A Service Localisation Platform

Luke Collins and Claus Pahl
School of Computing
Dublin City University
Dublin 9, Ireland
Email: luke.collins4@mail.dcu.ie, claus.pahl@dcu.ie

***Abstract*—The fundamental purpose of service-oriented computing is the ability to quickly provide software and hardware resources to global users. The main aim of service localisation is to provide a method for facilitating the internationalisation and localisation of software services by allowing them to be adapted to different locales. We address lingual localisation by providing a service translation using the latest web services technology to adapt services to different languages and currency conversion by using real-time data provided by the European Central Bank. Units and Regulatory Localisations are performed by a conversion mapping, which we have generated for a subset of locales. The aim is to investigate a standardised view on the localisation of services by using runtime and middleware services to deploy a localisation implementation. Our contribution is a localisation platform consisting of a conceptual model classifying localisation concerns and the definition of a number of specific platform services.**

Keywords - Service Localisation; Service-oriented Computing; Service-oriented Architecture.

I. Introduction

Distributed web services can provide business and private consumers with computing abilities which may not be feasible for them to develop in-house. These web services are currently in high demand in the context of cloud computing [3], [19]. However, the area of services computing introduces new issues, for example, in areas like Europe, where there is a wide range of languages spoken, services are very often only developed for single language and are only supported for that single language. Often it is the case that companies do not have the resources or capability to develop multilingual products. Localisation encapsulates a large number of issues which need to be addressed. These include, but are not limited to:

- Language Translation - conversion of services based on language. e.g., $English \rightarrow French$.
- Regulatory Compliance Constraints - conversion of services based on information such as taxation and other regulatory constraints.
- Currency Conversion - conversion of services based on currency, e.g., $Euro \rightarrow Dollar$.
- Units Conversion - based on standard units measurements, e.g., $Metric \rightarrow Imperial$.

Further concerns such as standardised vocabularies and conventions could be added.

Localisation is typically performed on textual content (i.e., strings) and refers to either languages only or physical location. However, the purpose of this work is to develop a method of localising services by implementing a 'mediator' type service which interacts between the Application Programming Interfaces (APIs) of the service provider and the requester. We are going to focus on a number of locale dimensions such as language, taxation, currency and units. An example of a request which requires localisation can be seen in Figure 1, which illustrates an example of a financial service provided to a range of locales (locations or regions requiring equal conversions).

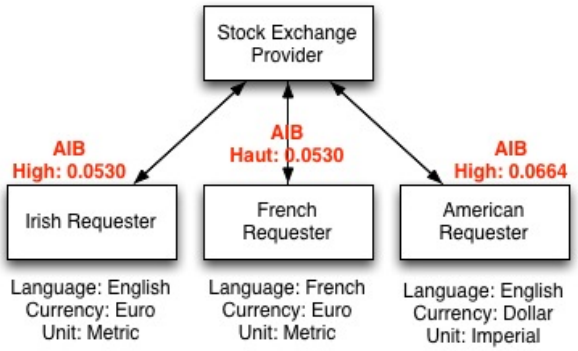

Fig. 1: Overview of Requests Requiring Localisation

We aim to provide service-level language translation techniques to localise services (including API interfaces) to different languages. Regulatory translation which includes currency, units and taxation among other legal governance and compliance rules will be provided by standards-based mappings. Regulatory translation is important for applications to comply with varying regional laws and regulations.

The objectives of service localisation include primarily the introduction of service-centric localisation techniques. A specific need is to make localisation techniques available at runtime for dynamic localisation, which is required for currencies and other variable aspects. Thus, Service Localisation (SL) provides a mechanism for converting and adapting various digital resources and services to the locale of the requester. A greater end-to-end personalisation of service offerings is an aim. A Localisation Provider act as an intermediary between the service provider and the requester. In our proposed platform, this is supported by a mediation service. By generating a common platform solution for these localisation issues, we allow the ability to dynamically localise Web services to be made with little effort. Our key contributions are:

- Software Localisation at Service Level - the main concern is a standardised mapping within a potentially

heterogeneous environment.

- Adaptation and Integration - the main concern is the maintenance of service quality after it has been localised through adaptation.

