Augmenting Digital Documents with Negotiation Capability

Jerzy Kaczorek
Dept. of Intelligent Interactive Systems
Faculty of ETI
Gdansk University of Technology, Poland
jkaczorek@gmail.com

Bogdan Wiszniewski
Dept. of Intelligent Interactive Systems
Faculty of ETI
Gdansk University of Technology, Poland
bowisz@eti.pg.gda.pl

ABSTRACT

Active digital documents are not only capable of performing various operations using their internal functionality and external services, accessible in the environment in which they operate, but can also migrate on their own over a network of mobile devices that provide dynamically changing execution contexts. They may imply conflicts between preferences of the active document and the device the former wishes to execute on. In the paper we propose a solution for solving such conflicts with automatic negotiations, allowing documents and devices to find contracts satisfying both sides. It is based on a simple bargaining model reinforced with machine learning mechanisms to classify string sequences representing negotiation histories.

Categories and Subject Descriptors

I.7.1 [**Document and Text Processing**]: Document and Text Editing—*Document management*; I.2.11 [**Artificial Intelligence**]: Distributed Artificial Intelligence—*intelligent agents*; H.5.3 [**Group and Organization Interfaces**]: collaborative computing—*computer-supported cooperative work*

Keywords

Active document, automatic bargaining, mobile computing

1. INTRODUCTION

Mobile Interactive Document (MIND) architecture developed in the MENAID project [4] enables proactive digital documents to travel through an open network of geographically separated locations, carry any useful content conforming to the MIME standard, and provide services enabling interaction with collaborators, their respective local devices and third-party external services [1]. Owing to the notion of policies incorporated in their logical structure, MIND documents are *mobile* and *intelligent*. The former involves a document workflow combining *activities* and *transitions*, while the latter implies ability to resolve conflicts between preferences of the active document and the characteristics of the device where it is executed. Activities are performed by collaborators interacting with document-agents carrying content to their personal devices, and involve such operations as text editing, merging, splitting, copying, form filling, and so on. Transitions from one collaborator to another are performed automatically by documents between their respective locations. While locations of collaborators are specified firmly with their email addresses, devices they use when performing activities, may change significantly – from a powerful workstation with a trusted company network connection, to a laptop accessing a public (open) WiFi network in a hotel room, to a smartphone or cellphone on a plane in flight, thus without any network access at all.

Clearly, an active document must be able to adjust to such dynamic execution contexts, often in a non-cooperative setting, as the execution devices (or rather their owners) may impose their own policies governing the way document workflow activities may be executed. Conflicts arising between active documents and execution devices must be resolved in a way that satisfies both parties, for otherwise a workflow process could not be completed. Therefore we propose to introduce negotiation as a general technique for adapting active documents to varying client system requirements. By "adapting" we mean reaching agreement on a certain set of attributes of a service the document receives from the device. Negotiation is necessary to cope with an unlimited number of attributes and their combinations, which may occur in various proactive document systems – depending on their particular semantics, classes of problems solved and types of execution devices used. Further in the paper we introduce the use of this technique for the MIND architecture as an example, using a set of service attributes used in our current implementation of the system.

2. NEGOTIATION PROCESS

We model offers with trees, which are based on the logical structure of *Collaboration Protocol Profile* (CPP) documents used by ebXML to declare preferences of collaborating partners [5]. Each path in such a tree constitutes an offer consisting of five items, representing attributes of possible execution contexts of each activity: who shall be its actual *performer*, and what are the current network *availability*, its *performance* characteristics, the current execution device *security* and its possible *reliability* levels.

Values of each component item are of two types: one is a *public* content of each possible offer, and another is *private*

http://dx.doi.org/10.1145/2494266.2494305

preference, used by each respective partner to rank offers. Therefore trees of each partner are internally sorted from the most to the least valued offer. Figures 1 and 2 indicate conflicting preferences of the execution device: a laptop owner preferring not to be using a company network, and an active document willing to do everything on its own, most preferably from inside a company network.

