ARCHITECTING SOCIAL INTERACTION: EXPERIENCES FROM E-HEALTH DESIGN RESEARCH IN THE U-CARE PROGRAM

Jonas Sjöström, Department of Informatics and Media & Department of Public Health and Caring Sciences, Uppsala University, Uppsala, Sweden, {jonas.sjostrom@im.uu.se}

Pär J. Ågerfalk, Department of Informatics and Media, Uppsala University, Uppsala, Sweden, {par.agerfalk@im.uu.se}

Accepted to the SIG Prag Workshop on IT Artefact Design & Workpractice Improvement, 5 June, 2013, Tilburg, the Netherlands

Abstract

Information systems design is increasingly oriented towards systems for social interaction. There is still a lack of design-oriented knowledge to support such design. In this paper, we present an action framework that adds a pragmatic layer to software. Through the framework, web applications are equipped with sophisticated logging and authorization capabilities. We assess the framework by presenting the concurrent and authentic evaluation that occurred through the design process, and provide two examples of how the pragmatic layer supported feature development. Further, we address how the accountability of software benefits from the logs rendered by the framework.

Keywords: Pragmatic, Framework, eHealth, MVC.

1 Introduction

Online interaction systems are now prevalent. Such systems – characterized by their features to support interaction between people for some purpose – include e-learning, knowledge sharing communities, social media applications, and online peer support in eHealth applications. Information and communication technology has affected social interaction across all walks of life, radically changing the ways in which people interact with one another in private life as well as professionally. For example, it has been reported that Facebook is close to having one billion users (SocialBakers, 2013) with access to more than nine million applications and websites integrated with the platform (Newsroom, 2012).

However, although research on information infrastructure is gaining momentum (Eriksson and Ågerfalk, 2010; Hanseth and Lyytinen, 2010; Iannacci, 2010) conceptual design of architectures for social interaction is yet not sufficiently addressed. Indeed, conceptual modeling has been central to the IS field since its inception (Wyssusek, 2006). A common perspective is that information systems represent reality (e.g. Wand & Wang, 1996). When adopting such a perspective, designers aim at creating accurate representations of reality in their models. While the representation perspective still stands strong in the IS field, it is subject to emerging criticism (e.g. Hirshheim et al., 1995; Holm, 1996; Ågerfalk, 2010). The fundamental reason for the criticism is that language is not used only to represent the world but also to engage in and changing the world. In contrast, a pragmatic perspective entails that the way we express ourselves using language should be seen as a fundamental way to bring about change to the world (Austin, 1962; Searle, 1969; Habermas, 1984). Following this perspective, information systems are instruments used to perform action (Goldkuhl & Lyytinen, 1982; Sjöström, 2010) and mediate actors' intentions. This pragmatic perspective on IS has been argued to be fundamental to our understanding of IS-related phenomena (Ågerfalk & Eriksson 2004; Ågerfalk et al., 2006; Ågerfalk, 2010) and may play an important role in the development of both descriptive and prescriptive knowledge in the field.

The aim of this work is to propose a new architectural concept to underpin sophisticated forms of social interaction systems. This new architectural concept is based on action-oriented conceptual modeling to underlie the design. The proposed concept is suggested to be a complement rather than an alternative to existing conceptual modeling. We refer to the new concept as the 'action framework', a pragmatic architectural foundation to support the design of social interaction systems.

The action framework emerged in a design research project in the context of health care. A proof-of-concept is provided, and a set of value-creating implications of the use of the framework are demonstrated. In terms of practical contributions, the action framework is presented in enough detail to make it applicable in other design situations. In terms of contributions to theory, the paper shows in a novel way how the pragmatic strand of IS research may be operationalized conceptually in systems design.

2 Conceptual background

In this section, we introduce the concerns that affected the design of the action framework.

2.1 A socio-material perspective

IS design always brings about change to the social world. The IS research field has a long history of exploring the relation(s) between human and machine agency. The contemporary discourse in the field has introduced the paradigm of socio-materiality as a means to understand, theorize and design such agency. Socio-materiality recognizes that conceptualizations in the IS field should acknowledge the emergent characteristics of the social world (Orlikowski and Iacono, 2001). With such a world-view, the mutability or changeability of artifacts becomes important. Preferably, IS design theory should encompass how the artifact, within reason, can be adapted to a continuously changing use context and evolving technological infra-structures (Gregor and Jones, 2007). We need to conceive of the

emergent properties of both technology and the social world, while also recognizing the complex interplay between them (e.g. Leonardi, 2012; Orlikowski and Scott, 2008; Walls, Widmeyer, and El Sawy, 2004).

We distinguish between (i) the need to relate design work to existing (and emerging) social and technological structures, and (ii) the socio-material insight that new (and existing) artifacts indeed transform the social world where they are put into play. With respect to technological change, designers need to adapt technological solutions to an existing infrastructure. New technology is thus an artefact in a sense, but it is also a contribution to an evolving compound of technology. Technology thus undergoes an evolution over time rather than being deliberately designed for a well-specified purpose known in advance. (Gill and Hevner, 2011) suggest that "the fitness of a design artifact must be estimated using a utility function that considers the full range of characteristics that can impact the likelihood that the artifact will further be reproduced and evolve" (p. 247). As design researchers, aiming at producing relevant and useful artifacts that impact society, it makes sense to factor in the likelihood of future use and evolution of the artifact, not only its aptness to solve a particular problem in a particular situation.

