Self-Adaptive Applications: On the Development of Personalized Web-Tasking Systems

Lorena Castañeda[*†‡]
[*]University of Victoria, Canada
[†]IBM Canada Software Laboratory, CAS Research
Markham, Canada
[‡]Icesi University, Colombia
lcastane@uvic.ca

Norha M. Villegas[‡]
[‡]Icesi University, Colombia
nvillega@icesi.edu.co

Hausi A. Müller[*†]
[*]University of Victoria, Canada
[†]IBM Canada Software Laboratory, CAS Research
Markham, Canada
hausi@uvic.ca

ABSTRACT

Personalized Web-Tasking (PWT) proposes the automation of user-centric and repetitive web interactions to assist users in the fulfilment of personal goals using internet systems. In PWT, both personal goals and internet systems are affected by unpredictable changes in user preferences, situations, system infrastructures and environments. Therefore, self-adaptation enhanced with dynamic context monitoring is required to guarantee the effectiveness of PWT systems that, despite context uncertainty, must guarantee the accomplishment of personal goals and deliver pleasant user experiences. This position paper describes our approach to the development of PWT systems, which relies on self-adaptation and its enabling technologies. In particular, it presents our runtime modelling approach that is comprised of our PWT Ontology and Goal-oriented Context-sensitive web-tasking (GCT) models, and the way we exploit previous SEAMS contributions developed in our research group, the DYNAMICO reference model and the SmarterContext Monitoring Infrastructure and Reasoning Engine. The main goal of this paper is to demonstrate how the most crucial challenges in the engineering of PWT systems can be addressed by implementing them as self-adaptive software.

Categories and Subject Descriptors

D.2 [**Software Engineering**]: Design, Software Architectures—*Representation, Domain-specific architectures*

General Terms

Software Engineering

Keywords

User-Centric, Personalized Web-Tasking, Runtime Models, Self-Adaptive Systems, Context-Awareness

SEAMS'14, June 2–3, 2014, Hyderabad, India
Copyright 2014 ACM 978-1-4503-2864-7/14/06...$15.00
http://dx.doi.org/10.1145/2593929.2593942

1. INTRODUCTION

Users rely on the internet to perform ordinary and repetitive tasks, and feel more comfortable when web systems provide high levels of personalization and automation. Despite the existence of applications to automate and personalize web interactions, these are insufficient to guarantee the accomplishment of user goals when changes in relevant context cannot be fully anticipated at design time. *Personalized Web-Tasking (PWT)* is defined as the automation of repetitive and mundane web interactions that, together with the exploitation of personal context (e.g., information from personal profiles, social relationships, and historical web interactions), seeks to optimize user experiences by assisting people in the fulfilment of personal goals using internet technologies [3].

Managing web interactions at runtime is a complex task. A first cause of this complexity is that internet infrastructures experience frequent and dynamic updates, as well as additions or removals of numerous devices, applications, communication protocols and services. A second cause is that the situations and preferences of users may change while performing web interactions. In regular web interactions, dealing with this complexity implies for the user to be aware of changes that may affect the accomplishment of personal goals to adjust web tasks accordingly. For example, by installing updates, consuming new services, learning about new applications, accessing web services from different devices, or creating multiple profiles. Under these circumstances, the effectiveness of web interactions to accomplish personal goals is limited by the level of expertise of the user to interact with specific technologies, thus compromising not only user experiences but also the value that businesses can obtain from internet technologies.

The success of PWT relies on the capability of automating mundane web-tasks and adapting the infrastructures and web applications that support such web-tasking in order to free users from the management of these complexities. Therefore, PWT software systems must be capable of understanding user needs, act upon changes in relevant contexts, and recognize users as central controllers of web interactions to assist them in performing web-tasks with a minimum of effort, while maximizing their satisfaction. For this, PWT applications must be designed and implemented as software systems that expose self-* properties [6, 7, 9], and support situation-awareness [4, 10]. They must be self-configurable

for dealing with dynamic context changes by adjusting their behaviour or structure at runtime. In particular, they must expose self-healing and self-optimizing capabilities to ensure the accomplishment of user goals in the presence of system failures, and guarantee pleasant user experiences under changing environments, user goals and situations.

