A Robust Client-Driven Distributed Service Localisation Architecture

Luke Collins, Ming-Xue Wang, Lei Xu and Claus Pahl
School of Computing
Dublin City University
Dublin 9, Ireland
Email: luke.collins4@mail.dcu.ie, mwang@computing.dcu.ie, lxu@computing.dcu.ie, claus.pahl@dcu.ie

***Abstract*—The fundamental purpose of service-oriented computing is the ability to quickly provide software 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 interface translation using the latest web services technology to adapt services to different languages and currency conversion as an example of regulatory localisation 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. We apply traditional software localisation ideas to service interfaces. Our contribution is a localisation platform consisting of a conceptual model classifying localisation concerns and the definition of a number of specific platform services. The architecture in which this localisation technique is client-centric in a way that it allows the localisation to be controlled and managed by the client, ultimately providing more personalisation and trust. It also addresses robustness concerns by enabling a fault-tolerant architecture for third-party service localisation in a distributed setting.**

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], [21]. 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.Equally, adapting services to different regulatory environments with different legal systems, taxation and units in place is equally challenging. 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 in this context. 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. This mediator largely automates the service interface localisation process. 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).

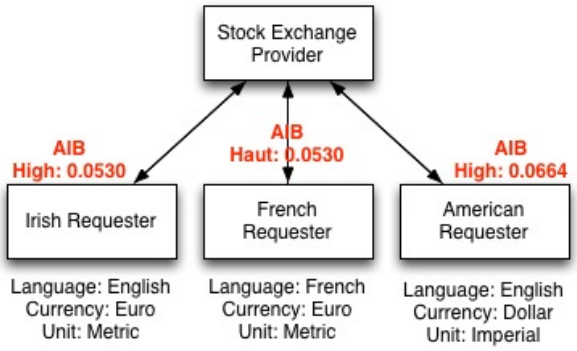

Fig. 1: Overview of Requests Requiring Localisation

We aim to provide service-level language translation techniques to localise services (including API interfaces) to different languages as part of a lingual translation idea. 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. A localisation mediator takes care of this task. 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. We will provide a novel architecture where in addition to the new concept of service interface localisation, this can even be controlled by the user at the client-side.

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.
- Client-side Control - the main concern is a robust, fault-tolerant coordination solution that allows localisation to be managed client-side.

The novelty of the proposed solution lies in filling a gap between service adaptation techniques (largely ignoring the regulatory and lingual aspects) and existing service internationalisation, which looks into localisation concerns, but only to a basic extent covering data formats and unit and currency conversions. An important aspect of this investigation is a robust coordination platform that not only allows service consumers to define and manage the localisation behaviour, this platform also needs to be able to address the challenges of services provided across a distributed setting with failure and non-applicability of localisation policies as a consequence. 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. In Section 5, we investigate the coordination solution for the client-side management of the localisation settings. Section 6 introduces the implementation and evaluates our solution to the Service Localisation problem. Section 7 contains the related work discussion. In Section 8, future directions and possible extended infrastructures are explored.

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 [17], [24].

A. Motivating Scenarios

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.

B. 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 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. Localisation Framework

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

A. Information Architecture

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)

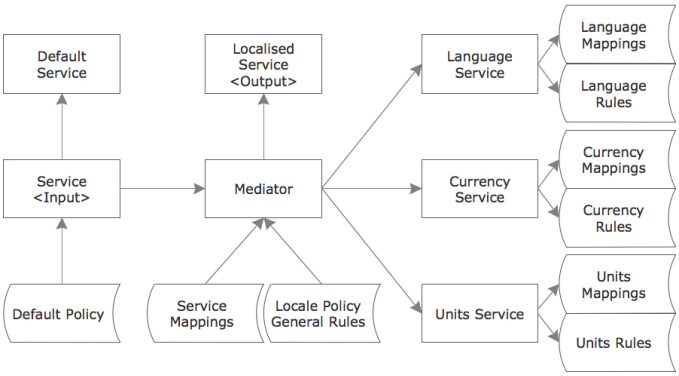

Fig. 2: Conceptual Architecture of the Localisation Platform.

- Messages exchanged between services

Services are normally written to be independent of locales, however localisation is often needed to further personalised or adapt a service to specific contexts. A localisation platform should be based on attributes which vary from locale to locale, like time or date format. Language is also an attribute which can be localised.

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 architecture, 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). Note, that we will discuss the distribution and management of services between client and provider in Section V.

