CHAPTER 5

DYNAMIC TRANSACTION AWARE
WEB SERVICES SELECTION

In view of frequent possibility of failures in the internet environment where any given composed service is executed, reliability of the composed service must be ensured. The reliability of the composed service is determined by the behavioural or transactional properties of component services. The component services for each activity of the workflow must be selected based on their behaviour so that their execution results in a consistent termination. Service selection must happen at run-time in order to consider the services available in the service registry at the time of execution. Towards this need, a dynamic transaction aware web service selection approach is proposed in this thesis that considers cancelable services as well, in addition to the traditional categories of services.

A model for web service composition and execution is proposed that enables dynamic service selection in subsection 5.1. The valid transactional properties of a composite service comprising of component services including cancelable services, that are composed either in sequence or in parallel are derived and formally verified in subsection 5.2. The proposed selection approach has two major features: first one is to assign transactional requirements to yield a composition with a valid transactional property; second is to assign transactional properties for workflow activities based on the transactional property of the composition. These features are
illustrated in subsections 5.3 and 5.4. The approach for including transactional capabilities in WSDL service descriptions is discussed in subsection 5.5. The algorithm for selecting the services at runtime by matching transactional capabilities and requirements is presented in subsection 5.6. The approach for including transactional guarantees as SLAs in contracts to ensure the trustworthiness of service providers is described in subsection 5.7. The performance of the proposed selection approach is assessed in subsection 5.8. Lastly, subsection 5.9 summarises the salient features and advantages of the proposed selection approach.

5.1 DYNAMIC INTERLEAVED TEMPLATE BASED APPROACH

There have been various approaches for composition of services wherein the services are discovered either at design-time (static) or at runtime (dynamic). If the behaviour of service is ignored while selecting the services involved in a composition, it may affect the overall behaviour of the composite service and might result in an inconsistent termination. Hence, while composing the services, only the services with appropriate behaviour that will result in a reliable execution of the overall composition must be chosen.

The behavioural or transactional properties of component services determine the reliability of the composed service. When the services are selected and composed statically, during execution of services, newer versions of services and services provided by new service developers that were deployed after composition may not be considered. Moreover, the services discovered at design-time may not be available during the execution of the application. Such static composition is suitable only when the functional requirements of the component services are completely known at design-time and do not often vary. However, in Business-to-Business (B2B)
and Business-to-Consumer (B2C) application scenarios, users would like to specify requirements based on the outcome of the previous service execution. Moreover, the business policies change frequently depending on the business trends and growth. Hence, services must be discovered dynamically at runtime.

Interleaved approach for web services composition and execution (Agarwal et al 2008) supports dynamic composition and it interleaves selection and execution of component services of a composite service. However, interleaved approach is not suitable for applications where the order in which the component services are composed is important. In enterprise applications, the business is performed in a predetermined manner. Hence, the template based composition is more suitable wherein the control flow among the business activities is represented. The existing works on dynamic composition address dynamic composition of web services at runtime but do not allow the interleaving of composition and execution steps. Existing works on interleaved composition and execution do not follow a specific ordering among business activities.

Existing language standard for composition - Business Process Execution Language (BPEL) (Jordan & Evdemon 2009) - supports only design-time composition and requires all the service parameter values to be supplied at the starting of the execution itself. Moreover, it can statically compose only SOAP-based services and it is not suitable for composing RESTful services due to its tight coupling with WSDL. Hence, a model for dynamic composition that involves dynamic service selection, interleaving the steps of composition and execution, based on a predefined workflow template is necessary. Towards this need, a hybrid model namely, Dynamic Interleaved Template Model (DILT) is proposed in this research that dynamically interleaves selection / composition and execution steps of
services in a composite business process, based on a predefined control flow, represented in the form of a workflow template.

The workflow template is an XML document that contains a set of business activities to be carried out for a particular application scenario and the control flow among the activities is represented using workflow patterns (Aalst et al 2003). The workflow template is validated with respect to a workflow schema which is a well-formed structure defined using the XML Schema Definition Language (XSDL) (Shudi et al 2007) and is shown in Appendix 1. The workflow patterns such as Sequence, AND-Split, AND-Join, XOR-Join, and XOR-Split have been considered in the schema. The workflow template is designed by a business analyst based on the manner in which the business needs to be performed. The workflow template is the abstract workflow which is converted into an executable workflow during the process of dynamic composition. The proposed DILT model shown in Figure 5.1 involves dynamic selection and execution of services. These steps are interleaved for each activity present in the template. Hence, the response of the previous service execution is useful for deciding the selection of services from the registry, for a subsequent activity in the workflow template.

![Diagram](image)

**Figure 5.1** DILT model for web service composition and execution
The activities of the workflow are either simple or complex (involving another workflow), resulting in a hierarchy of workflows spanning multiple organizations. The workflow that represents the business scenario is generated by business administrators as a template and is stored in the workflow repository. The business policies are obtained from administrators from which the transactional requirements of activities of the principal workflow are derived. While assigning the transactional requirements of each activity, they are restricted in such a way that when a matching service is discovered, its composition will certainly result in a TCWS. Web services whose definitions are embedded with their transactional properties are deployed by service developers as Tx-WSDL, in the service registry. Given a target transactional property for a composite web service, the proposed selection approach uses Tx-Selection middleware service that detects the required transactional properties of each underlying block of the workflow and for each single web service involved. Tx-Selection also discovers at runtime, suitable web services from the service registry corresponding to each activity of the workflow, by matching the functional and behavioural capabilities and requirements.

The transactional capabilities of the service as designed by the provider are advertised in Tx-WSDL and these descriptions are used for selecting the service. However, if the service does not exhibit the behaviour as described in Tx-WSDL, the reliable execution may not be a reality. Therefore, the transactional guarantees must be recorded as Service Level Agreements (SLAs) in a contract between the consumer and the provider. Mutually agreed contracts are helpful to ensure the trustworthiness of the provider and the provider is bound to ensure the guaranteed values, as the penalty for a violation can also be recorded in the contract. Hence, as soon as a service is discovered, the contract containing the functional and transactional guarantees is generated.
5.2 DERIVING TYPES OF TCWS AND ITS FORMAL VERIFICATION

The transactional property of a TCWS depends on the transactional properties of its component services. It is essential to derive the transactional property of a CWS based on the transactional properties of its component services in order to check whether the composition is reliable or in other words whether the composition will result in a TCWS. A set of theorems has been formulated in this section to derive the transactional property of a composite service, based on the transactional properties of its component services.

For simplicity, a TCWS comprising of two component services $S_1$ and $S_2$, which may be simple or composite are considered. These component services can either be executed sequentially ($S_1;S_2$) or concurrently ($S_1||S_2$). The theorems governing the transaction property of a TCWS composed in sequence and in parallel are discussed below.

