



SmartSociety

Hybrid and Diversity-Aware Collective Adaptive Systems
When People Meet Machines to Build a Smarter Society

Grant Agreement No. 600584

Deliverable D6.1 Working Package WP6

Static social orchestration: methods and specification

Dissemination Level (Confidentiality):¹	<i>PU</i>
Delivery Date in Annex I:	<i>31/12/2013</i>
Actual Delivery Date	<i>31/12/2013</i>
Status²	<i>F</i>
Total Number of pages:	40
Keywords:	<i>compositionality, social orchestration, abstract architecture</i>

¹PU: Public; RE: Restricted to Group; PP: Restricted to Programme; CO: Consortium Confidential as specified in the Grant Agreement

²F: Final; D: Draft; RD: Revised Draft

Disclaimer

This document contains material, which is the copyright of *SmartSociety* Consortium parties, and no copying or distributing, in any form or by any means, is allowed without the prior written agreement of the owner of the property rights. The commercial use of any information contained in this document may require a license from the proprietor of that information. Neither the *SmartSociety* Consortium as a whole, nor a certain party of the *SmartSociety*s Consortium warrant that the information contained in this document is suitable for use, nor that the use of the information is free from risk, and accepts no liability for loss or damage suffered by any person using this information. This document reflects only the authors' view. The European Community is not liable for any use that may be made of the information contained herein.

Full project title:	SmartSociety: Hybrid and Diversity-Aware Collective Adaptive Systems: When People Meet Machines to Build a Smarter Society
Project Acronym:	SmartSociety
Grant Agreement Number:	600854
Number and title of work-package:	WP6 Compositionality and Social Orchestration
Document title: methods and specification	Static social orchestration:
Work-package leader:	Michael Rovatsos, UEDIN
Deliverable owner:	Michael Rovatsos, UEDIN
Quality Assessor:	George Kampis, DFKI

List of Contributors

Partner Acronym	Contributor
UEDIN	Michael Rovatsos
UEDIN	Dimitrios I. Diochnos
TUW	Ognjen Scekic
TUW	Hong-Linh Truong
UOXF	John Pybus
UOXF	Kevin Page

Executive Summary

This document summarises the work performed in WP 6 of the SmartSociety during the first year of the project toward achieving a first specification of a static social orchestration architecture based on a survey of existing methods and their adaptation to the challenges of HDA-CAS.

We start by formulating the overall specific objectives and scientific vision of the workpackage, which are driven by the broader aim to understand how complex social computations are composed of many individual contributions of human users and machines. Based on the observation that traditional notions of compositionality break down when we move from “closed” systems with known, static computational components and a high degree of *a priori* interoperability, we propose a view of system composition that emphasises the use of *context* (hidden, but relevant, information revealed to the system through its human participants and machine analysis of data) and *collectives* (treating the behaviour of aggregates of contributing processes distinctly from individuals) to recover some of the compositionality in “open”, evolving HDA-CAS.

Secondly, we define an abstract model of social computation which captures the essential aspects of the kinds of computations we are interested in. These are: distributed data processing and exchange among distinct (human or machine) nodes of computation across a network structure; sequential, parallel, and hierarchical composition under minimal assumptions regarding synchronisation, communication facilities, and organisational structure; definition of the result of a social computation without reference to any specific implementation environment; linking a model of data-driven distributed computation to models of distributed, motivation-driven rational reasoning and decision making. Using this abstract architecture, we identify and formally define a set of core research problems that set the long-term research priorities of the workpackage, related to the automated synthesis, verification, and optimisation of social computations.

Thirdly, we propose a first social orchestration architecture which identifies a set of specific functional components (discovery, assignment, execution, feedback) within the broader abstract model and the way these can be put together to provide a general method for composing socially orchestrated collaborative tasks. This is still generic enough to capture a broad range of existing social computation systems, but constitutes a more concrete proposal for a specific “style” of orchestrating them. Instead of attempting to implement this kind of architecture within existing “closed” platforms, we propose a new, purely data-driven, lightweight computational architecture, which we call the “play-by-data” architecture that is better suited for mapping our conceptual framework to concrete implementations. Play-by-data emphasises RESTful web-based interaction, data-orientation, openness, and opportunistic, voluntary processing without explicit guarantees. It also proposes minimal standards for interoperability, though these have so far only been elaborated at a conceptual level, and will need to be further defined in future iterations. To illustrate how these principles can be applied in a real-world scenario, we present an exemplary implementation of these architectural principles in a ridesharing domain.

Finally, we critically review the work done so far, review the related literature, and discuss avenues for future work.

Table of Contents

1	Vision and objectives	7
2	Abstract social computation model	8
2.1	Definition	8
2.2	Core research questions	10
2.3	The decision-making perspective	11
3	Static social orchestration model	12
3.1	Definition	13
3.2	Implications for design	14
4	The Play-By-Data architecture	15
4.1	Introduction	15
4.2	Architecture	17
4.3	Benefits	19
5	An example	20
5.1	The ridesharing domain	20
5.2	Implementation	21
5.2.1	Overall architecture	21
5.2.2	Discovery	23
5.2.3	Assignment	23
5.2.4	Execution	24
5.2.5	Feedback	25
5.2.6	Social computation and compositionality	25
6	Discussion	26
6.1	Work so far	26
6.2	Next steps	27
7	Related work	28
7.1	Agent-based systems	29
7.2	Workflow-based systems	30
7.3	Human-based computation systems	31
7.3.1	Process-centric Collaboration	32
7.3.2	Ad-hoc Collaboration	32
7.3.3	Crowdsourcing Systems	33
7.4	Summary	37
8	Conclusion	37

1 Vision and objectives

The SmartSociety project is concerned with understanding the design principles, operating principles, and adaptation principles of hybrid and diversity-aware collective adaptive systems (HDA-CAS). Invariably, such systems involve the composition of a multitude of heterogeneous users, hardware and software components in complex socio-technical systems. Within the overall project, the aim of WP6, *Compositionality and Social Orchestration*, is to improve our understanding of how these kinds of systems are composed, and to develop novel methods that allow for effectively orchestrating the social computations (SCs) they perform.

Compositionality enables us to derive the meaning of composite structures based on the meanings of their constituents. It is an important property of many formal systems and models, in particular the traditional semantics of mathematics or formal logic (e.g. the meaning of $x + y$ can be solely defined based on the denotation of x , y , and the semantics of the operation “+”). Compositionality is key to successful modelling and system design, as it essentially allows us to anticipate the interactions between components ahead of time, and thus to build systems whose behaviour is well-understood. *Complex systems* often “break” this property, as the whole can be “less” or “more” than the sum of the parts. This may be due to side-effects that were not taken into account during composition (e.g. resource contention when multiple actors use resources at the same time, unaware that they are sharing the resource), or due to unanticipated regularities and redundancies (e.g. one is expecting to collect all different items of information from different users, but they give the same response, unaware what others are saying). From a modelling point of view, the problem arises from *contextuality*, i.e. the fact that there are factors affecting the overall process that lie outside the modelling boundary, and where not adequately taken into account at design time. This shifts systems from a traditional *closed-world* to an *open-world* nature, where what lies beyond the modelling boundary does affect the system, and leads to models not correctly reflecting reality, with negative consequences for the behaviour of systems based on these models. Achieving some “closure” of these open systems involves introducing notions of *collectives*. These make the interactions between components explicit, thereby allowing for relevant contextual information to be reintroduced in the model and capture aspects whose absence led to the original loss of compositionality features.

Our work in WP6 is guided by this overall model of iterations of compositionality loss and recovery, and by a design philosophy that could roughly be summarised by the tagline “*compositionality = context + collectives*”, which we call, somewhat informally, “the CCC principle”. The engineering methods we want to develop to utilise these principles are both human- and machine-based, and exploit the complementary capabilities of both types of actors: Humans are good at explicating context that wasn’t available to a system before, at selecting relevant information out of a plethora of data and possible design alternatives, and at recovering from unexpected failures in some way. Machines are good at processing high volumes of information, monitoring the operation of large numbers of interacting components, filtering and analysing data, and performing algorithmic tasks more generally at very high speed and with very high accuracy. Our overall objective is

to provide the right kind of models, algorithms, and architectures that will allow us to combine these human and machine capabilities in large-scale distributed systems so as to make these systems more scalable, resilient, and efficient.

This deliverable describes first steps toward this aim, in particular the specification of an initial social orchestration architecture. We start by defining an abstract model of *social computations* which allows us to talk about sequential, parallel, and hierarchical composition of human and machine tasks without making any commitment to a specific computational infrastructure, and to define a set of core research problems that set the scene for our future research. This is subsequently used to define a *social orchestration* model in terms of more concrete functional building blocks involved in organising common social computations, and, briefly discuss how this can be mapped to models of autonomous, decentralised decision making that allow us to describe the stakeholders' rationales and thus lay the foundation for future *dynamic* adaptation of our currently *static* model of social orchestration. In the third part of this document, we propose a lightweight *computational architecture* that realises the principles of the more abstract models, and this is then illustrated with the example of an implemented prototype of a collaborative ridesharing platform. Finally, we look at related work, discuss relationships to other parts of the project, and present an outline of planned future work.

2 Abstract social computation model

2.1 Definition

We start by presenting an abstract model of SCs that allows us to describe the research problems they give rise to more precisely. Assume a set of human/machine agents $A = \{1, \dots, n\}$ and a set of local functions $F = \{f_1, \dots, f_m\}$ these agents can compute (where, typically, $m \ll n$), such that $F_i \subseteq F$ are the functions of agent i . The global set of variables in the system $X = \{x_1, \dots, x_k\}$ determines what the inputs and outputs of each function are, where every variable x_l has a domain D_l , and $X_i \subseteq X$ indicates which variables i has access to. Note that as many (often most) agents will be human users, many of the f_i will not have (known) formally precise, algorithmic representations.

Every local function $f \in F_i$, has input and output sets $X_{in}^f, X_{out}^f \subseteq X$, and we have $f : D_{in}^f \rightarrow D_{out}^f$ where $D_{in}^f = D_{i_1} \times \dots \times D_{i_s}$ and $D_{out}^f = D_{o_1} \times \dots \times D_{o_t}$ denote the domains of the input and output variables sets of f , i.e. $X_{in}^f = \{i_1, \dots, i_s\}$ and $X_{out}^f = \{o_1, \dots, o_t\}$. We will assume that agent i has access to the input variables of its local functions ($X_{in}^f \subseteq X_i$), and that it can/will only compute the outcome of a local function for a restricted subset of the possible inputs $D_i^f \subseteq D_{in}^f$.

To overlay this collection of local functions with a network structure, we introduce a neighbourhood function $N : A \rightarrow 2^A$ which maps every agent i to a set of agents $N(i)$ that i has access to, including itself (these are nodes that can be found via a search, are acquaintances, etc). A set of discrete timesteps $T = \{t_1, t_2, \dots\}$ is used to specify values at specific points in time, e.g. x_l^t denotes the value of variable x_l at timestep t , f^t denotes that function f is invoked at timestep t and so on (if a computation takes k timesteps, we

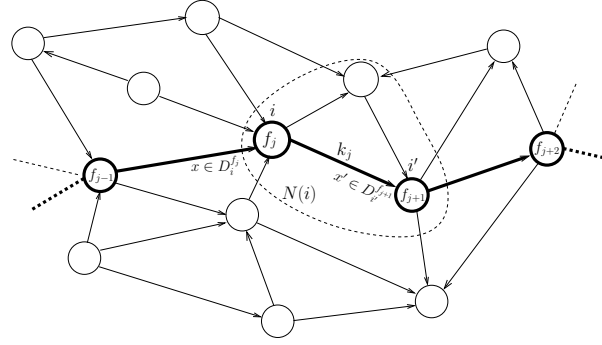


Figure 1: An abstract SC: Network edges depict neighbourhood relations N , with bold arrows for edges that are used to compute f . Agent i performs function $f_j \in F_i$ based on input x received from the previous node and, optionally, also variables locally known by i . After k_j timesteps the successor node f_{j+1} continues the computation.

have $f^t(x^t) = y^{t+k}$.

