ISSN 0005-1144
ATKAFF 50(3–4), 252–264(2009)

Siniša Srbljić, Dejan Škvorc, Daniel Skrobo

Widget-Oriented Consumer Programming

UDK 004.738.5
IFAC 2.8.3

Original scientific paper

Today's consumer network applications are composed of complex sets of interconnected hardware and software components. Application providers build component-level workflows to provide required functional properties and quality of service (QoS). However, to enhance the quality of user experience (QoE), applications are nowadays exposed to consumers as sets of widgets, i.e. compact and fully functional application modules displayed in a web browser. Widgets are built for various application domains, such as personal finance management or highly-specialized analyses of oceanographic, climate, and atmospheric data.

To augment application's built-in QoS settings with personalized QoE properties, consumers define personalized workflows on top of widgets. They select widgets using QoE-related criteria, such as widget data source reliability, data processing, and presentation features and interact with them through a graphical user interface (GUI). To automate consumers' manual operations over a set of widgets, we present *Geppeto*, a consumer-oriented framework for programming application-level workflows over widgets. *Geppeto* uses consumer-programmable widgets that integrate a set of application-specific widgets into the workflow and become integral parts of an application. The proposed framework enables each individual consumer to build personalized QoE-aware applications. Furthermore, the community is empowered with enormous *application development potential*, because of the large number of web consumers who can participate in software development.

Key words: Application development potential, Consumer programming, Personalization, Quality of experience, Web automation, Widget composition, Workflow programming

Potrošaču prilagođeno programiranje zasnovano na udomljenicima. Današnji potrošaču usmjereni primjenski programi grade se od složenog skupa umreženih sklopovskih i programskih komponenti. Pružatelji usluga povezuju komponente u radne tijekove kojima se postižu zahtijevana funkcijska svojstva i zadovoljavajuća razina kakvoće usluge. Međutim, za poboljšanje kakvoće korisničkog doživljaja, primjenski programi potrošaču se izlažu u obliku udomljenika, odnosno skupa programskih komponenti u obliku primjenskih programa kojima se rukuje putem web preglednika. Udomljenici se grade za različita područja primjene, od široko primjenjivog upravljanja osobnim financijama do usko specijaliziranih analiza podataka o stanju mora, klime i atmosfere.

Kako bi ugrađena svojstva kakvoće usluge nadopunili vlastitim mjerilima kakvoće doživljaja, potrošači nad skupom udomljenika grade poosobljene radne tijekove. Izgradnja radnog tijeka sastoji se od izbora potrošaču prikladnih udomljenika za pribavljanje, obradu i prikaz informacija te njihovog povezivanja u cjelinu međudjelovanjem putem pripadajućih grafičkih korisničkih sučelja. Kako bi se izbjeglo učestalo ručno upravljanje radom skupa udomljenika, u ovom radu predlažemo Geppeto, potrošaču-usmjerenu radnu okolinu za programiranje radnog tijeka na razini udomljenika. Izgradnja radnog tijeka zasnovana je na skupu potrošački programirljivih udomljenika, čijom se ugradnjom u radni tijek primjenskog programa automatizira rad primjenskih udomljenika. Predložena okolina potrošačima omogućava izgradnju primjenskih programa prilagođenih vlastitim mjerilima kakvoće doživljaja. Nadalje, ukupni stvaralački potencijal društvene zajednice mnogostruko je povećan zbog velikog broja korisnika kojima je omogućeno sudjelovanje u razvoju primjenskih programa.

Ključne riječi: stvaralački potencijal u razvoju primjenskih programa, programiranje prilagođeno potrošaču, poosobljavanje, kakvoća korisničkog doživljaja, automatizirana uporaba weba, usložnjavanje udomljenika, programiranje radnog tijeka

1 INTRODUCTION

The primary purpose of present consumer communications and networking is to inform, educate, and entertain the consumers. Digital TV, video-on-demand, online gaming, and P2P-based content distribution are typical examples of contemporary consumer networks. However, as the modern society becomes increasingly dependent on information technology in our daily lives, consumer networks go beyond these traditional realms. They become applica-

ble in broader area of consumer activities that range from interactive education, business and commerce, interaction with government agencies, to healthcare and social activities.

As consumer networks get more deeply integrated into consumers' everyday life, they strongly impact their privacy, commodity, and business productivity. Non-functional properties such as visual appearance, trustworthiness, and reputation of consumer network applications are becoming at least as important as their functional characteristics [1]. Where the data will be sourced from, what type of processing will be applied and how the results will be presented are subjective measures of consumer satisfaction with an application. Therefore, in information-sensitive applications, such as personal financial applications or NOAA (National Oceanic and Atmospheric Administration, US Department of Commerce) applications for analyses of oceanographic, climate, and atmospheric data, quality of experience (QoE) is recognized as a new driving force that makes the applications attractive to use. Since QoE is a subjective measure of consumer satisfaction with an application, a consumer becomes the central authority to assess the QoE parameters and define what the application workflow will look like [2]. In this article, we discuss the challenge of designing future consumer networks flexible enough to suit the subjective preferences of individual consumers.

Complex applications are nowadays exposed to consumers through widgets. Widgets are small web applications displayed in a web browser and equipped with a GUI for interaction with the background processes. To shape the quality of experience, consumers are given the ability to choose a preferred set of widgets and build personalized dashboards that are displayed in a web browser and control execution of complex applications. However, this requires intensive manual operation with multiple widgets, which is tedious and becomes impractical. In this paper, we present *Geppeto* (***G**adg**e**t **P**arallel **P**rogramming **To**ol*), a web automation framework based on consumer-programmable widgets for programming the workflows of consumer applications. We use widgets as basic building blocks of consumer applications, while programming-by-demonstration technique is applied to define the workflow among widgets. Although many tools for web automation exist on the market (*iMacros*, *Newbie*, *Chickenfoot*, *iRobot*), *Geppeto* is the first framework that uses consumer-programmable widgets to automate the operation of application-specific widgets. Using *Geppeto*, consumer-programmable widgets integrate application-specific widgets into the workflow and become an integral part of a consumer application.

The rest of the paper is organized as follows. In Section 2, we identify the elements that impact the quality of user experience of computer applications and recognize consumer programming as a methodology for advanced user experience management. In Section 3, we describe a model of building consumer applications based on widget networking. In Section 4, we present a framework based on three types of consumer-programmable widgets. The following three subsections describe features and application of each type of consumer-programmable widget. In Section 5, we present the usage of our framework in the design and development of a consumer network application. Section 6 introduces *application development potential* as a measure of creative power of the consumer community engaged in development of computer applications. Section 7 concludes the paper.

