Improving User Satisfaction in a Ubiquitous Computing Application

Kevin Eustice, Amir Mohsen Jourabchi, Jason Stoops, and Peter Reiher, *Member, IEEE*

***Abstract*—The Smart Party is a ubiquitous computing application based on the Panoply middleware. The Smart Party allows attendees at a party to transparently participate in the selection of music played at the party. The methods used to select music, based on the preferences of the partygoers, has a substantial impact on how satisfied these partygoers will be. This paper examines different algorithms for selecting music in a Smart Party, and discusses lessons from the research that are applicable to other socially-based ubicomp applications.**

***Index Terms*—ubiquitous computing, social computing, user satisfaction.**

I. Introduction

The Internet is revolutionizing the social interactions of its users, and the emerging ubiquitous computing environment is only a step behind. In the near future, ubiquitous computing will offer new and enhanced ways for people to meet, work together, and cooperate in a wide variety of activities. Such ubicomp applications will succeed by offer perceived value to their users, both by enabling new activities and applications, but also through improving our existing activities. For these latter applications, we must find ways to quantify their benefits and evaluate various techniques designed to increase application benefits.

Many social ubicomp applications are designed to help people interact in groups. Some groups, such as a school class or a club, are predefined, while others can be formed opportunistically, such as users in an area of poor Internet connectivity who pool their computing and data resources for the common good. By using common characteristics and goals to organize groups of users, ubicomp applications can improve both the individual and overall user experience.

This paper examines one specific illustrative example of using ubicomp technology to form users with common interests into groups, resulting in an improved social experience. The application is called the Smart Party [1]. The Smart Party application gathers musical preferences for guests attending a party in a user's house. Based on their preferences and available media, it chooses a music play list, adjusting to changing membership as guests come and go. The preferences and the actual music media are gathered from the guests' portable devices, giving a broader selection of possible music to play than that belonging to the host.

Manuscript received June 30, 2008. This work was supported in part by the U.S. National Science Foundation under Grant CNS 0427748.

Kevin Eustice, Jason Stoops, and Peter Reiher are with the Department of Computer Science, UCLA, Los Angeles, CA 90095 USA (310-825-8332; fax: 310-825-2273; e-mail: reiher@ cs.ucla.edu).

Amir Mohsen Jourabchi was also with UCLA. He is now with Parasoft (e-mail: amirmohsenj@yahoo.com).

The goal of the Smart Party application is to provide a cooperative, satisfying musical experience at a party. By varying the algorithm used to select which song to play in which room, one varies both individuals' average satisfaction with the party and the distribution of fairness among individuals. Similarly, examining different user strategies for deciding whether to stay in a particular room or move to another room can affect satisfaction and fairness. In this paper, we examine the effects on satisfaction and fair distribution of satisfaction among partygoers based on our exploration of these alternatives in simulation.

The paper is organized as follows. Section 2 describes the Smart Party application in more detail. Section 3 describes our simulation approach. Section 4 presents simulation results for various single room song selection algorithms. Section 5 describes the effects of making multiple rooms available to the guests of a Smart Party. Section 6 discusses related work. Section 7 describes future directions and discusses how we can generalize the results of these experiments to more general ubicomp social applications.

II. The Smart Party

The Smart Party was built using the Panoply ubiquitous computing middleware that we have developed [2]. This framework provides strong support for group formation and cooperation in ubiquitous environments.

In the Smart Party, a group of people attend a gathering hosted at someone's home. Each person carries a small mobile device that stores its owner's music preferences and song collection. The party environment consists of a series of rooms, each equipped with speakers. The home is covered by one or more wireless access points.

As each guest arrives, his mobile device securely and automatically associates with the correct network to connect it to the Smart Party infrastructure. As party attendees move within the party environment, each room programs an audio playlist based on the communal music preferences of the current room occupants, and the content they have brought to the party. Guests automatically and dynamically collaborate with the host network, which manages their collective preferences and steers the music choices. Decisions are based on information already in the users' devices, so no user intervention or input is required. **Figure 1** shows an example of a fully formed and configured Smart Party.