The novelty of the proposed solution lies in filling a gap between service adaptation techniques (largely ignoring the regulatory and lingual aspects) and service internationalisation, which looks into localisation concerns, but only to a basic extent covering data formats and unit and currency conversions. We aim to show through a concrete example an appropriate use of Service Localisation. The example also attempts to illustrate various benefits and use cases. We also discuss motivating factors behind using a localisation technique.

In the next section, we discuss the motivation behind developing a Service Localisation implementation. Section 3 defines a platform architecture for Service Localisation. In Section 4, we introduce aspect-specific localisation techniques which we investigated and implemented. Section 5 introduces the implementation and evaluates our solution to the Service Localisation problem. Section 6 contains the related work discussion. In Section 7, future directions and possible extended infrastructures are explored. We end with related work and concluding comments.

II. Motivation

Our main focus is a platform for service localisation, which makes a shift from the typical "one size fits all" scenario towards a more end-to-end personalised service scenario. Currently, services computing suffers from localisation and adaptability issues for multiple users in different regions. These issues could be overcome if a multi-lingual and multi-regional solution was developed [15], [22]. The different localisation issues of a service can be illustrated. The scenarios described below are used to illustrate benefits to service localisation:

- End-User Services: Some software-as-a-Service providers only support one region with one specific language. There is a possibility to perform localisation both statically (compile-time) and dynamically (run-time), which typically involves localising service values and interacting messages.
- Business Services: Various business centric applications including applications for documentation and analysis could be localised to support various legal and regional locales. Business services typically require more customisation than end-user consumers.
- Public Sector Services: As governments outsource their computing infrastructure to external providers, it is becoming more important for the providers to supply solutions which take into account various regulatory governance aspects such as currency and taxation and also lingual localisation.

Another scenario which provides a detailed view of the benefits of service localisation could be a service provider, used to manage company accounts for its customers. This could be a company which has offices in different global locations and would like to provide localisation based on customer region and localisation for its individual offices.

- Regulatory: Conversion of data between standards and their variants, e.g., based on different units of measurement $Metric \rightarrow Imperial$.
- Currency: Conversion of between currencies, e.g., $Euro \rightarrow Dollar$.
- Lingual: Translation of service related data between languages. This could include free text, but also specific vocabularies based on business product and process standards such as GS1 or EANCOM.
- Taxation: Different customers have different taxation requirements, e.g., VAT rates. Localisation of accounts software can take this into account for each locale.

A. Use Cases and Requirements

In order to demonstrate the need for localisation of Web services, we chose to demonstrate the issue using a concrete case of an environment which utilises service-level access to a stock exchange interface. Imagine an Irish user who wishes to access data from the New York Stock Exchange, which is provided in an English format with the currency in dollars. A user in France may also wish to access data from the New York Stock Exchange using a French interface where local regulations require French to be used for data and/or service operations. Therefore, there must be a mechanism to convert the currency to Euro or to another currency which the requester specifies. There must also be a mechanism to convert the language to that of the requester.

At application level, two sample calls of a stock exchange data retrieval service for the two different locales (IE-locale with English as the language and EUR as the currency and FR-locale with French as the language and EUR as the currency) retrieve the average stock price for a particular sector - in this case the financial sector as follows:

- $Retrieve(20/08/2012, Financial) \rightarrow 30.50\,EUR$
- $Récupérer(20.08.2012, Financier) \rightarrow 30,50\,EUR$

In the US-locale with English as the language and USD as the currency, the same API call could be the following:

- $Retrieve(08/20/2012, Financial) \rightarrow 38.20\,USD$

The following elements in this case are localisable:

- Date: in order to preserve regulatory governance, the date format requires to be changed depending on the requester locale.
- Language: names of functions from the API are translated between languages.
- Currency: values are converted as normal and this would apply to any other units.