To implement PWT applications as self-adaptive software (SAS) systems we propose a set of runtime models that allow us to represent user goals, system concepts and context entities as artefacts that maintain a causal connection and that can be manipulated while the system is running. To design and implement the PWT system presented in this paper, we used the DYNAMICO reference model proposed by Villegas et al. [11]. DYNAMICO supports the implementation of SAS systems that are highly affected by changes in goals and context situations at runtime. SAS systems derived from DYNAMICO implement three subsystems that might be in turn self-adaptive: (i) the control objectives subsystem maintains the relevance of the PWT system with respect to changes in the user's personal goals; (ii) the adaptation mechanism allows the system to act upon changes that occur either in the context or in the requirements of the system; (iii) the monitoring infrastructure keeps track of context events that are relevant to the execution of task sequences.

Monitoring mechanisms in our PWT system are provided by the SmarterContext monitoring infrastructure [12] and its ontologies, which we extended with our PWT ontology. Indeed, our extension instruments the SmarterContext Reasoning Engine (SCoRE) with basic capabilities required to infer, from personal context information, knowledge valuable to optimize future PWT executions. For example, by recommending other web services or a different web task sequence based on the similarity of the user with others (e.g., taking into account the user's gender, age, location, and preferences).

The contributions of this paper include our vision on the implementation of PWT systems as SAS systems. Our design is based in the DYNAMICO reference model, and also uses the SmarterContext infrastructure. Additionally, the self-adaptive capabilities of our PWT implementation are supported by our two runtime models presented in this paper: (1) our personalized web-tasking (PWT) model, an ontology that defines the concepts and artefacts of personalized web tasking; and (2) our goal-oriented context-sensitive web-tasking (GCT) model, a representation of the evolving web-tasking goals and web interactions of a user, as well as the relevant context that may affect the successful execution of the tasks.

The remaining sections of this paper are organized as follows: Section 2 describes an online grocery shopping application scenario that we use to illustrate the PWT domain. Section 3 presents our runtime models and the implementation of our PWT system. Finally, Section 4 discusses challenges, and Section 5 concludes the paper.

2. AN ONLINE GROCERY SHOPPING SCENARIO

We selected online grocery shopping as the personal goal for our case study because its activities are repetitive, performed periodically, and can involve social interactions (e.g., relatives, friends, and other users). Moreover, the fulfilment of grocery shopping goals requires a variety of services and different internet data sources.

In regular grocery shopping scenarios (i.e., those not supported by PWT systems), we identify four main web interactions that are manually executed by the user:

(1) The user *gets the shopping list* which implies to log into her preferred grocery list service using a web browser or a mobile application.
(2) The user *locates the proper stores* using geo-localization services to find nearby grocery stores that she will filter manually according to her preferences that are already available in her social network.
(3) The user *creates independent shopping lists* by matching both items and stores according to different criteria. For instance, the category *best deals of the season*, which implies to compare prices manually to select the best offer according to her budget, or best reviews for both products and stores.
(4) The user *proceeds with the purchase* selecting one of two possible ways: if the store provides an online purchase service the user can proceed with the payment and schedule the delivery; otherwise she will have to manually plan the grocery shopping visit while taking into account different conditions that can affect the pick-up process such as time, traffic, and the store's shopping hours.