With this, we can define a *sequential SC function* $SC = (f, \{F_i\}_{i \in A}, N, I, t)$ to compute f using agents A on an input set $I \subseteq D_{i_1} \times \dots \times D_{i_m}$ where $\{i_1, \dots, i_m\} \subseteq \{1, \dots, |X|\}$ as a procedure that calculates f for each $x^t \in I$ given at time t after k timesteps such that

$$f(x^t) = f_n^{t+k_1+\dots+k_{n-1}} \circ \dots \circ f_2^{t+k_1} \circ f_1^t(x^t) = y^{t+k}$$

and output set O as the set of values y^{t+k} that result from this computation, where the following conditions hold for every $1 \leq j \leq n-1$, and $k = \sum_{j=1}^n k_j$:

1. There is some agent i with $f_j \in F_i$, and $f_{j+1} \in F_{N(i)}$.
2. For any two $f_j \in F_i$ and $f_{j+1} \in F_{i'}$, $x^{t+k_1+\dots+k_j} \in D_i^{f_j}$ and $x^{t+k_1+\dots+k_{j+1}} \in D_{i'}^{f_{j+1}}$ if $f_j^t(x^t) = x^{t+k_j}$.

The idea behind this is fairly simple: An SC calculates a target function f that is the result of a sequential application on inputs received from predecessor functions (or from the environment – we impose no constraints on the inputs other than that they be accessible to the agent operating on them) and passed on to the subsequent computation node. Condition 1. restricts the SC to sequences that can be constructed using only neighbours of the currently executing agent in each step, i.e. the overall computation is constrained by the network structure. Condition 2., on the other hand, constrains the computation of every local function to those inputs that the respective agent can (or is prepared to) process. Figure 1 illustrates the structure of these sequential computations graphically. It is fairly easy to extend this model to parallel computations as long as they are synchronised: For this, we need a collection $\{SC_1, \dots, SC_m\}$ of sequential SCs, and a set of constraints defined on *synchronisation variables* $X_{sync} \subseteq X$ of the form $(t, x = x')$ where op specifies that the values of variables x and x' must be equal at time t . These constraints are added as additional conditions to those specified for sequential SCs above, and allow us to link variables pertaining to separate computation sequences explicitly, rather than having to share global names for them.

Note that our model neither requires that every agent needs to be different or only involved in a single step (e.g. the sequence could involve polling different agents and then aggregating the result in a central node), that agents are heterogeneous in terms of what functions they can perform, or that all of the output variables they compute are needed by the subsequent node – some of those just effect local changes that are irrelevant for the overall computation.

This model is deliberately abstract and simplified: it does not account for asynchronous processing, non-determinism, or aggregation. Also, it does not imply any commitment to the algorithmic representations that will be used for the implementation of SCs. However, it captures the key elements of the kinds of systems we're interested in: a network structure that provides connectivity between local processes, constraints on the circumstances under which local computation will be performed, and fully decentralised information and control. Importantly, it captures the central composition operators that are normally provided by computational systems: sequential composition of computations over time (through chaining of functions and input/output sharing), parallel composition through access to common resources (variables that are synchronised through constraints), and hierarchical abstraction through nesting (each component function in the above model can be modelled as an SC itself).

2.2 Core research questions

At the abstract level, our definition above does not appear very different from a traditional distributed systems model. The challenging aspects of SC arise from the fact that humans play a significant role in these computations, significantly limiting *observability* and *predictability* of the system. We discuss several implications of this in the following exposition of a number of core research problems formulated using our model:

Synthesis *Given f , input set I , and a set of agents A with capabilities $\{F_i\}_{i \in A}$, what is a concrete sequence $f_n \circ \dots \circ f_1$ that computes $f(I)$?* The solvability of this problem depends on the way in which the functions are represented, i.e. this question cannot be answered at the level of our above model, which may include functions that are neither machine- nor human-computable. Certainly, for many functions computed by humans, there is little hope that we can describe those using rigorous formalisation.

Verification *Does a given sequence $f_n \circ \dots \circ f_1$ compute the target function f correctly on inputs I ?* While in principle much simpler than synthesis, in many real-world domains no agent will be able to verify whether others' local functions have been (correctly) executed, e.g. when they involve spatially dispersed physical action in the environment, or when they involve genuinely non-verifiable results (opinions, expert knowledge). An important question here is how human-based verification can be used to improve the "safety" of the SC system, for example through reputation systems.

Recruitment *Given a function f , input set I , and agents A , how can we identify a set of participants $P \subseteq A$ that will compute $f(I)$?* While this could be solved through exhaustive enumeration in a system with complete information, in human-centric systems we will normally not know under which conditions participants can/will perform the task from the outset. Also, there is a circular dependency between task specification and

recruitment: How can users decide to participate before an overall description of the computation is presented to them, which would, in turn, require specifying which of them will contribute to this computation?

Incentivisation *Given special variables $X_{inc} \subseteq X$ that are under the control of an agent i , how should i choose this to solve the recruitment problem for a specific input set I ? This is a more specific sub-problem of recruitment: In our model, incentives can be viewed as variables X_{inc} whose values are set by the agent initiating an SC (e.g. modifying bank credit after task completion). How would these need to be chosen to persuade an adequate set of participants to contribute, and to execute their local tasks correctly?*

Synchronisation *Given a set $\{SC_1, \dots, SC_m\}$ of sequential SCs, what set of constraints $(i, j, t, x \text{ op } x')$ will enable all of them to be executed correctly? This essentially asks how we can resolve conflicts that could arise from the parallel execution of more than one sequential SC, and is important when we consider the open-world nature of the Web, where one SC may not be aware of the existence of the other, but may share resources/participants with it.*

Composition *Given a set $\{SC_1, \dots, SC_m\}$ of SCs that compute $\{f_1, \dots, f_m\}$, respectively, and a set of constraints of the form $(i, j, t, x \text{ op } x')$, what function f does the overall system compute? Complementary to synchronisation, in a sense, this question addresses more general problems of compositionality and emergent behaviour, as it may be the case that the joint effect of several SCs does not occur “by design”, but only as an indirect consequence of running several of them in parallel or in sequence.*

Optimisation *Given a quality measure q for SCs and an input set I , identify $SC^* = \arg \max_{SC} q(SC)$. Since SCs usually operate in resource-constrained environments, they will have to satisfy certain optimality criteria. Different from other kinds of systems, the quality of an SC is intrinsically multi-perspective and subjective (e.g. is it *fun*?), and its overall evaluation is subject to continual change. Hard, *a priori* optimality criteria are unlikely to work here.*

Casting these problems in an HDA-CAS context suggests a strongly incremental approach, where any successful solution method would need to specify (i) how it will discover new information over time to refine and improve an existing model; (ii) how it will adapt its operation to changing information; and, (iii) how it will expose its adaptability to designers and users that act as stakeholders in this process of evolutionary design.

2.3 The decision-making perspective

There is one aspect that is key to the analysis of HDA-CAS which has been deliberately left out from the above model, but which we want to touch upon briefly, as a “preview” to aspects that will become important in future stages of WP6 research: Our model so far is descriptive and does not allow us to formulate rationality constraints on behaviour that could serve as a basis for taking into account the motivations and preferences of human participants, or to specify how autonomously acting artificial agents should make rational choices based on their experience (be it at a global system-designer level, or as individual machine peers).

To move toward a model of rational decision making that would allow this, we start by translating the above to a discrete state-transition model, which shifts the focus of modelling from algebraic manipulation of variables to activities resulting from observed system states at the loci of decision making.

For this, we can define the set of global system states as $S = D_1 \times \dots \times D_k$, i.e. the set of all values the global set of variables can take on. The action set is given by $A = A_1 \times \dots \times A_n$ where agents' individual actions a_i are the $f \in F_i$ as applied to the appropriate subsets of their input variables. More specifically, the local state set for agent i is $S_i = (\times_{l \in X_i} D_l)$, and f changes the values of X_{out}^f depending on the values of X_{in}^f . With this, it is easy to introduce non-deterministic transition dynamic $T : S \times A \times S \rightarrow [0; 1]$ that enables us to describe the dynamics of the system even if there is uncertainty of execution (uncertainty of perception can also easily be accommodated, but for now we will assume this is captured as reduced predictability, if the modelling agent has an incorrect view of the state space or the action specifications).

To model preferences, we can assume that each of the variable configurations $x_i \in X_i$ agent i has access to can be mapped to a real-valued number $u(x_i)$ describing the utility of them having certain values (in practice most of these variables will be irrelevant to the agent, and only a small subset will matter). This can be used to define reward functions $R_i : S \times A \rightarrow \mathbb{R}^n$ for each agent, which are extended from states to state-action pairs, so as to take action cost into account, where relevant. A policy $\pi_i : S_i \times A_i \rightarrow [0; 1]$ for an agent in this multiagent Markov Decision Process then becomes a specific choice to invoke certain functions under certain circumstances, and allows us to use concepts from reinforcement learning, stochastic optimisation, and game theory to reason about collectives of utility-maximising agents.

In practice, rational reasoning will be over subsets of the overall state-action space relevant to a specific task setting. The modelling process outlined here serves as a general model which allows us to formulate criteria and mechanisms for rational behaviour which can be then considered in the design of HDA-CAS. Investigating the core research problems listed in the previous section in combination with the autonomous decision making perspective is the basis on which the longer-term research agenda of WP6 is built. In the remainder of this document, we map these general ideas to a specific static social orchestration framework and scenario as a first step toward this.

3 Static social orchestration model

To move from a model of social computation, which simply captures their composition from local computations and might arise in an unplanned and uncoordinated way from the individual activities of local nodes, to one of *social orchestration* (SO), we need to identify the functional building blocks that enable planned and coordinated collective activity. This cannot be achieved at the same level of generality as that aimed at with our previous constructions. We have to commit to a certain “style” of performing a social computation from the point of view of an agent or system designer who orchestrates it. For this, we take inspiration from the teamwork model of collaborative activity commonly

used in the multiagent systems, combined with elements of web-based collective intelligence systems (such as human-based computation and crowdsourcing platforms, online collaborative tools like wikis, and applications for coordinating human activity, such as meeting scheduling, task routing, etc). At this stage, the model we propose will be one of *static* social orchestration, i.e. the functions it uses do not change over time, and have a pre-specified semantics.

3.1 Definition

Many of the kinds of computations we are interested in consist of a common set of key functions: *discovery*, to identify appropriate peers who could perform it in principle; *assignment*, by which specific peers commit to participating in the computation in specific ways; *execution*, which produces the concrete behaviour of the peers that have agreed to participate; and *feedback*, which modify the state of the system after execution (these can be rewards and sanctions, ratings submitted by peers, reputation scores). In many systems, these functions are viewed as distinct and strictly ordered stages of the SO process, and, below, we will assume that this is true of at least the discovery-assignment-execution cycle, and that feedback is an update operation of the global state through local interventions that does not have to happen in close coupling to the task-orchestration stages (though it will normally depend on and refer to them).

