will need a more loosely coupled interaction scheme. Entities will be programmed with no prior agreement about pre-defined interfaces and rigid standards. Instead, they will be furnished with a semantical description of their behaviour, and the insurance that the interaction will have a correct issue.

1. INTRODUCTION

Today's software elements tend to be independently developed and deployed entities able to communicate with each other. For instance, components are software units of composition, with contractually specified interfaces. They are developed and deployed independently and are subject to late composition by third parties. Web services are software applications, whose interfaces and binding are defined, described and discovered. They are registered in giant phone-books, and their service descriptions implemented in a common XML format. They support direct interactions with other software applications using XML based messages [3]. Autonomous agents, operating without the direct intervention of humans, are reactive, pro-active, and communicate directly or indirectly with other agents for cooperation or competition purposes using common communication languages [7].

Even though software entities of today have the ability of engaging interactions with unanticipated partners at run-time, their development and deployment still rely on agreed, pre-defined communication elements. Component-oriented programming requires standards to allow independently created components to interoperate, and specifications enabling

the composer to decide what can be composed under which conditions. Web services development requires the programmers to search Web services repositories for retrieving the specification (interface, protocol, conversation), of the services they want to communicate with. Specifications are expressed using a common XML format. Finally, agents communicate by adhering to some common communication language, or standard ontological language.

Agreed interfaces and standards offer a good basis for interoperability, since they ensure syntactical compatibility, and communication protocol adherence. However, requests for Web services cannot be expressed on the basis of the caller's needs. They are driven by what is offered by potential furnishers. Similarly, agents requests must conform to standard primitives, thus tuning of particular requirements may be impossible.

In addition, once these entities enter in some interaction, there is no means to ensure the run-time correctness of the interaction. A Web service, invoking another Web service, assumes that the partner's behaviour is compatible with the published specification, and that the issue of their common interaction will not leave one of them in a run-time error state. This problem is even more acute for a Web service satisfying a request: it has no idea of the partner's behaviour, it is committed to satisfy the request, which is syntactically correct, but it has no guarantee on the issue of the interaction. By adhering to ontologies, agents also perform some assumptions on the partner's behaviour, and no guarantees are given regarding the interaction issues.

With the advent of wireless personal digital assistants (PDAs), and peer-to-peer (P2P) networks, the number of unanticipated entities wishing to communicate will increase drastically, and will exacerbate the situation. They will form a highly dynamic environment, with running entities leaving and arriving continuously, making it impossible to rely on pre-defined interfaces. Their unanticipated requests will cause standards and ontologies to become tools describing only approximatively their needs.

2. OUR VISION

We foresee that systems of the future will need flexible and loosely coupled communication models that go beyond agreed interfaces and standards. These systems will be composed of a large number of entities, willing to communicate, which have been programmed with no idea of *who* will be the communicating partners, of *how* to communicate with them in a pre-defined way, but with the *guarantee* that the

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.

Copyright 2001 ACM X-XXXXX-XX-X/XX/XX ...\$5.00.

interaction will have a correct issue.

The principles regulating this kind of communication are the following:

- an entity doesn't care about which entity satisfies its request (no naming);
- an entity doesn't care about the exact syntax needed to realize the communication;
- an entity cares only about *characterizing* what it wants, or what it is able to offer, and to which extent the partner is able to satisfy its needs;
- both partners care about the guarantee of the *correctness* of the issue.

Let us consider the following scenario: a European user desires to know the usual weather (temperature, humidity) in Adelaide (Australia) in June. There is some conference over there, and she wants to know how to prepare her luggage. She will first start its wireless application running on a PDA, where her agent wait for her requests. She asks "How is the weather in Adelaide in June?". The agent is then in charge of finding a partner, most likely located in Adelaide, able to answer this question. Ideally, the agent should simply characterize its request concerning the weather, indicating that it wants details regarding temperature, and humidity. In addition, it may place some requirements on the kind of multimedia information that is returned back, e.g. voice results are not desired since the PDA has no speaker.

