Modelling Task Knowledge Structures in Demos 2000

Jean Paul Degabriele, David Pym

HP Laboratories
HPL-2008-94

Keyword(s):
Task Knowledge Structures, Demos 2000, Modelling

Abstract:

Task Knowledge Structures provide an account of the knowledge structures that people possess and use when performing a task. Such models can be constructed using various techniques, such as direct observation, interviews, questionnaires, and others. TKS models can represent either knowledge structures that are possessed by a specific individual or, alternatively, a number of such individual TKS models can be amalgamated to form new TKSs. Demos 2000 is tool for specifying and executing mathematical simulation models of systems. The structure of Demos 2000 models can be considered to be based on the resources and processes that describe the system components and a stochastic representation of the (dynamic) environment within which the system resides. This report summarizes our work in translating models of human behavioural characteristics, as represented by Task Knowledge Structures, into Demos 2000 models.

External Posting Date: August 6, 2008 [Fulltext] Approved for External Publication
Internal Posting Date: August 6, 2008 [Fulltext]

Modelling Task Knowledge Structures in Demos 2000

Jean Paul Degabriele David Pym
jeanpaul.degabriele@gmail.com david.pym@hp.com

Systems Security Lab
Hewlett-Packard Laboratories,
Bristol BS34 8QZ, UK

Abstract. Task Knowledge Structures provide an account of the knowledge structures that people possess and use when performing a task. Such models can be constructed using various techniques, such as direct observation, interviews, questionnaires, and others. TKS models can represent either knowledge structures that are possessed by a specific individual or, alternatively, a number of such individual TKS models can be amalgamated to form new TKSs. Demos 2000 is tool for specifying and executing mathematical simulation models of systems. The structure of Demos 2000 models can be considered to be based on the resources and processes that describe the system components and a stochastic representation of the (dynamic) environment within which the system resides. This report summarizes our work in translating models of human behavioural characteristics, as represented by Task Knowledge Structures, into Demos 2000 models.

1. Introduction

The work presented in this report is part of a bigger project that is concerned with the study of human behaviour in information security. The project brings together various research areas such as Task Knowledge Structures (TKS) [4, 5], human-computer interaction in security, mathematical modelling, and discrete event computer modelling using Demos 2000 (henceforth Demos2k) [1, 3]. Demos 2000 models can be considered to be based on the resources and processes that describe the system components and a stochastic representation of the (dynamic) environment within which the system resides. Demos2k models execute as discrete event simulations. This project aims to build models that incorporate characteristics of human behaviour in order to analyse security policies and their associated risk mitigation and economic effectiveness.

The overall methodology of the bigger project, illustrated in Figure 1, provides a context for this work. We are concerned with developing, from the perspective of information security, a methodology for modelling whole systems: the users and the business processes, the technological system and its processes, including the threats incident upon both of these, and the economic environment within which the system and its users operate.

Figure 1 indicates our intended methodology. We begin with an empirical study of a system, capture the information from that study in a conceptual model (e.g., using TKSs), and then construct a corresponding, executable mathematical model (e.g. using Demos2k). Iterations in design and analysis are then performed, leading to a basis for an economic analysis of the value of aspects of the system.

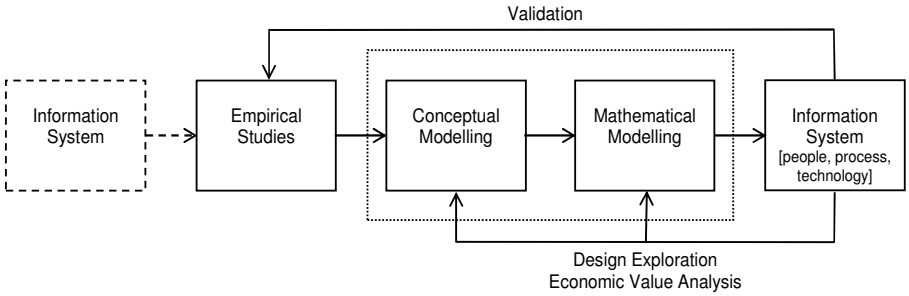

Figure 1: The overall modelling methodology

This report summarizes out work in translating models of human behavioural characteristics, as represented by Task Knowledge Structures, into Demos2k, the transition in the central box of Figure 1.

