Trustworthiness Monitoring of Dynamic Service Compositions

Hisain Elshaafi, Jimmy McGibney and Dmitri Botvich
TSSG, Waterford Institute of Technology
Waterford, Ireland

ABSTRACT
In Web service compositions the number of component services that need to be aggregated may be large and dynamically changing. Web service (WS) compositions require the capability to dynamically adapt to changes that may occur at runtime, including changes in the environment and in the component services themselves. Service composers must be able to respond swiftly to changed trustworthiness (TW) requirements and capabilities of service compositions, where those changes may not be easily predictable. With the availability of alternative services providing the same functionality as those already integrated in a service composition, service composers can take advantage of this by dynamically replacing degrading or unsatisfactory components. This paper provides techniques and an approach to dynamically detect and replace component services based on their trustworthiness.

Categories and Subject Descriptors
H.3.5 [Information Systems]: Online Information Services—web-based services; K.6.5 [Management of Computing and Information Systems]: Security and Protection

General Terms
management, reliability

Keywords
trustworthiness, monitoring, dynamic service composition

1. INTRODUCTION
Service Oriented Architecture and Web Services (WSs) are increasingly popular, with increased attention from industry. A key concept is that services can be dynamically or statically composed to create new services. Services are described, published, discovered, and assembled, providing loosely coupled distributed interoperable dynamic business processes. Service composition is the process of assembling multiple basic services into a single value-added composite service (CS). The resulting WS may be used directly by a service consumer or be recursively incorporated in further WS compositions. Dynamic WS composition is performed automatically at runtime as compared to static development-time composition.

For the CS to be trustworthy for the utilisation by its consumers, and for the services to be able to trust and rely on other possibly unknown services, WS composition techniques must be able to identify which component WSs are trustworthy. The composition techniques also must be able to maintain the most trustworthy and cost efficient CS. Maintaining trustworthiness helps consumer confidence and provides a safe environment for businesses to dynamically interact and carry out transactions. Therefore, addressing trust is essential for the success and adoption of the services paradigm.

We define trust as a relationship between two or more entities that indicates the contextual expectations from an entity towards another in relation to reliance in accomplishing an action at a certain quality. Trustworthiness (TW) of an entity is the level of trust that the trusting entity has in that entity. Reputation is the information available about an entity from various sources such as QoS and user ratings that can be used to determine its TW.

Aniketos is an EU research project [1] that addresses trustworthy and secure service compositions with run-time monitoring and adaptation of services. The adaptation is needed due to changing operational, business or threat environments or due to changes in service quality and behaviour. Among the challenges to addressing those problems is that the changes may not be easily predictable. Since down-time is costly, a CS must be able to operate even during an attack or increased demand, taking risks and adaptation costs into account. This paper focuses on those problems with the aim of maintaining TW of dynamic CSs.

The paper is structured as follows: Section 2 discusses the function and architecture of TW Monitoring module. Section 3 explains the mechanisms for the calculation of TW and cost of a CS based on its composition plan. The prediction of service TW based on its monitoring is examined in section 4. Section 5 describes experiments using simulations of TW monitoring and dynamic composition and adaptation. Related work is described in section 6 and conclusions are discussed in section 7.

2. TW MONITORING MODULE
In Aniketos, monitoring is the process of checking that WS
contracts are fulfilled over time, particularly if changes can occur. Monitoring is also used to detect vulnerabilities and discover attacks on a WS, e.g. by making use of intrusion detection systems or dynamic testing tools available in the environment. \(TW\) Monitoring component of the Anikutos platform is responsible for runtime monitoring of \(TW\) of CSs based on a set of mechanisms and metrics to ensure contract compliance.

