Quality-Driven Business Policy Specification and Refinement for Service-Oriented Systems

Tan Phan[1], Jun Han[1], Jean-Guy Schneider[1], and Kirk Wilson[2]

[1] Faculty of Information & Communication Technologies
Swinburne University of Technology
P.O. Box 218 Hawthorn, VIC 3122, Australia
{tphan,jhan,jschneider}@swin.edu.au

[2] CA Labs
One CA Plaza, Islandia, NY 11749, USA
{Kirk.Wilson@ca.com}

Abstract. Enterprise software systems play an essential role in an organization's business operation. Many business rules and regulations governing an organization's operation can be translated into quality requirements of the relevant software systems, such as security, availability, and manageability. For systems implemented using *Web Services*, the specification and management of these qualities in the form of *Web Service policies* are often complicated and difficult to be aligned with the initial business requirements. In this paper, we introduce the HOPE (High-Level Objective-based Policy for Enterprises) framework that supports, in a systematic manner, the specification of quality-oriented policies at the business level and their refinement into policies at the system/service level. Quality-oriented business requirements are expressed in HOPE as quality objectives applied to business entities and further refined or translated into system-level WS-Policy statements. The refinement relies on an application-specific business entity model and application-independent domain quality models. We demonstrate the approach with a case study involving policy specification and refinement in the security domain.

1 Introduction

Business rules, government acts such as Sarbanes-Oxley [1], industry standards such as Basel II [2], and enterprise-specific rules mandate non-functional or quality requirements of the various entities in an organization's IT environment. These requirements can often be formulated as high-level quality objectives (*e.g.*, *Customer data must be kept confidential*) and realized using various means for IT management and governance.

In recent years, Service-Oriented Architectures (SOA) and Web Services (WS) have offered a new way of implementing enterprise business processes. Core business functionalities are codified as network-accessible Web Services and enterprise software systems become live networks of interconnected services. To ensure that WS-based SOA systems are reliable and interoperable, various industry standards have been proposed to support the specification and management of quality aspects, most notably security, reliable messaging, and transactions [3]. In general, these standards are about system-level mechanisms to achieve some non-functional qualities. Example mechanisms in security or distributed transaction coordination are role-based access control, message encryption and signing, and content-based routing. The *Web Services Policy Framework* (WS-Policy) [4] is a standard that supports the specification of various quality properties for Web Services and service systems.

One of the issues that needs to be addressed is how to *align the high-level and often business-oriented quality objectives with the system-level realization mechanisms* offered by the WS standards. Currently, the quality objectives are often identified by practitioners who are either business analysts or IT compliance officers (hereafter referred to as *policy experts*). They have a good understanding of the business domain and regulations, and have a high-level understanding of the IT systems in general. However, policy experts are typically not SOA experts and often do not have an in-depth understanding of all the system-level realization mechanisms used to achieve the quality objectives. They view IT systems more from a business perspective and their concerns are to identify the quality objectives rather than how they can be realized. It is the system developers' responsibility to implement the quality objectives in the corresponding IT systems. The underlying processes are generally ad-hoc and, therefore, it is difficult to ensure that a system fully implements all required quality objectives. As such, a contribution of great value would be a systematic process and related techniques that can derive the system-level realization from the business-level requirements and can verify that the realization actually fulfills these requirements.

In this work, we address the issue of aligning business-oriented quality objectives with system-level WS quality properties by introducing the HOPE (High-level Objective-based Policies for Enterprises) framework. HOPE assists policy experts in specifying business policies and quality objectives and, by utilizing realization mechanisms available in the respective *quality domains*, refines them into system-level WS-Policy statements that prescribe quality properties for service-based enterprise software systems. This paper starts with a business case study as a motivating example. It then introduces the HOPE framework and the mechanisms for policy refinement, illustrated using selected examples from the case study. A prototyping tool for HOPE is also presented. The paper concludes with a summary of the main contributions and future work.

2 A Motivating Example

In this section, we introduce an example business process and identify applicable rules and regulations. A set of quality-oriented business policies is then derived from the rules and regulations. We discuss the limitations of current approaches with regards to specifying and realizing such policies, motivating our approach of the HOPE framework.

2.1 Business Case: The `Mortgage Loan Approval` Business Process

When a customer applies for a mortgage loan product at a hypothetical multi-national bank `SwinBank`, a `LoanOfficier` accepts the application and triggers the bank's loan approval business process. First, the bank arranges a professional appraiser to estimate the market value of the collateral property. Next, the customer's ID/social security number is forwarded to a credit checking unit to verify the customer's credit history. A list

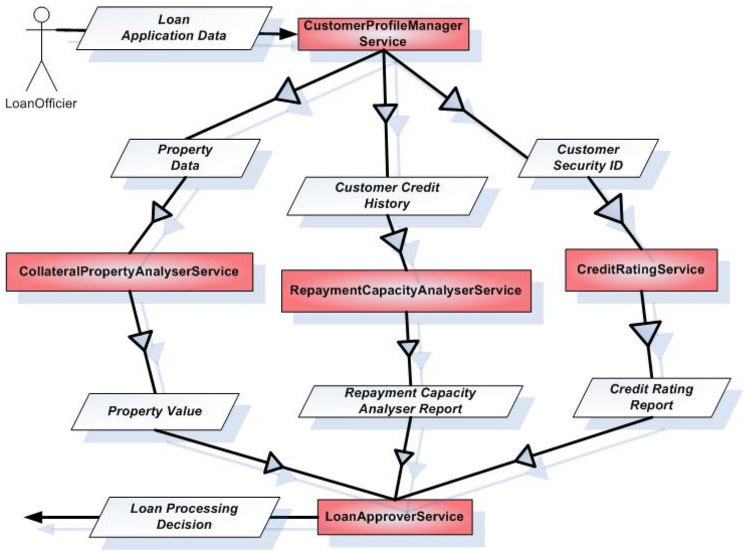


Fig. 1. The `Mortgage Loan Approval` business process

of credit scores from multiple credit rating agencies is then obtained. Finally, the repayment capacity of the customer is checked by judging the income against the amount to be repaid. Based on this information, an approval decision for the loan is made.

