Validation and Verification of Agent Models for Trust: Independent compared to Relative Trust

Mark Hoogendoorn*, Syed Waqar Jaffry* and Peter-Paul van Maanen*†

* Department of Artificial Intelligence, Vrije Universiteit Amsterdam
De Boelelaan 1081a, 1081 HV Amsterdam, The Netherlands
Email: {mhoogen, swjaffry}@cs.vu.nl
† Department of Cognitive Systems Engineering, TNO Human Factors
P.O. Box 23, 3769 ZG Soesterberg, The Netherlands
Email: peter-paul.vanmaanen@tno.nl

***Abstract*—In this paper, the results of a validation experiment for two existing computational trust models describing human trust are reported. One model uses experiences of performance in order to estimate the trust in different trustees. The second model carries the notion of relative trust. The idea of relative trust is that trust in a certain trustee not solely depends on the experiences with that trustee, but also on trustees that are considered competitors of that trustee. In order to validate the models, parameter adaptation has been used to tailor the models towards human behavior. A comparison between the two models has also been made to see whether the notion of relative trust describes human trust behavior in a more accurate way. The results show that taking trust relativity into account indeed leads to a higher accuracy of the trust model. Finally, a number of assumptions underlying the two models are verified using an automated verification tool.**

***Index Terms*—Trust, Multi-Agent Systems, Parameter Adaptation, Validation, Verification.**

1 Introduction

When considering relations and interaction between agents, the concept of trust is of utmost importance. Within the domain of multi-agent systems, the concept of trust has been a topic of research for many years (e.g., Sabater and Sierra, 2005; Ramchurn et al., 2004). Within this research, the development of models expressing how agents form trust based upon direct experiences with a trustee or information obtained from parties other than the trustee is one of the central themes. Some of these models aim at creating trust models that can be utilized effectively within a software agent environment (e.g., Maanen et al., 2007), whereas other models aim to present an accurate model of human trust (see e.g., Jonker and Treur, 1998; Falcone and Castelfranchi, 2004; Hoogendoorn et al., 2008). The latter type of model can be very useful when developing a personal assistant agent for a human with the awareness of the human's trust in different other agents (human or computer) and him- or herself (trustees). This could for example avoid advising to use particular information sources that are not trusted by the human or could be used to enhance the trust relationship with the personal assistant agent itself.

In order for computational trust models to be usable in real life settings, the validity of these models should be proven first. However, relatively few experiments have been performed that validate the accuracy of computational trust models upon empirical data. For instance, in (Jonker et al., 2004) an experiment has been conducted whereby the trends in human trust behavior have been analyzed to verify properties underlying trust models developed in the domain of multi-agent systems. However, no attempt was made to fit the model to the trusting behavior of the human.

In this paper, the results of a validation experiment for two computational trust models describing human trust are reported. An in previously studies utilized trust model (Maanen et al., 2007), which was inspired on the trust model described in (Jonker and Treur, 1998), has been taken as a baseline model. This model uses experiences of performance

in order to estimate the trust in different trustees. The second model which is validated in this study is a model which also carries the notion of relative trust (Hoogendoorn et al., 2008). The idea of relative trust is that trust in a certain trustee not solely depends on the experiences with that trustee, but also with trustees that are considered competitors of that trustee. A comparison between the two models has is also made to see whether the notion of relative trust describes human trust behavior in a more accurate way.

The validation process includes a number of steps. First, an experiment with participants has been performed in which trust plays an important role. As a result, empirical data has been obtained, that is usable for validating the two models. One part of the dataset is used to learn the best parameters for the two different trust models. Then these parameters are used to estimate human trust, using the same input as was used to generate the other part of the dataset. Finally, a number of assumptions underlying the two trust models are verified upon the obtained dataset using an automated verification tool.

This paper is organized as follows. First, the two trust models that have been used in this study are explained in Section 2. The experimental method is explained in Section 3. Thereafter, the results of the experiment in terms of model validation and verification are described in Section 4. Finally, Section 5 is a discussion.

2 Agent Models for Trust

In this section the two types of trust models which are subject of validation are described. In Section 2.1 a model that describes human trust in one agent that is independent of trust in other agents, whereas the model described in Section 2.2 is dependent of trust in other agents.

2.1 Independent Trust Model

As has been mentioned above, this model is independent of trust in other agents (Maanen et al., 2007; Jonker and Treur, 1998). Trust is calculated by means of the following formula:

$$\tau^a_{\text{indep}}(t) = \tau^a_{\text{indep}}(t-1) \cdot \lambda^a_{\text{indep}} + (1-p^a(t)) \cdot (1-\lambda^a_{\text{indep}})$$

where λ^a_{indep} is the independent model decay factor for trustee agent $a \in A$, with $0 \leq \lambda^a_{\text{indep}} \leq 1$, and $p^a(t)$ is the penalty of trustee agent a at time point $t \in \mathbb{N}$, with $0 \leq p^a(t) \leq 1$, and $\tau^a_{\text{indep}}(t)$ is the independent trust in agent a at time point t. The penalty is calculated by means of the current feedback on the performance of the trustee agent. If $p^a(t) = 0$, then the performance is good, whereas a performance of $p^a(t) = 1$ is very bad. The independent trust is calculated for each trustee agent. Eventual reliance decisions are made by determining the maximum of the independent trust over all trustee agents $a \in A$.

2.2 Relative Trust Model

This section describes the relative trust model (Hoogendoorn et al., 2009b). In this model trustees are considered competitors, and the human trust in a trustee depends on the relative experiences with the trustee to the experiences from the other trustees. The model defines the total trust of the human as the difference between positive trust and negative trust (distrust) on the trustee. It includes several human personality characteristics including trust decay, flexibility, and degree of autonomy (context-independence) of the trust.

In the relative trust model it is assumed that a set of trustees $\{S_1, S_2, \ldots, S_n\}$ is available that can be selected to give particular advice at each time step. Upon selection of one of the trustees (S_i), an experience is passed back indicating how well the trustee performed. This experience ($E_i(t)$) is a number on the interval $[-1, 1]$. Hereby, -1 expresses a negative experience, 0 is a neutral experience, and 1 a positive experience. In the trust model, the assumption is made that the trustee currently with the highest trust value is always selected for an experience.

As mentioned, the relative trust model includes several parameters representing human characteristics including trust flexibility β (measuring the change in trust on each new experience), decay γ (decay in trust when there is no experience) and autonomy η (dependence of the trust calculation considering other options). The model parameters β, γ and η have values from the interval $[0, 1]$.

