Facilitating Dynamic Collaborations with eContract Services

Shiping Chen, Surya Nepal, Chen Wang, John Zic
Networking Technologies Laboratory, CSIRO ICT Centre, Australia
{Firstname.Lastname}@csiro.au

Abstract

Electronic Contract (eContract) has been recognized as a good combination of technical specification and legal documentation for establishing and regulating virtual organizations built for dynamic collaborations. This paper presents a design and implementation of an eContract service with an aim of providing a trusted collaboration platform for collaborators. The implemented service uses web services technologies to facilitate its collaborators not only to contribute resources in an eContract, but also to negotiate and instantiate them through eContract. This paper describes the interface and protocols for the eContract service. The architecture, interface and protocols designed for the service are demonstrated using an example of providing universal connectivity service for telepresence application in the context of eResearch domain.

1. Introduction

Recently, there has been many interests in forming on-demand collaborations between companies, who although competitors, wish to collaborate on occasion because of some mutual business interest. Collaborations built around this concept are termed dynamic collaborations [1]. For example, in the eResearch domain, a number of research institutes may come together to solve a challenging scientific problem by forming a dynamic collaboration that lasts for the duration of a project where each institution contributes unique resources that collectively enable the solution towards the problem [2].

To establish such dynamic collaborations, all participants involved in the collaboration require a standard way to specify and/or agree: (a) what resources they require (requirements); (b) who contribute these resources (contributions); (c) how to access these resources (Access Polices). Moreover, the participants may also have to explicitly add the corresponding terms and conditions to the above technical specifications. As we can see, a contract that combines both technical specifications and legal declarations is a natural starting point to establish a dynamic collaboration. An electronic version of such a contract, whose notations and semantics can be interpreted and processed by software system is called eContract [3].

eContract has been recognized as a powerful concept (as well as a data model) for controlling and automating Business-to-Business (B2B) collaborations. The literature of eContracts has been spread across various aspects and functionalities of eContracts, including eContract representation (XML, ebXML, ECML, tpaML, RuleML) [4], eContract negotiation [5] and eContract enforcement and monitoring [6]. However, there are not many studies on applying eContracts in a concrete application.

Another driving force of this work is Web Service and Service-Oriented Architecture (SOA). The concepts and technologies behind Web Services and SOA have proven to be successful in dealing with issues related to interoperability across autonomous systems. A stack of Web Services standards such as WSDL, UDDI and SOAP are defined to build platform-independent services. Recently, resources such as storage and networking infrastructures, tools, software and data are implemented available as services to users. For example, a concept of Software-as-a-Service (SaaS) is introduced for software and Infrastructure-as-a-Service (IaaS) for storage [7] and networking infrastructure [12]. Therefore, it is possible to define and share resources as services in the context of dynamic collaborations, and compose all these (contributed resources and the third party services) to deliver dynamic collaboration services based on an eContract service.

In this paper, we present an implementation of an eContract service for telepresence-based dynamic collaborations in the context of eResearch. The eContract service operates as a web service that facilitates its clients to discover (or being discovered by) potential collaborators, form a collaboration via an eContract and establish the collaboration platform by instantiating the eContract. We have developed a framework for Web Service Collaborative Context Definition Language for dynamic collaboration, called WS-CCDL. It enables collaborating partners to unambiguously define an eContract. Our eContract service uses this framework to specify the collaboration participants’ details and resources requirements and contributions. A telepresence application is used to demonstrate how researchers get universal connectivity to each other with the eContract service. We next describe the motivations for this work.

2. Background and Motivations
The motivations for the work presented in this paper come from the Research and Development works in the telepresence. Telepresence refers to a set of integrated technologies that allow people, who are located on different places, to feel they are present at the same location, via vision, sound, and remote manipulation. The recent advances of ICT technologies (computers and networks) have significantly improved the quality of telepresence and made it feasible to be applied in various domains, such as teleconferencing, health, and education.