2.2 `MortgageLoan`: The `Mortgage Loan Approval` Application

`SwinBank` automates its mortgage loan approval process using SOA. The process is implemented in BPEL utilizing a number of services (depicted in Figure 1).

The service `CustomerProfileManager` provides customer account information whereas `CollateralPropertyAnalyser` calculates the value of a given property. The `CreditRating` service, acting as a gateway to other credit rating agencies' WS, forwards the customer's social security number to the agencies and obtains the customer's credit history report. The service `RepaymentCapacityAnalyser` checks the customer's repayment capacity based on his/her income and the amount of loan to be repaid, taking into account interest rate, inflation, and other factors. Finally, the service `LoanApprover` takes the output of the previous three services and makes a decision, with human input, as to whether the loan is approved or not.

2.3 Rules, Regulations and `SwinBank` Business Policies

The discussion of the mortgage loan approval business process and system for `SwinBank` has primarily focused on the business and application functionality. In reality, this business process is also subject to many rules and regulations that may be general or specific to the Banking Industry. The following are some of the relevant acts:

1. **Bank Secrecy Act of 1970** [5]: Any information disclosed by the applicants, any temporary data collected during the approval process, and any final decision need to be persistently stored and traceable.
2. **Australian Privacy Act 1988** [6]: Information related to loan applicants' credit information, customer identifier information must be available to only authorized personnel and not disclosed to the public.

By analyzing the rules and regulations, policy and compliance experts can identify the *business policies* applicable to the loan approval process and system. In general, business policies can be functional or non-functional (*i.e.* system's qualities). In this paper, we focus on the latter. The following are some of the non-functional quality-oriented business policies for the loan approval process and system:

(BP1): *Loan application data must be persistently stored.*
(BP2): *Information about customers' personal identifications and financial records must be kept secure during transmission.*
(BP3): *All activities related to loan processing must be recorded.*
(BP4): *Loan applicants must not be able to repudiate the lodgment of a loan and the bank must not be able to repudiate the receipt of a loan application.*
(BP5): *People working with customer information must be authorized.*

These business policies require `MortgageLoan` to have the corresponding quality properties related to the privacy and security of data and loan processing activities.

2.4 Realization of Quality-Oriented Business Policies

In SOA development, system requirements (functional and non-functional) are generally realized through services and the WS policies applicable to these services (at the *assembly* or *deployment* phase) [7]. The fact that some system requirements are realized through policies increases the flexibility and agility of the system. For example, WS policies can be updated to realize certain system changes without modifying the services' implementation. As the WS-Policy framework is predicated on WS interactions, only requirements that concern WS interactions can be realized through Web Service policies. Other requirements have to be realized in the services themselves. In this paper, we focus on quality-oriented non-functional requirements or *business policies*, and in particular those that can be realized through WS policies at the system-level (cf. Figure 2). For example, the requirement in **BP1** cannot be fully realized by the WS quality model and, therefore, needs to be realized programmatically or using other means (such as database transaction management).

In current SOA practice (cf. Figure 2(a)), it is the system developers' responsibility to realize the quality-oriented business policies in terms of WS policies. This process is generally ad-hoc and there is no easy way to ensure that all the relevant business-level policies are properly interpreted and implemented by the developers, who have generally limited understanding of the rules and regulations [7]. Furthermore, this process is often very tedious as well as error-prone.

To alleviate this problem, we have developed the HOPE framework with the aim to *automate* the process of refining high-level business policies into system-level quality

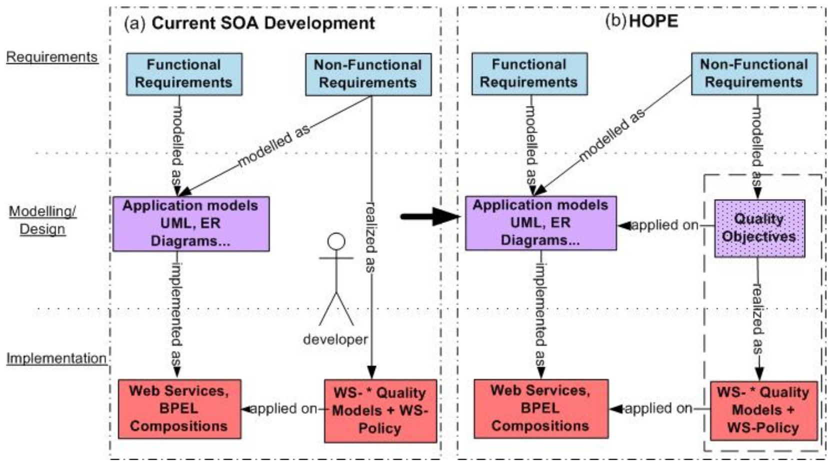

(a) Current SOA Development
(b) HOPE
Requirements
Modelling/ Design
Implementation
Functional Requirements
Non-Functional Requirements
modelled as
Application models UML, ER Diagrams...
implemented as
realized as
developer
Web Services, BPEL Compositions
applied on
WS-* Quality Models + WS-Policy
Quality Objectives

Fig. 2. Current SOA development (a) and the HOPE approach (b)

properties (cf. Figure 2). In particular, we introduced the concept of *quality objectives* to model non-functional business requirements in the form of required qualities applicable to the various entities in a system's design models. These quality objectives are then refined into WS policies applicable to the service-based system.

3 The HOPE Framework

HOPE is a framework for specifying high-level, quality-oriented business policies and refining them into system-level Web Services policies for Web Services-based enterprise software systems in a systematic manner. HOPE is built on a number of underlying models as illustrated in Figure 3.

The *quality models* are domain-specific and identify, for a given domain (*e.g.*, security), the relevant quality attributes and the mechanisms to realize them. The *application entity model* provides a layered hierarchy of business-oriented entities involved in the application, and can be extracted from the application's design models. The business concepts in this entity model are used to specify the *quality objectives* for the system and to annotate the system's WS elements (*portTypes*, *operations*, *messages* etc.). Based on the quality models and the WS annotations, HOPE refines and translates the high-level *quality objectives* into system-level policy statements asserting WS properties. The remainder of this section discusses these models in more detail.