As mentioned before, the model is composed from two models: one for positive trust, accumulat-

ing positive experiences, and one for negative trust, accumulating negative experiences. Both negative and positive trust are represented by a number between $[0, 1]$. The human's total trust $T_i(t)$ in S_i is the difference in positive and negative trust of S_i at time point t, which is a number between $[-1, 1]$, where -1 and 1 represent the minimum and maximum values of trust, respectively. In particular, also the human's initial total trust of S_i at time point 0 is $T_i(0)$, which is the difference in initial trust $T_i^+(0)$ and distrust $T_i^-(0)$ in S_i at time point 0.

As a differential equation the change in positive and negative trust over time is described in the following manner (Hoogendoorn et al., 2009b):

$$\frac{dT_i^+(t)}{dt} = E_i(t) \cdot \frac{(E_i(t)+1)}{2} \cdot \beta \cdot \Big(\eta \cdot (1 - T_i^+(t)) + (1-\eta) \cdot (\tau_i^+(t) - 1) \cdot T_i^+(t) \cdot (1 - T_i^+(t))\Big) - \gamma \cdot T_i^+(t) \cdot (1 + E_i(t)) \cdot (1 - E_i(t))$$

$$\frac{dT_i^-(t)}{dt} = E_i(t) \cdot \frac{(E_i(t)-1)}{2} \cdot \beta \cdot \Big(\eta \cdot (1 - T_i^-(t)) + (1-\eta) \cdot (\tau_i^-(t) - 1) \cdot T_i^-(t) \cdot (1 - T_i^-(t))\Big) - \gamma \cdot T_i^-(t) \cdot (1 + E_i(t)) \cdot (1 - E_i(t))$$

In the above equations, $E_i(t)$ is the experience value given by S_i at time point t. Also $\tau_i^+(t)$ and $\tau_i^-(t)$ are the human's relative positive and negative trust in S_i at time point t, which is the ratio of the human's positive or negative trust in S_i to the average human's positive or negative trust in all trustees at time point t defined as follows:

$$\tau_i^+(t) = \frac{T_i^+(t)}{\Big(\frac{\sum_{j=1}^{n} T_j^+(t)}{n}\Big)}$$

and

$$\tau_i^-(t) = \frac{T_i^-(t)}{\Big(\frac{\sum_{j=1}^{n} T_j^-(t)}{n}\Big)}$$

3 METHOD

In this section the experimental methodology is explained. In Section 3.1 the participants are described. In Section 3.2 an overview of the used experimental environment is given. Thereafter, the procedure of the experiment is explained in four stages: In Sections 3.3, 3.4, 3.5 and 3.6, the procedures of data collection, parameter adaptation, model validation and model verification are explained, respectively. The results of the experiment are given in Section 4.

3.1 Participants

18 Participants (eight male and ten female) with an average age of 23 ($SD = 3.8$) participated in the experiment as paid volunteers. Participants were selected between the age of 20 and 30 and were not color blinded. All were experienced computer users, with an average of 16.2 hours of computer usage each week ($SD = 9.32$).

3.2 Task

The experimental task was a classification task in which two participants on two separate personal computers had to classify geographical areas according to specific criteria as areas that either needed to be attacked, helped or left alone by ground troops. The participants needed to base their classification on real-time computer generated video images that resembled video footage of real unmanned aerial vehicles (UAVs). On the camera images, multiple objects were shown. There were four kinds of objects: civilians, rebels, tanks, and cars. The identification of the number of each of these object types was needed to perform the classification. Each object type had a score (either -2, -1, 0, 1 or 2, respectively) and the total score within an area had be determined. Based on this total score the participants could classify a geographical area (i.e., attack when above 2, help when below -2 or do nothing when in between). Participants had to classify two areas at the same time and in total 98 areas had to be classified. Both participants did the same areas with the same UAV video footage.

During the time a UAV flew over an area, three phases occurred: The first phase was the advice

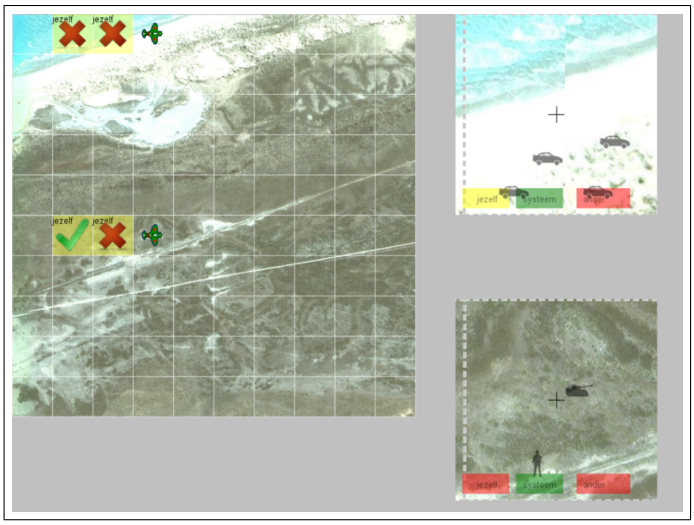

Figure 1. Interface of the task.

phase. In this phase both participants and a supporting software agent gave an advice about the proper classification (attack, help, or do nothing). This means that there were three advices at the end of this phase. It was also possible for the participants to refrain from giving an advice, but this almost did not happen. The second phase was the reliance phase. In this phase the advices of both the participants and that of the supporting software agent were communicated to each participant. Based on these advices the participants had to indicate which advice, and therefore which of the three trustees (self, other or software agent), they trusted the most. Participants were instructed to maximize the number of correct classifications at both phases (i.e., advice and reliance phase). The third phase was the feedback phase, in which the correct answer was given to both participants. Based on this feedback the participants could update their internal trust models for each trustee (self, other, software agent).

In Figure 1 the interface of the task is shown. The map is divided in 10×10 areas. These boxes are the areas that were classified. The first UAV starts in the top left corner and the second one left in the middle. The UAVs fly a predefined route so participants do not have to pay attention to navigation. The camera footage of the upper UAV is positioned top right and the other one bottom right.

Each experiment contained an easy as well as a difficult part. Difficulty was manipulated by the amount of objects visible on the screen. The advice of the self, other and the software agent was communicated via dedicated boxes below the camera images. The advice to attack, help, or do nothing was communicated by red, green, and yellow, respectively. On the overview screen on the left, feedback was communicated by the appearance of a green tick or a red cross. The reliance decision of the participant is also shown on the overview screen behind the feedback (feedback only shown in the feedback phase). The phase depicted in Figure 1 was the reliance phase before the participant indicated his reliance decision.