Online grocery shopping requires PWT systems for optimizing user experience and increasing revenue generation. This is because shopping web-tasking is exposed to uncertainty generated by the interoperation among different web services (e.g., changes in service compositions, incompatibility of data), and the changes in the user's context (e.g., location, preferences, special events, or her behavior when browsing the internet). Online grocery shopping activities are also affected by social context, that is, the information gathered from other people within the user's social sphere (e.g., friends, relatives, and colleagues). By effectively combining these context dimensions, it is possible to adjust the PWT sequence with the goal of achieving the goal according to the actual user situation or concerns. For example, a birthday party event in the user's calendar will update the shopping list with new items that are required only for this particular event, and might also imply the rescheduling of task's execution time. Similarly, based on the rankings that other users provide to services, the PWT system may suggest for example a better service to obtain discount coupons. The user could decide to include this new service into her task sequence to optimize the accomplishment of her shopping goal.

Having no PWT support to cope with these changes that might affect the achievement of personal goals, the user must adapt the set of required web interactions manually thus hampering the quality of the user's experience.

3. OUR PWT SOFTWARE SYSTEM

In the vision of creating personalized web-tasking applications, the user's identity, interactions, personal goals, preferences, and context, determine the decisions and the behaviour of the web interactions of the user. For this purpose, these applications must translate the user's goal into a sequence of web tasks that can be executed automatically in behalf of the user to provide a better user experience.

The realization of PWT requires the implementation of SAS systems to support as much automation and personaliza-

tion as possible in all aspects of the user's web-tasking. We posit that the design and implementation of PWT systems must be driven by the following requirements, according to our conceptual representation as depicted in Figure 1:

(1) *Personal goal identification:* The system must understand the user's personal goal in machine readable language that can be processed at runtime.

(2) *Web-task sequencing:* The system must define ordered sequences of web-subtasks required to achieve a personal goal, including the specification of dependencies, services, inputs and outputs.

(3) *Web-task personalization:* The system must exploit context information to define the web-task sequence to be executed. This context information can be a user's personal context [12], historical web-interactions, other users in her social network, and all relevant context from the environment.

(4) *Web-task execution:* The system must execute the selected web-task sequence while performing management activities such as life cycle control, conflict resolution, and fault recovery.

(5) *Context-awareness and self-adaptive support:* The system must act upon unexpected context changes by adapting itself at runtime, when applicable.

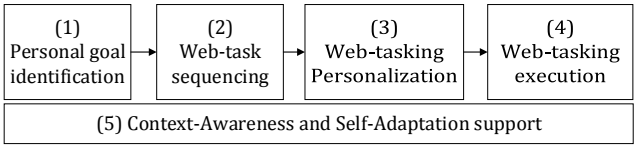

Figure 1: Personalized Web-Tasking conceptual elements

3.1 Runtime Models for Realizing PWT

To guarantee dynamism and flexibility in PWT, we require models that can be manipulated and adapted during the execution of the system [1, 8]. We have categorized modelling requirements for PWT systems into three concerns: user personal goals, web-task sequences, and context. Moreover, we surveyed different modelling approaches, in particular goal-oriented models [2], and ontologies, with the purpose of identifying the most appropriate models for our problem. As a result of this comparative study, we proposed two approaches that cover all the modelling requirements we identified for PWT systems: (1) Our personalized web-tasking (PWT) model, and (2) our goal-oriented context-sensitive web-tasking (GCT) model.

The PWT ontology model depicted in Figure 2, is the base model and an ontology that defines the concepts of personalized web tasking: personal goal, web-task sequence, execution plan, activities, information resources, inputs, conditions, and satisfaction properties. Our PWT ontology is available as a runtime model in the form of an OWL2/RDF file.[1] Moreover, our ontology is applicable in different domains (i.e., tasking supported by stand-alone software systems), and can be extended according to the evolution of the web.

Our GCT model is an extension and redefinition of the iStar Framework [5]. This model supports the specification of evolving web-tasking goals, personal web interactions, and the relevant contexts. In particular, we extended the iStar atomic notions of actor, goals, task and resources, to support the specification of web-tasking goals, task sequences, tasks and subtasks, as well as the relationships among tasks, subtasks and resources. Figure 3 depicts an example of the GCT model for our online grocery shopping scenario.