2.2 Action-oriented conceptual modelling

Following the pragmatic and socio-material perspective outlined above, Ågerfalk and Eriksson (2004) conceived of an action-oriented approach to conceptual modelling, which has inspired the action framework developed in the paper at hand. This approach distinguishes itself in two regards: First, it acknowledges the above-mentioned insight from speech act theory that language, and thus information systems, is used also for other purposes than describing a real world outside the system. Second, it went further than other speech-act-based IS approaches in terms of addressing the so-called propositional content of utterances, thus not focusing primarily on what speaking does but treating instead the relationship between what speaking does and what is spoken about. That is, giving equal weight to both the f and p of Searle's (1969) notion of a speech act as consisting of a propositional content p used with an illocutionary function f, i.e. f(p).

Essentially, from a modelling perspective the action-oriented conceptual modelling (AOCM) approach stresses that actions in and by themselves constitute important objects that we need to store information about. This means that, for example, instead of viewing a business process in terms of an order that changes state from being an offer, to an order and then to an invoice, these three concepts represent important business objects and need to be treated as separate entities in conceptual models and resulting database schema. Thus, the interplay between static and dynamic conceptual modelling becomes critical. Dynamic models are no longer simply means to show how entities of a static model changes over time, but become essential sources of knowledge for creating the static model.

2.3 The Model-View Controller (MVC) pattern

Parnas (1972) was an early proponent of separation of concerns, and dependencies between interfaces rather than dependencies between concrete implementations. A contemporary approach based on separation of concerns is the model-view controller (MVC) pattern. MVC separates business logic (models) from user interface logic (views) through a controller. In fact, Tryggve Renskaug (founder of the MVC concept), referred to the compound of a controller and its views as a tool, which resonates well with the pragmatic stance adopted here. In essense, we agree with the tool view, however we place the 'tool' in the context of human action, turning it into an instrument for social action. The actions in the controller are software functions that can be accessed from a web client through HTTP requests. The controller actions constitute a map of the software as an instrument for user actions.

MVC suggests to modularize our source code into models, views, and controllers. The basic principle is that the models represent the computational model of data processing, while the views should be designed to make sense for the users. Software functions in the controllers (called *actions*) retrieve parameters from the user (through the view) and utilize the model classes to produce a result. That result is returned to the user as a new view. We conceive of the models in MVC as a layer for implementation of business logic. Typically, the models are designed using other patterns that are

appropriate to implement specific problems. The models are also where the functionality to access secondary storage resides.

Interestingly, a controller action in MVC has a one-to-one correspondence with a user action. As an example, a user clicks a link in the view to read their e-mail. This causes the web browser to send an HTTP request to the web server. The MVC implementation on the web server looks up the corresponding controller action in the routing table, and executes the function. This correspondence is appealing from a pragmatic point of view, since a piece of code (the controller action) becomes an instrument for action. It creates an opportunity to govern the use of the software by different actors.

Our adoption of MVC into this design context is motivated both by its orientation towards action, and by its successful dissemination into software development practice.

3 Research Method

This paper draws on design experiences from the U-CARE program at Uppsala University. The overarching goal of the U-CARE program is to promote psychosocial health among patients diagnosed with somatic disease and their significant others, hopefully at a lower cost to the benefit of individuals and society. In order to achieve these goals, research is conducted in close collaboration between research groups within the fields of clinical psychology, information systems and economics at Uppsala University. Initial research activities are performed within the areas of paediatric oncology, adult oncology and cardiology in collaboration with clinicians at Uppsala Akademiska Hospital.

The U-CARE program aims at bridging scientific and organizational borders in order to offer a meeting place for researchers, clinicians, and students from different disciplinary backgrounds. During 2010-2013, research studies were designed, and a software platform for psychosocial care was developed. The platform will be applied to conduct randomized controlled trials within paediatric oncology, adult oncology, and cardiology to provide care and psychological treatment to patients and significant others, starting in the second half of 2011.

The design process was characterized by close collaboration between IS researchers (who also acted as system analysts and software developers) and other stakeholders in the research context. For the IS researchers, this development process was part of a design-oriented research endeavour (Hevner et al., 2004; Peffers et al., 2007), meaning that theory as well as practice informed the development (Hevner, 2007).

The multi-disciplinary approach in U-CARE resonates well with the ideals of rigorous evaluation in IS design research as put forth by Hevner et al. (2004). Contributions from psychology and economics, disciplines with a strong quantitative evaluation tradition, ingrain our research with evaluation methods that will be applied to evaluate (i) the clinical efficacy of psychosocial care delivered via the platform, and (ii) the health economics impact of delivering online psychosocial care. Evaluation of these aspects will be conducted through randomized controlled experiments that allow for statistically significant comparisons between treatment groups and control groups, with stratification for various variables.

It is beyond the scope of this paper to go into full details of the evaluation strategy. Instead, we focus on a few aspects of evaluation. The design efforts so far have rendered considerable experience worthy of reporting. The development followed a selection of agile development principles (ref). Development sprints lasted for 2–3 weeks, followed by sprint reviews where the ‘customers’ – various stakeholders (mainly researchers from other disciplines) – were exposed to the most current version of the platform. These review meetings rendered feedback and governed the continued development efforts. The feedback was documented and field notes were taken. E-mail correspondence has also been kept to provide additional corpus.

