Programming Incentives in Information Systems

Ognjen Scekic, Hong-Linh Truong, and Schahram Dustdar

Distributed Systems Group, Vienna University of Technology
{oscekic,truong,dustdar}@dsg.tuwien.ac.at
http://dsg.tuwien.ac.at

Abstract. Information systems are becoming ever more reliant on different forms of social computing, employing individuals, crowds or assembled teams of professionals. With humans as first-class elements, the success of such systems depends heavily on how well we can motivate people to act in a planned fashion. Incentives are an important part of human resource management, manifesting selective and motivating effects. However, support for defining and executing incentives in today's information systems is underdeveloped, often being limited to simple, per-task cash rewards. Furthermore, no systematic approach to program incentive functionalities for this type of platforms exists.
In this paper we present fundamental elements of a framework for programmable incentive management in information systems. These elements form the basis necessary to support modeling, programming, and execution of various incentive mechanisms. They can be integrated with different underlying systems, promoting portability and reuse of proven incentive strategies. We carry out a functional design evaluation by illustrating modeling and composing capabilities of a prototype implementation on realistic incentive scenarios.

Keywords: rewards, incentives, social computing, crowdsourcing

1 Introduction

Most 'traditional' incentive mechanisms used by companies today [1] have been developed for static business processes, where the actors are legally bound to the company (employees, workers), placed under human management and assigned to specific workflow activities, usually for a longer time period. Such setting allows direct monitoring of workers and subsequent direct application and adaptation of incentive mechanisms.

However, with the advent of novel, web-scale collaborative systems and collaborative patterns, starting from crowdsourcing, and moving towards the ever more complex socio-technical collaborative systems[1], we witnessed the actors in the system become numerous, anonymous and interchangeable with machines. They now engage with the system only occasionally, irregularly and for a short

[1] For example, Social Compute Units [2] and Collective Adaptive Systems (http://www.smart-society-project.eu)

time, performing different tasks with variable duration and quality. This means that most existing incentives, relying on the conventional understanding of the notions of career, promotion and working hours cannot effectively support this new type of collaborations.

In [3] we proposed adapting conventional incentive mechanisms to the new collaboration forms, and providing *programmable incentive management* functionalities to the information systems supporting them. In this paper we present some of the fundamental building elements of a framework for programmable incentive management in information systems – PRINC. These elements allow modeling, programming, execution, monitoring and reuse of various incentive mechanisms on top of existing workforce management and collaboration platforms. We carry out a functional design evaluation by illustrating modeling and composing capabilities of a prototype implementation on realistic incentive scenarios.

The paper is structured as follows. Related and background work is presented in Section 2. Section 3 introduces the main functionalities and the overall architecture of the PRINC framework. Section 4 discusses in more detail individual framework components and their design considerations. Section 5 evaluates the functionality and usefulness of the presented design by encoding an exemplary, realistic incentive strategy. Section 6 concludes the paper and presents the direction of our future work.

2 Related Work & Background

2.1 Related Work

Most related work in the general area of rewarding and incentives originates from economics, game theory, organizational science and psychology. Incentives are the principal mechanism for aligning interests of business owners and workers. As a single incentive always targets a specific behavior and induces unwanted responses from workers [4], multiple incentives are usually combined to counteract the dysfunctional behavior and produce wanted results. Opportunities for dysfunctional behavior increase with the complexity of labor, and so does the need to use and combine multiple incentives. The principal economic theory treating incentives today is the *Agency Theory* [4,5]. The paper [1] presents a comprehensive review and comparison of different incentive strategies in traditional businesses.

Only a limited number of computer science papers treat these topics, and usually within particular, application-specific contexts, like peer-to-peer networks, agent-based systems and human-labor platforms (e.g., Amazon Mechanical Turk). In [6] the aim is to maximize p2p content sharing. In [7] the authors seek to maximize the extension of social network by motivating people to invite others to visit more content. In [8] the authors try to determine quality of crowdsourced work when a task is done iteratively compared to when it is done in parallel. In [9] the authors investigate how different monetary rewards influence the productivity of mTurkers. In [10] the authors analyze two commonly

used approaches to detect cheating and properly validate submitted tasks on popular crowdsourcing platforms. A detailed overview of incentive and rewarding practices in social computing today can be found in [3,11]. The key finding is that incentives in use in today's social computing platforms are mostly limited to simple piece-rates that may be suited for simple task processing, but are inappropriate for the more advanced collaborative efforts. All these studies show that, depending on the environment, there always exist types of incentives that can provide the necessary motivation, and that incentive composition is the key to a successful local application of general incentive practices.