**Theorem 1:** A TCWS composed in sequence, $S_1;S_2$ or in parallel $S_1||S_2$ is fully recoverable iff both $S_1$ and $S_2$ have $\{cpccr\}$ property.

**Proof:** A fully recoverable TCWS is compensatable, retrievable, and cancelable. A TCWS is compensatable only when all its component services are compensatable. All the component services of a cancelable TCWS must be cancelable (Definition 2). Similarly, a TCWS is retrievable when all its component services are retrievable (Definition 3). Hence, a TCWS has a combination of three properties $\{cpccr\}$, only when all the component services have each one of these properties.

**Corollary 1:** When $S_1$ has the property $\{cpccr\}$, a sequentially composed TCWS $S_1; S_2$ will have the same property as $S_2$. 
Proof: Service $SI$ is compensatable as it has the property $\{cpccr\}$. When it is sequentially composed with $S2$ that is pivot or atomic, the resulting properties of TCWS include $\{a\}$, as it cannot be semantically undone after its completion. When $SI$ is composed in sequence with $S2$ whose properties include $\{cp\}$, the properties of the resulting TCWS also include $\{cp\}$, as both $SI$ and $S2$ are compensatable. The properties of the TCWS $SI;S2$ include $\{r\}$, $\{cc\}$, or $\{ccr\}$ if $S2$ includes $\{r\}$ (Definition 3), $\{cc\}$ (Definition 2), or $\{ccr\}$ (Definition 2 and 3) respectively.

Corollary 2: When $SI$ has the property $\{cpccr\}$, a concurrently composed TCWS $SI||S2$ will have the same property as $S2$ only when $S2$ is compensatable.

Proof: Service $SI$ is cancelable as it has property $\{cpccr\}$. When it is concurrently composed with $S2$, which is non-compensatable, it will result in non-atomic composition as $S2$ cannot be undone upon cancellation of $SI$. When $SI$ is composed in parallel with $S2$ whose properties include $\{cp\}$, the properties of the resulting TCWS also include $\{cp\}$ as both $SI$ and $S2$ are compensatable. The properties of the TCWS $SI||S2$ include $\{r\}$, $\{cc\}$, or $\{ccr\}$ if $S2$ includes $\{r\}$ (Definition 3), $\{cc\}$ (Definition 2), or $\{ccr\}$ (Definitions 2 and 3).

Theorem 2A: A sequentially composed TCWS $SI;S2$ is atomic iff one of the following conditions hold.

1. If $SI$ has the property $\{cp\}$, $S2$ must have a property from $\{p/a, pr/ar, pcc/acc, pccr/accr\}$.

2. If $SI$ has the property $\{cpr\}$, $S2$ must have a property from $\{p/a, pcc/acc\}$.
3. If \( S1 \) has the property \( \{cpcc\} \), \( S2 \) must have a property from \( \{p/a, pr/ar\} \).

4. If \( S1 \) has the property \( \{cpccr\} \), \( S2 \) must have the property \( \{p/a\} \).

5. If \( S1 \) has a property from \( \{p/a, pcc/acc\} \), \( S2 \) must have a property from \( \{pr/ar, cpr\} \).

**Proof.** In case of failure or cancellation of \( S2 \), for the TCWS \( S1;S2 \) to become atomic, \( S1 \) must be compensatable as \( S1 \) must be semantically undone. When \( S1 \) has the property \( \{cp\} \), \( S2 \) may either be a pivot or an atomic service resulting in an atomic sequential composition. Hence, the possible transactional properties that can be assigned to \( S2 \) are any of the properties from \( \{p/a, pcc/acc, pr/ar, pccr/accr\} \). When \( S1 \) has the property \( \{cpr\} \), \( S2 \) can be a pivot or atomic service but cannot include the retrievable property as this would make \( S1;S2 \) retrievable (Definition 3). When \( S1 \) has the property \( \{cpec\} \), \( S2 \) must be a pivot or an atomic service and cannot include the cancelable property as this would make \( S1;S2 \) cancelable (Definition 2). When \( S1 \) is fully recoverable, \( S2 \) must have the property \( \{p/a\} \) to make the sequential composition atomic (Corollary 1). If \( S2 \) is guaranteed to succeed, i.e. retrievable and non-cancelable, \( S1 \) need not be compensatable, since there is no need to compensate \( S1 \). Thus, \( S1 \) can be a pivot or an atomic non-retrievable service and the resulting \( S1;S2 \) will also be an atomic service (Definition 3).

**Theorem 2B:** A TCWS composed in parallel \( S1 || S2 \) is atomic iff the following condition holds.

- If either \( S1 \) or \( S2 \) has a property from \( \{p/a, pcc/acc\} \), the other one should have the \( \{cpr\} \) property.
**Proof:** A parallel composition $SI || S2$ is atomic only when one of the component services, say $SI$ is a pivot/atomic or cancelable service and properties of the other service, say $S2$, includes $\{cp\}$. Moreover, $S2$ should not fail or get cancelled, since $SI$ is non-compensatable (Definitions 2 and 3). Hence, the only possible transaction property that can be assigned to $S2$ is $\{cpr\}$.

**Theorem 3A:** A sequentially composed TCWS $S1; S2$ is atomic retrievable iff one of the following conditions hold.

1. If $SI$ has a property from $\{pr/ar, pccr/accr\}$, then $S2$ must have a property from $\{pr/ar, cpr\}$.

2. If $SI$ has the property $\{cpr\}$, then $S2$ must have a property from $\{pr/ar, pccr/accr\}$.

3. If $SI$ has the property $\{cpccr\}$, then $S2$ must have the property $\{pr/ar\}$.

**Proof:** Two retrievable services $SI$ and $S2$ that are composed in sequence result in an atomic retrievable TCWS (Definition 3). The possible properties that can be assigned to $SI$ are $\{pr/ar, cpr, pccr/accr, cpccr\}$. When $SI$ has the property $\{pr/ar\}$ or $\{pccr/accr\}$, $S2$ must be non-cancelable to achieve an atomic retrievable composition, since $SI$ is not compensatable (Definition 3). When $SI$ has the property $\{cpr\}$, $S2$ cannot be compensatable as it results in a compensatable TCWS, but it can be cancelable (Definition 3). If $SI$ has the property $\{cpccr\}$, $S2$ must have the property $\{pr/ar\}$ to make the sequential composition atomic retrievable (Corollary 1).

**Theorem 3B:** A TCWS composed in parallel, $SI || S2$ is atomic retrievable iff one of the following conditions hold.
1. Both $S_1$ and $S_2$ must have the \{pr/ar\} property.