The stakeholder-centric (Sjöström, 2010) and iterative approach promotes a focus on value creation – a continuous assessment of the platform as a means to contribute to the overarching goals of the U-CARE program. This allows for ideas and design principles to be gradually refined. A stakeholder-centric approach that combines Hevnerian evaluation ideals with action design research (Sein et al.

(2011) thus forms the core of the research design. Essentially, in keeping with Sjöström & Ågerfalk (2009) we embrace rigorous evaluation methods (Hevner et al., 2004) to understand better mutability in design theory (Gregor & Jones, 2007).

4 The Action Framework

In essence, the socio-material perspective and the orientation towards action resulted in the following overarching design principles:

1. Let design be governed by an ensemble view of the artifact. In essence, we are designing a socio-technical system, where technology and the social world are co-constitutive.
2. Recognize the emergent properties of socio-technical systems, and let that understanding influence design choices.
3. Manage metadata for all actions that are implemented as controller actions in the software. The metadata contextualizes actions and adds a pragmatic layer on top of the software, allowing us to map social action to particular software functions.
4. Allow for detailed logging of actions, in order to promote traceability of user actions through the pragmatic layer that is placed on top of semantic content.
5. Let software design be informed by appropriate software design patterns that resonate well with action-oriented conceptual modeling, including the following: (i) Design our platform on basis of a model-view controller (MVC) pattern, which turns out to delineate user-accessible software functions that has a one-to-one correspondence to human action. (ii) Enhance changeability of system by making metadata dynamic in parts that are likely to change[1].

The five principles are interrelated. Action metadata management enables a pragmatic logging of user actions. The action architecture should not allow action to be performed unless there is metadata for the action in the database. The MVC framework, through its controller actions, makes it possible to build a one-to-one relationship between action metadata objects and user-accessible functions in the software (controller actions). Metadata can be used to describe several aspects of action, e.g. whether or not the action should be logged. Thus, ideas #1 - #2 are pre-requisites to implementing idea #3 (detailed logging of actions). The need for logging is justified through the need for accountability. The healthcare context emphasized the need to protect the privacy of individuals. Any privacy breech should be able to investigate in detail retrospectively.

In the remainder of this section, we present how these principles are manifested in the action framework.

4.1 Static model of metadata and logging

The first aspect of the action framework is its metadata and logging (figure \ref{actionframework}). Metadata is managed through the `Action` table, which holds information about (among other things) the `controller_link`. Through the controller link, we can map action metadata to the request URL. On top of that, metadata includes if this action requires authorization, its description, whether or not it should be logged, what type of result it produces, *et cetera*. In brief, the metadata record allows us to store information that can be retrieved when any HTTP request is made to the server. Through metadata, the behavior of the software will adapt.

Further, action is categorized in two ways. First, the `Activity` (e.g. 'peer support'). Second, the `ActionType` (e.g. 'design' or 'peek').

[1] The design process revealed where requirements were ambigous or uncertain, and there was a continual ambition to assess the volatility of stakeholder requirements.

The `Log` table stores a record each time an action (configured to be logged) is performed. The log record is related to the performing actor (`User`) and the context (`Study`). Further, it stores a timestamp, potential error messages, execution time (for technical analysis purposes), and holds information about the object that was manipulated (`reference_class` and `reference_id`). There is also an attribute used to store the entire parameter list in the HTTP request (\verb"parameters"). Potentially, there is a large amount of data in the request. Therefore, we cut off the data in this column if it exceeds a specific size.