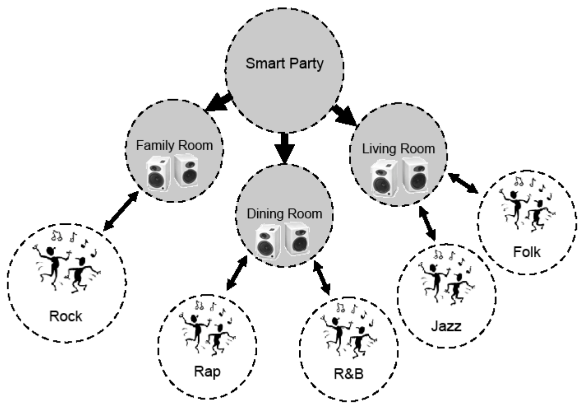

Figure 1. A fully configured Smart Party

Each time a room needs to select a song to play, it consults the mobile devices belonging to the users in that room. Based on their musical preferences (which are derived from information on their devices about which songs they listen to frequently), the Smart Party gathers information about candidate songs to play and uses some algorithm to choose a song. The simplest algorithm is round robin, which allows users in a room to take turns choosing a song. Each user gets to hear the song he most wants to hear, but perhaps the selections will not be pleasing to any of the other users, which can result in poor overall satisfaction.

A more sophisticated algorithm is to have users nominate songs, then vote on them proportionally to their liking of each nominated song. As we will see in section IV, this algorithm produces more satisfaction and greater fairness. (An important point for this and other algorithms is that measures are required to ensure that the same song is not played over and over.) A further enhancement is to analyze the preferences of users in the room and automatically form them into groups that share musical tastes. The group decides on the song its members would, collectively, like best, and the group pools its votes for that song. Group membership can change dynamically as partygoers come and go. This improves satisfaction and fairness over simple voting, as it tends to lead to popular compromises.

III. The Smart Party Simulation

We have built a working prototype of the Smart Party in our lab, but doing large scale testing on this prototype is impractical. We have instead performed testing of song selection algorithms in a simulation framework. This simulation is specific for this purpose, so it does not replicate all elements of the application or Panoply. For example, it does not simulate the localization mechanisms used to determine where users are, nor the protocols used to transfer media data from a user device to the Smart Party's speakers, though these and all other elements of the Smart Party are fully implemented in the real application. For more details on the actual application, see [1].

The simulation uses real user preference data gathered from LastFM, which is a web site that allows users to upload their media preferences from a variety of sources. This web site records which tracks the user has heard, and play counts for these tracks. For each simulated Smart Party, a random subset of users and media is selected. Many different random selections are simulated. The basic simulation runs the selection algorithm 30 times for each simulated party. Assuming 4 minute songs, this would yield a two hour party. For each selection, the simulation calculates the satisfaction for each user. Simulations were run for many different sizes of parties, from a small party of 4 users to a large party of 80. To minimize the random effects based on selecting some particular set of users and songs, we performed multiple runs with different sets of users and songs for each scenario investigated. We here report median results from the several runs of each scenario.

Song preference is measured on a scale of zero to five. The satisfaction gained from a song with a k-rating is 2^k (except songs with a 0 rating, which yield a satisfaction of 0). To determine a song's rating from the LastFM data, songs were separated into 5 buckets by play count. The distribution of songs into buckets ended up following an exponential curve, with the number of songs in the top bucket (the songs most often played) being smallest, and the bottom two buckets (songs that were played rarely) containing the majority of the songs. If a song is not in a user's profile, it is considered to have a 0 rating, and no satisfaction is gained by hearing the song at the party. In reality, a user is sometimes pleased by a song he has never heard before, but we have no realistic model of this effect, so we conservatively assume that unrated songs will not be liked.

In the party simulation, overall satisfaction is the sum of the satisfaction gained during the party by all users.

The fairness of the distribution of satisfaction is quantified by calculating the Gini Coefficient [3]. The Gini Coefficient is widely used as a measurement of the distribution of wealth in a population. It is a ratio between 0 and 1. A Gini Coefficient of 0 expresses perfectly equal distribution of available wealth among a population; a value of 1 expresses perfectly unequal distribution, where all wealth is held by one member of a population and others have no wealth.

Instantaneous fairness is a measure of the equality of the distribution of satisfaction gained by guests in a single round. Overall fairness is a measure of the equality in distribution of overall satisfaction among guests.

IV. Selecting Songs for a Single Room

We examined several algorithms by which users in a single room can participate in choosing what music to play. In the Round Robin algorithm, each user in turn is allowed to pick his favorite song. In the Non-cooperative Voting algorithm, users nominate songs and vote for the various songs nominated, based on relative personal preference. In Sphere-based Voting, users with common musical tastes form a Panoply group (called a *sphere of influence*) that effectively allows them to pool their influence to get more votes for something they will all like.