2. If either $S_1$ or $S_2$ has the \{cpr\} property, the other one must have a property from \{pr/ar, pccr/accr\}.

**Proof:** Two retriable services, $S_1$ and $S_2$ that are composed in parallel result in atomic retriable TCWS (Definition 3). When one of the concurrently running services has the property \{pr/ar\}, it is non-compensatable. Hence, the other one must be non-cancelable and retriable, i.e it may be assigned \{pr/ar\} or \{cpr\}, to achieve an atomic retriable composition (Definition 3). When a service composed in parallel has the property \{cpr\}, the other concurrent service cannot be compensatable as it results in a compensatable TCWS but it can be cancelable, i.e. it may be assigned \{pr/ar\} or \{pccr/accr\}.

**Theorem 4:** A sequentially composed TCWS $S_1;S_2$ is atomic cancelable iff one of the following conditions hold.

1. If $S_1$ has the property \{cpcr\} then $S_2$ must have a property from \{pcc/acc, pccr/accr\}.

2. If $S_1$ has the property \{cpecr\} then $S_2$ must have the property \{pcc/acc\}.

**Proof:** Two services $S_1$ and $S_2$ that are composed in sequence result in an atomic cancelable TCWS if both $S_1$ and $S_2$ include the property \{pcc/acc\} (Definition 2). $S_1$ must be cancelable and compensatable as $S_1$ can be undone in case $S_2$ fails or gets cancelled. Thus, the only possible properties for $S_1$ are \{cpcr\} or \{cpecr\} (Definition 2). When $S_1$ has the property \{cpecr\}, $S_2$ must be cancelable (Definition 2) but should not be compensatable, since it may result in a compensatable composition. If $S_1$ has the property \{cpecr\}, $S_2$ must have the property \{pcc/acc\} to make the sequential composition atomic cancelable (Corollary 1).
Theorem 5: A TCWS composed in sequence, $S_1; S_2$ or in parallel, $S_1||S_2$ is compensatable iff one of the following conditions hold.

1. Both $S_1$ and $S_2$ must have the \{cp\} property.
2. If one of them has a property from \{cpecc, cpccr\}, the other one must have the \{cp\} property.
3. If one of them has the property \{cpr\}, the other one must have a property from \{cp, cpecc\}.

Proof: In a compensatable sequential or parallel composition, all the component services must be compensatable. One of the component services, say $S_1$, can be assigned any property from \{ep, cpr, cpecc, cpccr\}. When $S_1$ has the property \{cp\}, $S_2$ can be assigned any of the four properties from \{cp, cpr, cpecc, cpccr\} to result in a compensatable $S_1; S_2$ or $S_1||S_2$. When $S_1$ has the property \{cpr\}, $S_2$ must be a compensatable service but cannot include the retriable property, since it results in a retriable $S_1; S_2$ or $S_1||S_2$ (Definition 3). When $S_1$ has the property \{cpecc\}, $S_2$ must be a compensatable service and cannot include the cancelable property as this will result in a cancelable $S_1; S_2$ or $S_1||S_2$ (Definition 2). When $S_1$ is fully recoverable, $S_2$ must have the property \{cp\} to make the sequential or parallel composition compensatable (Corollaries 1 and 2).

Theorem 6: A TCWS composed in sequence $S_1; S_2$ or in parallel $S_1||S_2$ is compensatable retrievable iff one of the following conditions hold.

1. Both $S_1$ and $S_2$ must have the \{cpr\} property.
2. If either $S_1$ or $S_2$ has the property \{cpccr\}, the other one must have \{cpr\}.

Proof: A sequentially or concurrently composed TCWS is compensatable and retrievable only when its component services are both compensatable and
retirable. Thus, the possible properties that can be assigned to one of the services are \{\text{cpr}\} or \{\text{cpccr}\}. When \( SI \) has the property \{\text{cpr}\}, \( S2 \) should be compensatable and retirable to obtain a TCWS \( SI;S2 \) or \( SI||S2 \) with the property \{\text{cpr}\}. Hence, the possible properties that can be assigned to \( S2 \) are \{\text{cpr}\} or \{\text{cpccr}\}. When \( SI \) has the property \{\text{cpccr}\} (Definition 3), composing it either sequentially or concurrently with \( S2 \) having the property \{\text{cpr}\} will yield a TCWS with the property \{\text{cpr}\} (Corollaries 1 and 2).

**Theorem 7:** A sequentially composed TCWS \( SI;S2 \) is cancelable retirable iff the following condition holds.

1. If \( SI \) has the property \{\text{cpccr}\} then \( S2 \) must have the property \{\text{pccr/accr}\}.

**Proof:** A TCWS sequentially composed from \( SI \) and \( S2 \) is atomic, cancelable, and retirable only when both \( SI \) and \( S2 \) are cancelable (Definition 2) and retirable (Definition 3) and \( SI \) is compensatable. The only possible transactional property for \( SI \) is \{\text{cpccr}\} and \( SI;S2 \) would be cancelable retirable \{\text{ccr}\}, provided \( S2 \) is assigned the property \{\text{cer}\} (Corollary 1).

**Corollary 3:** A TCWS composed in parallel, \( SI||S2 \) cannot be either cancelable or cancelable retirable.

**Proof:** An atomic cancelable or atomic cancelable retirable TCWS \( SI||S2 \) requires both the component services cancelable (Definition 2). However, two concurrent cancelable services will not result in an atomic composition.

**Theorem 8:** A TCWS composed in sequence \( SI;S2 \) or in parallel \( SI||S2 \) is compensatable cancelable iff one of the following conditions hold.

1. Both \( SI \) and \( S2 \) must have \{\text{cpcc}\} property.

2. If either \( SI \) or \( S2 \) has the property \{\text{cpccr}\}, the other one must have \{\text{cpcc}\}. 

Proof: A sequentially or concurrently composed TCWS is compensatable and cancelable only when its component services are both compensatable and cancelable. Thus, the possible properties that can be assigned to one of the services are \{cpc\} or \{cpcr\}. When \( S_1 \) has the property \{cpc\}, \( S_2 \) should be compensatable and cancelable (Definition 2) to obtain a TCWS \( S_1; S_2 \) or \( S_1 \parallel S_2 \) with the property \{cpc\}. Thus, the possible properties that can be assigned to \( S_2 \) are \{cpc\} or \{cpcr\}. When \( S_1 \) has the property \{cpcr\}, composing \( S_2 \) with the property \{cpc\} either sequentially or in parallel will yield a TCWS with the property \{cpc\} (Corollaries 1 and 2).

The transactional properties of a TCWS composed either in sequence or in parallel from two component services (single or composite) as derived using these theorems are summarized in Table 5.1. The invalid compositions that do not guarantee a consistent outcome are marked X in the table.

