Supporting e-commerce systems formalization with choreography languages

Mario Bravetti, Claudio Guidi, Roberto Lucchi, Gianluigi Zavattaro
Department of Computer Science, University of Bologna, Italy.
{bravetti,cguidi,lucchi,zavattar}@cs.unibo.it

ABSTRACT

E-commerce as well as B2B applications are essentially based on interactions between different people and organizations (e.g. industry, banks, customers) that usually exploit the Internet as communication media. Web Services provide a mean to deal with these aspects. In this paper we show, via a case study, how choreography and orchestration languages allow us to express behaviour policies between the involved entities (interactions modalities, interdependencies, security requirements); in particular we consider that they can be used not only for describing behavioural rules but also for designing and testing whether the involved entities move according with system specifications.

Categories and Subject Descriptors

J. [**Internet Applications**]: Middleware; D.3 [**Programming Languages**]: Miscellaneous; C.2 [**Communication Technology**]: Miscellaneous

Keywords

Web Services, choreography, orchestration.

1. INTRODUCTION

In this paper we want to focus the attention on Web Services as the technology which better matches e-commerce requirements and which most software designers will have to cope with in the following years. E-commerce applications and software in general will exploit more and more the Web as the natural choice for exchanging information both user-friendly and machine-friendly. Intranet and Internet applications will be soon indistinguishable and separated only by firewalls and access permissions. In this scenario e-commerce applications will play one of the most important role both governing world businesses among companies of different countries and also local cost-limited trades among people. Find a unique programming paradigm, which will unify both billion euros/dollars affairs and regional barters, is now a big challenge. Many are the aspects to take into consideration, in particular modularity and dynamic re-composition are needed in order to deal with both the uncontrolled explosion of the Web and the unpredictable demands of commercial issues. Security and reliability are necessary in order to guarantee safe business transactions and personal information secrecy. Interoperability is needed in order to achieve different participants interactions which have their own application implemented on different platforms with different technologies.

Not only technical requirements must be taken into account, but also social necessities will play a fundamental role. In particular, commerce, and e-commerce consequently, is essentially based on contracts between different people and organizations. In this sense let us imagine a – not so distant – future scenario where companies applications will interact re-designing themselves in an automatic way; how will be possible to prove behavioural errors and mistakes of commercial partners applications? What kind of documentation will have to be presented in order to demonstrate that a software will have not attended drawn up contracts? At the end, what kind of contracts will be able to govern software behaviours?

Considering all these aspects we are confident that Web Services will surely play a fundamental role. They were born from the simple concept of the remote procedure call wrapped by a standardized interface in order to realize interoperability. Loosely coupling, high level compositionality and service orientation are the most important features they are characterized by. Constructed on two basic specifications, WSDL[1] and SOAP[2], they are growing day by day with more specification layers which take into account different features. Recently, Web Services are one of the most discussed topic and, sometimes, probably over-discussed too. Nevertheless we believe that Web Services orchestration and Web Services choreography in particular can be the key to promote this technology as the de facto standard choice for e-commerce software and middleware designing in the next years.

One of the reason that allows us to state this is the fact that they are becoming the natural choice for programming Web Services composition, thus permitting to design new Web Service systems out of already available Web Services. Due to their nature indeed, Web Services are modular and interoperable and they can also support security and reliability if we consider other specifications as WS-Security[5], WS-SecureConversation[6], WS-Trust[3], WS-Policy[4] but it is still difficult to manage the complexity of a great number of Web Services. Orchestration engines go straithforward in this direction and some languages yet exist, the most famous is surely BPEL4WS[9]. They introduce invocation, concurrent and synchronization primitives for flowing information among different Web Services and carry out a main activity. But something is still lacking, the point of view of orchestration languages indeed, is always the orchestrator which is the center of the system and through which all the information pass. Often, e-commerce scenarios, due to their nature, require more than one orchestration engine running in parallel, thus increasing the complexity of the system. Therefore, a top view system description is necessary in order to fulfill and program correctly the different engines involved.

