Web services and Software Agents for Tailorable Groupware Design

Nader Cheaib, Samir Otmane, Malik Mallem

Web Services and Software Agents for Tailorable Groupware Design

Nader Cheaib, Samir Otmane, and Malik Mallem

Abstract We present a new groupware architecture model called UD^3 that explicitly introduces the notion of tailorability in designing collaborative applications. This model is based on the integration of web services and software agents technologies, thus using protocols of each while reinforcing their individual strengths in the context of tailorable groupware design. In our work, web services are dynamically invoked by software agents in order to bring new behaviors, and hence, enhancing the collaboration process by dynamically adapting the services offered in the system to the users' preferences and not the other way around. Web services and agents were originally developed with different standards, thus their integration becomes important in the context of groupware tailorability, giving a totally innovative approach in the area of CSCW (Computer Supported Cooperative Work). We apply our model on the DIGITAL OCEAN project for the creation and distribution of multimedia files on the Internet.

8.1 Introduction

As the use of the Internet and the services offered with it are emerging more and more, people are in an increasing need of flexible and agile applications. The emergence of collaborative work over the Internet was a solution to the high complexity of systems and the technical difficulties that could arise from their use, as users, geographically distributed want more and more to work together on a single task, but using rigid and often incompatible applications that may lead to interoperability problems. The aim of CSCW (Computer Supported Cooperative Work) is to find ways for groupware to enhance collaboration between individuals. For [36], groupware invention is a challenge, as the nature of collaborative work continually changes as a consequence of changing work needs, but also as a consequence of how the systems themselves tend to change work relationships and processes. As a

N. Cheaib (✉)
IBISC CNRS FRE 3190, University of Evry, 91020 Evry Cedex, France
e-mail: nader.cheaib@ibisc.fr

consequence, the author argues that systems must themselves adapt to reflect the unpredictable differences between the requirements of support for collaborative work on the Internet during analysis and the actual requirements.

Hence, research about tailorability for groupware originated from the gap between the design and use of collaborative systems. Making the system and the services offered within, tailorable by users is an essential and ongoing research field that needs much attention to yet be concrete. For this reason, tailorability has shown to be an essential property that should be taken in consideration, as it offers to users the possibility to adapt the application based on their needs and not the other way around. In this work, we present a new groupware architectural model called UD^3 (Universal Directory for Description and Discovery), which is based on the integration of web services technologies with software agents. The aim is to design a tailorable groupware architecture using the integration of both technologies, thus using properties of each while reinforcing their individual strengths. In fact, agent-oriented technology is claimed to become the next breakthrough in the development and implementation of large-scale complex systems, while web services are fast emerging technologies for connecting remotely executing programs via well established Internet protocols. Web services and agents were originally developed with different standards, thus their integration becomes important in the context of groupware tailorability, giving a totally innovative approach for designing collaborative applications on the Internet.

In fact, with the emergence and advancement of Internet technologies and the Web 2.0 [11], universal interoperability between collaborative applications is becoming a reality, while geographically distributed people are highlighting the flexibility of cooperation by exchanging universally accessible services on the web. However, these types of systems do not take in consideration the evolving and excessive need of users' to dynamically integrate new components in order to enhance collaboration with others. On one hand, web services have become one of the most important architectures for the cooperation of heterogeneous systems and have ushered in a new era of software design that focuses on implicit and explicit collaboration between organizations [11]. While computer networks have been able to pass data between different hosts, it was the emergence of web services that allowed these remote hosts to offer services in a more flexible and dynamic way. However, with this flexibility comes systems complexity. On the other hand, the autonomy and intelligence of agents have considerably increased software automation of some operational areas. An important benefit in the use of software agents in designing software and groupware applications is their ability to help, through collaboration, human beings and softwares' execution, while their concept is even older than web services and has been used successfully for the implementation of distributed applications. We present an innovative approach for a tailorable groupware architecture integrating web services with software agents. The idea is to exploit agents' proactive interaction capabilities in order to improve the behavior of web services in a service-oriented architecture, hence creating a cohesive entity that attempts to surpass the weaknesses of each technology, while reinforcing their individual advantages in the context of tailorable groupware design.

We will proceed as follows: In Sect. 8.2, we give a motivating scenario explaining the problem, and we explain the concept of tailorability with the need of a new architecture supporting it. In Sect. 8.3, we talk about few approaches in the literature that attempt to introduce tailorability in their design of groupware applications. In Sect. 8.4, we give a background on web services and software agents', along with the JADE platform [36] for deploying software agents. Section 8.5 presents our own approach for a tailorable architecture; we call it the UD^3 model. Section 8.6 describes system's implementation on an ongoing project for the exchange of multimedia information over the Internet, Oce@nyd. The last section presents a conclusion and future work in the field.

8.2 Motivating Scenario

Let us consider a scenario where users geographically distributed are using a collaborative application over the Internet, thus using Internet protocols and standards in order to collaborate and exchange messages between each other. In order to achieve tailorability, we should think of a way in order to make users able to search, invoke and use new components that could be directly integrated in their system to satisfy their needs according to the task being done. If we imagine the components of the system built using web services standards, then we should enable users to invoke new web services and dynamically integrate them into their applications. As a concrete example, if users in collaboration need a video stream mechanism, we imagine a web service deployed somewhere in public registries on the Internet and containing this mechanism as a part of the services it offers, and thus the system should be designed in a way to seamlessly search, invoke and integrate this web service into the application. By tailoring here we mean dynamically adding/modifying web services during runtime of the application without interrupting its execution, and thus of users' collaboration with others. However, for [25], current techniques for publishing and finding web services rely on static descriptions of service interfaces, forcing consumers to find and bind services at design time. This motivated us to make the process dynamic in the essential purpose of enhancing collaborative applications, in particular making them tailorable by users. In our research, we found that software agents are a promising technology that could solve such problems. Unfortunately web services and agents were originally developed separately with different standards and features, therefore their integration becomes important in this context. With this paradigm, groupware components, each representing a web service and an agent in collaboration, will interact to provide unified services according to users' preferences, and thus achieve groupware tailorability.

8.2.1 Tailorability and the Need of a New Architecture

Some definitions exist in the literature for the concept of tailorability, but it is still ambiguous in putting it forward in CSCW systems, where the technologies for im-

plementing such concept are still not explicitly identified. We retained few definitions that seemed most interesting to our work, as in [35] that defines a tailorable application as a system that can be adapted properly according to changes and the diversity of users' needs, or [32] that defines tailorability as the capacity of an information system to allow a person to adjust the application based on personal preferences or different tasks. For [10] tailoring is the continued development of an application by making persistent modifications to it. It is in fact initiated in response to an application being inefficient to use. However it remains to determine the mechanisms of evolution of tailorability. Morch [22] defines the concept in terms of customization, integration and extension. Tailorability by customization is limited by a set of predetermined number of components, tailorability by integration is to insert a new component in the architecture of the application, and tailorability by extension or radical tailoring is offering means to change or extend the components' implementation in order to derive the same flexibility as an "initial" application design. These mechanisms offer more flexibility but require more and more from the user computer skills, which partly explains why most of the current CSCW systems generally steer their tailorability to developers and expert users rather than end users that, paradoxically and from social sciences and humanities research, are those who need it the most. In our work we focus on the third type of tailorability, hence extending program code by new components depending on users' preferences. We assume that a component is a web service and an agent collaborating together in order to offer unified and dynamic services to users.