Task Analysis is a branch of psychology that examines what people do when carrying out a task and how they do it. Various contributions in this area led to the belief that people possess structured knowledge about the tasks they perform which is stored in long-term memory and processed in working memory [2, 6, 7]. In [5], it is argued that TKS models can be built which represent the knowledge structures that people possess and use when performing a task. Such models can be constructed using various techniques, such as direct observation, interviews, questionnaires, and others. TKS models can represent either knowledge structures that are possessed by a specific individual, or alternatively a number of such individual TKS models can be amalgamated into a single TKS through a process called *Generification*. Such a TKS would represent characteristics that are common among individuals when carrying out a particular task. Thus by modelling such a TKS in Demos2k we can simulate the average individual performing a particular task and thereby allow us to make further predictions.

Our modelling philosophy is based on classical applied mathematics, using tools from algebra, logic, computation theory, queuing theory, and probability theory [1, 3, 8, 9, 10]. Basically, for the purposes of this paper, we consider the following essential components when modelling a system: *environment*, *resources*, and *processes*. The complex dynamics pertaining to the environment of a system are represented stochastically. The notion of locations refers to the spatial, temporal, or the more abstract spaces inherent in a model. Lastly the aim of a system is to carry out processes which in turn require some kind of resources. Demos 2000 is a semantically justified modelling language which, to some extent, captures each of these conceptual structures [3].

In the sections that follow, we describe our experience in constructing Demos2k models from Task Knowledge Structures. We made substantive use of the Task Knowledge Structures and accompanying information listed in [4]. Johnson and Hyde provide a summary of the experiments they carried out on individuals constructing jigsaws collaboratively, and the corresponding TKS models they were able to

assemble from these experiments. We describe how we built on this work to translate these TKSs to Demos2k models, and compare results from our simulations with experimental values listed in [4].

2. A Basis for Mapping TKSs to Demos2K

After having completed this exercise, we feel that we are not yet in a state in which we can formulate a *formal* methodology for mapping Task Knowledge Structures onto Demos2k models. A mapping from TKS to Demos2k depends to some extent on one's programming style and the intent for which the model is built. Moreover, we do not know yet whether it would be possible to formulate such a mapping methodology without any loss of generality. Nonetheless, we give below a set of intuitive guidelines on which we based our mapping.

1. Task Knowledge Structures represent the knowledge a human stores in long-term memory on how to accomplish a task. However a TKS may not provide all the sufficient information to model a human doing the task under consideration. In the jigsaw example, the TKS does not give details about the time required to search a tile from a pile, or not even how the player searches for a jigsaw tile (i.e. whether he looks for a specific tile to start from or he picks the first two adjacent tiles he finds). Thus some reasonable assumptions or observations need to be taken to obtain this missing information: in our case, we attempted the task ourselves and observed our actions.
2. The purpose of a model is to produce predictive knowledge. So it is necessary to identify which components influence the aspects we are interested in. In the jigsaw case we analyse the model solely by the time it takes to complete the task. Thus abstract TKS sub-tasks such as *Create Workspace*, which have no direct influence on the time needed to complete the puzzle, were deliberately left out.
3. Often a TKS model represents the task-knowledge structure of a single individual. When constructing a predictive model, the features of interest are those that are common to the majority of the individuals, as these are essentially the features we are able to predict. Thus in such circumstances it is best to use a *generified* TKS [5], and account for individual peculiarities using a noise parameter in the model, or by the degree of uncertainty inherit in the model.
4. A TKS is composed of a goal, sub-goals, sub-tasks, a taskplan, objects and associated actions. For the sake of integrity it is important not to infringe any of the structure inherent in the TKS. The sequencing and chunking of sub-tasks portrayed in the TKS taskplan should be preserved in the Demos2k model. Additionally, events that occur in random sequence should be replicated as occurring in a random sequence.
5. A computer model is generally more useful the more flexible it is, in the sense that it can represent more scenarios that are of interest. So it is generally good practice to choose an abstraction level that allows the model to portray various strategies. For instance, in the jigsaw case, one strategy is to dissect the jigsaw into the main pictorial sections depicted. An alternative strategy is to dissect the jigsaw into the four borders and the centre. These strategies can be modelled separately by assigning different values for the number of *sections*,

the *tile-search-time*, and the *probability* with which the player consults the lid. The latter two parameters need to be adjusted since border pieces are easier to spot.