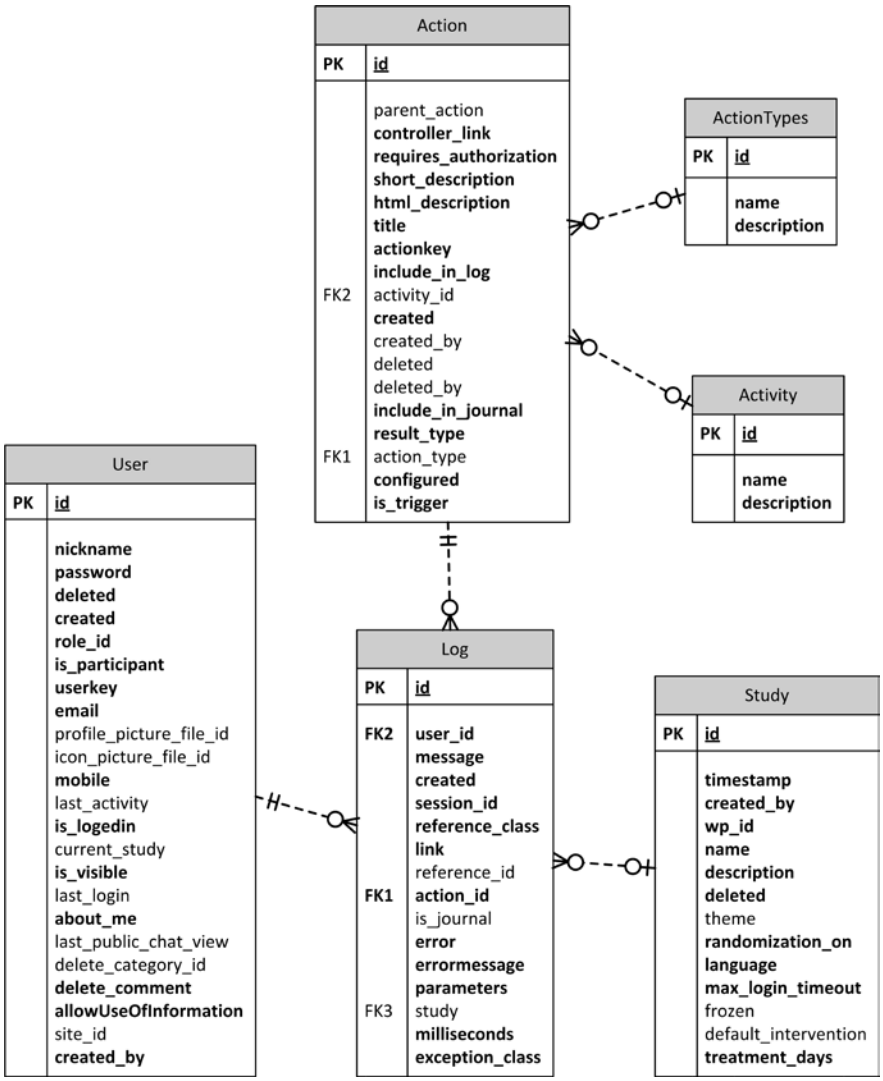

Figure 1. Static design of action metadata and logging

4.2 Static model of authorization

The second aspect of the action framework is the way it is used to authorize user action Figure 2). In the figure, the two mechanisms for authorization are highlighted:

- **Role filtering**. An actor is authorized to do things depending on their role. The `RoleAuthorization` table connects a role to a set of actions by specifying that the role is connected to a specific action type in a certain activity.

- **Context filtering**. An actor is authorized to do things depending on the context, i.e. the study the actor is currently in. Each record in the `StudyActivity` table signals that an activity is allowed for a certain study.

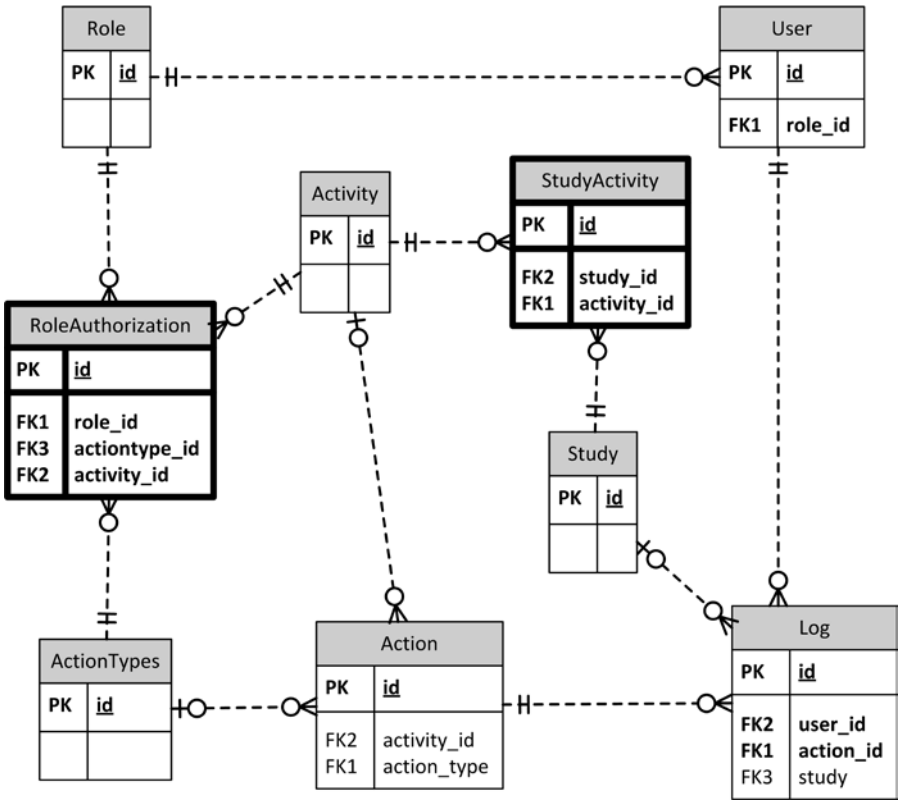

Figure 2. Static design of authorization

These types of filtering are only activated if the action metadata states that the action requires authorization.

4.3 Action Framework Dynamics

In this section we peek at the dynamics of authorization in the U-CARE portal.

The logon process. Whenever a user logs on to the portal, we initialize a table for the user that includes all the metadata for actions that the user is allowed to perform. This is done through a query which include action metadata that fulfils the following condition:

- **A user is authorized to perform an action if** (a) the action metadata states that it requires no authorization, *or* (b) the action is allowed in the study and the action is allowed for the role.
- **An action is allowed for a study if** the activity for the action has been activated for the study.
- **An action is allowed for a role if** the activity-action type combination has been activated for the role.

