

# Support for Adaptive Cloud-Based Applications via Intention Modelling

T. Baker, A. Taleb-Bendiab, M. Randles, Y. Karam

*School of Computing and Mathematical Sciences,*

*Liverpool John Moores University,*

*Byrom Street, Liverpool, L3 3AF, UK*

[t.baker@ljmu.ac.uk](mailto:t.baker@ljmu.ac.uk), [a.talebbendiab@ljmu.ac.uk](mailto:a.talebbendiab@ljmu.ac.uk), [m.j.randles@ljmu.ac.uk](mailto:m.j.randles@ljmu.ac.uk), [y.karam@ljmu.ac.uk](mailto:y.karam@ljmu.ac.uk)

## Abstract

*Recent years have seen a surge of research into cloud computing to providing tools, techniques and support infrastructure for delivering affordable on-demand dependable network-centric applications and services. Whilst, much is already known regarding the development of such applications via the application of autonomic computing, SOA, design principle to name but a few. However, cloud computing providers need to support hundreds of thousands of users and applications/services and ensure that they are reliable, secure, and available. In order to accomplish this goal, they will need to build a dynamic, intelligent cloud based-application modelling approach that allows the fulfilment of emergent users' requirements. To this end, this paper proposes a new modelling approach referred to as Provision, Assurance and Auditing (PAA), which provides via an Intention language a method for dynamically composing cloud-based services together with associated self-management support. The paper will present the PAA modelling method and its use to create highly dynamic cloud-based-applications that can respond to the new requirements that might occur suddenly at runtime software evolution. The paper uses the PetShop blueprint to illustrate the use of PAA together with the runtime support for software evolution.*

## 1. Introduction

In the last years, many evolving computing domains, such as grid computing, service-oriented-computing and on-demand computing; have turned towards service-oriented architectures (SOAs) [4] as an enabling paradigm, and amongst the possible technological implementations, Web services have emerged as the leading solution. A distinct characteristic shared by all these domains is that they all perform an important assumption-shift regarding the surrounding world. The applications we want to build are intrinsically distributed, and extremely dynamic at runtime, and we want to harness the possibilities the environment is capable of offering us. Cloud computing was shine here; it intends to provide the distributed services and components over grid just on demand, taking advantage

and putting forward the evolving computing domains' ideas.

The distributed nature of cloud-based applications, the intrinsic high level of dynamism and flexibility that cloud should offer to those applications, and the fact that we do not own all of the system's components, prove that we have to use a composition technique such as BPEL [1] that leverages remote services that we do not own and integrating applications that run in Grid or Cloud environment. However, this leaves us in an uncertain situation and number of unanswered questions; what cloud can do if new requirements arose at runtime? How cloud deals with the not responding or slow services and unavailable components for integration? Can the users modify their requirements according to what cloud provides at runtime? Consequently, we see an increasing interest towards developing a dynamic and adaptive Intention-Cloud-Based-Model that can be used in particular towards assuring integral cloud-based-applications and allow users to amend their requirements according to the emergent needs at runtime. This allows cloud-based applications to avoid the negative effects of such anomalies executing compensation actions. In this regard, PAA offers the following novel features: (i) Support for modelling and simulation of large scale Cloud-Based-Applications using the full adaptive Intention model which sits in the cloud and accessed and modified by the users, (ii) Support for assuring the fulfilment of the system specifications constraints and user requirements via CA-SPA [8]. Among the unique features of PAA, there are: (i) Flexibility to switch between different Intention models at runtime. These compelling features of PAA would speed up the achievement of the continuously changed requirements, as shown in figure 1. (ii) Another important factor behind the novelty of PAA assurance model, orthogonal to the points mentioned above, involve the way the Assurance strategy is determined and defined. In some approaches the assurance strategy is *predefined* at design time (static approach), in PAA, the assurance strategy can be defined *dynamically* at runtime, as explained later

(in section 4.2), depending on the actual situation. While the former approaches are easier to realize, they offer much less flexibility at run-time as well as the unexpected run time faults cannot be covered.