2 QUALITY OF USER EXPERIENCE

As shown in Fig. 1, three basic properties of computer applications that impact user experience are usefulness, usability, and performance. Usefulness is a measure of how using an application is beneficial to a consumer while solving a particular task. For example, a currency converter application is useful for managing personal finances, but not very useful for getting the latest weather report. Usability refers to the ease with which a consumer can use an application to achieve its intended purpose. Finally, performance is a measure of application execution efficiency. Each user experience property is affected by a different type of application element. Usefulness depends on application workflow, usability on the user interface, whereas performance depends on the execution environment that includes hardware and software equipment necessary to run the application.

To evaluate and adapt applications towards individual QoE preferences, QoE metrics have been introduced and algorithms for automated QoE management have been developed. Quality of service (QoS) mechanisms monitor performance-related dimensions, such as latency, availability, reliability, and price and adapt the network and application to maintain the desired level of performance [3, 4, 5]. On the other hand, GUI adaptation mechanisms based on explicit consumer rating or implicit usage tracking collect the data about task success rate, task duration, number of eye movements and mouse clicks necessary to interact with the application, and adapt the GUI to the particular preferences of the individual consumer [6, 7, 8].

However, an objective metric for evaluating the suitability of application workflow to individual consumer preferences does not exist. Workflow suitability is not an observable application characteristic, but is reflected through consumer satisfaction with application functional properties. Traditionally, the consumer delivered the application functional requirements to an application developer, who

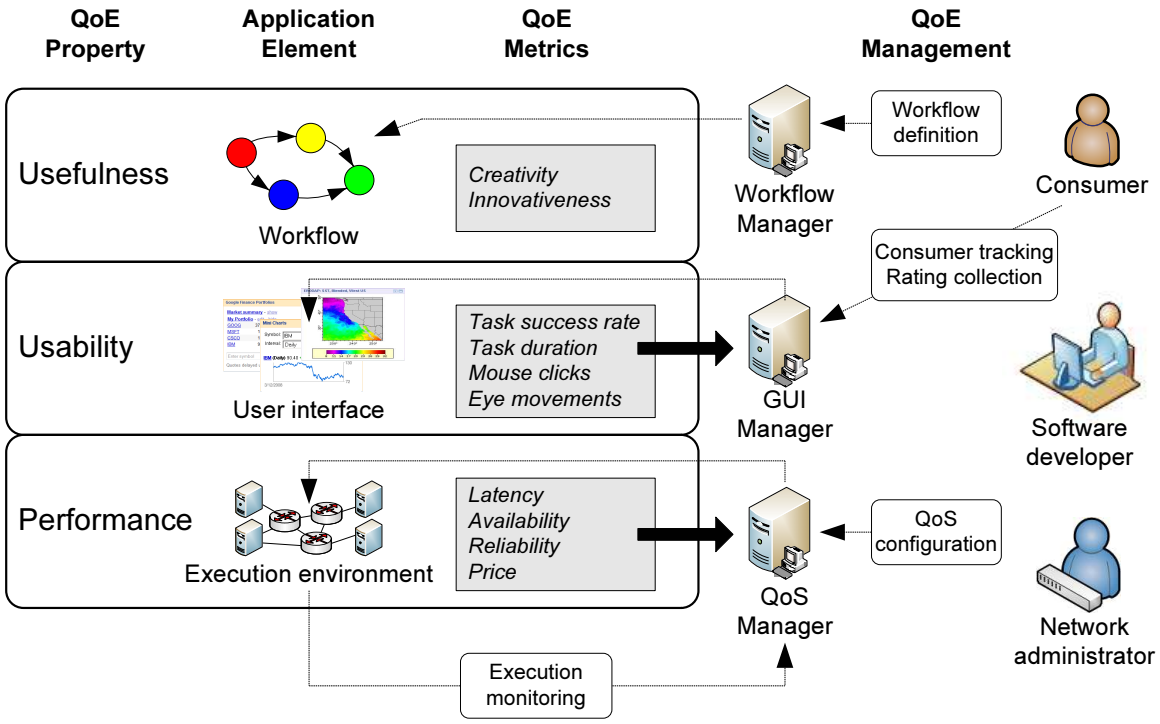

Fig. 1. Quality of user experience of computer applications

responded by implementing a workflow. However, to allow consumers to express their creativity and innovativeness on their own, we have developed a consumer-oriented workflow programming methodology based on widget networking. To measure the ammount of creativity the consumer community can deliver from widget networking, we define *application development potential*, as described in Section 6.

3 WIDGET NETWORKING

Widget networking allows consumers to build personalized computer applications out of widgets, i.e. handy and compact software components exposed as fully functional web applications displayed in a web browser. Consumers build their own applications by interconnecting independent widgets into personalized workflows that deliver new value-added functionalities. The ultimate goal of consumer-driven personalization of network applications is achieved as an end result of a multi-tiered model shown in Fig. 2.

In the lowest tier, the hardware networking platform provides facilities for interconnecting host machines, dedicated servers, and consumer devices through fixed and wireless networking. Consumer network applications are built using component-level integration and run on the hardware platform. Components are built and integrated using various technologies, such as CORBA, DCOM, JINI, and Web Services, and distributed across computing nodes. Along with providing functional correctness, one of the major issues with component integration is providing consumers with a required level of QoS. To that end, application providers design, implement, deploy, and fine-tune application performance parameters, such as processing time, data throughput, and memory consumption.

Once deployed on the consumer network nodes, the applications become accessible to consumers through GUIs. Application GUIs are built using a small, simple, and well-defined set of primitives for human-computer interaction, like buttons, textboxes, and dropdown lists, which are familiar to consumers. Over the past decade, these primitives were proven as successful abstractions for building GUI interfaces of many different types of applications. If designed in accordance with consumers' needs and expectations, GUI allows comfortable application usage. This in turn, has a positive effect on application's QoE perceived by consumers.