Assume a class of tasks $\mathcal{F}$, among which we want to achieve a specific SC³ f . We define, at a fairly abstract level the above SO stages as follows:

- **Discovery:** A function $d : F \rightarrow 2^A$ that returns a set of possible agents for the task, i.e. there is a sequence $f_1, \dots, f_n$ with $f_i \in F_j$ for some $j \in d(f)$ and $f = f_n \circ \dots \circ f_1$ (note that we are not necessarily looking for a set of agents that could perform a specific sequence, but, effectively, the union of all agent sets for which some such sequence exists). In our model, individual agents may not have access to appropriate agents that can solve the problem, in which case, in turn, this would become a SC itself, where $d = d_n \circ \dots \circ d_1$ and $d_i : F \rightarrow 2^{N(i)}$ with $d_i(f) \subseteq d(f)$ and $d(f) = \cup_i d_i(f)$ (this discovery will not always terminate, of course, and it is not guaranteed to always add genuinely novel agents)
- **Assignment:** A function $a : 2^A \rightarrow (F \times A)$ where $a(A') = (i, f_j)$, $f_j \in F_i$, and $f = f_n \circ \dots \circ f_j \circ \dots \circ f_1$ for all $1 \leq j \leq n$, and a will normally only be defined for a specific subset $A' \subseteq A$ of agents. Processes like negotiation of conditions, refinement of the task specification etc are hidden here by the fact that the f_j assigned to agent i is simply selected from all the things that agent could do, which will include variations as to what incentives it would require, etc. To describe assignment as a computation within our framework, agents and tasks need to be reified in variables, so the assignment process itself becomes an SC.

³For simplicity, our construction here is limited to sequential tasks and ignores time constraints. In reality the overall task would be a collection of sequential SCs plus a set of synchronisation constraints that would have to be solved within some time interval.

- **Execution:** A function $e : \mathcal{F} \times A \rightarrow \mathcal{F}$ determines agent i 's behaviour, such that $e(f_j, i) = f'_j$ denotes that agent i will perform f'_j when it has been assigned function f_j . Normally, it will be desirable that $f = f'$ for every participating agent, or at least that $f' = e(i_n, f_n) \circ \dots \circ e(i_1, f_1)$ on the relevant inputs with regard to the outputs we are interested in (i.e. side-effects that are generated by the actual computations performed by agents $i_1, \dots, i_n$ can be ignored may vary). The outcome of the execution phase is $f'(x^t) = y^{t+k}$ using the notation of our abstract SC above.
- **Feedback:** A function $k : \mathcal{F} \times A \rightarrow \mathcal{F}$ determines what additional functions will be performed by every agent to reflect the consequences of the task execution in terms of rewards, observations about the observed (e.g. agents' performance, success/failure, etc), where agents i (not necessarily only those involved in the execution of the task) compute $k(f', i) = k'$ based on (what they can perceive of) the global computation that has occurred.

With this, we can describe the outcome of an attempt to orchestrate an SC as performing the first three stages, i.e. $o(f) = e(a(d(f)))$, where various additional feedback steps $k(o(f))$, $k'(o(f))$, $\dots$ may be performed by various agents before, during, or after the computation. Note that the outcome $o(f)$ of the orchestration may be very different from the originally intended task f , i.e. the SC may fail, produce unintended side-effects etc.

3.2 Implications for design

The primary motivation for moving from a general, abstract model of SC to a more constrained, though still fairly high-level, specification of SO is to be able to ask more specific questions about the representations and algorithms that are necessary in order to build actual HDA-CAS to a specific set of requirements. A consequence of this transition to a more bounded design space is that we have to make certain assumptions and establish some general requirements for any implementation that follows this framework.

- **Communication network:** The requirement for each peer to have knowledge of at least some neighbours in the network is already captured by the abstract SC model. Performing the necessary interactions assumes that reliable and scalable decentralised communication channels are available. A commonly agreed communication language needs to be available, so that messages are interpreted correctly by all peers. Appropriate access control mechanisms need to be in place to ensure shared data is appropriately accessed and correctly manipulated.
- **Representational requirements:** The signature of the above functions requires that peers be able to talk about tasks, peers, and knowledge of the environment pertinent to the task in hand, so that they can process these as inputs and produce the right outputs. This implies that an agreed ontology about peers, objects, actions, task workflows, time, commitment, rewards and sanctions, and relevant social constraints is available, and that the status of the overall orchestration can be shared (e.g. to inform each other about agreed, completed, failed tasks, to refer to previous tasks when providing feedback, etc).

- Processing requirements: Each peer needs to dispose of an internal “algorithm” (which may not be formulated in a computational way in the case of human computation) for translating appropriate task specifications, i.e. generating some state-modifying behaviour given certain commitments made. Every peer needs to be able to inspect and transform local variables, and to determine when this should be done in accordance with synchronisation constraints; it also needs to have appropriate means for tracking the execution status of a complex task and knowing when its contribution is required (this may require additional sensing or communication to observe non-local variables).

At first glance, none of these requirements seem to be very different from those involved in the design of a traditional distributed system. The challenge and novelty arises from our aim to realise them in such a way that our architecture allows for a continual co-design of these systems by all stakeholders involved, so that the composition of complex HDA-CAS can successfully exploit the CCC principle. In the next part of this document, we propose a concrete, computational architecture which we have designed to make this possible.

4 The Play-By-Data architecture

4.1 Introduction

To enable individuals to build, participate in, and adapt HDA-CAS, we need to provide a computational infrastructure that is as *lightweight* as possible in terms of making assumptions that will invariably break as the system grows, the behaviour of participants changes, or the system environment changes. For these systems to achieve broad uptake and be composed in a loosely coupled way with each other, we also need to keep them lightweight in terms of ease of entry and use, and interoperable with existing systems.

To explain the intuition our architecture is based on, it’s worth thinking about human collaboration more generally, and to consider how people got tasks done collaboratively when they were not in the same place and when they didn’t need to be co-present to perform their local contributions to the task before digital communications existed?

A good example of this is correspondence chess, which people have been playing for decades. Using post-cards like the one shown in figure 2, two players would send each other information about their moves, following the turn-taking rules of the game. The information contained on these postcards involved not only details about the actual moves performed locally, but also about temporal constraints (e.g. by when a response is expected), debugging information (such as the statement “your move is not clear” or “your move is impossible”), and control messages like “I offer/accept Draw” that indicate termination with various outcomes. Importantly, this did not require the local state of the board to be communicated as long as the initial state was common knowledge. Everybody could maintain a local representation of the current state synchronised among the players at all times, as long as the right turn-taking rules were observed and commonly known.

Compared to social computation, this example is of course somewhat limited, in that it is competitive (though it can be easily replaced by a collaborative one, e.g. solving a

<table border="1"> <tr> <td></td><td>A</td><td>B</td><td>C</td><td>D</td><td>E</td><td>F</td><td>G</td><td>H</td><td></td></tr> <tr> <td>8</td><td>18</td><td>28</td><td>38</td><td>48</td><td>58</td><td>68</td><td>78</td><td>88</td><td>8</td></tr> <tr> <td>7</td><td>17</td><td>27</td><td>37</td><td>47</td><td>57</td><td>67</td><td>77</td><td>87</td><td>7</td></tr> <tr> <td>6</td><td>16</td><td>26</td><td>36</td><td>46</td><td>56</td><td>66</td><td>76</td><td>86</td><td>6</td></tr> <tr> <td>5</td><td>15</td><td>25</td><td>35</td><td>45</td><td>55</td><td>65</td><td>75</td><td>85</td><td>5</td></tr> <tr> <td>4</td><td>14</td><td>24</td><td>34</td><td>44</td><td>54</td><td>64</td><td>74</td><td>84</td><td>4</td></tr> <tr> <td>3</td><td>13</td><td>23</td><td>33</td><td>43</td><td>53</td><td>63</td><td>73</td><td>83</td><td>3</td></tr> <tr> <td>2</td><td>12</td><td>22</td><td>32</td><td>42</td><td>52</td><td>62</td><td>72</td><td>82</td><td>2</td></tr> <tr> <td>1</td><td>11</td><td>21</td><td>31</td><td>41</td><td>51</td><td>61</td><td>71</td><td>81</td><td>1</td></tr> <tr> <td></td><td>A</td><td>B</td><td>C</td><td>D</td><td>E</td><td>F</td><td>G</td><td>H</td><td></td></tr> </table>																	A	B	C	D	E	F	G	H		8	18	28	38	48	58	68	78	88	8	7	17	27	37	47	57	67	77	87	7	6	16	26	36	46	56	66	76	86	6	5	15	25	35	45	55	65	75	85	5	4	14	24	34	44	54	64	74	84	4	3	13	23	33	43	53	63	73	83	3	2	12	22	32	42	52	62	72	82	2	1	11	21	31	41	51	61	71	81	1		A	B	C	D	E	F	G	H		Partie Partie Game Partida Партия		Nr No No No No		Ihr Zug Votre coup Your move Su jugada Ваш ход		Nr No No No No		Mein Zug Mon coup My move Mi jugada Мой ход		Turnier Tournoi Tournament Torneo Турнир		Nr No No No No	
	A	B	C	D	E	F	G	H																																																																																																																									
8	18	28	38	48	58	68	78	88	8																																																																																																																								
7	17	27	37	47	57	67	77	87	7																																																																																																																								
6	16	26	36	46	56	66	76	86	6																																																																																																																								
5	15	25	35	45	55	65	75	85	5																																																																																																																								
4	14	24	34	44	54	64	74	84	4																																																																																																																								
3	13	23	33	43	53	63	73	83	3																																																																																																																								
2	12	22	32	42	52	62	72	82	2																																																																																																																								
1	11	21	31	41	51	61	71	81	1																																																																																																																								
	A	B	C	D	E	F	G	H																																																																																																																									
Ihr Zug ist unklar Votre coup n'est pas clair Your move is not clear Jugada maldefinida Несный ход																																																																																																																																	
Ihr Zug ist unmöglich Votre coup est impossible Your move is impossible Jugada imposible Невозможный ход																																																																																																																																	
Ihr Poststempeldatum Votre date de la poste Your postmark date Su fecha postal Дата вашего почтового штампа				Ihre Bedenkzeit Votre temps Your time taken Tiempo consumido por Ud Ваше время				Tage Jours Days Dias Дни		Ihre Zeit insgesamt Votre temps total Your total time Su tiempo total Ваше общее время				Tage Jours Days Dias Дни																																																																																																																			
Empfangen am Reçu le Received on Recibido el día Получено				Meine Bedenkzeit Mon temps My time taken Tiempo consumido por mí Мое время				Tage Jours Days Dias Дни		Meine Zeit insgesamt Mon temps total My total time Mi tiempo total Мое общее время				Tage Jours Days Dias Дни																																																																																																																			
Beantwortet am Répondu le Replied on Contestado el día Отвечено				Partie Partie Game Partida Партия				Ich biete Remis Je propose partie nulle I offer Draw Ofresco tablas Предлагаю ничью				- Ich nehme an - J'accepte - I accept - acepto tablas - соглашаюсь на ничью				- Ich lehne ab - Je refuse - I refuse - rechazo - Отказываюсь от ничьей																																																																																																																	
Urlaub Vacances Holidays Vacaciones Отпуск von du from del от bis au to hasta до				Herzliche Grüße Salutations Yours sincerely Saludos С приветом																																																																																																																													

Figure 2: Postcard for correspondence chess

puzzle together), very small-scale (though one can imagine groups of people playing simply through a rotating chain letter or a central co-ordinator broadcasting to everyone), and the task is quite simplistic in principle. However, it also has some interesting properties:

1. It shows how a co-ordinated activity can be performed despite spatial distribution using only relay communication (= snail mail).
2. Communication or reasoning errors have no damaging effect on the integrity of the local state representations (= chessboards in players' houses) or on the availability of local computation nodes (= players) for other activities (e.g. other games happening in parallel).
3. Synchronisation actions and local state update is left to the owner of the data (=recipient of the postcard).
4. The mechanism is oblivious to the extent to which the global computation is decentralised or centralised (e.g. all players could be deciding on their moves by using the same chess Web server or locally, and the process would look exactly the same).
5. The co-ordination complexity only grows in the number of messages exchanged, not in terms of local representations as the number of participants grows (a thousand players need five hundred chessboards, but these are completely decoupled from each other).