3.1 Domain Quality Models

For each quality domain a *quality model* defines the set of *quality attributes* concerning the domain and the *realization mechanisms* for these attributes. The quality model itself is application-independent and is often derived from *standards*, *ontologies*, *patterns*, and *best practices*, respectively, and the system-level WS quality specification WS-*, where * denotes the *quality domain name* for that domain. We have chosen

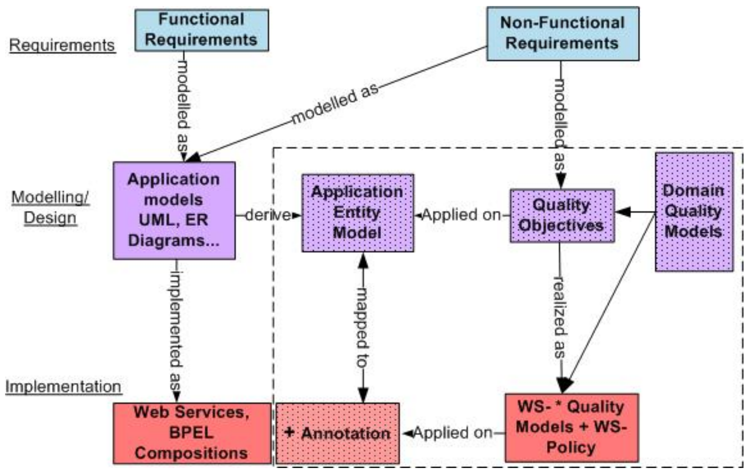


Fig. 3. The HOPE framework

XML Schema as a means to specify quality models and defined a corresponding *quality meta-model* each quality model must adhere to. In the following, we will further illustrate this approach using the security domain as an example.

The Quality Meta-Model and the Security Quality Model. The quality meta-model provides a formal structure to specify the entities of each quality domain. Throughout the remainder of this section, we will use a simplified security quality model based on [7,8,9,10,11] for illustration purposes (cf. Figure 4). We expect that other domain quality models can be expressed using the same set of notations and structure as defined in the security meta-model.

- ***Quality.*** a quality attribute or quality, q, specifies a desired quality aspect. In the security domain, the common quality attributes are *confidentiality* (preventing unauthorized access to sensitive data), *integrity* (preventing unauthorized modification of the data), *non-repudiation* (preventing a message sender from repudiating the fact that it was him who sent the message and a message receiver from repudiating the fact that it was him who received the message), *authentication* (proving the authenticity of a user), *authorization* (proving that the user is in the role he claims), and *audit* (making sure that actions are recorded and traceable).
- ***Quality realization mechanism.*** a quality realization mechanism defines how a quality q can be realized. It is a logical structure in *disjunctive normal form* (*OR* of *ANDs*) of *abstract quality functions*. For example, according to [12,13], (i) *confidentiality* can be realized by *encrypting* data that needs to be protected, (ii) *integrity* can be realized by *signing* the data, and (iii) *non-repudiation* of a message can be realized by *logging* the sending and receiving actions and *signing* the message.
- ***Quality function.*** a quality function specifies a measure that can be used to achieve one or more *qualities*. A quality function f is specified in the form $f(fb)$ where fb denotes the *function binding* (defined below) for that function. If fb is left empty, the function is called an *abstract function*, only specifying *what* needs to be done.

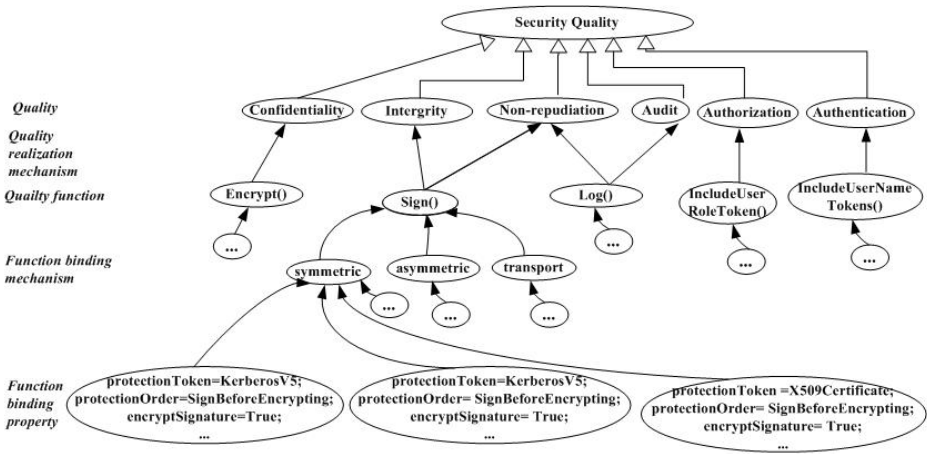

Security Quality
Quality
Quality realization mechanism
Quailty function
Function binding mechanism
Function binding property
Confidentiality
Intergrity
Non-repudiation
Audit
Authorization
Authentication
Encrypt()
Sign()
Log()
IncludeUser RoleToken()
IncludeUserName Tokens()
symmetric
asymmetric
transport
protectionToken=KerberosV5; protectionOrder=SignBeforeEncrypting; encryptSignature=True;
protectionToken=X509Certificate; protectionOrder= SignBeforeEncrypting; encryptSignature= True;

Fig. 4. The partial security meta-model

If fb is defined, the function is called a *concrete function*. For example, some of the common abstract security functions as defined in [12,13] are *encrypt*, *sign*, *log*, *includeUsernameToken*, and *includeUserRoleToken*.