Although a well designed GUI is important for successful adoption of an application, QoE is about more than just GUI's structure. QoE deals with a wide spectrum of factors, such as issues related to rules of data sourcing, processing, organization, presentation, and trustworthiness [9]. These QoE parameters are built on top of QoS mechanisms that are controlled and delivered by consumer network providers. However, unlike QoS, which can be

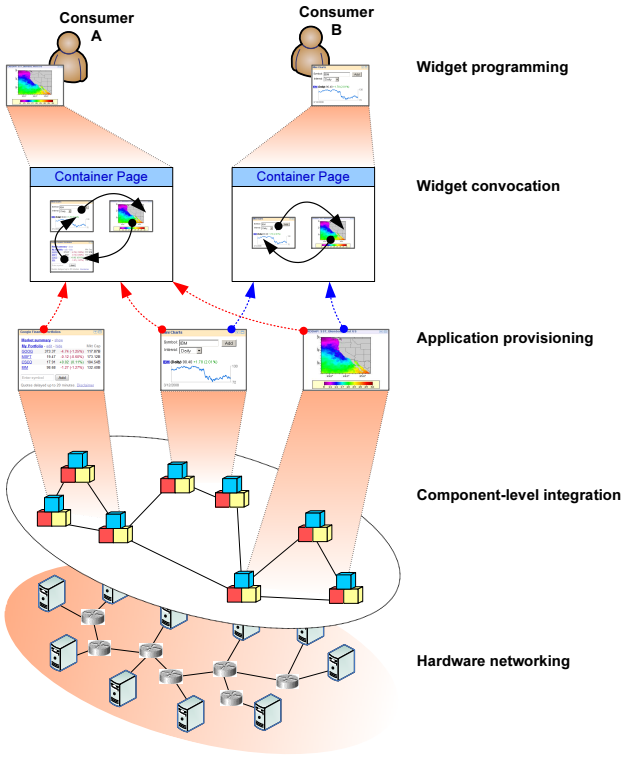

Fig. 2. A model for building consumer network applications based on widget networking

controlled and delivered by providers, QoE is assessed by consumers themselves using their subjective and individual measures [2]. To cope with the individual and subjective nature of QoE requirements, consumers must be given greater control over how applications are delivered to them. This in turn requires a higher level of consumer-controlled personalization.

A step in the direction of consumer-controlled personalization has already been taken using application containers for widgets. Widgets are derived from the idea of reusable code for development of web applications, and are fundamental elements of the component web [12]. The web pages containing a set of widgets are called container pages. Developers build widgets independently of container pages and publish their code on public servers. Developers of container pages reuse published widgets as third party modules and embed them into their container pages. Several web platforms utilize this feature to provide blank container pages and enable consumers to search for and personalize their pages with the preferred set of widgets. Examples of such web platforms are *iGoogle* with *Google Gadgets* [13], *My Yahoo!* with *Yahoo! Widgets* [14], *Windows Live Spaces* with *Microsoft Gadgets* [15], and *Netvibes* with *Netvibes Universal Widgets* [16].

Widget containers enable consumers to group their preferred widgets in a unified workspace and use them as an integrated whole. This type of consumer-controlled widget convocation assumes that consumers manually perform actions to interconnect widgets and deliver new functionalities. For example, consumers manually copy and paste data between widgets, and manually click through widget buttons to get the desired order of widget execution. Thus, the containers and widget convocation enable consumers to build QoE-enabled personalized applications. For example, in a personal financial application, the consumer might use two widgets: a stock market widget to fetch the current price for a given stock and a currency conversion widget to convert this price into the local currency. In another scenario, a consumer might use three widgets: a stock market widget to search for top movers, a company overview widget to fetch reports on company's recent achievements, and a translation widget to translate these reports into the local language. In NOAA application for analyses of oceanographic, climate, and atmospheric data, a consumer chooses a data source widget depending on the type of data that will be visualized (wind, currents, sea surface height deviation, concentration of the chlorophyll in the water, etc.), a data processing widget depending on how the results will be statistically analyzed (linear or nonlinear modeling, time-series analysis, etc.), and a presentation widget depending on how the results will be presented (table, graph chart, map, etc.).

In addition to the adjustments of application settings or variations in input parameters, QoE management also includes the possibility to compose the fragments of application logic into an arbitrary consumer application. Moreover, QoE also includes the dimensions of privacy, trust, and reputation of service providers, such as where the data are sourced from and who is processing the data.

Widgets as basic building blocks and containers as a platform for hosting widgets can be used to empower users with ability to create personalized QoE-enabled applications. Our widget programming framework enables consumers to combine preferred widgets in a personalized manner to match their QoE expectations.

4 CONSUMER-ORIENTED WORKFLOW PROGRAMMING

Although widget containers that are personalized with a preferred set of widgets provide a good foundation for tailoring the application workflow towards consumer's QoE expectations, advanced application scenarios still require an intensive manual usage of multiple widgets. To avoid repeating manual operations with a set of widgets, we have designed a framework that enables consumers to record and store the actions performed on widgets' GUIs. In addition, this framework enables consumers to edit the

recorded actions and to start their execution when necessary, similarly to macros in word processing applications.

Execution of a recorded workflow may be started either on an explicit consumer request, consumer may define the time when the workflow execution should start automatically and be repeated periodically if necessary, or consumer may define an event from the consumer network that will start the workflow execution. To be easy-to-use, the framework for recording, storing, and defining the workflow execution settings has to be based on the same set of techniques that consumers apply when manually interacting with the widgets. Therefore, we implement these functions as special-purpose widgets that consumer integrates into the application workflow.

Special-purpose widgets have to provide a programming-by-demonstration method for recording and storing of GUI actions. These widgets have to provide a calendar-like user interface to define the date, time, and repetition interval. Furthermore, they have to provide an interface with a list of events that may occur in the consumer network and allow consumer to subscribe to the selected events. In addition, these widgets have to allow consumer to define an arbitrary user interface to enter parameters, start the workflow execution, and display the final results.