5.3 DERIVING TRANSACTIONAL REQUIREMENTS FOR RELIABLE COMPOSITION

For the given workflow activity, all the Tx-WSDL documents of available services are discovered from the registry. Each of these Tx-WSDL documents is parsed to extract the transactional capabilities. The transactional requirements extracted from business policies along with the functional requirements are then matched with the capabilities of the services to discover a suitable service. Though the business policies represent the transactional requirements of some of the services, the recoverability level plays an important role in selecting a TWS. The recoverability level describes the degree to which a service may be recovered in case of a failure or an interruption. As discussed in Section 4.2, TWSs are grouped into different levels of recoverability based on their recovery cost. The recoverability levels and the corresponding transactional properties of a TWS/TCWS are tabulated in the increasing order of their recovery cost, in Table 5.2.
Table 5.1 Transactional properties of TCWS

<table>
<thead>
<tr>
<th>S1</th>
<th>S2</th>
<th>S1;S2</th>
<th>S1$S2</th>
</tr>
</thead>
<tbody>
<tr>
<td>p/a</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>pcc/acc</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>pr/ar</td>
<td>a</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>pccr/acc</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>cp</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>cpec</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>cpr</td>
<td>a</td>
<td>a</td>
<td></td>
</tr>
<tr>
<td>cpcr</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>p/a</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>pcc/acc</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>pr/ar</td>
<td>a</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>pccr/acc</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>cp</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>cpec</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>cpr</td>
<td>a</td>
<td>a</td>
<td></td>
</tr>
<tr>
<td>cpcr</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>p/a</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>pcc/acc</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>pr/ar</td>
<td>ar</td>
<td>a</td>
<td></td>
</tr>
<tr>
<td>pccr/acc</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>cp</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>cpec</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>cpr</td>
<td>ar</td>
<td>a</td>
<td></td>
</tr>
<tr>
<td>cpcr</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>p/a</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>pcc/acc</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>pr/ar</td>
<td>ar</td>
<td>a</td>
<td></td>
</tr>
<tr>
<td>pccr/acc</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>cp</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>cpec</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>cpr</td>
<td>ar</td>
<td>ar</td>
<td></td>
</tr>
<tr>
<td>cpcr</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>p/a</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>pcc/acc</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>pr/ar</td>
<td>ar</td>
<td>a</td>
<td></td>
</tr>
<tr>
<td>pccr/acc</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>cp</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>cpec</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>cpr</td>
<td>ar</td>
<td>ar</td>
<td></td>
</tr>
<tr>
<td>cpcr</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>

Table continued:

<table>
<thead>
<tr>
<th>S1</th>
<th>S2</th>
<th>S1;S2</th>
<th>S1$S2</th>
</tr>
</thead>
<tbody>
<tr>
<td>p/a</td>
<td>a</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>pcc/acc</td>
<td>a</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>pr/ar</td>
<td>a</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>pccr/acc</td>
<td>a</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>cp</td>
<td>cp</td>
<td>cp</td>
<td></td>
</tr>
<tr>
<td>cpec</td>
<td>cp</td>
<td>cp</td>
<td></td>
</tr>
<tr>
<td>cpr</td>
<td>cp</td>
<td>cp</td>
<td></td>
</tr>
<tr>
<td>cpcr</td>
<td>cp</td>
<td>cp</td>
<td></td>
</tr>
<tr>
<td>p/a</td>
<td>a</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>pcc/acc</td>
<td>acc</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>pr/ar</td>
<td>a</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>pccr/acc</td>
<td>acc</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>cp</td>
<td>cp</td>
<td>cp</td>
<td></td>
</tr>
<tr>
<td>cpec</td>
<td>cp</td>
<td>cp</td>
<td></td>
</tr>
<tr>
<td>cpr</td>
<td>cp</td>
<td>cp</td>
<td></td>
</tr>
<tr>
<td>cpcr</td>
<td>cp</td>
<td>cp</td>
<td></td>
</tr>
<tr>
<td>p/a</td>
<td>a</td>
<td>a</td>
<td></td>
</tr>
<tr>
<td>pcc/acc</td>
<td>a</td>
<td>a</td>
<td></td>
</tr>
<tr>
<td>pr/ar</td>
<td>ar</td>
<td>a</td>
<td></td>
</tr>
<tr>
<td>pccr/acc</td>
<td>ar</td>
<td>ar</td>
<td></td>
</tr>
<tr>
<td>cp</td>
<td>cp</td>
<td>cp</td>
<td></td>
</tr>
<tr>
<td>cpec</td>
<td>cp</td>
<td>cp</td>
<td></td>
</tr>
<tr>
<td>cpr</td>
<td>cp</td>
<td>cp</td>
<td></td>
</tr>
<tr>
<td>cpcr</td>
<td>cp</td>
<td>cp</td>
<td></td>
</tr>
<tr>
<td>p/a</td>
<td>a</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>pcc/acc</td>
<td>acc</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>pr/ar</td>
<td>ar</td>
<td>a</td>
<td></td>
</tr>
<tr>
<td>pccr/acc</td>
<td>accr</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>cp</td>
<td>cp</td>
<td>cp</td>
<td></td>
</tr>
<tr>
<td>cpec</td>
<td>cp</td>
<td>cp</td>
<td></td>
</tr>
<tr>
<td>cpr</td>
<td>cp</td>
<td>cp</td>
<td></td>
</tr>
<tr>
<td>cpcr</td>
<td>cp</td>
<td>cp</td>
<td></td>
</tr>
</tbody>
</table>
### Table 5.2  Recoverability levels

<table>
<thead>
<tr>
<th>Recoverability Level</th>
<th>Properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Pivot/Atomic {p/a}</td>
</tr>
<tr>
<td>2</td>
<td>Compensatable {cp}</td>
</tr>
<tr>
<td>3a</td>
<td>Cancelable {pcc/acc}</td>
</tr>
<tr>
<td>3b</td>
<td>CompensatableCancelable {pcc}</td>
</tr>
<tr>
<td>4a</td>
<td>Retriable {pr/ar}</td>
</tr>
<tr>
<td>4b</td>
<td>CompensatableRetriable {pcc/acr}</td>
</tr>
<tr>
<td>5a</td>
<td>CancelableRetriable {pccr/acr}</td>
</tr>
<tr>
<td>5b</td>
<td>CompensatableCancelableRetriable {cpccr}</td>
</tr>
</tbody>
</table>

The retrievable services are identified using the level of their importance. If the administrator deems a particular service to be very important, then the service should have property \{r\}. Business policies obtained from the administrator, through a questionnaire and the required recoverability level, together abstract transactional requirements or properties. The business analyst is provided with recoverability options for an activity, depending on the transactional requirement of the previous activity so that the resultant composite service when all the matching services are discovered and composed is at least atomic.