WS-CDL[10] which is a draft document of W3C, responds to these necessities introducing a description language which fixes the rules of the interactions between the parts involved in the system. We do not believe it is exhaustive but it has the potentiality on one side to aid the design of applications via a refinement process starting from the WS-CDL description, and on the other side to allow the verification of the compatibility of already available services willing to participate to a choreography described in WS-CDL. Web Services choreography indeed, is born as a sort of contract between the parts (which could be companies but single applications too) in order to rule their interactions. So, we can imagine that each part will design their own application and then verify its correctness exploiting the sentences of the WS-CDL document drawn up. In this sense, referring to what explained above, we could also imagine that WS-CDL can become a sort of actual valid contract which could be used to demonstrate other party mistakes and uncompliances. This last topic is not the center of our paper but it deserves to be investigated.

In this paper we discuss the validity of Web Services orchestration and choreography approach in order to deal with e-commerce issues highlighting differences and limits too. In order to achieve this goal we will exploit a case study where the involved participants are divided in buyers and vendors. We will study the case of buyer groups which collect their purchase intentions and supply them to an electronic auction where the vendors will compete to win the order bidding the lowest cost. It is a particular scenario where buyers sell something and vendors buy something. This paper wants to be the starting point for investigating choreographies and orchestration as the main candidates to deal with the designing of dynamic systems based on components. In this sense Web Services technology and e-commerce issues offer the straight workbench in order to develop formal models and executable tools. In particular we see in service oriented architectures and in choreography languages the key to govern the complex behaviour of the future Web applications.

In Section 2 we will present the differences between BPEL4WS and WS-CDL. In Section 3 we will present the case study, in Section 4 we will show a choreography proposal for the case study and in Section 5 the BPEL4WS implementation. In Section 6 conclusions and future works will be presented.

2. CHOREOGRAPHY AND ORCHESTRATION LANGUAGES

The terms choreography and orchestration are often considered as synonymous of service composition/coordination; in reality, there are essential differences as shown in [11]. Starting from the study of Web Services documentation, in particular WS-CDL[10] and BPEL4WS[9] specifications we will clarify their differences.

WS-CDL aims to constrict the behaviours of the Web Services involved in the system ruling the exchange of their messages instead of BPEL4WS that allows the design of a central entity which carries out an activity invoking other services. For the sake of clarity we will roughly present the two languages showing the most important features without giving an exhaustive description. We aim to highlight only the essential behaviours of choreography and orchestration in order to understand the role they could play for designing systems based on components.

In the following we list the features which we believe are essential in order to understand the main differences between orchestration (à la BPEL4WS) and choreography (à la WS-CDL):

- **Executable processes**: WS-CDL is a descriptive language and does not provide specifications for its execution. In this sense it is only a descriptive document without any direct computational purpose. On the contrary BPEL4WS assumes the existence of engines able to animate the specifications. A so called orchestrator engine executes BPEL4WS code and it is an essential part of the system which can invoke interactions and respond to requests. It must be noticed that BPEL4WS provides an abstract processes definition too. In this case it can be used for system description as WS-CDL even if, as it will be highlighted in the point below, *Interactions designing*, it is centered on a unique entity: the orchestrator engine.

- **Interactions designing**: WS-CDL provides a top view of the system focussed on the interactions between the participants. It is a definition of the rules which govern messages exchange among the parties involved in the choreographies. All the interactions have to be fulfilled between two entities and there is no notion of a central entity (e.g. a coordinator) which carries out the activity. On the contrary BPEL4WS is always centered on the orchestrator engine which drives all the interactions allowing services synchronization too. Indeed if there are two services which require to be synchronized, the former has to send the synchronization message to the orchestrator engine which will forward it to the latter.

- **Activity State**: In WS-CDL the state of the activity is distribuited among the entities. Indeed some internal variables of the entities involved are fundamental for the progression of the choreography. WS-CDL allows to express such variables as observable in order to change their value or explicitly align them after an interaction. After a message exchange indeed, the choreography may require that two variables which are located on the sender and on the receiver must be aligned. BPEL4WS, as said before, is centered on the orchestrator which is the entity which manages the communications and which stores all the state of the activity it is carrying out.

So far, we have highlighted the differences of the two languages but in the next sections we will present WS-CDL and BPEL4WS as complementary tools for system design. To do this we will exploit a case study presented in the section below.