In groupware, a mismatch between the task done by users and the corresponding technology they are using could affect the co-operating people [33], thus tailoring by end-users themselves is generally regarded as a suitable means to solve this problem. Due to a lack of a theoretical framework for tailorability and the corresponding evaluation methods, results of different studies for groupware tailorability are hard to compare. Our research is mainly concentrating upon:

- Development of a collaborative architecture supporting tailorability.
- Integration of Internet technologies that has not been exploited before in the context of groupware tailorability.

In the next section, we talk about few approaches in the literature that aim to introduce tailorability in the design of groupware.

8.3 Tailorability Approaches

Various approaches aiming to integrate tailorability in CSCW systems have received much attention in the literature [3, 32, 35]. However, most of these approaches apply only to certain specific domains, as support for synchronous groupware, workflow-based or collaborative writing, and it is not certain whether these approaches could be applied to generic domains as well. In our research, we found that introducing tailorability in the design of groupware is still very limited and theoretical, as there

exist various approaches without a sufficient support for comparison and classification. For this reason, we thought that providing a global view on some of these approaches is already a contribution for building a concise study of the problem, and finding suitable technologies to implement a solution. In the rest of this work, we will begin by building a global view on some approaches for tailorability in CSCW systems. We will mention respectively the activity theory [2], component-based [30] and building blocks [32] approaches, and finally we will see how we can use the Service-Oriented Architecture (SOA) for building collaborative applications [7].

8.3.1 Activity Theory

The author in [3] justifies that tailorability possesses a theoretical foundation enabling to apprehend it using fundamental properties of human activity. They propose a set of properties for constructing a conceptual model for a generic environment of CSCW systems, based on a fundamental theory, reflectivity. This environment is called DARE (Distributed Activities in a Reflexive Environment) [2]. In the realization of DARE, they propose a framework based on the concepts and mechanisms of the activity theory, which permits to distinguish two essential properties of the human activity:

- Reflexivity, that enables to access and modify the structure of the application during its execution.
- Crystallization or the reutilization of user's experiences. These experiences could be, for example, a specification of roles in a particular activity.

Based on the activity theory, all mediator elements influence the course of activity and thus it is impossible to predict its impact on a certain activity [3]. This is why, for the authors, the tool should be considered a fully mediator element, meaning that if it could influence the collaborative activity, then it should be modified by it. The authors were inspired by the Meta-Object Protocol (MOP) [2] for realizing DARE, as the reflexivity takes place with the introduction of a meta model whose main entity is the 'task,' that is a specification of the activity that describes the objectives, resources and roles that should take place in collaboration between actors.

8.3.2 Component-Based Architecture

A lot of research has been made for the design of component-based architecture for groupware [30, 32, 35]. The concept of a component-based architecture is independent of any application domain, and thus it is highly probable to adopt this kind of architecture to integrate tailorability in the design of groupware [30]. In a component-based approach, a groupware is designed as a collection of components in which they could be added, modified, or deleted. This type of applications will be

able to support the evolution that tailorability tries to introduce. The authors in [30] argue that an ideal collaborative system should be designed as a composable system where the integration of new components is build on top of a neutral basis. We will see here two component-based approaches, each using different ways and mechanisms to reach tailorability: A reflexive computational system [35] and building blocks architecture [32].

8.3.2.1 Reflective Computational System

The authors in [35] define a tailorable system as one that can be adapted for eventual modifications in its structure according to diversity of user's needs. The authors use the term adaptability to identify tailorability in its technical aspects. Here, the authors reused the notion of reflexivity in the activity theory seen in the first approach [3], by insisting that an adaptable application should include a representation of aspects of itself, and this self representation should be changeable by internal or external influences, and connected to certain aspects of the application. If the representation changes, the application changes as well, and only aspects included in the self representation of the application are susceptible to be affected by tailorability activities. As a simple example, consider an application with an initialization file that specifies the application's background color [35]. In this case, this initialization file is the self representation of the application, and the color is the adaptable aspect. This type of applications is seen as a "Reflective computational system." Note that a reflexive system is one that contains both representations of aspects of the real world, and representations of its own activities. In consequence, this type of application is capable of examining its own state and structure, and able to modify it according to user's and the context's needs, which implies that every modification of the (meta) representation is automatically shifted towards the behavior of the system.

8.3.2.2 Building Block Architecture

The authors in [32] propose an approach based on building blocks for constructing tailorable CSCW systems. They argue that the evolution in the utilization of groupware is nowadays one of the main reasons for designing tailorable systems. In fact, the authors consider a tailorable system as one that permits for its users to perform modifications on the technical structure of the application, after its implementation according to their needs, personal preferences or different tasks. For the simple reason that all the modifications could not be predicted in the design phase by the application designers, it would be possible, according to the authors, to equip the users with means to accommodate these changes.

The authors introduce the concept of tailoring to the extreme [32]. This concept implies the extension of the set of functions in the system with new modules that could be integrated dynamically. An example of this concept is to permit the

user to download modules from the Internet and plug them directly into the system (plug-ins, widgets, etc.). However, this approach requires that functional modules (building blocks) should be analyzed before integrating them in order to determine the functions that they could offer and the way in which they will communicate and interconnect to other modules for minimizing interference in the system. The authors here insist that interoperability standards are therefore essential between the building blocks that will be integrated into the system, probably resulting from different vendors, in order to standardize and facilitate the process of integration with other building blocks already existing, and therefore, insure the stability of the system as a whole. The authors implemented their concepts in the CooPS (Cooperative People and Systems) [32] project that describes the types of building blocks that form groupware applications and the relations between them.

8.3.3 Service-Oriented Architecture (SOA)

The demand for collaborative and flexible services is becoming more urgent as the competition in the marketplace is getting fiercer between service providers. For this reason, the authors in [7] propose the utilization of a Service-Oriented Architecture (SOA) for the construction of collaborative services. For the authors, SOA is becoming a new paradigm that aim at implementing loosely-connected applications which are extensible, flexible and integrate well with existing systems. Collaborative platforms have the potential of offering services on different layers of abstraction as their role is to offer a support tool for collaboration of activities [14].

SOA [7, 14] is a paradigm in full expansion that could be adapted to offer extensible services integrated in a platform for different users to collaborate between each other. Web services could facilitate the collaboration between groups or organizations, and can be defined by, for example, resource sharing, communication and interaction between collaborators (synchronous, asynchronous, communication channels, etc.), virtual rooms, organization management (calendar, mail, etc.). The support for web services offers interoperability between different collaborative or single-user systems [14], as they can be viewed as modular applications. The architecture considers a model of integrated services, where the interfaces of web services are described with a standardized language definition WSDL (Web Service Definition Language), and interact with each other using SOAP (Simple Object Access Protocol), while having their definitions saved in some norms of a web service catalogue using UDDI (Universal Description, Discovery and Integration).

In what follows, we give a brief a background on the CSCW domain, with a description of web services and software agents. We justify our choice for using the JADE platform in implementing software agents as a part of system's core. Our idea is to show that the CSCW domain can be leveraged by using web technologies in order to enhance collaboration between users.

8.4 Background

8.4.1 Ellis's 3C Model

We refer to the 3C model [8] for further understanding of the term collaboration and the functionalities behind it. In fact, according to [8], a groupware system covers three domain specific functions, production/cooperation, communication and coordination as we can see below: The production space designates the objects resulting from the activity of the group (ex: word document, paint, etc.). For Ellis [8], this production space is concerned with the result of common tasks to be achieved and it is the space where the productivity will take place. The coordination space defines the actors and their social structure, as well as different tasks to be accomplished in order to produce objects in the production space. Ellis eventually completed the model with the communication space that offers to actors in the coordination space means to exchange information in which the semantics concern exclusively the actor, and where the system only acts as a messenger.