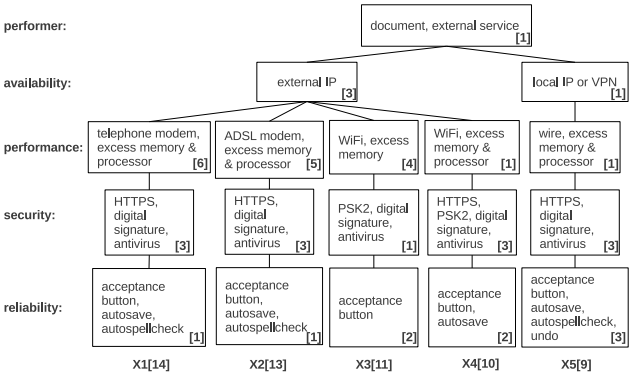

Figure 1: Preferences of an execution device

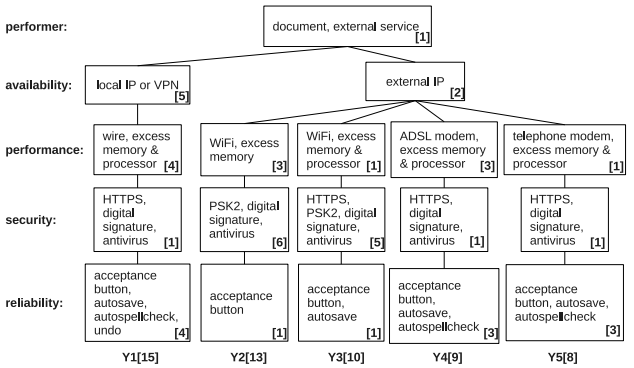

Figure 2: Preferences of a proactive document

2.1 Bargaining model

Negotiation of values of a multi-item contract, as specified by trees above, is a non-zero game, where each side can win some wealth. The question is how to organize the entire process, so that each party may get as much as it can in the current execution context. We have proposed for that a simple bargaining (economic) model [2] with a utility function valuating offers as a sum of element preferences along each respective CPP tree path. Offers are chosen by each partner from its tree, starting from the leftmost (most valuable) one. Bids are exchanged until one negotiating party repeats any offer presented by its opponent before.

Bids with trees in Figures 1 and 2 would be the following: $1 : Y_1[15] \Rightarrow X_5[9]$, $2 : X_1[14] \Rightarrow Y_5[8]$, $3 : Y_2[13] \Rightarrow X_3[11]$, $4 : X_2[13] \Rightarrow Y_4[9]$, $5 : Y_3[10] \Rightarrow X_4[10]$, $6 : X_3[11] \Rightarrow Y_2[13]$, where labels X and Y denote respectively the negotiating parties, an execution device and the document, indexes of the initiating party 1, 2, 3 denote consecutive rounds *(offer,counteroffer)*, brackets provide values of exchanged bids for each side, and $\Rightarrow$ points alternative valuation of the offer by the receiving opponent. Initial bid Y_1 was made by the document, while X_3 made by the execution device is the final one, since it repeats Y_2 offered before by the document. A contract is specified in Figure 3 with a tree similar to a logical structure of the *Collaboration Protocol Agreement* document used by ebXML [5].

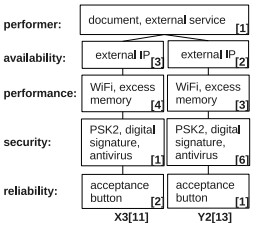

Figure 3: Negotiated collaboration agreement

CPP trees considered in the above example are relatively simple. In a more realistic scenario, however, execution contexts of active documents may be much more diversified – given the variety of personal devices used by collaborators, their preferences, as well as the abundance of workflow cases and classes of active documents that may be designed to resolve them. To avoid combinatorial explosion leading to the excessive number of rounds during the negotiation process more rigid classification of active documents and their execution contexts is necessary.

2.2 Negotiation policies

We started our analysis from the observation that not all offers that can be described with CPP trees make sense (or are worth considering) in each particular execution context. This has lead us to the concept of negotiation policies. The idea is that upon arrival an active document is aware of what kind of device it will be provided with, so what offers will not be considered for sure, and what is their commonly agreed *partial ordering*. Collaborators reserve the right, however, to keep detailed ordering of their CPP trees private, as they wish to satisfy the incoming document at the lowest possible cost. In consequence, bargaining can be performed in a *semi-cooperative* setting, where parties are free to make offers, but some *a priori* agreements concerning the execution circumstances apply. These "circumstances" are described with respective sets of rules for designing CPP trees for each party, which constitute in fact specific negotiation policies.