In this hypothetical scheme, the agent has at its disposal an ocean of unknown service furnishers, and among them, the one who will be able to give an answer. It does not know who to contact, and consequently, it does not know how. Therefore, it has to express the request about the needed information, focusing on the semantics of what is needed, but not on the exact syntax. In addition, in order to ensure a correct behaviour on the PDA, it will precise that it is unable to accept voice results.

3. "I'M FEELING LUCKY"

Some of our preliminary work [5], realizing these ideas in the framework of object-oriented components, have produced a conceptual model based on a service-oriented architecture, i.e., interactions are all based on services requests and satisfactions.

Similarly to the "I'm Feeling Lucky" option of the Google search engine, entities willing to communicate, simply build communication requests, and submit them. Among all other entities present at that moment, one of them, considered to be in the best position for fulfilling the request, satisfies the request (if it agrees).

Figure 1 depicts the approach:

- an entity providing some service registers it to the interaction cloud (1);
- an entity willing to have some request satisfied, i.e., willing to interact with some unknown entity, launches its request in the "air". Actually it will submit it to the interaction cloud. It has no a-priori knowledge of a possible service provider, and "feels lucky" that the request will be satisfied (2);

- the request is captured by the interaction cloud, i.e., a coordinating element responsible for retrieving a service provider able to satisfy the request. A request may require an answer or not, it may also simply trigger an activity (a computation) to the service provider. In addition, it may be fulfilled or not;
- if a service provider is found, it is requested immediately by the coordinating element to satisfy the request (3), i.e., the requester is lucky;
- otherwise, nothing happens, the requester has to try again (4), to find another means to realize its task, or to give up (it is not lucky).

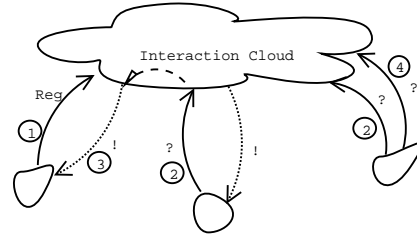


Figure 1: The "I'm Feeling Lucky" Approach

When trying a search on Google, with the query "How is the weather in Adelaide in June?", using the "I'm feeling lucky" option, the engine immediately returns the forecast news of some local television. In this case, the matching of a search request, expressed in a natural language, is performed on a syntactical basis. The returned result is simply the Web page that best matches syntactically the given question.

In the case of communicating entities, requests may need more input than natural language, and answers may consist in computations by the service provider, or in output results more complex than a Web page.

Therefore, unlikely to Google, where matching a search request is performed on a syntactical basis, the request specification must be precise enough to obtain reasonably controlled results. The request must contain sufficient information for characterizing itself, and for ensuring both partners that the interaction will have a correct issue.

Service publications and service invocations are made using "descriptions" of what is available and what is requested respectively: functionality, behaviour, and quality of service. The fundamental point is that there is *no prior agreement* between services, i.e., descriptions are built independently of each other. Service publications and service requests are sent to a coordinating element (not necessarily centralized). It is then responsible for performing a relaxed matching of publications and requests descriptions; and for actually invoking the service whose description best matches the request. In this scheme, programmers do not know in advance the description of available services. Executing entities wishing to invoke a service do not search for available service description. They simply build the description of the service that will best fulfill their needs, and submit it to the coordinating element. The latter is then in charge of finding the best partner and invoking it immediately. This model differs from a Linda-like coordination scheme, since it focuses on invocations, instead of data repositories.

An instantiation of this architecture for P2P networks of PDAs leads to a model, where the interaction cloud is composed of a set of interconnected servers. A PDA programming layer is being implemented on top of Lana [2], an agent-based programming language we have developed. Lana is designed for programming Internet systems with support for security and asynchronous disconnected operation. Programs are isolated from each other, so no program may gain access to another program private data. A user may lose contact with programs - or agents - deployed over the network without disrupting the whole computation. Once the network connection is restored, communication may continue as if the network has always been there. These properties are useful for programming wireless systems, since disconnection is frequent, and from a security viewpoint, there is no way of controlling the devices that may enter into the network. The Lana design has been implemented over the Java virtual machine.