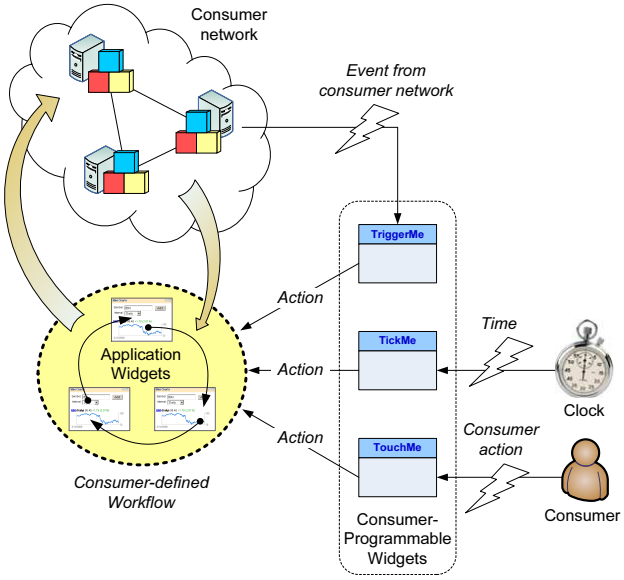

Fig. 3. Consumer programming framework based on consumer-programmable widgets

As shown in Fig. 3, we introduce three types of consumer-programmable widgets that are suitable for integration of the application-specific widgets: the *TickMe* widget, to define a time-initiated workflow, the *TriggerMe* widget, to define an event-initiated workflow, and the *TouchMe* widget, to define a consumer-initiated workflow. The *TickMe* and *TriggerMe* widgets are used to start simple workflows consisting of a single GUI action performed on a single application-specific widget. The *TouchMe* widget is used to start complex workflows consisting of multiple GUI actions performed on multiple application-specific widgets. If the *TickMe* or *TriggerMe* widget should start complex workflow, it is used in conjunction with the *TouchMe* widget, as shown in Section 5.

4.1 Time-Initiated Workflow Execution

The *TickMe* widget is used to define a time-initiated workflow. The operation of the *TickMe* widget is defined by three parameters: date and time the workflow execution should be started, repetition interval if the workflow has to be executed periodically, and the action that should be performed by the *TickMe* widget on the GUI of an application-specific widget when the workflow execution is started.

The user interface of the *TickMe* widget is split into two parts (*i* and *ii*), as shown in Fig. 4. A date is chosen through a user interface similar to a calendar or electronic organizer (*i*). To define a time and a repetition interval, we use a form-based user interface with *Time* and *Interval* input fields, respectively (*ii*). To define the action, we use the *Set action* recording button to switch the widget into programming-by-demonstration mode to record and store consumer action on the application-specific widget GUI (*ii*).

Programming the *TickMe* widget consists of five steps. Using a calendar-like interface, the consumer defines the date when the *TickMe* widget should start the workflow execution (1). Once the date is defined, the web form for defining the time, the repetition interval, and the action is opened. The time and the repetition interval are defined by filling out the web form input fields (2). A consumer defines the action by entering the programming-by-demonstration mode using a *Set action* button. In this mode, consumer performs the action that should be executed when the *TickMe* widget gets activated (4). Once programmed, the *TickMe* widget will execute the recorded action at the given date and time and repeat it with the given interval (5).

An example of programming the *TickMe* widget is shown in Fig. 4. The consumer application consists of the *TickMe* widget and an application-specific widget called *Chlorophyll* widget. The *Chlorophyll* widget consists of an image showing the concentration of the chlorophyll in the ocean and a button to refresh the image. The workflow refreshes the image on the *Chlorophyll* widget every day at the same time. This scenario is a typical use-case in the NOAA application for analyses of oceanographic, climate, and atmospheric data. Using the *TickMe* widget, consumer

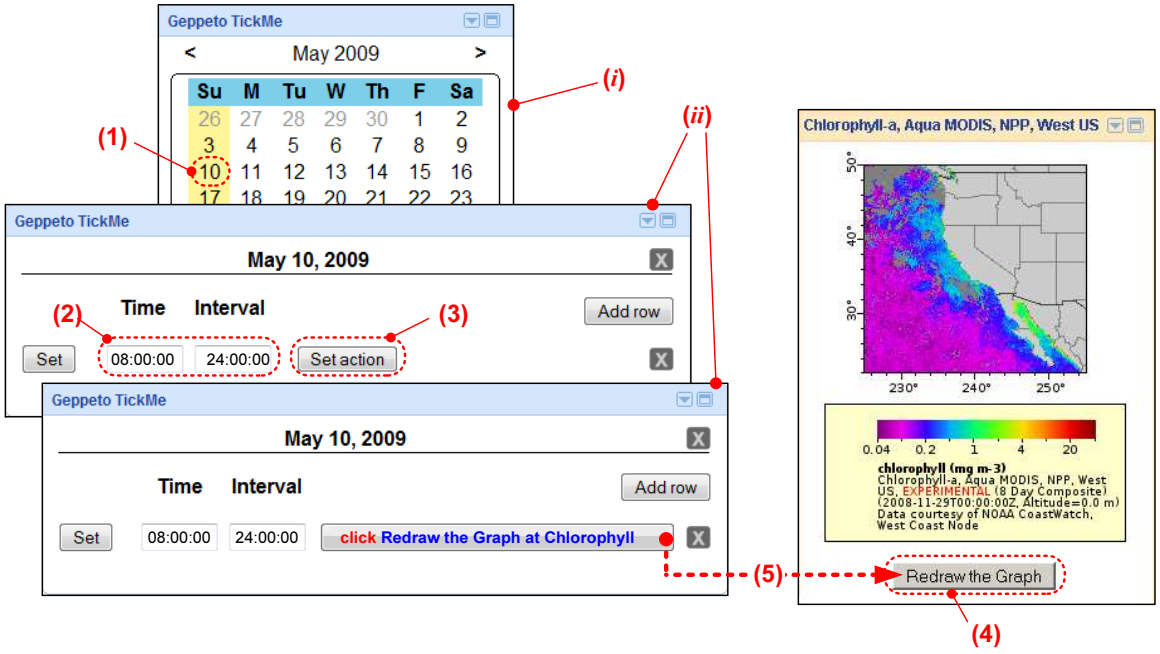

Fig. 4. Programming the TickMe *widget in* Geppeto

schedules the workflow execution for every morning at 8:00 am, beginning on May 10th 2009. The *TickMe* widget will start the workflow execution by simulating a click on the *Redraw the Graph* button on the *Chlorophyll* widget.

4.2 Event-Initiated Workflow Execution

The *TriggerMe* widget is used to define an event-initiated workflow. The operation of the *TriggerMe* widget is defined by two sets of parameters: a topic that a consumer can subscribe to and receive notifications about related events in the consumer network, and a set of actions associated with particular events that should be performed by the *TriggerMe* widget on the GUI of application-specific widgets when the workflow execution is started.