While Round Robin sounds reasonable, even a little thought will reveal some potential problems. In essence, Round Robin allows each partygoer to periodically choose the song to be played, which presumably would be that partygoer's personal favorite. However, it might well be that all the other partygoers do not care for that song at all. One user gets excellent satisfaction in each round, but the others might not get any. Thinking aesthetically, the result might well be a party in which the music played swings wildly from one genre or artist to another completely different one. An algorithm that seeks to balance the preferences of the many will probably do better.

For the two voting algorithms described, however, it is less clear which would make users happier. Is it better for each user to propose a song, then have all users vote on them? Or is it better for users with similar tastes propose songs their group will like, and then vote?

First, we must describe these voting algorithms in a little more detail. In the Non-cooperative Voting algorithm, each guest suggests his or her favorite song. That song is added to the list of the candidates that is submitted to all the guests for voting. Guests vote for a song by consulting their preference list. If that song or its artist exists in their preference list, they submit their rating for that candidate as their vote. Otherwise, their rating will be zero, which translates to a vote of zero for that song. Once all the votes are collected, the Smart Party sorts the candidates by their scores, from highest to lowest. It drops the bottom half of the list and re-submits the list of the candidates to the guests for another round of voting. This procedure is repeated until one song wins and is played.

In Sphere-based Voting, people with common interests form groups and submit their votes through the group leader called the "vote coordinator." The vote coordinator is the guest who has formed the group. Each vote coordinator chooses a set of preferred artists, which are the "goals" for that group. When a guest enters a room it looks for existing vote coordinators and checks to see if it is attracted to any of those groups by comparing the artists in his/her preference list against those in the group's goal set. If the attraction value is higher than a certain threshold, that guest joins the group. If not, the guest forms his own group, possibly attracting other guests to join him. When a vote is called for, group members can only suggest songs whose artists match the group's goals. Otherwise, the procedure is similar to Non-cooperative Voting. The groups are not static. Guests can leave their group and join others if the current set of goals for another group better matches their own tastes.

Figure 2 shows the satisfaction results for the two voting algorithms. Each bar shows the median of 150 different simulations, and the error bars show the first and third quartiles of the data.

For small Smart Parties of less than 8 users, round-robin wins, as users are able to, in turn, select their favorite songs. Because we use a 2^k model for calculating satisfaction, perfect satisfaction of one user ($k=5$), with no satisfaction for other users, gives a total satisfaction of 32. For four users, choosing a song that gives a 3 satisfaction for all of them yields the same level of satisfaction. So unless all users at the small party have some fairly strong tastes in common, the satisfaction metric we use would tend to favor satisfying someone perfectly, rather than a few users imperfectly. Thus, as expected, both Non-cooperative Voting and Sphere-based Voting perform worse than Round Robin. Examination of some cases shows that both result in the selection of lower-ranked songs shared in common. As more users participate, Non-cooperative Voting and Sphere-based Voting result in higher satisfaction because the presence of an increasing number of "popular" songs causes in an upswing in satisfaction, and a larger number of somewhat satisfied users ultimately overwhelms the benefit of a single very satisfied user.

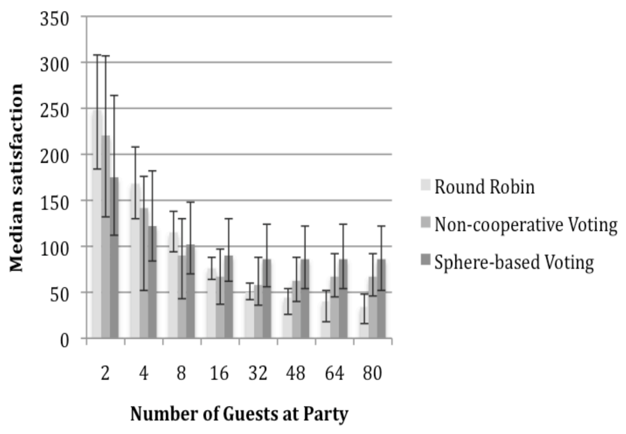

Figure 2. Overall Smart Party satisfaction for various song selection algorithms

Consider fairness for the same algorithms. **Error! Reference source not found.** shows the median overall fairness for the same parties as in **Figure 2**. Fairness here is averaged over an entire party, with the median selected from the 150 different parties tested for each size and algorithm. Fairness, as discussed in Section III, is based on the Gini coefficient, so lower numbers indicate greater fairness than higher numbers. As with satisfaction, the error bars indicate the first and third quartiles.