A number of telepresence projects have been conducted in CSIRO ICT Centre in the past several years, which led to the following telepresence systems for different applications:

**Virtual Critical Care Unit (ViCCU)** is a set of high-end telepresence units [8], which allows a specialist located at one hospital to supervise a resuscitation team located at a peripheral hospital as shown in Figure 1. ViCCU is designed so that all information required by the specialist to make judgments on patient treatment is available in real time, as if the specialist were present at the peripheral hospital. The real-time information streaming is done using the dedicated high bandwidth network and the sophisticated and fine tuned video/audio software.

![Figure 1. Two deployed ViCCU in action](image1)

**Braccetto** (meaning ‘arm in arm’ in Italian) is another telepresence platform [9], which aims at supporting distributed intense collaboration among creative teams. It consists of a suit of special devices (big screens, speakers, microphones, projectors, as shown in Figure 2) distributed across multiple sites, connected via high speed network, and operated with various application specific software/tools built on the underlying middleware (e.g., Avis etc). Braccetto is designed to be used in intense interactions involving highly serious and confidential collaborations, such as conferencing for combat planning in military application.

![Figure 2. Application of Braccetto](image2)

**Virtual Terminal (VT)** is a cross-platform high performance remote desktop sharing program [10]. The use of VT in remote education enables lecturers to view the screen of any students’ computers, which can be distributed in different locations but connected within a virtual classroom. Furthermore, the lecturer can also take control of a student’s computer and guide them through a problem.

While the above telepresence systems focus on different applications, these applications share the following common requirements:

**Universal Connectivity:** The above telepresence systems are supposed to be deployed at multiple sites, which are usually located in different places and owned by different autonomous organizations. Therefore, the foremost requirement for such systems is to enable an easy connection to each other across the business and infrastructure boundaries.

**QoS-Aware Resource Allocation:** The telepresence platforms usually require high qualities of video/audio, which heavily depend on the capacity of the underlying infrastructures, i.e. network bandwidth, server speed, transport protocols used, etc. Therefore, it is highly desirable that the underlying middleware is able to ‘intelligently’ allocate, schedule and configure the above resources to meet the certain QoS for a specific application.

**Security and Trust:** Some of the telepresence platforms will be used in multi-parties, mission-critical confidential applications (such as military combat meeting). This implies a strong requirement for the system to ensure the security and trust among the collaborators at both application and infrastructure levels.

**Ease of Operations (Usability):** A use of a telepresence system usually involves a configuration and set up of special hardware devices and software at multiple sites. This implies a huge amount of technical and tedious efforts and intensive coordination for resources (hardware/software) deployment, including resources configuration, scheduling meeting, resources checking and reservation, and resources activation. It is unlikely (or at least extremely difficult) for collaborators themselves (even IT support personnel) to conduct all the
above tasks in a consistent and efficient way among all the collaborators. As a result, there is a strong requirement for these platforms to enable the above operations to be conducted AEAP (as easily as possible) and AAAP (as automatically as possible) at both applications and underlying infrastructure levels.

This paper addresses these requirements by building an eContract service which uses eContract not only to capture the requirements of the collaboration, the contributed resources and their agreements, but also automatically instantiate them. We next describe the overview of the eContract format we use in this paper.

3. eContract

As one can see in the above discussed telepresence applications, all parties involve in the collaboration contribute resources towards meeting the requirements for the collaboration such as audio-visual equipments associated with ViCCU, Braccetto and VT. The management of contributed resources during the collaboration is a critical issue in dynamic collaborations. The management includes negotiation of resources, validation of resources, instantiating of resources/collaboration, monitoring resources and releasing resources when the collaboration terminates or when a partner leaves the collaboration. For example, during the process of resource negotiation, all participating enterprises contribute resources and they must agree with each other’s contributions. Similarly, during the process of instantiation, all resources should be instantiated and made available to the collaborators.

Existing dynamic collaborations deal these issues either in an ad hoc manner or manually. There is a need of a framework that enables the definition of collaborations in such a way that it can be unambiguously specified and automatically instantiated and managed. In order to address this problem, CSIRO ICT Centre has developed a framework for Web Service Collaborative Context Definition Language for dynamic collaboration, called WS-CCDL. It enables collaborating partners to unambiguously define the requirements for the collaboration as well as agreements for all the resources contributed to the collaboration as an eContract. We next describe the key concepts associated with WS-CCDL.