3.3 Data Collection

During the above described experiment, input and output were logged using a server-client application. The interface of this application is shown in Figure 2. Two other client machines, that were responsible for executing the task as described in the previous subsection, were able to connect via a local area network to the server, which was responsible for logging all data and communication between the clients. The interface shown in Figure 2 could be used to set the client's IP-addresses and ports, as well as several experimental settings, such as how to log the data.

Experienced performance feedback of each trustee and reliance decisions of each participant were logged in temporal order for later analysis. During the feedback phase the given feedback was translated to a penalty of either 0, $.5$ or 1, representing a good, neutral or poor experience of performance, respectively. This directly maps to the value of $p^a(t)$ in the independent trust model. For the relative trust model however, a translation has been made to $E_i(t)$ whereby the value 0 has been mapped to $E_i(t) = -1$, $.5$ to $E_i(t) = 0$ and 1 to $E_i(t) = 1$. During the reliance phase the reliance decisions were translated to either 0 or 1 for each trustee S, which represented that one relied or did not rely on S.

3.4 Parameter Adaptation

The data collection described in Section 3.3 was repeated on each group of two participants twice, called condition 1 and condition 2, respectively. The data from one of the conditions was used

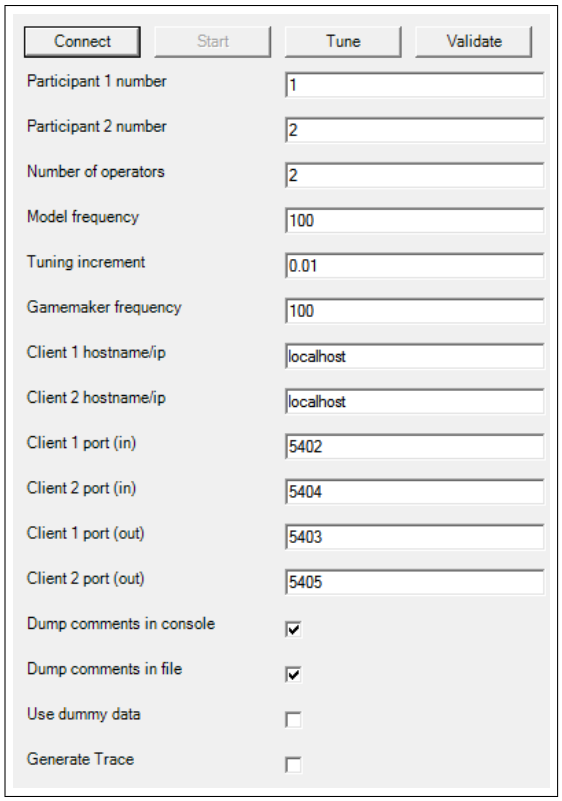

Figure 2. Interface of the application used for gathering validation data (Connect), for parameter adaptation (Tune) and validation of the trust models (Validate).

for parameter adaptation purposes for both models, and the data from the other condition for model validation (see Section 3.5). This process of parameter adaptation and validation was balanced over conditions, which means that condition 1 and condition 2 switch roles (i.e., parameter adaptation and model validation) for half of the validation efforts (i.e., cross-validation). Both the parameter adaptation and model validation procedure was done using the same application as was used for gathering the empirical data. The interface shown in Figure 2 could also be used to alter validation and adaptation settings, such as the granularity of the adaptation.

The number of parameters of the models presented in Section 2 to be adapted suggest that an exhaustive search for the optimal parameters is feasible (Hoogendoorn et al., 2009b). This means that the entire parameter search space is explored to find a vector of parameter settings resulting in the maximum accuracy (i.e., the amount of overlap between the model's predicted reliance decisions and the actual human reliance decisions) for each of the models. The corresponding code of the for this purpose implemented exhaustive search method is shown in Algorithm 1.

Algorithm 1 ES-PARAMETER-ADAPTATION(E, R_H)

1: $\theta_{\text{best}} \leftarrow 0$
2: **for all** parameter setting vectors X **do**
3: $\quad \theta_X \leftarrow 0$
4: $\quad$ **for all** time points t **do**
5: $\quad\quad e \leftarrow E(t)$
6: $\quad\quad r_M \leftarrow R_M(e, X)$
7: $\quad\quad r_H \leftarrow R_H(e)$
8: $\quad\quad$ **if** $r_M = r_H$ **then**
9: $\quad\quad\quad \theta_X \leftarrow \theta_X + 1$
10: $\quad\quad$ **end if**
11: $\quad$ **end for**
12: $\quad$ **if** $\theta_X > \theta_{\text{best}}$ **then**
13: $\quad\quad X_{\text{best}} \leftarrow X$
14: $\quad\quad \theta_{\text{best}} \leftarrow \theta_X$
15: $\quad$ **end if**
16: **end for**
17: **return** X_{best}

In this algorithm, $E(t)$ is the experience (i.e., performance feedback) at time point t, $R_H(e)$ is the actual reliance decision the participant made given a certain experience e, θ_X is the accuracy of a certain candidate parameter setting vector X, $R_M(e, X)$ is the predicted reliance decision of the trust model M (either independent or relative), given an experience e and candidate parameter setting vector X, θ_{best} is the accuracy of the best parameter setting vector X_{best} found so far, and is returned when the algorithm finishes. This parameter adaptation procedure was implemented in C#. Part of the C#-code is listed in the appendix of this paper, where the method "UpdateDistance()" corresponds to lines 5 until 10 in Algorithm 1 and $R_M(e, X)$ is calculated by the method "TrusteeWithMaxTrust()".

Here, if μ number of parameters with precision τ are to be estimated, N is the number of trustees, and B the number of reliance decisions (i.e., time points) made by human, then the worst case complexity of the method can be expressed as $O((10)^{\mu\tau} N B^2)$. In particular when $\mu = 3$ (i.e., the three parameters β, γ and η for the relative trust model) and also $\tau = 2$ (or precision .01), $N = 3$ and $B = 100$ then $3 \cdot 10^{10}$

steps are needed. For the independent trust model with only one parameter $3{\cdot}10^7$ steps are needed. The execution of an iteration took on average 3 minutes and 20 seconds.[1]

3.5 Validation

In order to validate the two models described in Section 2, the measurements of experienced performance feedback were used as input for the models and the output (predicted reliance decisions) of the models was compared with the the actual reliance decisions of the participant. The overlap of the predicted and the actual reliance decisions was a measure for the accuracy of the models. The results are in the form of dynamic accuracies over time, average accuracy per condition (1 or 2) and per trust model (independent or relative). A comparison between the averages per model and the interaction effect between condition role allocation (i.e., parameter adaptation either in condition 1 or 2) and model type, is done using a repeated measures analysis of variance (ANOVA).