Figure 1 illustrates the basic operation of WS composition and recomposition. A Service Composer is a service provider that is responsible for constructing WS compositions and offering them to consumers. A Service Composer is notified of important changes in \(TW\) of the CS as a result of one of its components. A component WS that is below the satisfactory \(TW\) level can be replaced with another WS offering the same capability but with better \(TW\). The Cost Engine determines the cost of the CS as a result of the change. The consideration of costs ensures that a balance is maintained between both \(TW\) and cost efficiency of the WS. The Process Manager is responsible for coordinating the execution of component WSs in a service composition. The main focus of our work is on \(TW\) monitoring and prediction which are independent but interoperable with other components and technologies commonly deployed in the services environments.

Figure 2 shows the architecture of \(TW\) Monitoring module. The trust events refer to the notifications received by the module from the Event Processing and QoS Monitoring components. Those events include QoS metrics and alerts that indicate violations or adherence to the WS contracts, threats or changes in the environment. In addition to the direct experience through those metrics and alerts, \(TW\) Monitoring can exchange recommendations with other online modules in relation to service \(TW\). Incoming events are evaluated by a rules engine to generate WS ratings. The rules calculate the rating for the event and add other attributes including the recency value (when event happened) and the type of event. Ratings are then stored and can be used for calculating the overall \(TW\) level of each WS. Context configurations allow the customisation of the trust context by adjusting the significance of types of trust events e.g. security and performance events. Policy configurations allow setting \(TW\) thresholds and algorithmic constants such as the rating half life. The trust engine is responsible for providing a prediction of \(TW\) level of a WS.

### 3. \(TW\) OF A COMPOSITION

We model \(TW\) level of a CS in general as a function \(g\) of \(TW\) of its components:

\[
T_{cs} = g\left(t_1, t_2, ..., t_m\right)
\]  

However, the calculation of \(TW\) level depends on the structure of the business process. Selection of component WSs statically (during design time) or dynamically uses information from which \(TW\) levels of the CS can be predicted. Selected WSs are executed in a business process. The process is viewed externally as a CS. The calculation of \(TW\) of the CS depends on how the abstract service is constructed. Component WSs may be invoked in a business process in one or more path constructs such as the following basic and commonly supported constructs:

- **Sequence**: WSs are invoked one after another.
- **Parallel with synchronisation (AND split/AND join)**: two or more WSs are invoked in parallel and their outcome is synchronised. All WSs must be executed successfully for the next WS to be executed.
- **Loop or iteration**: a WS is invoked in a loop until a condition is met. We assume that the number of iterations or its average is known at the time of composition.
- **Exclusive choice (XOR Split/XOR join)**: a WS is invoked instead of others if a condition is met. We assume that the likelihood of each alternative WS is to be invoked is known at the time of composition.
- **Discriminator (AND split/OR join)**: two or more WSs are executed in parallel but no synchronisation of the outcome of their execution.
- **Multi-choice with multi-merge (OR split/OR join)**: multiple WSs may be executed in parallel. Subsequent WSs can be executed once at least one of those WSs is completed.
- **Unordered sequence**: multiple WSs are executed sequentially but arbitrarily.

These and several other possible patterns varyingly supported by modelling languages and products are investigated in Workflow Patterns Initiative [2].

Table 1 shows our functions for calculating \(TW\) level \(T_m\) and cost \(C_{cm}\) per WS construct. For sequence, parallel and...
Table 1: TW aggregation per process construct

<table>
<thead>
<tr>
<th>Construct</th>
<th>TW ( (T_{cn}) )</th>
<th>Cost ( (C_{cn}) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sequence</td>
<td>( \prod_{i=1}^{n} t_i )</td>
<td>( \sum_{i=1}^{n} c_i )</td>
</tr>
<tr>
<td>Parallel</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Unordered sequence</td>
<td>( \prod_{i=1}^{n} (t_i)^s )</td>
<td>( n \cdot c_i )</td>
</tr>
<tr>
<td>Loop</td>
<td>( \sum_{i=1}^{n} p_i \cdot t_i )</td>
<td>( \sum_{i=1}^{n} c_i )</td>
</tr>
<tr>
<td>Exclusive choice</td>
<td>( 1 - \prod_{i=1}^{n} (1 - t_i) )</td>
<td>( \sum_{i=1}^{n} c_i )</td>
</tr>
<tr>
<td>Discrimator</td>
<td>( 1 - \prod_{i=1}^{n} (1 - t_i) )</td>
<td>( \sum_{i=1}^{n} c_i )</td>
</tr>
<tr>
<td>Multi-choice multi-merge</td>
<td>( 1 - \prod_{i=1}^{n} (1 - t_i) )</td>
<td>( \sum_{i=1}^{n} p_i \cdot c_i )</td>
</tr>
</tbody>
</table>