The eContract only provides a template for service level agreements among collaborating parties. A party (possible trusted third party) needs to provide a service so that all collaborators can use it to reach an agreement. Our eContract supports such negotiation as it captures the agreements explicitly within the contract.

The eContract only provides a template for service level agreements among collaborating parties. A party (possible trusted third party) needs to provide a service so that all collaborators can use it to reach an agreement. Our eContract supports such negotiation as it captures the agreements explicitly within the contract.

The set of requirements include collaboration name, required resources with specific quantity and quality metrics (e.g. 1Gbps expedited forwarding), access policies, etc. The set of contributions include resource services contributed by the participants and their SLAs including access policies. The set of agreements indicate who has agreed on whose contributions. The last component in the eContract is the set of participants, which includes the initiator and the participants invited by the initiator.

We capture this definition in WS-CCDL by defining an XML schema. The high-level schema is shown below. The detail descriptions of the eContract schema and its semantics are out of the scope of this paper.

The key features in our eContract can be summarized as follows.

- **Multiple Parties** – Our eContract supports participation of multiple parties as compared to two parties in many eCommerce applications.
- **Multiple Services** – Our eContract supports multiple resource services; for example network connectivity service and Virtual Terminal service in an eResearch collaboration.
- **Multiple Roles** – In our eContract, a party can play one of the three distinct roles: service provider, service consumer and supporting third party.
- **Multi Hop Negotiation/Agreement** – A contract may need to be transferred through multiple hops before reaching an agreement. Our eContract supports such negotiation as it captures the agreements explicitly within the contract.

The eContract only provides a template for service level agreements among collaborating parties. A party (possible trusted third party) needs to provide a service so that all collaborators can use it to reach an agreement. The next section describes a contract service built for a telepresence application using the above defined eContract.

Definition (eContract): An eContract is a quadruple <R, C, A, P> where
- R is the set of requirements for a collaboration,
- C is the set of contributions made by participants,
- A is the set of agreements among the participants,
- P is the set of participants.
4. eContract Service

In order to demonstrate the above concepts, we designed and implemented an eContract service with the focus on (but not restricted to) providing a network connectivity service for telepresence applications.

4.1. Architecture Design

We design an eContract service as a third party web service to provide a network connectivity service to collaborating parties and their tools (ViCCU, VT or BRaccetto), whose overall architecture is shown in Figure 3. We next describe each component in this architecture and how they interact with each other as follows:

At the beginning, each collaborator resides within its own administrative domain, even without knowing each other. To collaborate with others, they all subscribe services provided by the eContract service so that they can either set up or be invited to join into a dynamic collaboration via an eContract.

Suppose a collaborator wants to set a dynamic collaboration with other collaborators who have a particular research interest. He can initialize an eContract by discovering and adding the collaborators whom he/she likes to collaborate with as participants. He may also list the resources (such as Braccetto-compliant devices) required for this collaboration and his resource contributions in the contract. After the eContract is submitted, the invited collaborators will be automatically informed that they are invited to participate in a collaboration. They can decide to accept or decline the invitation via the eContract service. If they decide to join the collaboration, they can negotiate the content in the eContract with all the other participants following a specific negotiation procedure and protocol [5].

Once all participants agree and sign the eContract, the eContract service will buy the required services from the third party that can provide and prepare a VM (Virtual Machine) image [11] for each participant, in which all required OS, network, and applications are installed and configured according to the eContract, for deployment. Note that the pre-configured VMware images for different participants may be different according to the participants’ role and the specifications in the eContract. Each participant can download his VM image and deploy within his own domain with little configuration. Then, the collaboration is ready to run.

While eContract monitoring and termination are important phases for a dynamic collaboration, they are not discussed in this paper.