This list can vary depending on the environment where different regulatory constraints might apply. In general, it can be expected that there is always a linguistic element to the localisation of any product, but elements may also include taxation and units of measurement. If it was the case that the requesters were trying to access weather forecasts for their

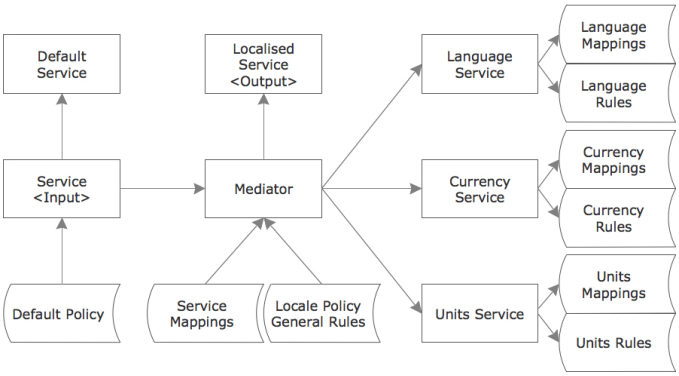

Fig. 2: Architecture of the Localisation Platform.

own region and formatted in their own locale, then it would be necessary to utilise a conversion for units of measurement:

- $Prévision(20.08.2012) \rightarrow 30°Celsius$
- $Forecast(20/08/2012) \rightarrow 15°Celsius$

In the US-locale with English and imperial units, the same API call could be $Forecast(08/20/2012) \rightarrow 87°Fahrenheit$.

III. Platform Architecture

Localisation of services requires a framework to be implemented to facilitate various localisation methods. These various methods, implemented as services in our proposed localisation platform, are used to facilitate the localisation of localisable elements or artefacts. This paper focuses on the dynamic localisation of service level descriptions.

With every service there are various elements which may be localised. These elements include:

- Service specifications/descriptions (APIs)
- Models (structural/behavioural)
- Documentation (for human consumption)
- Messages exchanged between services

Services are normally written to be independent of locales, however localisation is often needed. A localisation platform should be based on attributes which vary from locale to locale, like time or date format.

A service localisation platform requires a number of elements. These elements can be pre-translated fragments in static form or can be dynamic translation systems. Figure 2 aims to demonstrate the concept of a policy and mappings based system, which can be scaled when additional processes are attached to the mediation process. In the platform architecture, user-specific locale policies are applied to service endpoints. For example, in a WSDL file we may localise messages and operation names. Rules for each type of translation would be stored in a rules database (General Rules Repository). Similarly, mappings between common translations would be stored in a mappings database (Translation Memory).

A mediator operates between users (with different locales) and several service providers (with different locales) by providing core localisation services, such as currency conversion and language translation. The architecture supports the following:

- Static Mappings: these could be the mapping of one language to another or one unit to another, pre-translated in translation memories.
- Dynamic Localisation: when translation mappings are not stored, dynamic localisation is required in order to obtain a correct translation and store the mapping.
- Policy Configuration: in order to configure the various locale policies, we must generate particular translation rules, supported by a logical reasoning component.
- Negotiation: this is the exchange of locale policies through the form of XML and SOAP from a web services point of view.
- Localisation of Services: the mappings between the remote service and the localised service description must be stored in a mappings database (Translation Memory) so the localised service has a direct relationship with the remote service.

The workflow is concequently $Negotiation \rightarrow PolicyConfiguration \rightarrow Localisation \rightarrow Execution$.

Some examples shall illustrate the functionality of the platform. Table I defines two different locales in XML profiles. A mismatch between the requester locale and the provider locale needs to be bridged by the mediator localisation service. The language as a lingual aspect and country, currency and unit codes are regulatory concerns.

TABLE I: Sample Environment Setup

```
<SLContext>
  <Locales>

    <RequesterLocale>
      <LanguageCode>efr</LanguageCode>
      <CountryCode>FR</CountryCode>
      <CurrencyCode>EUR</CurrencyCode>
      <UnitCode>M</UnitCode>
    </RequesterLocale>

    <ProviderLocale>
      <LanguageCode>en</LanguageCode>
      <CountryCode>IE</CountryCode>
      <CurrencyCode>EUR</CurrencyCode>
      <UnitCode>M</UnitCode>
    </ProviderLocale>

  </Locales>
</SLContext>
```

The locale definitions decide how a given service API (in WSDL) is localised. Results from a sample execution of the localisation service (the mediator) is displayed in Tables II and III based on the XML locale definitions of the environment in Table I. Table II shows excerpts from an original WSDL file. Table III shows the localised WSDL after the application of lingual localisation in this case (translation from English (IE locale) into French (FR locale) – for simplicity of the example, we have focused on this single aspect only), compliant with the two locale definitions from the first listing.