While obtaining the recoverability level of an activity as an input, it is restricted to only certain options such that it does not result in a non-atomic composition. For example, if the transaction property of a service assigned to an activity is extracted as \{cp\}, the service corresponding to next sequential activity can be assigned any possible transactional property to yield a valid TCWS. Alternatively, for a service corresponding to an activity composed in parallel with another activity having compensatable property, assigning \{p/a\}, \{pcc/acc\}, \{pr/ar\}, or \{pccr/acr\} property will result in a non-atomic composition. The non-atomic compositions shown in Table 5.1 are not desirable and hence, such unreliable compositions are avoided. For each transactional property of a service assigned to a workflow activity, the possible recoverability levels of the service to be assigned to the subsequent activity are tabulated in Table 5.3.
Table 5.3 Valid recoverability levels for deriving a TCWS

<table>
<thead>
<tr>
<th>Tx-Property</th>
<th>Recoverability level of next service</th>
<th>Composed in Sequence</th>
<th>Composed in Parallel</th>
</tr>
</thead>
<tbody>
<tr>
<td>p/a</td>
<td>4a, 4b</td>
<td>4b</td>
<td></td>
</tr>
<tr>
<td>pcc/acc</td>
<td>4b</td>
<td>4b</td>
<td></td>
</tr>
<tr>
<td>pr/ar</td>
<td>4a, 4b</td>
<td>4a, 4b</td>
<td></td>
</tr>
<tr>
<td>pccr/accr</td>
<td>4a, 4b</td>
<td>4a</td>
<td></td>
</tr>
<tr>
<td>cp</td>
<td>1, 2, 3a, 3b, 4a, 4b, 5a, 5b</td>
<td>2, 3b, 4b, 5b</td>
<td></td>
</tr>
<tr>
<td>cpcc</td>
<td>1, 2, 3a, 3b, 4a, 4b, 5a, 5b</td>
<td>2, 3b, 4b, 5b</td>
<td></td>
</tr>
<tr>
<td>cpr</td>
<td>1, 2, 3a, 3b, 4a, 4b, 5a, 5b</td>
<td>1, 2, 3a, 3b, 4a, 4b, 5a, 5b</td>
<td></td>
</tr>
<tr>
<td>cpccr</td>
<td>1, 2, 3a, 3b, 4a, 4b, 5a, 5b</td>
<td>2, 3b, 4b, 5b</td>
<td></td>
</tr>
</tbody>
</table>

5.4 ASSIGNING TRANSACTIONAL REQUIREMENTS BASED ON TYPE OF COMPOSITION

Whenever the current workflow contains a component that by itself is a composite process, it involves another workflow comprising of multiple activities. The middleware service, \textit{Tx-Selection} assigns transactional requirements for the component activities of the composite workflow, in such a way that the required transactional property of the composition is satisfied using algorithm 1. The algorithm takes a workflow, \textit{WF} and its transaction property, \textit{Reqd_TxP} as input. In a \textit{WF}, the transactional property of any component activity depends on the transaction property of the overall \textit{WF} and the transactional property of the composition of all the previous activities. The transactional requirements are assigned from left to right in sequential patterns and from top to bottom in split patterns. A workflow of \textit{n} activities can be visualized as a composition of a composite process involving \textit{n-1} activities and \textit{n^{th}} activity as illustrated in Figure 5.2.

At any point of time, composition involves two services. Hence, the workflow is structured internally as a binary tree. In each node of the tree representing a workflow activity, the type of the activity (\textit{Type}), its transactional requirement (\textit{TxP}), and the name of the service to be selected (\textit{sName}) are stored.
(a) Sequential composition  
(b) Parallel composition

Figure 5.2 Transactional property assignments - step $n$

The different types of activity are simple, sequence, parallel, and composite. The simple activities are non-composite whereas composite activities are represented by workflows. Activity types such as sequence and parallel represent a sequentially composed process and a concurrently composed process respectively. The $Set_{Seq}$ ($Set_{Par}$) in the algorithm represents a set of properties that can be assigned to the left composition or activity of $WF$ that is sequentially (concurrently) composed. The function $any(set)$ returns any one of the transactional properties from set. Based on the required transactional property of a TCWS, the members of the $Set_{Seq}$ and $Set_{Par}$ are different in accordance with Table 5.1.

The algorithm processes each activity of the $WF$ and terminates when the transactional property of the $WF$ is assigned. When an activity of a $WF$ is another $WF$ that is either sequentially or concurrently composed, the algorithm is invoked recursively. For a simple or composite activity, any of the transactional properties from the respective set is assigned. Based on the transactional property assigned to the left composition or activity, a suitable property is assigned to the right activity that would result in the required property for their composition. For each type of transaction property, the algorithm has three parts as enumerated below:
1. Assign transaction property to the left composition or activity of current workflow

2. Assign transaction property to the right activity of current workflow based on step 1

3. Assign transaction property to the current workflow

---

**Algorithm 5.1 AssignTxP(WF, Req_TxP)**

**Input**
- WF //Workflow
- Req_TxP //Required transactional property of WF

**Output**
- TCWS //WF with transactional properties assigned to all
  //of its activities

**BEGIN**

Initialize Set_Seq

Initialize Set_Par

**WHILE** (WF.Type ≠ “simple” AND WF.Type ≠ “composite” AND
WF.TxP == NULL)

//Invoke AssignTxP() recursively with any property from respective set

**IF** (WF.left.Type == “sequence”) **THEN**

AssignTxP(WF.left, any(Set_Seq))

**ELSIF** (WF.left.Type == “parallel”) **THEN**

AssignTxP(WF.left, any(Set_Par))

**ELSIF** (WF.left.Type == “simple” OR WF.left.nodeType ==
“composite”) **THEN**

//Assign any property from respective set to the left composition
//or activity of WF

**IF** (WF.Type == “sequence”) **THEN**

WF.left.TxP ← any(Set_Seq)
ELSIF (WF.Type == “parallel”) THEN
    WF.left.TxP ← any(Set_Par)
ENDIF

ENDIF

//Initialize rightTxP based on the property assigned to left composition or activity
Initialize rightTxP
IF (WF.right.Type == “parallel” OR WF.right.Type == “sequence”) THEN
    //Invoke AssignTxProp() recursively with any property from rightTxP
    AssignTxP(WF.right, any(rightTxP))
ELSE
    //Assign any property from rightTxP to the right activity of WF
    WF.right.TxP ← any(rightTxP)
ENDIF

WF.TxP ← Reqd.TxP //Assign TxP to WF

ENDWHILE
END

In order to illustrate working of the algorithm, its trace for the child workflow of composite activity InsureVehicle of VPRS depicted in Figure 1.4 is pictorially shown in Figure 5.3. The workflow is represented as a binary tree shown in Figure 5.3. When the algorithm is invoked as AssignTxP(InsureVehicleWF, “ar”) with transactional requirement atomicretriable, the transactional properties assigned to simple activities and the contents of the intermediate sets Set_Seq and Set_Par are shown. The sets are initialized using Table 5.1 and one of the properties in the set is randomly assigned to respective activity node.
Figure 5.3 Trace of Algorithm 5.1 for InsureVehicle workflow of VPRS

Algorithm 5.1 involves traversing all the nodes of the binary tree used to represent the workflow $WF$. The workflow activities (say $n$ of them) represent the leaf nodes of the binary tree. The binary tree which is constructed by the algorithm has nodes of degree 0 (non composite services in leaf nodes) or 2 (composite services composed from two component services) only. Thus, the total number of nodes in the binary tree is $n + (n+1)$ at a maximum. Hence, the worst case time complexity of the algorithm would be $O(n)$.