B. Systems Architecture

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 of the mediator process 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 the format of 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 the Web service description language 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>
```

IV. Localisation Rules and Services

We have outlined the core platform architecture in the previous section with the central services. In order to provide

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>
```

the localisation platform services, we need to realise a number of localisation services to enable a modular service localisation platform. Their interaction is summarised in Figure 3. Details of underlying concepts of their operation are explained now. We will discuss the topology, i.e. where the individual services are provides and who manages them, behind this interaction specification in the following Section V.

A. Rule-based Locale Definition and Conversion

At the core of our service localisation platform is a language to specify the localisation policy 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.

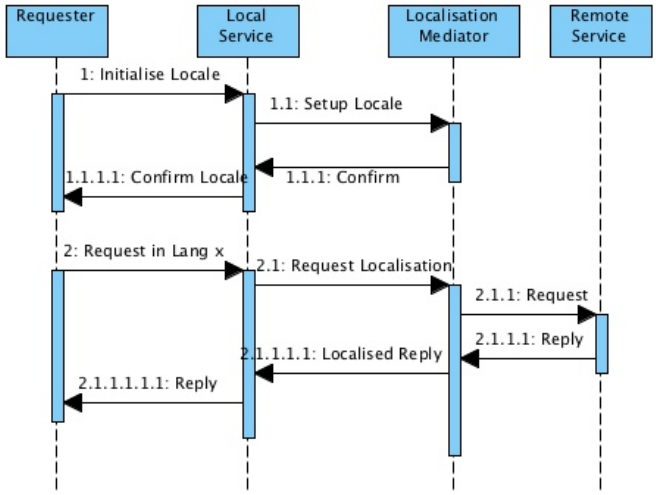

Fig. 3: A UML Sequence Diagram of the Platform.

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 localisation 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 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 service client and provider locale, any combination of mappings/translations can be generated by the core rules.

B. Localisation Mediator

Based on these locale policy definitions and conversion rules, a number of services operate. In order to provide a transparent localisation system, a central component acts as a mediator, as visualised in Figure 4, which in turn uses individual services for: Lingual Conversion, Currency Conversion, Regulatory Governance, Units Conversion, and WSDL Parsing & Generation. Within this mediator architecture, the mediator methods call the other localisation services of the platform.

During execution of the localisation platform, an XML filewith the ploicy definition is first passed to the mediator (we will look at the underlying coordination mechanism for this in Section V). The Mediator Service then sets up a localisation environment using the locale details provided in *LocaleConfig.xml*, the component performs this via the use of the respective 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 component is the work horse of the platform and can be extended with the introduction of other localisation classes, i.e. the architecture is modular.

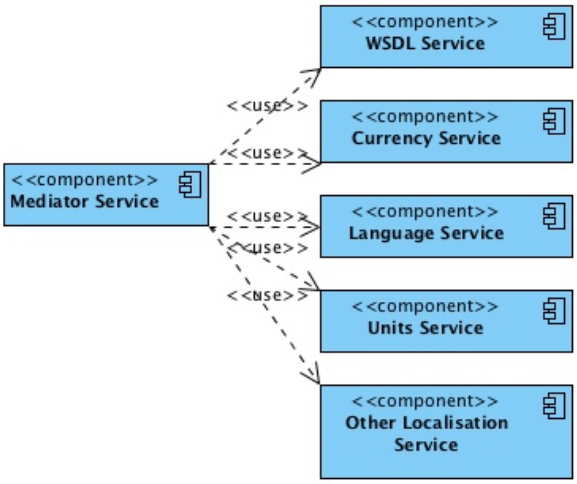

Fig. 4: A Component Diagram Displaying Extensibility.

Linguistic artefacts are one of the most widely 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 (a 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:

$$IELoc2FRLoc(?l1, ?l2) \leftarrow \\ hasLang(?l1, ?z1) \wedge \quad hasLang(?l2, ?z2) \wedge$$

$?z2 = convertLang(en, fr, ?z1)$

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 provide 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, we propose a separate service to deal 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 is used. This 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 service 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. Localistion - Coordination and Instrumentation

Figures 2 and 3 have defined the system architecture in abstract terms. A key feature of our solution is the possibility for clients to define the localisation constraints and policies and to manage the localisation themselves to achieve a higher degree of dynamic personalisation. In Figure 3, the Local Service is a client-side localised facade to the actual basic service as provided server-side. The mediator handles the required location as discussed in the previous Section IV. In order to manage the client-side definition and enforcement of localisation policies, a coordination framework is necessary, which is described in this section. For both the mediator and the server side, we assume BPEL process engines to manage the processes, like the mediator process $Negotiation \rightarrow PolicyConfiguration \rightarrow Localisation \rightarrow Execution$ that we introduced earlier. These generic processes and their constituent services need to be adapted to the needs specific localisation policies.