TABLE II: Sample Input - Provider Locale

```
<wsdl:message name="quoteResponse">
  <wsdl:part name="parameters"
             element="quoteResponse"/>
</wsdl:message>
<wsdl:message name="quoteRequest">
  <wsdl:part name="parameters"
             element="quote"/>
</wsdl:message>
<wsdl:portType name="Quote">
  <wsdl:operation name="getQuote">
    <wsdl:input name="quoteRequest"
                message="quoteRequest"/>
    <wsdl:output name="quoteResponse"
                 message="quoteResponse"/>
  </wsdl:operation>
</wsdl:portType>
```

TABLE III: Sample Output - Localised WSDL

```
<wsdl:message name="quoteReponse">
  <wsdl:part name="parameters"
             element="quoteReponse"/>
</wsdl:message>
<wsdl:message name="citerDemande">
  <wsdl:part name="parameters"
             element="citer"/>
</wsdl:message>
<wsdl:portType name="Citer">
  <wsdl:operation name="getCiter">
    <wsdl:input name="citerDemande"
                message="citerRequest"/>
    <wsdl:output name="citerDemande"
                 message="citerDemande"/>
  </wsdl:operation>
</wsdl:portType>
```

IV. Localisation Platform – Rules and Services

We have outlined the core platform architecture in the previous section with the central services. In order to provide the localisation platform services, we need to implement a number of classes to enable a modular service localisation platform. Their interaction is summarised in Figure 3. Details of underlying concepts of their operation are explained now.

A. Rule-based Locale Definition and Conversion

At the core of our service localisation platform is a language to specify the rules in relation to localisations. In most cases, languages like WSDL and other XML languages provide information regarding the services that are provided via an API. However, in order to encapsulate localisation information, there is a necessity to provide a language which will contain details in relation to the locales of the requester and the provider. For the purpose of our localisation platform, we use a policy language based on the Semantic Web Rule Language SWRL, which is based on the propositional calculus.

A localisation layer encapsulates the various forms of translations. It describes the relationships between localisable elements. For example, it contains the details of items which can be translated. For our localisation model these are documentation and descriptions, but also API messages and operations. The rule language is used to define policies of two types: firstly, locale definitions and, secondly, conversion (translation) rules. We motivate the rule set through examples.

Firstly, there are a number of locale definition rules provided, like *Loc* or *hasCur*, by which locales for specific

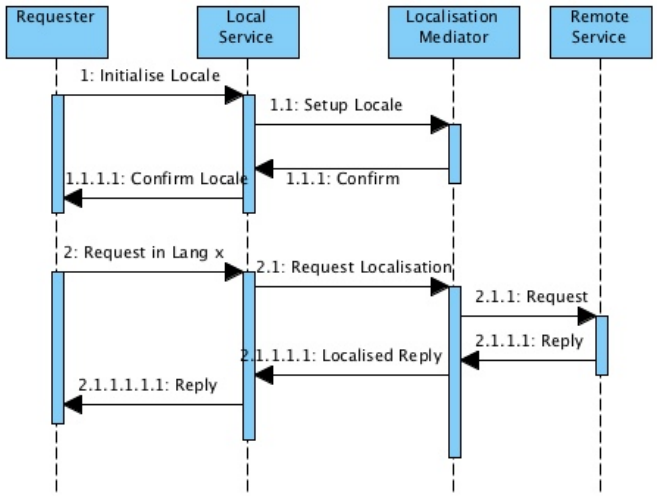

Fig. 3: A UML Sequence Diagram of the Platform.