We will use this decomposition of groupware's functionalities in order to introduce a collaborative architecture supporting the functional decomposition of services that can be present in a groupware system.

8.4.2 Web Services and the World Wide Web

W3C defines a web service as follows: "It is a software system that acts as an interoperable support in the machine–machine interaction. The system has an interface described in a form understood by the machine (specifically WSDL). Other systems interact with the web service depending on its description using SOAP messages that are typically transported through HTTP with an XML serialization in conjunction with other web standards." In fact, service-oriented architecture (SOA) emerged due to its simplicity, clarity and normalized foundations. The concept of web services currently revolves around three acronyms [24], as we can see in Fig. 8.1:

- SOAP (Simple Object Access Protocol) is a protocol for inter-application exchanging that is independent of any platform and based on XML. A SOAP service call is an ASCII flow embedded in XML tags and transported to the HTTP protocol.
- WSDL (Web Services Description Language) gives the XML description of web services by specifying the methods that can be invoked, their signatures and access point (URL, port, etc.). It is therefore equivalent in a way to the IDL language for CORBA distributed programming.
- UDDI (Universal Description, Discovery and Integration) is a standard of a distributed directory of web services, allowing both publishing and exploration. UDDI acts as a web service itself, whose methods are called using the SOAP protocol.

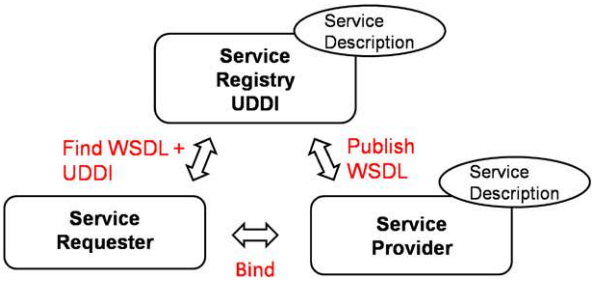

Fig. 8.1 Service-oriented architecture

Our choice of using web services in our system is driven by the fact they are: Language and platform independent (separation of specification from implementation), deployed over the Internet (no centralized control, use of established protocols), loosely coupled (using synchronous and asynchronous interactions) and interoperable (using standards already deployed and functional to support systems interoperability).

8.4.3 Software Agents

There exist several definitions of software agents in the literature. Khezami [15] has identified the agent as a computing object (in the sense of object-oriented languages) whose behavior can be described by a script with its own means of calculation, and can move from a place to another in order to communicate with other agents. The authors explain that some researchers have given the definition of agent through a good description of its functioning, where an agent must necessarily have the necessary motivation to achieve a certain goal for its existence to be worthwhile in its environment. An agent can communicate with other agents in the environment and must have means which enable it to achieve its goals. According to [19], an agent is a piece of software that acts on an autonomous basis to initiate charges on behalf of users. The authors here say that the design of many software agents is based on the approach that users need to only indicate a high level goal instead of issuing explicit instructions, leaving the decisions to the agent. The agent shows a number of features that makes it different from other traditional components, including self-direction, collaboration, continuity, character, communication, adaptation, mobility and temporal continuity.

8.4.4 JADE Platform

Java Agent DEvelopment framework (JADE) [36] is a middleware written in Java and conforms to the specifications of FIPA [9]. This environment simplifies the

development of software agents by providing basic services as well as a set of tools for the deployment. The platform contains a runtime environment where the JADE agents may evolve while being active on a given host, a library of classes used to develop agents and a suite of graphical tools that allow the administration and supervision of agents' activities at runtime. In fact, the main container contains two special agents:

- AMS (Agent Management System) which provides a service Namespace (i.e. it ensures that every agent in the platform has a unique name) and represents the authority in the platform (it is possible to create or kill agents in remote containers by calling the AMS).
- DF (Directory Facilitator), on the other hand, is analogous to the UDDI used by web services, and offers the Yellow Pages service through which an agent can find other agents that are providing the services it needs in order to achieve its goal. JADE defines a generic agent model that can perform any type of architecture while fully integrating the FIPA [9] communication model: interaction protocols, wrapping, ACL (Agent Communication Language), languages content and transport protocols. In what follows, we proceed with some related work exposing researchers' motivation in this domain affecting many applications areas.

8.4.5 Related Work—Web Services and Agents' Integration

According to [10], web services have become one of the most important architectures for the cooperation of heterogeneous systems. They present a platform that provides a runtime environment of a lightweight agent that is located within a web container, which adds agents' functionalities to existing web servers. The components of the platform are deployed as web services, where SOAP (Simple Object Access Protocol) over HTTP acting as a communication channel through standard XML messages. In this way, the support for mobile agents can be added to the existing web infrastructure, without the need to replace components or installing client software. In [20], the authors propose a solution for the selection and composition of web services with software agents. The use of the concept allows an agent to support the pro activity and autonomy of the composition process in which clients and suppliers can take an active role through autonomous operation and negotiation. The proposed architecture provides flexibility and scalability in the development of different solutions, while offering a set of integrated tools.

For the authors in [18], it is widely admitted that web service composition is essential rather than accessing only a unique service. Searching for web services, integrating them into a composite service, triggering and monitoring their implementation are among the operations that users will handle, whereas most of these operations are complex and repetitive, with a large portion adapted to the computer tool and automation. Therefore, for the authors, software agents are appropriate candidates to assist users in their operations, and therefore the integration of software

agents and web services in the same environment raises the importance of a specific approach. The authors employ a three-tiered approach: intrinsic, functional and behavioral, where each level has multiple properties that vary according to the component whose tier is applied, whether it is the agent or the web service. For the software agents' properties, the intrinsic level properties consist of an identifier, role, and type. For web services, the intrinsic level consists of identifier properties, description, type, input and output arguments, and cost and time of implementation. In [34], the Do-I-Care application has been designed to help users discover interesting changes on the web, using both technical means and social rights. Do-I-Care agents automate periodic visits to selected pages for detecting interesting changes on behalf of users, where they must keep their agents informed of the relevant pages and the quality of reported changes. Once an agent detects an interesting change, the user is notified by e-mail, and the change is attached to the web page associated with the agent. This web page is also used for the relevance of comments and the activity of cooperation.

Clearly, there is still no work in the literature that emphasizes the use of the two technologies in the domain of groupware tailorability. This gives originality to our approach and motivates us to present a model designed to tackle specifically the problem of tailorability along with interoperability problems between heterogeneous applications deployed on the Internet.

8.4.6 Purpose of Integration

For [25], current techniques for publishing and finding services (such as WSDL and UDDI) rely on static descriptions of service interfaces, forcing consumers to find and bind services at design time. However, web services are becoming one of the most important architectures used in heterogeneous cooperative information systems, as it was the appearance of web services that permitted Internet sites to offer services in a more flexible manner [10]. However, the concept of software agents is even older than web services, and it has been employed with success for executing distributed applications. Agents are defined briefly as is a piece of software that acts autonomously to undertake tasks on behalf of users. For [18], it is based on the fact that users only need to specify a high-level goal instead of issuing explicit instructions, leaving the how and when decisions to the agent. The same authors say that software agents exhibit a number of features that make them different from other traditional components including autonomy, goal-orientation, collaboration, flexibility, self-starting, temporal continuity, character, communication, adaptation, and mobility.