A coordination framework for localisation with protocols as the implementation of the localisation makes process consumers and providers contribute together to localisation to ensure that defined policies are enforced. For a localised service requested by a process consumer, there are a number of activities including those from subprocesses within a process that will participate in the coordinated execution (a kind of transaction is required). The WS-Coordination specifications are designed for transactions of distributed Web services rather than transactions of application processes. Adaptive processes for handling processes transactions lack coordination mechanisms for our case to guarantee all participants working together in a unified manner. The coordination framework we implemented for this localisation context addresses these limitations. It includes suitable protocols for the participants for any application process.

The localisation framework uses a mix of local localisation services (e.g. unit conversion), external services (currency conversions) and hybrid techniques (e.g. for translation). This mix of widely distributed services makes a coordination solution necessary that takes failure into account. Services might become unavailable. Localisation policies might not be applicable as a consequence. A solution that allows the applicability of policies to be check prior to localisation execution or post-execution are therefore required [26].

We first introduce a coordination model which focuses on message exchange or coordination contexts between clients and mediators that act as coordinators. A coordination protocol for localisation policy enforcement in service transactions is also defined. We introduce an approach based on BPEL templates to implement the protocols with BPEL processes at provider side.

A. The Coordination Model

The underlying coordination model is derived from WS-Coordination and also the XACML access control policy framework. We adapted this to the requirements of our coordination mechanism for localisation policy enforcement. The adapted coordination model uses two types of subcoordinators for process consumers and providers. In this scenario, a participant only interacts with its own coordinator type. The coordination model is defined as $< COOR, COOR_{context} >$ with $COOR = COOR^c \cup COOR^p$. Here, $coor^c \in COOR^c$ is a coordinator associated with the consumer and $coor^p \in COOR^p$ is a coordinator associated with the provider. $coor_{context} \in COOR_{context}$ captures the coordinaton context (involved services and locale definitions). Figure 5 illustrates how $coor^c$ and $coor^p$ interact in a coordination conversion. Protocol X and services X_c and X_p are instances in this coordination protocol.

1) The process consumer sends a create coordination context request to the activation service of $coor^c$. It will receive back an initialized localisation context $coor_{context}$ (Cc) that contains the identification, a service reference of the $coor^c$'s protocol service and other information for starting a coordination conversation.
2) The process consumer then sends a process request to the provider or localisation process containing the $coor_{context}$.
3) The context $coor_{context}$ is extracted from the SOAP message and passed to protocol service X_p at $coor^p$.

Now, the protocol service X_c service reference is known to the protocol service X_p and theactual localisation-oriented communication between the participating services can be established.

4) The localisation coordination conversation ends with the completion of the process execution.

B. Process Activity Protocol

The process activity protocol defines a coordination type for coordination conversations. It is based on the coordination model. A coordination conversation of a localisation process is established for the coordination of the activities within the overall process for the service consumer. The model behind the coordination protocol is activity-centric, which means it can be applied to any localisation process irrespective of specific combination of localisation techniques applied. This coordination protocol applies to all activities of the processes to be managed on behalf of the client/consumer during execution. A coordination protocol consists of two main elements in ($ct \in coor_{context}$):

1) a message schema defines the message structure needed for services communication between consumer $COOR^c$ and provider $COOR^p$ for the extension element of the $COOR_{context}$.
2) a Finite State Machine (FSM) of $COOR^c$ and $COOR^p$ defining the actual localisation behaviour in an abstract model, which we will describe in more detail now.

The process activity protocol defines runtime localisation management for localisation processes and the responsibilities of service providers and consumers in the acutal management and execution of localisation as a contract. This runtime mediation is formulated as an FSM defining the coordination protocol. There is anFSM for every activity in the processes that describes the behaviours of consumers and providers, $COOR^c$ and $COOR^p$, in the conversations. The idea behind this FSM design is to instrument the states into the process flow as these states determine which localisation polocies are applicable.