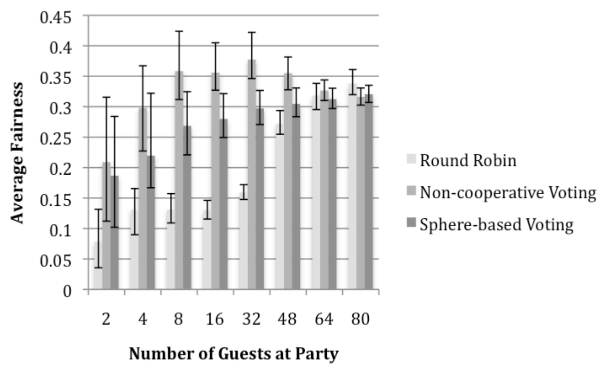

Figure 3. Median overall party fairness

Note that Round Robin is extremely fair for small parties, on an overall party basis. In essence, each guest gets his turn choosing a song, and thus has a fair share of the overall selection of the music at the party. Since only 30 songs are played at the party, when there are more than 30 guests, some guests never get to choose a song, and, unless they fortuitously like music chosen by others, they end up receiving much less satisfaction than those who get to choose a song. As a result, fairness for Round Robin selection drops substantially as the parties get larger. However, overall party fairness is no better for the other algorithms, even at large parties. If one is perpetually in the losing voting block, the party will not seem particularly fair.

Fairness can be computed on a whole party basis, as in **Figure 3**, or on a round-by-round basis. The latter is perhaps more meaningful, since if many partygoers perceive a few rounds of a party as being unfair to their interests, they might well leave. **Figure 4** shows median instantaneous (round-by-round) fairness for the same parties as discussed above.

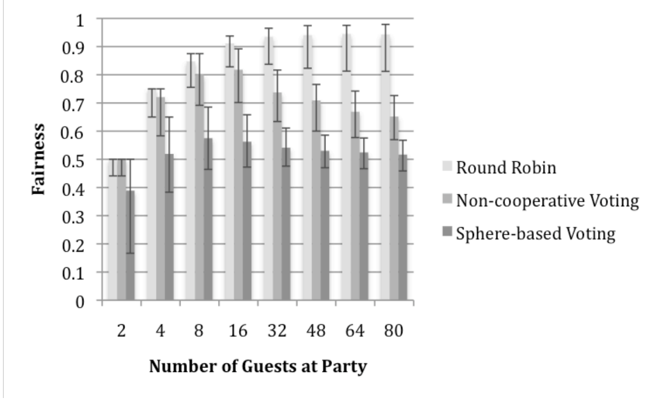

Figure 4. Instantaneous fairness

Round Robin's turn-taking in 2 person parties tends to cause one person to be happy and the other unhappy in each round, resulting in a fairness of .5 for a typical round. As the party gets bigger, one waits more rounds for one's turn, stewing in dissatisfaction all the while. This is reflected by the round-based fairness getting worse and worse as party size grows. While we see some of this effect for the voting algorithms, for sphere-based voting, round-based fairness more or less stabilizes at .5. Those in the winning voting group are generally happy in each round, and those out of it are generally unhappy. Of course, even within the winning group, some members like the chosen song more than others, so there is a further satisfaction skew that worsens the fairness. Still, these round-based fairness results suggest that a group-based voting scheme will be less likely to drive away partygoers in the midst of a party.

V. Considering User Movement

Another choice an unsatisfied user has at a Smart Party is to move to another room where the music might be more to his liking. To evaluate the effects that guest movement has on the Smart Party experience, we investigated methods of improving user satisfaction based on user mobility.

We developed several mobility models. These models are, at their core, sets of rules that specify when and to which room a guest should move. These rules only incorporate information that the guest has available to them in the party; they do not require any oracular orchestration or knowledge.

In all experiments, guests are initially distributed evenly between available rooms. The actual Smart Party prototype does not synchronize the playing of songs in different rooms, so rooms may call for votes and play songs asynchronously, and guests may move between rooms at any time. However, there is no song length information associated with the Last.FM data we use for our simulation, and the simulation is simplified by treating all votes for a song round as simultaneous and synchronous. As a result, all songs are assumed to be the same length, and the party operates in phases, with guests moving, voting and listening in lockstep.