4.2. Messaging-Oriented Interface

Web Services are supposed to be message oriented and loosely coupled. However, most of the current Web Services implementations are still RPC-based and do not support asynchronous messaging very well. This means that any updating on the service functionalities may result in changes at both service-side interfaces (WSDL) and client-side codes (generated from the WSDL). On the other hand, the schema of the eContract may also be updated from time to time to enrich its semantic and representation. This would also result in updating the eContract service interface if the schema of the eContract is visible in the interface.
To address the above two issues, we designed a single messaging-oriented interface and a generic message type as follows:

```java
MessageType Send(MessageType msg);
```

where the attribute (‘ServiceName’) is used to specify which service/operation required to process the message; AppData is a set of (XML) data used as application (functional) data for the specified service; MetaData is a set of (XML) data used as ‘extra’ (non-functional) data for the service, such as username, encrypted password, encrypt keysID, exception, performance testing etc. The details about AppData and MetaData are as follows:

```xml
ns1:AppDataType
  | attributes
  | SchemaName
  | ns1:Data
```

where Data is defined as xs:string to host any XML, including an eContract.

```xml
ns1:MetaDataType
  | attributes
  | GroupName
  | ns1:Property
```

where Property is a name and value pair that describes any properties required for a specific transaction.

This web service interface design offers the following features:

- **Stability**: Since this interface and data model is very generic, it can remain relatively stable for both client and service side code-base.
- **Extendibility**: Because of the flexible data model, users can exchange various application data and add new metadata for a variety of purposes without changing the overall data structure and re-generating underlying web service code.
- **Portability**: Having a single interface and a generic data model, it allows us to easily bind our web service to other messaging protocols and/or web services middleware.

### 4.3. Services and Protocols

To facilitate eContract-based collaborations, the eContract service provides the following basic service operations:

- **Registration**: subscribe the service by signing on;
- **SearchCollaborators**: find the collaborators;
- **SearchContracts**: find a list of eContracts;
- **GetContract**: get a specific eContract;
- **SubmitContract**: submit a eContract;
- **Deployment**: deploy an eContract;

The detailed specifications and protocols for each operation are provided in Table 1. We explain these protocols using a simple scenario, where a collaboration is formed among three participants (A, B, C), as shown in Figure 4.

### 4.4. Implementation

We implemented the above design using the following products/technologies:

- eContract Web Service: SUN Glassfish v2.0
- BackendDatabase: MySQL v5.1.x
- VPN Server: OpenVPN v2.1
- Virtual Machine: VMPlayer v2.0.1
- eContract Client: SUN JDK 1.6, including Swing and JAXB
- Virtual Terminal (VT): in-house prototype used for the case study, which is explained in the next section.

We demonstrate the eContract service using a collaborative post-filming education as an application in the following section.
<table>
<thead>
<tr>
<th>Services</th>
<th>Description</th>
<th>Items</th>
<th>Descriptions/Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Registration</td>
<td>A client can register itself by sending the following AppData.</td>
<td>Request</td>
<td>• Subject=&quot;Registration&quot;&lt;br&gt;• AppData: SchemaName=&quot;Participant&quot; Data=&quot;&lt;Participant&gt;…&lt;/Participant&gt;&quot;&lt;br&gt;• MetaData: GroupName=&quot;Security&quot;&lt;br&gt;  o Property: name=&quot;MyPublicKey&quot; value=&quot;…&quot; (for exchanging public key)</td>
</tr>
</tbody>
</table>
|               |                                                                             | Response                           | • Subject="RegistrationReply"<br>• AppData = null<br>• MetaData: Attributes: GroupName="Security"<br>  o Property: Name="userID" Value="XYZ"<br>  o Property: Name="password" Value="#$%&*&&*C"
| SearchCollaborators | A client can search potential collaborators with keywords.                  | Request                            | • Subject="SearchCollaborators"<br>• AppData: SchemaName="String" Data="Keyword1&Keyword2…"<br>• MetaData: GroupName="Security"<br>  o Property: name="UserID" value="…" |
|               |                                                                             | Response                           | • Subject="SearchCollaboratorsReply"<br>• AppData: SchemaName="Participants" Data="<Participants>…</Participants>" |
| CheckContracts | A client can check all contracts associated with itself, including invited, jointed, assigned | Request                            | • Subject="CheckContracts"<br>• AppData: SchemaName="String", Data="Setup|Invited|Joined|All"
|               |                                                                             | Response                           | • Subject="CheckContractsReply"<br>• AppData: a list of SchemaName="String", Data="ContractID|Description|State" |
| GetContract   | A client can get a specific contract with the contract ID                   | Request                            | • Subject="GetContract"<br>• AppData: SchemaName="Contract", Data="<eContract>…</eContract>"
|               |                                                                             | Response                           | • Subject="GetContractReply"<br>• AppData: SchemaName="Contract", Data="<eContract>…</eContract>" |
| SubmitContract| A client can submit a contract for different proposes                       | Request                            | • Subject="SubmitContract"<br>• AppData: SchemaName="Contract", Data="<eContract>…</eContract>"
|               |                                                                             | Response                           | • Subject="SubmitContractReply"<br>• AppData: SchemaName="String" Data="ContractID"
|               |                                                                             |                                    | • MetaData=null<br>• Property: Name="Type" Value="new|update|decline|sign" |
| Deployment    | A client deploys a contract by downloading a VMware image from the eContract service | Request                            | • Subject=Download<br>• AppData: : SchemaName="String" Data="ContractID"
|               |                                                                             | Response                           | • Subject=DownloadReply<br>• AppData: SchemaName="VMwareImage" Data=null (using SOAP attachment) |
5. Case Study