The whole FSM is divided into two parts that are responsible for $COOR^c$ and $COOR^p$ separately. The $COOR^c$ FSM is a submachine of the FSM of $COOR^p$. Here, process providers only follow that part of the protocol that is actually defined for $COOR^p$. Similarly, consumers follow the FSM of $COOR^c$. As the FSM implementation is executed at the consumer and provider separately to achieve independence and, as already emphasised, the control of the consumer, $COOR^c$ needs sufficient information about process execution in order to execute process services at the provider side. The FSM of $COOR^c$ is defined for the submachine in the FSM of $COOR^p$, isolated from the process. Consequently, the protocol message schema only covers the activity information instead of the process state information. The execution of the $COOR^c$ FSM does only require information about the weaving request, which is instruments the original service with the localisation techniques to adapt the service. The execution of the $COOR^p$ FSM on the other hand does only require information about the weaving response. The motivation here is that the same message schema can be used for different coordination settings. A consumer can customize the FSM of $COOR^c$ for itself without affecting the $COOR^p$ FSM and other process consumers, which is one of the novelties and selling points of our solution. Furthermore, this solution reduces complexity in the state machine execution for both sides. Each side does not need to know any implementation details of other participants for its own implementation.

The design with two separate FSMs reduces the number of states in the FSM of $COOR^p$, hence reducing the message exchange times required between $COOR^c$ and $COOR^p$ for coordination conversations. This reduces the performance overhead, which is generally caused by any communication between the services. Depending on current network properties between consumer and provider, the message exchange could reduce in delays and low performance, often not acceptable in runtime adaptation situations as the localisation here. Of course, we need to note here that this create an additional requirement for the consumer side, because of the $COOR^c$ FSM needs to be implemented at the client side. On the other hand, this allows different protocols for different localisation settings to be defined for $COOR^p$.

We now formalise the FSMs. The FSM of $COOR^p$ specifies the protocol which is responsible for $COOR^p$ - the FSM of $COOR^c$ is specified in more detail in [23], [24]. The states reflect the status of the execution such as 'executing' or 'waiting' or 'completed'. A number of these states such as 'violated' or 'replacing' is necessary to deal with error situations that can occur in a distributed context where services or infrastructure can fail. These error states are based on common error handling strategies, as explained in [23], [24].

The FSM of $COOR^p$ is defined as a 5-tuple $(S, s_{start}, F, TA, \delta)$, where

- $S = S_g \cup S_{\neg g}$ is a set of states. S_l is a set of localisation states $\{s_{man\_val_{pre}}, s_{man\_val_{post}}, s_{handling_{pre}}, s_{handling_{pre}}, s_{cancelling}\}$ directly involved with process consumers or policies. The $S_{\neg l}$ is a set of non-localisation states $\{s_{start}, s_{violated_{pre}}, s_{executing}, s_{replacing}, s_{waiting}, s_{skipping}, s_{violated_{post}}, s_{compensating}, s_{com+rep}, s_{com+ign}, s_{completed}, s_{end}\}$ not directly involved with consumers.
- $s_{start} \in S_{\neg l}$ is an initial state. The coordination can only be started by the process provider, i.e. not directly involved with the consumers.
- $F \subseteq S_{\neg l}$ is a set of final states $\{s_{end}\}$.
- $TA = TA_l \cup TA_{\neg l}$ is a set of input symbols for the localisation actions. TA_l is a set of localisation transaction actions $\{ta_{violate}, ta_{validated}, ta_{ignore}, ta_{replace}, ta_{skip}, ta_{cancel}, ta_{compensate}, ta_{retry}, ta_{com+ign}, ta_{com+rep}\}$ expected from process consumers, again to handle errors. $TA_{\neg l}$ is a set of transaction actions which are not expected from process consumers $\{0, 1\}$. The input stream of the FSM regarding $TA_{\neg l}$ is decided by the process providers based on the process state information which is not covered by the FSM (remember that the FSM is only activity-scoped as explained earlier).

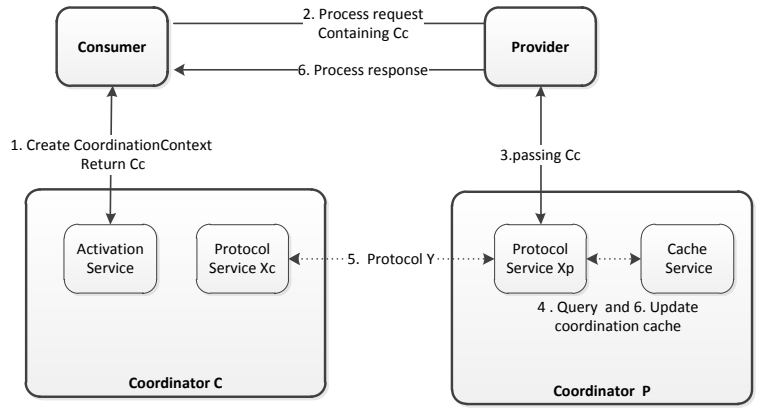

Fig. 5: Policy Coordination Architecture.