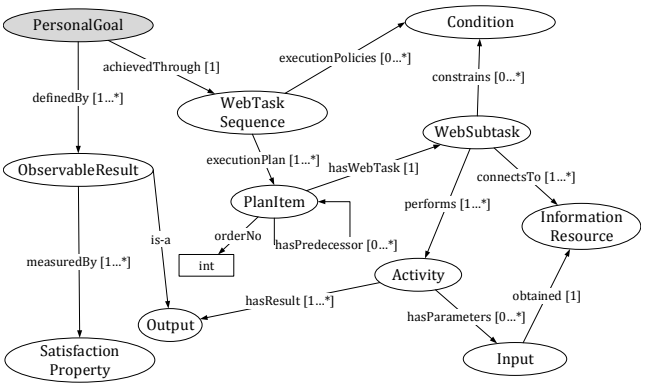

Figure 2: Simplified view of our PWT ontology model

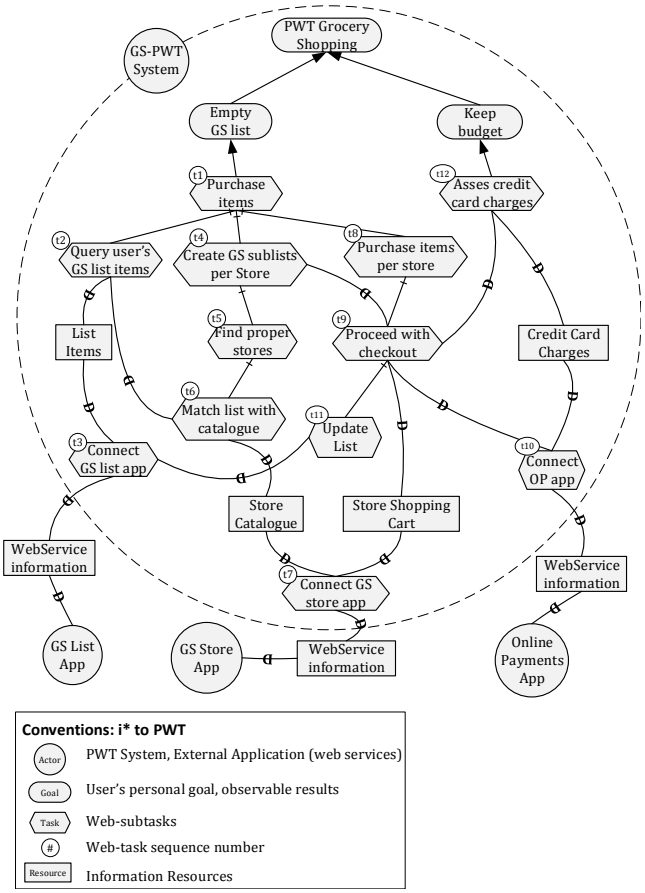

Figure 3: Goal-oriented Context-sensitive web-tasking (GCT) model expressed using the i* Framework [5].

To demonstrate the practical feasibility of our modelling approach, we implemented a prototype of our PWT runtime

[1] http://www.rigiresearch.com/research/pwt/pwtOntology.owl

models for the grocery shopping scenario. We used Apache JENA—an open source Java framework for building semantic web and linked data applications,[2] to create the runtime models required in our online grocery shopping scenario.[3] Each RDF graph constitutes a runtime instantiation of a user's web-tasking. These graphs are processed by our PWT System to execute the web-tasking on behalf of the user.

3.2 Components of our PWT System

Our PWT system comprises the following main software modules:

(1) *Web-Tasking Knowledge Infrastructure:* provides the instrumentation to express personal user goals (i.e., the control objectives of the PWT system—the upper layer in DYNAMICO) in the form of a GCT model. The components in this module perform two main activities: (i) record information from a user's web interactions, and (ii) represent this information in the form of a GCT model instance (cf. Figure 3). To record a user's web interactions we need to identify, interpret and characterize web actions (e.g., click, selection, or inputs) and data, which implies instrumenting the browser, devices and web sites to extract this information. Finally, the system translates the information gathered from the user's web interactions into a GCT model.

