2013 Senior Thesis Project Reports

Iliano Cervesato* **Kemal Oflazer*** **Mark Stehlik***
Thierry Sans* **Soha Hussein*** **Behrang Mohit***
Khaled Harras* **Abderrahmen Mtibaa***

May 2013
CMU-CS-QTR-119

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

*Qatar campus.

The editors of this report include the members of the Senior Thesis Committee on the Qatar campus and the students' advisors.

Abstract

This technical report collects the final reports of the undergraduate Computer Science majors from the Qatar Campus of Carnegie Mellon University who elected to complete a senior research thesis in the academic year 2012–13 as part of their degree. These projects have spanned the students' entire senior year, during which they have worked closely with their faculty advisors to plan and carry out their projects. This work counts as 18 units of academic credit each semester. In addition to doing the research, the students presented a brief midterm progress report each semester, presented a public poster session in December, presented an oral summary in the year-end campus-wide Meeting of the Minds and submitted a written thesis in May.

Keywords: Web Security, Programming Languages, Authorization, Natural Language Processing, Machine Translation, Word Segmentation, Mobile Computation, Offloading, Mobile Cloud.

Contents

Lulwa Ahmed El-Matbouly
An Authorization Model For The Web Programming Language Qwel 1
Advisors: Thierry Sans and Soha Hussein

Hanan Alshikhabobakr
Unsupervised Arabic Word Segmentation and Statistical Machine Translation 14
Advisors: Kemal Oflazer and Behrang Mohit

Afnan Fahim
Computational Offloading in Mobile Device Clouds 27
Advisors: Khaled Harras and Abderrahmen Mtibaa

An Authorization Model For The Web Programming Language Qwel

Lulwa Ahmed El-Matbouly

lulwa@cmu.edu

Advisors:

Thierry Sans tsans@qatar.cmu.edu

Soha Hussein sohah@qatar.cmu.edu

CARNEGIE MELLON UNIVERSITY

SCHOOL OF COMPUTER SCIENCE

May 3, 2013

Abstract

With the fast growth of web technology, it is becoming easier for developers to design and deploy complex web applications. However, securing such web applications is becoming an increasing complex task as current technology provides limited support. Developers are required to reason about distributed computation and to write code using heterogeneous languages, often not originally designed with distributed computing in mind nor built-in security features.

Qwel is an experimental type-safe functional programming language for the web that has dedicated primitives for publishing and invoking web services. In this paper, we propose to extend Qwel with a decentralized authorization model allowing service providers to secure web applications written in Qwel. This extension provides web developers with built-in primitives to issue credentials to users and to express access control policies. Therefore, when a protected web service is deployed, the security policy is evaluated dynamically based on the credentials supplied by the user invoking this web service. As a result, we show how these new language features can be used to implement common scenarios as well as more sophisticated ones.

1 Introduction

With the fast growth of web technology and cloud computing, it is becoming increasingly popular to move software and data to the cloud. In this paradigm, the software is no longer a standalone application installed on the user's computer but it is offered as a web application. For instance Google Docs is an office suite (word processor, spreadsheet and presentation) that can be used through a web browser. From the user perspective, Google Docs is not very different from standalone office suites like Microsoft Office or Open Office. However, from the developer perspective, building web applications is a significant shift in the way to design, implement and deploy software. Indeed, correctness and security have always been the main concerns but it takes a new dimension in the context of web applications. For instance, a bug or a crash that occurs in a standalone application may impact the platform's owner only. However, a bug or a crash that occurs on the server side of a web application may impact all users registered to the service. In the same way, a vulnerability in standalone application may expose data from the platform's owner only. However, a vulnerability in a web application may expose data of all users registered to the service.

Securing web applications is a complex task and attacks targeting web applications are on the increase [6]. If we look at the range of vulnerabilities affecting web applications [5], we can classify them into two families: Injection vulnerabilities such as SQL injection, cross-site scripting, cross-site request forgery, content spoofing and Response splitting are resulting from an incorrect handling of unexpected user inputs. Incomplete mediation vulnerabilities such as information leakage, insufficient authorization and predictable resource location are resulting from bad application design and/or misconfiguration of the platform in controlling user access to data or resources.

These attacks are hard to mitigate as current technology provides limited support. Developers are required to reason about distributed computation and to write code using heterogeneous languages, often not originally designed with distributed computing in mind nor built-in security features. In [4], Sans and Cervesato proposed Qwel, a small functional programming language extended with primitives for mobile code and remote procedure calls, two distinguishing features of web programming. The initial goal was to provide the developer with a programming language to write both client side and server code ensuring adequate interactions between them. Since Qwel is type safe language, it is more likely to mitigate injection attacks when user's inputs do not match the appropriate type. However, in its original version, Qwel does not have built-in features to mitigate incomplete mediation attacks.

In this paper, we propose to extend Qwel to provide the developer with language-level security features to control access to web applications. The main contributions of this work are 1) an extension of Qwel syntax and semantics with distributed access control mechanisms proposed by Abadi and al. [1] 2) a formalization of the policy interpreter based on the sequent calculus logic and 3) an implementation of the policy interpreter. The rest of the paper is structured as follows: Section 2 summarizes existing work in distributed access control. Section 3 introduces Qwel and lays the motivations for extending it with new security primitives for access control. Section 4.4 describes the extended Qwel syntax. Section 5 shows how this extension can be used to express common access control policies as well as more sophisticated ones. Section 6 provides a formalization of the language semantics. Section 7 concludes and provides an outline of future developments.

2 Related Work

Access control is a restriction of operations on resources such as files and services to specific users. In calculus for access control, Abadi et al. [1] present a calculus that combines authentication which is the problem of determining the identity of the requester (principal), and authorization which is the problem of determining if the principal is allowed to access certain service. Basically, access control models consist in a set of logical that grants permission to principals access resources. In [1], the concept of *principal* can be:

- Users and machines.
- Channels, such as input devices and cryptographic channels.
- Conjunction of principals, of the form $[A \wedge B]$.
- Groups, define groups of principals. The use of the group is to decide whether a principal is a member of a group.
- Principals in roles, of the form $[A \text{ as } R]$. Where principal A may adopt the role R and act under the name $[A \text{ as } R]$.
- Principals on behalf of principal, of the form $[B \text{ for } A]$. Where principal A may delegate authority to B, and B can then act on behalf of A, using the identity $[B \text{ for } A]$.

Each object have an access control list (ACL), where a request to an object will be granted if the principal is authorized according to this list. Determining whether a request from a principal granted or denied is based on the logical model that extends the algebra of principals.

3 Motivating Example

Consider an example where *Alice*, a student at *Univ* needs to submit an assignment for her course through a web portal called *Submission*. To avoid plagiarism, her professor ask to check her own assignment using an online service called *NoPlagiarism*. As a proof, her professor requires her to submit her assignment along with the similarity report obtained previously.

Qwel is an experimental programming language for the web proposed by Sans and Cervesato in [4]. At its core, Qwel is a basic functional language extended with primitives for publishing and calling web services. Using Qwel, the example can be implemented as follows:

- *NoPlagiarism.com* publishes a web service that takes a document as argument and returns the corresponding similarity report (figure 1).
- *Submission.org* publishes a service that stores a document and its similarity report both given as argument (figure 2).
- Assuming that these two services are deployed beforehand, *Alice* calls the service *similarityReport@NoPlagiarism.com* with her homework, obtains the similarity report in return and then forwards it to the *submit@submission.org* service (figure 3).

```
publish doc : string
  let
    report = calculateSimilarity(doc)
  in
    report
  end
```

Figure 1: similarityReport@NoPlagiarism.com

```
publish $s$ as $\langle doc, report\rangle$   =   $store(s)$
```

Figure 2: submit@Submission.org

```
let
    $doc$          =   "Once upon a time ..."
    $simReport$    =   call $similarityReport@NoPlagiarism.com$ with $doc$
in
   call $submit@Submission.org$ with $\langle simReport, doc\rangle$
end
```

Figure 3: Alice combining web services to submit her assignment

Beyond the functional aspects of this example, we would like to express security policies. For example, *NoPlagiarism* could express that "only students from *Univ* can get a similarity report". In the same way, *Submission* could express that "only students from *Univ* can submit their assignments" and that "similarity reports must have been issued by *NoPlagiarism*".

However, Qwel has no language features allowing service providers to express such policies. Hence, we will extend Qwel with the built-in primitives that enables developers to 1) express a local security policy protecting a published service and 2) issue credentials to users. The language interpreter will grant access to a service if and only if the local security policy is satisfied according to the credentials carried by the principal calling the service.

4 Extending Qwel With a Distributed Access Control Model

In the example above, *NoPlagiarism* wants to ensure that "only students from *Univ* can get a similarity report". In this scenario, *NoPlagiarism* does not know who is a student at *Univ*. Instead, it will expect *Univ* to issue a proof a.k.a a credential saying that *Alice* is a student. This is a typical example of distributed access control where parties can express security policies locally based on credentials issued by others. In this section, we extend the Qwel syntax with the distributed access control model proposed by Abadi and al. in [1].