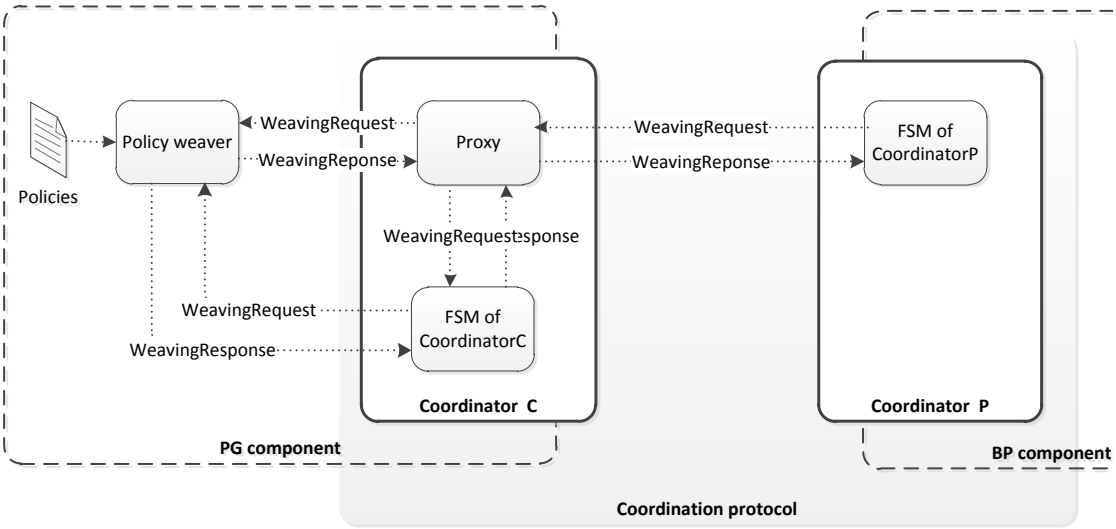

Fig. 6: Message flow diagram

- δ is a transition system $\delta : S \times TA \rightarrow S$, see transition graph in Figure 7.

C. Coordination Implementation and BPEL Instrumentation

The coordination protocol needs to be implemented to enable coordination. The difficulty is on the provider side, since all activities within a localisation process need to comply with the defined protocol during the BPEL execution.

We designed a set of templates for BPEL to avoid platform dependency, i.e. to allow this to be applied to different BPEL engines. The protocol needs to be implemented with a BPEL process as a $coor^p$ for activities. The process contains the flow logic to be executed and can be driven by protocol messages. A process instance, i.e. not the BPEL process, is associated with a coordination conversation belonging to a consumer to enable user-centric customisation.

In order to separate concers, we divide the FSM of $COOR_p$ into two sections. The first is process-independent, i.e., does not require awareness of the process states. This part of the FSM implementation is wrapped up in the main BPEL process. The second part continues the FSM to the end state of the main process. The first part can be implemented as BPEL processes, but as processes separate from the main process. Using such a hybrid approach, we can achieve a platform-independent approach that also keeps the main BPEL code simple. As a limitation we need to note that the BPEL processes here are protocol-specific. We can use the BPEL transaction scope concept in order to implement the FSMs with BPEL as long-running transactions (LRTs). These LRTs in BPEL focus on scopes and these scopes can even be nested. That means that when a fault occurs, all previously committed activities can either be compensated within the faulty process, or compensated as an activity in the parent process.