6. Again for the sake of generality and flexibility, it is best to include factors that represent the individual's skill in conducting the task as model input parameters. Referring again to our model such parameters include the mean time needed to pick a tile from a pile, the error rate, and how often the individual needs to consult the box lid.
7. As in every kind of programming, it helps to keep some degree of structure, and in this case it is better to make this structure conform to the structure of the TKS. Most important is to separate clearly between sub-goals. This does not, however, necessarily require each chunk of sub-tasks, corresponding to a sub-goal, to be implemented as distinct entities.
8. In collaborating models, identify the kind of collaboration that is taking place; that is, whether it is a form of parallelization, pipelining, or any other. Moreover, identify any competing behaviour between the collaborating entities such as contention for resources.
9. The TKS of a task carried out individually is generally different from a TKS of the same task when carried out collaboratively. It is therefore important to use the appropriate TKS instead of adapting it for the scenario at hand.

3. Model of a Collaborative Jigsaw Construction Task

Our first attempt was to model the collaborative TKS portrayed in Figure 2. The full Demos2k model can be found in the Appendix. The model employs three classes: *shakeBox*, *player*, and *calculatePileSize*. The purpose of shakeBox is for initialization: it populates the box with randomly oriented jigsaw tiles and it initializes the *piles* bin with numbered tokens. The number of tokens in the bin is determined by the model parameter *pilesAmt* which reflects the number of piles into which the players will categorise the jigsaw pieces. Finally, shakeBox signals its completion by placing a token in the bin *startPlay* upon which two instances of the class player are created.

The *player* class is at the heart of the model, as this is where the Task Knowledge Structure is embodied. The class is segmented into two stages which correspond to the two sub-goals in the TKS. Stage 1 starts with the player scrutinising the box lid, represented by the resource *Lid*, for a time span sampled from an exponential distribution. Thus while one player is scrutinizing the box lid, the other will be on hold waiting for his turn to examine the box lid. Once a player had a first look at the box lid he starts sorting the jigsaw pieces into piles. Each player picks a tile at a time from the box, flips it if necessary, and then does one of two things, either he places it directly into the workspace or sorts it into one of the piles. This is determined by the model parameter *placeInWorkSpace*, which determines the probability of either case to occur. Moreover, after picking a tile the player may occasionally consult the box lid to determine where it belongs.

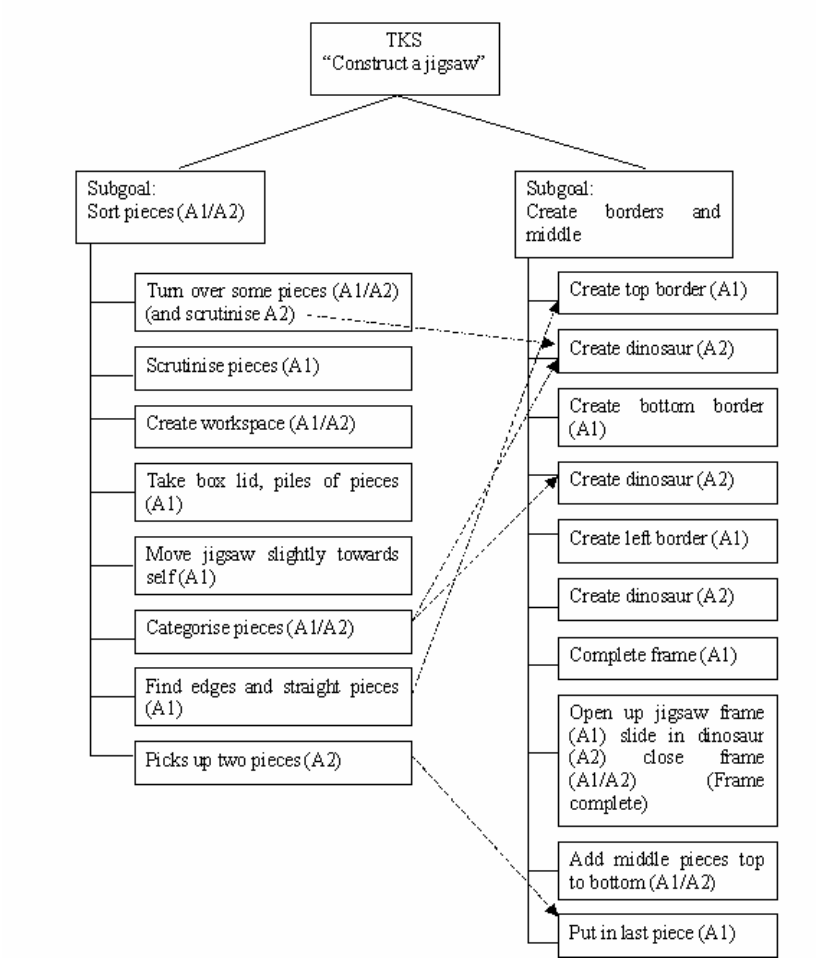

Figure 2: TKS model of a collaborative jigsaw construction (reproduced from [4]).