In contrast to the described work, which focuses on specific application scenarios, we propose developing general models and techniques for programmable incentive management. To the best of our knowledge, there exist no other similar comprehensive approaches.

2.2 Background

The work presented in this paper is part of the ongoing effort to conceptualize a general approach to model and encode most of the incentive mechanisms for use in social computing environment today [3]. Currently, for every social computing system a context-specific, tailored incentive functionality is developed anew. This is a clear overhead, as most incentive strategies could be composed out of a limited number of reusable, basic elements and then tweaked to fit a specific application area. In particular, this paper builds upon the important concepts introduced in [12].

The *Rewarding Model* (**RMod**) is a low-level, abstract model for representing the state of a social computing system, allowing composition and execution of various incentive mechanisms. RMod reflects the quantitative, temporal and structural aspects of an external social computing system. The execution of incentives implies changing the internal state of the RMod through programmed application of rewards. Due to space constraints, the formal definition of the RMod is provided as supplement[2].

Workers are represented as nodes of a graph representation in RMod. *Relations* connect the worker nodes and are associated with a set of tags that determine their types. Relation types are used for structural modifications. Each worker contains a set of associated local attributes (quantitative data). The attributes represent performance and interactions metrics. A set of system-level quantitative data also needs to be stored.

Time management in our framework is expressed through the notions of *timeline*, *clock ticks*, *iterations*, and past and future *events* [12]. The timeline is a concept providing a unified time management functionality in the model. It is in charge of producing clock ticks, delimiting iterations, storing past events, scheduling, canceling, and reordering future events. Clock tick is the basic time measurement unit. They have a fixed, predefined duration time. They are used to express iteration lengths. Iterations have variable duration and are used to

[2] `http://tinyurl.com/princ-caise2013`

model various productive cycles in real-world environments (e.g., working days, project phases, software development cycles).

An *event* contains the scheduled execution time, execution priority and a *query*. The query contains the logic necessary to perform an incentive condition evaluation or rewarding action. This implies reading and/or modifying global and worker attribute data (quantitative aspect), past data and future scheduled events (temporal aspect) or the current node structure (structural aspect). Technically, it means running database queries, graph matching/transformation queries or logical expression evaluation. For reusability purposes queries should be composable.

The RMod state can be changed in three ways: a) through the execution of events (queries) performing rewarding actions; b) through the execution of events notifying of externally-originated changes (e.g., the arrival of new employee, cancellation of project); and c) through direct manipulation.

3 PRINC Framework

3.1 Requirements

The **PRINC** (***PR****ogrammable* ***INC****entives*) framework aims to provide an end-to-end solution for programmable incentive management. This includes formal specification, automatic deployment and runtime management of *incentive mechanisms* [3] in information systems. The core idea behind PRINC is to enable translation of system-independent incentives, such as the following:

"Give reward R to the workers who performed better than the average of their collaborators in the past month." or *"If the overall team effort does not increase to the satisfactory level in the next quarter replace the team leader with the best performing subordinate worker."*

into concrete instantiations depending on the context-specific definitions of the notions of: "reward R", "performance", "effort level" and "satisfactory effort level". This means that entire incentive strategies can be specified in a system-independent fashion and then automatically deployed by the framework on particular information systems. This approach promotes the reuse of proven incentive strategies and lowers the risks.

The major requirements for the framework, therefore, include the development of the following components:

1. A model and a human-friendly notation for composing declarative, portable descriptions of *incentive mechanisms*.
2. A model to represent incentive mechanisms through imperative *rewarding actions*.
3. A mapping model for instantiating system-specific rewarding actions out of generic variants.
4. An extensible API allowing the system to communicate with different underlying social computing systems.

5. Automated translation of generic incentive strategies into system-specific rewarding actions.
6. Functionality to execute and monitor rewarding actions.

In this paper we focus primarily on the requirements 2-4, and present the design and evaluation of components necessary to support them.