A template approach allows for easier management. Two templates for BPEL process development reduce the protocol implementation effort. A template defines the abstract skeleton of an algorithm. One or more of the algorithm steps can be overridden by subclasses allowing to define differing localisation behaviours by the consumer while at the same time ensuring that the overall protocol is followed. We extract the first section of FSM as the non-transactional requirement FSM for localisation process activities. The second section is then an extension for activities to support transactions. So, the FSM

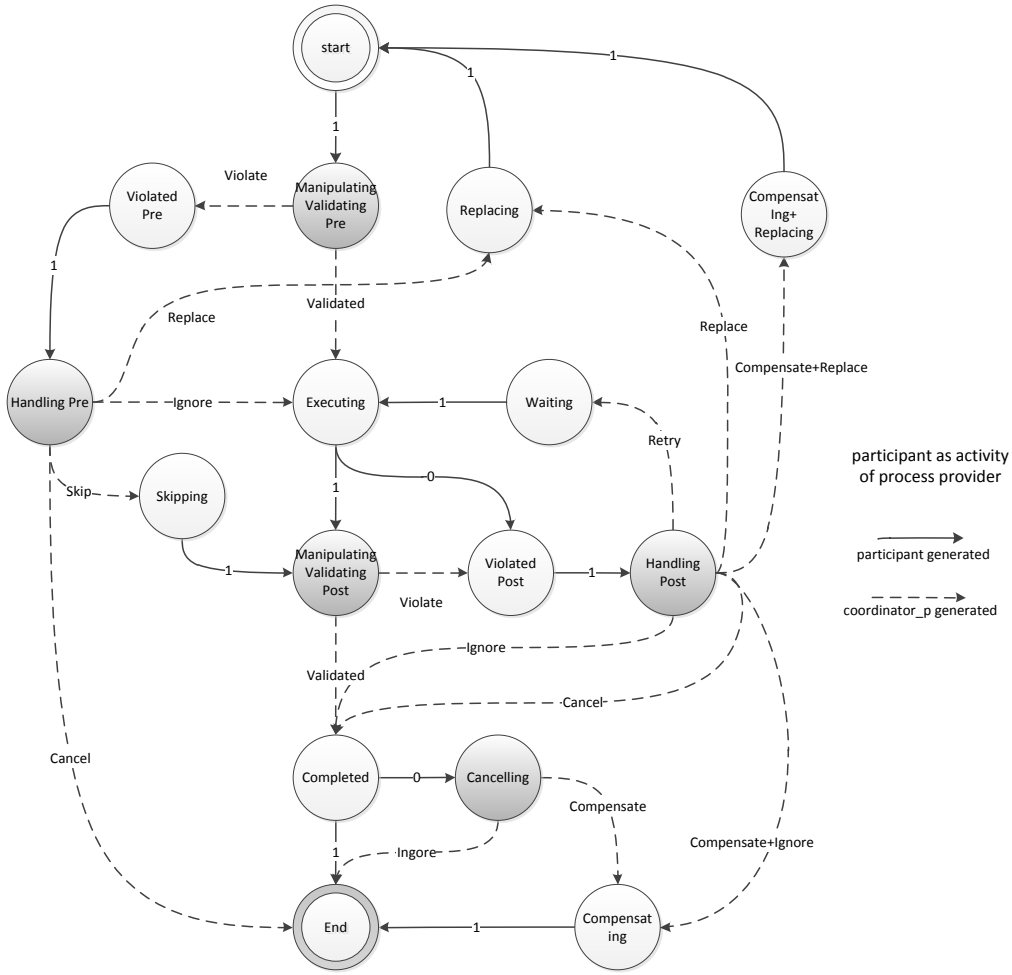

Fig. 7: Transition graph for FSM for $Coor^p$

is devided into two implementation parts with two respective templates: the wrapper service template and the main process template.

This process template is an implementation of the second part of the FSM containing activity states from $s_{completed}$ to the s_{end} state. When the process is in a cancelling state, the previous successfully executed activities can be compensated if that is necessary. The template is designed with an activity scope and a process scope, respectively.

Figure 8 shows the BPEL template for the activity scope associated with activity states is also needed. The template for each activity is a separate scope. The two services inside the template are highlighted by grey boxes. The first service is the wrapper service for the first part of the FSM implementation. The required variables are passed into the BPEL process by a BPEL $<$*assign*$>$ activity. With the BPEL $<$*if*$>$ control structure, a $<$*throw*$>$ activity throws a defined fault if the *comp* variable is set to false. An attached BPEL $<$*catchAll*$>$ handler catches the fault and marks this scope as faulty. The BPEL $<$*compensationHandler*$>$ attachment is only triggered by a successful scope if the process in a cancelling state. In this situation, e.g. if the execution state $s_{executing}$ is skipped in the first FSM part, the compensation handler for the activity scope is then triggered. The scope is marked as faulty instead. The last $<$*if*$>$ conditional control structure marks the process as being in cancellation status. It in this cases throws a defined fault and to be caught in a $<$*catchAll*$>$ handler defined in the process scope template. Thus, the $<$*compensationHandler*$>$ handler at the corrwesponding activity scope is triggered. The process activities are executed from state $s_{completed}$ to state $s_{cancelling}$ if necessary. A utility service within the $<$*compensationHandler*$>$ moves on from the activity state $s_{cancelling}$ to the state s_{end}.