When both players have finished sorting the tiles, the players simultaneously move on to the second stage. Each of them gets hold of a pile by grabbing a token from the piles bin. This is the point where the third class, *calculatePileSize*, is invoked by the players. The role of this class is similar to that of a function which calculates the number of tiles in a pile. Jigsaw construction is modelled by having the players look for a particular tile among their pile and placing it in the workspace until the jigsaw section corresponding to their pile is complete. Intuitively it makes sense that the time taken to find a tile within a pile is in some way related to the size of the pile. We model this time interval by sampling an exponential distribution to obtain a random variable, *searchTimePerTile*, and multiply this value by the pile-size. Thus on average a player will pick tiles faster as he works through a pile. Moreover while a player is working through a jigsaw section there is the possibility that he picks the wrong tile. Such errors are more likely to occur when searching through a large pile and the probability of errors decreases as the pile gets smaller. For each tile that is picked the probability of it being the wrong one is given by

$$P\{\text{error}\} = (S - 1) / (\varepsilon S_0),$$

in which S is the pile-size before the tile was picked, S_0 is the initial pile-size, and **ε** is a scaling parameter. The pile-size is decremented by 1 such that when the pile-size is 1 no error can occur. If the tile picked is the correct one it is placed in its position in the workspace, otherwise it is put back in the pile and the player may want (according to a predefined probability) to consult the box lid. This process goes on until the jigsaw is completed.

While the structure of the model represents the strategy that is inherent in the TKS, the model parameters need to be tuned to reflect the users' skill and the complexity of the jigsaw. Such parameters are the *searchTimePerTile*, *requireLid4Sorting*, *invErrorProb*, *placeInWorkSpace*, etc. The number of piles however depends both on the strategy employed and the picture portrayed in the jigsaw. One more thing to note is that the total number of errors does not depend on the number of piles into which the jigsaw tiles were divided. The average total number of errors only depends on **ε,** which we call *invErrorProb* in the model.

4. Comparing Two Strategies for Constructing Jigsaws

The paper by Johnson and Hyde [4] includes a summary of two participants independently constructing a jigsaw puzzle. There are some important differences in the way the two participants, A1 and A2, construct the dinosaur jigsaw. Each of the participant's strategy is summarized in the TKS models shown in Figures 3 and 4. In brief, A1 has a more structured strategy and follows it very strictly, while A2 takes a less structured and more opportunistic approach in completing the jigsaw. A1 first flips all the tiles, then sorts them, then first completes the frame, attaches pieces to the frame, and then completes the jigsaw from top to bottom. On the other hand, A2 flips a tile and either he directly places it in the jigsaw or he sorts it in one of the piles. Similarly to A1, most of the jigsaw is completed from top to bottom. Another main difference between the two is the degree of categorization that they employ in sorting the pieces. While A1 sorted tiles only into two piles, A2 was at different stages grouping tiles into 4, 5, and 6 piles[*].

[*] Private communication with Hilary Johnson

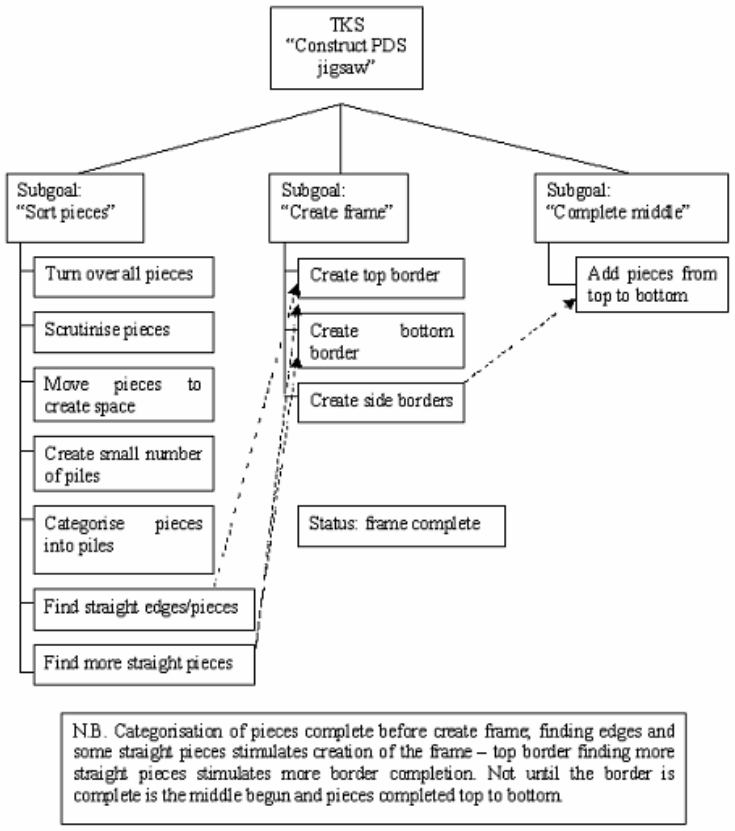

Figure 3: A TKS model for A1 (reproduced from [4]).