3.2 Architecture Overview

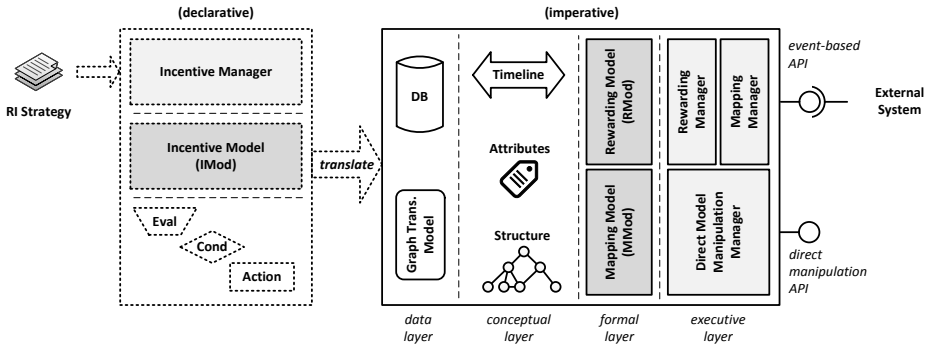

Fig. 1. The PRINC Framework

Figure 1 shows the architecture of the PRINC framework.

The *Incentive Model* (**IMod**) encodes the declarative, portable version of the strategy. The *Rewarding Model* (**RMod**) encodes the imperative, system-specific version of the strategy. It constantly mirrors the state of the external system and executes incentive mechanisms on it. The *Incentive Manager* translates the IMod version into the RMod form. The *Mapping Model* (**MMod**) defines the mappings needed to properly interpret the system-independent version of the strategy in the context of a specific social computing platform (*external system*). The mapping itself is performed by the *Mapping Manager*.

The execution of an incentive strategy implies executing a number of incentive mechanisms. This is done by scheduling a number of future rewarding actions to be executed as events over RMod. Execution of rewarding actions modifies the internal state of RMod, which is then propagated to the external system. At the same time, the state of RMod can be changed via events originating from the external system. The *Rewarding Manager* implements the RMod (Section 2.2), performs and interleaves all event-based operations on RMod and ensures its consistency and integrity (e.g., by rejecting disallowed structural modifications or preventing modification of the records of past behavior).

The *Direct Model Manipulation Manager* (*D3M*) provides direct RMod manipulation functionalities without relying on the event mechanism and without enforcing any consistency checks. The direct access to the RMod is needed for

offering the necessary functionalities internally within PRINC, but also to allow more efficient monitoring and testing. D3M is therefore used to load initial state of the system, and to save snapshots of the system's current state.

The communication between PRINC and the external system is two-way and message-based. The external system continuously feeds the framework with the necessary worker performance data and state changes and receives rewarding action notifications from PRINC. For example, PRINC may notify the external system that a worker earned a bonus, suggest a promotion or a punishment. Similarly, it may need to send an admonition message to the worker, or display him/her a motivating visual information (e.g., rankings). The external system ultimately decides which notifications to conform to and which to discard, and reports this decision back in order to allow keeping the RMod in consistent state.

3.3 Intended Usage

PRINC will allow companies and organizations (clients) managing existing and future social computing platforms make use of programmable incentive management and monitoring by integrating PRINC with their platforms. After having provided appropriate message mappings and context-specific metrics (Section 4.2), client's platform just needs to inform PRINC of relevant state changes and receive in exchange suggestions of which concrete incentive actions to execute over workers.

Clients can assemble and adapt incentive strategies suitable for their particular contexts out of a number of existing incentive mechanisms known to be effective for the same class of collaboration patterns, thus cutting the risks and lowering the overall costs. An incentive strategy will be composed/edited in a human-friendly notation that we intend to develop (Requirement 1 in Section 3.1). Entire strategies or particular mechanisms can be publicly shared or commercially tailored by experts in the field. Incentive management can also be offered as a service, which can be of particular use to crowd-based SMEs unable to invest a lot of time and money in setting up a full incentive scheme from scratch.

4 Design Considerations

4.1 Rewarding Model (RMod)

The structural aspect of RMod's state is reflected through a typed graph, containing nodes representing workers and edges representing relations (Section 2.2). Structural modifications are performed by applying graph transformations (graph rewriting) [13]. Examples of graph transformations can also be found in [13]. Similar transformations are used to achieve relation rewiring between workers in RMod, and thus represent various structural incentives (e.g., promotion, change of team structure and collaboration flow). The temporal aspect of RMod's state is maintained by the `Timeline` class. It is a façade class offering complete iteration and event handling functionality. A database is used for storing and querying

past events and iterations, and a separate in-memory structure for keeping and manipulating future events. Clock ticks can be internally generated or dictated by the external system. If internally generated, the duration is specified in MMod (Section 4.2) to best fit the necessities of the external system.