Before going to a detailed example consider a coding scheme of offer components defined in Table 1. All five previously mentioned execution aspects are classified and coded with symbols, used later in expressions defining the respective policy rules. For example, D denotes a class of active documents that can perform a given activity automatically, e.g., fetch data from an indicated user's repository, as opposed to a document of class W, which requires a user (worker) to do the job, e.g. read and approve its content. The reliability aspect (in the lowest row of Table 1) has just four simple classes, shown already in Figures 1 and 2, i.e. class H includes all such features as acceptance and undo buttons, autosave and auto spell-check, while the remaining symbols L, B and F represent subsets including only some of them.

2.2.1 Execution device context

Execution devices may be one of the following: workstations (WS), laptops (LT), tablets (TB), smartphones (SM), and cellphones (CP). Their technical features vary, what may significantly affect the way a given activity is per-

Table 1: Negotiated items

execution aspect	offer components
performer	document (D), worker (W), both (J)
availability	separated (S), connected from outside (E) or inside (I) organization
performance	network unknown (U), WiFi (R), phone (M) or ADSL (A) modem, wire (N)
security	public (P) or private (K) network, HTTPS (T), secure connection (C)
reliability	low (L), back-up (B), no fail (F), high (H)

formed. For example, workstations and laptops do not differ very much in their computational power, but the latter are mobile, thus more often using alternative network connections, like WiFi, ADSL or telephone modems. Tablets or smartphones in turn do not have regular keyboards, what may limit interaction between active documents and their users, while cellphones add to a limited keyboard also a low speed and costly network connection. Each device of the classes listed above may perform the activity when connected to the network or disconnected. This also determines what an active document could and could not do while on the execution device. Based on that we distinguish ten execution contexts:

$$\{WS, LT, TB, SM, CP\} \times \{connected, disconnected\} \quad (1)$$

In Table 2 we specify rules for building context specific sets of offers as CPP trees for two example execution contexts, a laptop connected to any network, and a smartphone out of reach of any network. The CPP tree root is denoted by ϵ, possible connections between tree levels with $\rightarrow$, and elements at respective levels with symbols from Table 1. Moreover, elements in curly brackets $\{\}$ may be specified in a tree in any order.

Table 2: Example device policy rules

LT connected	SM disconnected
$\epsilon \rightarrow D$	$\epsilon \rightarrow D$
$D \rightarrow \{E, I\}$,	$D \rightarrow S$
$E \rightarrow \{R, M, A, N\}$, $I \rightarrow \{R, N\}$	$S \rightarrow U$
$\{M, A, N\} \rightarrow \{P, T\}$, $R \rightarrow \{P, K, T, C\}$	$U \rightarrow P$
$\{P, K, T, C\} \rightarrow \{L, B, F, H\}$	$P \rightarrow \{L, B, F, H\}$

2.2.2 *Document classes*

MIND documents split in three classes, depending on what kind of functionality they bring to the execution device and what level of autonomy a document may get from the current activity performer. Passive (PS) documents make their content accessible for processing with user tools, installed locally on the execution device. Reactive (RA) documents are more autonomous in keeping under control what users can do with their content, but expect users to initiate certain actions. Finally, proactive (PA) documents initiate all necessary actions, call local or external services and take over the interaction with the user. Depending on the actual security policy they may consider their content *protected* or *open*, and the strain imposed on the execution device when executing the activity as *heavy* or *light*. We get 12 classes of documents, each with its own set of policy rules:

$$\{PS, RA, PA\} \times \{protected, open\} \times \{heavy, light\} \quad (2)$$

Table 3 specifies rules for building CPP trees for proactive documents with a protected content and light stress on the execution device in two alternative execution modes: with and without network access.