In order to verify in part the validity of our modelling methodology, we decided to replicate these two strategies in Demos2k and compare the results with the actual experimental values. The full Demos2k models for A1 and A2 can be found in the Appendix. When examining the two models, it is evident that they are fairly similar. The main differences lay in stages 1 and 2 of the player class. In stage 1 A1 flips all the jigsaw tiles that are facing down and then examines the box lid before moving to stage 2 where he sorts all the pieces into 2 piles. In contrast, Player A2, in stage 1, merely examines the box lid, while in stage 2 tiles are picked, flipped if necessary and either placed directly into the workspace (with a probability of 10%) or sorted into one of the piles (with 90% probability). Construction of the jigsaw occurs in Stage 3, which is done in a similar way to the collaborating model. Participant A2 is configured to have 5 piles throughout the simulation run. As the effective time to search a tile in a pile is assumed proportionate to the pile size in the model, this is another distinguishing factor in the model that makes A2's strategy faster. It is reasonable to assume that categorizing tiles among five piles rather than two is a more complex process. Accordingly the time required for sorting a tile was modelled by two exponential distributions of mean values 1 and 2 seconds for A1 and A2 respectively. In [4], however, it is claimed that A1 took longer than A2 to sort the tiles, but since no values are specified we decided to stick with these values. One more parameter in the models that reflects the players' skill is the Inverse Error

Probability. This was tuned to reflect the experimental values listed in [4]. As can be seen from the tables below, the numbers of failed attempts in the two sample runs shown in Table 2 conform with the actual experimental values shown in Table 1.

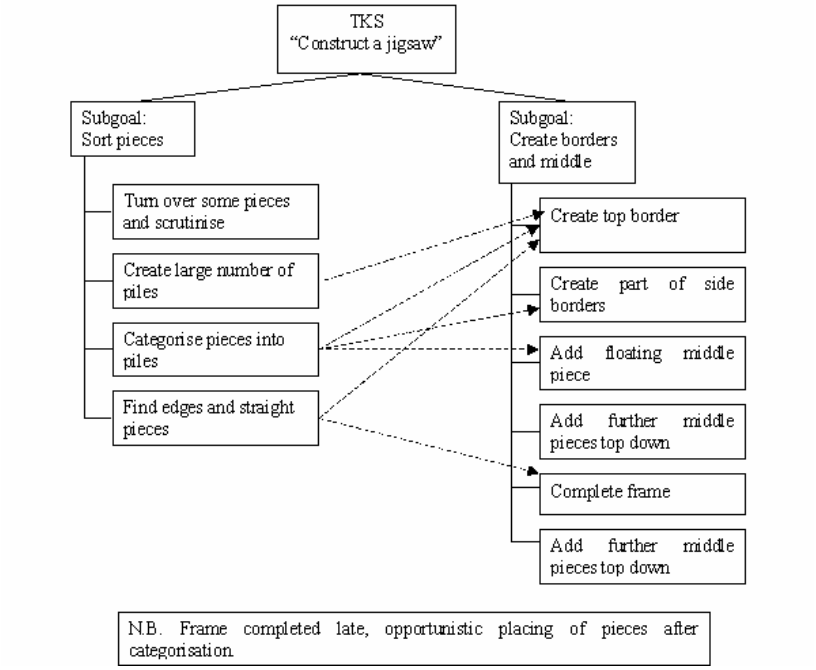

Figure 4: A TKS model for A2 (reproduced from [4]).

Participant	Total Time taken (s)	No. of pieces	Failed Attempts
A1	1320	120	44
A2	940	120	29

Table 1. Experimental results obtained from [4].

Participant	Total Time taken (s)	No. of pieces	Failed Attempts
A1	1316	120	45
A2	941	120	30

Table 2. Single Sample runs of our Demos2k models.