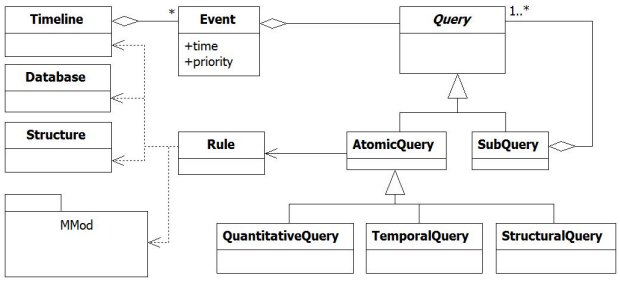

Fig. 2. Simplified UML diagram showing principal components of the event mechanism.

`Event`s can originate from the external system, or be generated by the incentive logic. A `Rule` is the basic (atomic) piece of incentive logic in our system. In the PRINC framework, the rewarding rules would be products of the automated translation of an IMod-encoded incentive strategy with applied mappings from the MMod. Rules are used as reusable components that can be executed independently (wrapped into an `AtomicQuery`) or used to build more complex `Query`-ies(Figure 2). A rule/query execution results in either computation and returning of a result (a non-modifying evaluation), or generation of (multiple) `Event` objects for later execution. A rule can be marked for execution either at each clock tick and/or upon receiving a message coming from the external system. If a rule execution modifies the model's state, not only the change itself, but the fact that this particular rule was executed is also recorded into the database for monitoring end evaluation purposes.

4.2 Mapping Model (MMod)

In order for our framework to couple with an external system (e.g., of a company) we need to provide a number of mappings that enable the application of generic incentive strategies within this company's particular context. These mappings are provided through the Mapping Model.

PRINC supports dynamic mappings. This means that the parameter values can be modified during the runtime to allow a dynamic adaptation of the incentive strategy. The company can easily and dynamically switch between different basic incentive mechanisms, evaluation methods, rewarding actions and metrics. Individual mechanisms can easily be turned on and off, giving the company a flexible, composite incentive strategy.

The functionalities of MMod include:

– *Definition of system-specific artifacts, actions, attributes and relation types.* These definitions inform PRINC of the unique names and types of different company-specific *artifacts*, *actions*, attributes and relation types that need to be stored and represented in PRINC for subsequent reasoning over conditions for applying rewards. Actions represent different events happening in the external system. Artifacts represent objects of the actions (Section 4.3). For example, a design company may want to define an artifact to represent the various graphical items that its users produce during design contests, and an action to denote the act of submitting a design artifact or evaluation.

– *Definition and parameterization of metrics, structural patterns and incentive mechanisms.*
Metrics are attributes that are calculated by PRINC from other attributes provided by the external system. They are used to express different performance aspects of individuals or groups of workers. For example, a context-independent incentive strategy may rely on worker's *trust* metric in a reward application condition. However, for different companies, the trust metric is calculated differently. For example, the trust of a worker may depend on the percentage of the peer-approved tasks in the past (as in Section 5.2), or it may involve a calculation based on trust values of nearest neighbors.
PRINC offers a number of well-known, predefined metrics that cover many real-world application scenarios as library functions (e.g., trust, productivity, effort), thus cutting the time needed to adapt a generic incentive strategy to a particular scenario. Predefined metrics then only need to be parameterized. For example, in case of trust calculation, our client only needs to choose one of the predefined trust metric calculations and provide some attributes and parameters as inputs to tweak the calculation to his needs (e.g., what is the time interval used for the calculation).
In cases where a library metric definition is unable to express a system-specific aspect, clients can provide their own definition. This is usually the case with company-specific *predicates*, which we can define in MMod. One common use of predicates is to define criteria of team membership. A criterion can be structural (e.g., all workers managed by 'John Doe'), logical (e.g., workers with the title 'Senior Java developer'), temporal (e.g., workers active in the past week) or composite (e.g.,'Senior Java developers' active in the past week).
In the same way, predefined structural patterns and entire incentive mechanisms can be parameterized in MMod. For example, the library pattern `COLLABORATORS (Worker W, RelationType RT, Weight w)` returns for a given `Worker` node `W` a set of workers that are connected with `W` via `RT`-typed relations, having the weight greater than `w`, where `w` is a client-provided value. In case a translated incentive strategy relied on using this library pattern, the client could be asked to provide only a value for `w`, while PRINC would initialize the other parameters during the execution.

– *Message mappings.*

In case a condition for performing a rewarding action is fulfilled, PRINC needs to inform the external system. For each rewarding action we need to specify the type of message(s) used to inform the external system and the data they will contain. The data contained may include metric values to be used as a justification for executing a reward/punishment, or a structural pattern suggesting a structural transformation to the external system. Also, we need to specify which messages PRINC expects to get as an answer to the suggested action. Only in case of a positive answer will PRINC proceed to update its internal model. Otherwise, the rewarding action is ignored.

