Automated Trust Model for P2P Systems

Nardine Z. Osman
May 2008

1 Introduction

One of the issues peers face in P2P systems is selecting the right peer to collaborate with. Trust is one of the main criteria for this selection. In this document, we present a model for peers to compute this trust measure in an automated way without requiring any changes to the way the peer or the P2P system is engineered.

The OpenKnowledge (OK) project (Siebes et al., 2007) provides the required platform for achieving this. The general goal of the OK project is to achieve practical open and distributed systems in which peers may participate at low and acceptable costs. To achieve this, the project proposes to shift the focus from the different system entities (such as peers, agents, web services, etc.) to the interactions that connect these entities together. An interaction, defining precisely how each peer role may execute its part of the interaction, is specified through an LCC interaction model (Robertson, 2004). Issues, such as commitments or semantic agreements, are then dealt with at the interaction level. Grounding such issues to a specific interaction model facilitates the process of addressing them in a more realistic and dynamic approach. The OK system provides the methods needed for finding, sharing, and executing interactions. However, a crucial question remains: “How do peers select the other peers they need to collaborate with?”

Trust is one of the issues peers have to take into consideration when selecting other peers. The OK project deliverable 4.5 (Giunchiglia et al., 2007) proposes a trust algorithm for computing how much a given peer is trusted in performing a given set of actions when playing a specific role in a given interaction. OK deliverable 4.8 (Pane et al., 2007) presents a trust module that is made available by the OK system for helping peers compute their trust in others. This paper builds upon these proposed trust algorithms and illustrates how the entire trust maintenance and computation process may be fully automated.

Section 2 provides the background needed for understanding the trust algorithm this paper relies on along with a general overview of our proposed automated trust mechanism. Section 3 proposes some interaction models that would help us achieve this automation. Finally, the conclusion is drawn in Section 4.

2 Automated Trust

One approach of computing trust measures, as illustrated by OK deliverable 4.5 (Giunchiglia et al., 2007), is by relying on past experience. The basic idea is that deciding to engage in a given interaction requires the peer to commit to a predefined set of actions. Throughout the interaction, other peers may then be able to observe whether the original peer has fulfilled its commitments or not. For instance, did the seller deliver the items as committed to? A database of past experiences may then be constructed using these observations. If a seller peer did not deliver the item bought then the buyer peer will be able to observe and remember that the seller has broken its commitment. In a future scenario it will know that this seller peer is not to be trusted. OK deliverable 4.8 presents a trust module for calculating such trust measures.

Instead of manually using such a trust module, this document presents a method for automating the
entire trust process. This is achieved by defining sample generic interaction models, which we call trust interactions, that peers can use when communicating with a trust module. Two different classes of these trust interactions would exist:

1. Interaction models that allow peers to obtain relevant past experiences and calculate trust measures accordingly.
2. Interaction models that allow peers to save new experiences each time an observation is made concerning a previous commitment.

Specifying the trust mechanism through interaction models results in several interesting aspects:

1. This method has no additional engineering requirement on the P2P system or on the peers playing in such a system. For example, instead of having one dedicated trust module with a centralised database of past experiences, the system might contain a collection of trust web services or peers with trust calculation capabilities. Any interested peer would then communicate with such services (or peers) just like communicating with any other service or peer in the system: via an interaction model.
2. A trust interaction model may be generic enough to be used in many different use cases. For example, one trust interaction might be used by a peer interested in using only its personal history of past experiences, as well as a peer relying only on the past experiences of its close and trusted circle of 'friends', or a peer resorting to a public trust service with a huge history database.
3. A fully automated trust mechanism may easily be achieved with such generic trust interaction models. For example, some interaction models might require a peer to decide who it will be collaborating with at runtime. In such a case, another trust interaction model may be automatically instantiated at runtime to help the peer retrieve relevant past experiences and compute its trust measure based on that. Another example is that the peer has to automatically update the database of past experiences every time an observation is made in interaction I concerning a previous commitment, or every time the user gives a feedback concerning its satisfaction with interaction I. To achieve this, interaction I may be automatically modified in such a way that every time a message is received (i.e. an observation can be made), an experience is added to the database of past experiences stating the result of this new observation. Also, every time a given role completes its task in an interaction, the user is automatically asked for its feedback, which is also saved in the database of past experiences.