3.6 Verification

Next to a validation using the accuracy of the prediction using the models, another approach has been used to validate the assumptions underlying existing trust models. The idea is that properties that form the basis of trust models are verified against the empirical results obtained within the experiment. In order to conduct such an automated verification, the properties have been specified in a language called Temporal Trace Language (TTL) (Bosse et al., 2009) that features a dedicated editor and an automated checker. The language TTL is explained first, followed by an expression of the desired properties related to trust.

Temporal Trace Language (TTL): The predicate logical temporal language TTL supports formal specification and analysis of dynamic properties, covering both qualitative and quantitative aspects. TTL is built on atoms referring to states of the world, time points and traces, i.e., trajectories of states over time. In addition, dynamic properties are temporal statements that can be formulated with respect to traces based on the state ontology Ont in the following manner. Given a trace γ over state ontology Ont, the state in γ at time point t is denoted by state(γ, t). These states can be related to state properties via the formally defined satisfaction relation denoted by the infix predicate $\models$, i.e., state(γ, t) $\models p$ denotes that state property p holds in trace γ at time t. Based on these statements, dynamic properties can be formulated in a formal manner in a sorted first-order predicate logic, using quantifiers over time and traces and the usual first-order logical connectives such as $\neg$, $\wedge$, $\vee$, $\Rightarrow$, $\forall$, and $\exists$. For more details on TTL, see (Bosse et al., 2009)

Properties for Trust Models: Within the literature on trust, a variety of properties have been expressed concerning the desired behavior of trust models. In many of these properties, the trust values are explicitly referred to, for instance in the work of (Jonker and Treur, 1998) characteristics of trust models have been defined (e.g., monotonicity, and positive trust extension upon positive experiences). In this paper however, the trust function is subject of validation and therefore cannot be taken as a basis. Therefore, properties are expressed on an external basis, solely using the information which has been observed within the experiment to see whether these behaviors indeed comply to the desired behavior of the trust models. This information is then limited to the experiences that are received as an input and the choices that are made by the human that are generated as output. The properties from (Hoogendoorn et al., 2009a) are taken as a basis for these properties. Essentially, the properties indicate the following desired behavior of human trust:

1) Positive experiences lead to higher trust
2) Negative experiences lead to lower trust
3) Most trusted trustee is selected

As can be seen, the properties also use the intermediate state of trust. In order to avoid this, it is however possible to combine these properties into a single property that expresses a relation between the experiences and the selection (i.e., the above items 1 + 3 and 2 + 3). Two of these properties are shown below. In addition, a property is expressed which specifies the notion of relativity in the experiences and the resulting selection of a trustee. The first property expresses that a trustee that gives the absolute best experiences during a certain period is

[1]This was on an ordinary PC with an Intel(R) Core(TM)2 Quad CPU @2.40 GHz inside.

eventually selected at least once within, or just after that particular period, and is shown below.

P1(min_duration, max_duration, max_time): Absolute more positive experiences results in selection

If a trustee a_1 always gives more positive experiences than all other trustees during a certain period with minimal duration min_duration and maximum duration max_duration, then this trustee a_1 is selected at least once during the period [min_duration, max_duration + max_time].

Formal:

P1(min_duration:DURATION, max_duration:DURATION, max_delay:DURATION) $\equiv$
$\forall\gamma$:TRACE, $t_{\text{start}}, t_{\text{end}}$:TIME, a:TRUSTEE
[[$t_{\text{end}} - t_{\text{start}} \geq$ min_duration & $t_{\text{end}} - t_{\text{start}} \leq$ max_duration
& absolute_highest_experiences(γ, a, t_{start}, t_{end})
$\Rightarrow$ selected(γ, a, t_{start}, t_{end}, max_delay)

where

absolute_highest_experiences(γ:TRACE, a:TRUSTEE, t_{start}:TIME, t_{end}:TIME) $\equiv$
$\forall t$:TIME, r_1, r_2:REAL, a_2:TRUSTEE $\neq a$
[[$t \geq t_{\text{start}}$ & t $< t_{\text{end}}$ &
state(γ, t) $\models$ trustee_gives_experience(a, r_1) &
state(γ, t) $\models$ trustee_gives_experience(a_2, r_2)]
$\Rightarrow r_2 < r_1$]

selected(γ:TRACE, a:TRUSTEE, t_{start}:TIME, t_{end}:TIME, z:DURATION) $\equiv$
$\exists t$:TIME [$t \geq t_{\text{start}}$ & $t < t_{\text{end}} + z$ &
state(γ, t) $\models$ trustee_selected(a)]

The second property, P2, specifies that the trustee which gives more positive experiences on average during a certain period is at least selected once within or just after that period.

P2(min_duration, max_duration, max_time, higher_exp): Average more positive experiences results in selection

If a trustee a_1 on average gives the most positive experiences (on average more than higher_exp better than the second best) during a period with minimal duration min_duration and maximum duration max_duration, then this trustee a_1 is selected at least once during the period [min_duration, max_duration+max_time].

Formal:

P2(min_duration:DURATION, max_duration:DURATION, max_delay:DURATION, higher_exp:REAL) $\equiv$
$\forall\gamma$:TRACE, $t_{\text{start}}, t_{\text{end}}$:TIME, a:TRUSTEE
[[$t_{\text{end}} - t_{\text{start}} \geq$ min_duration & $t_{\text{end}} - t_{\text{start}} \leq$ max_duration
& average_highest_experiences(γ, a, t_{start}, t_{end}, higher_exp)]
$\Rightarrow$ selected(γ, a, t_{start}, t_{end}, max_delay)]

where

average_highest_experiences(γ:TRACE, a:TRUSTEE, t_{start}:TIME, t_{end}:TIME, higher_exp:REAL) $\equiv$
$\forall t$:TIME, r_1, r_2:REAL, a_2:TRUSTEE $\neq a$
[$t \geq t_{\text{start}}$ & $t < t_{\text{end}}$ &
[$\sum_{\forall t:\text{TIME}}$ case(experience_received(γ, a, t, t_{start}, t_{end}, e), e, 0) >
($\sum_{\forall t:\text{TIME}}$ (case(experience_received(γ, a, t, t_{start}, t_{end}, e), e, 0)) + higher_exp * $t_{\text{end}} - t_{\text{start}}$)]]

In the formula above, the case(p, e, 0) operator evaluates to e in case property p is satisfied and to 0 otherwise.

experience_received(γ:TRACE, a:TRUSTEE, t:TIME, t_{start}:TIME, t_{end}:TIME, r:REAL) $\equiv$
[$\exists r$:REAL, $t \geq t_{\text{start}}$ & $t < t_{\text{end}}$ &
state(γ, t) $\models$ trustee_gives_experience(a, r)]

The final property concerns the notion of relativity which plays a key role in the models verified throughout this paper. The property expresses that the frequency of selection of a trustee that gives an identical experience pattern during two periods is not identical in case the other trustees give different experiences.