(2) *PWT Model Processor:* processes our PWT models to generate the corresponding RDF graphs (GCT models) that specify all the information about the user web-tasking, which includes web-task sequences, web services, inputs, and satisfaction metrics. The *PWT Model Processor* might determine that for one personal goal there are several RDF graph candidates based on the user's historical behaviour. In this case, these candidates are sent to the next module of the system; *the Personalization Engine* will select the proper web-task sequence to be executed.

(3) *Personalization Engine:* selects the proper RDF graph based on personal context information provided by the SmarterContext *Reasoning Engine* [12]. This context information is relevant for its particular goal, and can be either static (e.g., age or gender) or dynamic (e.g., location, preferences, or social information). The result is one single RDF graph.

(4) *Web-Tasking Effector:* executes a sequence of web interactions on behalf of the user, and at the same time acts as a controller of the system by preforming the following activities: web-task life cycle control (i.e., the start and end of the web-task), web services invocation (i.e., communication protocol and data exchange), user interaction requests (i.e., tasks that can not be fully automated), conflict resolution (e.g., service unavailability, or data type incompatibility), and a final assessment to guarantee the fulfilment of the personal goal. Additionally, it supports three levels of automation: manual (i.e., the user explicitly specifies which web interactions correspond to a particular personal goal), assisted by recommendations (i.e., the system suggests web interactions to the user), and fully automated (i.e., the system executes web interactions on behalf of the user).

[2] http://jena.apache.org/

[3] http://www.rigiresearch.com/research/pwt/GroceryShopping.rdf

3.3 Realizing Self-Adaptation in our PWT System

As a reference model, DYNAMICO provides software engineers with a characterization of the structure and behaviour of the minimal set of components required to implement context-driven SAS systems. This model defines three feedback loop subsystems causally connected as depicted in Figure 4: control objectives manager (CO-FL), adaptation controller mechanism (A-FL), and monitoring infrastructure (M-FL). We designed our PWT system using the DYNAMICO reference model.

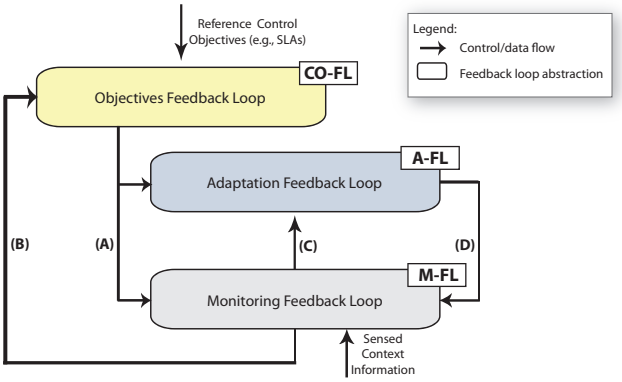

Figure 4: DYNAMICO reference model for context-driven self-adaptive software systems [11]

Figure 5 presents our runtime models and PWT system's modules mapped onto the DYNAMICO layers. The upper level corresponds to the *Web-Tasking Knowledge Infrastructure* that receives the user's personal goals, the control objectives of DYNAMICO. This level keeps track of changes in the user's goals to trigger the adaptation of the PWT system (second level) and/or the monitoring infrastructure (third level). The main objective of the *Web-Tasking Knowledge Infrastructure* is to guarantee that the PWT system is really the one that will allow the user to accomplish her personal goals. The middle level comprises the adaptation mechanism and the target system. The latter corresponds to our *PWT Model Processor*, *Personalization Engine*, and *Web-tasking Effector*. Our PWT system exposes behavioral and structural adaptation. With respect to behavioral adaptation, our system supports the modification of existing algorithms or the injection of a new algorithm to process newer versions of the GCT model. Regarding structural adaptation, our system supports the reconfiguration of its architecture according to different grid-computing topologies, in particular to optimize the execution of parallel web-tasks. The lower level of DYNAMICO corresponds to the SmarterContext infrastructure, which enables our PWT system with dynamic context-aware capabilities. Finally, our runtime models are required across all the DYNAMICO levels of our PWT system. This is expected since the system must maintain a causal connection not only between these models, but also between them and the system implementation.