The working of Algorithm 5.1 can be clearly understood using a finite automaton depicted in Figure 5.4 showing possible TCWSs. Each transactional property is modeled as a terminal state and state 1 is the initial one. When one of the terminal states is reached, a TCWS is obtained. A transition from state 1 to state 2 on input indicates that composition of a state 1
service with another service having transactional property input, results in a state2 TCWS. The input also indicates whether the composition is performed in sequence or in parallel. For example, let us consider a retrievable composite service CS1 consisting of (S1;S2)||S3. This composition can be viewed as CS2||S3, where the composite service CS2 comprises of S1;S2. The state {ar}
can be reached from the states \( \{ar\} \ \{cpr\} \), and \( \{accr\} \) on concurrent composition as shown in Figure 5.4. One of these properties, say \( \{cpr\} \) is assigned to CS2. The transition from the state \( \{cpr\} \) to the state \( \{ar\} \) (required property of CSI) happens on input which is either \( \{pr/ar\} \) or \( \{pccr/accr\} \). Hence, S3 is assigned one of these two properties, say \( \{pccr\} \). Two transitions lead to \( \{cpr\} \) (the required property of CS2) on sequential composition from states \( \{cpcr\} \) and \( \{cpr\} \). Among these properties, \( \{cpcr\} \) is randomly assigned for S1. The transition from state \( \{cpcr\} \) leading to state \( \{cpr\} \) happens on input \( \{cpr\} \). Hence, S2 must be assigned transactional property \( \{cpr\} \). Thus, the recursive algorithm works on composition of two services at any time. Therefore, it is observed that the binary tree representation of the workflow is suitable for the approach.

5.5 SPECIFICATION OF TRANSACTIONAL CAPABILITIES IN SERVICE DESCRIPTIONS

Web services can be discovered from the service registry based on their transactional properties, only if the service descriptions include their transactional behaviour. An XML-Schema based approach is proposed for transactional description of services in this section. In web services environment, WSDL is used to describe only functional aspects of a service. The other features of the service such as QoS and transactional properties can be expressed as policies using WS-Policy. The default structure of policy assertions in WS-Policy does not allow expression of transactional properties. Hence, an XML schema is defined to describe the structure of transactional policies with possible transactional properties. Using this schema, transactional policies namely Tx-Policies are defined in WS-Policy.

The schema for Tx-Policies is shown in Appendix 4. Tx-Policies are defined to specify the transactional properties of a service according to the schema defined above. The defined Tx-Policies need to be integrated along
with WSDL to generate transactional service definitions namely, Tx-WSDL. Such transactional service definitions are necessary for matching the transactional capabilities with the requirements while selecting the web services. The Tx-WSDL document of an order processing service namely BrightVolvoEnquiryWS, in the VPRS scenario is illustrated below. BrightVolvo service provider offers the order processing service with compensatable property.

```xml
<definitions xmlns:xsd="http://www.w3.org/2001/XMLSchema"
             xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd"
             xmlns:TxPolicy="http://TxPropertySchema.xsd"
             ..........>
  <wsp:Policy wsu:Id="BrightVolvoEnquiryPolicy">
    <wsp:ExactlyOne>
      <TxPolicy:TxPropertyAssertion TxProperty="compensatable" />
    </wsp:ExactlyOne>
  </wsp:Policy>

......

  <binding name="BrightVolvoEnquiryPortBinding" type="tns:Alpha">
    <wsp:PolicyReference URI="#BrightVolvoEnquiryPolicy"/>
    ......
  </binding>

......

</definitions>
```

The policies are defined under `wsp:Policy` element of WS-Policy using policy operators such as `ExactlyOne`, and `All`. These policies are embedded in the `definitions` element and referenced from the `binding` element.
of WSDL. The Id of Tx-Policy is referred in the URI attribute of PolicyReference element, from WS-PolicyAttachment (Siddharth et al 2006) specification. A user-defined namespace TxPolicy is defined to hold all transactional property references which are defined in TxPropertySchema.

5.6 SERVICE SELECTION BY MATCH-MAKING

After obtaining a TCWS using Algorithm 5.1, the required transactional properties of all the workflow activities are known. These requirements must be matched against the capabilities of services that are deployed in the service registry at run-time. For the given workflow activity, all Tx-WSDL documents of available services are discovered from the registry. Each of these Tx-WSDL documents is parsed by WSSelectTxProp() as shown in Algorithm 5.2 to extract the transactional capabilities. The transactional requirements, bReq[ ] are then matched with the capabilities of the services and the service endpoints of suitable services are returned as output.

---

**Algorithm 5.2 WSSelectTxProp()**

**INPUT**
- TxWSDL //Transactional service definitions obtained from
  //service registry
- wActivity //Workflow activity
- bReq[ ] //Transactional requirements for the wActivity

**OUTPUT**
- Matched[s] //URL of TxWSDL documents matching bReq[ ]

**var**
- string: offeredTxProp[ ] //Transactional capabilities extracted from
  //TxWSDL document
- string: element[ ] //Elements of TxWSDL document of a service
  //suitable for wActivity
- int: i, j, k, m, n
- boolean: match
BEGIN
  i ← 1
  WHILE (more TxWSDL documents corresponding to w:Activity) do
    Read elements of TxWSDL document and store into element[ ]
    k ← 1
    FOR {j ← 1 to element.length}
      IF (element[j] == “TxPropertyAssertion”) THEN
        offeredTxProp[k] ← getAttributeValue(“TxPropertyAssertion”,
                                              “TxProperty”)
      //getAttributeValue() returns the value of attribute TxProperty
      //in element TxProperty.Assertion
      k ← k + 1
      ENDIF
  ENDFOR
  REPEAT
    match ← FALSE
    n ← 1
    FOR (m ← 1 to offeredTxProp.length)
      IF (breq[n] == offeredTxProp[m]) THEN
        match ← TRUE
      ENDIF
    ENDFOR
    n ← n + 1
    UNTIL (n ≤ breq.length AND match == TRUE)
  IF (match == TRUE) THEN
    matched[i] ← URL(TxWSDL)
    i ← i + 1
  ENDIF
ENDWHILE
END
5.7 TRANSACTIONAL CONTRACTS