regions are described. A locale can also be described by other rules such as *hasTax*, *hasLang* and *hasUnit*. Examples of three region's locales - IE, US, and FR - are:

$$IELoc(?l) \leftarrow Loc(?l) \wedge hasLang(?l,?z) \wedge hasCur(?l,?c) \wedge hasUnit(?l,?u) \wedge ?z = en \wedge ?c = EUR \wedge ?u = metric$$

$$USLoc(?l) \leftarrow Loc(?l) \wedge hasLang(?l,?z) \wedge hasCur(?l,?c) \wedge hasUnit(?l,?u) \wedge ?z = en \wedge ?c = USD \wedge ?u = imperial$$

$$FRLoc(?l) \leftarrow Loc(?l) \wedge hasLang(?l,?z) \wedge hasCur(?l,?c) \wedge hasUnit(?l,?u) \wedge :?z = fr \wedge ?c = EUR \wedge ?u = metric$$

The benefit of a formal framework for the rules is that other rules can be inferred by from partial information. For example, if we knew that a locale had USD as its currency we may be able to infer its country from it:

$$?c = USD \rightarrow ?l = USLocale.$$

These inferred rules do not apply in general - this may not work if we know a currency is Euro in which case it could be one of many locales in Europe. The purpose of these rules could be to determine inconsistencies, however. Preconditions can clarify the remit of these rules.

Secondly, a generalised conversion between locales, e.g., $Locale\ A \rightarrow Locale\ B$, is given by the following general conversion rule:

$$IELoc2USLoc(?l1,?l2) \leftarrow hasLang(?l1,?z1) \wedge hasLang(?l2,?z2) \wedge hasCur(?l1,?c1) \wedge hasCur(?l2,?c2) \wedge hasUnit(?l1,?u1) \wedge hasUnit(?l2,?u2) \wedge ?z2 = convertLang(en,en,?z1) \wedge$$

$?c2 = convertCur(EUR, USD, ?c1) \wedge$
$?u2 = convertCur(metric, imperial, ?u1)$

$IELoc2FRLoc(?l1, ?l2) \leftarrow$
$hasLang(?l1, ?z1) \wedge\ hasLang(?l2, ?z2) \wedge$
$hasCur(?l1, ?c1) \wedge\ hasCur(?l2, ?c2) \wedge$
$hasUnit(?l1, ?u1) \wedge\ hasUnit(?l2, ?u2) \wedge$
$?z2 = convertLang(en, fr, ?z1) \wedge$
$?c2 = convertCur(EUR, USD, ?c1) \wedge$
$?u2 = convertCur(metric, metric, ?u1)$

Depending on requester and provider locale any combination of mappings/translations can be generated by the core rules.

B. Localisation Mediator

Based on these local definition and conversion rules, a number of services operate. In order to provide a transparent localisation system, one class acts as a mediator, as visualised in Figure 4, which could use individual services for: Lingual Conversion, Currency Conversion, Regulatory Governance, Units Conversion, and WSDL Parsing & Generation. Within this mediator, implemented as a Java class, we have various methods which call the other localisation services of the platform.

During execution of the localisation platform, an XML file is first passed to the mediator. The Mediator Service then sets up a localisation environment using the locale details provided in *LocaleConfig.xml*, the class performs this via the use of Java Interfaces. Once the locale is set up, the service Web Service Description Language (WSDL) file is parsed and various elements are localised resulting in a localised WSDL file which can be used to access localised operation mappings. This class is the work horse of the platform and can be extended with the introduction of other localisation classes.

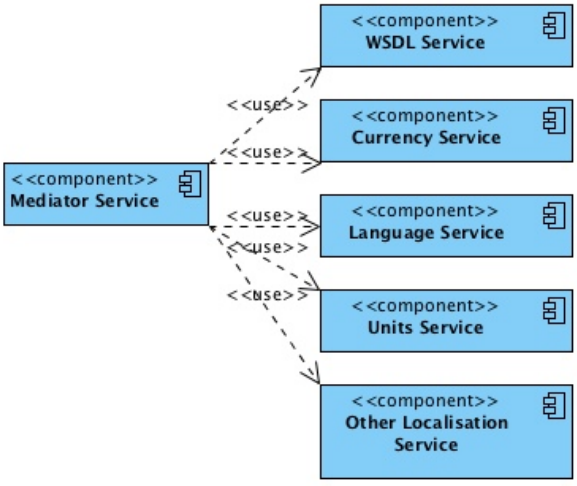

Fig. 4: A Component Diagram Displaying Extensibility.