P3(interval_length, min_difference, max_time): Relative trust

If a trustee a_1 gives an identical experience pattern during two periods [t_1, t_1+ interval_length] and [t_2, t_2+ interval_length] and the experiences of at least one other trustee is not identical (i.e., more than min_difference different at each time point), then the selection frequency of a_1 will be different in a period during, or just after the specified interval.

Formal:

P3(interval_length:DURATION, min_difference:REAL, max_time:DURATION) ≡
$\forall\gamma$:TRACE, t_1, t_2:TIME, a:TRUSTEE
[[same_experience_sequence(γ, a, t_1, t_2, interval_length) &
$\exists a_2$:TRUSTEE $\neq a$
[different_experience_sequence(γ, a, t_1, t_2, min_difference)]
$\Rightarrow \exists i$:DURATION $<$ max_time
$\sum_{\forall t:\text{TIME}}$ case(selected_option(γ, a, t, $t_1 + i$,
$t_1 + i+$ interval_length), 1, 0) /
$(1 + \sum_{\forall t:\text{TIME}}$ case(trustee_selected(γ, t, t_1,
$t_1 + i+$ interval_length), 1, 0)) $\neq$
$\sum_{\forall t:\text{TIME}}$ case(selected_option(γ, a, t, $t_2 + i$,
$t_2 + i+$ interval_length), 1, 0) /
$(1 + \sum_{\forall t:\text{TIME}}$ case(trustee_selected(γ, t,
$t_2 + i$, $t_2 + i+$ interval_length), 1, 0))

where

same_experience_sequence(γ:TRACE, a:TRUSTEE,
t_1:TIME, t_2:TIME, x:DURATION) ≡
$\forall y$:DURATION [$y \geq 0$ & $y \leq x$ & $\exists r$:REAL
[state(γ, $t_1 + y$) $\models$ trustee_gives_experience(a, r) &
state(γ, $t_2 + y$) $\models$ trustee_gives_experience(a, r)]]

different_experience_sequence(γ:TRACE, a:TRUSTEE,
t_1:TIME, t_2:TIME, x:DURATION, min_difference:REAL) ≡
$\forall y$:DURATION [$y \geq 0$ & $y \leq x$ & $\exists r_1, r_2$:REAL
[state(γ, $t_1 + y$) $\models$ trustee_gives_experience(a, r_1) &
state(γ, $t_2 + y$) $\models$ trustee_gives_experience(a, r_2) &
$|r_1 - r_2| >$ min_difference]]

trustee_selected(γ:TRACE, t:TIME, t_{start}:TIME, t_{end}:TIME)
≡ $\exists a$:TRUSTEE
[$t \geq t_{\text{start}}$ & $t < t_{\text{end}}$ & state(γ, t) $\models$ trustee_selected(a)]

4 Results

In this section the results of the model validation and verification are given in Sections 4.1 and 4.2, respectively.

4.1 Validation Results

From the data of 18 participants, one dataset has been removed due to an error while gathering data. This means that there are 2 (condition role allocations, i.e., parameter adaptation either in condition 1 or 2) times 17 (participants) = 34 data pairs (accuracies for 2 models). Due to a significant Grubbs test, from these pairs 3 outliers were removed. Hence in total 31 pairs were used for the data analysis.

In Figure 3 the main effect of model type (either independent or relative trust) for accuracy is shown.

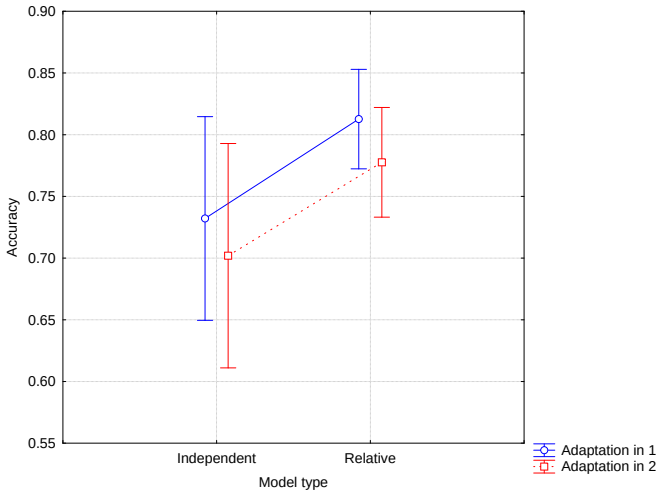

Figure 4. Interaction effect between condition role allocation and model type on accuracy.

A repeated measures analysis of variance (ANOVA) showed a significant main effect ($F(1, 29) = 7.60$, $p < .01$). This means that indeed the relative trust model had a higher accuracy ($M = .7968$, $SD = .0819$) than the independent trust model ($M = .7185$, $SD = .1642$).

Figure 4 shows the possible interaction effect between condition role allocation (parameter adaptation in condition 1 or in condition 2) and model type (either independent or relative trust) on accuracy. No significant interaction effect was found ($F(1, 29) = .01$, $p = .93$). Hence no significant learning effect between conditions was found. Cross-validation was not needed to balance the data, but the procedure still produced twice as much data pairs.

4.2 Verification Results

The results of the verification of the properties against the empirical traces (i.e., formalized logs of human behavior observed during the experiment) are shown in Table 1. First, the results for properties P1 and P2 are shown. Hereby, the value of max_duration has been kept constant at 30 and the max_time after which the trustee should be consulted is set to 5. The minimal interval time (min_duration) has been varied. Finally, for property P2 the variable higher_exp indicating how much higher the experience should be on average compared to the other trustees is set to .5.