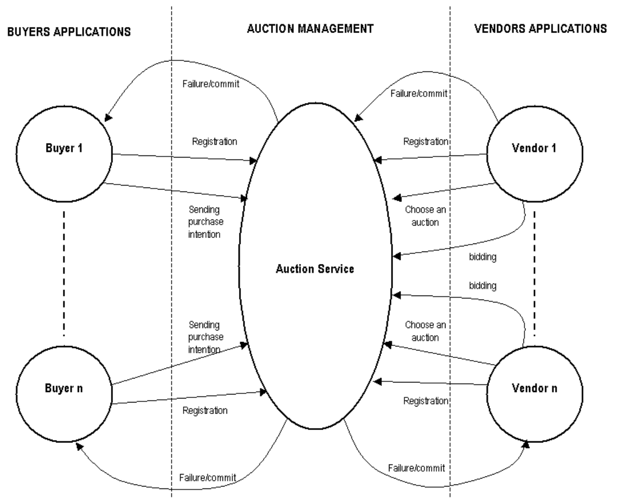

Figure 1: The case study

3. THE CASE STUDY

The case study we are going to explain deals with electronic auctions. It is a particular case where "the buyers sell something and the vendors buy something". At a first glance someone can argue that it is a strange way to make business but let us consider the case of a group of buyers that are intented to buy the same type of a car and that have signed the following agreement: *buying all or no-one*. In this case the car vendors have to compete for obtaining the order: there is a lot of difference between selling ten cars or selling nothing! So we can imagine a sort of auction where buyers present their *purchase intention* and the vendors bid the lowest price for obtaining it. In order to transfer this idea in a software system we see the design of three main parts: buyers application, auction management and vendors applications.

- **Buyers applications**: They can be seen as software interfaces through that human buyers can express their purchase intentions. Moreover they also can be thought as intelligent agents which formulate their purchase intentions following their internal rules. For the sake of simplicity we highlight three main tasks for buyers application: *registering the buyer on the Auction Service*, *sending a purchase intention to the Auction Service*, *waiting for a commit or a failure from the Auction Service*. In Fig. 1 an arrow from a buyer application to the Auction Service represents the request for a registration. Obviously this is a graphical simplification which does not show all the messages exchange that a registration request implies. In this sense arrows must be understood as main task representations which group all the information flow between two entities.

- **Auction management**: This part of the system deals with the management of the auction. It stores the registration data of both the buyers and the vendors. It collects the purchase intentions of the buyers and groups them according to the goods to buy then starts an auction for each group of intentions received. The vendors can compete for obtaining the order sending to the Auction Service their bids. When the auctions finish the Auction Service will send commit or failure messages both to the vendors and to the buyers. Summarizing, the Auction Service have to *manage the buyers registration data*, *manage the vendors registration data*, *collect and group the intentions*, *managing the auctions*.

- **Vendors applications**: As for the buyers they can be imagined as both simple interface between human users and the Auction Service and intelligent agents which automatically choose an auction and formulate their bids according to their goals and requirements. We highlight four main tasks: *registering the vendor on the Auction Service*, *choosing an auction*, *bidding* and *waiting for a commit or a failure message from the Auction Service*.

In Fig. 1 we have depicted the three services and their interrelationships.

4. THE CHOREOGRAPHY OF THE CASE STUDY

In Fig. 2 it is presented the WS-CDL choreography of the case study. For the sake of brevity some tags and attributes are omitted besides other elements that are not expanded and detailed. Our aim is to briefly show how a WS-CDL document is highlighting some important peculiarities. Inside the root tag `<package>` there are a declarative part followed by the main choreography `<choreography name=''GeneralChoreo''>`. The former declares the roles of the entities involved in the choreography and their relationships whereas the latter expresses the variables used inside the choreography and the activities to fulfill. Choreography tags can be nested for obtaining a structured choreography. Each choreography contains the variables, which are visible to the enclosed choreographies too, and the activities to be performed. Variables and activities deserve to be briefly discussed:

- Variables can be *information exchange variables* which means that they will be parts of the messages exchanged by the entities and *State Variables* which express the observable state of the entities. In this sense the choreography global state is the set of all the state variables distribuited among the entities. Here we mention also the silent-action attribute of the tag `<variable>`; this means that it is not directly controlled by the activities of the choreography. In other words they are assumed to be the direct consequence of the internal actions of the entities where the variable is located.