3.3.1 Context-Awareness Support

By extending the SmarterContext ontology to be used in the Personalized Web-Tasking domain, we instrumented the SmarterContext *monitoring infrastructure* to identify and understand relevant context that is required during

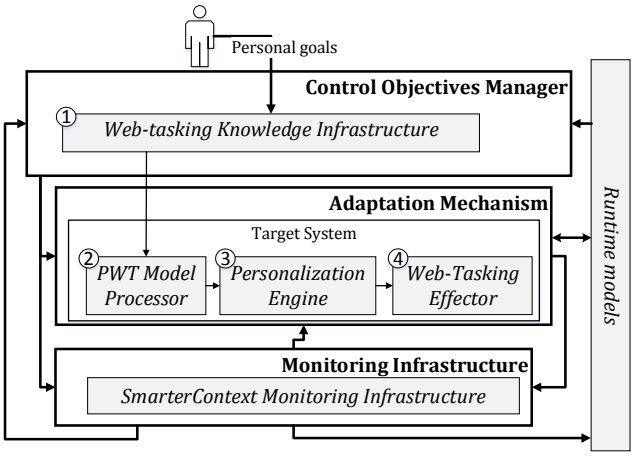

Figure 5: Mapping our runtime models and system modules onto the DYNAMICO reference model.

the execution of our PWT system. Instances of context entities and information relevant to PWT systems include: (i) external context such as other systems that interact with the PWT system during the web-tasking execution. For example, in our shopping scenario, external context can be the grocery store web services used to retrieve information from product catalogues; (ii) internal context that includes all the elements within the system, the communication channels among them and any information about the execution of the system. For instance, the information exchanged between two web-tasks that are sequenced, and the life cycle controller; (iii) domain-related context that affects the way users interact with the web to achieve a personal goal. For example, web technologies and devices, correlation among web applications; finally, (iv) user context, that includes not only personal information (e.g., location), but also social information (e.g., food preferences of the family members that live with the user). Personal context includes also the web behaviour of the user, for example, the way of interacting with a new service, the frequent addition or removal of steps (web interactions) to standard executions, or the modification of input information.

3.3.2 Levels of Self-Adaptation

In our PWT system, self-adaptation occurs at two levels: (1) the runtime models, and (2) the PWT system components. Adaptation in the runtime models are triggered by changes in the PWT problem domain that affect the user's web-tasking behaviour (e.g., technologies used to execute grocery shopping). The second level corresponds to behavioural or structural adaptations in the PWT system triggered by context changes. Certainly, the modifications in the runtime models (first level) necessarily imply adaptations in the PWT system (second level) required for it to understand, instantiate, and execute the PWT models properly.

As an illustration, let us assume that the user scans with her mobile phone a QR code found in the product labels. These labels contain detailed information such as type of producer (e.g., organic, industrial, artisan, or imported), and transportation (e.g., type of vehicle, transport conditions, or original and passing locations). Figure 6 (upper figure) shows a sequence diagram of the PWT system activities to perform self-adaptation as follows: (1) By scanning the QR code in the products, the user notifies the system that a

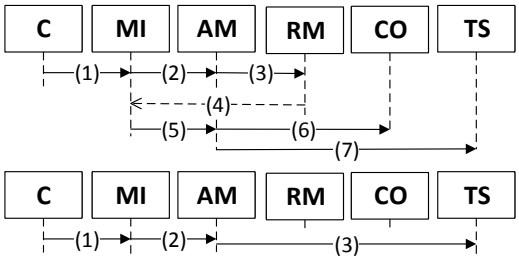

Figure 6: Self-adaptation sequence diagrams for a change in the context. Legend: C: Context, MI: Monitoring Infrastructure, AM: Adaptation Mechanism, RM: Runtime Models, CO: Control Objectives, and TS: Target system.