The architecture which we propose, called “*play-by-data*” (PBD) follows this “play-by-mail” metaphor in an attempt to provide a framework for implementing our above social orchestration model that inherits the same properties, adapted to the reality of modern digital communications: Nowadays, we obviously don’t need postcards, we have the Web. In the broadest sense, the postcards can be documents with content data expressed using common Web standards like RDF and XML, and accessed over HTTP, just like the correspondence chess postcard is, in a sense, a document on a piece of paper. PBD is based on precisely the idea, namely that the data in a social computation *is* the computation.

It is important to point out the difference between PBD and existing platforms to manage decentralised autonomous interactions: Unlike many of these systems, PBD does not suggest a bespoke infrastructure which can only manage interactions when mediated by platform-specific messaging protocols and software components residing on “gated” server or custom client-side applications (such as common workflow engines, multiagent platforms, common peer-to-peer systems, or existing human-based computation/crowdsourcing web sites). Instead, it allows peers to describe SCs by virtue of the interaction models they permit using only common Web standards and transformations on resources that are exposed to the extent necessary to achieve “approximate shared state” (*true* shared state cannot be achieved if we want to ensure systems are robust to failure while operating without strict synchronisation and heavy assumptions on interoperability). We argue that viewing social computations as collections of clients and servers accessing each other’s data as the computation unfolds is the most scalable, robust, and lightweight way of implementing them using current technology.

4.2 Architecture

The “play-by” idea in PBD captures both the fact that all interaction is mediated by persistent, shareable, and locally owned data, and also that this data specifies how the computation is performed, as in “play by the rule” (set out by the data). The principles of this paradigm are the following:

1. A global *playcore* ontology is provided that is used to bootstrap new PBD interactions. This contains basic constructs to describe peers, tasks, messages, and interaction models that define admissible sequences of messages. It also specifies basic constraints that any node needs to implement who wants to participate in distributed PBD computations, and can be used to form more complex, application-specific constraints. To enable very minimal discovery, it will also contain a set of basic exploration protocols to find potential interaction partners, and to solicit information about them.
2. Each so-called *playnode*, representing a peer or group of peers that may participate(s) in PBD computations, is associated with a base URI (and can be either directly addressable server itself or be contained as an entity representing a client on another server), and is essentially a datastore that specifies rules for how this datastore responds to incoming messages in terms of updates to its local data and messages that will be sent in response to others. This information is provided as a *playspec* local to the node, expressed using the *playcore* ontology.

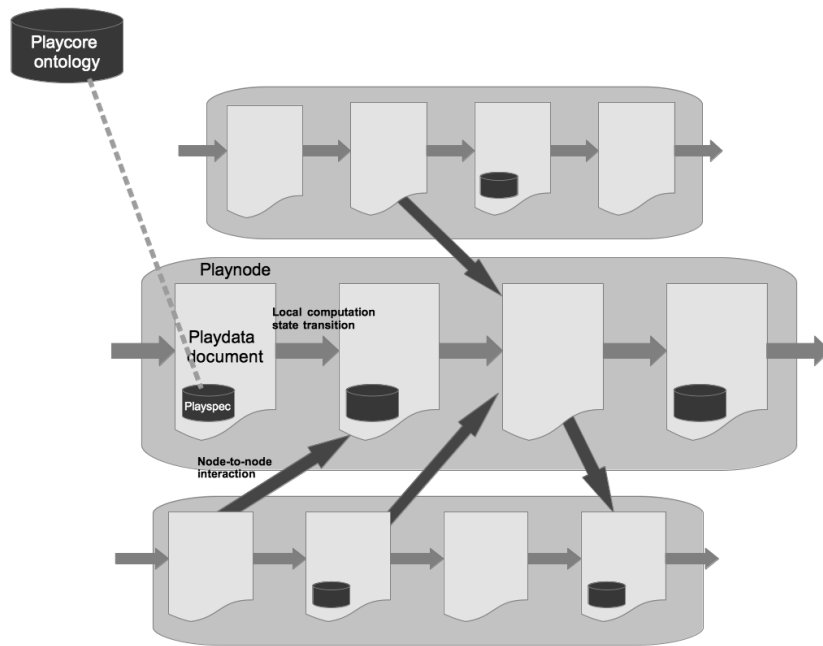


Figure 3: Schematic overview of the Play-By-Data architecture for social orchestration

3. The *playdata* stored with a *playnode* describes any local information that is to be exposed to others in order to share information or capture globally relevant aspects of the state of the overall computation. It may involve results of locally performed functions, attributes of peers, their concrete action capabilities, incentives they can offer or may require, groups among peers that are defined by relationships and roles between individuals, conventions and norms, as well as provenance data including past interaction logs, trust and reputation values, etc.
4. Both local computations and node-to-node computations are realised as transformations of resources that produce new resources (normally, Web documents) in line with the REST paradigm (i.e. only using ordinary HTTP messages, and sharing state by providing access to appropriate resources). These transformations are triggered by exposed *playservices*, and provide the communication interface among clients and servers, where some clients may of course be servers at the same time.

Figure 3 shows a schematic overview of the PBD architecture. This shows how individual playnodes, as the agents i of our abstract SC model, manipulate variables X_i on web resources using the atomic functions F_i they can perform, and obtaining inputs/sharing outputs with other playnodes. The different resources created over time correspond to the application of a function $f_j(x)$ to the respective inputs and outputs. The new element PBD introduces are the *playspecs* defined in terms of it, which are required for a reification and description of the elements required to perform the building blocks for social *orchestration* rather than abstract computation, and the communication interfaces defined by *playservices*. Note that *playspecs* may remain valid for sequences of individual computations or change themselves under particular circumstances (this is hinted at by

playspec containers appearing only in some playdata resources.

4.3 Benefits

At this stage, it is useful to justify the design choices we have made above to explain the rational behind them. The following is a list of key advantages of the approach:

1. It follows the principles of the architecture of the Web. This means we can look at social computations, both past and ongoing ones, as linked data with all the flexibility that comes with that, e.g. using third-party services and data on the Web, and scaling up computation models by using the decentralised data storage and computation facilities that Web components share by virtue of simple, generic interoperability standards.
2. It does not require a bespoke computational runtime application. As many previous efforts have shown, unless the purpose of the overall system is so broad that it can attract millions of users (as, e.g., in P2P filesharing systems), it is unrealistic to assume that we can build a platform that will achieve the uptake needed to demonstrate how new social computation techniques work in the real-world.
3. It provides a separation of concerns between the shared computational process and actual runtime processing. Following this architecture does not require making commitments as to whether and how interaction flows are planned and executed at runtime. Playnodes can be implemented at various levels of abstraction, for individuals or collectives, managing arbitrary portions of the global playdata space.
4. It enables a unified way of managing and analysing data. Individual nodes can specify access policies to their local data through disclosure services with appropriate authentication methods that ensure privacy. Subject to privacy constraints, comprehensive descriptions of past interactions can be made available for analysis, following common Web standards that make them amenable to the use of Big Data methods.
5. It lends itself to human-in-the-loop, open-ended modelling of context, knowledge, and inference. We envision that automated support will be provided by (locally or globally provided) inference services, for example to plan interaction sequences, to search for appropriate peers, to automatically generate execution sequences for social workflows, or to repair broken computations. Where such automated inference fails to produce complete, executable, and verifiable computations, humans can fill in the gaps, using human-level intelligence and interpretation capabilities.

A separate argument needs to be made regarding why social orchestration needs consider architectural issues at all – after all, it is about the *organisation*, rather than *implementation* of complex decentralised computations in HDA-CAS! The reason for this is that the computational architecture on which these will run *does* affect their design in two important ways:

Firstly, to capture the nature of voluntaristic collectives of collaborating (human and machine) computation units inherent to HDA-CAS, we need to move away from a traditional “platform” view of distributed computation where these units are ultimately under the control of a “container” that runs them. In the kinds of systems we’re interested in, control of data and process is restricted to the playdata pertaining to a playnode, and any manipulation performed on is the result of the behaviour of an autonomous peer. A systems architecture like PBD is required if we want to study the behaviour of individuals and collectives as autonomous actors and identify what incentives and motivations will lead to certain behaviours.

Secondly, as our abstract SC model has already shown much earlier, we are interested in an environment where the sources and results of computations are simply “out there in the wild”, i.e. there are no guarantees as to whether they are correct, up-to-date or synchronised with each other, pertain to computations that are still relevant, are fixed or still evolving, have been corrupted, or whether they are linked to each other in a correct way. This is exactly what is true of data and users on the Web: their validity and usefulness is only verifiable at the moment when they are accessed. Only by embracing this openness we can attempt to develop a principled way of understanding and managing the kinds of systems HDA-CAS operate in.

5 An example

To illustrate the workings of the conceptual and architectural principles laid out above, we describe a first design for an application for a ridesharing HDA-CAS, of which we are currently developing a prototypical implementation. We start by introducing the ridesharing scenario and explaining its value for our work. We then present an outline of its implementation, mapping its individual components to the social orchestration architecture introduced above. This is then followed by a discussion of important research issues that have arisen from this design and implementation effort.

5.1 The ridesharing domain

Ridesharing is the activity of several human travellers sharing means of transportation for (parts of) a journey. As it may contribute to a global reduction of traffic, pollution, and energy consumption, and may improve the utilisation of private and public means of transportation, it is a valuable means of addressing important societal and environmental challenges. At the same time, it offers concrete benefits to participants in terms of improving comfort and mobility while reducing travel cost. Beyond these features, ridesharing has many further properties that make it ideal as a problem domain for the development of SmartSociety systems: The complex route and resource constraints combined with the different travel needs of individual travellers create a highly complex combinatorial problem, where machine computational can be useful in identifying and matching appropriate peers in a large-scale market of potential travel requests, plan routes and timings for them, and analyse global effects arising from overall patterns of use within a certain geographical areas. Further, the contributions required by human participants involve sequences

of individual, inter-dependent activities, such as driving or travelling on public means of transportation, meeting each other, and reacting to unexpected delays and failures. They also involve a host of complex psychological and social constraints regarding individuals' preferences, safety expectations and liability arrangements, as well as financial exchanges.

The ridesharing prototype we are developing in the project aims to provide a “minimal” implementation of this kind of system. In the current phase of the project, we deliberately limit ourselves to car sharing, where co-travellers share the entire journey, and negotiate the precise terms of the deal themselves without machine intervention, and a centralised ridesharing server performs matchmaking of ride requests, tracking of agreed rides, and provides a facility for submitting feedback. Also, we do not assume that the trip itself can be observed by the ridesharing software platform, or that the system will be necessarily notified when trips are completed (whether successfully or not).