The reason behind our motivation to integrate software agents with web services is driven by the fact that agents put in practice the concept of mobile code, and through coordination with their flexible architectures, can easily be adapted to highly dynamic and heterogeneous environment as the web. Web services however are the fast emergence of dominant means for connecting distributed applications through well established Internet protocols.

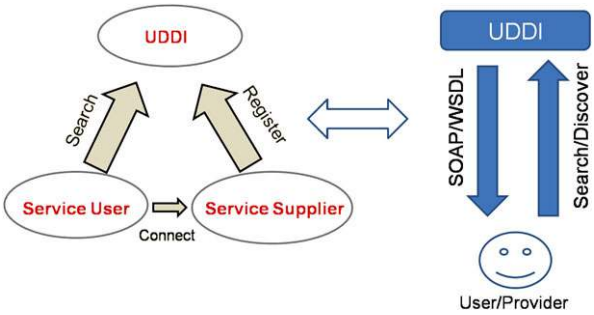

Fig. 8.2 Classical SOA vs. tailorable SOA

Furthermore, software agents can be one of the essential developments to web services for the fact that they are functional entities instead of being just simple interaction delegations or communication means [31]. To sum up, the idea is to explore the capacities of agents' proactive interactions to enhance the behavior of web services in a service-oriented architecture (SOA). With this paradigm, software components, where each one is representing a service and an agent in collaboration, can interact with each other for providing unified services in a specified environment, as for example the exchange of multimedia applications on the Internet (we are currently working on such system, we call it Oce@nyd). This is aligned with the authors in [31]: "agents will become an essential part of most web-based applications, serving as the 'glue' that makes a system as large as the web manageable and viable."

8.4.6.1 Classical SOA vs. Tailorable SOA

In Fig. 8.2, we can see the transformation of the classical SOA found in the literature to our vision of a tailorable SOA. In the classical SOA, there exist two actors: the service provider that registers the definitions of web services (WSDL) in the public registry (UDDI). The user in this kind of architecture has only the possibility to send SOAP requests to interrogate the UDDI about a needed service, but does not have the possibility to modify the UDDI by adding new services that could satisfy more his or her needs. This limits the use and the flexibility of the approach, as users would only be limited to use the services already existing in the system, and thus wouldn't be able to adapt the application to their needs, but rather the other way around.

In the tailorable SOA, the idea is to modify the structure of the classical SOA in a way that the service user is the service provider himself, as we can see in Fig. 8.2. In other words, the user will then have the privilege to interrogate the UDDI (self representation of the application, as seen in the reflexive computational system in Sect. 8.3.2) using standard SOAP requests, but also modify it using the same type of messages formats by adding the new web services definitions into the UDDI with mechanisms that will assure this type of modification. The protocols provided

(SOAP) in the SOA will be in charge of reconfiguring the links between the services added and the services already present in the system. In fact, the self representation part could be seen as an open implementation mechanism [16] where the users would be able to modify the structure of the application (inserting new service definitions through their WSDL files) without recompiling the system and stopping its execution. Also, this kind of system will satisfy the evolution of the system's use due to temporal or behavioral changes. In this case, the classical Service-Oriented Architecture will be transformed into a tailorable Service-Oriented Architecture by giving the user tools to accommodate these changes. In fact, the dynamic integration of new web services will be the task of software agents, where comes the main purpose of integrating software agents with web services in our system, and that is to insure service tailorability in a collaborative environment, as we will see in the next section.

8.5 The UD^3 Theoretical Model

As mentioned earlier, the aim behind our model is to integrate software agents and web services into a cohesive entity that attempts to surpass the weakness of each technology, while reinforcing their individual advantages [31]. W3C clearly expresses the notion that, "software agents are the running programs that drive web services—both to implement them and to access them as computational resources that act on behalf of a person or organization." In fact, the concept of software agents is older than web services and it has been employed with success for executing distributed applications, while their main aim is based on the fact that users only need to specify a high-level goal instead of issuing explicit instructions, leaving the how and when decisions to the agent to discover web services deployed on the Internet and integrate them in the system. This reinforces the reason behind our motivation to integrate software agents with web services, and is driven by the fact that agents put in practice the concept of mobile code, and through coordination with their flexible architectures, can easily be adapted to the highly dynamic and heterogeneous environment as the web.

We extend the work in [4] for the use of SOA in the design of a tailorable groupware, as it offers the needed interoperability and reconfigurability between system components, and the importance of using software agents in order to enhance the discovery of web services by making them proactive and dynamic. Moreover, we rely on the Arch model [1] by offering a canonical decomposition of the main structure of the system into five main components (Functional core, Functional core adapter, Physical Interaction, Logical Interaction and Dialog Controller components), each having a specific functionality in the system. However in our work we will concentrate on the design of the functional core which is the main component of the system, along with the system interfaces, and we will make no assumption about the other components. We rely also on Dewan's model [5] that structures a groupware system into a variable number of layers, each representing

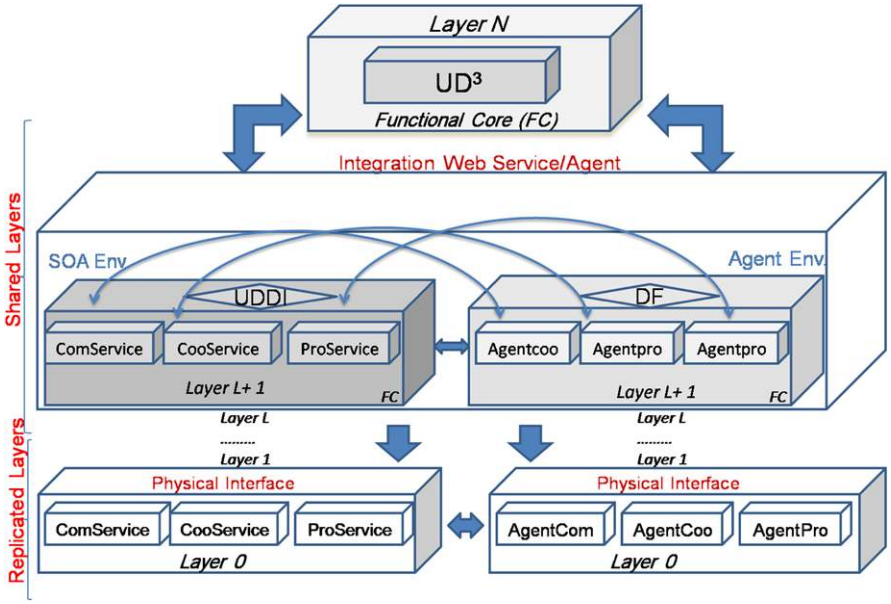

Fig. 8.3 The UD^3 model

specific levels of abstraction, where the highest layer is the semantic layer that corresponds to the functional core of the system (coincides with the one of the Arch model), and the lowest layer representing the material or the hardware level (Arch's Physical Interaction component), and eventually we compare our model with the clover model [17] that is itself built using the later models. Note that Fig. 8.3 representing our proposed architecture shows only the functional core of the system, along with the physical interaction layer that implements the interactions with the user. In the next section, we implement the physical component as a web interface serving as a case study of our model.