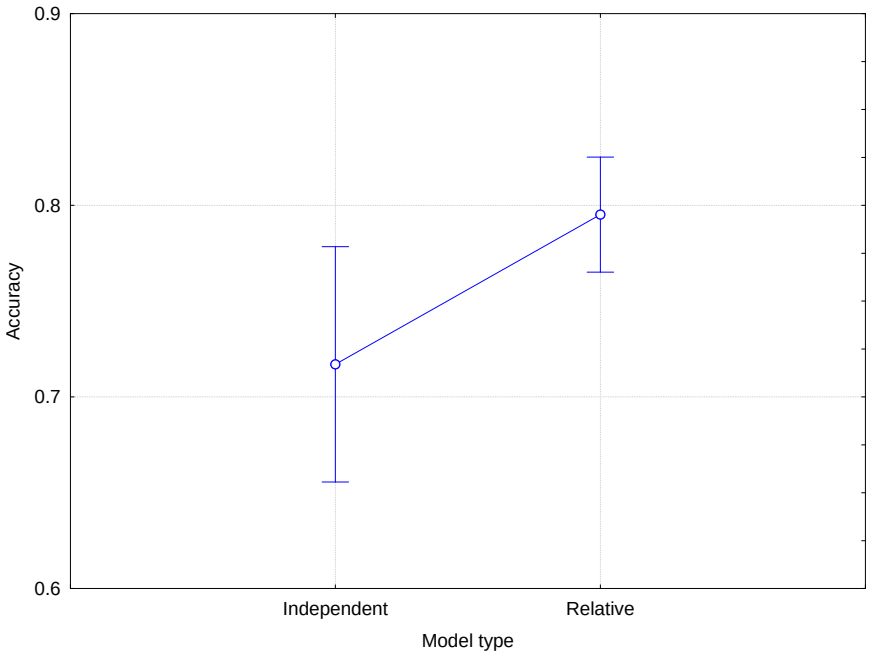

Figure 3. Main effect of model type for accuracy.

TABLE 1
RESULTS OF VERIFICATION OF PROPERTY P1 AND P2.

min_duration	% satisfying P1	% satisfying P2
1	64.7	29.4
2	64.7	29.4
3	86.7	52.9
4	92.3	55.9
5	100.0	58.8
6	100.0	70.6

The results in Table 1 indicate the percentage of traces in which the property holds out of all traces in which the antecedent at least holds once (i.e., at least one sequence with the min_duration occurs in the trace). This has been done to avoid a high percentage of satisfaction due to the fact that in some of the traces the antecedent never holds, and hence, the property is always satisfied. The table shows that the percentage of traces satisfying P1 goes up as the minimum duration of the interval during which a trustee gives the highest experience increases. This clearly complies to the ideas underlying trust models as the longer a trustee gives the highest experiences, the higher his trust will be (also compared to the other trustees), and the more likely it is that the trustee will be selected. The second property, counting the average experience and its implication upon the selection behavior of the human, also shows an increasing trend in satisfaction of the property with the duration of the interval during which the trustee on average gives better experiences. The percentages are lower compared to P1 which can be explained by the fact that they might also give some negative experiences compared to the alternatives (whereas they are giving better experiences on average). This could then result in a decrease in the trust value, and hence, a lower probability of being selected.

The third property, regarding the relativity of trust has also been verified and the results of this verification are shown in Table 2. Here, the traces of the participants have been verified with a setting of min_difference to .5 and max_time to 5 and the variable interval_length during which at least one trustee shows identical experiences whereas another shows different experiences has been varied.

It can be seen that property P3 holds more frequently as the length of the interval increases, which makes sense as the human has more time to perceive the relative difference between the two. Hence, this shows that the notion of relative trust can be seen in the human trustee selection behavior

TABLE 2
RESULTS OF VERIFICATION OF PROPERTY P3.

interval_length	% satisfying P3
1	0
2	41.1
3	55.9
4	67.6
5	66.7
6	68.4

in almost 70% of the cases.

5 DISCUSSION AND CONCLUSIONS

In this paper, an extensive validation study has been performed to show that human trust behavior can be accurately described and predicted using computational trust models. In order to do so, an experiment has been designed that places humans in a setting whereby they have to make decisions based upon the trust they have in others. In total 18 participants took part in the experiment. The results show that both an independent (see Maanen et al., 2007; Jonker and Treur, 1998) as well as a relative trust model (see Hoogendoorn et al., 2008) can predict this behavior with a high accuracy (72% and 80%, respectively) by learning on one dataset and predicting the trust behavior. Furthermore, it has been shown that the underlying assumptions of the trust models (and many other trust models) also show in the data of the participants.

Of course, more work on the validation of trust models has been performed. In (Jonker et al., 2004) an experiment has been presented in which human experiments in trust have been described. Although the underlying assumptions of trust models have to some extent been verified in that paper, no attempt has been made to fit a trust model to the data. Other papers describing the validation of trust models for instance validate the accuracy of trust models describing the propagation of trust through a network (e.g., Guha et al., 2004).

In the future research the considered parameter adaptation methods will be extended for the case of real-time adaptation, which accounts for human learning. Furthermore, a personal assistant agent will be implemented that is able to monitor and balance the functional state of the human in a timely and knowledgeable manner.

ACKNOWLEDGMENTS

This research was partly funded by the Dutch Ministry of Defense under progr. no. V929. Furthermore, this research has partly been conducted as part of the FP7 ICT Future Enabling Technologies program of the European Commission under grant agreement no. 231288 (SOCIONICAL). The authors would like to acknowledge Francien Wisse for her efforts to gather the necessary validation data and implementing the experimental task. The authors would also like to thank Tibor Bosse, Jan-Willem Streefkerk and Jan Treur for their helpful comments.

REFERENCES

Bosse, T., Jonker, C., Meij, L. v. d., Sharpanskykh, A., and Treur, J. (2009). Specification and verification of dynamics in agent models. *International Journal of Cooperative Information Systems*, 18:167–193.

Falcone, R. and Castelfranchi, C. (2004). Trust dynamics: How trust is influenced by direct experiences and by trust itself. In *Proceedings of the 3rd International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS'04)*, pages 740–747, New York, USA.

Guha, R., Kumar, R., Raghavan, P., and Tomkins, A. (2004). Propagation of trust and distrust. In *Proceedings of the 13th international conference on World Wide Web (WWW '04)*, pages 403–412, New York, NY. ACM.

Hoogendoorn, M., Jaffry, S., and Treur, J. (2008). Modeling dynamics of relative trust of competitive information agents. In Klusch, M., Pechoucek, M., and Polleres, A., editors, *Proceedings of the 12th International Workshop on Cooperative Information Agents (CIA'08)*, volume 5180 of *LNAI*, pages 55–70. Springer.

Hoogendoorn, M., Jaffry, S., and Treur, J. (2009a). Modelling trust dynamics from a neurological perspective. In *Proceedings of the Second International Conference on Cognitive Neurodynamics (ICCN'09)*. Springer Verlag. To appear.

Hoogendoorn, M., Jaffry, S. W., and Treur, J. (2009b). An adaptive agent model estimating human trust in information sources. In Baeza-Yates, R., Lang, J., Mitra, S., Parsons, S., and Pasi, G., editors, *Proceedings of the 9th IEEE/WIC/ACM*

International Conference on Intelligent Agent Technology (IAT'09), pages 458–465.