- Concerning the activities, here we cite the `<interaction>` activity which implies the message exchange between the participants, the `<assign>` activity which allows to copy the value of a variable to another one and the `<perform>` activity which allows to perform an enclosed choreography. Furthermore, the activities can be both structured exploiting `<sequence>`, `<parallel>`, and `<choice>` tags whose meaning is easy to understand and constrained inside a `<workunit>` which allows to perform the activities depending on a guard condition.

Now we analyze the activity of the `GeneralChoreo` choreography which describes the behaviour of the system. Looking at the code of Fig. 2 there are six main activities included in a `<parallel>` tag; thus they are performed concurrently. In the following we briefly describe them:

- `<perform choreographyName=''BuyerRegChoreo''>`: this is an enclosed choreography which deals with the registration of the buyers to the Auction Service. The tag is not expanded but it is assumed to contain all the interactions between the role of the buyer and the role of the Auction Service in order to achieve the registration.

- `<perform choreographyName=''VendorRegChoreo''>`: this is an enclosed choreography which deals with the registration of the vendors to the Auction Service. Also in this case the tag is not expanded and it has to be assumed that it contains all the interaction between the role of the vendor and the role of the Auction Service in order to achieve the vendor registration.

- `<workunit name=''StartAuction''>`: this workunit is responsible for starting an auction. The guard verifies if the variable `AuctionCreate` is available, if this is the case it means that a group of Purchase Intentions is available for an auction. It has to be noticed that the `AuctionCreate` variable is defined with the `silent-action` attribute set to `true` and represents a state of the Auction Service. Verify the availability of a silent-action variable means that it becomes observable. Referring to the code, if the guard is verified the contents of the workunit `StartAuction`, which is the choreography `AuctionChoreo`, must be performed. The `AuctionChoreo` is specified inside the `GeneralChoreo` choreography and deals with the interactions required to govern the vendor offers and the auction termination. In particular the workunit

BiddingManagement, which is not expanded, deals with the bidding interactions between the Vendor and the Auction Service whereas the workunit AuctionTerminate deals with the termination of the auction. These two workunits are located inside a <choice> tag which means that only the workunit matching the guard condition can react. Furthermore, they are inside the workunit AuctionManagement which is repeated until the variable Finished is equal to false. The execution of the workunit AuctionTerminate can change the value of the variable Finished which implies the skipping of the next performation of the workunit AuctionManagement. At this point the auction is terminated and the workunit AuctionFinished can be performed. It is assumed to contain all the interactions required to signal the termination to the buyer and the vendor.

- <workunit name=''SendAuctionList''>: this element is a workunit which manages the vendor request to obtain the list of the active auctions in order to partecipate to one of them. For the sake of simplicity the element is roughly expanded. Here we show only the fact that it contains an interaction between the AuctionService role and the Vendor role.

  In particular the workunit should be guarded by a state variable of the Vendor not explicited in Fig. 2 which express the fact that the vendor wants to perform the request. If the guard is verified the interaction inside the workunit can be performed.

- <workunit name=''SendPurchaseIntention''>: this workunit contains an interaction between the Buyer role and the AuctionService role. As for the tags above, it is not expanded in a detailed way but it is assumed to contain the messages exchange between the buyer and the Auction Service where the former sends its Purchase Intention to the latter. Also this workunit should be guarded by a state variable (not explicited in Fig. 2) of the Buyer which express the fact that the buyer wants to send a Purchase Intention.

Summarizing five main activities must be performed concurrently: *The registration of the buyers*, *the registration of the vendors*, *the creation and the management of an auction*, *the sending of the active auction list* and *the sending of the Purchase Intentions*. Obviously we have presented a simplified version of the choreography which does not take into account other important interactions but the goal of this paper is not to give a detailed WS-CDL documentation. Indeed, we aim to show how WS-CDL and choreography languages in general should be used as a first step of e-commerce systems design. In the next section a BPEL4WS implementation of the case study will be presented.

5. FROM THE CHOREOGRAPHY TO THE ORCHESTRATION

The choreography shown in Fig. 2 gives a top view of the system from which many implementations can be derived. In particular here we make the implementation choice to split the main entity Auction Service into two different entities: the Purchase Intention Service (PIS) and the Auction Management Service (AMS). Due to space limitation we simply sketch the orchestration we consider as an implementation of the choreography of Fig. 2. The full description can be found in [14].