change in the context domain had occurred (i.e., a new user interaction). This event is sensed by the MI and it is classified as relevant, given that the interactions with products is a concern in the grocery shopping personal goal of the user. (2) The MI reports the relevant event to the AM which recognizes that the current representation of the domain (cf. Figure 3) does not include information to understand and process those QR codes. (3) As a result, the AM effects the corresponding adaptations over the RM to update the PWT models with this new source of information relevant for the web-tasking of the user. (4) Moreover, the MI notices a new version in the runtime models that jeopardizes the proper execution of the PWT System. Naturally, (5) the MI reports the event to the AM, who in response adapts both (6) the CO (specifically the Web-Tasking Knowledge Infrastructure) to understand this new information and generate the proper GCT model realizations (RDF Graph), and (7) the TS to process, personalize, and execute the RDF Graph accordingly.

It is worth mentioning that not all the adaptations in the second level (i.e., the PWT system components) are generated by changes in the runtime models. In our approach the MI is aware of all the relevant contexts. For example, let us assume that the user travels to another city for two months. Figure 6 (lower figure) depicts the following sequence of activities: (1) The MI realizes that the user has changed her location and (2) reports the event to the AM. In response, the AM (3) effects the corresponding adaptations in the TS, more specifically in the *Personalization Engine* to include the new location of the user as an important element for personalizing the web-tasking. In this case, the adaptation was not triggered by changes in the models or the control objectives.

The separation of concerns introduced by DYNAMICO allows us to address the variety of design and implementation concerns inherent in PWT applications. In particular, it facilitates the management of the dynamic nature of goals, models, and systems separately, while maintaining the required causal connections among them.

4. DISCUSSION

We presented our approach to the realization of PWT applications as SAS systems. First, we presented our runtime modelling approaches (i.e., PWT Ontology and GCT models), which support the implementation of dynamic, flexible and self-adaptive PWT systems. In addition, we demonstrated its practical feasibility by implementing RDF graphs to make

our GCT models available at runtime, and by describing the software modules of our PWT system. Our approach to implement PWT systems as adaptive software exploits previous SEAMS contributions, in particular the DYNAMICO reference model [11], and the SmarterContext monitoring infrastructure and reasoning engine [12].

Finally, we discuss two challenges that are crucial for the improvement of self-adaptive capabilities in PWT systems: (1) web instrumentation and (2) social context.

Web Instrumentation. Current web applications are not fully instrumented to be recorded, characterized, and modelled by third party applications as required by PWT systems. There are some approaches to record and replay web interactions (e.g., ClickTale,[4] Selenium,[5] Solex,[6] or Mouseflow[7]). However, they are mostly limited to register user actions (i.e., click, selection, and browsing) and inputs (e.g., the autofill mode of the browsers). Moreover, these approaches are not designed to respond to unpredictable context changes, nor to understand how a user's changing personal goals affect web interactions. We posit that the implementation of dynamic web environments (i.e., applications, browsers, and devices) that enable PWT systems to acquire contextual details from the user's web experience, and to adapt web interactions accordingly, is a major *web instrumentation* challenge in the self-adaptation realm. Web instrumentation implies the development of self-adaptation middlewares to address the complexity raised by the dynamic nature of user web interactions.