8.5.1 Description of the Functional Core (FC)

The overall architecture as we can see in Fig. 8.3 is constituted of a root representing shared layers, meaning that it is shared among all the users in the system, and several branches constituted by replicated layers for every user. The layers communicate vertically using interaction events, and use collaboration events for communication between layers of different branches. However, in contrast to the clover model [17] where the functional core is also split into two layers: one private and shared, while the other is replicated and public, the functional core in our model is represented by two layers that are both shared and constitute the root of the system: The first layer of the Functional Core (FC) at the level N represents the highest semantic

level in the system, while the other FC layer at the level $N-1$ is divided into two distinct parts: a service oriented environment (SOA), and a JADE agent layer. One can imagine two different environments evolving in parallel, while having a layer on the level N with the essential requirement of projecting the two environments on the level $N-1$, hence integrating web services with the corresponding software agents. The use of two shared layers as a functional core is to increase the separation of functionalities, and thus to increase the modularity of the code. In this article, we will skip the details about the layers between the functional core and the physical interaction components, and we will concentrate on the essence of the architecture represented by its functional core composed of web services and agents, and the interfaces that residing on the lowest layer (Layer 0) of the system.

8.5.2 FC Decomposition

The shared layers of the architecture constituting the system's FC enable all users to manipulate domain objects and have access to various services during the interaction with the system, while the replicated layers handles the set of services and the state of the system that is private for every user in collaboration. We extend this layer abstraction as in [17] by decomposing each layer of the architecture into subcomponents, each dedicated to one facet of Ellis' 3C model, while providing and managing specific services for communication, coordination and production (defined by the term cooperation in [8]). However, we suppose that only the layers on the level $N-1$ and on the lowest level (Layer 0) satisfy these three main classifications, while we have made no assumption till now about the decomposition of the highest semantic layer in the architecture, that is for us mainly composed of one single component for integrating web services with agents, as we will see later in the description of our model. The sub-components on the level $N-1$ are enclosed in a software interface exposing its functionalities to the clients, by dividing the services in the system into three main services: communication, coordination and production services.

Indeed, for [28], a component is a distributable and executable software module that provides and receives services through a well-defined interface. Specifically, a component is a module, an object, a unit of calculation or data. Hence we concretize the notion of component by defining it as a web service and a software agent interacting together and creating a dynamic environment for groupware tailoring, by offering common and unified services to satisfy users' preferences. We will explain in more details the FC of our system, beginning with the layer $N-1$ that encloses an SOA environment and a JADE agents' environment respectively.

8.5.3 SOA Environment

As we can see in Fig. 8.3, the first component on the level $N-1$ is based on an SOA environment. This component contains all the web services in the system grouped

into 3 main services: communication services, coordination services and production services. By classifying services in the system into these three main categories, the main spaces of the software collaboration process defined by the 3C model [8], as we have mentioned, are satisfied. Note that we use the term 'Production' to mean 'Cooperation' of activities (used in the 3C model: Communication, Coordination and Cooperation):

- ComService: contains all services offering means of communication between users in collaboration (videoconference service, voice recorder service, etc.).
- CoorService: contains services implementing rules of coordination by codifying their interaction (i.e. workflow).
- ProService: contains services that are the collaborative product of using the architecture. (Ex: Paint application, Word document, etc.).

These services can be considered as orchestrations of various other services in the system [29], and include services based on the functionalities they offer. Compared to the architecture proposed in [4], the UDDI is viewed as a dynamic registry for web services description enhanced with software agent's capabilities, and containing definitions of services running in the system that are susceptible of undergoing tailorability activities. The definitions of these web services are provided using the standardized language for web services description (WSDL) and are connected to adaptable aspects that are the services themselves, residing in the SOA environment.

8.5.4 JADE Agents' Environment

In parallel to the SOA environment, a JADE environment constitutes the other part of the FC on the level $N-1$. This layer is populated with software agents that are deployed on a JADE environment using its libraries for implementing agents' behaviors. The adopted paradigm of communication between layers is an asynchronous message passing with a format specified by the ACL (Agent Communication Language) defined by FIPA [9]. This format includes a number of fields, more specifically the sender of the message, the list of recipients, the communication intention that indicates the purpose of sending the message, the message content and its language i.e. the used syntax to explain the content that could be understood by the sender and the recipient, and finally the ontology i.e. the vocabulary of symbols used in the contents and their meanings that also should be understood both by the sender and the recipient of the message. As in the SOA environment, all agents are grouped into three main classes: communication, coordination and production agents. The use of agents, as we have mentioned before, is to make the discovery of new services in the system dynamic, meaning that new web services will be actively integrated into the FC without stopping the execution of the system. These services are normally used by users on the physical layer, which is the lowest layer in the architecture. The functional decomposition of the layer into three main subcomponents corresponding to the 3C model will fasten the interaction with web

services in the system, while every agent in one particular sub-component would know exactly where to search for a particular web service in the SOA environment that best suits the functionalities it can offer. Each sub-component in this layer manipulates semantic objects dedicated to one of the 3C model functionalities, and performs specific processing functions on its services.

8.5.5 Universal Directory for Description and Discovery

We describe the highest semantic layer of the architecture constituting the core component in the proposed model. The name of the UD^3 model is derived from integrating the UDDI [24] (Universal Description Discovery and Integration) used by web services and the DF [36] (Directory Facilitator explained in Sect. 8.4.4) used by software agents. Hence emerges the Universal Directory for Description and Discovery, or the UD^3 model. In fact, building a model using both technologies would constitute a new approach built on integration's synergy, which is still not been exploited till now in the context of tailorable groupware design. We can view this component on the highest level of the architecture as a shared registry with dynamic interaction mechanisms for the discovery of web services. We visualize a scenario where a client agent searches for a service in the system. The model can then trigger a mechanism to look up for available services in the web service environment. Hence the advantages of classifying web services into three main services (Communication, Coordination and Production) for making the search for available web services faster and more efficient by separation of their functionalities. If the web service is found, its corresponding ontology and interaction models are generated, then the agent registers the corresponding web service as its own service, and communication between the invoking agent and the service would be able to start. Hence, the aim of this part of the FC is for:

- Software agents to discover publish and invoke web services in web service registries (UDDI).
- Web service clients to discover software agents services in the Directory Facilitator (DF) of the JADE Platform.
- Web services to be published in the Directory Facilitator (DF) as a agent service.
- Web service clients to invoke Software Agents in order to integrate new web services in the system.

8.5.6 Dynamic Discovery and the Semantic Web

Existing syntax based on SOA standards such as WSDL allows for dynamic invocation of web services. However, this is only true for web services that have already been discovered, as WSDL does not allow attachment of semantic meaning to data in order to discover external web services on the fly. Dynamic discovery, on the

other hand, needs to deal with unknown services on the Internet. It will hence require the use of semantic metadata that software agents can understand and interpret without human intervention. Hence a specific language should be used to add semantics to inputs and outputs of the web service by attaching semantic metadata that reference ontology concepts. Agents can then understand service descriptions without intervention from the human user, thus enabling dynamic discovery as well as dynamic invocation of web services, and hence tailoring groupware services in relation to user's preferences. The W3C community developed the Web Ontology Language (OWL) to address this problem [26]. It is a machine-understandable description language that is capable of describing resources in a richer manner than traditional flat taxonomies and classification systems, by providing a set of concepts specifically for describing web services inputs, outputs, operations, and categorization. However, UDDI is not capable of storing and processing semantic service descriptions written in OWL. Therefore it is clear that a registry that supports semantic annotation and matchmaking of web services will produce much more refined search results, invocation and integration mechanisms which we are exploiting in our model by implementing services' tailorability.