unordered sequence constructs, TW is calculated as a product of that of constituent WSs \( \{1, ..., n\} \): \( T_{cn} = \prod_{i=1}^{n} t_i \). The cost in those cases is simple sum of the cost of the components.

TW and cost of a loop construct containing \( n \) iterations of a WS \( s \) is the same as a sequence construct of \( n \) copies of \( s \) i.e. \( T_{cn} = (t_i)^n \) and \( C_{cn} = n \cdot c_i \).

Each WS in the alternative WSs in the exclusive choice has a probability \( p \) that it will be executed and \( \sum_{i=1}^{n} p_i = 1 \).

The aggregation of TW and cost in the exclusive choice is the sum of that of each component WS multiplied by its probability.

Since a discriminator construct only fails if all constituent WSs fail, its TW is as follows: \( T_{cn} = 1 - \prod_{i=1}^{n} (1 - t_i) \). However, the cost in this case is the sum of the cost of all WSs as they are all executed.

For a multichoice multi-merge construct, each non-exclusive WS is associated with a probability \( p \) that it will be executed. Note that in this case \( \sum_{i=1}^{n} p_i \geq 1 \) due to the non-exclusiveness. The calculation of TW and cost in multichoice multi-merge case is similar to that in the discriminator construct with consideration of the probabilities of executions.

4. TW PREDICTION OF A WS

Here we propose an algorithm that is more efficient than those proposed for multiagent systems in REGRET [5] and FIRE [6] since no need for recursively running through all the ratings with each new rating received. In this algorithm, TW level is determined using moving averages that are updated with every new rating. Older ratings reduce in value over time. The comparison with those algorithms is further discussed in the evaluation.

TW level \( T_s \) is determined by two values \( T_s = (R_s, F_s) \) where \( R_s \) is the trust score for WS \( s \) and \( 0 \leq R_s \leq 1 \). \( F_s \) is the confidence in the score and \( 0 \leq F_s \leq 1 \). We calculate the trust score for WS \( s \), \( R_s \) as a dynamically weighted moving average of the rating scores. When a new rating is received the trust score is updated. First we update the total weight of all received ratings. The weighting is based on the recency of the previous ratings \( w_t \). Recency weight \( w_t \) decays exponentially and \( 0 \leq w_t \leq 1 \), as follows:

\[
 w_t = \left( \frac{1}{2} \right)^{\frac{t}{l}} \tag{2} \]

where \( \Delta t \) is the age of the rating i.e. the difference between the current time and the time when the rating took place; and \( l \) is the customizable half life of the rating.

The accumulated weight of the trust score \( w_s \) \( (w_s > 0) \) is updated as follows:

\[
 w_s = w_s \cdot w_{i,j} + w_j \tag{3} \]

where \( w_j \) is the weight of the new rating \( r_{i,j} \); \( w_j = w_{i,j} \) but for a rating that is generated at the time of calculation i.e. \( \Delta t = 0 \) and \( w_{i,j} = 1 \), the new accumulated weight \( (w_s = w_s \cdot w_{i,j}) \).