As outlined above, the Provision, Assurance, Auditing (PAA) modelling approach allows a given cloud-based application to be dynamically adapted at runtime by modifying the intention model of the application and its components. Thus, adding new components, removing undesired services, modifying the flow/sequence of the processes/activities and updating the intention attributes to reflect the emergent users' requirements, can be managed through the intention adaptation. While the functional behaviour of a cloud-based applications is typically controlled by user input through the Intention Model, the assurance technique of PAA focuses on the process Validation & Verification (V&V) using the Concept Aided-Situation Prediction and Action (CA-SPA) construct consisting of three major features:

- (i) *Situation*: A copy of the intention model, for the application, is saved within in the assurance class.
- (ii) *Prediction*: A comparison is made, between the situation and intention model, each time the intention model is updated
- (iii) *Action*: When a failure or unexpected event is detected in a process during a runtime adaptation, the Action part of CA-SPA is treated as a real Situation facilitating the injection of new CA-SPA rules using the Assurance Editor to define the context of the new behaviour.

In short, this paper presents a developed modelling approach (PAA) to enable creating highly trusted full adaptive Cloud-Based-Applications through the use of Intention-Cloud-Based-Model (ICBM).

## 2. Cloud-Based Applications Adaptation Categories

Cloud-based applications intend to provide distributed services and components on-demand to heterogonous environment. Those services and components can be integrated together in the cloud to deliver different functions. As the runtime users' requirements can be changed periodically, numerous adaptation failures may occur at runtime and have different impacts on the functionality of the cloud-based application. In general, adaptations for Cloud-Based-Applications can be categorized into two main runtime adaptation categories:

- *Configuration adaptation*: includes modification of certain application parameters and properties (e.g. non-functional properties) or modification of services involved in the composition (e.g. replacement of one service with another). A typical scenario for describing configuration adaptation is where there is a desire to change one or more of the constituent services. The change may happen when one

or more of the constituent services become unavailable or if their quality-of-service characteristics degrade. In this case adaptation occurs by substituting "bad" services to ameliorate the errant behaviour.

- *Composition adaptation*: deals with changes in the application structure, i.e., the flow of the activities or processes are modified. As examples, additional activities are invoked, previous operations are undone, or completely different composition fragments are executed. The process/composition model of the Cloud-Based-Applications is changed, and then this may be termed composition adaptation. These changes may be entailed by various factors, such as the necessity to customize the application in order to deal with particular user types, or the necessity to operate in a new environment, where policies and constraints for the given application are different. In this case, the application should change the executed flow of activities in accordance with the adaptation specification. This may include, for example, skipping certain activities, performing additional ones, executing completely different process fragments, or rolling back already performed activities. A typical scenario for composition adaptation, using predefined strategies, is process migration: a new process model is defined and all the running instances should be changed in order to correspond to this new model [2,3]. Adaptation in this case changes those parts of the running application instance that have not been executed yet. Other approaches define potential variants in the application model and conditions, under which the variant applies [4]. In some approaches the adaptation specification has a form of meta-Level instructions to be applied to the process model, such as "redo a part of the process", "roll back to safe point", "skip to a particular execution point", "substitute a part of the process", etc [5]. Using certain application and users requirements, the composition is created automatically and may be even recomposed in case of certain changes or problems. In [6] the approach relies on a set of functional and non-functional requirements and constraints, on which the services are composed and mediated dynamically. In [7] a domain process model with different deviations from the norm is used as a basis for the composition.

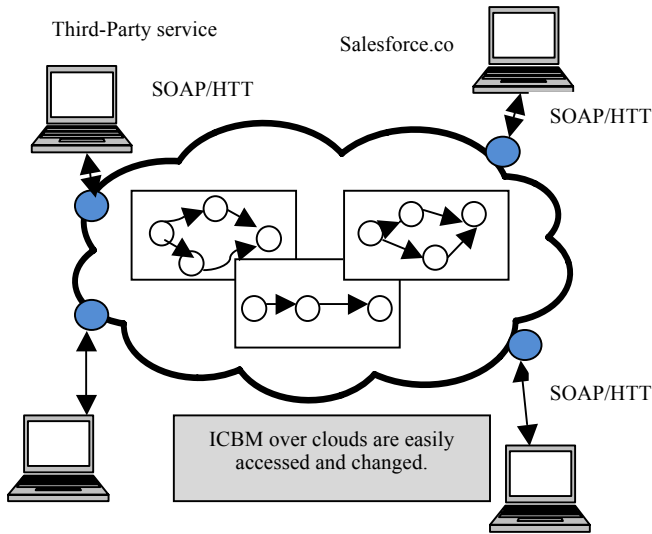


Figure 1: ICBM dispersal in the Cloud

### 3. Assured Adaptation via Intention Model Modification

In order to provide a flexible method to support the two adaptation categories listed above, and Intention-Cloud-Based-Model (Fig. 1), based on the distribution of the Intention models over public or private clouds. The users then can access those models and use them. In addition to this, the user can even modify the intention model via PAA by adding, removing and updating services and the processes. The intention model is composed of *flow model* and *logic model*. Whereby,  $I$  representing the intention model can be defined as:

$$I = \langle T, c_p \rangle$$

where  $T$  represents a set of tasks to be executed to achieve the goals, which define the Intention model, and  $c_p$  represents a set of conditions which define and/or control the associated task set. In other words,  $T$  represents the flow model, and  $c_p$  represents the control flow model of a given Intention model. The following xml code shows a simplified PetShop intention model with both  $T$  and  $c_p$

```

<PetShopIntention>
  <process>
    <startpoint id="StartPoint1">
      <moveto>Checkout</moveto>
    </startpoint>
    <action id="Checkout">
      <input type="text">
<message><![CDATA[checkout.aspx]]</message>
      </input>
      <moveto result="[Not
set]">ValidateAccount</moveto>
    </action>
    .....
    <ask id="Valid" NeptuneFunction="process">

```

```

      <input type="text">
<validation><![CDATA[1]]</validation>
<message><![CDATA[0]]</message>
      </input>
      <moveto result="1">Billing</moveto>
      <moveto result="0">EndPoint1</moveto>
    </ask>
    .....
    <endpoint id="EndPoint1" />
  </process>
</PetShopIntention >

```

Figure 2: Original ICBM of Order process of PetShop

As is shown in the above Intention Model, there are number of tasks should be executed in this Intention to achieve the desired goals of it (e.g. checkout action). These tasks represent  $T$  in  $I$  definition. The `<moveto>` represents the  $c_p$  as defined above.

As illustrated in Figure 3, the ICBM as input to the PAA framework is first interpreted to generate the valid abstract process and control model, which will be used by the *Semantic Linker* to generate a BPEL like orchestration model. During this stage, using both tasks and discovered software services – including web services, and software components -- ontological and WSDL definitions the *Semantic Linker* automatically associates tasks with required services (endpoints) that can be invoked to perform the required task.

The *Composer* composes those services and components together to produce a new functionality. The produced composition will be sent concurrently to the *Execution Engine* (e.g. BPEL Engine) and to the *Assurance Model* in *PAA Shell* through the *Assurance Checker*. Once the composition is received by the *PAA Engine*, the *Provision Generator* and the *Auditing Generator* start creating the Provision, and Auditing models. Thus, the user can use those models again when he/she wants without the need to go through the *Semantic linker* path.

As a result, as shown in (Fig. 3), there are options to use PAA, (i) as a new user by accessing the ICBM over cloud, which can then be modified and adapted via PAAEditor, and interpreted via the *interpreter* and *semantic linker*. (ii) As an old user who has accessed the ICBM before and the orchestration model has been generated via PAA. Thus there is no need to go through the *semantic linker* and the *choreographer* processes again as long as the user does not want to modify the ICBM. In this case, this would save the time spent over the *semantic linker* and the *choreographer* processes.

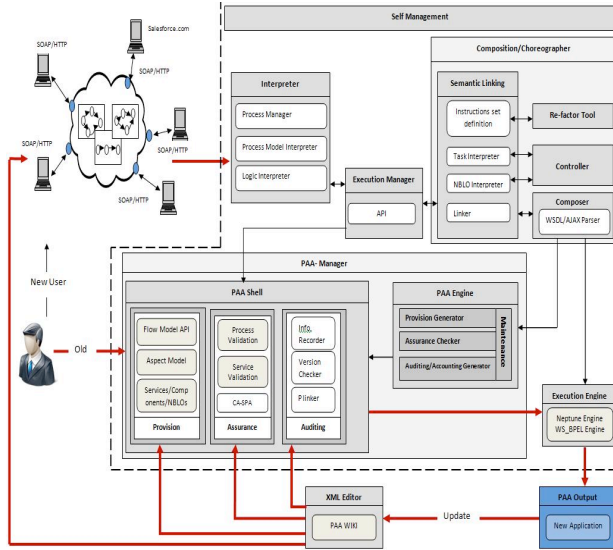


Figure 3: ICBM and PAA interaction

Programmatically, the following example shows definitions of the associated PAA objects (*Provision*, *Assurance* and *Accounting*) with *PetOrderDetails* meta-information in NeptuneScript. The first significant entry, *PetOrderDetails* is a Neptune concept used to define the data requirements for the *Pet Order*, which will be exchanged between processes as required. This assumes that some additional concepts have previously been described (*Pet*, *Address* and so on), though these are omitted from this definition.