5. Results, Conclusions, and Future Work

The two models described above were run 1000 times each producing the plots shown in Figures 5 and 6. We think that the experimental values in Table 1 are sufficiently close to the means of the two distributions, to sustain the validity of our models, and proceed with our line of work. Our plans for future work are to use such models to analyze the impact of security policies on humans and their everyday tasks. In

particular, we intend to examine the impact of security policies on employees' productivity, and use similar models to assess the effectiveness of alternative security policies. Complex security policies may render users more prone to commit other security-critical errors which expose the system to further risks.

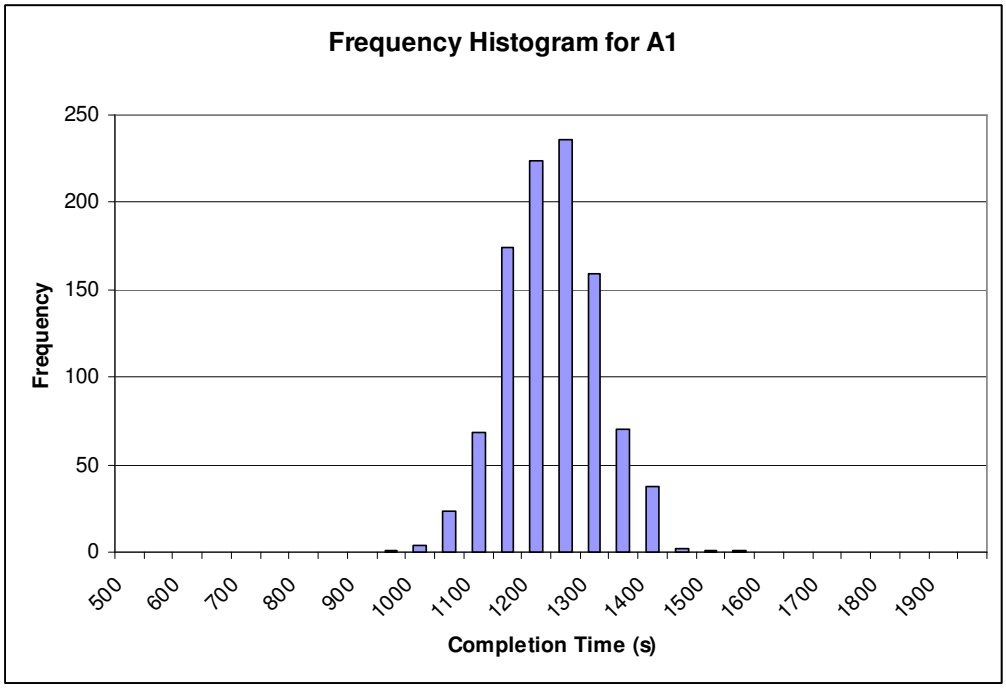

Figure 5: Frequency distribution of a 1000 sample runs of model A1.

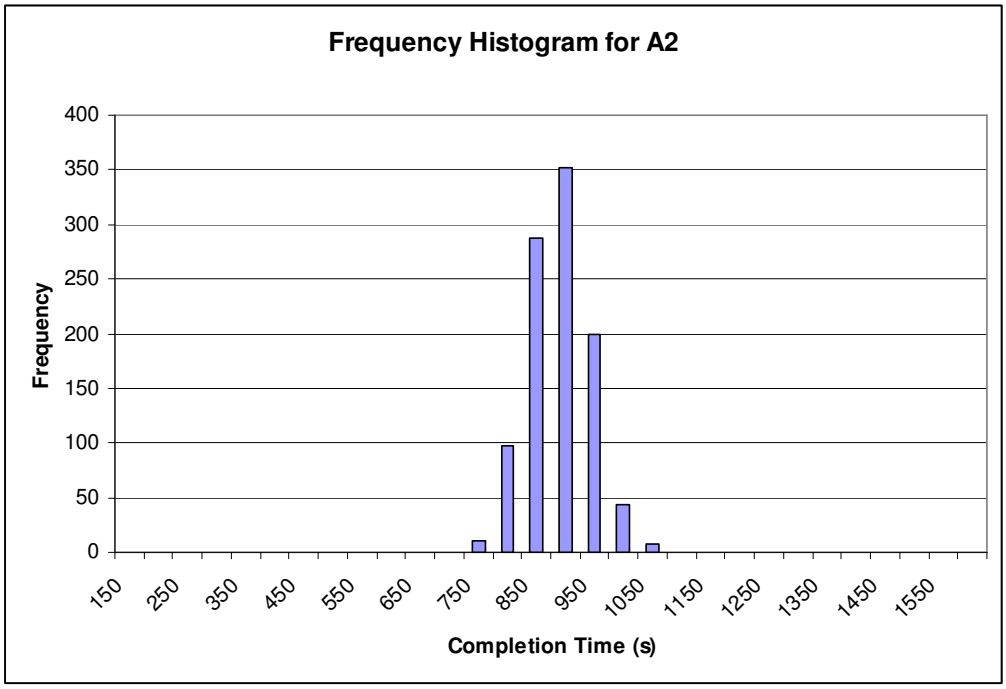

Figure 6: Frequency distribution of a 1000 sample runs of model A2.