Table 3: Example document policy rules

network required	network not required
$\epsilon \rightarrow D$	$\epsilon \rightarrow D$
$D \rightarrow \{E, I\}$,	$D \rightarrow S$
$E \rightarrow \{R, M, A, N\}$, $I \rightarrow \{R, N\}$	$S \rightarrow U$
$\{M, A, N\} \rightarrow T$, $R \rightarrow \{K, C\}$	$U \rightarrow P$
$\{K, T, C\} \rightarrow \{B, H\}$	$P \rightarrow \{B, H\}$

2.3 Reproducibility of contracts

The notion of a semi-cooperative setting based on negotiation policies leads to the significant reduction of the size of CPP trees needed by active documents to bargain over contracts with execution devices. However, it does not provide any mechanism that may help active documents to pass one another any knowledge on what contracts could be won with specific devices in various contexts. A naive approach would be when each document stores its negotiation histories with each encountered device and shares it with other companion documents. With that, reproducibility of contracts could be provided, i.e., instead of negotiating contracts all over again a document could repeat the one known from its past encounters, or encounters of its companions. The problem is, however, that the negotiation histories might be to large to be stored in a mobile document agent memory. We propose to solve this problem by introducing a machine, or rather document, learning approach, used in sequence classification [6].

3. DOCUMENT LEARNING APPROACH

The example contract specified with the tree in Figure 3 has been negotiated by an active document and its execution device in three rounds. Obviously, trees specified in Figures 1 and 2 would have much more paths if execution aspects listed in Table 1 were specified in more detail. This is the case of our prototype implementation of MIND documents, where some CPP trees may reach the size of 80000 or more paths, leading to excessively long negotiation histories. The question to be asked here is whether it is possible to represent negotiation histories as sequences of symbols and train active documents to classify n-symbol sequences by recognizing k-grams, short sequences of up to k consecutive symbols ($k << n$). If trained, a document may be able to guess a contract during the next encounter with the execution device, as well as pass this ability to other documents, in case they may encounter this device context in the future. Because of a semi-cooperative setting – in this case a commonly agreed assumption that offers in CPPs are always sorted in the decreasing order, documents may be trained with sequences containing offers communicated only by the devices.

3.1 Coding of offers

A more detailed analysis of execution contexts indicated by Formula 1 yields a certain number of subclasses of each negotiated item listed before in Table 1. Subclasses of negotiated items are identified with unique labels, used in *context dictionaries* to specify what subclasses may appear at each level of a CPP tree in each particular execution context. For example, context dictionary of a laptop described in Table 2 is shown in Table 4; it details a CPP tree in Figure 1. For example, D_1, D_2 and D_3 denote subclasses of contexts enabling documents to perform their activities solely with the embedded functionality, or with local tools provided by the

device, or with services external to both, E_5, E_6 and E_7 denote various types of resources available to documents, and so on. Detailed description of each subclass is beyond the scope of this short paper – we list them just to indicate the size of the space of possible negotiation histories.

Table 4: An example context dictionary

item	item subclass
performer	$D1$, $D2$, $D3$
availability	$E5$, $E6$, $E7$, $I5$, $I6$, $I7$
performance	$A4$, $M4$, $N4$, $R1$, $R2$, $R3$, $R4$
security	$C4$, $K2$, $K3$, $K4$, $T2$, $T3$, $T4$
reliability	$H3$, $H4$, $F1$, $F2$, $F3$, $F4$

3.2 Training sets

Negotiation histories $h_m = \sigma_{m_1}\sigma_{m_2}...\sigma_{m_n}$, $m = 1, 2, .., Q$ are sequences of length $|h_m|$, with offers from set $HS = \{\sigma_1, \sigma_2, ..., \sigma_q\}$. Each offer in HS uses symbols from the related context dictionary, q is the maximum number of offers in one CPP tree, and Q is the number of all paths in all CPP trees, possible in a given context. Special coding function $PREC : HS \times HS \rightarrow \{0, 1\}$, such that $PREC(\sigma_i, \sigma_j) = \{0 : i > j,\ 1 : i < j\}$, defines conversion of a set of negotiation histories into a training set:

$$LE = (map\ PREC) \circ (filter\ i \neq j) \tag{3}$$

Negotiation history h for the contract in Figure 3 consists of $|h| = 3$ offers: $\sigma_1 = D_3E_7M_4T_4H_4$, $\sigma_2 = D_3E_7A_4T_4H_4$, $\sigma_3 = D_3E_7R_2K_4F_1$. Note that the upper bound for the maximum size of a single CPP tree with symbols from a given context dictionary is a product of the numbers of symbols provided for each item, i.e. $||HS|| \leq 5292$, while the upper bound for the number of possible negotiation histories is a product of their respective permutations, i.e. $m \leq 79{\cdot}10^{12}$. In order to convert a set of negotiation histories into a training set each h_m is coded as sequence $seq(h_m)$ of unique natural numbers. For h considered before we get $seq(h) = 3, 2, 1$, and generate <(4,3,1),(4,2,1),(3,2,1),(3,4,0),(2,4,0),(2,3,0)> with LE. Such vectors of pairs of offers derived from negotiation histories is put to a neural network along with the negotiated contracts to train it. Note that the training set describes relative orderings of each possible pair of offers in recorded histories, rather then define ordering of entire sequences. A trained network can later guess contracts based on the initial k offers returned by the execution device.

3.3 Testing

Document learning based on the presented model of negotiation histories has been implemented with a neural network simulated in MATLAB. It has ten neurons in one hidden layer and two neurons in the output layer. Our initial test results are promising – after training the network with a vector derived with LE from a set of $Q = 12$ k-grams (with $k = 4$) of sequences related to the context specified in Table 4, it was able to precisely guess over 50% of contracts, and further 40% close to the optimal ones.

It was possible to improve the above results by retraining the network, but a better solution was increasing the number of histories to have more pairs of offers in the training set.

4. CONCLUSIONS

A question may be asked what happens when an undertrained active document makes an imprecise guess of the contract during its encounter with an execution device. For the bargaining model used in our approach it does not pose any problem, as an imprecisely guessed contract may just delay reaching an agreement – it will be rejected by the execution device and negotiations continued for a few more rounds. To speed that up and make parties more willing to compromise a discount factor $\delta \in (0, 1)$ may be used to diminish valuation of the next round offer compared to the presently considered one [2].

When adapting proactive documents to their execution contexts a technique based on the media queries may be considered [3], as an alternative to negotiation and machine learning proposed here. However, the problem with media queries is that their sets of *media types* and *media features* have to be defined beforehand, as a commonly agreed standard to all documents and devices. In real life, proactive documents may not know the type of a device they would execute on, so queries might not resolve to true. Negotiation can cope with that, as only the set of service attributes in a negotiated contract has to be agreed beforehand, regardless of the actual classes of available devices. Classes of these devices may be implicitly learned by documents with each successfully negotiated contract.

Our plans for future work include experiments to determine the optimal size of training sets and k-grams for contract prediction – using neural networks and the naive Bayes classifier first, and next adopting reinforcement learning mechanisms to enable active documents to discover new execution contexts in a truly open environment.

5. ACKNOWLEDGMENTS

This work was supported by the National Science Center grant no. DEC1-2011/01/B/ST6/06500.

6. REFERENCES

[1] M. Godlewska and B. Wiszniewski. Distributed MIND - a new processing model based on mobile interactive documents. In *Proc. PPAM 2009*, volume 6068 of *LNCS*, pages 244–249. Springer, 2010.

[2] J. Kaczorek and B. Wiszniewski. A simple model for automated negotiations over collaboration agreements in ebXML. In *Proc. 13th Conf. on Commerce and Enterprise Computing (CEC)*, pages 167–172. IEEE, 2011.

[3] Media Queries. W3C Recommendation. `http://www.w3.org/TR/2012/REC-css3-mediaqueries-20120619/`, June 2012.

[4] MeNaID project home page. Methods and tools of next generation document engineering. `http://www.menaid.org.pl`.

[5] OASIS. *Collaboration Protocol Profile and Agreement Specification Version 2.0*, September 2002.

[6] Z. Xing, J. Pei, and E. J. Keogh. A brief survey on sequence classification. *SIGKDD Explorations*, 12(1):40–48, 2010.