All allowed actions are be loaded into the user session[2]. The authorizations will be emptied when the user logs out or when the session expires. The authorizations will be re-initialized at the next logon, or when the user switches study. By storing the allowed actions in a hash table in the user session, they

[2] In the web context, any user visiting a web site has a 'session' that is alive from the first http request until the session expires. The expiration timeout is configurable by the developers. The server uses so-called session objects to keep track of data for a specific user.

can be quickly accessed to perform authorization controls. The function `isAuthorizedTo()` in the `User` class is heavily used for this purpose, as shown in the code snippet below.

```
if (currentUser.isAuthorizedTo("/User/ChangeProfilePicture"))
      // Do something…
```

It should be noted that the framework is limited to managing *rights* to perform action. During the design of the framework, other relations such as *should* and *must* were discussed, but it was deemed that such actions were related to the use of the action framework, rather than to the framework as such. The action framework focuses the conceptualization and logging of single actions. In doing so, it supports development of other deontic aspects of action, e.g. that an actor is obliged to do something. In the evaluation section, we provide an example of the development of an indicator framework that suggests what therapists *should* do. In essence, the framework is designed to completely formalize rights, but only to support the management of other deontic relations.

The action filter. The `ActionFilter` class is at the heart of the action framework. An MVC filter allows us to assume control over software execution before and after a controller action is executed. Thus, a filter solution is appropriate to manage authorization and logging mechanisms in MVC-based software.

In order to allow the controllers to share the behavior in the action filter, we created the `PragmaticController` base class. By applying the filter to the base class, all derived classes inherit the behavior defined in the filter. The filter is applied by implementing the `ActionFilter` class, which overrides the `ActionFilterAttribute` implementation of the `IActionFilter` interface. The implication of this mechanism is that the action filter assumes command of the program execution before and after any controller action is invoked, which gives us the possibility to generically define the pre- and post-execution logic for controller actions, as elaborated below.

- **ActionFilter.OnActionExecuting().** This method is invoked before the actual controller action. In this method, we implement the authorization mechanism to match if the requested URL is part of the link of allowed actions for the user (as setup after the user logs on). Further, some initializations are made, such as creating a log entry (if the action is configured to be logged).
- **ActionFilter.OnActionExecuted()**. This method is invoked after the actual controller action. The method updates the log object (e.g. to store specific action results and to measure action execution time).

The filter adds a pragmatic infrastructure layer in the application that allows our controller actions to be clean from details regarding logging, authorization, and other pragmatic logic that is common for all action. Figure 3 shows an overview of the whole filter-based execution of authorization and logging actions. In addition, the pragmatic controller base class offers a set of utilities to retrieve request parameters and update the log object created by the action filter (to add additional success or error info to the log).