The following section provides a sample of these proposed trust interaction models.

3 Trust Interaction Models

3.1 Calculating Trust

According to the trust algorithm presented in the previous section, the main trust question that peers ask is defined as: “Is peer $P$ trusted in fulfilling its commitment $Cm$?” To answer this question, interaction model $IMc$ is defined for calculating this trust measure. This measure is usually calculated for a given role in a given interaction. Although this information may easily be appended, we currently drop this information for the sake of keeping our example simple and clear.

Figure 1 specifies interaction $IMc$ in the lightweight coordination calculus (LCC), a typical process calculus (Osman, 2008, Section 2.2.5). In short, LCC peers are defined by their roles and identifiers ($a(Role,Id)$). When defining interaction models, it is common to specify the role while keeping the
peer identifier as a variable. This allows the interaction model to be instantiated by different peers, each running its own instance of the interaction. The peer’s behaviour may be defined in terms of three different classes of atomic actions. First, a peer can send a message to another peer (Message=>Peer) or receive a message from one (Message<=Peer). Second, a peer can take on a different role (a(Role,Id)). Third, a peer can do nothing (null), which is usually used when the peer needs to perform internal computations. Complex peer behaviour may then be built on top of these three classes of atomic actions using the sequential (then), choice (or), and conditional (<=-) operators. The conditional operator is used for linking constraints to atomic actions.

As illustrated by Figure 1, interaction IMc is initiated by peer TI, playing the role trust_inquirer, who is interested in calculating how much is peer P trusted in fulfilling its commitment Cm. To achieve this, peer TI first obtains the list of gossipers GList that might have information about peer P’s commitment. Peer TI then takes the role request_texperiences for requesting the related past experience of each of the gossipers. This is achieved by sending the message “request_texperience(Peer,Cm)” to each gossiper (Peer) of the list GList. When the message has been sent to all gossipers, peer TI takes the role collect_texperiences for collecting the related past experience of each gossiper. In this role, after each gossiper’s list of past experiences (ExpList) is received, peer TI calculates the reliability of each of the gossiper’s experiences, resulting in a modified list of experiences (NewExpList). This new list is then appended to TT’s old list of past experiences.

```prolog
a(trust_inquirer(Peer,Cm,Result),TI) ::
a(request_texperiences(GList,Peer,Cm),TI) <=-- gossipers(Peer,Cm,GList) then
a(collect_texperiences(GList,Peer,Cm,[],Exps),TI) then
aggregate_texperiences(Exps,TPars) => a(trust_calculator,TC) <=-- trust_parameters(Peer,Cm,TPars) then
trust_measure(Result) <= a(trust_calculator,TC).

a(request_texperiences(GList,Peer,Cm),TI) ::
null <=-- GList=[]
or
( request_texperience(Peer,Cm) => a(gossiper,G) <=-- GList=[G|NewGList] then
a(request_texperiences(NewGList,Peer,Cm),TI) ).

a(collect_texperiences(GList,Peer,Cm,OldExps,Exps),TI) ::
null <=-- GList=[] and Exps=OldExps
or
( select(G,GList,NewGList) <=-- trust_experience(ExpList) <= a(gossiper,G) then
null <=-- compute_reputation(G,Peer,Cm,ExpList,NewExpList) and append(NewExpList,OldExps,NewExps)
a(collect_texperiences(NewGList,Peer,Cm,NewExps,Exps),TI) ).

a(gossiper,G) ::
request_texperience(Peer,Cm) <= a(request_texperiences(_,_),TI) then
null <=-- experiences_db(Mg) and get_experiences(Mg,Peer,Cm,ExperienceList) then
texperience(ExperienceList) => a(collect_texperiences(_,_,_),TI).

a(trust_calculator,TC) ::
aggregate_texperiences(Exps,TPar) <= a(trust_inquirer,TI) then
trust_measure(Result) => a(trust_inquirer,TI) <=-- aggregate_trust(Exps,TPar,Result).
```

Figure 1. The LCC specification of interaction IMc
The role *collect_experience* terminates when all gossipers have sent their lists of experiences. The trust computation usually relies on a number of parameters, such as a default decay probability distribution, an ideal enactment probability distribution, a decay factor, parameters specifying the importance of interaction models similarity, role similarity, etc (Giunchiglia et al., 2007). These parameters are not usually fixed. Each peer has to choose how trust will be computed. Therefore, peer *TI* will then have to decide what are its trust parameters. When this information is obtained, peer *TI* then asks the *trust_calculator* role to calculate the trust measure accordingly. Finally, the trust measure is received from the *trust_calculator*.