- ***Function Binding Mechanism1.*** a function binding mechanism represents a method of realizing a quality function based on a type of quality infrastructure. A quality domain typically has a limited set of alternatives for realizing a given quality function. For example, security functions such as *encrypt* or *sign* have the three binding mechanisms (i) *transport* (using transport security such as HTTPS), (ii) *symmetric* (using a shared key), and (iii) *asymmetric* (using a pair of public and private keys).
- ***Binding Property Set.*** a binding property set $\{(p_1, v_1), \ldots, (p_n, v_n)\}$ contains name-value pairs where p_i is the name of a property and v_i the corresponding value. Example security binding properties are `AlgorithmSuite=SHA1`, indicating that the `SHA1` algorithm suite is used, and `SignatureProtection=TRUE`, indicating that both the signature and the signature confirmation elements must be encrypted.
- ***Function Binding Tree.*** a function binding tree is formed by detailing abstract functions with *binding information*. The root of a binding tree is the *abstract function* itself containing empty binding information. The direct children of the root node contain the function's *binding mechanisms*. Child-nodes of the binding mechanism nodes are leaf nodes containing all possible values of available binding properties for that mechanism. Furthermore, function binding trees can be assigned priorities for particular binding mechanisms and/or properties. For the security function *encrypt*, the binding tree is formed by having a root node being the abstract *encrypt* function, the direct children of the root nodes are the binding mechanisms *transport*, *symmetric*, and *asymmetric*, and the sub-nodes contain detailed binding properties for each of the binding mechanism (*e.g.*, `Algorithm=SHA1`).

Binding trees can potentially become quite large as the number of possible branches is defined as the Cartesian product of all available *binding mechanisms* and all applicable *binding property* values. However, an organization often follows a certain security profile which has a limited number of predefined binding options. For example, the *Basic Security Profile*, Version 1.0 [14] mandates the use of message level mechanisms and places some constraints on values of certain binding properties (*e.g.*, *`SHA1`-based algorithms must be used for interoperability purposes*).

System-Level Web Services Quality Models. At the system-level, qualities are applied to Web Services in the form of *WS-Policy statements*. The WS Policy framework (WS-Policy) allows the specification of *qualities* and their realization details for WS. For each quality domain, there is a WS-*Policy standard, such as the WS-SecurityPolicy [15] for security, that allows for the specification of the qualities and their realization details in that domain. WS-Policy is extensible and specifications for new quality domains can be defined and included in the framework. It is aimed at defining non-functional properties that govern service-service or service-client interactions, but not the implementation details of the services themselves. However, there are qualities that cannot be supported by the WS quality model, *e.g. durability* for persisting data. Therefore, when defining a quality model, the system-level WS quality model for that domain is taken into consideration in order to filter out the qualities, functions, mechanisms or properties that are not supported by the system-level model.

3.2 The Application Entity Model

An application's *entity model* defines application-specific business concepts. Policy experts work with this model and apply policies on the entities in the model in the form of quality objective requirements. In general, such a model is extracted from an application's analysis and design models (*e.g.*, ER or UML diagrams) made available by business analysts or system architects during system analysis. In HOPE, a *Business-Entity* represents a business-oriented concept from the application and can be classified into one of the following basic entity types:

- *Processor:* performs business logic at request (*e.g.*, `LoanProcessor`),
- *DataItem:* holds business data (*e.g.*, `CustomerTaxFileNumber`), and
- *UserRole:* represents user roles in an organization, has access to *DataItems*, and can ask *Processors* to perform actions (*e.g.*, `LoanOfficier`).

In a HOPE entity model, each entity is a direct or indirect specialization of one of the three basic entity types. Although an entity in a given application can be the specialization of more than (super-)entity, it can only be the (direct or indirect) specialization of *one* of the basic entity types. For example, an entity cannot be a specialization of both, *DataItem* and *Processor*, respectively. Using this approach, applications can be viewed as compositions of interacting entities.

A HOPE entity model can be represented as a directed graph of entities: a node corresponds to an entity and a (directed) edge represents an entity specialization. Figure 5 represents part of the entity model for the motivating example `MortgageEntityModel` introduced in Section 2. The entities `CustomerData` and `PersonalIdentifier` and their specializations `LoanApplicationData`, `LoanApplicantCreditHistory`, and `LoanApplicantTaxFileNumber` are specializations of the basic entity type *DataItem*, `LoanProcessor` and `CreditVerifier` are specializations of *Processor*, and finally `LoanOfficier` and `Teller` are specializations of the basic entity type *UserRole*.

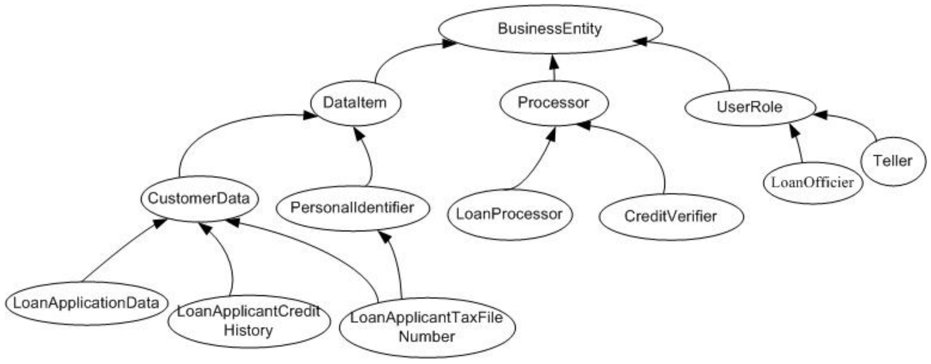


Fig. 5. Example `MortgageEntityModel`

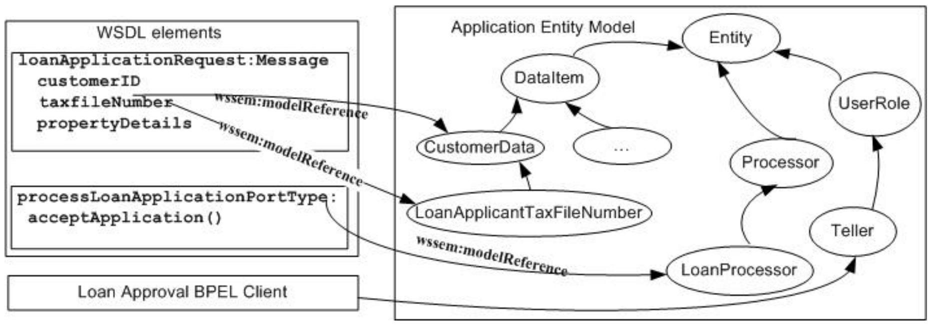


Fig. 6. Example Web Services and entity model mapping using WSDL-S

Annotating Services and Messages. Once the high-level entity model is defined, the Web Services elements *portTypes*, *operations*, and *messages* of an application's implementation need to be mapped to the high-level business concepts defined in the model in order to perform policy refinement. The mapping is done via annotations using the WS Semantic (WSDL-S) framework (chosen for its simplicity and tool support) [16].