The following mobility models were tested in the Smart Party simulation framework:

- **No movement** – Guests never move for the duration of the party.
- **Random movement** – Guests pick a room at random after each song.
- **Threshold-based random movement** – Guests track several previously heard songs, and if the average satisfaction gained from these songs drops below some threshold, the guest chooses a room at random. Otherwise, the guest does not move.
- **Threshold-based to room with highest satisfaction** – If unsatisfied with the current room, the guest looks at the last several songs played in all rooms and moves to the room with the highest average satisfaction over this historical data.

The mobility models are compared with populations of 18, 30, and 60 guests in 3, 5, and 10 rooms respectively. In each scenario, there were thus an average of six guests per room, and the party starts with exactly six guests in each room, placed randomly. For all except the No-movement case, no restrictions were subsequently placed on the number of guests in a room. In all cases, songs were selected using the Non-cooperative Voting algorithm.

Figure 5 shows that offering music selections in multiple rooms results in much more satisfying parties. The No-movement case is roughly comparable to the satisfaction seen in **Figure 2** for the Non-cooperative Voting algorithm, with slight differences because, effectively, the No-movement case has several independent six-guest rooms, while the **Figure 2** results were plotted for 4 and 8 guests.

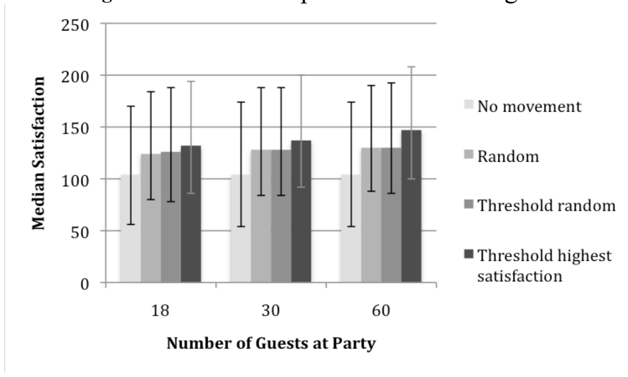

Figure 5. Median satisfaction for different movement algorithms

Once we allow even the most basic of movement algorithms, satisfaction increases dramatically. In the Random-movement case, each user randomly chooses a new room to move to at the end of each song. There is no attempt to choose a "good" room or move in concert with other users sharing similar tastes. Yet even this non-intelligent movement algorithm results in a 20% or better improvement in satisfaction.

Why? Any guest movement "stirs up" the state of the party, making previously unselectable songs more likely to be chosen. For example, consider three users in three different rooms who have the same favorite song, but no other users have heard of this song. These users may not be able to get the song played in their separate rooms. However, when users can move between rooms, perhaps at some point these three users find themselves in the same room. Suddenly, these three guests now have enough votes to get the song played, thus increasing their overall satisfaction.

The Threshold-based Random and pure Random movement algorithms produce nearly identical results. One might have expected that having satisfied users stay where they are and only unsatisfied users switching to a new room would be much better. However, the "stirring up" effect is more valuable. One characteristic of the Smart Party is that it is biased against replaying songs during a single party, which means that there is a decreasing chance that a user who is satisfied in one location will remain equally satisfied there as time goes by. The songs he likes that he can get played in that room, with the cooperation of others in the room, are gradually exhausted. He is eventually more likely to find opportunities to play the songs he likes in a randomly chosen room than he is in the room he currently occupies.

The best algorithm is to move to a room that has been playing the music you like best, when you are not satisfied with the music you are currently hearing. This algorithm can provide up to 41% improvement in median satisfaction over the No-movement case.

This result is not surprising. That other room contains guests who probably share your musical tastes, and, since the guests there who are winning the elections are likely happy with the chosen music, they will stay there. (The use of a threshold to prevent movement when the differences are only trivial amplifies this effect.) Therefore, you are likely to hear music in the future similar to the music you have heard in the past. Without actually forming groups, this algorithm achieves a similar effect to that shown in **Figure 1** without ever trying to identify genres or groups of users with similar preferences. Just as in that figure, the party moves to a state where one room plays R&B, another plays hip-hop, a third plays oldies, and so on. As **Figure 5** also shows, this effect improves as you have more rooms available, since now there is an opportunity for more specialized groups to form based on tastes that are narrow, but perhaps deeply felt.