The two PAA definitions are, in this instance, used to define acceptable criteria for a given order. The first (*ValidPreShippingOrderPAA*) defines criteria that specify an available and valid *Pet Order* which is ready for despatch, but has not yet been despatched, nor scheduled. The second (*ValidShippingOrderPAA*) defines criteria that specify an available and valid *Pet Order* which has either been scheduled for despatch, or has been despatched in the past.

```

-----
// indicates that the following metadata is presented
as NeptuneScript
definition format as NeptuneScript
// definition of data object
define PetOrderDetails as NBLO
{
  features{
    feature SelectedPet      as Pet;
    feature ShippingAddress as Address;
    feature ShippingBand    as Band;
    feature ShippingDate    as Date;
  }
}

// definition of key Provisioning / Validation
condition for pre shipping order
define ValidPreShippingOrderPAA as PAA
{
  provisioning
  { order as PetOrderDetails }
  assurance
  {

```

```

    order.ShippingAddress is VALID
    order.SelectedPet     is INSTOCK
    order.ShippingDate    is NULL
  }
}
Accounting{/*error handling or logging defined here*/}
}
// definition of key Provisioning / Validation
condition for:
define ValidShippingOrderPAA as PAA
{
  provisioning
  { order as PetOrderDetails }
  assurance
  {
    order.ShippingAddress is VALID
    order.ShippingDate    is VALID // either
    past or future is fine
  }
}
Accounting{/*error handling or logging defined here*/}
}
-----

```

Figure 4: Data objects definitions and PAA Validation

### 3.1 CA-SPA: PAA Assurance Mechanism

As described above, there is still a lack of adequate support and associated tools and methods for assurance techniques that can handle and trigger recovery in cases of unanticipated system failures. Such failures are likely to arise spontaneously at runtime through software evolution; for example modifying non-functional properties, replacing one service with another or different composition fragments executing. Thus new assurance techniques should be adaptive at runtime by allowing the injection of new assurance rules to the application according to the encountered failures. Thus, in PAA the assurance takes the responsibility for assuring that the system and the amended parts of the Intention model are working well according to some predefined policies and reports come from the Intention Interpreter, however, the currently defined policies added by CA-SPA, can tackle this issue. To this end, the following are used together to allow fixing and recovery from unanticipated failure:

1. The Concept Aided-Situation Prediction Action (CA-SPA) mechanism is specified and implemented to support the PAA Assurance mechanism. In other words, CA-SPA policies are used to control the behaviour of the systems<sup>1</sup>.
2. The PAA Assurance Editor allows the injection of new CA-SPA rules to the application at runtime according to new situations.

<sup>1</sup> The full description of the language is outside the scope of this paper but can be found in [8].