4.1 Credentials

A credential is a collection of claims. For instance, *Alice* is a student according to *Univ* and *NoPlagiarism* is the issuer of the similarity report. Previous work [1] introduces the the modality says to represent such a claim. A claim is a relation between a fact, defined as a predicate, and the principal emitting such a predicate (e.g $Univ$ says $student(Alice)$). In our model, predicates can take as attributes other principals $student(Alice)$ and/or values $issuer(report)$. Claims are told to be true if and only if they are part of a credential (e.g cred($Univ$ says $student(Alice)$)). A credential cred$(e_0, \ldots, e_n)$ can have one or many claims. However, principals cannot create arbitrary credentials on behalf of others. The programmer does not use the constructor cred directly. Instead, we define say e that takes a fact and returns a credential that contains a claim emitted locally. For instance, say $student(Alice)$ creates a new credential cred($Univ$ says $student(Alice)$) when evaluated at *Univ*.

Since a credential cred$(e_0, \ldots, e_n)$ can have one or many claims, we define $e_0 \oplus e_1$ that combines different credentials. For example, once *Alice* has obtained a credential from *Univ* saying that she is a student and another from *NoPlagiarism* saying that *NoPlagiarism* is the issuer of the report returned to her. *Alice* can combined these two credentials into one and submit it to *submit@Submission.org*

$$\text{cred}(Univ \text{ says } student(Alice)) \oplus \text{cred}(NoPlagiarism \text{ says } issuer(report))$$

4.2 Access Control Policies

An access control policy restricts who can access a published service. In our model, an access control policy is an expression defining a constraint on the credentials carried by the service caller. For instance, *NoPlagiarism* can express that *Alice* must be a student to call its service *similarityReport*. To define this elementary policy, we define $\mathsf{pol}(Univ\ \mathsf{says}\ student(Alice))$ that specify that *Alice* must carry a credential in which *Univ* claims that she is a student.

To express more complex policies, we add logical operators such as $e_0 \wedge e_1$, $e_0 \vee e_1$, $\exists\ x.e(x)$ that allow the developer to combine constraints and express more complex policies. For instance, *Submission* can express that *Alice* must be a student from *Univ* and *NoPlagiarism* must be the issuer of the similarity report:

$$\mathsf{pol}(university\ \mathsf{says}\ student(Alice)) \wedge \mathsf{pol}(plagiarism\ \mathsf{says}\ issuer(report))$$

4.3 Issuing Credentials and Evaluating Policies

As introduced above, $\mathsf{pol}(Univ\ \mathsf{says}\ student(Alice))$ is the access control policy that says that *Alice* must be a student to call *similarityReport@NoPlagiarism.com*. However, it is unlikely that *NoPlagiarism* has to explicitly mention *Alice* (and any other potential other principal) in its policy. Instead, *NoPlagiarism* should write that anybody calling its service must be a student. Therefore, we need to be able to write an access control policy based on a variable representing the principal calling the service. To do so, we redefine the construct publish $w.x : \tau \Rightarrow e$ in such a way that w will be instantiated with the principal calling the service during the evaluation.

Finally, to check if an access control policy is satisfied based on the credential submitted as argument. We introduce the construct $\mathsf{check}(e_0, e_1)$ that will verify that the credentials e_0 satisfies the policy e_1.

4.4 Full Extended Syntax

To summarize, Qwel is extended with the following constructs:

$$\begin{array}{lcl}
\text{Type} & \tau & := & \mathsf{world} \ \mid\ \tau \rightsquigarrow \tau' \\
 & & \mid & \mathsf{fact} \ \mid\ \mathsf{claim} \ \mid\ \mathsf{credential} \ \mid\ \mathsf{policy} \\
\\
\text{Expression} & e & ::= & \mathsf{url(w)} \ \mid\ \mathsf{here} \\
 & & \mid & \mathsf{url}(\mathsf{w}, u) \ \mid\ \mathsf{publish}\ w.x : \tau \Rightarrow e \ \mid\ \mathsf{call}\ e_1\ \mathsf{with}\ e_2 \ \mid\ \mathsf{expect}\ e\ \mathsf{from\ w} \\
 & & \mid & p(e_1, \ldots, e_n) \ \mid\ e_1 \Rightarrow e_2 \ \mid\ e_0\ \mathsf{says}\ e_1 \\
 & & \mid & \mathsf{say}\ e \ \mid\ \mathsf{cred}(e_0, \ldots, e_n) \ \mid\ e_0 \oplus e_1 \\
 & & \mid & \mathsf{pol}(e) \ \mid\ \exists\ x : \tau.e_1 \ \mid\ e_0 \wedge e_1 \ \mid\ e_0 \vee e_1 \\
 & & \mid & \mathsf{check}(e_0, e_1)
\end{array}$$

We have not introduced the construct $e_1 \Rightarrow e_2$ yet. This construct was suggested by [1] and will be illustrated in an example shown in 5.2

4.5 Syntactic Sugar

For convenience, we extend the syntax with syntactic sugar allowing the developer to define a protected service:

$$\begin{array}{lcll}
\mathsf{publish}\ w.x : \tau \times \mathsf{credential} \Rightarrow e_0 & \triangleq & \mathsf{publish}\ w.x : \tau \times \mathsf{credential} \Rightarrow & \mathsf{if\ check}(\mathsf{snd}\ x, e_1)\ \mathsf{then}\ e_0 \\
\mathsf{protect}\ e_1 & & & \mathsf{else\ raise}\ AccessDeniedException
\end{array}$$

5 Examples

5.1 Example 1: University Submission System

Based on the extension suggested above, we are now able to write in Qwel the example presented in section 3:

1. To get the *Univ* credential, *Alice* calls *getStudentCred@Univ.edu*. The server checks if *Alice* is a student and returns a credential saying that the caller of the service is a student (figure 4).

2. To get the similarity report, *Alice* sends her university credential and her assignment to the *similarityReport@NoPlagiarism.com*. In return, she gets the similarity report and a credential specifying that *NoPlagiarism.com* is the issuer of the similarity report.

3. Finally, to submit her assignment, she combines the credentials obtained from *Univ* and from *NoPlagiarism*. Thus, she forwards the similarity report and the aggregated credential to the submission service.

```
let
  $doc$                            =  “Once uppon a time ...”
  $univCred$                       =  call $getStudentCred@Univ.edu$ with ()
  $\langle simReport, plagCred\rangle$  =  call $similarityReport@NoPlagiarism.com$ with $\langle doc, univCred\rangle$
  $univPlagCred$                   =  $univCred \oplus plagCred$
in
    call $submit@Submission.org$ with $\langle simReport, univPlagCred\rangle$
end
```

Figure 4: Alice calls for the services

```
publish $w.x$ as $x$ : unit $\Rightarrow$
  if $checkStudent(w)$
     then  say $student(w)$
     else  raise $AccessDeniedException$
```

Figure 5: getStudentCred@Univ.edu

```
publish $w.x$ as $\langle doc, cred\rangle$ : string $\times$ credential $\Rightarrow$
  let
      $report$    =  $\langle doc, calculateSimilarity(doc)\rangle$
      $plagCred$  =  say $issuer(report)$
  in
     $\langle report, plagCred\rangle$
  end
protect
  pol($Univ.edu$ says $student(w)$)
```

Figure 6: similarityReport@NoPlagiarism.com

5.2 Example 2: Managing Medical Reports at the Hospital

Consider an example of an hospital in which medical reports are managed electronically. In figure 8, the hospital system publishes a service that returns the medical report corresponding to the id given as argument. This service is ruled by the following policy:

```
publish $w.x$ as $\langle report, cred\rangle$ : (string × string) × credential $\Rightarrow store(s)$
protect
  pol($Univ.edu$ says $student(w)$) ∧ pol($NoPlagiarism.com$ says $issuer(report)$)
```

Figure 7: submit@Submission.org

- **Rule 1**: the patient can access his own medical report
- **Rule 2**: any doctor working for the hospital can access any medical report
- **Rule 3**: anybody that is explicitly allowed by the patient can access to the patient medical report

Rule 1 and rule 2 can be implemented using the constructs introduced previously. However, rule 3 can be seen as a delegation rule: "anybody speaking on behalf of the owner can access the medical report". For instance, *Alice* can delegate authority to her *Grandma* in order for her to access Alice's medical report. For that purpose, we introduce the construct $w_1 \Rightarrow w_2$ (also defined by [1]) that allows a principal to delegate authority to another principal.

To satisfy rule 1, *Alice* must obtain a credential from the hospital saying that she is the owner of a certain medical report with a specific id (figures 9 and 12).

To satisfy rule 2, *Bob* must obtain a credential from the hospital saying that he is a doctor (figures 11 and 10).