This model performs best with a shorter history length (two songs). This is because the amount of movement occurring in the party causes the history data to become stale quickly. Since roughly four in ten guests change rooms every round under the satisfaction-based model, a song that played in a room five rounds ago is likely a very poor indicator of what song will be played next in that room.

We also tested two models where movement was based on the number of guests in a room, on the theory that moving to a lightly populated room would give a user a higher chance of having his preferred songs selected. These models did not perform even as well as random movement, though they were still significantly better than not moving at all. By examining the movements that would occur in a threshold-based model that moved to the least crowded room every time, we see why this type of selection creates a problem. At the beginning of the party, all guests are evenly distributed between rooms. Therefore, when a guest's satisfaction drops below the threshold, the guest leaves the room, choosing a room at random since there is a tie between all rooms in the party for the least-populated room. The next guest to have their satisfaction drop below the threshold and desire a room change is then forced into the room that the first guest just left, since it is now the least crowded room. Therefore, only half of the guests moving in the party get a room choice, and the other half are forced into the room just abandoned.

Fairness is significantly improved by user mobility, as one might guess. Users who are perpetually unhappy when forced to stay in one room can, instead, go somewhere else. **Figure 6** plots the median overall party fairness values for the 18 guest, three room case only, since the other cases produced similar results. All movement algorithms produce much better fairness than no movement, with the algorithm calling for users to move to the most congenial room doing best. . This is likely to mirror real human behavior. If one hates the music one is hearing in one room, and one knows different music is playing in another room, what could be more natural than trying that other room?

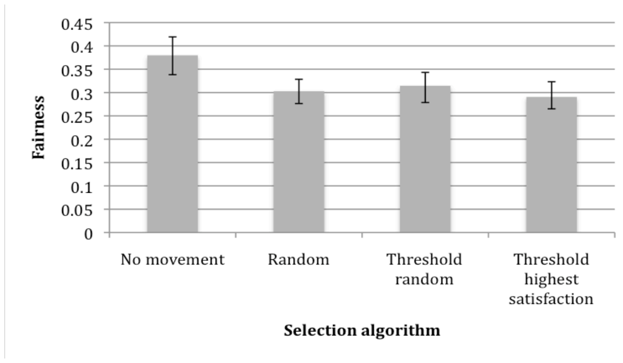

Figure 6. Median overall fairness for movement algorithms

VI. Related Work

Other projects have investigated issues of fairness and user satisfaction for group-based music experiences. The most related work is MusicFX [4] and FlyTrap [6], which both explore adaptive social music experiences situated in physical environments. Both projects focus on equitable music selection within a shared environment. MusicFX and FlyTrap both use RFID-based detection of user badges to determine user presence, and then activate user agents on a centralized server that represent the users and vote on their behalf for tracks (FlyTrap) or music channels (MusicFX). Follow-on work to MusicFX [7] investigates the use of an economics-based model to improve the fairness of overall channel selection. Extra vote weight is given to individuals who are forced to listen to non-preferred music. The use of the Gini coefficient by Prasad and McCarthy [7] directly inspired our use of the same technique. The economic scheme proposed by [7] could be adapted to our Smart Party to improve overall fairness.

Our Smart Party differs in several ways from these other projects. We support a Smart Party consisting of multiple environments with dynamic membership. Members may bring in new music via their mobile devices, and members coordinate their voting based on dynamically-built preference groups.

Another related project is Poolcasting, [8], a web-based social radio system. In Poolcasting, users attach to the system, share music, and create radio channels. The Poolcasting system schedules music for the channels using a metric that incorporates satisfaction, fairness, and variety measures to select the next song for each channel. To ensure musical continuity, Poolcasting leverages a large pool of user-generated playlists to identify related songs. This is an interesting feature and one that we are considering incorporating in the future.

Additionally, a group at the International School of New Media at the University of Lübeck in Germany has developed what they refer to as "Campus Party," or the "Smart Party" [9]. The Campus Party also uses RFID-based user detection, as well as a centralized database of user preferences and static music repository. The Campus Party uses a simple preference evaluation heuristic to select the current "best" song based on current occupancy of a room. The details of their selection algorithm are not described in their paper. Again, we differ from this work in terms of our support for multiple connected environments, coordinated voting within preference groups, dynamic membership, etc.

VII. Future Work and Conclusions

The Smart Party is one example of using ubiquitous computing techniques to augment a social experience. Many more such applications will be developed. The lessons we can learn about general techniques to improve how these applications do their jobs are more important than the particular improvements seen in the Smart Party itself, as are lessons related to the limitations of the technology.