Social context. Social context comprises all the information gathered from the social relationships (whether virtual or real) of the user that is relevant to a personal web-tasking situation. Context sources of this type include the user's social sphere (e.g., friends, relatives, or coworkers), people in the same event as the user (e.g., birthday, graduation, or holidays), or other PWT system users that are similar. A big challenge is that the availability of this context information is limited to certain public sources (e.g., social networks), which are insufficient to optimize user experiences in PWT systems. In order to exploit this type of context effectively, we require software systems to discover, monitor, manage, and process context sources different from the traditional ones. Thus, SAS systems must be able to cope with the challenges posed by the dynamic nature of the user's social persona (i.e., how, where, when, and with whom the user relates), and the hardware and software technologies that support social interactions (e.g., social networks, collaborative applications, or communication devices). Indeed, managing social context in PWT systems requires significant self-adaptive instrumentation.

5. CONCLUSIONS

In summary, we showed how to use self-adaptive systems technology to solve challenges in the realization of PWT applications. Moreover, we demonstrated how PWT systems can be designed and implemented on top of existing SEAMS technologies. Indeed, PWT is an attractive application to leverage the advantages of self-adaptive systems. Finally, we identified two challenges that we are addressing in our ongoing research: the need to (1) develop web instrumentation to enable web interactions with dynamic capabilities, and (2) to exploit self-adaptation to gather relevant context from non-traditional context sources, that not only can appear at runtime but also cannot be fully specified at design time.

[4]http://www.clicktale.com/
[5]http://docs.seleniumhq.org/
[6]http://solex.sourceforge.net/
[7]http://mouseflow.com/

6. REFERENCES

[1] G. Blair, N. Bencomo, and R. France. Models@ run.time. *Computer*, 42(10):22–27, 2009.

[2] D. Bolchini and J. Mylopoulos. From task-oriented to goal-oriented web requirements analysis. In *Proceedings 4th International Conference on Web Information Systems Engineering*, pages 166–175. IEEE, 2003.

[3] L. Castañeda, N. M. Villegas, and H. A. Müller. Towards personalized web-tasking: Task simplification challenges. In *Proceedings 1st Workshop on Personalized Web-Tasking (PWT 2013) at Ninth IEEE World Congress on Services (SERVICES 2013)*, pages 147–153, 2013.

[4] M. R. Endsley. Toward a theory of situation awareness in dynamic systems: Situation awareness. *Human factors*, 37(1):32–64, 1995.

[5] J. Horkoff, E. Yu and G. Grau. iStart Guide. Online: http://istar.rwth-aachen.de/tiki-index.php?page=istarQuickGuide. Oct. 2013.

[6] J. O. Kephart and D. M. Chess. The vision of autonomic computing. *Computer*, 36(1):41–50, 2003.

[7] Lemos et al. Software engineering for self-adaptive systems: A second research roadmap. In *Software Engineering for Self-Adaptive Systems II*, volume 7475 of *Lecture Notes in Computer Science*, pages 1–32. Springer Berlin Heidelberg, 2013.

[8] B. Morin, O. Barais, J. Jezequel, F. Fleurey, and A. Solberg. Models@ run.time to support dynamic adaptation. *Computer*, 42(10):44–51, 2009.

[9] H. Müller, M. Pezzè, and M. Shaw. Visibility of control in adaptive systems. In *Proceedings of the 2nd international workshop on Ultra-large-scale software-intensive systems (ULSSIS 2008)*, pages 23–26, 2008.

[10] V. E. Silva Souza, A. Lapouchnian, W. N. Robinson, and J. Mylopoulos. Awareness requirements. In *Software Engineering for Self-Adaptive Systems II*, volume 7475 of *Lecture Notes in Computer Science*, pages 133–161. Springer Berlin Heidelberg, 2013.

[11] N. Villegas, G. Tamura, H. Müller, L. Duchien, and R. Casallas. DYNAMICO: A Reference Model for Governing Control Objectives and Context Relevance in Self-Adaptive Software Systems. In *Software Engineering for Self-Adaptive Systems II*, volume 7475 of *Lecture Notes in Computer Science*, pages 265–293, 2013.

[12] N. M. Villegas. *Context Management and Self-Adaptivity for Situation-Aware Smart Software Systems*. PhD thesis, University of Victoria, Canada, February 2013.