The user interface of the *TriggerMe* widget is split into two parts (*i* and *ii*), as shown in Fig. 5. A list of available topics is used to select the topic that a consumer wants to subscribe to (*i*). A list of events related to the selected topic and a set of associated *Set action* recording buttons are used to define the actions that handle the occurrences of particular events (*ii*). Similarly to the *TickMe* widget, recording buttons are used to switch the widget into the programming-by-demonstration mode to record and store consumer actions on the application-specific widget GUI.

The operation of the *TriggerMe* widget is based on two auxiliary services. The topics available in a consumer network are registered in the *Event Repository* service (e1). Using the *TriggerMe* widget, a consumer browses through the topics (e2, e3) and events related to the selected topic (e4, e5). Consumer subscribes to the selected events by defining actions (e6). Using the *Event Publisher* service, the consumer network announces the occurrences of particular events (e7), while the *Event Publisher* service forwards them to the consumer through the *TriggerMe* widget (e8).

Programming the *TriggerMe* widget consists of four steps. Using the list of available topics, consumer selects a topic to subscribe to (p1). Once the topic is selected, the list of events related to the selected topic is displayed. For each event in the topic, a consumer can define an action that should be performed by the *TriggerMe* widget when the event occurs. To define the action, a consumer enters the programming-by-demonstration mode using a *Set action* button associated with a particular event (p2). In this mode, consumer performs the action on the GUI of an application-specific widget that should be executed when the event occurs (p3). Once programmed, the *TriggerMe* widget is subscribed to the events related to the selected topic and executes the action associated with a particular event each time it occurs (p4).

An example of programming the *TriggerMe* widget is shown in Fig. 5. The consumer application consists of the *TriggerMe* widget and two application-specific widgets, called *Chlorophyll* and *Wind*. Each widget consists of an image showing the concentration of the chlorophyll in the ocean and the speed of wind, respectively, as well as a button to refresh the image. Using the *TriggerMe* widget, a consumer defines the workflow that refreshes the image

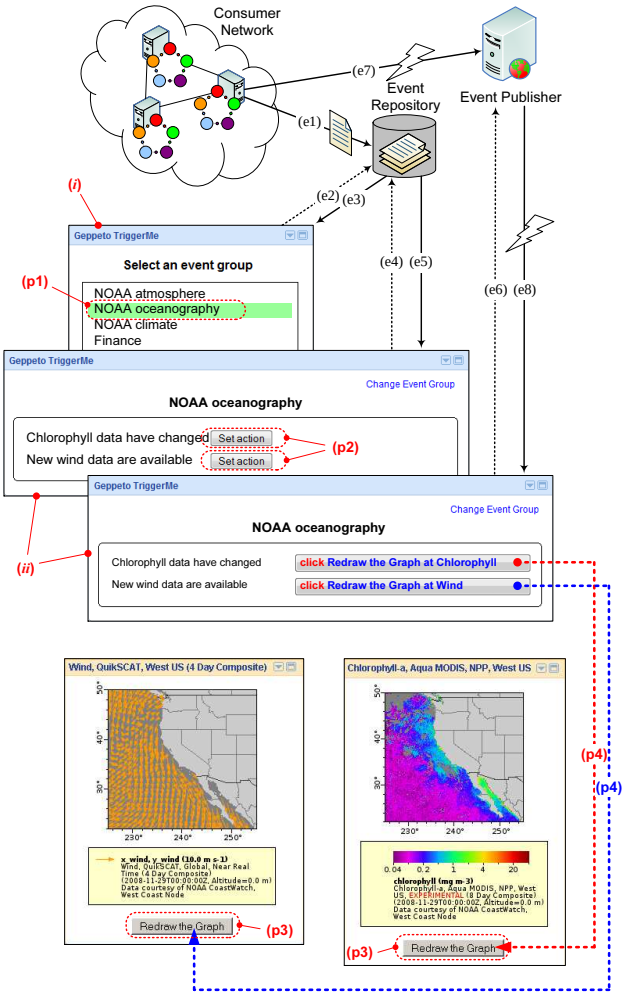

Fig. 5. Programming the TriggerMe *widget in* Geppeto

at the corresponding widget when new chlorophyll or wind data are available. The workflow is started by simulating a click on the *Redraw the Graph* button at the corresponding widget.

4.3 Consumer-Initiated Workflow Execution

The *TouchMe* widget is used to define a consumer-initiated workflow. In contrast to the *TickMe* and *TriggerMe* widgets, the *TouchMe* widget enables consumers to define workflows composed from multiple widgets and to define the GUI to control the workflow execution. Therefore, operation of the *TouchMe* widget is defined by two parameters: a consumer-programmable GUI used to control the workflow execution and a consumer-programmable logic that implements the workflow. Workflow logic consists of actions performed on the GUIs of application-specific widgets.

A consumer builds the interface of the *TouchMe* widget by assembling GUI elements from the application-specific widgets. Consumer selects a subset of GUI elements from the application-specific widgets and copies them to the *TouchMe* widget. When copying the elements, he or she chooses the elements that will be used to enter the values, activate the workflow execution, and display the final results. To copy the GUI elements from application-specific widgets to the *TouchMe* widget, consumer uses a pop-up menu, as shown in Fig. 6. When copying elements, consumer marks each element either as input, output, or an activation element. Prior to starting the workflow execution, a consumer uses input elements to enter values through the GUI of the *TouchMe* widget. At the beginning of the workflow execution, these values are transferred to the original application-specific widgets. Output elements are used to display the results of workflow execution at the GUI of the *TouchMe* widget. At the end of the workflow execution, the values from these elements are transferred from the original application-specific widgets back to the *TouchMe* widget. Finally, consumer uses the activation elements to start the workflow execution.

To program a workflow, a consumer sets the *TouchMe* widget into programming-by-demonstration mode and then performs a set of actions on the GUIs of application-specific widgets. As shown in Fig. 6, a consumer uses a pop-up menu to switch the *TouchMe* widget into programming-by-demonstration mode. On the interface of the *TouchMe* widget, a consumer selects the activation element, starts the recording, and performs the actions. The *TouchMe* widget records the actions until the recording is stopped. Once programmed, the *TouchMe* widget will repeat the recorded actions each time the activation element is clicked.