4. FUTURE WORK

Right now, our works have addressed the characteristics of the communication by providing an asynchronous, disconnected communication scheme, based on requests descriptions. However, requests are still based on matching ontologies and signatures descriptions.

The next steps consist in investigating ways of removing the barrier of signatures descriptions and pre-defined ontologies. We intend to furnish a means for defining the semantics of the furnished/requested behaviour, in a very abstract way, using some formal specification language (e.g., a description logic). In order to ensure correctness of interactions, we will integrate properties and proofs.

Since an entity must be self-sufficient, it has to incorporate more information than its operational behaviour, and to publish more data than its signature. We want to program a component with a functional part (traditional behaviour); and a non-functional part describing all the necessary abstract/semantical information favoring interactions with unanticipated entities, and ensuring correct interaction issues. The idea is then to program, for each component, the following elements:

- a *functional* part implementing the behaviour of the component - the traditional program code;
- an *abstract description* of the component behaviour - the semantical behavioral description;
- the *proof* of its correctness - useful for run-time proof of correctness.

The last two points are intended to be publicly available at run-time by other components. This extra information (description and correctness) is the glue, embedded in the component, and used to combine the components at run-time. It provides to other components, the necessary meaningful representation of the current component, which will allow run-time discovery of components and correct composition.

5. RELATED WORKS

Steps to move away from pre-defined standards are already under way. Indeed, in the case of multi-agent systems, the semantic space framework [6] enables agents to negotiate at run-time the semantics of the communication language.

On the lead for integrating semantical information to service descriptions, we can mention an approach, based on description logics (DL). It allows to enrich the description of a service using an Interface Definition Language (IDL), with the formal intended semantics of the service. Thus, it provides the consumer the guarantee that the service will work in a particular context as anticipated [1].

In the framework of software components, the approach of [4] proposes to describe the behavior of software components using formal specifications (pre- and post-conditions). A relaxed matching of the specifications enables to compare software components, and to determine whether one component can be substituted for another.

6. CONCLUSION

Our feeling is that, in the future, components will require loosely coupled interaction schemes. New communication mechanisms will allow interaction with unanticipated parties. Independent development and deployment will rely on no prior agreement on interfaces and standards.

Anticipating such a tendency, this article presents the ideal view of the interoperability of the future, and brings some preliminary ideas on how to realize it. Ideally, a software entity should just start its execution and begin to communicate with entities already present, without caring on pre-defined rigid standards.

The presented service-based communication architecture provides an asynchronous communication mechanism, relying on services descriptions and matching. Service invocation and publication convey: (1) the meaning, instead of the syntax only, of what is expected and provided respectively; (2) the insurance of correct behaviour, instead of an adherence to a protocol only. Such an architecture allows independence from the communication link, and easy development and deployment.

7. REFERENCES

- [1] A. Borgida and P. Devanbu. Adding more 'DL' to IDL: Towards More Knowledgeable Component Inter-Operability. In *21st International Conference on Software Engineering*, pages 378–387. IEEE Computer Society, 1999.
- [2] C. Bryce, C. Razafimahefa, and M. Pawlak. Lana: An Approach to Programming Autonomous Systems. In *16th European Conference on Object-Oriented Programming, ECOOP'02*, 2002.
- [3] E. Christensen, G. Curbera, F. Meredith, and S. Weerawarana. Web Services Description Language (WSDL) 1.1. W3C Note, 2001.
- [4] A. Moormann Zaremski and J. Wing. Specification matching of software components. *ACM Transactions on Software Engineering and Methodology*, 6(4):333–369, October 1997.
- [5] M. Oriol. Evolution of Code through Asynchronous Services. In *Workshop on Unanticipated Software Evolution. ECOOP'02*, 2002. submitted.
- [6] C. Reed, T. Norman, and N. Jennings. Negotiating the Semantics of Agent Communication Languages. *Computational Intelligence*, 2002. to appear.
- [7] G. Weiss, editor. *Multiagent Systems - A Modern Approach to Distributed Artificial Intelligence*. The MIT Press, Cambridge, Massachusetts, 1999.