Finally, Figure 9 shows the BPEL template for the process scope. All process activities are within a process scope – this is associated to a $<$*catchAll*$>$ handler. If a defined fault for process cancellation is caught by the handler with the process scope, all $<$*compensationHandler*$>$*s* of activity templates of fault-free activities are executed in reverse order. Activities in the $s_{completed}$ state will transfer to the cancellation state $s_{cancelling}$. In case of nested processes, the parent would handle the situation. The violation handling would depends on the fault policy defined in the parent process. We have not covered these fault aspect here in details, as our focus was on the core localisation activities, but not their fault

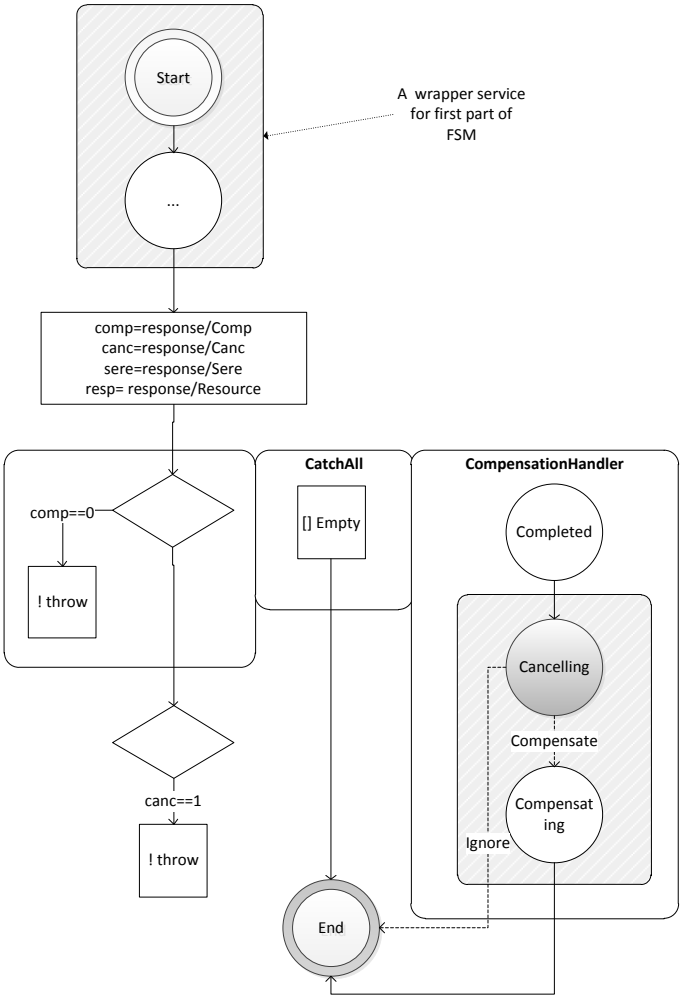

Fig. 8: Activity scope BPEL template

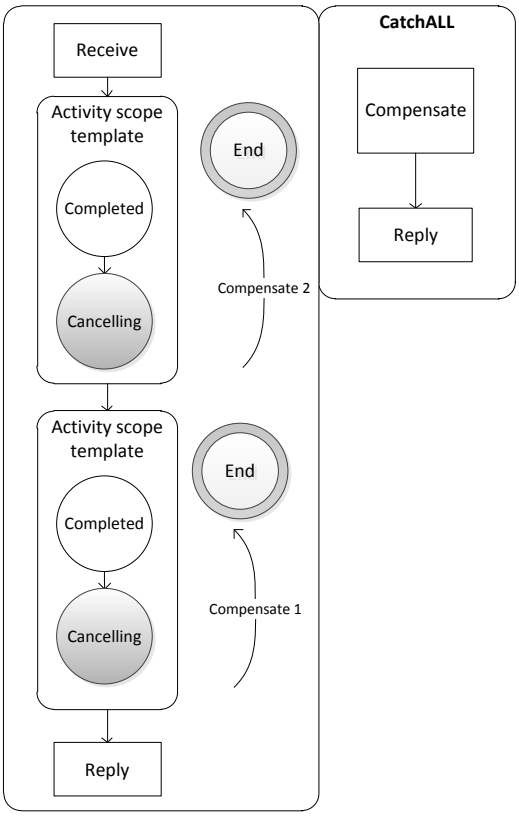

Fig. 9: Process scope BPEL template

handling. However, given the distributed nature of control and processing, the provision of fault handling is necessary to provide a credible solution architecture.