Acknowledgements

We are grateful to Brian Monahan, Hilary Johnson, and Mike Yearworth for discussions about this work.

References

[1] Birtwistle, G. *Demos - Discrete event modelling on Simula*. Macmillan, 1979.

[2] Card, S.K., T.P. Moran, and A. Newell (1983). *The psychology of human-computer interaction*. Hillsdale, N.J.: Lawrence Erlbaum Associates.

[3] Christodolou, A., R. Taylor, and C. Tofts (2000). Demos 2000. http://www.demos2k.org

[4] Johnson, H. and J. K. Hyde (2003). Towards Modelling Individual and Collaborative Construction of Jigsaws Using Task Knowledge Structures (TKS). ACM Transactions on Computer-Human Interaction 10(4): 339-387.

[5] Johnson, P., H. Johnson, R. Waddington, and A. Shouls (1988). Task-related knowledge structures: Analysis, modelling and application. In D.M. Jones and R.Winder, Eds. *People and Computers: From research to implementation*. Cambridge: Cambridge University Press, 35-62.

[6] Kieras, D and P.G. Polson (1985). An approach to the formal analysis of user complexity. *International Journal of Man-Machine Studies*, 22, 365-394.

[7] Payne, S. J. and T.R.G. Green (1986). Task-Action Grammars: a model of the mental representation of task languages. *Human Computer Interaction,* 2, 93-133.

[8] Pym, D. and C. Tofts (2006). A calculus and logic of resources and processes. *Formal Aspects of Computing* 18(4): 495-517. Erratum: Collinson, M., D. Pym, and C. Tofts (2007). *Formal Aspects of Computing* 19: 551-554.

[9] Pym, D. and C. Tofts (2007). Systems Modelling via Resources and Processes: Philosophy, Calculus, Semantics, and Logic. *Electronic Notes in Theoretical Computer Science* 172, 545-587. Available from: http://www.sciencedirect.com/science/journal/15710661 Erratum: Collinson, M., D. Pym, and C. Tofts (2007). *Formal Aspects of Computing* 19: 551-554.

[10] Yearworth, M., B. Monahan, and D. Pym (2006) Predictive Modelling for Security Operations Economics. *Proc. I3P Workshop on the Economics of Securing the Information Infrastructure*. Washington DC, 23-24 October, 2006. Available from: http://www.hpl.hp.com/techreports/2006/HPL-2007-125.pdf