Service contracts are essential to ensure that the selected service provider offers the service at the agreed service levels. It is proposed to extend WS-Agreement (Andrieux et al. 2007), a standard for contract specification to include transactional agreements resulting in Transactional Contracts known as Tx-Contracts. WS-Agreement uses XML language to establish a contract between a service provider and a consumer. In WS-Agreement standard, GuaranteeTerm is used to represent assurances to the service consumer. It contains elements such as Service Scope, Qualifying Condition, Service Level Objective, and Business Values List. A service level objective is expressed as a target for a Key Performance Indicator (KPI) such as average response time, completion time, and availability associated with a service. The KPIName and Target elements of KPITarget represent the name of the key performance indicator and the target value respectively. The design of KPI is outside the scope of WS-Agreement specification. Hence, in order to record the transactional guarantees of a service in the contract resulting in a Tx-Contract, a new element KPITransactionTarget is introduced. The allowed values for a Target element under KPITransactionTarget are restricted to valid transactional properties of a TCWS. The extended schema of WS-Agreement is shown in Appendix 5.

Contract templates facilitate and simplify the matchmaking process and therefore allow a more dynamic form of establishing partnerships. Based on the extended schema for the transactional contract, a template is generated and kept in a template library. The template consists of three sections namely name, context, and terms. The name section is optional which contains the name of the agreement. The agreement context provides information about the involved parties in the contract. The terms section comprises of one or more guarantee terms. Service level objective is included under each guarantee term, to represent
key performance indicators such as transaction properties. The structure of the
template for transactional contract is depicted in Figure 5.5.

Tx-Contracts are generated whenever there is a requirement for a
service consumer to use a third-party service and a suitable provider is
identified and bound. In the life cycle of contract generation, negotiation
happens to reach a mutual agreement after identifying the suitable services.
However, when services are selected based solely on transactional properties
that are non-numeric, negotiation is not required. The availability of the
services is ensured by binding the abstract services with the respective
concrete services and then the Tx-Contracts are dynamically generated. A
middleware service for Tx-Contract generation *Tx-ContractBuilder* has been
designed and deployed. The service accepts the values of all the elements in
the contract template as input parameters. The input values are filled against
the appropriate elements while generating the contract.

![Figure 5.5 Template for transactional contract](image-url)
The service execution must be monitored to check whether the guarantees recorded in the contract are ensured. In case a violation is detected, suitable corrective action must be taken. However, runtime monitoring of services is not in the scope of this thesis.

5.8 PERFORMANCE ANALYSIS OF SELECTION APPROACH

In order to evaluate the performance of the proposed selection approach, experiments were conducted by implementing it on a 2.4 GHz Intel Core2 Duo processor machine with 4 GB RAM, operating with Ubuntu, using J2EE middleware and Netbeans IDE. The clients, domain services of different service providers and middleware services are deployed in different machines connected through an intranet. In order to test the heterogeneity among the WS, the services have been created with application servers such as Glassfish, Apache Tomcat 6.0, JBoss Application Server 5.1, database servers like MySQL 5.1, Derby, and operating systems like Windows and Linux. OpenUDDI and NovellUDDI browsers have been used to implement the service registry. Three experiments were conducted with the following objectives:

1. To observe the performance of the proposed dynamic selection algorithm by increasing the service registry size as well as the workflow size.

2. To find the impact of the transactional property or type of the selected service on the performance of the proposed selection algorithm.

3. To compare the proposed run-time selection approach with design-time selection approach by adapting these approaches in the VPRS prototype.
5.8.1 Experiment 1

Five arbitrary workflows with 5, 15, 25, 35, and 45 activities were constructed. The services for the activities of a workflow were selected from the registry at runtime according to the proposed approach of transaction-aware selection. The number of suitable services for each of the activity in the workflow was varied from 25 to 200. For each of the five workflows, the execution times were measured for 10 sample executions and the maximum values are shown in Table 5.4. The proposed selection algorithm takes only 0.443 seconds to select the services, for each of the 45 activities of a workflow from a registry containing 9000 services, with an average of 200 services per activity. It emerges that for a 72 fold increase in the service registry size (from 125 to 9000), the selection effort increases merely by a factor of 2 (0.2s to 0.4s).

Table 5.4 Performance analysis of selection algorithm

<table>
<thead>
<tr>
<th>Number of Activities</th>
<th>No. of Services per activity</th>
<th>Selection Time in ms</th>
<th>%increase</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>25</td>
<td>50</td>
<td>100</td>
</tr>
<tr>
<td>5</td>
<td>224</td>
<td>237</td>
<td>244</td>
</tr>
<tr>
<td>15</td>
<td>243</td>
<td>257</td>
<td>296</td>
</tr>
<tr>
<td>25</td>
<td>265</td>
<td>292</td>
<td>309</td>
</tr>
<tr>
<td>35</td>
<td>288</td>
<td>307</td>
<td>321</td>
</tr>
<tr>
<td>45</td>
<td>310</td>
<td>339</td>
<td>366</td>
</tr>
<tr>
<td>%increase</td>
<td>38.4</td>
<td>43.0</td>
<td>50.0</td>
</tr>
</tbody>
</table>

The execution times for each of the five workflows with varying available services per activity for selection are plotted in Figure 5.6.
Similarly, each curve in Figure 5.7 captures the execution times of all the five workflows with a specific number of services per activity.

![Graph showing execution time analysis of selection approach by varying number of services](image)

**Figure 5.6 Execution time analysis of selection approach by varying number of services**

From this experiment, it is also observed that, the execution time increases with the increase in the number of activities of a workflow as well as the number of available services for selection. Figure 5.8 shows the impact of increasing the number of available services per activity on execution time taken by the selection approach. The number of services available for selection varies uniformly from 25 to 200 for each workflow. The execution time increases by 34% on an average, for an eight fold increase (700%) in the number of available services per activity from 25 to 200. The impact of increase in the number of activities in a workflow on the execution time is shown in Figure 5.9. The number of activities uniformly varies from 5 to 45, for a given number of services per activity.
Figure 5.7  Execution time analysis of selection approach by varying number of activities

Figure 5.8  Impact of service registry size on selection time
Figure 5.9  Impact of workflow size on selection time

For instance, with 200 services available for an activity, it is observed that, for a nine fold increase (800%) in the number of activities from 5 to 45, the execution time increases only by 54%. The increase in execution time for an 800% increase in the number of activities is observed to be 48% on an average.