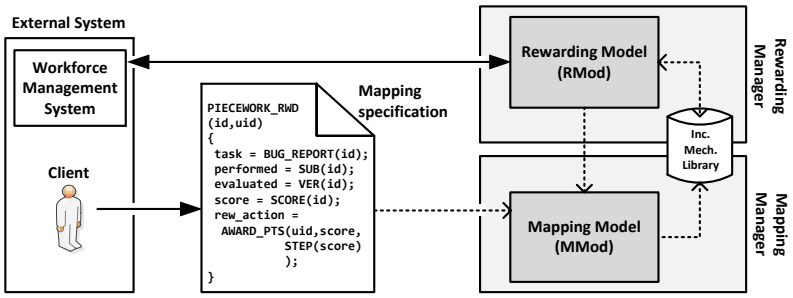

Fig. 3. Adapting a general piece-work incentive mechanism for software testing company use-case.

Example. A software testing company wants to setup quickly an incentive mechanism that awards every bug submitter a certain number of points for every verified bug. The amount of points assigned is company-specific and depends on bug severity. There is a number of real crowdsourcing companies that rely on such mechanisms (e.g., translation companies and design companies).

A pre-designed library incentive mechanism `PIECEWORK_RWD(···)` works with the concept of a 'task'. Once the task is 'performed', an 'evaluation' process on its quality is started. The evaluation phase ends with obtaining a 'score'. The 'rewarding action' is then executed if a predicate taking the evaluation score as one of its input parameters returns true.

In this particular case the testing company can define an artifact named `BUG_REPORT` to represent a bug report in our system, containing a bug ID, severity, and other fields. The act of submitting a bug report can be defined as the `SUB(id)` action, the act of verifying a bug report as the `VER(id)` action. What is left to do is to simply map these actions, artifacts and metrics to the incentive mechanism parameters (Figure 3). In this case, the concept of 'task' is mapped to the `BUG_REPORT` artifact. Performing of the task is signaled by a message containing the `SUB(id)` action. The voting phase ends with the arrival of the

`VER(id)` action. From then on, the corresponding score can be accessed as the metric `SCORE(id)`.

Assignment of rewards to the bug submitters can also be automatically handled by one of the library rewarding actions we indicate in the mapping. For example, the action `AWARD_PTS(userID, score, mappingFunction(score))` simply informs the company's system of how many points the user should be awarded, based on his artifact's score and a mapping function. The mapping function in this case can be a step function or a piecewise-linear function, both available as library implementations.

4.3 Interaction Interfaces

The framework provides two APIs for manipulation of the internal state: a) An API for direct manipulation of RMod and MMod (*DMMI*); and b) A message API for event-based RMod manipulation (*MSGI*), meant for the external system.

DMMI is intended for internal use within the PRINC framework. This API exposes directly the functionalities which are not supposed to be used during the normal operation of the framework since the consistency of the model's state cannot be guaranteed. External use should therefore be limited to handling uncommon situations or performing monitoring. MSGI is intended for exchange of notifications about external system state changes or suggested rewarding actions. (Un-)marshalling and interpreting of messages is handled by the Rewarding Manager. The functionalities offered by the APIs are summarized in Table 1. Abstract representation of the message format is shown in Figure 4. This format can be used for both incoming and outgoing messages.

The *Action* defines the message identifier, type, timestamp and importance. In case of an incoming message, the type can represent the following: (a) A system-specific activity that needs to be recorded (e.g., task completion, sick leave) for later evaluation; (b) Update of an attribute (e.g., hourly wage offered); or (c) Update of the worker/team structure.

The *Artifact* specifies the object of the action. It contains the new value of the object that needs to be communicated to the other party. In case of an incoming message, the *Artifact* can correspond to: (a) an activity notification (expressed as

API	Functionality	Description
MSGI	State updates	Notify framework of external structural/temporal/attribute changes.
	Rewarding	Suggest a rewarding action to the external system.
	Notifications	Mutually exchange artifact, action and attribute updates/events.
DMMI	Database API	Manipulate DB records. Execute DB scripts.
	Rules API	Directly execute RMod rules and queries.
	Timeline API	Modify past and future iteration parameters.
	Structure API	Directly perform graph transformations.
	Mappings	Change mappings in runtime (dynamically).