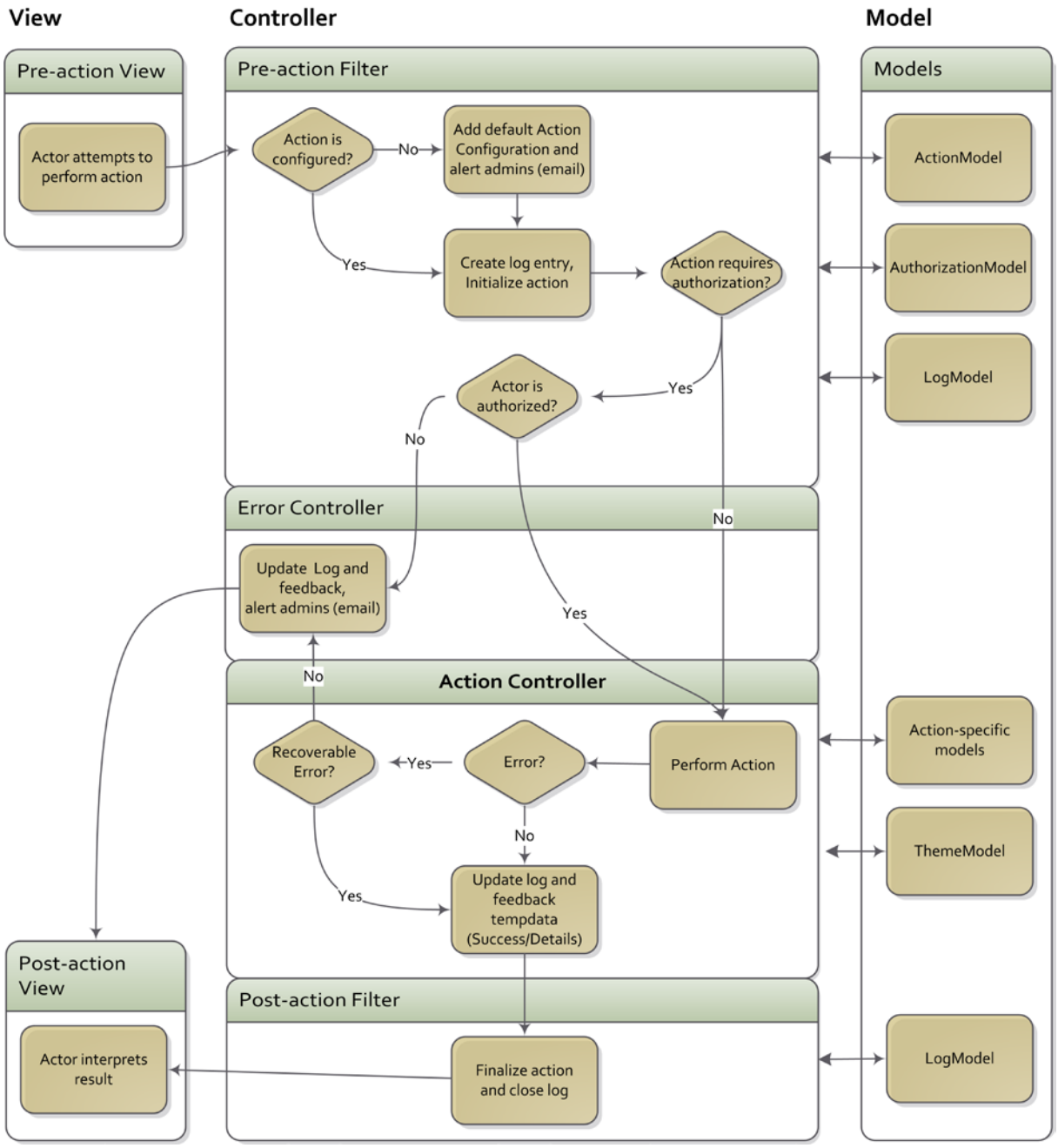

Figure 3. The action framework workflow based on the action filter.

4.4 Additional Logging

On top of the framework-powered logging explained in this chapter, three additional logs are kept in the database. Most contents of these logs could be derived from the logging powered by the action framework. However, the separate logs are kept to maintain a redundancy (for accountability reasons) and to improve query performance.

- **Respondent Behaviour Logging**. This log is used to specifically keep track of how actors fill in questionnaires. The log is kept both for safety (an additional backup facilitating re-construction of potentially damaged, incomplete or lost data) and for research purposes. The log facilitates a reconstruction of an actor's *process* of filling in a questionnaire, rather than

just the end result. Potentially, such data could be valuable to evaluate and re-design questionnaires.

- **Privacy Breech Logging**. The health context promotes us to specifically log privacy breeches. The log facilitates retrospective queries to find out who accessed personal information, and when. Every time a staff user decides to de-identify an individual, a separate log record is stored. This information would be able to derive from the framework-powered log, but we find it appropriate to maintain such information redundantly in a dedicated table.
- **Message Logging**. Due to the fact that the software sends external messages (SMS and E-mail), there is good reason to maintain a specific log of outgoing messages (in addition to the internal messaging supported by the software). Such a log allows for us to build user interfaces that will give us a complete account of how the software has sent messages to any actor. Without stored information about external messages, our aggregated knowledge about communication would be fragmented.

4.5 Evaluation of instantiated IT artifact

The action framework is a novel approach to providing a 'pragmatic layer' to software. The idea is that the use of the framework should keep a pragmatic account of user actions, which has been argued to be an imperative foundation for design of actable IT systems (Goldkuhl and Ågerfalk, 2002; 2005). In addition, user actions are related to an institutional context, in which authorization rules can be defined and upheld by the framework. Our proposition is that this pragmatic layer supports the mutability of the software, which is important given the socio-material world-view (acknowledging the emergent properties of a practice).

In this section, we provide an informed argument (Hevner et al, 2004) to demonstrate the qualities of the designed IT artifact, i.e. the instantiated framework. The argument is split into three categories: (i) formative and concurrent evaluation, (ii) enabling feature development, and (iii) the value of action history.

Formative concurrent and authentic evaluation. First, the action framework has been implemented into a rather complex web application, consisting of approximately 40 000 lines of code and 100 database tables. All actions in the software are configured and accessed through the action framework. The framework has been used to map all controller actions into a workpractice model of activities and actions, and to define the various roles and authorizations in the workpractice context.

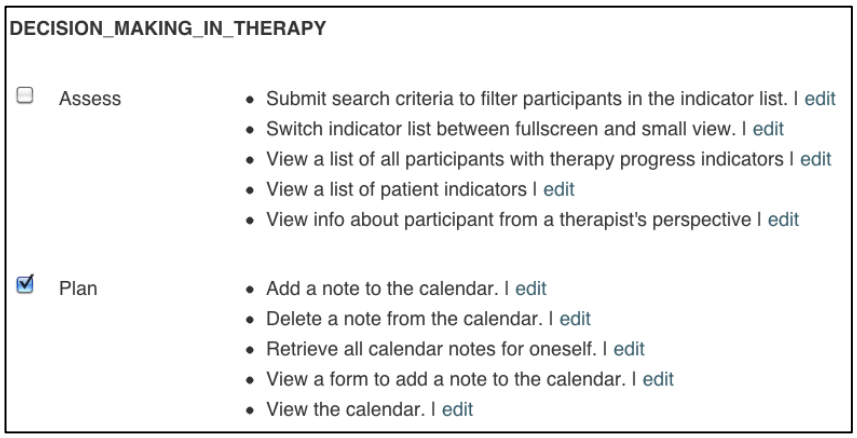

Figure 4. Screenshot showing authorization of a role to access an action type in the activity "decision making in therapy".

The development process was initiated in late 2010. The action framework has been evolving as part of the development since mid 2011, and it has undergone evaluation concurrent with the development process. The framework has evolved along with the software as a whole, undergoing changing and

new requirements. The action framework, however, has been 'stable' for over a year, even though the software as such is still changing. During this time, the system has been available for end-users for beta-testing. Professionals appropriated the software to design therapeutic protocols and research, patient groups have been using the software, and a smaller pilot study has been conducted, in which teenage participants with cancer accessed the software to consume self-help content and cognitive behavioural therapy. In summary, the framework is the result of a stakeholder-centric design process, and it has reached a 'mature' stage that allows for continued development of the software without the need to change the framework. The controller actions as such utilize the framework, allowing them to focus on the task at hand, rather than issues that are common for all actions.