8.5.7 FC Implementation

As we have mentioned earlier, the FC on the level *N* of the architecture as shown in Fig. 8.4, should allow mechanisms for translating web services' invocations into a language understood by software agents, and vice versa. In fact, few related work in the literature have been identified dealing with such translation mechanisms. The authors in [25] present their tool, WS2JADE, that is based on two distinct layers: An interconnection layer that glues agents and web services together, and a static management layer that creates and controls these interconnection entities called WSAG (Web Service Agents), that are able to communicate and deliver web services as their own services by producing and deploying WSAG at runtime. For [21], agents can represent service consumers and providers that are independent, and collaborate together in order to dynamically configure and reconfigure service-based applications. Their approach implements an agent-based architecture and is realized in a web service agent platform (WSAF) that uses QoS ontology and an XML language enabling consumers and service providers to expose their preferences.

Our primary concern in the design of our model is to have a tool translating web services' and agents' invocation messages, and creating interaction mechanisms in order to tailor web services in the system. In fact, a useful tool in using JADE is the WSIG ('Web Service Integration Gateway') that meets our needs by providing means to register web services in the JADE DF [36] (see Sect. 8.4.4) "mapped" with descriptions of agents. In our case, registered web services can be called by agents by directing the invocation to the WSIG. Thus, a web service is published as JADE agent service, and an agent service can be symmetrically published as an "end-point" of a web service. As shown in Fig. 8.4, the highest level of the FC

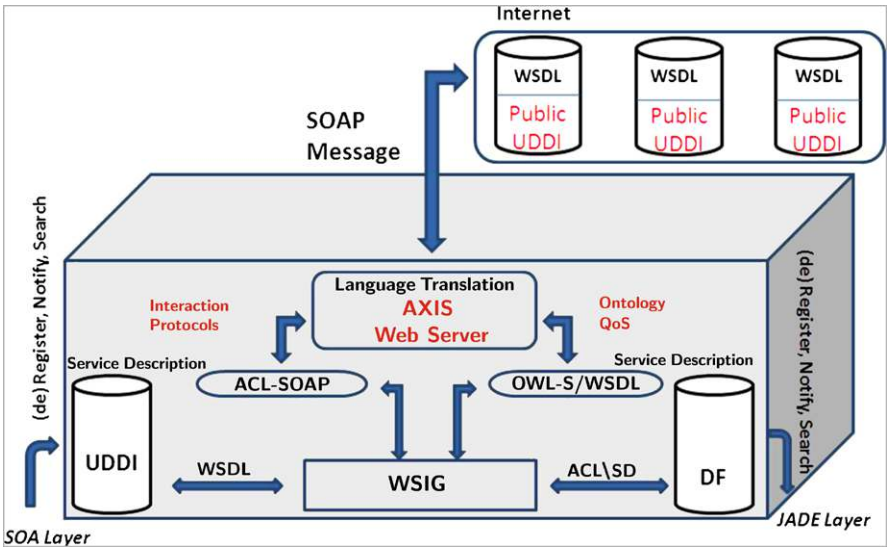

Fig. 8.4 FC implementation

contains a WSIG consisting of several components, each linked either directly or indirectly to two registries, the DF that is not visible outside the platform, and the UDDI that is visible internally to the WSIG, and externally to web services and web service clients, but not directly to agents, and hence the WSIG acts as the interface point between the agents and SOA environment.

In order to be visible in both environments, WSIG is registered as a special agent service in DF (Directory Facilitator) and a special web service endpoint in the UDDI directory, where any service description registered with either the DF or UDDI is automatically translated into an entry for the other. The purpose is to ensure that any registered web service is visible to agents via the DF and any registered agent is visible to web service clients via the UDDI. We describe the local components of the WSIG [13] of the JADE platform:

- OWL-S/WSDL codec (OWL-S is the semantic markup for web services [12]) that has the purpose of generating OWL ontologies from WSDL specifications. In fact, WSDL is known to have limited ability in order to express semantic information about the web service in order for it to be manipulated by an agent in the system. And thus, a specific need is to have a codec mapping WSDL elements or operations into OWL-S atomic processes, and alternatively, OWL-S class types of the inputs and outputs of an atomic process mapped onto WSDL elements. Of course, not all the operations in the WSDL can be mapped into OWL descriptions, therefore some descriptions would be added manually by the programmer. A scenario of use is described in Sect. 8.6.3 for further understanding of the invocations process between an agent and a web service.
- ACL-SOAP codec that is responsible for parsing ACL (Agent Communication Language [36]) messages received from the DF in order to extract the encoded

service descriptions (SD) held within their content, then translating them into a web service invocation and returning the results to the WSIG for registration in the UDDI. It is also responsible for parsing ACL messages sent to the WSIG to invoke a web service into a corresponding SOAP message. This codec operates also in a bidirectional manner in order to translate SOAP and service specification information into correctly encoded ACL messages and DF entries [13].

- Axis' JAX-RPC (Java API for XML based Remote Procedure Call) is an application program interface (API) that enables Java developers to include remote procedure calls (RPCs) with any web-based applications. It is aimed at making it easier for applications or web services to call other applications. The JAX-RPC programming model simplifies the development by abstracting SOAP protocol-level runtime mechanisms and providing mapping services between Java and the web services description language (WSDL).

As we can see in Fig. 8.4, language translation between the two environments for making the discovery and integration of new services in the system dynamic, leads in our model to explicitly implementing tailorability in a collaborative system, which is leveraged by reusing and identifying existing technologies instead of reinventing the wheel [25], with the use of the WSIG [13] component in the JADE platform. In the next section we will see how the UD^3 model can be put into practice in the Oce@nyd project.

8.6 Case Study—Oce@nyd

Figure 8.5 illustrates the application of the UD^3 on an ongoing project in our laboratory, Oce@nyd, which is a part of a national project DIGITALOCEAN [6] for the distribution and creation of multimedia applications (audio, video, text, etc.). In fact, it is designed to enable the public to discover, online, underwater environments. Our aim is to deal with the collaboration aspects of the project, as well as questions concerning its integration, interoperability and mainly tailorability.

Moreover, as the nature of collaborative work in this domain continually changes as a consequence of changing work needs, the system must adapt to reflect the unpredictable requirements of users' in collaboration. Hence emerges the need for a new architectural model that could satisfy these requirements in a real application. We have applied our model on Oce@nyd in order to satisfy these problems.

At the conceptual level we applied the FC shown in Fig. 8.3 that includes two shared layers. At the implementation level, Oce@nyd is a client–server application deployed on a Netbeans [23] platform using JADE libraries, along with other libraries for implementing the web services environment in the system. Both shared layers of the FC are deployed on the server side and other layers are replicated on users' machines, while the client/server communication is based on network streams. We will discuss mainly the FC of the model in a real application along with the physical interaction layer and the services offered that are tailored by users, while we make no assumptions, as we have mentioned before, for the other layers between the FC core and the physical interaction.