Table 1. Functionalities exposed through the APIs

an artifact defined in MMod); (b) an attribute update; (c) a structural update; or (d) an iteration update. In case of an outgoing message, the artifact can correspond to: (a) an activity notification; (b) a metric update; or (c) a rewarding action notification.

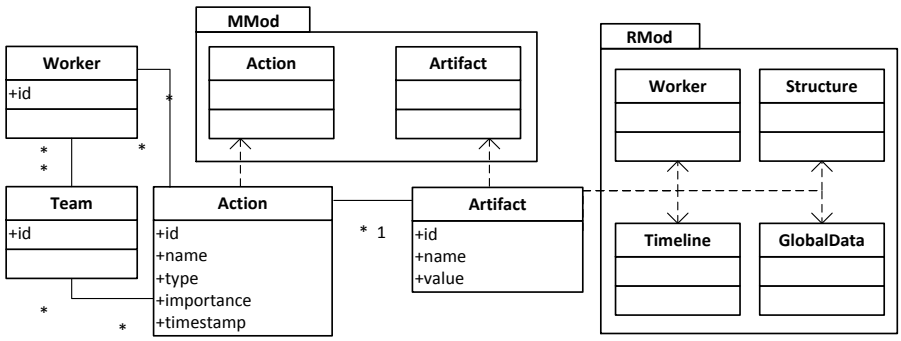

Fig. 4. Abstract representation of the MSGI message format.

Structural updates can be expressed either as library-defined structural modification patterns or as completely new descriptions of the graph (sub)structure defined in a formal language. Iteration updates notify the system of the (re-)scheduling of future iterations and the duration changes of the currently active ones. *Worker* and *Team* parts of the message specify the workers the message applies to. As already explained, the team identifiers are defined in MMod and serve to target all individual workers fulfilling a condition, or as a simple shorthand notation.

5 Prototype & Evaluation

5.1 Prototype Implementation

The prototype we implemented consists of the framework components we presented in Section 4. The same components are framed with full borderlines in Figure 1. Components outlined with dotted lines will be the subject of our future work. The current implementation is capable of expressing and executing only imperative incentive mechanisms.

The prototype was implemented in C#, using Microsoft SQL Server database. Structural modifications are performed using the GrGen.NET [14] library. GrGen is a versatile framework for performing algebraic graph transformations, including a graph manipulation library and a domain-specific language for specifying declarative graph pattern matching and rewriting. At this point, we use a number of pre-compiled graph transformation patterns, which are able to capture structural requirements of the incentive mechanisms we intend to support.

Structural incentives [3] will be subject of our future work, as it goes beyond the scope of this paper.

The prototype uses the imperative rewarding rules and MMod mappings provided by the user via initialization scripts. At the moment, they are specified as C# code. This obviously makes sense only for a proof-of-concept purpose. We plan to develop the mapping notation and the domain-specific language that the clients will use to encode new incentive strategies or parameterize existing ones. The implemented message-based API supports binary or XML messages, following the format presented in Section 4.3.

5.2 Evaluation

The elements of the PRINC framework we presented deal with the low-level, imperative techniques for modeling incentive mechanisms. The goal of this evaluation is to show that these techniques are expressive enough to model the functional capabilities of some typical, real-world incentive strategies, allowing us to use them as the foundation for the rest of the framework. It is important to clarify that our intention is not to invent novel incentive mechanisms, nor to compare or improve existing ones.

A broad overview of the incentive strategies used in social computing today is presented in [3]. Out of the strategies presented there, we decided to model and implement a slightly simplified version of the strategy used by the company *Locationary*[3]. The reason for choosing this particular company was primarily because their strategy is a very good example to demonstrate how a number of simple incentive mechanisms targeting different behaviors can be combined into one effective strategy.

Locationary's incentive strategy. Locationary is a company that sells access to a global business directory. In order to have a competitive advantage over a number of companies already offering traditional and internet business directories they need to maximize the number, accuracy and freshness of their entries. For this reason, they need to incentivize users spread around the world to add and actively update local business data. They combine a number of incentive mechanisms in their strategy. The three most important ones are the *conditional pay-per-performance/piece-rate* (or 'lottery tickets' as they name it), the *team-based compensation* (based on the 'shares' of added companies), and the *deferred compensation*, based on the trust scores of the users.