Linguistic artefacts are one of the most broadly localised elements of software today. We propose machine translation (MT) to achieve automation. While further research into a tailored MT solution is required to specifically address limited textual context and controlled vocabularies for APIs, language translation within the proposed platform is provided by the Google Translate API. In the interest of performance, our platform tries to make as few API calls to Google as possible. Instead it stores translations of popular words and glossaries within a local language mapping database (Translation Memory) for later retrieval. A local machine translation system may also reduce this latency, as it would no longer have to depend on TCP/IP performance. The conversion rule for language translation is given by:

$IELoc2FRLoc(?l1, ?l2) \leftarrow$
$hasLang(?l1, ?z1) \wedge\ \ hasLang(?l2, ?z2) \wedge$
$?z2 = convertLang(en, fr, ?z1)$

Regulatory localisation through adaptation to other regulatory standards is based on localising regulatory concerns. These concerns include, but are not limited to the following: Taxation, Currency, and Units of Measurement. We have chosen to localise a subset of these concerns. For the purpose of units localisation, we developed an interface to a repository of unit conversion formulae. These formulae provided conversions between the metric and imperial units of measure. The conversion rule for units is given by:

$IELoc2USLoc(?l1, ?l2) \leftarrow$
$hasUnit(?l1, ?u1) \wedge\ hasUnit(?l2, ?u2)$
$\wedge\ ?c2 = convertUnits(metric, imperial, ?u1)$

Due to a large number of currencies used globally, it was necessary to develop a class which dealt with currency conversion. For the purpose of currency localisation, we use exchange rates from the European Central Bank. This is in our case supported by a MySQL database. Currencies are manipulated based on their rate compared to Euro as the base currency. The conversion rule for currency is given by:

$IELoc2USLoc(?l1, ?l2) \leftarrow$
$hasCur(?l1, ?c1) \wedge\ hasCur(?l2, ?c2)$
$\wedge\ ?c2 = convertCur(EUR, USD, ?c1)$

In order to parse the input in the form of WSDL files, a WDSL service class is used. This class contains the methods required to manipulate both incoming WSDL files of the service provider and has the ability to generate a localised WSDL file. The class can be considered as an I/O Manager. XLIFF is an XML standard for translation that proved useful when it comes to the localisation of WSDL file.

V. Implementation and Evaluation

The localisation platform presented here was fully implemented in a Java prototype that aims at studying the feasibility of the conceptual solution. It shall be assessed on the following criteria here: Performance and Extensibility. These criteria have different effects on the end-user experience of the product. These criteria are key performance indicators (KPI) and critical success factors (CSF) of the localisation platform described.

Poor performance often tends to affect software exponentially as multiples of users consume a service at the same time. The core question here is the overhead created by adding localisation dynamically to service provisioning. Our results show an acceptable overhead of 10-15 % additional execution time for fully localised services (i.e., localisation involving different localisation aspects). The overhead is still low compared to network latency and the average service execution time [22]. As the application deals with multiple users, the latency would increase due to extra loads placed on the platforms services. This makes latency one of the KPIs of the project. Latency, is also an area to be assessed as adding the localisation platform to the workflow of an existing process has the potential to add lag-time. This lag-time exists due to time required to compute and also the time to initialise the various variables. The propagation latency is displayed in Table IV below. It should be noted that figures can be affected by environmental changes or the locale we are transforming from and the locale we are transforming to.

TABLE IV: Latency Table - Localisation of Service

Service	Prior (μs)	Post (μs)	Δt (μs)
NASDAQ	132	182	50
FTSE	110	152	42

As a general strategy, we have aimed to improve performance by using pre-translated aspects (through stored mappings). A related concern is scalability of software becomes more important when a service may have large multiples of users. Scalability has not been empirically addressed for this phase of research and will be evaluated in later prototypes.

- Some components of the platform would require modification to effectively allow the infrastructure to vertically scale-up or scale-out efficiently. Solutions here are stateless programming and data externalisation. Through our rule base, and the suggested pre-translation repositories some suitable architectural decision in this direction have already been made.
- Horizontal scalability - i.e., the addition of more localisation concerns - is conceptually easily supported by the modular mediator architecture, which we will address further below in the extensibility context from an implementation view.