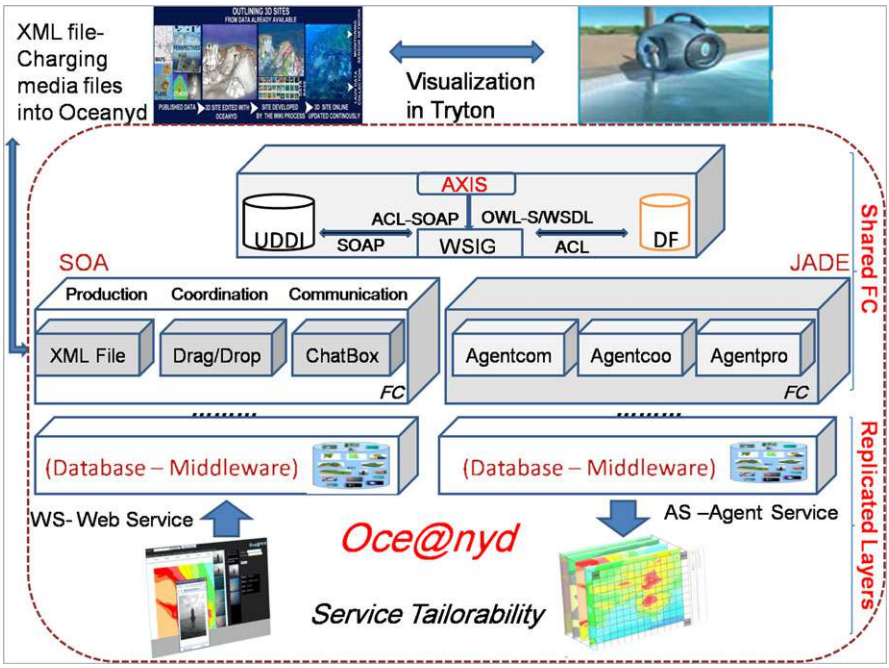

Fig. 8.5 Case study—Oce@nyd

8.6.1 Physical Layer

At the physical layer of the application, lays a web interface manipulated by users in collaboration. This web interface enables users to drag/drop multimedia files on a map of a specified underwater site taken by another partner for the project, using sensors, cameras and a GPS. The aim is to enrich this map with multimedia files by professional divers or users having real photos taken underwater when scuba diving in this particular location. We assume for now that the interface offers three mechanisms implemented with web services technologies, each dedicated to one aspect of the 3C model as follows:

- Communication: the application provides a chat mechanism enabling users to exchange information about the files dragged onto the loaded map.
- Coordination: a service is encapsulated into the system that divides the map into variable zones, and which will detect the coordinates of the dragged file onto the shared zone that is showing a loaded map of a particular underwater site (the number of zones dividing the map will be defined by the user, hence insuring tailorability of coordination capabilities in the system).
- Production: the system provides a mechanism for dumping the information of the multimedia applications dropped by users, into an XML file defining data attributes for every file dragged onto the map: date/time, nature (image, video or text), description (user's description of a particular file), and coordinates: the

user will be prompt to enter the exact coordinates of the file (x, y, z) taken in a particular underwater site. In the case where the actual coordinates are missing, the system will give an (x, y) coordinates of the file dropped by the user on the 2D loaded map, while a (z) coordinate will be given by the administrator depending on the description of the user of the particular file.

8.6.2 Shared FC

These services reside in the functional core of the system, more particularly in the SOA environment while their WSDL files reside in the UDDI implemented one level up in the conceptual model, along with the DF and a set of codec provided by using JADE libraries. The main aim is to ensure consistency by binding one agent to its corresponding web service via the WSIG as described in the previous section. In the JADE part of the system, every agent is bound with its corresponding web service. For example, a communication web service is managed by a communication agent insuring its integration and invoking in the system's physical interface. The part constituting the FC that contains all the agents are handled by the WSIG and JADE tools, providing basic methods for the registration of agents and their communication with the SOA. The agents use classes inherited from various behaviors offered by JADE libraries for implementing interaction protocols between agents: CyclicBehaviour that listens to messages exchanged between various agents, AchieveREInitiator/AchieveREResponder that provides an effective implementation for all the FIPA-Request-like interaction protocols, Contract NetInitiator/ContractNetResponder that offer various API and functionalities [13], etc.

8.6.3 Invocation of an External Web Service by an Agent

In the case where the system is prompted for an external web service, an agent handling the particular class of the web service (communication, coordination or production) sends an ACL Request message to the WSIG containing the identity, name or any parameters identifying the web service to be invoked. Received ACL messages are parsed and a SOAP message is constructed in order to prompt external public registries for this particular web service, using the WSDL of the service to be invoked. If the web service is found and a response is expected, a temporary endpoint is established on the web server in order to receive responses, where the incoming SOAP message is parsed into an ACL Inform message and sent to the invoking agent, while registering the service in the local UDDI as a regular web service. This particular agent will then be responsible for offering the requested web service to the user directly integrated into the web interface.

Hence, on the design level, web services can act as semi-autonomous agents that can be employed for describing the external behaviors and services offered by

software agents, and where every agent works in relation to the environment as a regular web service. In consequence, agents are used to establish high level, flexible and dynamic interaction models, while the web services will be more appropriate for resolving the problems of interoperability in the system. At the execution level, UDDI WSDL and SOAP will provide capacities such as the discovery, deployment and communication.

8.6.4 Properties and Discussion

The originality of our model is the use of existing technologies' synergy in order to create a tailorable and interoperable architecture for groupware. Moreover, our model is inspired by the Arch and Dewan's model for separating the core functionality (logic of the application) of its interfaces, and thus carrying with it many essential properties such as modifiability, which is also crucial in the HCI domain. However, the two layers constituting the FC are both shared and handle exclusively the services and their dynamic integration, which is different from the clover model [17] that advocates a replicated functional core for every user by managing their private domain-dependent objects. We thought that a functional core adapter situated between the functional core and the physical layer (which was not discussed in our work presented) is more suited to handle this type of data, while dedicating the core of the application solely to handle tailoring system's services, and hence every newly added service will be shared by all users' participating in a particular session.

The functional core breakdown according to Ellis' 3C model contains several properties. In fact, from the implementation perspective, the functional breakdown will result in a greater modularity which reduces the complexity of groupware's implementation. For example, In the Oce@nyd system, it would be easy to add a new communication web service by adding, for example, a video stream service without affecting existing web services in the system. In addition, we have made no assumptions about the other layers in the system according to the functional decomposition of Ellis' model. This could reduce the development cost and computational time, while enabling the addition of independent and heterogeneous layers to improve the distribution of features and increase the modularity of code, and also by insuring interoperability on every layer of the architecture (by using FIPA and W3C specification and standards). As for the branching point discussed in [27], we have fix it after the FC layers, which induce a lower replication degree than the Clover model, but convenient in order to ensure state consistency of services, as well for collaborating users to share discovered services and reusing them when needed. Finally, our model identifies the implementation architecture that is deduced from the theoretical model in order to achieve tailorability in collaborative applications, where opposed to other models, it identifies explicitly a component as a web service and a software agent collaborating together to offer unified services in a specified environment, and where agents implement the concept of mobile code, while coordination with their flexible architecture would enable them to easily adapt to highly dynamic

and heterogeneous environments as the web. Furthermore, this model accepts equity between roles of agents and web services to support tailorability, which is different from the traditional view that agents are considered on an upper level from web services and take solely the roles of web services providers and consumers.