Application Scenario: A Sydney-based film institute (Participant A) wants to conduct intensive post-filming processing training courses hosted by other organizations (Participant B and C) across Australia. Since the training context is very professional and the lecturing cost is very expensive, only two small classes are scheduled to run in parallel in two cities. Due to the nature of the training course, the students are required to conduct some computer-aid post-filming process tasks in the class and the lectures need to intensively interact with each student based on the art work done on his/her computer. In addition, to protect the intellectual properties of the lecturing, the communications between the lecturer and students should be secured at both application and network levels.

Why we choose VT? The VT system consists of three basic components: VTServer (streams the local PC’s desktop, and mouse/keyboard events to the VTProxy); VTProxy (forwards data from the VTServer to all others VTClients) and VTClients (receives data and events streams from the server, rendering them out to the local screen. It also captures the local mouse/keyboard events and publishes them to the proxy that forwards the events to all other clients, including VTServer). VT enables distributed users (lecturer and students) to collaborate on a shared application (post-filming process software) on a central server (VTServer) via the network.

Why do we need eContract Service? To deploy VT for the above remote interactive lecturing scenario, we first need high bandwidth yet secure network connection across the multiple sites/domains owned by different organizations. Secondly, installation and configuration of VT is a complex and tedious job. The local IT supports may not have resources and capacity to conduct the job. In this case, it may be an efficient and cost-effective way to get connectivity and deploy VT as a single service via the eContract service.
How does it work? As shown in , first of all, Participant A, B, C registered with the eContract Service. Then, the filming institute (Participant A) initialized an eContract by inviting the two host universities (Participant B and C). In the contract, three resources are required and specified as follows:

Once all participants signed the contract, according to the eContract, the eContract will prepare a VMWare image, in which the following software is pre-installed and configured: (a) a OpenVPN client [12]; (b) VTServer and VTClient [10]. When all participants download this pre-configured VMWare image, they get all required resources (application and connectivity) to conduct the collaboration, i.e. distributed interactive lecturing in our case.

Note that the VPN Server and VTProxy can be hosted either by the eContract service or any other third party services, depending on the eContract business model. The detail discussion on it is out of the scope of this paper.

6. Conclusion

In this paper, we present a Web Services based solution to facilitate dynamic collaborations. Our solution is based on the concept of eContract that was used for collecting and negotiating the basic resource requirements for a dynamic collaboration, as well as for instantiating these resources as a single service deliverable by the eContract service. To decouple the eContract schema with the eContract web service interface, we designed a single web service interface with a generic message based data model to accommodate a variety of service operations and the corresponding protocols. We demonstrated the usefulness of the eContract service using a post-filming telepresence application with our in-house built virtual terminal system. We believe that the combination of the eContract concept and the SOA implementation is capable of facilitating a variety of dynamic collaborations in different application domains.

Our planned future work include (a) extending the eContract semantics and customizing its resources types; (b) enhancing the eContract service by building essential functional components, such as a monitoring service and policy management module. We also plan to deploy and offer the eContract service to our other existing collaboration partners through the use of existing platforms such as ViCCU in eHealth and Braccetto in eResearch.

Reference