Note that the *gossiper*'s role definition is straightforward. After receiving a request concerning its past experiences related to peer *P*'s commitment *Cm*, it fetches the list of related experiences from its database *Mg* and sends this list back to the requesting peer.

Similarly, the *trust_calculator*'s role definition is also straightforward. After receiving the list of experiences *Exps* and the trust parameters *TPam*, it calculates the trust measure by making use of the defined list of past experiences (*Exps*) and the inquirer's chosen trust parameters (*TPam*). The result is finally sent back to the inquirer.

For a clear visual illustration, Figure 2 defines interaction *IMc* through the state graphs of its roles.

3.2 Recording Experiences

Trust calculations, in this document, are based on the collection of past experiences. Therefore, with each new interaction, peers should save the results of this interaction as a new record in the database of past experiences. Note that the previous issue of retrieving related previous experiences and calculating the resulting trust measure may all be defined in one complete interaction model, such as interaction *IMc*. This interaction model may then be instantiated from within other interactions whenever needed to help peers select, at runtime, the other peers they wish to collaborate with.

However, the issue of observing the results of previous commitments and recording these results in the database of past experiences does not have a neat solution as the former issue. There can be no one complete interaction model that would help achieve this. This is because observations are made from within original interactions. Every time a message is received, an observation may be made. Furthermore, observations made in one interaction model need to be saved in one experience record, as required by the trust algorithm presented in OK deliverable 4.5 (Giunchiglia et al., 2007). Therefore, the interaction model being observed should be modified in such a way that:

1. Every time a message is received an observation is saved. For example, if the observing peer was playing the role *Role* in interaction *IM*, then the role definition *a(Role,Peer1)* would be modified in such a way that every time a message *ObservedMsg* is received (i.e. observed) an additional line of LCC code, marked with asterisks (*) in the following illustration, is appended to the role definition *a(Role,Peer1)*:

*a(Role,Peer1) ::

\[\ldots\]

\[\text{ObservedMsg} \leq a(Role,Peer2)\]

* \text{observed([Peer1,Peer2,IM,Role,ObservedMsg,LCCMsg,CommitMsg,Constraints,T,BinaryResult])}*

* \text{=> a(experience_recorder(_,_),ER)}*

* \text{<-- committed(IM,ObservedMsg,LCCMsg,CommitMsg,Constraints,T,BinaryResult)}*

\[\ldots\]
We name this additional (and generic) line of code sub-role \textit{observer1}. Sub-role \textit{observer1} performs two main actions. First, the observing peer \textit{Peer1} recalls the commitment made earlier by peer \textit{Peer2}, in which \textit{Peer2} would have committed in interaction \textit{IM} at time \textit{T} to sending the message (\textit{CommitMsg}) instead of the originally defined message (\textit{LCCMsg}). The commitment states that \textit{Peer2} is also capable of fulfilling the constraints \textit{Constraints} attached to this message. If the observed message (\textit{ObservedMsg}) matches the message committed to (\textit{CommitMsg}), then the \textit{BinaryResult} is set to \textit{true}. Otherwise, it is set to \textit{false}. After completing this step, the observing peer \textit{Peer1} sends its observation result to the \textit{experience_recorder} peer \textit{ER}, which is responsible for saving this observation in the database of past experiences.