Figure 6 shows an example of mapping Web Service elements to concepts of the application entity model using WSDL-S for `MortageLoan`. The WSDL-S annotations `wssem:modelReference` are used to map Web Service *messages* such as `loanApplicationRequest` and *message parts* such as `CustomerID` and `TaxFileNumber` (represented in the WSDL description of the service) to specializations of *DataItem*. In a similar manner, *portTypes* such as `processLoanApplicationPortType` and *operations* such as `acceptApplication` are mapped to the corresponding specializations of *Processor*. Any service client that uses these services is also annotated with *UserRole* information, indicating which user role constraints this client has to adhere to.

3.3 Quality Objectives and Policies

A central concept in HOPE is the *quality objective*. A quality objective, denoted by *q[e]*, specifies the application of the quality *q* on the business entity type *e*, meaning that

Table 1. Business policies and corresponding quality objectives

Business Policy	**Quality Objectives**
BP2	(*"Information about customers' ... secure during transmission"*, *confidentiality*[LoanApplicantTaxFileNumber], *confidentiality*[LoanApplicantCreditHistory], *integrity*[LoanApplicantTaxFileNumber], *integrity*[LoanApplicantCreditHistory])
BP3	(*"All activities related to loan processing must be recorded"*, *audit*[LoanApplicationData], *audit*[LoanApplicantCreditHistory], *audit*[LoanApplicantCreditResult]
BP4	(*"Loan applicants must not be able ... of a loan application"*, *non-repudiation*[LoanApplicationData])
BP5	(*"People working with customer... must be authorized* **"**, *authorization*[LoanOfficier])

quality q must hold on all entities of type of e and all of its specializations. Furthermore, a policy is defined as a n-tuple $p(t,\ q_1[e]_1,\ \ldots,\ q_{n-1}[e_{n-1}])$ where t is the textual representation in natural language of the business policy requirement, $q_i[e_i]$ is a quality objective and $\{q_1[e]_1,\ \ldots,\ q_{n-1}[e_{n-1}]\}$ is the set of quality objectives meeting the requirements specified by the policy. This information is made available to the policy experts when they apply a quality onto an entity.

Some qualities can only be applied to certain types of entities. For example, *manageability* qualities like *notifiability*, *controllability*, or *introspectability* can only be applied to specializations of *Processor*. In the security domain, *confidentiality*, *integrity*, and *non-repudiation* can only be applied to specializations of *DataItems* whilst *authentication* and *authorization* are only applicable to specializations of *UserRoles*.

The example business policies specified in Section 2 can be decomposed into *quality objectives* as given in Table 1. As mentioned before, **BP1** cannot be supported by HOPE. Apart from that, quality objectives of the other four policies can be refined into WS-SecurityPolicy assertions, based on the security quality model and the annotated Web Service descriptions (cf. Section 4).

4 Generating WS-Policy Assertions

The refinement of a quality objective into WS-Policy Assertions involves two major steps: (i) the quality objective is realised in terms of *concrete quality functions*, according to the quality model and (ii) the *concrete quality functions* are mapped to corresponding WS-Policy statements for that domain. The statements are applied on the relevant WS that will be manifested when the services operate at runtime. In this process, mapping information available in WS annotations is used to identify the relevant WS elements that correspond to the business entities in the original quality objectives.

Each quality domain has its own way of generating WS-*Policy statements from the domain's *concrete functions*. We will discuss the methods for generating WS-Security-Policy assertions from concrete security functions throughout the rest of this section.

The current version of the WS-SecurityPolicy [15] specification defines different types of assertions for specifying the mechanisms of applying security measures on SOAP messages. There are basically three types of assertion relevant to our mechanism: (i) *protection assertions*, (ii) *token assertions*, and (iii) *binding assertions*.

4.1 Mapping an Abstract Functions to WS-Security Assertions

Security Functions for *DataItems*: The WS-SecurityPolicy *protection assertions*, specifying what security measures need to be applied on which parts of SOAP messages, can be used to describe security functions for *DataItems* such as *encryp* or *sign*. For a security function `function`$_X$ to be applied on the entity `DataItem`$_X$ which, via annotation, is known to be carried by a collection of `<Message`$_1$, . . ., `Message`$_N$`>`, the corresponding WS-SecurityPolicy protection assertion for the function `function`$_X$ is

```
<functionXAssertion>
   <Xpath>Message1 </Xpath>
   ...
   <Xpath>MessageN </Xpath>
</functionXAssertion>
```

where `<Xpath>Message`$_i$`</Xpath>` is the path pointing to the message or message part relative to the SOAP document, and the mapping between `function`$_X$ and its assertion is as follows:

Quality	Realization Functions	WS-SecurityPolicy assertions
Integrity	*encrypt*	`SignedElements`
Confidentiality	*sign*	`EncryptedElements`
Non-repudiation	*encrypt* AND *log*	`EncryptedElements` //Log assertion has not been defined

Security function for Processors and UserRoles: The focus of WS-Security and, therefore, WS-SecurityPolicy, is not to protect *UserRoles* and *Processors*. However, existing mechanisms can be leveraged to support the realization of *authentication* and *authorization* by using *token assertions* as follows:

Quality	Realization Functions	WS-SecurityPolicy assertions
Authentication	*IncludeUsernameToken()*: Attach a username token in messages originated from the user.	`<wsse:SecurityToken wsp:Usage="wsp:Required"> <wsse:TokenType> wsse:UsernameToken </wsse:TokenType> </wsse:SecurityToken>`
Authorization	*includeToken – SAML.* Attach a SAML token messages originated from the user.	`<wsse:SecurityToken wsp:Usage="wsp:Required"> <wsse:TokenType> wsse:SAMLToken </wsse:TokenType> </wsse:SecurityToken>`

To associate these assertions with the WS *portTypes* and operations, we use the mechanisms specified in WS-PolicyAttachment [17]. The security functions applied on a *UserRole* are, via entity mapping information, also applied on WS, WS *portTypes*, or WS *operations* that the role might have access to (*i.e.* invoke) accordingly using a similar mechanism. For example, `LoanOfficier`, via annotation is known to have access to the *portType* `LoanProcessor`, thus quality objectives such as *authorization*[LoanOfficier] are translated into corresponding WS-Policy assertions that are, via WS-PolicyAttachment, applied on the `LoanProcessor` *portType*.

The reader may note that the WS-SecurityPolicy [15] standard does not define assertions for all well known security functions as WS-Security itself focuses more on message *confidentiality* and *integrity*. For example, an assertion for *logging* is not available. However, the standard is still evolving and it is expected that additional support will be accommodated in future versions.

4.2 Mapping Function Binding to WS-SecurityPolicy Binding Assertions

The general structure of a WS-SecurityPolicy binding assertion is as follows

```
<BindingMechanism>
   <Structured collection of Binding property assertions>
</BindingMechanism>
```

The *BindingMechanism* can be *symmetric*, *asymmetric* or *transport* binding. The "*structured collection of Binding property assertions"* is generally a logical AND of the assertions that specify the value of the binding properties. In WS-Policy syntax, this logical AND can be represented using a `wsp:Policy` or a `wsp:All` container. For a concrete security function, the function `BindingMechanism` in the quality function is mapped to the corresponding WS-SecurityPolicy *BindingMechanism* assertions and each of the *BindingProperty* is mapped to the corresponding WS-Policy *BindingProperty* assertions. The mapping is relatively straightforward and is one to one.

4.3 An Example Assertion Generation

Figure 7 shows the generated WS-SecurityPolicy fragment for the quality objective *integrity*[LoanApplicantTaxFileNumber] in the business policy **BP2**. This quality objective can be realized using the security function *sign* to sign the Web Services message parts related to `LoanApplicantTaxFileNumber`. *sign* is mapped to the WS-SecurityPolicy protection assertion `signedElements` (Line 17). We use the *entity mapping* information as in the example introduced in Section 3.2 to map the message part `taxFileNumber` of the message `processLoanRequest` to the business concept `LoanApplicantTaxFileNumber` (Line 18). We assume that *SymmetricBinding* is used (Line 2, 16) and that the preferred *binding properties* are as follows:

```
{protectionToken = KerberosV5ApReqToken11 (line 4-12),
ProtectionOrder = SignBeforeEncrypting (line 13),
EncryptSignature = True (line 14)}
```

meaning that a `KerberosV5ApReqToken11` is used as the protection token, the digital signature to be computed over plain text (before the message content is encrypted), and that the signature itself should also be signed, respectively.

```
<sp:Policy>
  <sp:>SymmetricBinding>
    <wsp:Policy>
       <sp:ProtectionToken>
          <wsp:Policy>
             <sp:Kerberos.../>
                <wsp:Policy>
                   <sp:>WSSKerberosV5ApReqToken11/>
                <wsp:Policy>
             </sp:Kerberos>
          </wsp:Policy>
      </sp:ProtectionToken>
      <sp:>SignBeforeEncrypting/>
      <sp:>EncryptSignature/>
    </wsp:Policy>
  </sp:>SymmetricBinding>
  <sp:>SignedElements...>
     <sp:XPath>processLoanRequest/taxFileNumber</sp:XPath>
  </sp:>SignedElements>
</sp:Policy>
```

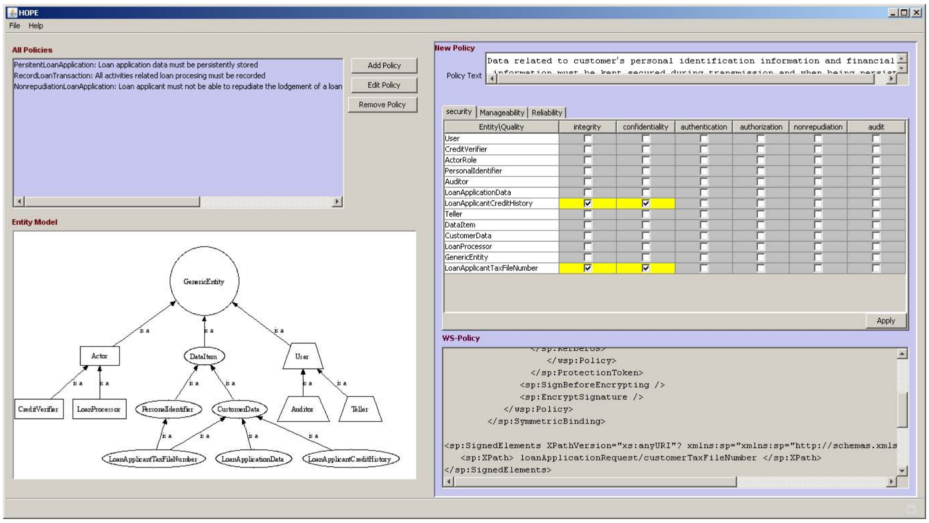
Fig. 7. WS-SecurityPolicy fragment for *integrity*[LoanApplicantTaxFileNumber]

5 Prototype Tool

We have implemented a supporting prototype for the HOPE framework. The prototype assumes the existence of domain quality models and application entity models. The tool also needs the WSDL descriptions of Web Services and service compositions of a given application. As illustrated in Figure 8, the prototype allows users to specify and manage business policies, and have them refined into WS-Policy statements. For policy editing, the tool presents the application entities and the qualities for each of the three domains (*security*, *manageability*, and *reliability*) in tabular format with one dimension being the set of qualities available in a domain and the other dimension being the list of entities in the application entity model (as seen in the top right corner of Figure 8). A user ticks a checkbox corresponding to a (*entity, quality)* pair to apply *quality* to *entity* to form a *quality objective*. The application entity model is also visualized (bottom left corner).