Enabling feature development. An apparent value from the framework is that the logs and metadata has been put into use to develop features in the software. There are numerous examples; we will focus on two of them here.

The *first* feature enabled by the action framework is a dashboard for patient indicators (ref). The dashboard should include various indicators of patients' progress in therapy and self-help. The logging in the action framework made such analyses possible without the need to add tailored code for each indicator. Instead, indicators may now point out a specific action in the workpractice (such as *view a forum thread* or *send internal message*). This makes it possible to add indicators for any workpractice action into the dashboard without adding *any* new code to the platform, since the only required information is metadata about the action, and access to the log file. Based on the metadata, a generic indicator class can tap into the log to calculate the indicator for a patient, for the entire study, or a ratio between the two showing the relative activity of a patient within a study. From a software maintenance point of view, we can add new indicators without having to make changes to the existing code. The indicator framework provides decision support to therapist to help them decide what to do in relation towards their patient group, e.g. give feedback to homework or respond to an internal message from a patient.

The *second* feature empowered by the framework is the automated agent feature, which allows us to configure business rules for a specific study. Such a rule defines how an automated agent (defined as a class in the software) should be triggered as a consequence of an action performed by a user. If action metadata specifies that an action is a trigger, the post-action filter invokes a rule engine. If the performed action has a rule in the current study, the configured agent is put to work. The action framework provides the agent with information about the current user, the current study, the action metadata, and the logged result. The agent thus has 'complete' data about the context as well as the action, which makes it feasible to write agents that perform complex tasks. Since these agents are configurable for each action context (study), the system is ready to manage big variations in business rules between studies. An example of an agent is the patient alert feature, which is triggered whenever a patient submits a questionnaire. The software calculates expressions based on the answers. An agent is triggered, which analyses the expression results. If there is an expression named 'alert' being evaluated to 'true', the agent sends a message to selected staff in the study, telling them that the participant may be suicidal. The business rule is that the therapist must respond to the situation by personally contacting the patient, and take appropriate action to prevent the potential suicidal tendency.

The design of automated agents would not have been possible in such a generic manner without the action framework. Clearly, such a function could be built into any piece of software. However, the action framework makes it possible to manage requirements such as this one without changing the existing code. It allows us to make a plug-in, which can be triggered by any user action, and respond in any desirable way depending on the action results. Again, this is a non-intrusive form of development, where new agents can be added and configured to be included into one context, without risk of affecting what is already there.

The value of action history. The two features mentioned above should only be seen as examples of the potential use of a pragmatic layer in software. From a pragmatist standpoint, the access to the pragmatic layer is fundamental for design. The log allows us to trace any object in the system to the action that created it. This possibility is valuable for at least two purposes: *accountability* and *clarity*.

Regarding accountability, there is often a need to keep a record of things to make it possible to inquire into the past. This may be particularly evident in the eHealth context, where logs can be used to provide a rich account of the treatment process, but also to follow up on potential misuse of information. Regarding clarity, logs are useful to build better user interfaces that reveal important past actions to inform current action. The indicator dashboard is one such example.

In summary, the IT artifact at hand serves as a proof-of-concept of the action framework, illustrating that it is implementable into software, and that it supports various kinds of pragmatic analyses, both to support feature development and accountability.

5 Concluding discussion

The presentation of the artifact shows an instantiation of the action framework. However, it also illustrates the framework as a conceptual solution. The core ideas are reflected in figures 1 – 3. Conceptually, we might use the term 'actor' instead of the term 'user', and we might speak of 'institutional context' instead of 'study'. Such changes in terminology make the model less bound to the situation. Further theoretical elaboration would probably cause us to revise the terminology further, e.g. question the 'action type', which is loosely coupled to the literature[3]. Except for these small issues, the models from the instantiation serve well to illustrate the action framework as a concept. Conceptually, the key elements of the design include:

- Action metadata that allows us to investigate the character of user request
- The management of roles and institutional context, to classify actions through metadata
- The definition of activity and action type, to further classify actions through metadata
- The filtering capability, i.e. a re-routing of the web server request so that we may implement pre- and post-action filtering to include the authorization and logging logic
- The actual action log

The specific attributes of the action and log entities in figure 1 – 2 serve the particular situation well, and we have no reason to believe that they would be less useful in other design contexts.

It should be noted that the model-view controller approach is not a requirement to implement the action framework. It does however provide a structure that simplifies the implementation of the framework, due to the strong support to map controller actions to user actions using configurable request routing.

We have proposed that the action framework adds a valuable pragmatic layer to software design, and given examples of its value in a case of software design. An implementation of the framework supports development by letting the developer focus on the core task of controller actions, while leaving the responsibility for authorization and logging to the framework. Still, the developer can utilize the framework to support feature development (as shown in the examples with the patient dashboard and the automated agents). In addition to supporting development, we have argued that the logging provided by the framework enhances the accountability of the software by facilitating retrospective analyses of user actions.

Future research will include a large-scale evaluation of framework performance, as well as continued monitoring of how the framework supports the continued development of features in the software.

References

Ågerfalk P J (2010) Getting Pragmatic, European Journal of Information Systems, Vol 19(3), 251–256.