An interesting model to investigate the scalability is a tuple space-based coordination approach [6], [7], [11], which would allow a flexible and elastic assignment of localisation services to multiple requests.

Extensibility becomes important when dealing with complete platforms like a localisation platform. During an initial development, it is often the case that features need to be included due to various constraints. In the case of the localisation platform described here, some localisation services where not developed, some of which include a service to handle taxation. However, the platform was designed to be extendable. At a platform level, this allows for the addition of further services and the support for more locales.

VI. Related Work

We provide a different view and perspective on the subject compared to other publications [9], [15], [18]. The area of localisation in its wider adaptivity and customisation sense has been worked on in various EU-supported research projects, such as SOA4ALL [17] and 4Caast [**?**]. These projects address end-user adaptation through the use of generic semantic models. Areas such as software coordination are also covered. The mOSAIC project adds multi-cloud provision to the discussion. Our framework however is modular and extensible and aims to provide a one-stop shop for all localisation methods.

The platform which is described here addresses the need for dynamic localisation of various artefacts by use of a translation memory and a set of logical rules. Software Localisation refers to human consumption of data which are produced by the software - namely messages and dialogues. Our focus is on the localisation of the service level. Service internationalisation is supported by the W3C Service Internationalisation activity [16], [20]. Adaptation and Integration of services based on locales and using a translation memory with rules and mappings is new [18]. The problem of multi-tenancy is a widespread issue in the area of cloud computing [22]. This is an area where a lot of research is being invested in order to provide a platform for different users with different business needs to be kept separate and their data to be kept private. Semantics involves the matching of services with various locales using mappings and rule-based system [2], [4], [9].

There are implementations which can perform localisation operations on web services [10]. The use of some of these however is restricted due to the nature of them. Some of the other implementations require a specific Integrated Development Environment or specific proprietary libraries. They also typically enable localisation at compile time - the proposed implementation in this paper is to enable service localisation at run time. IBM has presented a static localisation solution suitable for web services using its WebSphere platform [10], which requires the WSDL files to be generated within the Integrated Development Environment prior to deployment. This differs from our proposed localisation platform as our solution aims to perform transformations between locales dynamically.

VII. Conclusion and Future Work

Service localisation falls into the service personalisation and adaptation context. There are particular engineering methods and tools which can be employed to allow services to be adapted to different locales. A Service Localisation implementation should allow for automatically adjusting and adapting services to the requesters' own locales. We have presented a modular implementation which can enable services to be introduced into emerging markets which have localisation issues. Localisation hence provides a mechanism to widen a service provider's target market by enabling multi-locale solutions. The easiest solution is for a service provider to provide a 'mediator' service which could act as middleware between a requester and the service provider.

By allowing services to be localised, we are enabling the provision of multi-locale services to create interoperable service ecosystems (such as clouds). Due to the nature of third-party services, it is more intuitive for service localisation

to be performed dynamically through the use of a mediator service. Service localisation thus enables higher availability of services through its use of innovative interfacing. This type of localisation would be value-add for a company which may not have the resources to perform localisation in-house.

The objectives of Service Localisation have been presented in two forms. Firstly, presented was a conceptual framework which demonstrated key motivational reasons for developing a multi-locale support framework. The second part presented a modular platform, which is extensible to allow the support of further localisable artefacts. The platform which was implemented was using Java libraries was discussed as this programming solution copes well with the problem of extensibility.

The proposed service localisation fills a gap. Software adaptation has looked into adapting for instances services in terms of their user's interface needs such as data types and formats. The two focal localisation concerns lingual and regulatory add new perspectives to this area of research. A different activity is the Web services internationalisation effort, which looks into basic localisation concerns such as units, currency or the format of dates. Our localisation solution includes these (as we have demonstrated with the currency aspect), but expands these into a comprehensive framework.

The context of adaptation and translations/mappings used to facilitate this is a broad field. Our aim here was to integrate difference concerns into a coherent localisation framework. This relies on individual mappings. As part of our future work, we aim to add a semantic layer, which would support to concerns. Firstly, it would allow more reliable translations for non-trivial concerns if overarching ontologies were present. Secondly, the different concerns themselves could be integrated by determining interdependencies. Another direction of future research is to look into composition and, specifically, the behaviour of individual service localisation in for instance service orchestrations or other coordination models (e.g., tuple spaces as suggested above).