If a user applies a quality to an entity, the quality is also automatically applied to all specializations of this entity. Qualities that cannot be realized by the Web Services quality model (*i.e.* not supported by WS-*, WS-*Policy) are not displayed for selection. Invalid combinations, that is, qualities that are not applicable on some types of entities, as discussed in Section 3.3, are also disabled from user selection.

If a user clicks "Apply", the tool associates the formed quality objectives with the natural language representation of the policy. It then refines the business policy into a system-level policy by generating a set of WS-Policy statements that correspond to these objectives. In the current implementation, the tool only supports refinement in the security domain and assumes the *Basic Security Profile 1.0* [14] to be applied. During refinement, the tool automatically uses the first available branch in the *binding tree*



All Policies

PersitentLoanApplication: Loan application data must be persistently stored
RecordLoanTransaction: All activities related loan procesing must be recorded
NonrepudiationLoanApplication: Loan applicant must not be able to repudiate the lodgement of a loan

New Policy

Policy Text: Data related to customer's personal identification information and financial

Entity\Quality	integrity	confidentiality	authentication	authorization	nonrepudiation	audit
User	☐	☐	☐	☐	☐	☐
CreditVerifier	☐	☐	☐	☐	☐	☐
ActorRole	☐	☐	☐	☐	☐	☐
PersonalIdentifier	☐	☐	☐	☐	☐	☐
Auditor	☐	☐	☐	☐	☐	☐
LoanApplicationData	☐	☐	☐	☐	☐	☐
LoanApplicantCreditHistory	✓	✓	☐	☐	☐	☐
Teller	☐	☐	☐	☐	☐	☐
DataItem	☐	☐	☐	☐	☐	☐
CustomerData	☐	☐	☐	☐	☐	☐
LoanProcessor	☐	☐	☐	☐	☐	☐
GenericEntity	☐	☐	☐	☐	☐	☐
LoanApplicantTaxFileNumber	✓	✓	☐	☐	☐	☐

WS-Policy

```
              </wsp:Policy>
           </sp:ProtectionToken>
         <sp:SignBeforeEncrypting />
         <sp:EncryptSignature />
      </wsp:Policy>
   </sp:SymmetricBinding>

<sp:SignedElements XPathVersion="xs:anyURI"? xmlns:sp="xmlns:sp="http://schemas.xmls
   <sp:XPath> loanApplicationRequest/customerTaxFileNumber </sp:XPath>
</sp:SignedElements>
```

Fig. 8. Screenshot of the HOPE prototype

unless the tree is annotated with user's preferences. In that case, the branch with the highest priority is followed.

6 Related Work

Our work is related to a number of areas, including business rule specification and management, business rule refinement, and Service Level Agreement (SLA) specification and management.

There has been a body work on the specification and management of SLA and Service Level Objectives that focuses on ensuring the delivery of quality services to clients. Keller and Ludwig [18] proposed the Web Service Level Agreement (WSLA) standard to allow the specification of service level agreements and objectives for Web Services. SLANG [19] is another effort of specifying end-to-end service level contracts between a client and the service. Their work considers quality of service more from a client point of view and thus focuses on the specification of rules to govern client-service interactions. Our research investigates quality of services from an enterprise system governance and management viewpoint of which the purpose is to have a dependable/interoperable SOA ecosystem including the SOA services and service clients and also all the applications built on top of the services themselves. As such, the types of interactions we are concerned with are not limited to client-service interactions.

The objective of our work is more aligned with that of business rule specification and management as we are concerned with services being compliant to high-level business rules and regulations. In this field, several business rules specification languages and frameworks have been proposed [20,21] and some rule engines have been built to support the execution of business rules. However, the target business rules are more concerned with business logic and functional logic (*e.g.*, "*if (FlightBookingActivity is performed) then (Role type is airline)*" [20]) while we focus on the non-functional quality properties related to the various entities.

In the area of policy-based specification and refinement, frameworks such as [22,23] support platform-independent specification of non-functional requirements such as those related to access control or configuration management in the form of policy statements. However, these frameworks are mainly for resource management and cannot be easily adapted for SOA systems as discussed in our previous work [24].

There have been a number of attempts to apply model-driven architecture (MDA) techniques for the modeling and translation of SOA qualities into system-level realization mechanisms [25,26]. In these approaches, quality properties of services and applications are modeled in platform-independent ways and then transformed into platform-dependent code and configurations for middle-ware to realize these qualities. However, the entities being modeled, even though being platform-independent, are still technical entities (*i.e.* they represent technical concepts such as *filter*, *connector*, *service*, or *proxy*), not business-oriented entities. This not only limits the participation of business analysts and IT compliance officers in the modeling process, but also makes it difficult to align the models with the original business requirements.

7 Conclusions and Future Work

In this paper, we introduced the HOPE framework that, in a systematic manner, assists practitioners in defining quality-oriented business policies and refining them into system-level Web Service policies in order to realize quality requirements in the service-based applications. Central to the framework are domain-specific quality models, each of which codifies the quality attributes and their realization mechanisms in a given domain. Based on these quality models and an application's business entity model, quality-oriented business policies applicable to the application can be stated as quality objectives. Again using the quality models, quality objectives can be refined into Web Service policies as part of the application's Web Service-based implementation. This framework assists system developers in performing such tasks with a systematic approach and associated models, techniques, and tool support.

In general, HOPE does not aim for fully automated policy refinement as decomposing a high-level business policy into a set of system-level policies requires complex modeling and reasoning. HOPE's approach is that human decision should be leveraged when a policy statement in natural language needs to be interpreted and decomposed into a set of quality objectives. On the other hand, automation is provided (as much as possible) to refine these quality objectives into system-level statements, thereby abstracting away the complexity of the system-level infrastructure.