5.8.2  Experiment 2

A workflow with 25 activities was constructed and the performance of the selection approach that results in a TCWS with the required transactional property among the possible eight was analyzed. The experiment was carried out by increasing the number of services for each activity from 25 to 200 with 3 to 25 services per each transactional property. The experiment was repeated 10 times for each of the eight properties of TCWS and the average is computed. The computed values are shown in Table 5.5.
The execution time incurred by the proposed middleware service for selection, \textit{Tx-Selection} ranges from 319 to 349 milliseconds for 200 services per activity. This clearly demonstrates that the performance of the proposed selection approach is uniformly the same irrespective of the transactional property of the required TCWS.

\textbf{Table 5.5 Selection time analysis for various types of TCWS}

\begin{center}
\begin{tabular}{|c|c|c|c|c|c|c|c|c|}
\hline
\textbf{Transaction Property} & \textbf{No. of Services in WS Registry} & \textbf{625} & \textbf{1250} & \textbf{1875} & \textbf{2500} & \textbf{3125} & \textbf{3750} & \textbf{4375} & \textbf{5000} \\
\hline
\textbf{a} & 256 & 271 & 275 & 281 & 285 & 290 & 302 & 319 \\
\hline
\textbf{cp} & 265 & 280 & 285 & 296 & 303 & 309 & 319 & 321 \\
\hline
\textbf{cc} & 275 & 283 & 287 & 290 & 297 & 304 & 317 & 323 \\
\hline
\textbf{cpcc} & 268 & 273 & 275 & 280 & 297 & 304 & 317 & 323 \\
\hline
\textbf{r} & 271 & 292 & 295 & 300 & 308 & 311 & 321 & 326 \\
\hline
\textbf{cpr} & 275 & 281 & 288 & 290 & 312 & 327 & 331 & 338 \\
\hline
\textbf{ccr} & 278 & 273 & 280 & 283 & 311 & 328 & 336 & 344 \\
\hline
\textbf{cpccr} & 280 & 284 & 290 & 291 & 314 & 331 & 338 & 349 \\
\hline
\end{tabular}
\end{center}

\textbf{5.8.3 Experiment 3}

In order to compare dynamic and static selection approaches, both of them were adapted in VPRS prototype and throughput of the application in terms of number of requests completed per second was measured. For dynamic selection, the workflow template for the VPRS was used. The web services registry has 25 services available to choose from, for each workflow activity. For design-time selection, instead of the workflow template, a
concrete workflow where all the services were discovered statically for each of the activities was considered. The databases accessed by each of the services in VPRS contained 10000 records on an average. The experiment was performed by varying the number of simultaneous user requests from 25 to 200. Each execution was repeated ten times and the average values have been used in plotting the graphs shown in Figures 5.10 and 5.11. From Figure 5.10, it can be seen that, the average execution time of the application is found to be 40% more with dynamic selection as compared to static selection of services. In addition, from Figure 5.11, it is seen that the average throughput of VPS with dynamic selection is approximately half of that is associated with static selection of services. The degradation in performance of VPS with dynamic selection approach is due to the fact that the selection effort is a part of execution effort in contrast to the static selection process where the services are selected in design-time itself.

![Graph showing execution time vs. number of simultaneous requests](image)

Figure 5.10 Transaction aware dynamic selection versus static selection - comparison in terms of selection time
Figure 5.11 Transaction aware dynamic selection versus static selection - comparison in terms of throughput

The additional overhead associated with the run-time selection can be justified due to the advantages offered by this dynamic approach, such as considering the present state of the services and flexibility for the user to specify preferences based on the execution of previous services. Further, there is also less increase (34%) in selection time even with a substantial increase in the number of available services (700%).

5.9 SUMMARY

The transaction aware web service selection is a prerequisite to achieve reliable execution of compositions. A template based model that interleaves selection and execution steps in an interleaved manner is proposed in this thesis. The hybrid model - DILT- lends itself naturally for composing heterogeneous services.
For a given pair of services with different transactional properties, the transactional property of their sequential as well as concurrent composition is determined and formally verified. Using this result, the proposed selection approach determines transactional requirements for each workflow activity based on business policies and recoverability levels so that it results in reliable composition. Further, when any of the activities represents a workflow, based on its transactional requirements, the proposed approach determines the transactional requirements for each underlying activity. The proposed dynamic transaction aware web service selection approach is compared with existing approaches in Table 5.6.

Table 5.6 Comparison of proposed dynamic service selection approach with existing works

<table>
<thead>
<tr>
<th>Related Work</th>
<th>Features</th>
<th>Shortcomings</th>
<th>Proposed Work</th>
</tr>
</thead>
<tbody>
<tr>
<td>Li et al (2007), Liu et al (2006; 2009), Haddad et al (2010)</td>
<td>Transactional properties based selection</td>
<td>Design-time selection of services Cancelable property of services is not considered</td>
<td>Run-time selection of services Cancelable services are also considered</td>
</tr>
</tbody>
</table>

An XML schema based approach is proposed for specification of transactional service descriptions by integrating WSDL with WS-Policy. The proposed approach is compliant with the existing standard WSDL, in contrast to the existing works.
At run time, *Tx-Selection* middleware service discovers suitable service for each workflow activity based on the functional requirements specified by the user and transactional requirements determined from business policies and comparing them with the functional and transactional capabilities specified in Tx-WSDL documents of the service registry. If the transactional capabilities in the service descriptions may not be provided during service execution, reliable execution of the composition cannot be achieved. None of the existing works for static or dynamic contract generation addressed the specification of transactional guarantees. Hence, the transactional guarantees are included as SLAs and the transactional contracts are generated as soon as the service from a third party provider is selected. An approach is proposed to extend WS-Agreement to include transactional agreements and dynamically generating such Tx-Contracts.

The runtime selection algorithm has been experimented for the Vehicle Purchase and Registration System prototype and compared with its static selection. It incurs a total of 0.443 seconds only to select services, for all the 45 activities of a workflow from a registry containing 9000 services, with an average of 200 web services per activity. Since the runtime selection approach introduces the selection overhead during execution, the overall performance of the application is decreased by 40% in terms of execution time and by 50% in terms of throughput. However, even when the workflow size is increased by 800% and service registry size is increased by 700%, the performance of the proposed selection algorithm decreases only by 54% which has rather marginal impact on the overall performance of the application. In comparison with the existing static selection approach, the proposed dynamic selection approach is worthwhile for the advantages and flexibility that it offers.
The proposed approach is suitable for B2B applications where the user preferences and business policies frequently change. This approach is user-friendly and is adaptive to the frequent changes in user requirements as well as business policies. The proposed approach considers the present state of services for selection, since services are selected on-the-fly. The approach enables integration of different types of services, even though, different registries need to be searched and binding processes are different based on the type of service such as SOAP based and Restful. Since business applications follow a predefined ordering among the activities of the business process, the template based approach adapted in the proposed selection suits such applications. The proposed model requires the user to specify the input parameter values for any service, only when the service is composed and executed. Thus, the user can preserve the privacy of data and can avoid security risks.