To satisfy rule 3, *Alice's Grandma* must obtain a credential from Alice saying that she can speaks on her behalf (figures 13 and 14).

```
publish $w.s$ as $\langle id, cred\rangle$ : int × credential $\Rightarrow retrieve(id)$
protect
  pol($hospital$ says $owner(id, w)$)
  ∨ pol($hospital$ says $doctor(w)$)
  ∨ $\exists\, w'$ : world.$w \Rightarrow w' \wedge hospital$ says $owner(id, w')$
```

Figure 8: Get medical report service at the hospital

```
publish $w.x$ : unit $\Rightarrow$
  let
    $id = getPatientId(w)$
  in
    say $owner(id, w)$
  end
```

Figure 9: getMedicalReportCred@hospital

```
let
  $doctorCred$ = call $getDoctorCred@hospital$ with ()
in
  call $getMedicalReport@hospital$ with $\langle 2136, doctorCred\rangle$
end
```

Figure 10: Call get medical report service by doctor (Bob)

```
publish $w.x$ : unit $\Rightarrow$
  if $isDoctor(w)$
    then say $doctor(w)$
    else raise $AccessDeniedException$
```

Figure 11: getDoctorCred@hospital

```
let
  $reportCred$ = call $getMedicalReportCred@hospital$ with ()
in
  call $getMedicalReport@hospital$ with $\langle 2136, reportCred\rangle$
end
```

Figure 12: Call get medical report service by the patient (Alice)

```
publish $w.x$ : unit $\Rightarrow$
  if $isMyGrandma(w)$
    then say $w \Rightarrow$ here
    else raise $AccessDeniedException$
```

Figure 13: getAliceDelegationCred@Alice

```
let
  $delegationCred$ = call $getAliceDelegationCred@Alice$ with ()
in
  call $getMedicalReport@hospital$ with $\langle 2136, delegationCred\rangle$
end
```

Figure 14: Call get medical report service by grandma who speaks for Alice

6 Semantics

The static semantics of the proposed Qwel extension is defined in figures 15 and 16. The dynamic semantics is defined in figures 17, 18, 19 and 20. We formalized the policy evaluation based on an extended sequent calculus logic (figure 21). The sequent calculus [3] is a simple set of rules that can be used to show the truth of statements in first order logic. As a proof of concept, we have developed a policy evaluator implementing the sequent calculus rules (see appendix).

$$\frac{}{\Sigma; \Gamma \vdash_{\mathsf{w}} \mathsf{url}(\mathsf{w}') : \mathsf{world}}\ \texttt{of\_url} \qquad \frac{}{\Sigma; \Gamma \vdash_{\mathsf{w}} \mathsf{here} : \mathsf{world}}\ \texttt{of\_here}$$

$$\frac{}{\Sigma, u : \tau \rightsquigarrow \tau' \mathbin{@} \mathsf{w}'; \Gamma \vdash_{\mathsf{w}} \mathsf{url}(\mathsf{w}', u) : \tau \rightsquigarrow \tau'}\ \texttt{of\_url} \qquad \frac{\Sigma; \Gamma, w : \mathsf{world}, x : \tau \vdash_{\mathsf{w}} e : \tau'}{\Sigma; \Gamma \vdash_{\mathsf{w}} \mathsf{publish}\ w.x : \tau \Rightarrow e : \tau \rightsquigarrow \tau'}\ \texttt{of\_publish}$$

$$\frac{\Sigma; \Gamma \vdash_{\mathsf{w}} e_1 : \tau \rightsquigarrow \tau' \qquad \Sigma; \Gamma \vdash_{\mathsf{w}} e_2 : \tau}{\Sigma; \Gamma \vdash_{\mathsf{w}} \mathsf{call}\ e_1\ \mathsf{with}\ e_2 : \tau'}\ \texttt{of\_call} \qquad \frac{\Sigma; \Gamma \vdash_{\mathsf{w}'} e : \tau}{\Sigma; \Gamma \vdash_{\mathsf{w}} \mathsf{expect}\ e\ \mathsf{from}\ \mathsf{w}' : \tau}\ \texttt{of\_expect}$$

Figure 15: Typing rules for modified Qwel constructs

$$\frac{\Sigma;\Gamma \vdash_{\mathsf{w}} e_1 : \tau_1 \quad \ldots \quad \Sigma;\Gamma \vdash_{\mathsf{w}} e_n : \tau_n}{\Sigma;\Gamma \vdash_{\mathsf{w}} p(e_1,\ldots,e_n) : \mathsf{fact}}\ {\scriptstyle \mathtt{of\_p}} \qquad \frac{\Sigma;\Gamma \vdash_{\mathsf{w}} e_1 : \mathsf{world} \quad \Sigma;\Gamma \vdash_{\mathsf{w}} e_2 : \mathsf{world}}{\Sigma;\Gamma \vdash_{\mathsf{w}} e_1 \Rightarrow e_2 : \mathsf{fact}}\ {\scriptstyle \mathtt{of\_speaksfor}}$$

$$\frac{\Sigma;\Gamma \vdash_{\mathsf{w}} e_0 : \mathsf{world} \quad \Sigma;\Gamma \vdash_{\mathsf{w}} e_1 : \mathsf{fact}}{\Sigma;\Gamma \vdash_{\mathsf{w}} e_0 \ \mathsf{says}\ e_1 : \mathsf{claim}}\ {\scriptstyle \mathtt{of\_says}}$$

$$\frac{\Sigma;\Gamma \vdash_{\mathsf{w}} e : \mathsf{fact}}{\Sigma;\Gamma \vdash_{\mathsf{w}} \mathsf{say}\ e : \mathsf{credential}}\ {\scriptstyle \mathtt{of\_say}} \qquad \frac{\Sigma;\Gamma \vdash_{\mathsf{w}} e_0 : \mathsf{claim} \quad \ldots \quad \Sigma;\Gamma \vdash_{\mathsf{w}} e_n : \mathsf{claim}}{\Sigma;\Gamma \vdash_{\mathsf{w}} \mathsf{cred}(e_0,\ldots,e_n) : \mathsf{credential}}\ {\scriptstyle \mathtt{of\_cred}}$$

$$\frac{\Sigma;\Gamma \vdash_{\mathsf{w}} e_0 : \mathsf{credential} \quad \Sigma;\Gamma \vdash_{\mathsf{w}} e_1 : \mathsf{credential}}{\Sigma;\Gamma \vdash_{\mathsf{w}} e_0 \oplus e_1 : \mathsf{credential}}\ {\scriptstyle \mathtt{of\_join}}$$

$$\frac{\Sigma;\Gamma \vdash_{\mathsf{w}} e : \mathsf{claim}}{\Sigma;\Gamma \vdash_{\mathsf{w}} \mathsf{pol}(e) : \mathsf{policy}}\ {\scriptstyle \mathtt{of\_pol}} \qquad \frac{\Sigma;\Gamma, x:\tau \vdash_{\mathsf{w}} e : \mathsf{policy}}{\Sigma;\Gamma \vdash_{\mathsf{w}} \exists\, x:\tau.e : \mathsf{policy}}\ {\scriptstyle \mathtt{of\_exists}}$$

$$\frac{\Sigma;\Gamma \vdash_{\mathsf{w}} e_0 : \mathsf{policy} \quad \Sigma;\Gamma \vdash_{\mathsf{w}} e_1 : \mathsf{policy}}{\Sigma;\Gamma \vdash_{\mathsf{w}} e_0 \wedge e_1 : \mathsf{policy}}\ {\scriptstyle \mathtt{of\_and}} \qquad \frac{\Sigma;\Gamma \vdash_{\mathsf{w}} e_0 : \mathsf{policy} \quad \Sigma;\Gamma \vdash_{\mathsf{w}} e_1 : \mathsf{policy}}{\Sigma;\Gamma \vdash_{\mathsf{w}} e_0 \vee e_1 : \mathsf{policy}}\ {\scriptstyle \mathtt{of\_or}}$$

$$\frac{\Sigma;\Gamma \vdash_{\mathsf{w}} e_0 : \mathsf{credential} \quad \Sigma;\Gamma \vdash_{\mathsf{w}} e_1 : \mathsf{policy}}{\Sigma;\Gamma \vdash_{\mathsf{w}} \mathsf{check}(e_0,e_1) : \mathsf{boolean}}\ {\scriptstyle \mathtt{of\_check}}$$