We expect that the HOPE framework can be adapted or generalized to be used with other policy frameworks/platforms in addition to WS-Policy. As part of future work, we will further examine the relationships among business policies, system requirements, system qualities, service/composition design, and system-level policies to improve the system development process and a system's adaptability and evolvability.

References

1. Sarbanes, P.: Sarbanes-Oxley Act of 2002. The Public Company Accounting Reform and Investor Protection Act. Washington, DC, US Congress (2002)
2. Basel, I.: Basel II: International Convergence of Capital Measurement and Capital Standards: a Revised Framework (2004)
3. O'Brien, L., Merson, P., Bass, L.: Quality attributes for service-oriented architectures. In: SDSOA 2007: Proceedings of the International Workshop on Systems Development in SOA Environments, Washington, DC, USA, p. 3. IEEE Computer Society, Los Alamitos (2007)
4. Bajaj, S., Box, D., Chappell, D., Curbera, F., Daniels, G., Hallam-Baker, P., Hondo, M., Kaler, C., Langworthy, D., Malhotra, A., et al.: Web Services Policy Framework (WS-Policy). Version 1(2), 2003–2006 (2006)
5. America, Bank secrecy act of 1970 (1970)
6. Australia, Privacy act 1988 (1988)
7. Bücker, A.: ITS Organization IBM Corporation, Understanding SOA Security Design and Implementation. Books24x7.com (2005)
8. Nadalin, A., Kaler, C., Hallam-Baker, P., Monzillo, R., et al.: Web Services Security: SOAP Message Security 1.0 (WS-Security 2004). OASIS Standard 200401 (2004)
9. Kim, A., Luo, J., Kang, M.: Security ontology for annotating resources. In: Meersman, R., Tari, Z. (eds.) OTM 2005. LNCS, vol. 3761, pp. 1483–1499. Springer, Heidelberg (2005)
10. I. JTC, SC27/WG3. Common Criteria for Information Technology Security Evaluation (1998)
11. Khan, K.M., Han, J.: Assessing Security Properties of Software Components: A Software Engineer's Perspective. In: Han, J., Staples, M. (eds.) Proceedings of the 17th Australian Software Engineering Conference (ASWEC 2006), Sydney, Australia, pp. 199–208. IEEE Computer Society Press, Los Alamitos (2006)
12. Meier, J., Mackman, A., Dunner, M., Vasireddy, S.: Building Secure ASP .NET Applications: Authentication, Authorization, and Secure Communication. Microsoft Patterns and Practices. Microsoft Corporation, pp. 354–362 (2002)
13. Steel, C., Nagappan, R., Lai, R.: Core Security Patterns. Prentice-Hall, Englewood Cliffs (2006)
14. McIntosh, M., Gudgin, M., Morrison, K., Barbir, A.: Basic Security Profile Version 1.0. WS-I Standard 30 (2007)
15. Kaler, C., Nadalin, A., et al.: Web Services Security Policy Language (WS-SecurityPolicy) (2005)
16. Akkiraju, R., Farrell, J., Miller, J., Nagarajan, M., Schmidt, M., Sheth, A., Verma, K.: Web Service Semantics-WSDL-S, W3C Member Submission (2005)
17. Bajaj, S., Box, D., Chappell, D., Curbera, F., Daniels, G., Hallam-Baker, P., Hondo, M., Kaler, C., Malhotra, A., Maruyama, H., et al.: Web Services Policy Attachment (WS-PolicyAttachment), W3C Member Submission (April 2006)
18. Keller, A., Ludwig, H.: The WSLA Framework: Specifying and Monitoring Service Level Agreements for Web Services. Journal of Network and Systems Management 11(1), 57–81 (2003)
19. Lamanna, D., Skene, J., Emmerich, W.: SLAng: A Language for Defining Service Level Agreements. In: Proc. of the 9th IEEE Workshop on Future Trends in Distributed Computing Systems-FTDCS, pp. 100–106 (2003)
20. Orriens, B., Yang, J., Papazoglou, M.P.: A Framework for Business Rule Driven Web Service Composition. In: Jeusfeld, M.A., Pastor, Ó. (eds.) ER Workshops 2003. LNCS, vol. 2814, pp. 52–64. Springer, Heidelberg (2003)

21. Horrocks, I., Patel-Schneider, P., Boley, H., Tabet, S., Grosof, B., Dean, M.: SWRL: A Semantic Web Rule Language Combining OWL and RuleML, W3C Member Submission (2004)
22. Damianou, N., Dulay, N., Lupu, E., Sloman, M.: The Ponder Policy Specification Language. In: Sloman, M., Lobo, J., Lupu, E.C. (eds.) POLICY 2001. LNCS, vol. 1995, pp. 18–38. Springer, Heidelberg (2001)
23. Uszok, A., Bradshaw, J., Jeffers, R., Suri, N., Hayes, P., Breedy, M., Bunch, L., Johnson, M., Kulkarni, S., Lott, J.: Kaos policy and domain services: toward a description-logic approach to policy representation, deconfliction, and enforcement. In: Proceedings of 4th International Workshop on Policies for Distributed Systems and Networks (POLICY 2003), June 2003, pp. 93–96 (2003)
24. Phan, T., Han, J., Schneider, J.-G., Ebringer, T., Rogers, T.: A Survey of Policy-Based Management Approaches for Service Oriented Systems. In: Hussain, F.K., Chang, E. (eds.) Proceedings of the 19th Australian Software Engineering Conference (ASWEC 2008), Perth, Australia, pp. 392–401. IEEE Computer Society Press, Los Alamitos (2008)
25. Wada, H., Suzuki, J., Oba, K.: A Model-Driven Development Framework for Non-Functional Aspects in Service Oriented Architecture. International Journal of Web Services Research 5(4), 1–31 (2008)
26. Nakamura, Y., Tatsubori, M., Imamura, T., Ono, K.: Model-Driven Security based on a Web Services Security Architecture. In: Proceedings of International Conference on Services Computing, July 2005, pp. 7–15 (2005)