Jonker, C. M., Schalken, J. J. P., Theeuwes, J., and Treur, J. (2004). Human experiments in trust dynamics. In *Proceedings of the Second International Conference on Trust Management (iTrust 2004)*, volume 2995 of *LNCS*, pages 206–220. Springer Verlag.

Jonker, C. M. and Treur, J. (1998). Formal analysis of models for the dynamics of trust based on experiences. In Garijo, F. J. and Boman, M., editors, *Multi-Agent System Engineering, Proceedings of the 9th European Workshop on Modelling Autonomous Agents in a Multi-Agent World, MAAMAW'99*, volume 1647, pages 221–232, Berlin. Springer Verlag.

Maanen, P.-P. v., Klos, T., and Dongen, K. v. (2007). Aiding human reliance decision making using computational models of trust. In *Proceedings of the Workshop on Communication between Human and Artificial Agents (CHAA'07)*, pages 372–376, Fremont, California, USA. IEEE Computer Society Press. Co-located with The 2007 IEEE IAT/WIC/ACM International Conference on Intelligent Agent Technology.

Ramchurn, S. D., Huynh, D., and Jennings, N. R. (2004). Trust in multi-agent systems. *The Knowledge Engineering Review*, 19(1):1–25.

Sabater, J. and Sierra, C. (2005). Review on computational trust and reputation models. *Artificial Intelligence Review*, 24(1):33–60.

APPENDIX

In this appendix part of the C#-code that was used for parameter adaptation and model validation is listed and described. This part is called the DModel class and is used to calculate the trust values at a certain time point (that is, for one of the areas) and given a certain operator (either of the two operators), certain parameter settings (within certain intervals) and model type (either the independent or relative trust model). The in this appendix listed code consists of the following four methods:

1) **Dmodel:** This is the constructor of the DModel class, which initializes the independent or relative trust model. For each operator (indicated by "operatornumber") and each model type (indicated by "type") a DModel-object is created.
2) **UpdateValuesIndepdendentModel:** Calculates the independent trust value (called "data.values") for each trustee (the operator/-participant him- or herself, the other participant and the supporting software agent).
3) **UpdateValuesRelativeModel:** Calculates the relative trust value for each trustee. The "UpdateValues"-methods are called from the parent class of DModel for each time step.[2]
4) **UpdateDistance:** Calculates the distance (the inverse of the accuracy, depicted by θ_X in Algorithm 1) between the by the trust model predicted reliance decision (depicted by $R_M(e, X)$ in Algorithm 1) and the actual human reliance decision (depicted by $R_H(e)$ in Algorithm 1). For parameter adaptation purposes, this method is called from the parent class of DModel for each time step. This is done after either "UpdateValuesIndepdendentModel" or "UpdateValuesRelativeModel" is called to calculate the current trust values. In this procedure values for "data.dmodelParameters" are altered as is shown in Algorithm 1.
5) **TrusteeWithMaxTrust:** Calculates the trustee for which there is currently the maximum trust value. This method is called from the method "UpdateDistance()" in order to determine the predicted reliance decision. The current trust values are calculated either by the above described "UpdateValuesIndepdendentModel()" or the "UpdateValuesRelativeModel()" method.

Below the code of the DModel class is listed.

```
1  namespace UAVtrustServer
   {
3   /// <summary>
    /// This class was written by Waqar Jaffry
        and Peter-Paul van Maanen 2010. Nothing
         of this code may be used or copied
        without the permission of the authors.
        This software estimates the current
        trust of a UAV-operator (operator 1 or
```