REFERENCES

[1] 4CaaSt. "Building the PaaS Cloud of the Future". *EU FP7 Project*. http://4caast.morfeo-project.org/. 2013.

[2] D. Anastasiou. "The impact of localisation on semantic web standards." *European Journal of ePractice*, 12:42–52. 2011.

[3] M. Armbrust, A. Fox, R. Griffith, A. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin and I. Stoica. "A view of cloud computing." *Communications of the ACM*, 53(4):50–58. 2010.

[4] K.Y. Bandara, M.X. Wang and C. Pahl. "Dynamic integration of context model constraints in web service processes." *International Software Engineering Conference SE'2009*. IASTED. 2009.

[5] K. Chen and W. Zheng. "Cloud computing: System instances and current research." Second International Conference on Future Networks, 2010. ICFN '10, pp. 88–92. 2010.

[6] G. Creaner and C. Pahl. "Flexible Coordination Techniques for Dynamic Cloud Service Collaboration." In Proceedings Workshop on Adaptive Services for the Future Internet WAS4FI. ServiceWave 2011. 2011.

[7] E.-E. Doberkat,W. Hasselbring, W. Franke, U. Lammers, U. Gutenbeil, and C. Pahl. "ProSet - a language for prototyping with sets." In International Workshop on Rapid System Prototyping 1992. pp. 235-248. IEEE, 1992.

[8] P. Fingar. "Cloud computing and the promise of on-demand business inovation." *InformationWeek*, July 13, 2009.

[9] K. Fujii and T. Suda. "Semantics-based context-aware dynamic service composition." ACM Transactions on Autonomous and Adaptive Systems (TAAS), 4(2):12. 2009.

[10] IBM. "IBM Developer Technical Journal: Developing internationalized Web services with WebSphere Business Integration Server Foundation V5.1." 2010.

[11] C. Pahl. "Dynamic adaptive service architecturetowards coordinated service composition." In European Conference on Software Architecture ECSA'2010. pp. 472-475. Springer LNCS. 2010.

[12] C. Pahl. "Layered Ontological Modelling for Web Service-oriented Model-Driven Architecture." *European Conference on Model-Driven Architecture - Foundations and Applications ECMDA'05*. Springer. 2005.

[13] C. Pahl, S. Giesecke and W. Hasselbring. "An Ontology-based Approach for Modelling Architectural Styles." European Conference on Software Architecture ECSA'2007. Springer. 2007.

[14] C. Pahl, S. Giesecke and W. Hasselbring. "Ontology-based Modelling of Architectural Styles." Information and Software Technology. 1(12): 1739-1749. 2009.

[15] C. Pahl. "Cloud Service Localisation." European Conference on Service-Oriented and Cloud Computing ESOCC 2012. Springer. 2012.

[16] A. Phillips. "Web Services and Internationalization." Whitepaper. 2005.

[17] SOA4All. "Service Oriented Architectures for All". *EU FP7 Project*. http://www.soa4all.eu/. 2012.

[18] H. Truong and S. Dustdar. "A survey on context-aware web service systems." Intl Journal of Web Information Systems, 5(1):5–31. 2009.

[19] W. Voorsluys, J. Broberg and R. Buyya. "Cloud Computing: Principles and Paradigms." John Wiley and Sons. 2011.

[20] W3C. "Web Services Internationalization Usage Scenarios." W3C. 2005.

[21] M.X. Wang, K.Y. Bandara and C. Pahl. "Integrated constraint violation handling for dynamic service composition." IEEE Intl Conf on Services Computing. 2009. pp. 168-175. 2009.

[22] M.X. Wang, K.Y. Bandara and C. Pahl. "Process as a service distributed multi-tenant policy-based process runtime governance." International Conference on Services Computing (SCC), pp. 578–585. IEEE. 2010.

[23] H. Weigand, W. van den Heuvel and M. Hiel. "Rule-based service composition and service-oriented business rule management." Proceedings of the International Workshop on Regulations Modelling and Deployment (ReMoD'08), pp. 1–12. 2008.