VI. Implementation and Evaluation

The localisation platform presented here was fully implemented in a Java prototype for the localisation techniques, combined with the coordination solution based on WS-Coordination and BPEL, that aims at studying the feasibility of the conceptual solution. It shall be assessed based on the following criteria here: Performance and Extensibility of the localisation techniques. 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. Please note that other relevant aspects of the solutions, for instance the performance of the coordination solution in a generic form have been presented elsewhere [23], [24]. These have established an overhead of around ten per cent for the coordination framework - however, the ten per cent essentially come into effect if fault handling is needed, as the templates in the previous section indicate. As said, the focus here is on the localisation services themselves that are facilitated through the coordination platform.

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 [24]. 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 key concerns 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 to processing delays. This delay 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. The figures are based on randomly distributed service calls to external and internal localisation services (e.g. external currency conversion or internal unit conversion or hybrid conversions as for translations) based on stock market services (NASDAQ, FTSE). For a number of localisation policies, the individual response times have been aggregation and normalised. 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.

As a general strategy, we have aimed to improve per-

TABLE IV: Latency Table - Localisation of Service

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

formance of the prototype by using pre-translated aspects through stored mappings e.g. for currency conversions and standard translations, which suggests that further optimisations are possible.

A related concern is scalability of software becomes more important when a service may have large multiples of users, which can be the case if several clients use the localisation framework at the same time. The performance evaluation has been carried out for a single user to determine the overhead of localisation for a single service call. Scalability has not been empirically addressed for this phase of research and will be evaluated in later prototypes that will implement a more scalable base architecture.

- 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 [8], [9], [13], which would allow a flexible and elastic assignment of localisation services to multiple requests. Work by Creaner and Pahl [8] suggest a good scalability poential through tuple-space for coordination.

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.

VII. Related Work

We provide a different view and perspective on the subject compared to other publications on service adaptation [4], [11], [17], [20] that look at adaptation from a technology perspective. The area of localisation in its wider adaptivity and customisation sense has been worked on in various EU-supported research projects, such as SOA4ALL [19] and 4Caast [1]. 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 [18], [22]. Adaptation and Integration of services based on locales and using a translation memory with rules and mappings is new [20]. The problem of multi-tenancy is a widespread issue in the area of cloud computing [24]. 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], [6], [11].

There are implementations which can perform localisation operations on web services [12]. The use of some of these, however, is restricted due to their nature. 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 [12], 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.

VIII. 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 defined by language or regulatory environment. We have presented a modular implementation which can enable software 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. In order to enhance the dynamic evolution of localisation settings, our coordination infrastructure allows a client-side driven management of localisation settings.

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, controlled by the client and enacted by the provider. 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. It also allows service consumers more influence on the type of localisation and frequency of localisation changes.

The objectives of Service Localisation have been presented in terms of three aspects. 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 third part introduces the coordination platform, which is necessary to coordinate conversations between different service providers and consumers and manage potential failure.

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 wider localisation concerns in an integrated format. 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 would be 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 to deal more specifically with scalability problems).

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] L. Baresi and S. Guinea. "Self-supervising bpel processes." *IEEE Transactions on Software Engineering*, 37(2):247 – 263. 2011.

[5] R. Barrett, L. M. Patcas, C. Pahl and J. Murphy. "Model Driven Distribution Pattern Design for Dynamic Web Service Compositions." *International Conference on Web Engineering ICWE06*. Palo Alto, US. ACM Press. 2006.

[6] 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.

[7] 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.

[8] G. Creaner and C. Pahl. "Flexible coordination techniques for dynamic cloud service collaboration." In: Cubo, J. and Ortiz, G., (eds.) Adaptive Web Services for Modular and Reusable Software Development: Tactics and Solutions. IGI Global, pp. 239-252. 2012.

[9] 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.

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

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

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

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

[14] 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.

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

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

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

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

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

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

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

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

[23] 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.

[24] 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.

[25] 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.

[26] Y. Wu and P. Doshi. "Making bpel flexible and adapting in the context of coordination constraints using ws-bpel." *Intl Conf on Services Computing*. 2008.