The Purchase Intention Service deals with the collection and the grouping of the Purchase Intentions from the buyers whereas the

```
<package>
  <role name="Buyer"/>
  <role name="AuctionService"/>
  <role name="Vendor"/>
  <relationship name="Buyer_AS">
     <role type="Buyer"/>
     <role type="AuctionService"/>
  </relationship>
  <relationship name="AS_Vendor">
     <role type="AuctionService"/>
     <role type="Vendor"/>
  </relationship>
  <channelType/>
  <choreography name="GeneralChoreo">
     <relationship type="AS_Vendor">
     <relationship type="Buyer_AS">
     <variableDefinitions>
        <variable name="AuctionCreate"
                            silent-action="true"
          role="AuctionService"/>
     <variableDefinitions/>
     <choreography name="BuyerRegChoreo"/>
     <choreography name="VendorRegChoreo"/>

     <!-- AuctionChoreo choreography -->
     <choreography name="AuctionChoreo">
        <relationship type="AS_Vendor">
        <variable name="Finished"
                   mutable="false"/>
        <variable name="Terminate"
               silent-action="true">
        <assign>
           <!-- Finished="false" -->
        </assign>
        <workunit name="AuctionManagement"
            guard="cdl:getVariable("Finished",
            "AuctionService")=false" repeat="true">
           <choice>
             <workunit name="BiddingManagement"/>
             <workunit name="AuctionTerminate"
                guard="cdl:getVariable("Terminate",
                "AuctionService")" block="true">
              <assign>
              <!-- Finished="true" -->
              </assign>
             </workunit>
            </choice>
          </workunit>
        <workunit name="AuctionFinished"/>
     </choreography>

     <!-- GeneralChoreo activities -->
     <parallel>
        <perform choreographyName="BuyerRegChoreo"/>
        <perform choreographyName="VendorRegChoreo"/>
        <workunit name="StartAuction"
            guard="cdl:getVariable("AuctionCreate",
            "AuctionService")" block="true" re-
peat="true">
             <perform choreographyName="AuctionChoreo"/>
         </workunit>
         <workunit name="SendAuctionList">
             <interaction name="SendAuctionListInteraction">
                <partecipate relationship="AS_Vendor"/>
             </interaction>
         </workunit>
 <workunit name="SendPurchaseIntention">
     <interaction name="SendPurchaseIntentionInteraction">
                 <partecipate relationship="Buyer_AS"/>
             </interaction>
         </workunit>
      </parallel>
   </choreography>
</package>
```

Figure 2: The WS-CDL choreography of the case study

Auction Management Service deals with the creation of the auctions and the management of the interaction with the vendors. We assumed that the PIS will send a request to the AMS for an auction when it has a collection of Purchase Intentions and after an unspecified amount of time sends to the AMS an auction termination request too. Vendors and Buyers play the same tasks described in section 2. At this point four different kinds of orchestrator can be designed following the interactions constraints explicited into the main choreography: the Buyer orchestrator, the Vendor orchestrator, the Purchase Intention Service Orchestrator (PISO) and the Auction Management Service Orchestrator (AMSO).

Now, let us consider choreography and orchestration languages as complementary frameworks for designing e-commerce systems.

WS-CDL provides a system top view which allows both to fix the interaction constraints among the entities and to manage the main activity state which, in the actual implementation, is distributed on local state variables. For this reason it could be seen as the first system design step which could be followed by the implementation phase where BPEL4WS can be exploited. Referring to the case study, here we briefly discuss how a top view variable `AuctionCreate` of the `GeneralChoreo` choreography is actually implemented in a distributed fashion, as its value should reside on both the PISO and the AMSO orchestrators.

At the level of choreography, the `AuctionCreate` variable is related to an internal state of the Auction Service and when it becomes observable an auction must be performed. The implementation we have chosen with two orchestrators (PISO and AMSO) implies the splitting of that variable. Indeed, even if the Purchase Intention Service Orchestrator is not described here, we assume it uses an internal variable, which we name now `ActivePISO`, that stores the corresponding boolean value of the `Active` variable we have used in the description of the AMSO orchestrator.This variable is exploited in order to be able to denote that an auction is still running (the auction terminates when the value associated to the variable is changed from true to false). The `ActivePISO` and the `Active` variables must be aligned in order to be consistent with the `AuctionCreate` choreography variable. To guarantee the alignment, an aknowledgement reply is sent to the PISO from the AMSO after that the auction creation request is received. Obviously. this a very simple alignment protocol but other classical protocols (e.g. a two phase commit protocol) could be used instead.