There are several motivations for following such a minimal design: Firstly, we want to replicate the functionality of existing ridesharing platforms using our lightweight social orchestration principles to show that they conform to the kind of functionality provided by existing social computation systems. Secondly, we deliberately want to produce a system that is “maximally human-oriented” so that we can gain insight into the contribution of intelligent automation support in a bottom-up, incremental way. Our previous work [1] on automated ride planning algorithms can be used to add further machine computation to support more advanced algorithmic processing in the future: The algorithm presented there clusters potential travellers together in likely groups of co-travellers and computes plans for complex, multi-modal journeys that involve real-world public transport as well as private cars for large numbers of travellers and transport connections/services. Thirdly, the purpose of the prototype is to inform our further research through simple simulations and human experiments, rather than produce a full application demonstrator.

It should also be pointed out that the current implementation has not involved designing the playcore ontology and playspec structures that would allow for automating more of the social orchestration functionality based on machine-readable specifications of peers, tasks, and interaction processes. Rather, the current prototype has all these elements still hardcoded into a domain-specific Web application, so that we can extract requirements for a future domain-independent design of these elements of our architecture from a concrete case study in the next phase of the project.

5.2 Implementation

5.2.1 Overall architecture

The ridesharing prototype is designed as a PBD system with two machine playnodes providing automated support for the discovery/assignment (“matchmaking service”), and feedback (“reputation service”) stages of our social orchestration model, realised as Web servers. The remaining playnodes can be arbitrary human users who execute the actual rides, and can manipulate their local data (e.g. choosing rides that they are interested in, or generating feedback) and submit it to the respective server. As these human nodes do not need to be addressable for the ridesharing application in hand, they are simply realised

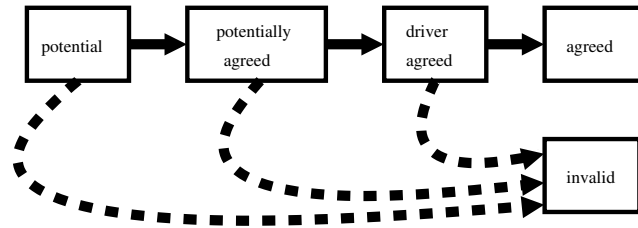


Figure 4: Different states of a ride plan in a ride request. Transitions happen automatically on the matchmaking server based on information provided by users who appear in the plan.

as “clients” in the Web architecture sense, though this concept is rather misleading here, since in fact they can simply store their data locally, manipulate it, and communicate with the servers. The only implication of them not being servers is that they have to “pull” information on globally exposed state changes in the system since those cannot be “pushed” to them by the machine nodes. This represents a fairly simple PBD design, but one that is very similar to existing Web-based social computing platforms.

The matchmaking service allows users to create profiles with persistent preferences (currently we use preferences regarding “smoking in the car” and “pets in the car” as examples of these, based on experience BGU has with a ridesharing system locally used at their university for staff and students), post requests for rides either as drivers or commuters tied to specific constraints (timing, cost, etc), and agree rides with each other (through a multi-stage process explained further below). The reputation service allows user to submit feedback for rides they have participated in, and retrieve reputation reports for other users. Authentication data and provenance information regarding all relevant data transformations performed on the machine nodes of the system are also stored by the system, so that a trace of all past interactions is available at all times.

The feedback services provide users with a facility to give feedback and rate other users with whom they have travelled together in the past. Part of the rating process involves users indicating what *opinion* they hold about other users. This captures how well they blend together with the particular matched users of the system, based on previous experience. Also, users specify a *ride quality threshold*, which indicates the minimum accepted average opinion that must be held about others in a ride plan for a user to be willing to negotiate further with these potential co-travellers.

Figure 4 presents the different states ride plans can be in. For every ride request received by a user the matchmaking service generates plans that are classified either as *potential ride plans*, or as *potentially agreed ride plans*. The latter refer to ride plans acceptable to all participants in terms of opinion thresholds. Ride plans become *agreed* if all participants have indicated that they are willing to commit to a ride and perform it. Finally, ride plans where agreement has not been reached yet may become *invalid* when at least participant in a ride plan commits to a different plan that was generated by the system for the same ride request; thus all other options that involve this participant (and correspond to the same ride request) are automatically rendered invalid. As a consequence, all other users involved in these plans will be notified about this change the next time they

fetch an updated overview of plans that match their ride request.

There is a subtle issue that is important here, namely that ride preferences can be specified at different levels of generality: they may be globally valid for all ride requests, specific to a particular request, or even specific to a suggested ride plan returned for a request. At the moment we are treating all preferences (and opinions) as global, but the system allows these to be specified at finer-grained levels if necessary.

5.2.2 Discovery

The process of discovery in the ridesharing prototype unfolds in a “pull” rather than “push” style, which is heavily based on matchmaking rather than on exploration of a peer network: Individual users post ride requests, and the matchmaking service advertises these requests to every peer who might be interested. Its main functionality is to identify potential matches based on origin, destination, time of departure, expected time of arrival, cost, (and, in the case of requests coming from drivers, available number of seats), and to construct potential ride proposals (i.e. ride plans) for potentially matching users.

This simple matchmaking process already raises important algorithmic problems that relate directly to compositionality concerns: Firstly, even with relatively small numbers of travellers between similar destinations, the set of possible ride plans can be prohibitively large (imagine 100 people travelling between the same locations, where 30 of them are drivers of cars offering up to 3 spare passenger seats). In practice, this means that we must produce subsets of all possible solutions (possibly ranked using contextual information), and abandon completeness. Secondly, while in our simplistic prototype the satisfaction of ride constraints is very straightforward and involves no route planning, their computation based on real map data and with revised estimates for travel time and more general models for cost-matching would involve a significant amount for computation. This, in combination with the previous issue, effectively means that we would be looking for a *group recommender* system that prioritises a reduced set of solutions in such a way that their acceptance for all travellers is more likely, to avoid wasting computational effort (and, thus, responsiveness of the ridesharing system). This highlights a fundamental, complex feedback loop that underlies the compassion of social computations in HDA-CAS: knowing what computations are feasible requires working out their details algorithmically, but these require knowledge of the humans who would contribute to them, who, in turn, cannot reliably be asked whether they are willing to contribute until these details are known.

5.2.3 Assignment

Figure 5 presents the different states in which users can participate in a ride plan. Here, on the level of individual ride plans, the distinction whether a *user* is considered to be *potential* or *potentially agreed* is manipulated based on the opinions of the user about the other participants in the specific ride plan as well as the value of the user’s ride quality threshold. This state transition occurs automatically by the matchmaking service when the ride plan is generated and can be manipulated when the user changes her opinion about others in the plan, or modifies her ride quality threshold appropriately. If all users who

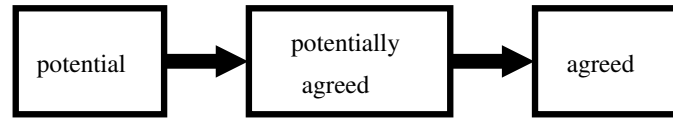


Figure 5: Different states of users in a ride plan.

appear in a plan have a positive *opinion* of each other (all appear as *potentially agreed* in Figure 5), the plan itself becomes a potentially agreed ride plan on the level of the relevant ride requests (Figure 4), i.e. a serious candidate solution for the users' requests. The plans might lose their status as such (or regain it in the future) if *at least one* reputation value falls under the relevant threshold due to feedback submitted after their first calculation (or if the reputation values for *all* the participants climb above quality thresholds).

In terms of social computation, the above steps involve the composition of the ride generation service with the reputation service, which are coupled here through the identities of users involved in future potential rides known to the matchmaking play node, and mentioned in previous testimonials stored on the reputation and provenance play node. Synchronisation between the two services is kept as lightweight as possible: feedback can be added to the reputation service at arbitrary points in time. To complete assignment, however, we need an additional step which involves obtaining actual commitment from human participants, i.e. negotiation.

Negotiation is initiated by drivers. As owners of vehicles, their consent is essential. The driver selects one of the plans appearing as potentially agreed and indicates that she is an *agreed participant* (Figure 5) for that plan. The platform then automatically updates the ride requests of the involved users reflecting this change; i.e. the specific plan is now a *driver agreed ride plan* (Figure 4). It is now the turn of the commuters to continue with the negotiation process. Once all commuters appearing in the ride plan agree on this specific plan, assignment is completed and the ride plan is automatically promoted to an *agreed ride plan* (Figure 4) and a *ride record* is created, which contains links that can be used by the participants to post feedback about it. The reputation playnode is ready to accept feedback to the assigned social computation.

In terms of PBD implementation details, the information and data flow is organised as follows: Users post ride requests to the ride sharing service. These requests (documents) are then modified by the ride sharing service so that they include links to the ride plans (again documents) that are generated by the matching algorithm. Users are modifying their local copies of ride plans during the negotiation phase and these changes are automatically reflected on the level of ride requests by the system. Of course the other way of influencing negotiation is by changing opinions about others in the plan or modifying ride quality thresholds and notifying the system about this change in local playdata.

5.2.4 Execution

As stated above, our ridesharing system does not track or monitor individuals during the execution of a ride. The only globally exposed state transition that occurs once assignment is complete is that participants may now submit feedback during (or after) the execution

of the ride. More precisely, feedback can be submitted as soon as a ride record has been created (which happens automatically upon mutual agreement).

Though one may imagine that in more complex systems execution information may be directly available to the system (e.g. by tracking the GPS location of the participants' mobile devices, or allowing them to submit information about completion (or failure) of sub-steps of the ride plan), leaving execution opaque is a deliberate choice to illustrate how parts in a social computation may not be observable by those not directly involved in a human activity *outside* a computationally managed infrastructure. This “opaque” mode of execution is in fact quite common in the real world, for example when human users trade physical goods on electronic consumer-to-consumer markets. In many scenarios that involve this kind of execution, the only information the machine nodes in the SC obtain is through feedback.

5.2.5 Feedback

Feedback can be submitted for agreed ride plans both by the drivers as well as commuters at any point after agreement (the system has no way of checking whether the ride has already started, has been completed, or has actually been abandoned). Apart from feedback reports submitted by users, two more reputation reports are generated: One of them is based on statistics computed over interactions recorded by the application about a ride request, while the other is a summary of the reputation reports submitted about a particular user. Moreover, the feedback service allows users to change their opinion about other participants in an agreed ride. Such feedback allows users to be better matched by the system in future ride requests they may post to the matchmaking service.

This illustrates how contextual information obtained from collectives of human users is used by the system in the composition of future tasks, and constitutes a simple implementation of our CCC principle as suggested in the introductory sections of this document. Effectively, what the system is trying to achieve here is to solicit information about factors that affect the future probability of success for proposed SCs from human users that is not directly available to it, thereby using human intelligence and machine-driven data analysis to augment the compositionally properties of the system. The design principle this method is based on is that while information about ride requests and negotiation outcomes (known to the system) alone will *not* be sufficient to compute future rides successfully, supplying user-provided context to the algorithm that computes them *will*.

5.2.6 Social computation and compositionality

In terms of social computation, the overall task of the system is to improve and explicate existing “neighbourhood” structures (as described in our abstract SC model) that will increase the likelihood for individual computations (individual agreed rides) to occur so that the overall social computation (collection of all rides) leads to better resource utilisation and mobility for all involved. This is essentially achieved by allowing the neighbourhoods of the individual human users to evolve over time, as they participate in rides and/or as a consequence of the feedback submitted about others, which reinforces good neighbourhood links and thus connectivity among the social network.

As regards task composition, each of the individual rides is considered independent, i.e. the system does not check, for example, whether a single user has over-committed themselves to parallel rides. In terms of sequential composition, each ride may involve contributions from a small number of peers (typically 2-5, depending on the spaces available in a driver's car) which all have to agree sequentially to participating in a given ride before that ride is complete (in terms of agreement, not of execution, which is currently not captured by the system as described above).