[2]Due to limitations of space, the code of the parent class of DModel is omitted. Those further interested in this code are referred to `http://www.few.vu.nl/~pp/trust`.

```
    2) in different trustees (self, other
    and system).
/// </summary>

public class DModel : Model {
 double[] PositiveTrust;
 double[] NegativeTrust;

 public DModel(int opnr, Data d, bool tune)
  : base(0, opnr, d, tune) {
  // Constructor partly inherited from
      parent (code ommited)
  PositiveTrust = new double[data.values.
      GetLength(2)];
  NegativeTrust = new double[data.values.
      GetLength(2)];
 }

 // Method called to do iteration of the
     independent trust model to update to
     the next value
 public override void
     UpdateValuesIndependentModel() {
  int gridx, gridy;
  double totalpenalty, newtrust, oldtrust;
  double decay = data.dmodelParameters[
      operatornumber, 0];

  // Update trust for each trustee
  for (int trusteenr = 0; trusteenr < data.
      values.GetLength(2); trusteenr++) {
   // Use default values or values from the
       parameters file
   if (data.modelLoopNumber[operatornumber,
       type] == -1)
    data.values[operatornumber, type,
       trusteenr] = data.dmodelParameters[
       operatornumber, trusteenr + 1];
   else { // Otherwise calculate the new
       trust values
    totalpenalty = 0;

    // Update trust for each UAV
    for (int uavnr = 0; uavnr < data.
       currentUavWorld[operatornumber,
       type].uavs.Length; uavnr++) {
     gridx = data.currentUavWorld[
       operatornumber, type].lastfeedback
       [uavnr, 0];
     gridy = data.currentUavWorld[
       operatornumber, type].lastfeedback
       [uavnr, 1];

     totalpenalty += data.currentUavWorld[
       operatornumber, type].grid[gridx,
       gridy].penalty[0, trusteenr];
    }

    newtrust = totalpenalty / data.
       currentUavWorld[operatornumber,
       type].uavs.Length;
    oldtrust = data.values[operatornumber,
       type, trusteenr];
    data.values[operatornumber, type,
       trusteenr] = decay * oldtrust + (1
       - decay) * newtrust;
    data.dmodelParameters[operatornumber, 1
       + trusteenr] = data.values[
       operatornumber, type, trusteenr];
       // For storing the last value as
       parameter
   }
  }
 }

 // Method called to do iteration of the
     relative trust model to update to the
     next value
 public override void
     UpdateValuesRelativeModel() {
  double[] deltaPositiveTrust;
  double[] deltaNegativeTrust;
  double SigmaPositiveTrust = 0,
      SigmaNegativeTrust = 0;
  deltaNegativeTrust = new double[data.
      values.GetLength(2)];
  deltaPositiveTrust = new double[data.
      values.GetLength(2)];
  int gridx, gridy;
  double Gama = data.dmodelParameters[
      operatornumber, 0];
        double Beta = data.
            dmodelParameters[
            operatornumber, 1];
        double Eta = data.
            dmodelParameters[
            operatornumber, 2];
  double INTERVAL_LENGTH = 0.1;

  // Use default values or values from the
      parameters file (after adaptation)
        if (data.modelLoopNumber[
            operatornumber, type] == -1)
             {
   PositiveTrust[0] = data.dmodelParameters
       [operatornumber, 3];
   NegativeTrust[0] = data.dmodelParameters
       [operatornumber, 4];
   PositiveTrust[1] = data.dmodelParameters
       [operatornumber, 5];
   NegativeTrust[1] = data.dmodelParameters
       [operatornumber, 6];
   PositiveTrust[2] = data.dmodelParameters
       [operatornumber, 7];
   NegativeTrust[2] = data.dmodelParameters
       [operatornumber, 8];
  }
  else {
   // Update trust value for each UAV
   for (int uavnr = 0; uavnr < data.
       currentUavWorld[operatornumber, type
       ].uavs.Length; uavnr++) {
    double penalty = 0;
    gridx = data.currentUavWorld[
       operatornumber, type].lastfeedback[
       uavnr, 0];
```

```
75      gridy = data.currentUavWorld[
            operatornumber, type].lastfeedback[
            uavnr, 1];

77      SigmaPositiveTrust = 0;
        SigmaNegativeTrust = 0;
79      for (int truseern = 0; truseern < data.
            values.GetLength(2); truseern += 1)
             {
         SigmaPositiveTrust += PositiveTrust[
             truseern];
81       SigmaNegativeTrust += NegativeTrust[
             truseern];
        }
83
        // Update trust value for each trustee
85      for (int CurrentTrustee = 0;
            CurrentTrustee < data.values.
            GetLength(2); CurrentTrustee += 1)
            {
         penalty = data.currentUavWorld[
             operatornumber, type].grid[gridx,
             gridy].penalty[0, CurrentTrustee];
87
         if (penalty < 0.5) {
89        deltaPositiveTrust[CurrentTrustee] =
              Beta * (Eta * (1 - PositiveTrust[
              CurrentTrustee]) - (1 - Eta) * (1
               - data.values.GetLength(2) *
              PositiveTrust[CurrentTrustee] / (
              SigmaPositiveTrust)) *
              PositiveTrust[CurrentTrustee] *
              (1 - PositiveTrustee
              ])) * INTERVAL_LENGTH;
          deltaNegativeTrust[CurrentTrustee] =
              (-(1 - Gama) * NegativeTrust[
              CurrentTrustee] * INTERVAL_LENGTH
              );
91       }
         else if (penalty == 0.5) {
93        deltaPositiveTrust[CurrentTrustee] =
              (-(1 - Gama) * PositiveTrust[
              CurrentTrustee] * INTERVAL_LENGTH
              );
          deltaNegativeTrust[CurrentTrustee] =
              (-(1 - Gama) * NegativeTrust[
              CurrentTrustee] * INTERVAL_LENGTH
              );
95       }
         else {
97        deltaNegativeTrust[CurrentTrustee] =
              Beta * (Eta * (1 - NegativeTrust[
              CurrentTrustee]) - (1 - Eta) * (1
               - data.values.GetLength(2) *
              NegativeTrust[CurrentTrustee] / (
              SigmaNegativeTrust)) *
              NegativeTrust[CurrentTrustee] *
              (1 - NegativeTrust[CurrentTrustee
              ])) * INTERVAL_LENGTH;
          deltaPositiveTrust[CurrentTrustee] =
              (-(1 - Gama) * PositiveTrust[
              CurrentTrustee] * INTERVAL_LENGTH
              );
99       }
        }
101
         for (int CurrentTrustee = 0;
             CurrentTrustee < data.values.
             GetLength(2); CurrentTrustee += 1)
             {
103       PositiveTrust[CurrentTrustee] +=
              deltaPositiveTrust[CurrentTrustee
              ];
          NegativeTrust[CurrentTrustee] +=
              deltaNegativeTrust[CurrentTrustee
              ];
105      }
        }
107    }

109    for (int CurrentTrustee = 0;
           CurrentTrustee < data.values.
           GetLength(2); CurrentTrustee++) {
        data.values[operatornumber, type,
            CurrentTrustee] = (PositiveTrust[
            CurrentTrustee] - NegativeTrust[
            CurrentTrustee] + 1) / 2;
111     data.dmodelParameters[operatornumber,
            CurrentTrustee * 2 + 3] =
            PositiveTrust[CurrentTrustee]; //
            For storing the last value as
            parameter
        data.dmodelParameters[operatornumber,
            CurrentTrustee * 2 + 4] =
            NegativeTrust[CurrentTrustee]; //
            For storing the last value as
            parameter
113    }
      }
115
      // Update the distance between the
          generated model output (either
          independent or relative) and the
          validation data for one time step
117   // The trust values (data.values) have
          already been updated for the 3
          trustees given the current parameter
          settings (data.dmodelParameters)
      public override void UpdateDistance() {
119    int gridx, gridy, distance, maxtrust;

121    // Update the distance to the proper
           value
       for (int uavnr = 0; uavnr < data.
           currentUavWorld[operatornumber, type
           ].uavs.GetLength(0); uavnr++) {
123     gridx = data.currentUavWorld[
            operatornumber, type].lastfeedback[
            uavnr, 0];
        gridy = data.currentUavWorld[
            operatornumber, type].lastfeedback[
            uavnr, 1];
125
       maxtrust = TrusteeWithMaxTrust();
127
```

```
      // distance == 0 when indeed the human
          relied on the trustee with the
          highest model's estimated trust
          value, otherwise the distance is
          higher (1 or 2)
129   if (data.currentUavWorld[operatornumber,
          type].grid[gridx, gridy].reliedon
          [0, maxtrust] == 1)
       distance = 0;
131   else if (data.currentUavWorld[
          operatornumber, type].grid[gridx,
          gridy].reliedon[0, 0] != 1 && data.
          currentUavWorld[operatornumber, type
          ].grid[gridx, gridy].reliedon[0, 1]
          != 1 && data.currentUavWorld[
          operatornumber, type].grid[gridx,
          gridy].reliedon[0, 2] != 1)
       distance = 0;
133   else
       distance = 1;
135
      // Update value according to the given
          distance
137   data.modelDistance[operatornumber, type]
          += distance;
     }
139  }

141  // Return the trustee (0, 1 or 2) with the
        most trust according to the current
       trust values (data.values)
     public int TrusteeWithMaxTrust() {
143   int index = 0;
      for (int trusteenr = 1; trusteenr < data.
          values.GetLength(2); trusteenr++) //
          for all trustees
145   if (data.values[operatornumber, type,
          trusteenr] > data.values[
          operatornumber, type, index])
       index = trusteenr;
147
      return index;
149  }
    }
151 }
```

dmodel.cs