6. CONCLUSIONS AND FUTURE WORKS

In this paper we have considered choreography and orchestration languages (namely, WS-CDL[10] and BPEL4WS[9]) and we have discussed their exploitation within a new approach for the development of e-commerce applications based on Web Services. Typically, choreography and orchestration are considered as two alternative approches for supporting service composition: the former based on an orchestration engine responsible for gluing together the collaborating services, the latter based on a direct communication among the services themselves. Our approach, on the other hand, proposes to use choreography languages as high-level description languages, while the orchestration languages are exploited for programming the actual engines needed for implementing the collaboration described at the level of choreography. We have described this approach via an auction service use case. The advantage of this approach is twofold: on the one hand, it permits to achieve Web Services composition in a top-down manner; on the other hand, it could support the verification of the conformance of already available service to the constraint described at the choreography level.

As a future work we intend to develop a formal framework for proving the conformance of orchestration services with respect to the roles described at the choreography level. We intend to exploit this framework also for proving the compatibility between two different orchestrations, one obtained as a refinement of the other one. In this way, we have not only one phase of refinement from choreography directly to orchestration, but we have also several choreographies at different levels of abstraction obtained via a stepwise refinement.

Clearly, this framework requires also the formal definition of the semantics of the BPEL4WS and WS-CDL languages. In this sense some works exist, in particular we cite here [12] and [13] which are proposals of formalization for BPEL4WS.

7. REFERENCES

[1] Francisco Curbera et al. - Web Service Description Language (WSDL) Version 1.1. W3C Note 15 March 2001 [http://www.w3.org/TR/wsdl].

[2] Martin Gudgin et al. - SOAP Specifications ver. 1.2. W3C Recommendation 24 June, 2003[http://www.w3.org/TR/soap/].

[3] Steve Anderson et al. - Web Services Trust Language (WS-Trust). [http://www-106.ibm.com/developerworks/library/specification/ws-trust/].

[4] Siddharth Bajaj et al. - Web Services Policy Framework (WS-Policy) [http://www-106.ibm.com/developerworks/library/specification/ws-polfram/].

[5] Bob Atkinson et al. - Web Services Security (WS-Security). [http://www-106.ibm.com/developerworks/webservices/library/ws-secure/].

[6] Steve Anderson et al. - Web Services Security Conversation (WS-SecureConversation). [http://www-106.ibm.com/developerworks/library/specification/ws-secon/].

[7] Luis Felipe Cabrera et al. - Web Services Coordination [http://www-106.ibm.com/developerworks/library/ws-coor/].

[8] Felipe Cabrera et al. - Web Services Transactions [http://www-106.ibm.com/developerworks/webservices/library/ws-transpec/].

[9] Tony Andrews et al. - Business Process Execution Language for Web Services Version 1.1. [http://www-106.ibm.com/developerworks/library/ws-bpel/].

[10] N. Kavantzas, d. Burdett, G. Ritzinger - Web Services Choreography Description Language Version 1.0. Working draft 27 April 2004, [http://www.w3.org/TR/2004/WD-ws-cdl-10-20040427/].

[11] C. Peltz - Web Services Orchestration and Choreography. Web Services Journal, [http://www.sys-con.com/webservices]

[12] Manuel Mazzara, Roberto Lucchi - A Framework for Generic Error Handling in Business Processes. In *Proc. of 1st International Workshop on Web Services and Formal Methods (WS-FM'04)*, Mario Bravetti and Gianluigi Zavattaro, editors, *Electronic Notes in Theoretical Computer Science*, Elsevier.

[13] Mirko Viroli - Towards a Formal Foundation to Orchestration Languages. In *Proc. of 1st International Workshop on Web Services and Formal Methods (WS-FM'04)*, Mario Bravetti and Gianluigi Zavattaro, editors, *Electronic Notes in Theoretical Computer Science*, Elsevier.

[14] M.Bravetti, C.Guidi, R.Lucchi,G.Zavattaro - Supporting e-commerce systems formalization with choreography languages. Full version available at [http://www.cs.unibo.it/people/phd-students/cguidi/Publications/2004-17.pdf]