6 Discussion

In this section, we review what our work on lightweight social orchestration has achieved so far, and discuss the next steps that are necessary to accomplish the longer-term aims of the workpackage and its relationship to other activities in the project.

6.1 Work so far

In order to achieve a *lightweight* design, our work has so far proceeded in a strictly bottom-up fashion. We developed an abstract, purely algebraic model of *social computation* that allows us to capture important compositionality issues in HDA-CAS starting from first principles, while making no specific assumptions about its computational realisation. This very simple model allows for sequential and parallel composition, hierarchical organisation, and embeds social computation within networks of interaction. We then showed how this can be translated to a model of distributed rational decision making for autonomous agents. This alternative view is essential if we want to analyse and predict emergent patterns in human behaviour, and allows both normative and descriptive models of rationality to be super-imposed onto the previous abstract model.

The next step has been to propose specific functional building blocks that can be used to orchestrate a broad range of real-world social computations. Our *social orchestration* architecture instantiates the abstract social computation model with key functions performed by human and machine nodes which are needed to bootstrap organised computer-mediated man-machine workflows. By providing specifications of these and indirectly assuming that their purpose and nature are understood by all participants, it fills the gap between an abstract view of “social algorithms”, and provides guidance for the specification of specific computational artefacts that support the development of implemented HDA-CAS.

Enabling such implementation requires a minimal computational infrastructure that can be assumed. Our proposed Play-By-Data architecture is not a “software architecture” *per se*, but much more a set of architectural principles to which any concrete social orchestration implementation must adhere. It is strictly data-driven, imposes minimal communication requirements, and maps the architectural principles of the Web to the requirements of our conceptual and theoretical models. Following these principles is likely to result in loosely coupled and robust large-scale systems which do not require reliable, persistent communication links, allow for redundant information storage and caching, and are thus well-equipped to cope with the demands of complex HDA-CAS.

Finally, we presented an implement example of our social orchestration framework in a PBD-based system to illustrate the principles of both ideas while adhering to our initial vision and abstract framework. While it is certainly not the case that a system like our ridesharing prototype is, in itself, novel in its functionality, the main achievement of our specification and design efforts has been to produce a consistent “pipeline” from conceptual analysis to concrete implementation.

6.2 Next steps

As proposed in the project workplan, our work so far has produced a *static* social orchestration architecture which ignores adaptation. This limitation manifests itself in different ways, and each of these manifestations directly suggest what the next steps in our work on social orchestration and compositionality will be:

Firstly, the SO specification is one where concrete, domain-specific functions need to be implemented in the different components of the social computation system. This is not only evident from the fact that the communication protocols and data transformations in the ridesharing example are implemented on an *ad hoc* basis in a fairly traditional Web programming style to perform the functions needed to bring about the overall computation. It is also obvious from the lack of actor, task, and process ontologies (the “play specs” alluded to above in a very cursory fashion) that are missing from our specification. Without these, and execution engines that can operate on them, the system essentially relies on full *a priori* agreement on the role each node will play in the interaction, and existing specifications cannot be adapted to new ones. The view we take regarding adaptation here would be that in order to achieve interoperability within a new or modified system, there needs to be a common minimal set of interoperable standards so that peers can define novel social computations in terms of existing ones. (After all, remembering our proposed CCC vision of compositionality, this is exactly the human- and machine-driven process of evolution by which we expect contextual information to be used to incrementally enhance systems.) Our social orchestration model makes some headway toward this, in that it hints at what conceptual categories will need to be included in such dynamic orchestration methods, where execution engines could “run” shared specifications of social computations. Developing such ontologies and engines will be a focus of our work in the next project phase, and will also involve an exploration of the extent to which RESTful web applications can be built from generic descriptions of actors, tasks, and interaction models, which is something that, to our knowledge, has not been attempted before. Work on these issues will involve close collaboration with WP4 on peer profiling, WP1 on the formal modelling framework, and WP8 on the SmartSociety architecture.

Secondly, our orchestration method does not propose any specific methods for *algorithmic adaptation*. By this we mean the “intelligent” part of orchestrating social computations, which involves automated methods for adapting incentives and social rules, making recommendations, and resolving conflicts. We have already mentioned two fundamental research challenges above that need to be addressed to enable such intelligent support for orchestration: (i) the tension between synthesising complex tasks and eliciting contributions from their potential participants to these tasks, where each of these two steps

cannot be performed in isolation from the other and (ii) the problem of group recommendation, i.e. prioritising proposed tasks based on their likelihood to be acceptable for collectives rather than individuals. We have seen (fairly trivial) examples of how these issues can be addressed in the ridesharing example, namely its mechanism for improving recommendations based on past feedback, and the sequential agreement process it employs, respectively. To achieve a deeper understanding of these problems, further work is necessary on various more specific issues related to the modelling of collectives and their behaviour. Firstly, this is necessary to capture the structure of collectives and the operations they permit, such as agreement, delegation, representation, and stereotyping, and to use these for more advanced recruitment and task suggestion methods based on analysis of past behaviour. Secondly, to be able to detect emergent functionality when various computations run in parallel and impact on each other (a simple example of this would be to detect congestion problems caused by parallel rideshares, or improving the reliability of participants by policing overcommitment and imposing stricter execution monitoring rules – e.g. that they have to report their location in fixed intervals so other co-travellers can be reassured about the feasibility of an already initiated ride).

Work on these issues has many different facets related to distributed autonomous decision making, planning and execution monitoring, and machine learning, and touches on fundamental problems in AI, such as uncertainty and partial observability, strategic models of individual and collective behaviour, and mechanism design. On the incentives design part, it will involve close collaboration with WP5, and on the data analysis side with WP2. The intelligence of how HDA-CAS are adapted will also of course involve human intervention, and we will make sure, in collaboration with WP1 and WP3 that our models and architectures encompass facilities for such intervention and its utilisation by the system.

7 Related work

In this section we present a survey of the existing work on composing machine and/or human elements to perform complex computations, and relate their contributions to ours. Within this very broad space, we focus on three specific areas: agent-based systems, which provide key coordination techniques for the kinds of systems we are interested in, focusing on the representations, reasoning, and interactions involved; workflow systems, which address issues relating to the organisation, management, and execution of complex processes and services; and frameworks for human-based computation, which provide concrete programming support for building systems that involve human services.

We should remark that, to our knowledge, there exist no methods that fully support our view of social orchestration in the HDA-CAS context, i.e. none of them considers hybridity, collectivity and adaptability jointly. However, considered separately, previous solutions give insight into different possible approaches for addressing some of the key research issues we are interested in.

7.1 Agent-based systems

The agent-based systems literature [2, 3, 4] abounds with techniques for coordinating autonomous, rational agents. These range from specifications of agent communication languages and interaction protocols via negotiation mechanisms to multiagent plan coordination, norms and institutions, trust and reputation mechanisms, and multiagent learning approaches. Such methods provide a very rich arsenal of conceptual abstractions, formal modelling and specification tools, representation and reasoning methods, and algorithms that can all be used to support coordination (in the sense of “the effective management of interactions”).

Our conceptual framework for social computation is inspired by the literature on network-based modelling of strategically interacting systems [5], as well as that on (multi-agent) rational decision making in sequential stochastic systems [6, 7]. While this connection will be brought to full fruition only in our future focus on reasoning and adaptation, it is crucial to make it from the outset to ensure that our more practical orchestration and implementation methods are designed in adherence to these frameworks. It is also worth noting that our models are much more *data-* (rather than outcome- or strategy-) oriented, which will hopefully make them more directly applicable to the analysis and design of concrete Web applications, rather than requiring various prior steps of conceptual abstraction by a human expert.

Our social orchestration model borrows heavily from the *teamwork* framework [8], in that it replicates its main stages of identifying a goal that cannot be solved by an agent on her own, negotiation and agreement on a plan to achieve this goal, and joint execution of the plan. The feedback stage, and the effect it has on matchmaking is not present in the original framework, and extends it by adaptation capabilities based on user experience. Also, our architecture interprets previous work in this area in a very lightweight fashion: While existing work emphasises the modelling of individual and collective mental states (e.g. intentions and joint intentions) as well as flexible forms of negotiation, planning, and execution monitoring, we are looking for the most basic set of procedures that will enable human-oriented collaboration. On the one hand, this means that we are not (yet) making use of many of the methods the area has to offer. On the other, it drastically reduces the amount of assumptions we have to make regarding the infrastructure, representations, and computational mechanisms agents to bootstrap team activity. This is crucial if we want to move from small sets of elaborate computational agents among which a high degree of *a priori* assumed interoperability to large-scale open-ended Web applications with very diverse populations of participants.

In terms of a computational architecture, our PBD model is somewhat akin to work on electronic institutions [9], which is concerned with specifying the rules by which interactions are managed in a decentralised system, going through various stages of interactions (so called “scenes”, e.g. information exchange, negotiation, etc) where different contributions are possible, and different constraints are applied to obtain the outcomes required for the overall functionality to be realised. A web-centric application of electronic institutions that emphasises shared ontologies and interaction models has been proposed in [10], and provides the inspiration for our (future) aim to develop appropriate ontologies for

playspecs and execution engines for them. Though this work made significant progress in terms of developing advanced methods that help achieve interoperability among heterogeneous components interacting within open computational infrastructures, it adheres to a “big” Web service paradigm, where services essentially interact through remote procedure calls, rely on persistent messaging connections, and hand over direct control to non-local execution engines. Our focus on RESTful services, which are purely driven by asynchronous transitions among the states of local data exposed to other components, takes a very different approach. Here, local computations are handled by genuinely autonomous nodes in the network, and rely on a much more lightweight communication infrastructure.

7.2 Workflow-based systems

Workflow-based systems support the specification, execution and monitoring of complex computations, typically able to combine many web services, or computational elements. The existing workflow systems concentrate on the orchestration of computational services, and offer limited support for human involvement in the execution of workflows.

Generic systems, such as the Business Process Execution Language BPEL⁴, focus on the composition of Web Services for generic enterprise use. Though there have been efforts to use BPEL for scientific research [11], many workflow systems have also been created specifically to support the scientific research process [12, 13], and the tools have evolved along with the communities of users in specific disciplines. These can be used to provide access to services, or to orchestrate the data access and processing nodes needed for large scale computation, such as with Grid computing.

Workflows defined in such languages provide a specification of the tasks and dependancies between them, with most workflow systems providing graphical interfaces to allow domain experts to specify the form of the computation without having to deal with the underlying workflow language. Depending on the particular system, the dependancies in the workflow either represent data-flow between services, or control-flow defining the order of execution of tasks. The nodes in the workflow form a directed acyclic graph which, even though most workflow languages provide constructs to encode conditional execution and looping, still provides only a static description of the shape of the resulting computational system.

In order to support the enactment of a workflow, the workflow system needs to map each node to a specific instance of a service or computational resource. Support for this varies considerably: Mapping may be a manual process, requiring the user to specify each resource explicitly, as in scientific workflow systems such as Taverna [14] and Kepler [15], or it may be handled by the middleware. In the case of BPEL, the mapping may be the result of the workflow specifying services according to an abstract WSDL description, with the workflow engine being responsible for matching these to concrete instances at the point of execution.

In the case of Grid computing systems, mapping involves allocating the compute resources for the individual jobs which make up the workflow. HTCCondor [16] is a Grid

⁴<https://www.oasis-open.org/committees/wsbpel/>

middleware to support High-Throughput Computing. Its DAGMan workflow system allows the composition of individual condor processing jobs into complex sequences of jobs. Each job is represented by a ClassAd (a "classified advert") which describes the details of the compute environment it requires (such as processor type, available RAM, Operating System). The workflow system uses the Condor matchmaker service which compares job ClassAds with those representing the available system resources, to map tasks to specific systems with available compute time.