8.7 Conclusion

In this work, we have proposed a new architectural model that supports tailorability in CSCW domain, where existing models are still lacking in putting it forward and identifying technologies supporting it in the field. Our model relies on the integration of web services and software agents that build system's components, and offers interoperability between heterogeneous applications by providing a synergy of technology used for the dynamic discovery and integration of web services. This leads us to conceive a totally innovative approach, where research about web services and agent's integration is, until now, never been exploited in the context of groupware tailorability, and hence bringing innovation in both the CSCW domain and the web. Moreover, our model relies on the Arch and Dewan's model, offering a canonical separation of the application's core from its interfaces, thus greater modularity, and also on Ellis' model by decomposing system's functionality into three facets: communication, coordination and production, which satisfies various properties in the HCI (Human–Computer Interaction) and CSCW (Computer Supported Cooperative Work) domains. However, storing and processing semantic service descriptions (OWL-S) in existing UDDI registries may not be the ideal solution in the long run, while compared to registries specifically designed to handle semantic service description and queries, it will have drawbacks with respect to some functionalities as well as efficiency in terms of both speed and storage. However, our approach will provide a cost effective and functional short-term solution that builds on top of existing registry infrastructure.

In our future work, we aim to complete our implementation for the UD^3 model. Our concern will be to modify incoming SOAP messages' headers of external web services in order to fasten their integration into the architecture according to their functionalities (communication, coordination and production). We believe that our preliminary approach for groupware tailorability will continue to mature through the use of web services and software agents, which revealed to be appropriate to bring this concept from theory to practice.

References

1. Bass, L.: A metamodel for the runtime architecture of an interactive system. SIGCHI Bulletin **24**(1), 32–37 (1992). User Interface Developers' Workshop
2. Bourguin, G.: Un support informatique a l'activite cooperative fonde sur la Theorie de l'Activite- le projet DARE. Thesis in computer science, University of Lille, France (2000)

3. Bourguin, G.: Les lecons d'une experience dans la realisation d'un collecticiel reflexif. In: Proc. of 15th IHM Conference, pp. 24–28 (2003)
4. Cheaib, N., Otmane, S., Mallem, M.: Integrating Internet technologies in designing a tailorable groupware architecture. In: Proc. of 12th IEEE CSCWD, Xi'an, China, pp. 141–147 (2008)
5. Dewan, P.: Architectures for collaborative applications. Computer-Supported Cooperative Work **7**, 169–193 (1999)
6. Dinis, A., Fies, N., Cheaib, N., Otmane, S., Mallem, M., Nisan, N., Boi, J.M., Noel, C., Viala, C.: DIGITAL OCEAN: A national project for the creation and distribution of multimedia content for underwater sites. In: Proc. of 14th International Conference on Virtual Systems and Multimedia (VSMM'08), Limassol, Cyprus (2008)
7. Dustdar, S., Gall, H., Schmitt, R.: Web services for groupware in distributed and mobile collaboration. In: Proc. of 12th Euromicro Conference on Parallel, Distributed and Network-Based Processing, pp. 241–247 (2004)
8. Ellis, C.A., Wainer, J.A.: Conceptual model of groupware. In: Proc. of CSCW, pp. 79–88. ACM, New York (1994)
9. FIPA: http://www.fipa.org/ (2008)
10. Foukarakis, I.E., Kostaridis, A.I., Biniaris, C.G., Kaklamani, D.I., Venieris, I.S.: Webmages: An agent platform based on web services. Computer Communications **30**(3), 538–545 (2007)
11. Gannod, G.C., Burge, J.E., Urban, S.D.: Issues in the design of flexible and dynamic service-oriented systems. In: Proc. of SDSOA'07: ICSE. IEEE Comput. Soc., Washington (2007)
12. http://www.w3.org/Submission/OWL-S/
13. JADE Board: JADE Web Services Integration Gateway (WSIG) Guide, Whitestein Technologies AG, Zürich (2005)
14. Jorstad, I., Dustdar, S., Thanh, D.V.: A service oriented architecture framework for collaborative services. In: Proc. of 14th IEEE International Workshops on Enabling Technologies, Infrastructure for Collaborative Enterprise, pp. 121–125. IEEE Press, New York (2005)
15. Khezami, N.: Vers un collecticiel basé sur un formalisme multi-agent destiné à la téléopération collaborative via Internet. Phd thesis, University of Evry Val d'Essone, Evry, France (December 2005)
16. Kiczales, G., Lamping, J., Lopes, C., Maeda, C., Mendhekar, A.: Open implementation design guidelines. In: Proc. of 19th International Conference on Software Engineering, pp. 481–490. ACM, New York (1997)
17. Laurillau, Y., Nigay, L.: Clover architecture for groupware. In: Proc. of the 2002 ACM Conference on Computer Supported Cooperative Work, pp. 236–245. ACM, New York (2002)
18. Maamar, Z., Akhter, F., Lahkim, M.: An agent-based approach to specify a web service-oriented environment. In: Proc. of WET ICE, pp. 48–49. IEEE Comput. Soc., Washington (2003)
19. Maamar, Z., Sheng, Q.Z., Benatallah, B.: Interleaving web services composition and execution using software agents and delegation. In: AAMAS Workshop (2003)
20. Matskin, M., Küngas, P., Rao, J., Sampson, J., Petersen, S.A., Link, I., Back, J.: Enabling web services composition with software agents. In: Proc. of IASTED, pp. 15–17 (2005)
21. Maximilien, E.M., Singh, M.P.: A framework ontology for dynamic web services selection. IEEE Internet Computing **8**(5), 84–93 (2004)
22. Morch, A.: Three levels of end-user tailoring: customization, integration, and extension. In: Computers and Design in Context, pp. 51–76. MIT Press, Cambridge (1997)
23. Netbeans Platform. http://www.netbeans.org/
24. Newcomer, E.: Understanding Web Services: XML, WSDL, SOAP, and UDDI. Pearson Education, Boston (2002)
25. Nguyen, T.X., Kowalczyk, R.: WS2JADE: Integrating web service with Jade agents. In: Service-Oriented Computing: Agents, Semantics, and Engineering, pp. 147–159. Springer, Berlin (2007)
26. OWL Web Ontology Language Reference. Copyright W3C. http://www.w3.org/TR/owl-ref/ (2004)
27. Patterson, J.F.: A taxonomy of architectures for synchronous groupware applications. SIGOIS Bulletin **15**(3), 27–29 (2005)

28. Payet, D.: L'enrichissement de message comme support pour la composition logicielle. Phd thesis, University of Monptellier, France (2003)
29. Peltz, C.: Web services orchestration. A review of emerging technologies, tools and standards. Hewlett Packard White Paper (January 2003)
30. Roseman, M., Greenberg, S.: Simplifying component development in an integrated groupware environment. In: Proc. of 10th Annual ACM Symposium on User Interface Software and Technology, New York, pp. 65–72 (1997)
31. Shen, W., Hao, Q., Wang, S., Li, Y., Ghenniwa, H.: Agent-based service-oriented integration architecture for collaborative intelligent manufacturing. Robotics and Computer-Integrated Manufacturing **23**(3), 315–325 (2007)
32. Slagter, R., Biemans, M., Hofte, H.T.: Evolution in use of groupware: Facilitating tailoring to the extreme. In: Proc. of CRIWG, pp. 68–73 (2001)
33. Slagter, R., Biemans, M.: Designing tailorable groupware for the healthcare domain. In: Proc. of CRIWG, pp. 58–73. Springer, Berlin (2003)
34. Starr, B., Ackerman, M.S., Pazzani, M.: Do- I-Care: A collaborative web agent. In: Conference on Human Factors in Computing Systems, pp. 273–274. ACM, New York (1996)
35. Stiemerling, O., Cremers, A.: Tailorable component architectures for CSCW-systems. In: Proc. of 6th Euromicro Workshop on Parallel and Distributed Programming, pp. 21–24 (1998)
36. Telecom Italia Lab: JADE (Java Agent Development Framework). http://sharon.cselt.it/projects/jade/