Appendix

```
// Demos2k for TKS & Jigsaws

// Parameter initialization

// Scaling constants (hrs = timing unit)

   cons hrs    = 3600;
   cons mins   = 60;
   cons secs   = 1;
   cons msecs  = secs/1000;

   cons days   = 24 * hrs;
   cons weeks  = 7 * days;
   cons months = 28 * days;
   cons years  = 365 * days;

// Model parameters

   cons jigsawSize = 120;
   cons pilesAmt = 4;
   cons players = 2;
   cons invErrorProb = 3; // between 2 and 4 sounds reasonable

// Stochastic parameters

   cons probUp = pud[(0.5,0),(0.5,1)]; // 0 = Facing Down,
                                       // 1 = Facing Up
   cons flipTime = negexp(1 * secs);
   cons scrutinyTime = negexp(5 * secs);
   cons placeInWorkSpace = pud[(0.9,0),(0.1,1)];
   cons requireLid4Sorting = pud[(0.8,0),(0.2,1)];
   cons requireLid4Searching = pud[(0.7,0),(0.3,1)];
   cons examineLid4Sorting = negexp(3 * secs);
   cons examineLid4Searching = negexp(3 * secs);
   cons pickTile = pud[(0.23, 1),(0.32, 2), (0.2, 3), (0.25, 4)];
   cons rand = uniform(0, 1.0);
   cons searchTimePerTile = negexp(500 * msecs);
   cons sortTime = negexp(1500 * msecs);
   cons placeTime = negexp(2 * secs);

// Universal variables
```

```
   var t = 0;
   var workSpace =0;
   var errors = 0;

// Resources

   res(Lid,1);
   res(lockWS, 1);
   res(lockE, 1);

// Bins

   bin(box, 0);
   bin(piles, 0);
   bin(pile, 0);
   bin(readySorting, 0);
   bin(startPlay, 0);

// Class definitions

class shakeBox = {

   local var i = 0;
   local var up = 0;

// populate box

   while [i < jigsawSize]
   {
      up := probUp;
      putVB(box, [up]);
      i := i + 1;
   }

// set number of piles

   i := 1; do pilesAmt {putVB(piles, [i]); i := i + 1;}

   putB(startPlay, 1);
}

class player(id) = {

   local var up = 0;
   local var pileGroup = 0;
   local var pileSize = 0;
   local var pileVar = 0;
   local var initSize = 0;
   local var pid = 0;
   local var pidVar = 0;
   local var flag = 0;

// Stage 1: Sort or place tiles

   getR(Lid, 1); hold(scrutinyTime); putR(Lid, 1);

   while [getVB(box, [up], true)]
   {
      try [up == 1] then {}
      etry [] then {hold(flipTime);}
```

```
      try    [requireLid4Sorting    ==    1]    then    {getR(Lid,1);
hold(examineLid4Sorting); putR(Lid,1);}
      etry [] then {} // Examine Lid for Sorting or Placing

        try [placeInWorkSpace == 1] then
      {
         hold(placeTime);
         getR(lockWS, 1); workSpace := workSpace + 1;
         putR(lockWS, 1);
         trace("workSpace=%v", workSpace);
      }
      etry [] then
      {
         hold(sortTime);
         pileGroup := pickTile;
         putVB(pile, [pileGroup, 0]);
      }
      trace("Player %v (%n) finished Stage 2 (sorting)", id);
   }

// Wait for everyone to finish, so that all piles are complete

   do (players - 1) {putVB(readySorting, [id]);} // announce I am
                                                 // ready to everyone

   pid := 1;
   while [pid < (players + 1)] // check everyone is ready
   {
      try [pid == id] then {}
      etry [getVB(readySorting, [pidVar], pidVar == pid)] then {}

      pid := pid + 1;
   }

// Stage 2: Complete the rest of the jigsaw

   pileGroup := 1;

   while [pileGroup < pilesAmt + 1]
   {
      try [getVB(piles, [pileVar], pileVar == pileGroup)] then
      // take a pile
      {
         syncV(calcPileSize,  [pileGroup],  [pileSize]);  initSize  :=
pileSize;
         hold(1 * msecs); // initialise pileSize

         while [getVB(pile, [pileVar, flag], pileVar == pileGroup)]
         {
            hold(searchTimePerTile * pileSize); // model time to
                                                // search a tile

            try [rand > ((pileSize - 1)/(invErrorProb * initSize))]
then //Prob Correct tile was picked

            {
               hold(placeTime);
               getR(lockWS,   1);   workSpace   :=   workSpace   +   1;
putR(lockWS, 1);
               pileSize := pileSize - 1;
```

```
               trace("workSpace=%v", workSpace);
            }
            etry [] then
            {
               getR(lockE, 1); errors := errors + 1; putR(lockE, 1);
               trace("errors=%v", errors);

               try [requireLid4Searching == 1] then
              {getR(Lid,1); hold(examineLid4Searching); putR(Lid,1);}
// consult Lid
               etry [] then {}

               putVB(pile, [pileGroup, flag]); // put tile back
            }
         }
      }
      etry [] then {}
      pileGroup := pileGroup + 1;
   }

   trace("Player %v (%n) is ready", id);
}

class calculatePileSize = {

   local var i = 0;
   local var index = 0;
   local var pileIndex = 0;
   local var flag = 0;

   repeat {
      getSV(calcPileSize,[pileIndex], true);

      i := 0;
      while [getVB(pile, [index, flag], (index == pileIndex)&& (flag
      == 0))]
      {
         i := i + 1;
         putVB(pile, [pileIndex, 1]);
      }

      putSV(calcPileSize, [i]);
   }
}

// Run simulation

   entity(SHAKEBOX, shakeBox, 0);
   entity(CALCULATEPILESIZE, calculatePileSize, 0);

   try [getB(startPlay, 1)] then
      {t := 1; do players {entity(PLAYER, player(#t), 0);
       t := t + 1;}}

   try [workSpace == jigsawSize] then {close;}
```