This investigation into improving user satisfaction and fairness in the Smart Party offers several general lessons. First, using ubiquitous technology to help people cooperate is a good idea. Song selection mechanisms that either explicitly or implicitly led to groupings of users with shared tastes not only improved overall satisfaction, but also fairness. Since ubicomp technology can augment existing human capabilities to find congenial companions, by proper leveraging of this technology we can help people join in groups that they might not otherwise realize are possible or worthwhile.

Second, choices are good. Algorithms that do not include some capability for human choice to operate tend not to perform as well. This is most obvious in the room movement algorithms, where a postulated human choice to move to another room greatly improves satisfaction and fairness. As mentioned earlier, the key point is giving users different options, in our case rooms that play different kinds of music. The group algorithms also point in this direction. While the current algorithms are based purely on computer-available information, algorithms that incorporate human input in group formation are likely to do even better.

Third, even fairly simple algorithms can offer substantial benefit. Random movement works quite well in a multiroom case, and the only slightly more complex algorithm based on observing recent song selections is even better. While the more complex group formation algorithm is better than simple voting, even a basic voting algorithm does well for the Smart Party. Other ubiquitous computing social applications are likely to see similar effects, suggesting that it is always a good idea to investigate the simple ways to improve the application's performance before worrying about complex algorithms.

But this work does have its limits, as well. Our measurements of satisfaction and fairness are all simulated, based on past human behavior. People are, however, much more complicated than that, and we would thus expect real Smart Parties to display much different human behavior than we see in our simulations. Tastes vary based on circumstances, and there are other reasons to stay in or leave a room than whether you like the music. Methods of more accurately judging how well human users are served by their ubicomp applications are clearly necessary. More generally, it is vital to remember that the Smart Party application is a servant to a real human party, not its master. Ubicomp applications should be designed to serve people, not to force them to behave as the application designers envisioned.

The Smart Party could be enhanced in many ways. As one obvious example, unsatisfied users have at least two options to increase satisfaction, based on these results: join a group to increase your voting power, or move to another room where things might be better. Which should users do, or, as suggested above, which are users actually likely to do in a real party? To what extent should the application expect to get more explicit feedback and guidance from its users, and how can such input be easily gathered and effectively used? Is a party whose music is jointly selected by its attendees actually a better experience for humans than a party where a host or a DJ selects all the music for them? These questions, too, can be generalized to apply to a broad range of social-based ubiquitous computing applications.

References

[1] K. Eustice, V. Ramakrishna, N. Nguyen, and P. Reiher, "The Smart Party: A Personalized Location-aware Multimedia Experience," in *Proc. Consumer Communications and Networking Conference (CCNC)*,Las Vegas, 2008.

[2] K. Eustice, *"Panoply: Active Middleware for Managing Ubiquitous Computing Interactions,"* Ph.D. dissertation, Dept. Comp. Sci., UCLA, Los Angeles, CA, 2008.

[3] C. Gini, "Variabilitá e mutabilita" 1912 reprinted in *Memorie di metodologica statistica* (Ed. Pizetti E, Salvemini, T). Rome: Libreria Eredi Virgilio Veschi 1955.

[4] J. McCarthy and T. Anagost, "MusicFX: An Arbiter of Group Preferences for Computer Supported Collaborative Workouts," in Proc. of the ACM 1998 Conference on Computer Supported Cooperative Work, 1998.

[5] A. Crossen, J. Budzik, and K. Hammond, "Flytrap: Intelligent Group
[6] Music Recommendation," in *Proc. of the 7th Intl. Conf. on Intelligent user interfaces.*, San Francisco, 2002.

[7] M. Prasad and J. McCarthy, "A Multi-Agent System for Meting Out Influence in an Intelligent Environment," in *Proc. of the 11th Innovative Applications of Artificial Intelligence Conf.,* 1999.

[8] C. Baccigalup and E. Plaza, "Poolcasting: A Social Web Radio Architecture for Group Customization." in *Proc. of 3rd Intl. Conf. on Automated Production of Cross Media Content for Multi-channel Distribution*, 2007.

[9] E. Nikolova, H. Tamari, A. Saha, and A. Schrader, "Pervasive Campus: Smart Party," in *Proc. of the 2nd Intl. Conf. of Digital Live Art,* 2007.