When supporting the execution of the workflow instance, the workflow system is responsible for arranging for the transfer of data to and from services or processing nodes, and monitoring progress to provide feedback to the user. The system may also record provenance information to ensure a record of the processes which led to the creation of any workflow outputs is available.

Since workflows often comprise many services, or large numbers of long running computations, it is important that workflow systems are able to deal with error conditions in parts of the workflow. Taverna can be made to retry a service if it is unavailable, or alternatives can be specified by the user and it can attempt to use one of these alternative service implementations if the first choice fails or is unavailable. HTCCondor's DAGMan jobs are often run on scavenged compute cycles where machines are otherwise unused, or underutilised. This leaves a job vulnerable to interruption at any point. The scheduler will attempt to checkpoint the job and requeue it to resume later, if that isn't possible then the job will be restarted. If the workflow engine detects errors from which it is unable to recover then the whole workflow will be checkpointed, so that the human user can investigate and potentially resubmit the workflow to carry on from where it left off, without needing to repeat the successfully completed parts.

In contrast to this, our orchestration model and architecture rely on the voluntary contributions of participants, and so far do not address recovery from errors. In fact, we make no explicit distinction between correct executions and error states or exceptions – computations may simply continue from any intermediate point if a play node is able to perform its computations on them, and is free to generate any resource as a consequence of that computation, with no guarantees for the integrity or usefulness of these resources.

7.3 Human-based computation systems

In this final section, we turn toward more concrete proposals for systems that involve *human-based computation* on the Web. These are relevant for our own work, as they address the *hybridity* aspect of HDA-CAS.

Following the expansion of portable computing devices, the traditional service-oriented computing (SOC) moved on to include people as providers of online services. How this is achieved depends mainly on the type of collaborations where human services are needed. Collaborations can range from fully orchestrated ones (*process-centric*) to fully unconstrained (*ad-hoc*) ones. We explore how this choice affects the design decisions using two exemplary systems. Then, we look at *crowdsourcing* systems that are more concerned with aggregating results from repeated execution by large numbers of humans rather than with organising collaboration among individuals. Together, the more collaboration-centric

and more aggregation-centric paradigms address different dimensions of compositionality which, taken together, are at the core of our approach to this theme.

7.3.1 Process-centric Collaboration

*Apache HISE*⁵ is a system implementing WS-HumanTask⁶ specification for process-centric collaborations, such as those described by BPEL4People⁷. Human interaction is modelled through the concepts common to business processes. Humans take specific *roles* (e.g., owner, initiator, stakeholder) with respect to *tasks*. Roles specify the possible actions that (a group of) humans can perform over a task. In order to perform a task, different people with different roles are assigned to the task. The concept of task encapsulates the human service, i.e. human tasks are services “implemented” by people. Each task has two interfaces, one exposing the actual service that the task offers, and the other allowing humans who work on the task to manage it.

In order to support task lifecycle management, the system implements a number of other elements need: a state machine for tracking the task progress, temporal and role-based constraints, GUI rendering of tasks, a number of common, prescribed interaction patterns (escalation, delegation), and notifications.

In terms of orchestration, business processes are executed by an execution engine. When a human task needs to be performed, a WS-HumanTask is created and its lifecycle managed by the state machine. Depending on the current state, predefined roles are invoked to perform necessary actions. Humans can perform delegations and assignments of tasks to other. Humans can notify and be notified through asynchronous messages, and they are offered a GUI and an API for managing the task lifecycle. At the implementation level, tasks are invoked through WSDL interfaces, both synchronously and asynchronously.

The system makes no provisions for discovery or recruitment: Participants/nodes are known in advance and assigned predefined roles are defined to carry out specific actions on a task. Each task is then assigned to (groups of) people fulfilling specific roles.

The advantages of this type of system, similar in spirit to workflow-based systems, is the ability to precisely control the collaboration, and to reuse process models, at the cost of limited flexibility requiring process remodelling in case of collaborative pattern changes. Its reliance on traditional services is similar to that often encountered in the workflow systems literature, and we have already commented above on how we want to follow different architectural principles in our work.

7.3.2 Ad-hoc Collaboration

At the opposite end of the spectrum *Human-Provided Services (HPS) Framework* [17] supports *ad-hoc human collaborations*, i.e. human interactions without a predefined control flow. The framework allows humans to use a high-level editor to specify the interface of the service they intend to provide. The service is then stored in an XML-based service

⁵<http://incubator.apache.org/hise/>

⁶<http://docs.oasis-open.org/ns/bpel4people/ws-humantask/200803>

⁷<http://docs.oasis-open.org/bpel4people/bpel4people-1.1.html>

repository and made available through different interfaces (SOAP, REST) and for different message formats (XML, JSON). A proprietary human task format is used when a request is submitted (so called *task announcement*). Upon submission, the user is presented with a list of potentially matching services. The framework supports both synchronous and asynchronous communication, and manages message delivery and service invocation delays when needed to accommodate the human nature of services. Another powerful feature of the framework is interaction handling, which gives users the option of specifying their own collaboration patterns by providing a set of interaction rules.

Architecturally, the HPS middleware runs a centralised XML-based registry of services, tasks, messages and user profiles, along with different modules for management of service discovery and matching, message routing and handling and interaction management (rules engine). The middleware's functionalities – definition and deployment of services, service discovery and matching and service invocations are exposed through a layer supporting different protocols (SOAP, REST, Atom). On top of it, a number of web-based GUI tools is provided, offering the user a visual facility for importing/specifying service descriptions and interaction rules.

Workers freely contribute their service descriptions into the repository, and, at design time, who will provide the service at invocation time is not specified. When a system user submits a task to the system, the service discovery offers service listings in response via Atom feeds. The services can be matched based on different matching and ranking algorithms (cf. [18, 19]). The requester can further restrict the services he wants to use for task processing by limiting it to a role-based group of service providers.

Service providers also provide rules governing the allowed interaction patterns with the service, which are then imposed by a rule engine at runtime. The framework imposes no restrictions as to what kind of rules can be specified. However, no support for automated service composition is reported.

In terms of synchronisation, the middleware manages conversion and routing of messages to appropriate services. In order to support the inherently unstable availability of human-provided services, the platform manages asynchronous communication between the human service providers and the task requesters by caching messages and delivering them across different devices.

The system makes no provisions for task aggregation or decomposition, all tasks are atomic. The main strength of this system is that it is a general-purpose platform with support for arbitrary collaboration patterns, but as it has not been fully implemented yet its real-world applicability is unclear. We envision that concrete SmartSociety implementations of our orchestration architecture will adopt similar principles in many ways, but enhance them with intelligent composition and analysis mechanisms, which are amiss from this system.

7.3.3 Crowdsourcing Systems

In contrast to collaboration systems, crowdsourcing platforms use large numbers of (mostly unskilled) workers to perform human intelligence tasks. In the following paragraphs, rather than focusing on popular crowdsourcing platforms like Amazon Me-

chanical Turk, Galaxy Zoo etc themselves, we focus on programming frameworks that aid the process of building specific applications using Web-based programming techniques, and thus provide important insights for social orchestration architecture design.

TurKit [20] TurKit is a JavaScript library layered on top of Amazon’s Mechanical Turk, aiming to provide a seamless integration of crowdsourcing into general programming. It does so by introducing a novel programming model (*crash-and-rerun*) designed specifically for conventional microtask-based crowdsourcing platforms. Although a detailed discussion of its programming model is out of scope here, the general principle is important within the context of social orchestration. The crash-and-rerun model implies that the entire orchestration and synchronisation is left to the programmer, who must divide the work into appropriate micro-tasks. When a program is run, the human task results are stored into a database, together with the execution trace, allowing a repetition of a blocked/delayed/unsuccessful human computation with different actors, until the computation is successfully completed (memoization). Each subsequent re-run reuses the stored results of the previously successfully executed human tasks, and offers the unfinished tasks again to the crowd, attracting possibly different workers.

All these properties are directly dependent of the underlying crowdsourcing platform, in this case Amazon’s MTurk. Therefore, the programmers can specify task descriptions, the offered price, and the interfaces exposed to the workers as specified by this underlying platform. The workers then simply decide to accept the task and perform it. The only constraint that a programmer can specify is to explicitly prohibit certain workers to participate in a given computation.

The entire synchronisation and aggregation process is left entirely to the programmer, who needs to implement it on an *ad hoc* basis. TurKit offers a programming primitive allowing to fork a code block for parallel execution and a join primitive to wait for the forked branches to finish. Inter-worker synchronisation is out of the programmer’s reach.

Since the TurKit approach relies on re-offering the same microtasks to the crowd, it inherently implies that the computation task must be decomposable into simple subtasks that can be offered to arbitrary workers, i.e. no matching to individual workers’ capabilities can be performed. Alongside the absence of team composition and inter-worker coordination control, this effectively limits the applicability of the platform to conventional crowdsourcing tasks – tagging, translation, comparisons, preference votes, etc.

An important aspect of the programming model assumed by this system is that it embraces the uncertainty involved in human-based computation: If needed, redundant computations can be easily run, and majority votes can be used for controlling the calculation of results. This is certainly a perspective that is largely overlooked by most agent- and workflow-based systems, and will be one that we will have to consider in future developments of our own social orchestration platform.

Jabberwocky [21] Jabberwocky is a programming framework for human-based computation that is composed of three components:

1. Dormouse – a cross-platform middleware enabling computations on top of different

underlying commercial crowdsourcing platforms. In addition to the bare functionality offered by Mechanical Turk, this layer allows richer user profiles, and integrating social information from different social networks, such as Facebook.

2. ManReduce – a component implementing the novel *ManReduce* programming model. As the name suggests, the model is inspired by the MapReduce model. Programmers are required to break down the task into appropriate *map* and *reduce* steps, each of which can then be performed by a machine or by a set of humans workers.
3. Dog – a high-level, user-friendly procedural level language with a syntax slightly resembling SQL, allowing non-expert users to specify a certain class of problems which then get executed by being translated into the aforementioned ManReduce paradigm.

The computational architecture of the system operates in the following way: A user compiles a high-level script in the Dog language, which then gets translated into a ManReduce program. This program is then executed on the Dormouse platform. The platform creates tasks, as defined by ManReduce. Machine tasks are dispatched for execution through a queue to a cluster of machine compute units or services. People tasks are similarly dispatched in form of JSON descriptors to workers residing on possibly different underlying platforms. Machine execution is suspended (de-queued) until human computation is performed.

In terms of recruitment, at the ManReduce level, it is left to the programmer to specify the mappings from human tasks to workers. However, as opposed to TurKit, Jabberwocky provides more support as the programmer can impose declarative constraints to specify what types of workers are eligible to apply for the task, and this mechanism extends to social relationships, and can be used, for example, to consider only Facebook friends (though the constraint language is far from allowing general relationship constraints). At the Dog level, only a number of predefined constructs can be used for specifying eligible workers and collaboration patterns at a fairly high level, for example, “friends from ‘facebook’ where university=‘MIT’ ” will be asked to perform one of the following actions {‘Vote’, ‘Label’, ‘Compare’, ‘Answer’}”.

Synchronisation and aggregation mechanisms are dictated by the map & reduce variant in use: A number of *map* steps can be performed in sequence, followed by possibly multiple *reduce* steps. Any of these can be performed by human or machine nodes, and human computations are blocking.