Tickets are used to enter users into occasional cash prize draws. Chances of winning are proportional to the number of tickets held. Tickets are not tied to any particular company. Users are given different ticket amounts for adding, editing or verifying different directory entry fields. Ticket amounts assigned depend on the value of that field to the company. For example, a street view URL is more valuable than the URL of the web page of the place. Similarly, fixing incorrect

[3] `http://www.locationary.com/`

data from other users is also highly appreciated. This mechanism incentivizes the increased activity of the users, but also motivates the users to cheat, as some people will start inputting invalid entries.

This is where the deferred compensation comes into play. The users are only allowed to enter the prize draws if they collected enough tickets (*quota system*) and if their trust score is high enough. The trust metric plays a crucial role. Trust is proportional to the percentage of the approved entries, and this metrics discourages users to cheat. The entries can be approved or disapproved only by other highly trusted users (an example of *peer evaluation*). Trusted users are motivated to perform validation tasks by getting more lottery tickets than they would get for adding/editing fields. On the other hand, cheaters are further punished by subtraction of lottery tickets for every incorrect data field they provided.

The strategy described so far does a good job of attracting a high number of entries and keeping them fresh and accurate. However, it does not discriminate between the directory entries themselves. That means that it equally motivates users to enter information on an insignificant local grocery store, as it motivates them to enter information on a high-profile company. As Locationary relies on advertising revenues, that means that an additional incentive mechanism that attracts higher numbers of profitable entries needs to be included on top of the strategy described so far. The team-based compensation plays this role. Locationary shares 50% of the revenues originating from a company with the users holding 'shares' of that company. Shares are given to the people who are first to add information on a company. Again, cashing out is allowed only to the trusted users.

This example shows how a composite incentive strategy was assembled to fit the needs of a particular company. However, its constituent incentive mechanisms (piece-rates, quotas, peer evaluation, trust, deferred compensation, team-based compensation) are well-known and general [3,11]. A different combination of the same mechanisms could yield a different strategy, optimized for another company.

Implementing Locationary's incentive strategy with PRINC. In Section 4.2 we showed how a general rewarding mechanism for piece-work can be adapted to fit the needs of a software testing company. Here we used the same mechanism to reward workers with lottery tickets, and the same rewarding action `AWARD_PTS(···)` to simulate cash payouts.

A lottery is a frequently used mechanism when the per-action compensation amount is too low to motivate users due to a high number of incentivized actions. Listing 1.1 shows the pseudo-code declaration of a general lottery mechanism we implemented as part of our incentive mechanism library. In order to use this mechanism, we simply need to parameterize the general mechanism by providing the necessary mappings (values, metrics, actions and predicates), as shown in Listing 1.2. Once the incentive strategy is running, we can easily adapt it by changing which metrics, predicates and actions map to it.

```
% Library definitions in RMod
interface T_LOTTERY_TCKT    % Predefined artifact interface.
{
  id;
  uid;                      % Owner ID.
  value = 1;                % Ticket value. Default is 1.
}

LOTTERY                     % Predefined (library) incentive mechanism.
{
  id;                       % Auto-generated, or assigned during the runtime.
  tickets[];                % Collection of T_LOTTERY_TCKT objects.
  ...
  type;                     % To choose from various sub-types.
  timing;                   % Periodic, conditional or externally-triggered.
  numberOfDraws;            % How many tickets should be drawn.
  external_trigger;         % User-declared action triggering a lottery draw.
  ticketType;               % User-defined artifact that represents a ticket.
                            % Must be derived from the predefined
                            % T_LOTTERY_TCKT interface.

  rew_action;               % Action to execute upon each owner
                            % of a winning ticket.

  prize_calculation;        % Metric used to calculate the total reward
                            % amount for a draw. Usually proportional
                            % to the number of the tickets in the draw.

  entrance_cond;            % Predicate used to evaluate whether a ticket
                            % is allowed to enter the draw.
  ...
}
```

Listing 1.1. Definitions of library incentives.

This example also shows how we can combine different incentive mechanisms. For example, the predicate that controls user's participation in a lottery draw requires the user to possess a certain quota of tickets. The threshold is managed by another parameterized incentive mechanism, namely `LOTTERY_QUOTA`. To express trust we use one of the predefined metrics. The remaining mechanisms are similarly implemented, demonstrating that our approach is capable of functionally modeling realistic incentive strategies.

6 Conclusions and Future Work

In this paper we introduced foundational models and techniques for supporting programming of incentives in a dynamic and flexible fashion. These elements represent the building blocks of the envisioned PRINC framework, intended to provide an end-to-end solution for programmable incentive management in information systems. The implemented part of the PRINC framework was functionally evaluated to demonstrate its capability of encoding real-life incentive strategies. Our approach supports platform portability, while enabling dynamic incentive composition, adaptation, and deployment.