To facilitate the recalculation of TW level when new ratings are received, value of \( w_s \) is stored after each update.

\[
 R_s = \frac{(w_s - w_{i,j}) \cdot R_s + w_{i,j} \cdot r_{i,j}}{w_s} \tag{4} \]

We calculate the confidence value of service \( s \), \( F_s \) as:

\[
 F_s = f_n \cdot f_s \tag{5} \]

where \( f_n \) is the rating quantity confidence; and \( f_s \) is the rating quality confidence. \( f_n \) is calculated as follows:

\[
 f_n = 1 - e^{-\alpha \cdot w_s} \tag{6} \]

where \( \alpha \) is a constant parameter that can be used to adjust the slope of the relationship between the sum of the ratings’ weights and the quantity confidence. The higher the value of \( \alpha \) the faster the full confidence (i.e. 1) is reached. It can be set to any positive value but for gradual increase in confidence it should typically be set to a value between 0 and 1. The confidence increases proportionate to the number of ratings and to the degree of their recency.

The quality confidence \( f_s \) is calculated as follows:

\[
 d_s = \frac{(w_s - w_{i,j}) \cdot d_s + w_{i,j} \cdot |R_s - r_{i,j}|}{w_s} \]

\[
 f_s = 1 - d_s \tag{7} \]

where \( d_s \) is the deviation history of the WS trust ratings around the trust score. To facilitate the recalculation of TW level when new ratings are received, value of \( d_s \) is stored after each update. \( |R_s - r_{i,j}| \) is the absolute difference between the overall trust score and individual rating score. \( f_s \) indicates the deviation of the rating scores around the overall trust score and ranges between 0 (highest deviations) and 1 (lowest deviations).

For optimal selection of a component WS for service compositions, we use the following formula:

\[
 \frac{T_{cs}}{P_{cs}} \tag{8} \]

where \( P_{cs} \) is the cost of the CS and \( T_{cs} \) is a representation of the trustworthiness calculated from trust \( (R_{cs}) \) and confidence scores \( (F_{cs}) \) as in (9).

\[
 T_{cs} = R_{cs} \cdot F_{cs} \tag{9} \]
rating scores. Those fluctuations may indicate that a WS is not scalable enough to meet demands during peak times. Hence a service composer might be in a situation where it has to choose between for example two component WSs at runtime; s1 with medium trust score and low confidence score (caused by deviations in the ratings) and s2 with medium trust score and medium confidence score. Despite the same trust score, s2 will have higher TW because of its better scalability and performance over peak times.

The simulation creates a CS consisting of five abstract services and selects a component WS for each abstract service from ten available WSs based on TW and cost. WSs continue to receive new ratings over the duration of their runtime. During each simulation each of the WSs receives 1000 ratings. The ratings trigger checking of TW of CS. A significant reduction in TW (set at 5% in these experiments) results in adaptation of CS by first searching for component WSs that have optimal TW and cost balance. Certain periods during the simulation are set to represent high operational demand which results in reduced performance of some of the component WSs and consequently lower ratings. The ratings values vary depending on the WS instance, the time of the rating and whether there is an increased demand. The high demand is set to cause consistent low performance (resulting in mainly low trust score) or fluctuations in performance (resulting mainly in low confidence) in some of the WSs. A Gaussian random number generator is used to generate new ratings where the mean and the variance varies for each WS and the time of rating (e.g. high demand). In our simulations the peak demand time is set between 600 and 800.

Figure 3: 5% Drop in trust score

![Figure 3: 5% Drop in trust score](image)

In figure 3 component WS s1 is selected for a composition together with other WSs based on their TW and competitive costs at the time of selection as described in section 4. During the runtime of the composition, the trust score (and confidence score to a lesser extent) of WS s1 starts to decrease significantly at a point in time P1 due to increased demand and low scalability. Consequently, TW of CS starts to decline. As the decline reaches 5% it results at P2 in the replacement of s1 with s2 that still maintains a good TW level. In Figure 4 service s1 maintains a good trust score but due to continuous fluctuations in the ratings starting at P1, its confidence and consequently that of CS decline (below 5%). It results in the search for an alternative component WS and the adaptation of CS (at P2) and s1 is replaced with s2. The promptness of the replacement is dependent on the availability of an alternative WS with better TW-cost balance.