Figure 16: Typing Rules for new Qwel constructs

$$\frac{}{\mathsf{url}(\mathsf{w}')\ \mathsf{val}}\ {\scriptstyle \mathtt{val\_url}} \qquad \frac{m}{j}\ {\scriptstyle \mathtt{val\_url}} \quad n\mathsf{url}(\mathsf{w}', u)\ \mathsf{val}$$

$$\frac{}{\Delta\ ;\ \mathsf{here}\ \mapsto_{\mathsf{w}}\ \Delta\ ;\ \mathsf{url}(\mathsf{w})}\ {\scriptstyle \mathtt{ev\_here}}$$

$$\frac{}{\Delta\ ;\ \mathsf{publish}\ w.x:\tau \Rightarrow e\ \mapsto_{\mathsf{w}}\ (\Delta, u\ @\ \mathsf{w} \hookrightarrow w.x\!:\!\tau.e)\ ;\ \mathsf{url}(\mathsf{w}, u)}\ {\scriptstyle \mathtt{ev\_publish}}$$

$$\frac{\Delta\ ;\ e_1\ \mapsto_{\mathsf{w}}\ \Delta'\ ;\ e_1'}{\Delta\ ;\ \mathsf{call}\ e_1\ \mathsf{with}\ e_2\ \mapsto_{\mathsf{w}}\ \Delta'\ ;\ \mathsf{call}\ e_1'\ \mathsf{with}\ e_2}\ {\scriptstyle \mathtt{ev\_call_1}} \qquad \frac{v_1\ \mathsf{val} \quad \Delta\ ;\ e_2\ \mapsto_{\mathsf{w}}\ \Delta'\ ;\ e_2'}{\Delta\ ;\ \mathsf{call}\ v_1\ \mathsf{with}\ e_2\ \mapsto_{\mathsf{w}}\ \Delta'\ ;\ \mathsf{call}\ v_1\ \mathsf{with}\ e_2'}\ {\scriptstyle \mathtt{ev\_call_2}}$$

$$\frac{v_2\ \mathsf{val}}{\underbrace{(\Delta^*, u\ @\ \mathsf{w}' \hookrightarrow w.x\!:\!\tau.e)}_{\Delta}\ ;\ \mathsf{call}\ \mathsf{url}(\mathsf{w}', u)\ \mathsf{with}\ v_2\ \mapsto_{\mathsf{w}}\ \Delta'\ ;\ \mathsf{expect}\ [\mathsf{url}(\mathsf{w}), v_2/w, x]e\ \mathsf{from}\ \mathsf{w}'}\ {\scriptstyle \mathtt{ev\_call_3}}$$

$$\frac{\Delta\ ;\ e\ \mapsto_{\mathsf{w}'}\ \Delta'\ ;\ e'}{\Delta\ ;\ \mathsf{expect}\ e\ \mathsf{from}\ \mathsf{w}'\ \mapsto_{\mathsf{w}}\ \Delta'\ ;\ \mathsf{expect}\ e'\ \mathsf{from}\ \mathsf{w}'}\ {\scriptstyle \mathtt{exp_1}} \qquad \frac{v\ \mathsf{val}}{\Delta\ ;\ \mathsf{expect}\ v\ \mathsf{from}\ \mathsf{w}'\ \mapsto_{\mathsf{w}}\ \Delta\ ;\ v}\ {\scriptstyle \mathtt{exp_2}}$$

Figure 17: Evaluation Rules for modified Qwel constructs

$$\frac{v_0\ \mathsf{val}\quad \dots \quad v_n\ \mathsf{val}}{p(v_0,\dots,v_n)\ \mathsf{val}}\ \texttt{val\_pred} \qquad \frac{}{\mathsf{url}(\mathsf{w}_0) \Rightarrow \mathsf{url}(\mathsf{w}_1)\ \mathsf{val}}\ \texttt{val\_speaksfor} \qquad \frac{v\ \mathsf{val}}{\mathsf{url}(\mathsf{w})\ \mathsf{says}\ v\ \mathsf{val}}\ \texttt{val\_says}$$

$$\frac{\Delta\ ;\ e_0\ \mapsto_{\mathsf{w}}\ \Delta'\ ;\ e_0'}{\Delta\ ;\ p(e_0,\dots,e_n)\ \mapsto_{\mathsf{w}}\ \Delta'\ ;\ p(e_0',\dots,e_n)}\ \texttt{pred}_1 \qquad \frac{v_i\ \mathsf{val}\quad \Delta\ ;\ e_{i+1}\ \mapsto_{\mathsf{w}}\ \Delta'\ ;\ e_{i+1}'}{\Delta\ ;\ p(\dots,v_i,e_{i+1},\dots)\ \mapsto_{\mathsf{w}}\ \Delta'\ ;\ p(\dots,v_i,e_{i+1}',\dots)}\ \texttt{pred}_2$$

$$\frac{\Delta\ ;\ e_0\ \mapsto_{\mathsf{w}}\ \Delta'\ ;\ e_0'}{\Delta\ ;\ e_0 \Rightarrow e_1\ \mapsto_{\mathsf{w}}\ \Delta'\ ;\ e_0' \Rightarrow e_1}\ \texttt{speaksfor} \qquad \frac{\Delta\ ;\ e_1\ \mapsto_{\mathsf{w}}\ \Delta'\ ;\ e_1'}{\Delta\ ;\ \mathsf{url}(\mathsf{w}_0) \Rightarrow e_1\ \mapsto_{\mathsf{w}}\ \Delta'\ ;\ \mathsf{url}(\mathsf{w}_0) \Rightarrow e_1'}\ \texttt{speaksfor}_2$$

$$\frac{\Delta\ ;\ e_0\ \mapsto_{\mathsf{w}}\ \Delta'\ ;\ e_0'}{\Delta\ ;\ e_0\ \mathsf{says}\ e_1\ \mapsto_{\mathsf{w}}\ \Delta'\ ;\ e_0'\ \mathsf{says}\ e_1}\ \texttt{says} \qquad \frac{\Delta\ ;\ e_1\ \mapsto_{\mathsf{w}}\ \Delta'\ ;\ e_1'}{\Delta\ ;\ \mathsf{url}(\mathsf{w}_0)\ \mathsf{says}\ e_1\ \mapsto_{\mathsf{w}}\ \Delta'\ ;\ \mathsf{url}(\mathsf{w}_0)\ \mathsf{says}\ e_1'}\ \texttt{says}_2$$

Figure 18: Evaluation Rules for Qwel claim constructs

$$\frac{v_0\ \mathsf{val}\quad \dots \quad v_n\ \mathsf{val}}{\mathsf{cred}(\mathsf{url}(\mathsf{w}_0)\ \mathsf{says}\ v_0,\dots,\mathsf{url}(\mathsf{w}_n)\ \mathsf{says}\ v_n)\ \mathsf{val}}\ \texttt{val\_cred}$$

$$\frac{\Delta\ ;\ e_0\ \mapsto_{\mathsf{w}}\ \Delta'\ ;\ e_0'}{\Delta\ ;\ \mathsf{cred}(e_0,\dots,e_n)\ \mapsto_{\mathsf{w}}\ \Delta'\ ;\ \mathsf{cred}(e_0',\dots,e_n)}\ \texttt{cred}$$

$$\frac{\Delta\ ;\ e_{i+1}\ \mapsto_{\mathsf{w}}\ \Delta'\ ;\ e_{i+1}'}{\Delta\ ;\ \mathsf{cred}(\dots,\mathsf{url}(\mathsf{w}_i)\ \mathsf{says}\ v_n,e_{i+1},\dots)\ \mapsto_{\mathsf{w}}\ \Delta'\ ;\ \mathsf{cred}(\dots,\mathsf{url}(\mathsf{w}_i)\ \mathsf{says}\ v_n,e_{i+1}',\dots)}\ \texttt{cred}_2$$

$$\frac{\Delta\ ;\ e\ \mapsto_{\mathsf{w}}\ \Delta'\ ;\ e}{\Delta\ ;\ \mathsf{say}\ e\ \mapsto_{\mathsf{w}}\ \Delta'\ ;\ \mathsf{say}\ e'}\ \texttt{say}_1 \qquad \frac{v\ \mathsf{val}}{\Delta\ ;\ \mathsf{say}\ v\ \mapsto_{\mathsf{w}}\ \Delta'\ ;\ \mathsf{cred}(\mathsf{url}(\mathsf{w}_0)\ \mathsf{says}\ v)}\ \texttt{say}_2$$

$$\frac{\Delta\ ;\ e_0\ \mapsto_{\mathsf{w}}\ \Delta'\ ;\ e_0'}{\Delta\ ;\ e_0\ \oplus\ e_1\ \mapsto_{\mathsf{w}}\ \Delta'\ ;\ e_0'\ \oplus\ e_1}\ \texttt{join}_1 \qquad \frac{\Delta\ ;\ e_1\ \mapsto_{\mathsf{w}}\ \Delta'\ ;\ e_1'}{\Delta\ ;\ \mathsf{cred}(v_{0_0},\dots,v_{0_n})\ \oplus\ e_1\ \mapsto_{\mathsf{w}}\ \Delta'\ ;\ \mathsf{cred}(v_{0_0},\dots,v_{0_n})\ \oplus\ e_1'}\ \texttt{join}_2$$

$$\frac{}{\Delta\ ;\ \mathsf{cred}(v_{0_0},\dots,v_{0_n})\ \oplus\ \mathsf{cred}(v_{1_0},\dots,v_{1_n})\ \mapsto_{\mathsf{w}}\ \Delta\ ;\ \mathsf{cred}(v_{0_0},\dots,v_{0_n},v_{1_0},\dots,v_{1_n})}\ \texttt{join}_3$$

Figure 19: Evaluation Rules for Qwel credential constructs

7 Conclusion and Future Work

In conclusion, the main goal of the thesis is to extend Qwel language syntax and semantics providing developers with a mean to issue credentials and protect web services with access control policies. This extension defines an expressive language, yet simple and easy to use for the purpose of building web services with embedded security constraints.