[3] The action type was needed to further classify action within activities to support the user interface for authorizations.

Ågerfalk P J and Eriksson O (2004). Action-Oriented Conceptual Modelling, *European Journal of Information Systems*, 13(1), pp. 80–92.
Eriksson, O., and Ågerfalk, P. J. (2010) Rethinking the Meaning of Identifiers in Information Infrastructures, *Journal of the Association for Information Systems*, Vol 11 (8), pp. 433–454.
Gill, T.G., Hevner, A.R. (2011) A fitness-utility model for design science research. *Procedings of DESRIST 2011, LNCS 6629*. pp. 237–252 (2011).
Goldkuhl G and Ågerfalk P J (2002) Actability: A Way to Understand Information Systems Pragmatics, In *Coordination and Communication Using Signs: Studies in Organisational Semiotics 2*, (Eds, Liu K, et al.) Boston: Kluwer Academic Publishers, 2002, pp. 85–113.
Goldkuhl G and Ågerfalk P J (2005) IT Artefacts as Socio-Pragmatic Instruments: Reconciling the Pragmatic, Social, Semiotic and Technical, *International Journal of Technology and Human Interaction*, Vol 1(3), pp. 29–43.
Goldkuhl, G., and Lyytinen, K. (1982). A language action view of information systems. In M. Ginzberg & C. Ross (Eds.), Proceedings of the 3rd international conference on information systems (ICIS'82) (pp. 13–29). Ann Arbor, MI.
Gregor, S. and Jones, D. (2007) The anatomy of a design theory. *Journal of the Association for Information Systems,* 8(5):312–335.
Habermas, J. (1984). The Theory of Communicative Action. Cambridge: Polity.
Hanseth, O., and Lyytinen, K. (2010) Design theory for dynamic complexity in information infrastructures: the case of building Internet, *Journal of Information Technology*, Vol 25 (1), pp. 1–19.
Hevner, A.R. (2007) A Three Cycle View of Design Science Research, *Scandinavian Journal of Information Systems*, Vol 19(2), pp. 87–92.
Hevner, A.R., March, S.T., Park, J. and Ram, S. (2004) Design science in information systems research, MIS Quarterly, 28(1), pp. 75–105. Hevner, A.R. (2007) A Three Cycle View of Design Science Research, *Scandinavian Journal of Information Systems*, Vol 19(2), pp. 87–92.
Hirschheim, R., Klein, H., and Lyytinen, K. (1995). Information Systems Development and Data Modeling: Conceptual Foundations and Philosophical Foundations. Cambridge, UK: Cambridge University Press.
Holm, P. (1996). On the design and usage of information technology and the structuring of communication and work. Doctoral Dissertation, Stockholm University, Sweden.
Iannacci, F. (2010) When is an information infrastructure? Investigating the emergence of public sector information infrastructures, *European Journal of Information Systems*, Vol 19 (1), pp. 35–48.
Leonardi, P.M. (2012) Materiality, Sociomateriality, and Socio-Technical Systems: What Do These Terms Mean? How Are They Related? Do We Need Them? In: Leonardi, P.M., Nardi, B.A., and Kallinikos, J. (eds.) Materiality and Organizing: Social Interaction in a Technological World. pp. 25–48. Oxford: Oxford University Press
Newsroom (2012) http://newsroom.fb.com/Platform/ (Accessed 2012-12-01)
Orlikowski, W. and Iacono, C. (2001). Research commentary: desperately seeking the "IT" in IT research - A call to theorizing the IT artifact. *Information systems research*, 12(2):121–134.
Orlikowski, W.J., and S.V. Scott. 2008. Sociomateriality: Challenging the separation of technology, work and organization. *Annals of the Academy of Management* 2, no. 1: 433–74.
Parnas, D.: On the criteria to be used in decomposing systems into modules. Communications of the ACM. 15, 1053–1058 (1972).
Peffers, K., Tuuanen, T., Rothenberger, M. A., Chatterjee, S. (2007) A Design Science Research Methodology for Information Systems Research, *Journal of Management Information Systems*, Vol 24(3), pp. 45–77.
Searle, J. (1969). Speech acts: An essay in the philosophy of language. Cambridge Univ Press, London.
Sein, M. K., Henfridsson, O., Purao, S., Rossi, M., & Lindgren, R. (2011). Action Design Research. *MIS Quarterly*, 35(2).
Sjöström, J. and Ågerfalk, P. J. (2009). An Analytic Framework for Design- Oriented Research Concepts. In *Proceedings of the 15th Americas Conference on Information Systems (AMCIS 2009)*, pages 302–310.

Sjöström, J. (2010). Designing Information Systems – A pragmatic account. Doctoral Dissertation, Uppsala University, Sweden. ISBN 978-91-506-2149-5.
SocialBakers (2013) http://www.socialbakers.com/facebook-statistics/ (Accessed 2013-04-01)
Walls, J., Widmeyer, G., Sawy, O. El: Assessing information system design theory in perspective: How useful was our 1992 initial rendition. Journal of Information Technology Theory and Application (JITTA). 6, 43–58 (2004).
Wand, Y. and Wang, R. Y. (1996) Anchoring data quality dimensions in ontological foundations. Communications of the ACM, 39(11), 86–95.
Wyssusek, B. (2006). Ontological Foundations of Conceptual Modelling. Scandinavian Journal of Information Systems, 18(1), 63–80.