Figure 4: 5% Drop in confidence

![Figure 4: 5% Drop in confidence](image)

Figure 5: Processing time for FIRE & our algorithm

FIRE [6] is a widely cited trust management model and algorithm for the assessment of TW of agents in open multi-agent systems. FIRE extends REGRET system developed by Sabater [5]. Unlike our approach of using a moving average, FIRE algorithm recursively runs through all the ratings with each new rating received. This results in an increasing delay in responding to requests for TW evaluation as ratings increase in quantity. Figure 5 compares the processing times of new ratings required by the algorithm described in this paper and that of FIRE. The figure clearly shows our algorithm is considerably more efficient as the number of ratings available for assessment increases.

Figure 6: Trust changes in discriminator construct

The above experiments evaluate TW of a CS based on the assumption that its component WSs are in constructs
that calculate their TW as a product of TW of the constituent WSs e.g. sequence, parallel. However, TW of CS may be affected differently by changes in TW of its components if they are in other types of constructs. For example, consider the effect of the decline in the trust score of one of components (s1) constituting a discriminator construct \{s1, s2, s3\} as in figure 6. This decline does not affect CS as the calculation method suggests as long as other WSs in the construct maintain their TW. TW of CS here is less than that of component WSs because of other sequence component WSs (i.e. s4, s5) that are part of CS. Worthy of notice in this construct is that the cost of a component plays more significant role than its TW since it is the sum of that of the component WSs. Likewise, in the case of exclusive choice construct TW of CS is only partially affected by decline of TW of one of the WSs depending on its probability of execution. On the other hand, a moderate decline in TW of a component WS executed in a loop results in a significant decline. In this case TW values are more significant than cost of a component WS because of the exponential effect of changes in its TW.

6. RELATED WORK

In the past decade there has been large amount of activity in the area of computational trust and reputation, with applications in security, multi-agent systems, game theory, and spam filtering [3-7]. The terms trust and trust models are used in Web service standards e.g. WS-Trust [8] – but they are limited to the context of being able to trust the service identity [9]. However, establishing a service identity may not mean that it is trustworthy as an authenticated service could for instance be temporarily unreliable or unavailable.

There is also work on trust and reputation specific to the WSs domain. In [10] the authors present an agent based trust model that enables rating of individual WSs as well as providers. A framework for reputation based WS selection is proposed in [11]. Service consumers submit their ratings to a “Reputation Manager Service” which computes the service’s reputation based on those ratings. Singh [12] discusses the challenges of trustworthy service composition. He states that current approaches fail to adequately address the challenges for trust in service oriented computing. In [13] the authors introduce RATEWeb; a framework for establishing trust in service oriented environments. RATEWeb operates by aggregating reputation ratings from consumers in a P2P fashion. It aims to support the use of trust in WS selection and composition. However, unlike the work described in this paper, it does not consider the computation of TW in a CS. It also does not provide mechanisms for responding to dynamic changes in the environment that for instance may affect a service’s TW. Authors in [14] discuss a dynamic charging approach for service compositions including duration of usage and discounts but do not specifically address the role of cost and trust in service selection.

7. CONCLUSION

This paper has presented an approach to maintaining TW of dynamic composite services. TW Monitoring module continuously monitors the adherence of the WSs to their contracts and receives metrics and alerts relating to the QoS and security events. WS ratings are generated using a rules engine and stored in the module's ratings store. The calculation of TW and cost depends on the construction of the composite service.

Ongoing and future work includes the development of novel optimisation mechanisms for dynamic WS composition and adaptation to provide robust algorithms that take into account multiple objectives including TW, cost and security.

Acknowledgement: The research leading to these results has received funding from the European Union’s Seventh Framework Programme (FP7/2007-2013) under grant agreement no 257930.

8. REFERENCES

[1] Aniketos (Secure and Trustworthy Composite Services), http://www.aniketos.eu

29