There are several avenues for future work, in our model the caller is responsible for getting the credentials and sends them to the server that will try to prove that they satisfy the policy. This can be overwhelming for the server

$$\frac{v\ \mathsf{val}}{\mathsf{pol}(\mathsf{url}(\mathsf{w})\ \mathsf{says}\ v)\ \mathsf{val}}\ \texttt{val\_pol} \qquad \frac{}{\exists\, x : \tau.e\ \mathsf{val}}\ \texttt{val\_exists} \qquad \frac{v_0\ \mathsf{val} \quad v_1\ \mathsf{val}}{v_0 \wedge v_1\ \mathsf{val}}\ \texttt{val\_andp} \qquad \frac{v_0\ \mathsf{val} \quad v_1\ \mathsf{val}}{v_0 \vee v_1\ \mathsf{val}}\ \texttt{val\_orp}$$

$$\frac{\Delta\,;\, e \ \mapsto_{\mathsf{w}}\ \Delta'\,;\, e}{\Delta\,;\, \mathsf{pol}(e) \ \mapsto_{\mathsf{w}}\ \Delta'\,;\, \mathsf{pol}(e')}\ \texttt{pol}$$

$$\frac{\Delta\,;\, e_0 \ \mapsto_{\mathsf{w}}\ \Delta'\,;\, e_0'}{\Delta\,;\, e_0 \wedge e_1 \ \mapsto_{\mathsf{w}}\ \Delta'\,;\, e_0' \wedge e_1}\ \texttt{and}_1 \qquad \frac{v_0\ \mathsf{val} \quad \Delta\,;\, e_1 \ \mapsto_{\mathsf{w}}\ \Delta'\,;\, e_1'}{\Delta\,;\, v_0 \wedge e_1 \ \mapsto_{\mathsf{w}}\ \Delta'\,;\, v_0 \wedge e_1'}\ \texttt{and}_2$$

$$\frac{\Delta\,;\, e_0 \ \mapsto_{\mathsf{w}}\ \Delta'\,;\, e_0'}{\Delta\,;\, e_0 \vee e_1 \ \mapsto_{\mathsf{w}}\ \Delta'\,;\, e_0' \vee e_1}\ \texttt{or}_1 \qquad \frac{v_0\ \mathsf{val} \quad \Delta\,;\, e_1 \ \mapsto_{\mathsf{w}}\ \Delta'\,;\, e_1'}{\Delta\,;\, v_0 \vee e_1 \ \mapsto_{\mathsf{w}}\ \Delta'\,;\, v_0 \vee e_1'}\ \texttt{or}_2$$

$$\frac{\Delta\,;\, e_0 \ \mapsto_{\mathsf{w}}\ \Delta'\,;\, e_0'}{\Delta\,;\, \mathsf{check}(e_0, e_1) \ \mapsto_{\mathsf{w}}\ \Delta'\,;\, \mathsf{check}(e_0', e_1)}\ \texttt{check+1} \qquad \frac{v_0\ \mathsf{val} \quad \Delta\,;\, e_1 \ \mapsto_{\mathsf{w}}\ \Delta'\,;\, e_1'}{\Delta\,;\, \mathsf{check}(v_0, e_1) \ \mapsto_{\mathsf{w}}\ \Delta'\,;\, \mathsf{check}(v_0, e_1')}\ \texttt{check}_2$$

$$\frac{v_0\ \mathsf{val} \quad \ldots \quad v_n\ \mathsf{val} \quad v\ \mathsf{val} \quad v_0, \ldots, v_n \models v}{\Delta\,;\, \mathsf{check}(\mathsf{cred}(v_0, \ldots, v_n), v) \ \mapsto_{\mathsf{w}}\ \Delta\,;\, \mathsf{true}}\ \texttt{check}_3 \qquad \frac{v_0\ \mathsf{val} \quad \ldots \quad v_n\ \mathsf{val} \quad v\ \mathsf{val} \quad v_0, \ldots, v_n \not\models v}{\Delta\,;\, \mathsf{check}(\mathsf{cred}(v_0, \ldots, v_n), v) \ \mapsto_{\mathsf{w}}\ \Delta\,;\, \mathsf{false}}\ \texttt{check}_4$$

Figure 20: Evaluation Rules for Qwel policy constructs

$$\frac{}{\Gamma, p(x_1, \ldots, x_n) \vdash p(x_1, \ldots, x_n), \Delta}\ \texttt{pred} \qquad \frac{}{\Gamma, \mathsf{w}_1 \Rightarrow \mathsf{w}_2 \vdash \mathsf{w}_1 \Rightarrow \mathsf{w}_2, \Delta}\ \Rightarrow \qquad \frac{\rho \vdash \rho'}{\Gamma, \mathsf{w}\ \mathsf{says}\ \rho \vdash \mathsf{w}\ \mathsf{says}\ \rho', \Delta}\ \texttt{says}$$

$$\frac{\Gamma, \rho_1, \rho_2 \vdash \Delta}{\Gamma, \rho_1 \wedge \rho_2 \vdash \Delta}\ \wedge\mathrm{L} \qquad \frac{\Gamma \vdash \rho_1, \Delta \quad \Gamma \vdash \rho_2, \Delta}{\Gamma \vdash \rho_1 \wedge \rho_2, \Delta}\ \wedge\mathrm{R}$$

$$\frac{\Gamma, \rho_1 \vdash \Delta \quad \Gamma, \rho_2 \vdash \Delta}{\Gamma, \rho_1 \vee \rho_2 \vdash \Delta}\ \vee\mathrm{L} \qquad \frac{\Gamma \vdash \rho_1, \rho_2, \Delta}{\Gamma \vdash \rho_1 \vee \rho_2, \Delta}\ \vee\mathrm{R}$$

$$\frac{\Gamma \vdash \rho_1, \Delta \quad \Gamma, \rho_2 \vdash \Delta}{\Gamma, \rho_1 \to \rho_2 \vdash \Delta}\ \to\mathrm{L} \qquad \frac{\Gamma, \rho_1 \vdash \rho_2, \Delta}{\Gamma \vdash \rho_1 \to \rho_2, \Delta}\ \to\mathrm{R}$$

$$\frac{\Gamma, \rho(x) \vdash \Delta}{\Gamma, \exists\, x.\rho(x) \vdash \Delta}\ \vee\mathrm{L} \qquad \frac{\Gamma \vdash \rho(z), \Delta}{\Gamma \vdash \exists\, x.\rho(x), \Delta}\ \vee\mathrm{R}$$

Figure 21: Policy evaluation

when dealing with multiple parallel service calls. In Proof carrying authorization (PCA) [2], the service provider sends its policy to the caller. The latter must build a proof that his/her credentials satisfy the policy. If such a proof can be derived, this proof is sent sent back to the server. Hence, the server simply need to verify the soundness of the proof rather than trying to find one. In the future, we want to adapt the PCA model in Qwel.

References

[1] M. Abadi, M. Burrows, B. Lampson, G. Plotkin, J. Kohl, C. Neuman, and J. Steiner. A calculus for access control in distributed systems, 1991.

[2] Ljudevit Bauer. *Access control for the web via proof-carrying authorization*. PhD thesis, Princeton, NJ, USA, 2003. AAI3107865.

[3] Gerhard Gentzen. Investigations into logical deduction. *American philosophical quarterly*, 1(4):288–306, 1964.

[4] Thierry Sans and Iliano Cervesato. QWeSST for Type-Safe Web Programming. In Berndt Farwer, editor, *Third International Workshop on Logics, Agents, and Mobility — LAM'10*, volume 7 of *EPiC*, pages 96–111, Edinburgh, Scotland, UK, 15 July 2010. EasyChair Publications.

[5] WhiteHat Security. Website Statistics Report, 2012.

[6] Symantec. Internet Security Threat Report, 2013.