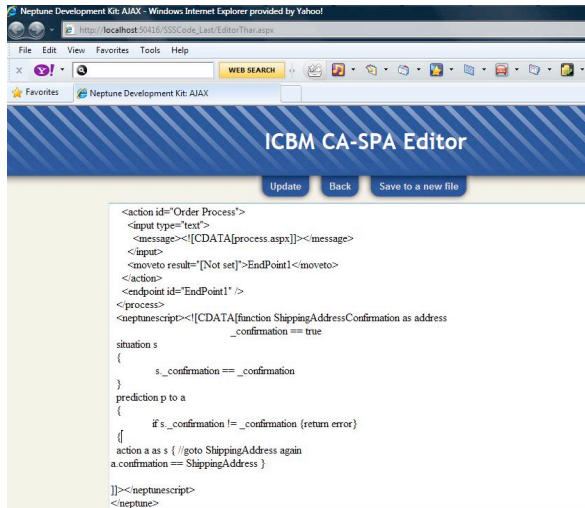


Figure 5: ICBM CA-SPA editor

CA-SPA translate into the high level CA-SPA policy to indicate that when the confirmation activity is not executed the prediction condition is triggered which in turn means the movement to the action as situation condition. Back to our PetShop scenario, if at runtime the shipping address is not confirmed (e.g. address outside UK), then in this case, the pet order cannot be processed as the address is not allowed by the system. The ICBM CA-SPA editor (Fig. 5) can be used to add a new CA-SPA rule which can tackle this problem. The Situation represents the default system state (e.g. the address is true). If that is the state, then there is no need to take any further action. In the Prediction part, the predicted situation is added here (e.g. if the address is abroad), then the action part should be executed to correct the predicted situation. The using of the above CA-SPA Assurance editor allows the assurance rules to be updated at runtime by adding a new rule, removing an existing rule and modifying a bad rule.

#### 4. Case Study: Designing PetShop via PAA

PetShop is an architectural blueprint developed by Microsoft based on the original Sun Microsystems PetStore benchmark enterprise architecture for e-commerce and enterprise systems [9]. The application produced by the PetShop blueprint builds a *web-based* application for browsing and purchasing pets. In particular, the PetShop application is implemented as a Web Service composition that perform the main processes outlined in Figure 6, which for instance enable administrators of the system to add, remove, and modify the animals available for sale, with their data stored in a database.

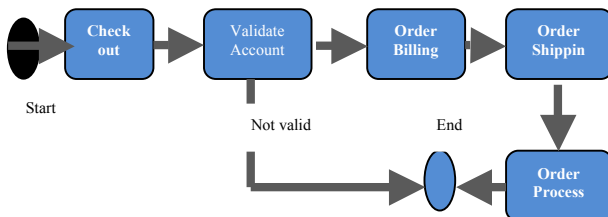


Figure 6: The original PetShop Order Process

In this case-study, we outline the evaluation details of the developed PAA method and framework to support runtime adaptation of cloud-based applications using the PetShop benchmark. To this end, this section outlines the adaptation processes when a new process or service can inject (or simply added) at runtime to the standard PetShop process model.

The standard PetShop blueprint does not support autonomic design principles, thus, it is static design approach. In other words, the process model cannot be adapted at runtime (e.g. add a check delivery to the original order process). The whole system should be taken offline before the adaptation, and then re-uploading it again online after the adaption. Of course, this step should be done by the system administrator. So the user will not be able to even add any new function or service to the original process model on the cloud. Hence, the new way for designing the PetShop via PAA by using the PetShop ICBM which then can be accessed and modified by the user, is a highly trusted and dynamic design approach. It allows the user to modify the entire process model and uploading the modified version at runtime. However, the PAA framework facilitates users' runtime adaptation of different cloud components and links them to the original model. In this regard, the Check delivery process can be added easily to the original process model by using the PAAEditor (Fig. 5). Figure 7 shows the original ICBM of the order process of PetShop.

```

<PetShopIntention>
  <process>
    .....<!--this part in figure 2-->
    <action id="Order Shipping">
      <input type="text">
        <message><![CDATA[shipping.aspx]]></message>
      </input>
      <moveto result="[Not set]">Order
Process</moveto>
    </action>
    <!--inject check delivery process here -->
    <action id="Order Process">
      <input type="text">
        <message><![CDATA[process.aspx]]></message>
      </input>
      <moveto result="[Not
set]">EndPoint1</moveto>
    </action>
    <endpoint id="EndPoint1" />
  </process>
</PetShopIntention>

```

Figure 7: Original ICBM of Order process of PetShop We can see in the above ICBM, the transition from one process to another one is done via `<moveto>` tag. This would necessitate the mission of injecting a new process

to the ICBM and changing the flow of the entire processes. However, if the user or the system administrator wants to add (check delivery) process to the ICBM, this would be done by adding the following xml code using PAAEditor:

```

-----
<action id="CheckDelivery">
  <input type="text">
<message><![CDATA[Checkdelivery.aspx]]></message>
e>
  </input>
  <moveto result="[Not
set]">OrderProcess</moveto>
</action>
-----

```

Figure 8: a new added CheckDelivery Process

This means that the new process is added between the OrderShipping process and OrderProcess (see the green font in Fig. 7), that's why the *moveto* in the new process is linked to the OrderProcess, Fig 9.