2. Every time a role completes its task successfully in an interaction, the user may be asked to provide a feedback. This feedback is appended to all the observations previously made in this interaction. For example, if the observing peer was playing the role \textit{Role} in interaction \textit{IM}, then the role definition \textit{a(Role,Peer1)} would be modified in such a way that at the very end an additional line of LCC code, marked with asterisks (*) in the following illustration, is appended to it:

\begin{figure}
\centering
\includegraphics[width=\textwidth]{state_diagram.png}
\caption{The state graphs of interaction IMc’s roles}
\end{figure}
We name this additional (and generic) line of code sub-role _observer2_. Sub-role _observer2_ performs two main actions. First, it gets the user feedback by invoking a GUI. This is achieved by the function _get_user_feedback_ that returns a result _R_. Then, the result is sent to the _experience_recorder_ so that it is appended to all previous observations of this interaction.

Note that sub-roles _observer1_ and _observer2_ communicate their results with the role _experience_recorder_, which is responsible for saving these results in the database of past experiences. The role _experience_recorder_ is then defined as follows:

\[
\text{a(experience_recorder(RecordedExperiences),ER) ::=}
\]

\[
\text{a(experience_recorder([],RecordedExperiences),ER).}
\]

\[
\text{a(experience_recorder(InitialExperiences,RecordedExperiences),ER) ::=}
\]

\[
\text{save_experiences(InitialExperiences,Result,Id,RecordedExperiences) } \quad \text{get_user_feedback(Result) } \quad \text{a(_,Id)}
\]

\[
\text{or}
\]

\[
\text{( append_commitment(Id,O,InitialExperiences,NewExperiences) } \quad \text{observed(O) } \quad \text{a(_,Id) then}
\]

\[
\text{a(experience_recorder(NewExperiences,RecordedExperiences),ER) ).}
\]

The _experience_recorder_ role definition states that every time and observation _O_ is received, it is appended to the list of experiences of this current interaction. Finally, when the user feedback is received, it is appended to all previously saved observations of this interaction, and all these observations are then saved as one experience record in the database of past experiences.

For a clear visual illustration, Figure 3 defines the _observer1_ and _observer2_ sub-roles along with the _experience_recorder_ role through their state graph.

4 Conclusion

This document has proposed an interaction model approach for calculating trust based on past experiences. Two generic interaction models have been presented: one for retrieving past experiences and calculating trust accordingly, and the other for observing commitments and recording experiences. This new approach for dealing with trust offers several contributions.

First, the issue of trust is dealt with like any other issue in the system. If the peer is not capable of performing, or decides not to perform, the trust calculations entirely on its own, then it may consult other peers in the system. For example, it may consult its friends to provide additional past experiences when needed. It may also decide to use a trust web service for obtaining the trust measure when it believes it does not have the computational capabilities of doing so itself. This approach drops the need for a centralised trust module. Furthermore, it poses no additional engineering requirements on both the P2P system and the peers participating in such a system.

Second, there is no one right answer for calculating trust. Trust calculations are usually personalised to the taste of the peer interested in obtaining the trust measure. For example, it is the peer that decides whether it will use its own database of past experiences, consult the past experiences of its trusted circle of friends, consult a public trust service, etc. The peer also decides what trust parameters should be used in this current computation. Will it consider a small or large decay factor? Will it give a high preference to interaction model similarities or focus solely on the action in question? Another decision the peer has to make is whether it has the computational capabilities of performing the
computation itself, or will it use another peer for obtaining this service? Furthermore, all these decisions are context based. In different scenarios the peer will make different decisions when answering such questions. Nevertheless, the interaction models presented in this document are generic enough to be used in all such possible use cases. For instance, in interaction $IMc$, if the peer $TI$ playing the role trust_inquirer also decides to play the role trust_calculator, then this implies that the peer will be performing the trust computations itself. Similarly, if the peer $Peer1$ playing the sub-role observer1 also decides to play the role experience_recorder, then this implies that the peer will actually be updating its personal database of past experiences.

Finally, for future work, we suggest the addition of an automated process for modifying the existing interactions that will be observed. This paper shows how such an automation may be achieved by laying down the rules of such a process. For example, interaction $IMc$ may be called any time a peer has to select another peer to collaborate with at runtime. Sub-roles observer1 and observer2 may be added to existing role definitions following rules well defined in Section 3.2. And so on. These rules lay down the model for achieving an automated experience (or reputation) based trust mechanism.

Figure 3. The state graph of the experience_recorder role and the observer1 and observer2 sub-roles
References