An example of programming the *TouchMe* widget is shown in Fig. 6. Consumer uses two application-specific widgets to define a personalized stock market monitor: the *Finance* widget to retrieve the current price for a given stock symbol in US dollars, and the *Currency Conversion* widget to convert the reported price between various currencies.

The interface of the *TouchMe* widget is built by copying the *Symbol* text field from the *Finance* widget as an input element (a1), the *Go* button from the *Finance* widget as an activation element (a2), and the *Price* text field from the *Currency Conversion* widget as an output element (a3). The *TouchMe* widget automatically generates the logic to transfer the value from the *Symbol* text field at the *TouchMe* widget into the original *Symbol* text field at the *Finance* widget (b1), as well as the logic to transfer the value from the original *Price* text field at the *Currency Conversion* widget into the *Price* text field at the *TouchMe* widget (b2).

To build the rest of the workflow logic, a consumer selects the *Go* button at the *TouchMe* widget as a workflow

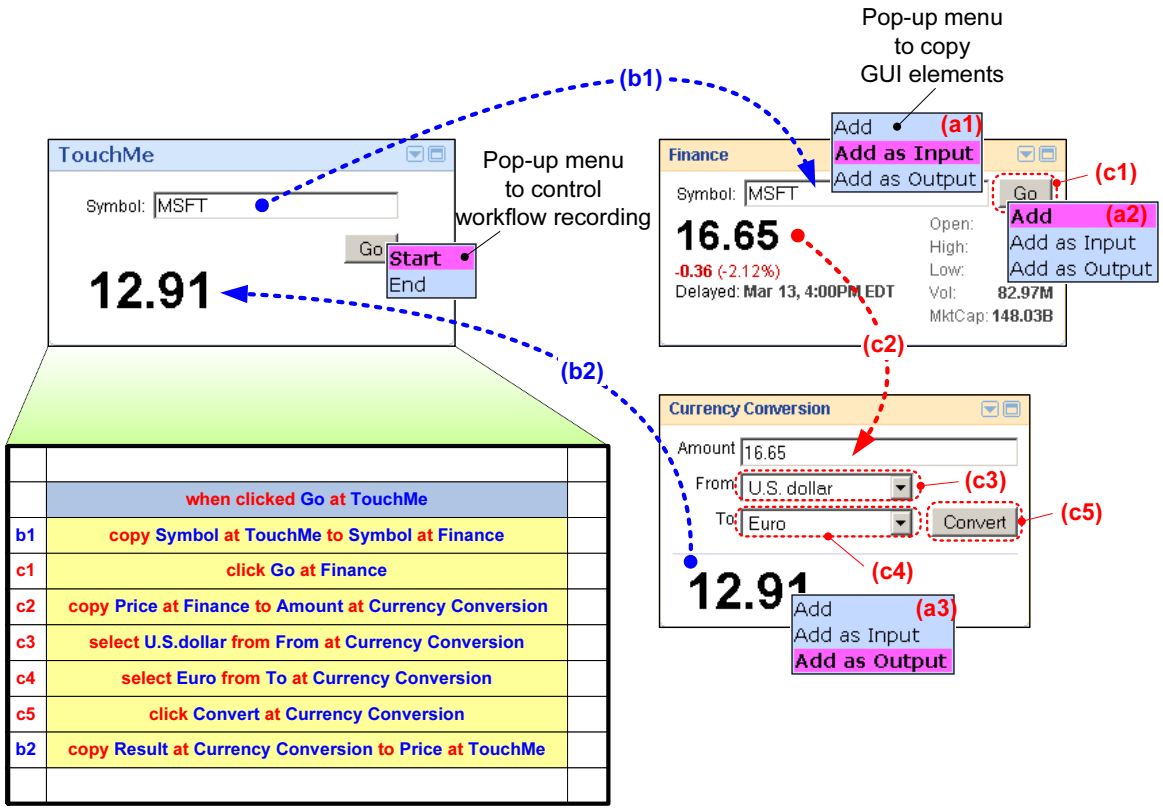

Fig. 6. Programming the TouchMe *widget in* Geppeto

activation element and starts the workflow recording. During the recording, a consumer performs a sequence of GUI actions as shown in Fig. 6: type the stock symbol into the *Symbol* text field at the *Finance* widget (note, this constant input action is not recorded, because the *Symbol* text field is used as an input element at the *TouchMe* widget) and click the *Go* button at the same widget (c1), copy the stock price from the *Finance* widget and paste it into the *Amount* text field at the *Currency Conversion* widget (c2), select the source and target currency in the *From* and *To* drop-down menus at the *Currency Conversion* widget (c3, c4), and click the *Convert* button on the same widget (c5). Each time a consumer clicks the *Go* button at the *TouchMe* widget, the *TouchMe* widget transfers the values from the input elements to the application-specific widgets, executes the recorded actions, and transfers the results from the application-specific widgets to the output elements on the *TouchMe* widget.

Consumer actions in the user interfaces of application-specific widgets are stored into a 2D table that is a part of the *TouchMe* widget. In our example, four types of actions are stored: ***when clicked***, ***copy to***, ***click***, and ***select***. An example of the first type of action is

```
when clicked Go at TouchMe
```

where ***when clicked*** are keywords denoting the action that postpones the workflow execution until a consumer makes a click at the activation element, while *Go* ***at*** *TouchMe* is a parameter denoting the activation element *Go* in the *TouchMe* widget. An example of the second type of action is

```
copy Price at Finance
to Amount at Currency Conversion
```

where ***copy*** and ***to*** are keywords denoting the copy/paste operation that transfers the data from the *Price* text field at *Finance* widget to the *Amount* text field at the *Currency Conversion* widget. An example of the third type of action is

```
click Convert at Currency Conversion
```

where ***click*** is a keyword denoting the operation of clicking the *Convert* button in the *Currency Conversion* widget. Finally, an example of the fourth type of action is

```
select Euro
from To at Currency Conversion
```

where ***select*** and ***from*** are keywords denoting the operation of selecting an item from the *To* drop-down menu at the *Currency Conversion* widget, while *Euro* is the menu item to be selected.

Representation of GUI actions within the 2D table reduces the complexity of managing the workflow to a simple spatial arrangement of these actions within the table cells. Each cell of the table is either empty or contains a GUI action. Time progresses from left to right and from top to bottom simultaneously. If two non-empty cells are adjacent, the actions associated with the cells are executed sequentially. On the other hand, if two non-empty cells are separated with at least one empty cell, they can be executed in any order, including in parallel. Organization of actions within the 2D space enables a simple construction of sophisticated workflow patterns for sequential and parallel execution of GUI actions. In the example shown in Fig. 6, the consumer workflow consists of one sequence of non-empty adjacent cells representing a workflow with sequential execution.