A SML implementation of the policy evaluator

```
type world = string

type value = string

type pr = string

datatype expression = v of value
                    | w of world

datatype proposition = Pred of world * pr  * expression list
                     | speaksfor of world * world * world

datatype formula = prep  of proposition
                 | andf of formula * formula
                 | orf of formula * formula
                 | Exists of expression * formula

(* listequal: list * list -> boolean*)
fun listequal [] [] = true
    | listequal (x::l1) (y::l2) = (x=y) andalso listequal l1 l2
    | listequal _  _ = false


(* getValueDomain: proposition list -> proposition list *)
fun  getValueDomain(Pred(w',q,l)::pl) =
            (List.filter (fn v(e) => true | w(e) => false) l)@getValueDomain(pl)
    | getValueDomain(speaksfor((w1,w2,w3))::pl) = getValueDomain(pl)
    | getValueDomain([]) = []

(* getWorldDomain: proposition list -> proposition list *)
fun getWorldDomain(Pred(w',q,l)::pl) = w(w')::(List.filter (fn v(e) => false | w(e) => true) l)
                                             @getWorldDomain(pl)
  | getWorldDomain(speaksfor((w1,w2,w3))::pl) =  w(w1)::w(w2)::w(w3)::getWorldDomain(pl)
  | getWorldDomain([]) = []


(* replaceValueList: value * value * expression list -> expression list *)
fun replaceValueList(v1,v2,v(expr)::l) =  if(v1=expr)
                                          then v(v2)::replaceValueList(v1,v2,l)
                                          else v(expr)::replaceValueList(v1,v2,l)
    | replaceValueList(v1,v2,w(expr)::l) = w(expr)::replaceValueList(v1,v2,l)
    | replaceValueList(v1,v2,[]) = []

(* replaceValue: value * value * formula -> formula *)
fun replaceValue(v1,v2, prep(Pred(w',q,l))) = prep(Pred(w',q,replaceValueList(v1,v2,l)))
  | replaceValue(v1,v2, prep(speaksfor(w3,w4,w5))) = prep(speaksfor(w3,w4,w5))
  | replaceValue(v1,v2, andf(f1,f2)) = andf(replaceValue(v1,v2,f1),replaceValue(v1,v2,f2))
  | replaceValue(v1,v2, orf(f1,f2)) = orf(replaceValue(v1,v2,f1),replaceValue(v1,v2,f2))
  | replaceValue(v1,v2, Exists(v(expr),f)) = if (expr = v1) then Exists(v(expr),f)
                                             else Exists(v(expr),replaceValue(v1,v2,f))
  | replaceValue(v1,v2, Exists(w(expr),f)) = Exists(w(expr),replaceValue(v1,v2,f))

(* replaceWorldList: value * value * expression list -> expression list *)
fun replaceWorldList(w1,w2,w(expr)::l) =  if(w1=expr)
                                          then w(w2)::replaceWorldList(w1,w2,l)
                                          else w(expr)::replaceWorldList(w1,w2,l)
  | replaceWorldList(w1,w2,v(expr)::l) = v(expr)::replaceWorldList(w1,w2,l)
  | replaceWorldList(w1,w2,[]) = []

(* replaceWorld: value * value * formula -> formula *)
fun replaceWorld(w1,w2, prep(Pred(w',q,l))) = if (w'=w1)
                                              then prep(Pred(w2,q,replaceWorldList(w1,w2,l)))
                                              else prep(Pred(w',q,replaceWorldList(w1,w2,l)))
| replaceWorld(w1,w2, prep(speaksfor(w3,w4,w5))) = let
                                                    val w3' = if (w3=w1) then w2 else w3
                                                    val w4' = if (w4=w1) then w2 else w4
                                                    val w5' = if (w5=w1) then w2 else w5
                                                   in
                                                    prep(speaksfor(w3',w4',w5'))
                                                   end
| replaceWorld(w1,w2, andf(f1,f2)) = andf(replaceWorld(w1,w2,f1),replaceWorld(w1,w2,f2))
| replaceWorld(w1,w2, orf(f1,f2)) = orf(replaceWorld(w1,w2,f1),replaceWorld(w1,w2,f2))
| replaceWorld(w1,w2, Exists(w(expr),f)) = if (expr = w1) then Exists(w(expr),f)
                                           else Exists(w(expr),replaceWorld(w1,w2,f))
| replaceWorld(w1,w2, Exists(v(expr),f)) = Exists(v(expr),replaceWorld(w1,w2,f))

val removeDuplicates:(expression list -> expression list) =
        List.foldl (fn (x,b)=> if (List.exists (fn y=>(y=x)) b) then b else x::b) []



(* prove: proposition list * formula -> boolean*)
fun prove(model, policy) =
  let
    (* check: expression list * expression list * proposition list * formula list -> boolean *)
    fun check(d, d', model, prep(p)::f) = (List.exists (fn x => case (x,p)
                                              of (Pred(w',q,l),Pred(w'',q',l')) => (w''=w')
                                                  andalso (q=q') andalso (listequal l l')
                                               | (speaksfor(w1,w2,w3),speaksfor(w1',w2',w3')) =>
                                                     (w1=w1') andalso (w2=w2') andalso (w3=w3')
                                               | _ => false)
                                                        model) orelse check(d,d',model,f)
      | check(d, d',  model, andf(pol1,pol2)::f) = check(d,d', model,pol1::f)
                                                   andalso check(d,d', model, pol2::f)
      | check(d, d', model, orf(pol1,pol2)::f) = check(d,d', model, pol1::pol2::f)
      | check(d, d', model, Exists(v(x),pol)::f) =
            check (d,d', model,(List.map (fn v(y):expression => replaceValue(x,y,pol)
                                                      | w(y) => pol) d')@f)
      | check(d, d', model, Exists(w(x),pol)::f) =
            check (d,d', model,(List.map (fn w(y):expression => replaceWorld(x,y,pol)
                                                      | v(y) => pol) d)@f)
      | check(d, d', model, []) = false

  in
      check(removeDuplicates(getWorldDomain(model)),
            removeDuplicates(getValueDomain(model)),
            model, [policy])
  end
```

Carnegie Mellon University Qatar

Unsupervised Arabic Word Segmentation and Statistical Machine Translation

Senior Thesis

School of Computer Science

Hanan Alshikhabobakr

halshikh@qatar.cmu.edu

Advisor: Kemal Oflazer

ko@cs.cmu.edu

Co-advisor: Mohit Behrang

behrang@cmu.edu

May 2013

ABSTRACT

Word segmentation is a necessary step for Natural Language Processing (NLP) for morphologically rich languages, such as Arabic. In this thesis, we experiment with unsupervised word segmentation systems proposed in the literature, to perform segmentation on Arabic, and couple word segmentation with Statistical Machine Translation (SMT). Our results indicate that unsupervised segmentation systems turn out to be inaccurate and do not help with improving SMT quality. Although minimal automatic post-processing improves the translation accuracy, word baseline accuracy turn out to be better. We conclude that semi-supervised word segmentation systems have more potential to improve Arabic to English translation in SMT.

ACKNOWLEDGEMENTS

I sincerely thank my advisors Prof. Kemal Oflazer and Dr. Mohit Behrang for their constant support and guidance throughout this research. Although it has been my first time to get exposed to NLP, my advisors were extremely helpful and patient in my learning process. I would also like to forward my sincere gratitude to Prof. Mark Stehlik for his constant motivation and support to me throughout the year.

I definitely owe a lot to my friends and family who were by my side whenever I needed them and for their support to me during all the hard times I faced.

Contents

1. Introduction 4

2. Literature Review 4

2.1 Word Segmentation 4

2.2 Unsupervised Word Segmentation Systems 5

2.2 Statistical Machine Translation 6

3. Methodology 7

3.1 Data 7

3.2 The Segmentation Task 7

3.3 The Translation Task 8

4 Evaluation 9

4.1 Evaluation of Word Segmentation 9

4.2 Evaluation of Statistical Machine Translation 9

5. Experiments and results 10

6. Conclusions 11

7. References 12

1. Introduction

Word segmentation plays an important role for morphologically rich languages in many NLP applications. Arabic is a morphologically rich language, so we use it in this research as the target language for segmentation. Although there are accurate word segmentation systems for Arabic, such as MADA (Habash, 2007), they are manually-built systems that incorporate rules of the Arabic language and their exceptions. In this work, we look at unsupervised word segmentation systems to see how well they perform word segmentation, without relying on any linguistic information about the language. Hence the methodology of this research can be applied to many other morphologically-complex languages. We focus on three leading unsupervised word segmentation systems in the literature: Morfessor (Creutz and Lagus, 2002), ParaMor (Monson, 2007), and Demberg's system (Demberg, 2007). For each of the three systems, we train segmentation models from the same training set and test accuracy on a test set. We then apply the word segmentation model in an NLP application, statistical machine translation (SMT). As a result we observe that Morfessor works best with SMT, and when we apply minimal post-processing on its segmentations, it gets closer to the baseline, as it improves translation by a factor of 3 from the original result obtained from Morfessor.

Based on our observation we conclude that 1) unsupervised segmentation models does not seem to improve MT output quality, 2) unsupervised segmentation accuracy does not predict SMT output quality, and 3) some additional post-processing could help.

2. Literature Review

2.1 Word Segmentation

Word segmentation break words into grammatically meaningful segments, which we refer to as morphemes. For example, "meaningless" could be segmented into "mean+ing+less", where each segment (or morpheme) has a grammatical meaning/function. Figure 1 illustrates a word segmentation example for the word "talking" and for its Arabic equivalent in meaning:

In this work we investigate three unsupervised word segmentation systems and one manually-built system.