In the future we will focus on extending our design with new incentive mechanisms, with a special focus on structural mechanisms. We are also developing

```
% User definitions in MMod
action RUN_LOTTERY(int id);
artifact LOCATIONARY_TICKET extends T_LOTTERY_TCKT {...};

metric CALC_PRIZE(int id, float prizePerTicket)              % A user-defined
{                                                            % metric.
  LOTTERY L = getLottery(id);
  return prizePerTicket * L.tickets.count;
}

predicate ENTER_LOTTERY_PREDICATE(int lotteryId, int userId) % User defined
{                                                            % predicate.

  return TRUST(userID) > 0.65    &&                          % Trust and
         LOTTERY_QUOTA(userId);                              % lottery quota
}                                                            % are library
                                                             % elements.

% User mappings in MMod
LOCATIONARY_LOTTERY = LOTTERY                                % Parameterizing
   {                                                         % a general inc.
     ...                                                     % mechanism.
     timing = "triggered";
     numberOfDraws = 1;
     external_trigger  = RUN_LOTTERY;
     ticketType = LOCATIONARY_TCKT;
     rew_action = AWARD_PTS(ticket.uid, amount, amount);     % Previously
                                                             % explained.

     prize_calculation = CALC_PRIZE(id, 0.0025);             % Here we use a
                                                             % custom metric.
     entrance_cond = ENTER_LOTTERY_PREDICATE(id, ticket.uid);
   }
```

Listing 1.2. Defining customized incentive mechanisms with library elements.

a simulation framework to allow us better testing the incentive composability. The following step will be the work on the declarative domain-specific language for expressing incentive strategies.

References

1. Prendergast, C.: The provision of incentives in firms. Journal of economic literature **37**(1) (1999) 7–63
2. Dustdar, S., Bhattacharya, K.: The Social Compute Unit. Internet Computing, IEEE **15**(3) (2011) 64–69
3. Scekic, O., Truong, H.L., Dustdar, S.: Incentives and rewarding in social computing. Communications of the ACM, Forthcoming (2013)
4. Laffont, J.J., Martimort, D.: The Theory of Incentives. Princeton University Press, New Jersey (2002)
5. Bloom, M., Milkovich, G.: The relationship between risk, incentive pay, and organizational performance. The Academy of Management Journal **41**(3) (1998) 283–297
6. Sato, K., Hashimoto, R., Yoshino, M., Shinkuma, R., Takahashi, T.: Incentive Mechanism Considering Variety of User Cost in P2P Content Sharing. In: Global Telecommunications Conference, 2008. IEEE GLOBECOM 2008. IEEE, IEEE (2008) 1–5

7. Yogo, K., Shinkuma, R., Takahashi, T., Konishi, T., Itaya, S., Doi, S., Yamada, K.: Differentiated Incentive Rewarding for Social Networking Services. 10th IEEE/IPSJ International Symposium on Applications and the Internet (July 2010) 169–172
8. Little, G., Chilton, L.B., Goldman, M., Miller, R.: Exploring iterative and parallel human computation processes. Proceedings of the 28th of the international conference extended abstracts on Human factors in computing systems - CHI EA '10 (2010) 4309
9. Mason, W., Watts, D.J.: Financial incentives and the performance of crowds. In: Proceedings of the ACM SIGKDD Workshop on Human Computation (HCOMP '09). Volume 11., Paris, France, ACM (May 2009) 77–85
10. Hirth, M., Hossfeld, T., Tran-Gia, P.: Analyzing costs and accuracy of validation mechanisms for crowdsourcing platforms. Mathematical and Computer Modelling (2012)
11. Tokarchuk, O., Cuel, R., Zamarian, M.: Analyzing crowd labor and designing incentives for humans in the loop. IEEE Internet Computing **16** (2012) 45–51
12. Scekic, O., Truong, H.L., Dustdar, S.: Modeling rewards and incentive mechanisms for social bpm. In Barros, A., Gal, A., Kindler, E., eds.: Business Process Management. Volume 7481 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (2012) 150–155
13. Baresi, L., Heckel, R.: Tutorial Introduction to Graph Transformation: A Software Engineering Perspective. In: Proceedings of the First International Conference on Graph Transformation (ICGT '02), London, UK, Springer (2002) 402–429
14. Jakumeit, E., Buchwald, S., Kroll, M.: GrGen. NET. International Journal on Software Tools for Technology Transfer (STTT) **12**(3) (2010) 263–271