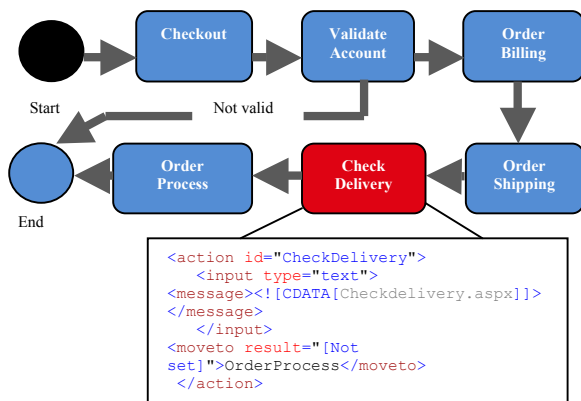


Figure 9: amended ICBM

Thus, at any second during the Cloud-Application execution, the user can modify the Intention model of the application and creates new application behaviour. Managing the cloud-based by Intention models and the intelligent PAA modelling approach helps the cloud-baseds' providers to support users and services and ensure that they are reliable, secure, and available.

## 5. Conclusions

This paper has argued the use for an adaptable intention model through the using of the PAA approach as it will be straightforwardly, quickly and reliable amend the cloud-based-applications at run time without the need for the recoding and the republishing of the application. The perceived value of PAA has driven the development of Cloud-Applications through this approach. The assurance mechanism used here (CA-SPA) and the way to add, remove and amend CA-SPA rules at runtime ensures the highly trusted creation of the cloud-baseds.

In future work, the PAA models are to be rigorously tested and evaluated to ensure the design of PAA tools is correctly envisaged. But there is still a large body of outstanding research issues, which have to be resolved before the full benefits of exploitation can be realised.

## 6. References

- [1] IBM, BEA Systems, Microsoft, SAP AG, and Siebel Systems, Web page for the specification of the Business Process Execution Language for Web Services, version 1.1 (updated 08 feb 2007), 2007, <http://www-128.ibm.com/developerworks/library/specification/ws-bpel/> (accessed Nov. 25, 2009).
- [2] Reichert, M., Rinderle, S., Kreher, U., Dadam, P., "Adaptive Process Management with ADEPT2", *Proceedings of the 21<sup>st</sup> international conference on Data Engineering (ICDE 2005) IEEE Computer Society, April (2005)*.
- [3] Hallerbach, A., Bauer, T., Reichert, M., "Managing Process Variants in the Process Life Cycle", *Technical Report, University of Twente, TR-CTIT-07-87 (2007)*.
- [4] Siljee, J., Bosloper, I., Nijhuis, J., Hammer, D., "DySOA: Making Service systems Self-Adaptive", In: *Benatallah, B., Casati, F., Traverso, p., ICSOC 2005, LNCS, vol. 3826. Springer, Heidelberg, pp. 255-268 (2005)*.
- [5] Baresi, L., Guinea, S., Pasquale, L., "Self-Healing BPEL Processes with Dynamo and the JBoss Rule Engine", In *ESSPE 2007, International workshop on Engineering of software services for pervasive environment: in conjunction with the 6th ESEC/FSE joint meeting, Publisher ACM, (2007)*.
- [6] Verma, K., Gomadam, K., Sheth, A.P., Miller, J.A., Wu, Z., "The METEOR-S Approach for Configuring and Executing Dynamic Web Processes", *Technical Report, LSDIS Lab Technical report, University of Georgia, Athens, Georgia, (2005)*.
- [7] Lazovik, A., Aiello, M., Papazoglou, M.P., "Associating Assertions with Business Processes and Monitoring their Execution", *ICSOC '04: Proceedings of the 2nd international conference on Service Oriented Computing, Publisher ACM, pp 94-104 (2004)*.
- [8] Philip, M., Talebbendiab A., "CA-SPA: Balancing the Crosscutting Concerns of Governance and Autonomy in Trusted Software", *Proceedings of the 20th International Conference on Advanced Information Networking and Applications - Volume 2 (AINA '06) (Washington, USA), IEEE Computer Society, pp. 471-475, (2006)*.
- [9] Microsoft Download centre, "PetShop 3.0 .NET Sample Application" <http://www.microsoft.com/downloads/details.aspx?FamilyId=E2930625-3C7A-49DC-8655A8205813D6DB-&displaylang=en> (Accessed Dec. 01, 2009).