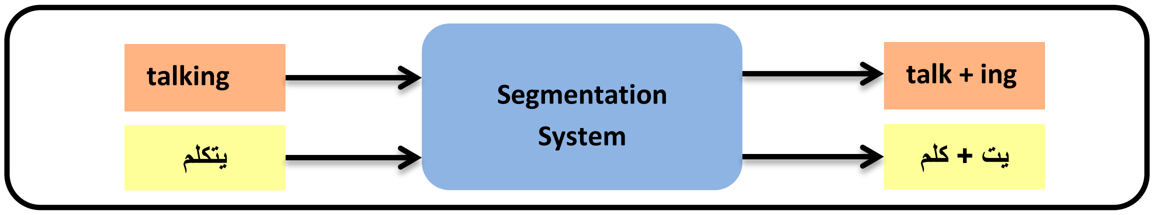

Figure 1: Examples of word segmentation for English and Arabic

2.2 Unsupervised Word Segmentation Systems

An unsupervised word segmentation system is one which learns the segmentation from a list of words that are not annotated or pre-processed in any way that helps the system to predict the correct segmentation. The main task of an unsupervised system is to create a segmentation model that then can take new words and output their segmentation.

We study the word segmentation performance of three unsupervised systems: Morfessor (Creutz and Lagus, 2002), ParaMor (Monson, 2007), and Demberg's system (Demberg, 2007). We briefly describe each of the systems below. We also experiment with a manually-built system for Arabic words Segmentation, MADA (Habash et al., 2008), and use it as a standard for some of our evaluations.

Morfessor

Morfessor tries to discover the most compact description of the data (that is, the set of words). It does that through finding substrings that appears frequently enough in several word forms, so that it can propose them as morphemes. This is called the Minimal Description Length (MDL) principle: Morfessor tries to minimize the total description length of unique morphemes to account for the training data.

Demberg's word segmentation model

Demberg's segmentation model is based on RePortS (Keshava and Pitler, 2006) but adds some extensions to it. RePortS uses words that appear as substring of other words and transition probabilities between letters in a word, to detect morpheme boundaries. RePortS assumes that root words do appear in the corpus, which may not be the case for all languages. Demberg's model adds to RePortS algorithm, an extension to fix this assumption by having an intermediate step which creates a candidate list of root words.

ParaMor

Segmentation in ParaMor is carried out by identifying the morpheme boundaries using letter transition probabilities, and then identifying morpheme-internal bigrams or trigrams. ParaMor then discovers the relationship between pairs of words. Finally, it uses an information-theoretic approach to minimize the number of letters in the morphemes of the language.

MADA

MADA (Morphological Analysis and Disambiguation for Arabic) (Habash, 2007) is the state-of-the-art manually-built morphological analysis system of the Arabic language. Along with word segmentation, MADA is an excellent word-in-context analyzer, and therefore provides accurate segmentation of a word in its context in a sentence. MADA has a high accuracy of usually over 94%. TOKAN, a component of MADA, allows a user to specify the tokenization (or segmentation) scheme. Each scheme has its own characteristics. This work uses two of the schemes: D1 and D2; D1 is a less aggressive in segmentation than D2, that is, D1 produces less overall segments than D2, on the average.

2.2 Statistical Machine Translation

Machine Translation is the task of automatically converting a text from one language to another. Statistical Machine Translation uses statistics from a parallel corpus to build a statistical model of translation.

An SMT model for Arabic and English is created through the following steps:

1. An Arabic-English parallel corpus (i.e., Arabic sentences and their aligned English translations) is given as input to the SMT learner which produces a corresponding SMT model.
2. The resulting SMT model is then used to translate Arabic into English with an SMT decoder.

Table 1 illustrates the matching alignment between Arabic and English sentences in the table below. Notice here that some English words correspond to only a morpheme (substring) in Arabic words. So we can see that word segmentation could be useful for Arabic to English translation.

English	The boy is playing with the ball	The boy is play+ing with the ball
Arabic	يلعب الولد بالكرة	يـ+لعب الـ+ولد بـ+ـالـ+كرة

Figure 1: Example of a sentence translated from Arabic to English. The matching substrings are highlighted with the same color.

In this research, we use the MOSES toolkit (Koehn et al., 2007), an SMT tool that allows a user to build an SMT system for any pair of languages using a parallel corpus.

3. Methodology

We now describe the method in which we perform the unsupervised segmentation learning task, the core of this research. We then describe how to carry out the machine translation task. Finally, we explain how we couple word segmentation task with SMT.

3.1 Data

In this work, we used two sets of data:

Set 1: A list of 1.7 million unique and punctuation-free words extracted from a corpus of 400 million words. These then were transliterated to Buckwalter transliteration for processing purposes (Buckwalter, 2004).

Set 2: An Arabic-English parallel corpus of 120,000 sentences, of which 119,000 were used for SMT training, and a 1,000 for SMT testing.

3.2 The Segmentation Task

For each of the unsupervised word segmentation systems, we have two phases:

1. **Training:** We input a list of unique Arabic words, each word on line without annotation, into the learner. We get a segmentation model after this step. (Figure 2, step 1)
2. **Testing:** We use the resulting segmentation model from the first phase and use it to segment a smaller Arabic word list, again each word in a line. (Figure 2, step 2)

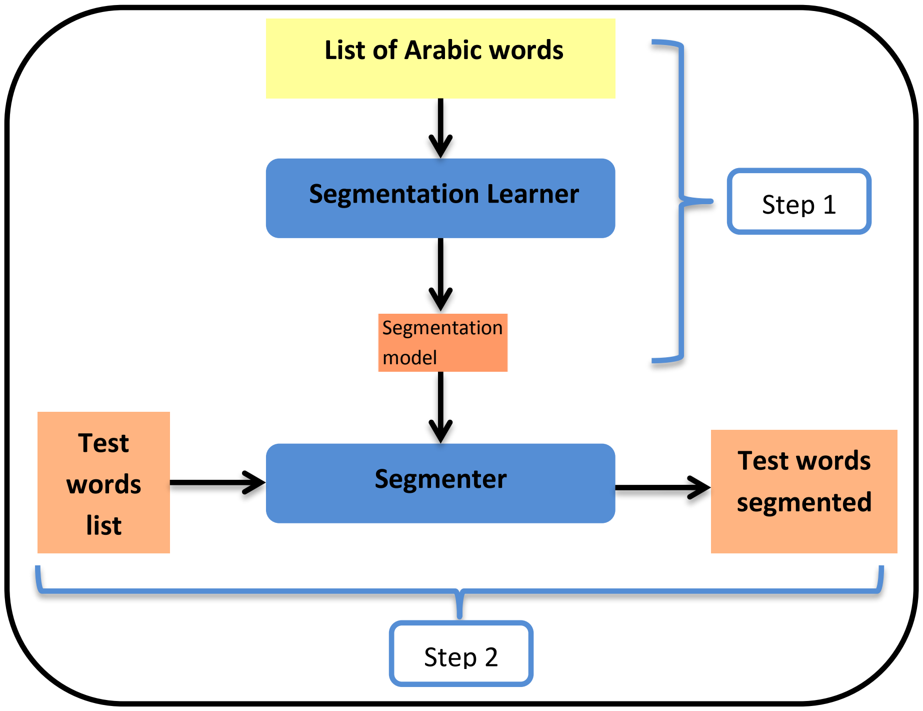

Figure 2: Unsupervised word segmentation

3.3 The Translation Task

Figure 3 shows the block diagram of the SMT data flow. We explain the diagram in three steps:

1. We run the Arabic side corpus through a segmenter and replace it with the original Arabic corpus, while keeping the English unsegmented, and input this modified parallel corpus into the SMT learner which produced an SMT model.
2. We run Arabic test corpus that we wish to translate through the same segmenter used in step-1. Now er run the segmented Arabic test set through the SMT decoder to get the English translation.
3. We compute the translation accuracy through running BLEU on translation comparing with gold-standard translations.