Once recorded and represented in a tabular form, a consumer can optionally use advanced functionalities, such as edit, add, and delete actions, as well as rearrange their spatial organization to construct arbitrary workflow patterns. In addition, a consumer can save the logic and the GUI of the *TouchMe* widget, recover a previously saved widget or reset the widget. Finally, consumer can use multiple tabs to construct multiple GUIs for the *TouchMe* widget, as well as manage the visibility of the application-specific widgets used in the workflow. For example, consumers may choose to hide some or all of the application-specific widgets to leave only the *TouchMe* widget visible. This can significantly reduce the visual overhead imposed by a number of application-specific widgets.

5 CONSUMER NETWORK APPLICATION EXAMPLE

In this section, we demonstrate the integration of all three types of consumer-programmable widgets into a personalized consumer network application. We present an application for advanced stock market monitoring. The goal of the application is to get an overview of the desired stock information once a day, but also on consumer request if necessary. In addition, if more significant changes happen to the stock on the market, an alert is sent to a consumer via SMS.

Fig. 7 illustrates the workflow of this application. In addition to the *TouchMe*, *TickMe*, and *TriggerMe* widgets, a consumer uses three application-specific widgets. The *Finance* widget is used to retrieve the current stock information for a given stock in US dollars. The *Currency Conversion* widget is used to convert a given amount of money between various currencies. The *SMS Sender* widget is used to send SMS messages to a given phone number.

Using the *TouchMe* widget, a consumer implements the workflow that is composed of the *Finance* and the *Currency Conversion* widgets to retrieve the stock price in the desired currency. This part of the workflow is defined in the same way as shown in Fig. 6. The consumer initiates the execution of the workflow by clicking the *Go* button on the *TouchMe* widget. Furthermore, the consumer uses the *TickMe* widget to autonomously refresh the stock price every morning at 8:00 am. The *TickMe* widget is programmed to execute a click on the *Go* button of the *TouchMe* widget. Finally, the consumer uses the *TriggerMe* widget to send an SMS alert if the value of the MSFT stock drops by 5% or more. The *TriggerMe* widget is programmed to execute a click on the *Send* label at the *SMS Sender* widget.

The table representing the GUI actions recorded within the *TouchMe* widget consists of two sequences. The upper sequence represents the actions to retrieve a price for a given stock in a desired currency using the *Finance* and *Currency Conversion* widgets. Execution of this sequence is started either by the consumer or by the *TickMe* widget. The lower sequence is used to configure the *SMS Sender* widget with a phone number and message text, and to send out the alert. Execution of this sequence is started by the *TriggerMe* widget. In addition to the four types of actions described in the previous section, this sequence contains a fifth type of action used to enter a constant value into a text field. An example of this type of action is

```
type "+385 1 234 56 78"
into To at SMS Sender
```

where ***type*** and ***into*** are keywords denoting this type of action, *"+385 1 234 56 78"* is a value to enter, while *To* ***at*** *SMS Sender* is a parameter denoting the target text field.

6 APPLICATION DEVELOPMENT POTENTIAL

To quantify how widget-oriented programming improves the creativity in software development, we define *application development potential* (*ADP*). ADP is an estimation of the total amount of new applications a community of developers can produce in a given time frame using a specific programming methodology.

The time required to develop a piece of software depends on the size of software and the effort included into software development (e.g. the amount of man power). The usual method used to estimate the size of a software product during project resource planning is *Functional Size Measurement* (*FSM*) [17, 18]. FSM is an internationally recognized and ISO standardized technique for measuring the size of software based on its functional requirements. Since software functional requirements are independent of any constraint of how the software is built, FSM enables to estimate the software size regardless of the tools, techniques, and technologies used to build the software. Using FSM, the size of software is normalized into a number