The main limitation of this system is that the “MapReduce-style” class of problems is not general enough. On the other hand, the framework makes a good attempt to provide a programming interface that can be used by non-expert users by introducing the Dog language. This is an interesting feature that should be further explored in our project, though it is not central to the aims of the social orchestration and compositionally work.

Automan [22] AutoMan is very similar to the previous systems, but simpler, providing only functionality for crowdsourced multiple-choice question answering in the Scala programming language. Its authors main focus is on automated management of quality and

correctness of answers and on pricing policies. Each question is offered to the crowd in a number of copies. The exact number of copies depend on the desired confidence interval. Once the question is answered a number of times, an automated procedure decides if the answer is correct with respect to a minimum level of confidence, or another round of answering should be performed. New rounds of answering offer better prices, but also exclude previously participating workers from answering again, stimulating the workers to respond correctly the first time.

Although simple, the reason why we chose to include this system in our review is because it demonstrates how *indirect* worker recruitment works: Instead of actively choosing workers to perform a task by selecting them based on their properties, skills or social connections, this approach employs a mechanism offering tasks to the crowd in multiple rounds and adjusting the price in order to attract the workers.

In this sense, it provides a very interesting incentive scheme to solve the recruitment problem which is worth studying for the development of our own recruitment mechanisms. At the same time, it does not cover many of the other stages of social orchestration that we have discussed.

CrowdLang [23] While offering similar functionalities as the systems just describe, such as cross-platform applicability and human result memoization, CrowdLang a number of novel features, primarily with respect to the collaboration synthesis and synchronisation.

CrowdLang enables users to (visually) specify a hybrid machine-human workflow by combining a number of generic collaborative patterns (e.g. iterative, contest, collection, divide-and-conquer), and to generate a number of similar workflows by differently recombining the constituent patterns, in order to generate a more efficient workflow. The use of human workflows also enables indirect encoding of inter-task dependencies.

An IDE is used as interface to users of the systems, which can define tasks and perform workflow recombinations to generate hybrid man-machine workflows. These workflows are orchestrated by the CrowdLang Engine, that also exposes Web Service interfaces, although details are not provided in the cited paper. The engine invokes human tasks through an abstraction layer that supports task deployment on different commercial crowdsourcing platforms, such as MTurk, Clickworker or CrowdFlower.

CrowdLang offers an array of collaborative patterns which can be recombined, in order to enable versatile human-machine workflow compositions. Initial task decomposition and final result aggregation are also represented as collaborative patterns and then executed by the crowd as part of the workflow. Similarly, synchronisation is also achieved by specifying appropriate patterns.

At the time of writing, the system has been evaluated only on a limited scope of tasks, such as text translation, which can be expressed with standardised workflow patterns in a straightforward way. It remains to be seen how applicable this approach can be for more general human computations, and whether we can reuse some of its ideas for (semi-)automated task design. However, it is worth highlighting that CrowdLang is currently the only available crowdsourcing system that allows for directly specifying and combining collaboration patterns, thereby adding elements of the (otherwise missing) collaboration layer on top of conventional crowdsourcing.

7.4 Summary

This section has surveyed important contributions from various areas on issues relating to social orchestration and compositionally. Where possible, we have attempted to explain how these relate to our own approach, and to identify elements of these contributions that have been, or are expected to become, important for our work.

The survey, just like earlier parts of this document, illustrates that our research aims are at the intersection of many research topics, which involve implementation and architectural concerns, algorithmic issues, and fundamental computational problems. It is important to emphasise that we believe the contribution of the workpackage to be precisely at the intersection of all these issues, rather than in attempting to make significant contributions to all of them. The fact that our own work so far does not even attempt to reproduce the features of state-of-the-art systems by using them “off-the-shelf” is indicative of this.

With this respect, the main conclusions from the survey of related work, and the research gaps it has uncovered which seem relevant to the development of next-generation HDA-CAS, are as follows: Firstly, methods are needed that address the requirements of Web-based, voluntary collaboration in open-ended collectives while providing more elaborate methods for task composition and aggregation *without* assuming the heavy machinery that intelligent and flexible coordination methods such as those provided by agent- and workflow-based systems provide. Secondly, we need to develop automated methods that better support humans in designing, executing, and adapting social orchestration systems: our literature review shows that current systems reach their limits fast when more than one of the three main dimensions of complexity (complexity of the process model, number of actors, lack of a priori interoperability) is increased at the same time. Finally, existing contributions appear across largely disconnected research areas and differ very much in terms of their aims and assumptions – combining their methods in the novel context of HDA-CAS will certainly lead to new insights and hopefully benefit all of them.

8 Conclusion

This document has summarised the work done in WP6 within the first year of the SmartSociety project. It describes how we have developed methods for static social orchestration that attempt to be as lightweight as possible, surveys relevant work from the literature, and sets the scene for the work to be done in this workpackage for the remainder of the project. We believe it illustrates that we have made significant progress so far, and that our preliminary results have allowed us to formulate important longer-term research challenges. Whereas the focus of the work has been mostly on laying the conceptual and architectural groundwork for future collaboration (which is important as social orchestration provides the methodological “glue” for the core nscientific workpackages WP2/3/4/5 and their interface to the more implementation-oriented work in WP7 and WP8), this focus will shift toward an investigation of algorithmic methods for intelligent orchestration support, which will contribute crucial adaptation capabilities to the HDA-CAS SmartSociety is endeavouring to build.

References

- [1] J. Hrnčir and M. Rovatsos, “Applying strategic multiagent planning to real-world travel sharing problems,” in *Proceedings of the 7th International Workshop on Agents in Traffic and Transportation (ATT 2012)*, Valencia, Spain, June 5, 2012.
- [2] M. Wooldridge, *An Introduction to Multiagent Systems, 2nd edition*. Chichester, England: John Wiley & Sons, 2009.
- [3] Y. Shoham and K. Leyton-Brown, *Multiagent Systems – Algorithmic, Game-Theoretic, and Logical Foundations*. Cambridge University Press, 2009.
- [4] G. Weiß, Ed., *Multiagent Systems. A Modern Approach to Distributed Artificial Intelligence*. Cambridge, MA: The MIT Press, 1999.
- [5] E. David and K. Jon, *Networks, Crowds, and Markets: Reasoning About a Highly Connected World*. New York, NY, USA: Cambridge University Press, 2010.
- [6] R. Sutton and A. Barto, *Reinforcement Learning. An Introduction*. Cambridge, MA: The MIT Press/A Bradford Book, 1998.
- [7] C. Boutilier, “Sequential Optimality and Coordination in Multiagent Systems.” in *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI-99)*, Stockholm, Sweden, 1999.
- [8] D. V. Pynadath and M. Tambe, “An Automated Teamwork Infrastructure for Heterogeneous Software Agents and Humans,” *Autonomous Agents and Multi-Agent Systems*, vol. 7, pp. 71–100, 2003.
- [9] M. Esteva, B. Rosell, J. A. Rodríguez-Aguilar, and J. L. Arcos, “Ameli: An Agent-Based Middleware for Electronic Institutions,” in *Proceedings of the 3rd International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2004)*, N. Jennings, C. Sierra, L. Sonenberg, and M. Tambe, Eds., 2004, pp. 236–243.
- [10] D. Robertson, C. Walton, A. Barker, P. Besana, Y.-H. Chen-Burger, F. Hassan, D. Lambert, G. Li, J. McGinnis, N. Osman, A. Bundy, F. McNeill, F. van Harmelen, C. Sierra, and F. Giunchiglia, “Interaction as a grounding for peer to peer knowledge sharing,” in *Advances in Web Semantics*, vol. 1, 2007.
- [11] K. L. L. Tan and K. J. Turner, “Orchestrating grid services using bpel and globus toolkit 4,” 2006.
- [12] E. Deelman, D. Gannon, M. Shields, and I. Taylor, “Workflows and e-science: An overview of workflow system features and capabilities,” *Future Gener. Comput. Syst.*, vol. 25, no. 5, pp. 528–540, May 2009. [Online]. Available: <http://dx.doi.org/10.1016/j.future.2008.06.012>
- [13] V. Curcin and M. Ghanem, “Scientific workflow systems - can one size fit all?” in *Biomedical Engineering Conference, 2008. CIBEC 2008. Cairo International*, 2008, pp. 1–9.

- [14] K. Wolstencroft, R. Haines, D. Fellows, A. Williams, D. Withers, S. Owen, S. Soiland-Reyes, I. Dunlop, A. Nenadic, P. Fisher, J. Bhagat, K. Belhajjame, F. Bacall, A. Hardisty, A. Nieva de la Hidalga, M. P. Balcazar Vargas, S. Sufi, and C. Goble, "The taverna workflow suite: designing and executing workflows of web services on the desktop, web or in the cloud," *Nucleic Acids Research*, vol. 41, no. W1, pp. W557–W561, 2013. [Online]. Available: <http://nar.oxfordjournals.org/content/41/W1/W557.abstract>
- [15] I. Altintas, C. Berkley, E. Jaeger, M. Jones, B. Ludascher, and S. Mock, "Kepler: An extensible system for design and execution of scientific workflows," in *Proceedings of the 16th International Conference on Scientific and Statistical Database Management*, ser. SSDBM '04. Washington, DC, USA: IEEE Computer Society, 2004, pp. 423–. [Online]. Available: <http://dx.doi.org/10.1109/SSDBM.2004.44>
- [16] D. Thain, T. Tannenbaum, and M. Livny, "Distributed computing in practice: the condor experience." *Concurrency - Practice and Experience*, vol. 17, no. 2-4, pp. 323–356, 2005.
- [17] D. Schall, H.-l. Truong, and S. Dustdar, *Socially Enhanced Services Computing*, S. Dustdar, D. Schall, F. Skopik, L. Juszczuk, and H. Psaiar, Eds. Vienna: Springer Vienna, 2011. [Online]. Available: <http://link.springer.com/10.1007/978-3-7091-0813-0>
- [18] D. Schall, "Dynamic Context-Sensitive PageRank for Expertise Mining," in *Proceedings of the Second international conference on Social informatics SocInfo'10*, ser. Lecture Notes in Computer Science, L. Bolc, M. Makowski, and A. Wierzbicki, Eds., vol. 6430. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 160–175. [Online]. Available: <http://www.springerlink.com/index/10.1007/978-3-642-16567-2>
- [19] D. Schall, F. Skopik, and S. Dustdar, "Expert Discovery and Interactions in Mixed Service-Oriented Systems," *IEEE Transactions on Services Computing*, vol. 5, no. 2, pp. 233–245, Apr. 2012. [Online]. Available: <http://doi.ieeecomputersociety.org/10.1109/TSC.2011.2http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5710867>
- [20] G. Little, L. B. Chilton, R. Miller, and M. Goldman, *TurKit: Tools for iterative tasks on mechanical turk*. IEEE, Sep. 2009. [Online]. Available: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5295247>
- [21] S. Ahmad, A. Battle, Z. Malkani, and S. Kamvar, "The jabberwocky programming environment for structured social computing," *Proceedings of the 24th annual ACM symposium on User interface software and technology - UIST '11*, p. 53, 2011. [Online]. Available: <http://dl.acm.org/citation.cfm?doid=2047196.2047203>
- [22] D. W. Barowy, C. Curtsinger, E. D. Berger, and A. McGregor, "Automan: A platform for integrating human-based and digital computation," pp. 639–654, 2012. [Online]. Available: <http://doi.acm.org/10.1145/2384616.2384663>

- [23] P. Minder and A. Bernstein, “How to translate a book within an hour: towards general purpose programmable human computers with crowdlang,” *Proceedings of the 3rd Annual ACM Web*, no. June, 2012. [Online]. Available: <http://dl.acm.org/citation.cfm?id=2380745>