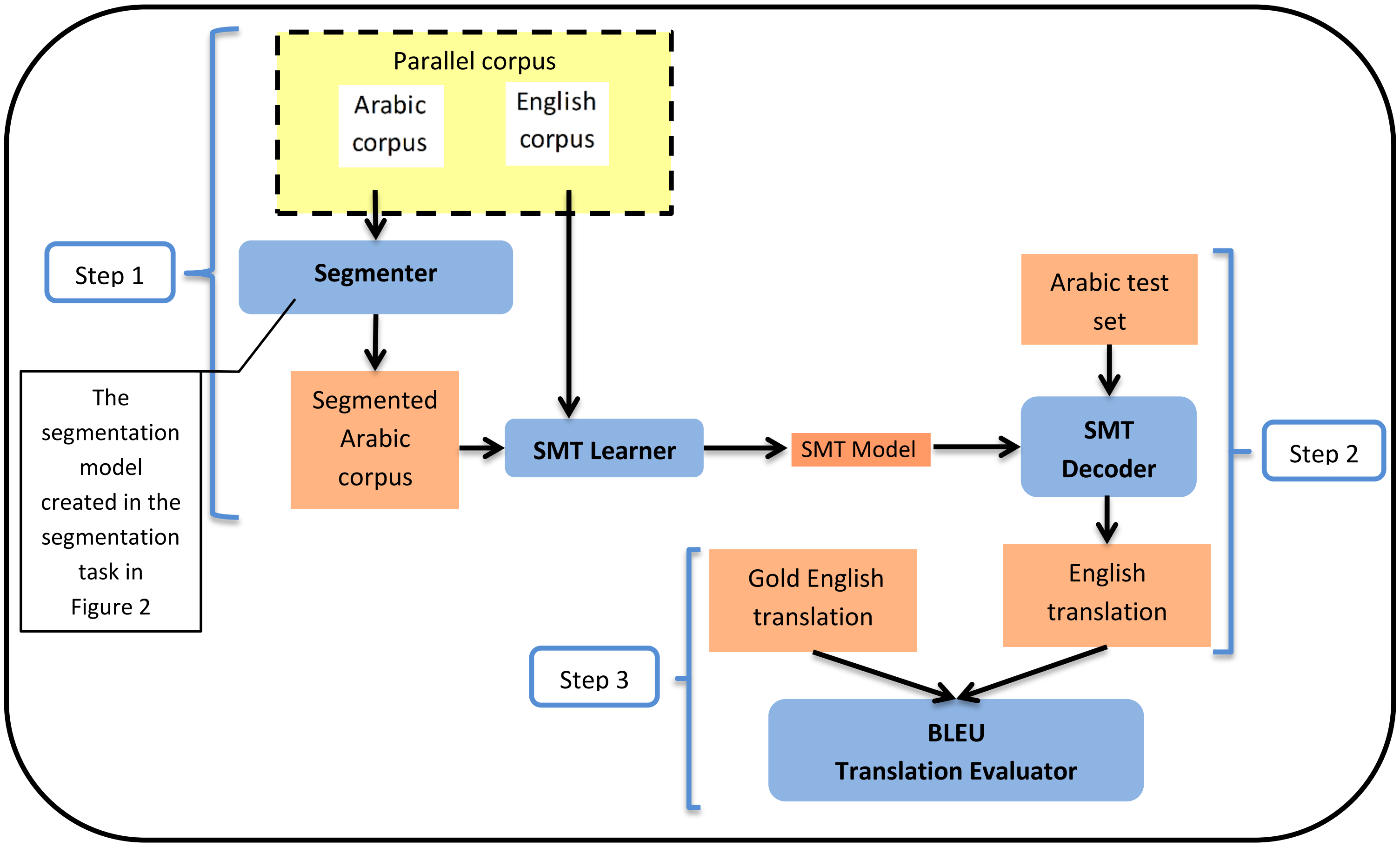

Figure 3: SMT methodology. Note that the "Segmentation Model" is created by the Segmentation task.

4 Evaluation

We evaluate both the accuracy of segmentation intrinsically and then evaluate the impact of different segmentation schemes on SMT.

4.1 Evaluation of Word Segmentation

The accuracy of a segmentation system is computed in the following way:

$$Accuracy = \frac{number\ of\ correctly\ segmented\ test\ words}{total\ number\ of\ test\ words}$$

where the number of the correctly segmented words is calculated either manually or by comparing it against MADA.

We run the following segmentation experiments:

1. **10-fold experiment**: We use a list of unique words of size 1,700,000 from which we create 10 experiments. In each experiment (or fold) the training set is 9 times the size of the test. We evaluate the correctness of segmentation by comparing it against MADA's segmentation.
2. **200 words test**: We compute the segmentation accuracy of 200 words output by each of the unsupervised systems and compare them against (1) MADA's segmentation and (2) manual segmentation.
3. **100 words test:** We take 100 words from the parallel corpus that is later to be translated and we evaluate the segmentation accuracy manually.

4.2 Evaluation of Statistical Machine Translation

One of the most common metrics to evaluate machine translation is through Bilingual Evaluation Understudy (BLEU) (Papineni et al., 2002). BLEU evaluates a translation by matching n-grams between a translation and a gold standard translation. Thus BLEU not only evaluates the accuracy of the words in the translation, but also evaluates the order of the words, quantifying the fluency of a translation. BLEU also allows for multiple human translation references as standard. In this research, we use four correct translation references to evaluate translation with BLEU.

5. Experiments and results

In Table 2, we present the results obtained for all the experiments. As we can see, Morfessor produces the best segmentation in two of the experiments, while ParaMor surpasses Morfessor in two of the experiments. Demberg's system overall has lower accuracy. Notice here that in the test of 200 words, once against MADA and once against manual segmentation, the accuracy does not match because although MADA is accurate, it does not cover all segmentation cases.

System	Morfessor	ParaMor	Demberg
10-fold vs. MADA	25.88%	**32.97%**	27.20%
200 words vs. MADA	**49.00%**	47.00%	31.00%
200 words vs. Gold	48.00%	**65.00%**	47.00%
100 words vs. Gold	**66.00%**	24.00%	37.00%

Table 2: Accuracy of the unsupervised segmentation systems for each experiment.

For the translation task, we use BLEU to evaluate the translation accuracy and fluency. In Table 3, we report the BLEU translation score for each system. Note that the baseline score refers to SMT model without using word segmentation. Also note that we have two scores for MADA: D1 and D2 due to using two different schemes for segmentation, where D2 is a more aggressive segmentation than D1.

	Baseline	MADA-D2	MADA-D1	Morfessor	ParaMor	Demberg	Morfessor+
BLEU	41.31%	36.87%	**43.78%**	38.29%	20.89%	36.73%	41.17%

Table 3: BLEU scores for the word baseline and for all the segmentation systems used.

We notice that amongst the three unsupervised systems, Morfessor is performing the best in translation. Although ParaMor performs better than Morfessor in word segmentation task, Morfessor outperforms ParaMor in translation. We claim that this is because although ParaMor has a better segmentation accuracy, it segments the words aggressively. As we can see from the Table 4, the number of unique segments that ParaMor produces is much higher than what Morfessor produces.

System	Morfessor	ParaMor	Demberg
Unique morphemes of words used in the translation evaluation for 7954 unique words	4,280	6,618	6,615

Table 4: Number of unique morphemes obtained by each segmentation system

As Morfessor is the best unsupervised segmentation system (Table 3), we now created a modified version, Morfessor+, a post-processing modification of Morfessor, where we try to make the segmentation less aggressive. We added three simple rules: attach “A” (Alef equivalent in Buckwalter) at the beginning of a word, attach “Al” (Alef-Lam equivalent in Buckwalter) at the beginning of a word, and remove segmentation from any two letter words. We see an improvement in translation from Morfessor to Morfessor+. But nevertheless, none of the systems proposed beat the baseline and MADA-D1.

6. Conclusions

We conclude that accurate manually-built word segmentation does improve translation (as the case for MADA-D1), especially while keeping word segmentation is balanced. However, even manually-built word segmentation may not improve translation, if segmentation was aggressive. As we see MADA-D2 has a lower BLEU compared to the baseline. The usefulness of balanced word segmentation in SMT also applies to the unsupervised systems. We have seen that even if segmentation is more accurate (in the case of ParaMor), it performs poorly when coupled with translation, and the more balanced the segmentation is (in the case of Morfessor), the better the translation score obtained. We also see that lowering the number of segmentation in Morfessor generates a better SMT (the case of Morfessor+).

We also see potential of unsupervised word segmentation to improve when post-processing is applied (as in the case form Morfessor to Morfessor+), and is very close to outperform the baseline. Therefore we propose that semi-supervised word segmentation has more potential to improve machine translation in SMT.

7. References

C. Mathias and K. Lagus. 2005b. Morfessor in the Morpho Challenge. In Mikko Kurimo, Mathias Creutz, and Krista Lagus, editors, Unsupervised segmentation of words into morphemes – Challenge 2005, pages 12–17, Helsinki University of Technology, Helsinki.

V. Demberg. 2007. A language independent unsupervised model for morphological segmentation. In Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics, pages 920–927, Prague.

S. Keshava and E. Pitler. 2006. A simpler, intuitive approach to morpheme induction. In Proceedings of 2nd Pascal Challenges Workshop, pages 31–35, Venice, Italy.

C. Monson. 2009. ParaMor: From Paradigm Structure to Natural Language Morphology Induction. Ph.D. thesis, Carnegie Mellon University.

P. Koehn, H. Hoang, A. Birch, C. Callison-Burch, M. Federico, N. Bertoldi, B. Cowan, W. Shen, C. Moran, R. Zens, C. Dyer, O. Bojar, A. Constantin, E. Herbst, Moses: Open Source Toolkit for Statistical Machine Translation, Annual Meeting of the Association for Computational Linguistics (ACL), demonstration session, Prague, Czech Republic, June 2007.

R. Roth, O. Rambow, N. Habash, M. Diab, and C. Rudin. Arabic morphological tagging, diacritization, and lemmatization using lexeme models and feature ranking. In Proceedings of Association for Computational Linguistics (ACL), Columbus, Ohio, 2008.

K. Papineni, S. Roukos, T. Ward, and W. Zhu. 2002. BLEU: a method for automatic evaluation of machine translation. In Proceedings of ACL, pages 311–318, Philadelphia, PA.

T. Buckwalter. 2004. Buckwalter Arabic Morphological Analyzer Version 2.0. Linguistic Data Consortium (LDC2004L02).