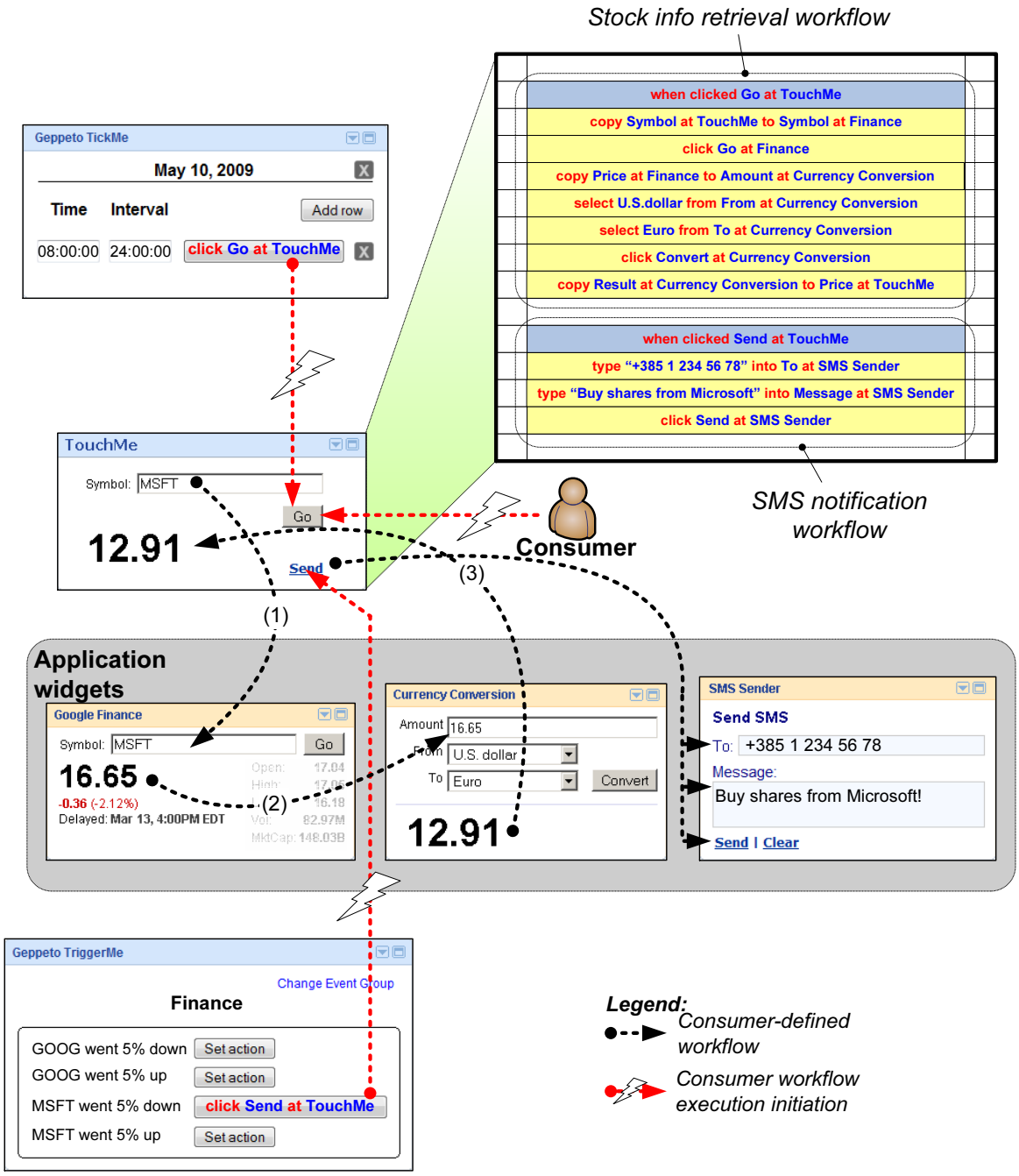

Fig. 7. Personalized consumer network application for advanced stock market monitoring

of Albrecht's function points, according to Allan Albrecht who developed the method in the late 1970s. The time required to develop an application is, therefore, proportional to the number of Albrecht's function points and the time necessary to develop a single Albrecht's function point. On the other hand, the time required to develop a piece of application normalized into an Albrecht's function point depends on the programming paradigm and tools being used during application development. That time is proportional to the lines of code necessary to develop a single Albrecht's function point in a given programming language.

Table 1 shows the application development potential of widget-oriented programming compared to object-oriented and domain-specific programming paradigms. The first row of Table 1 shows estimated number of lines of code (LOC) required to develop a single Albrecht's function

point (FP) using various programming paradigms [19]. Object-oriented programming typically requires a few dozen lines of code, depending on the language being used. Domain-specific languages and tools, such as spreadsheets and database query languages, require five to ten lines of code per Albrecht's function point. Such tools improve the efficiency of software development by an order of magnitude as long as they are used within the intended application domain. On the other hand, widget-oriented programming, which is based on reusing and composing coarse-grained and fully functional application blocks into consumer-specific workflows, reduces the time required to develop an application by a further order of magnitude. Since the program logic to handle one user input or one client-server transaction are typical examples of Albrecht's function points, widget-oriented programming allows implementation of the whole function point using a single or a couple of lines of code.

Table 1. Application development potential of different programming paradigms

	Object-oriented programming	**Domain-specific languages and tools**	**Widget-oriented programming**
LOC/FP	20-70	5-10	1-3
Number of users	20 million professional programmers	500 million end-user programmers	2 billion web consumers
ADP	1-3	50-100	700-2000

In addition to the time necessary to develop an application, second dimension used to estimate the ADP is the total number of people within a particular community with the ability to use the particular programming paradigm. The second row of Table 1 shows an estimated number of various groups of users who will participate in software development in 2010 worldwide. According to [20, 21], there will be around 20 million professional programmers with the ability to use object-oriented programming and 500 million end-users who will use domain-specific languages to develop software supporting their work. At the same time, there will be around 2 billion regular web consumers with the ability to use widgets and widget-oriented programming [22]. Moreover, the number of web consumers is still growing rapidly, especially in Asia.

Based on these projections, we calculate the *application development potential* as the number of Albrecht's function points developed by the whole community in a certain time unit. The calculation of ADP is described by

$$ADP = \frac{number\ of\ users}{function\ point\ development\ time}. \quad (1)$$

The *function point development time* used in Equation (1) is an approximation of the time required to develop a single Albrecht's function point, which is proportional to the lines of code per Albrecht's function point.

The third row of Table 1 shows the comparison of ADP indices among object-oriented, domain-specific, and widget-oriented programming. ADP indices show that in a time frame in which the professional programmer community produces 1 to 3 new applications, the end user community can produce 50 to 100 applications, while the consumer community can produce up to 2000 applications. In terms of ADP, widget-oriented programming outperforms the object-oriented and domain-specific programming by one to three orders of magnitude.

7 CONCLUSION

This paper presents *Geppeto*, a framework for building applications from widgets by consumers not trained in programming, but with an understanding of available functionality on the consumer network. The main goal is to provide a seamless environment for widget composition built on usage patterns familiar to consumers. Our assumption is that consumers understand the problem space they work on and what the available widgets provide. *Geppeto* strives to give them the ability to build new personalized QoE-aware applications important for solving their particular problems, while at the same time exposing newly created functionality to new potential consumers.

The work of *Geppeto* is based on workflows of widgets. *Geppeto* is the first framework that uses consumer-programmable widgets to program the operation of application-specific widgets. Using consumer-programmable widgets, *Geppeto* enables consumers to define the workflow among application-specific widgets and to start it on request, schedule it in time, or define an event from the consumer network that will trigger its execution. While programming the workflow in *Geppeto*, consumers use the same set of techniques as during the manual operation with widgets.

Our methodology allows ordinary computer consumers to participate in software development. To quantify how widget-oriented programming improves the creativity in software development, we define *application development potential* (*ADP*). ADP is an estimation of the total amount of new applications a community of developers can produce in a given time using a specific programming methodology. In terms of ADP, widget-oriented programming outperforms object-oriented and domain-specific programming by one to three orders of magnitude, providing the community with more innovative ideas and technical solutions than isolated groups of professional programmers or domain-oriented end users ever could.

Geppeto is implemented as an extension of the *Apache Shindig* project [23]. *Apache Shindig* is an open source

widget container and widget rendering server. It was originally started by *Google* as a reference container for hosting *OpenSocial* and *Google Gadgets* compatible widgets. Any *Google Gadgets* compatible widget can be used as an application-specific widget within the *Geppeto* widget container. We extended *Apache Shindig* with consumer-programmable widgets and a widget repository to store and share the widgets among consumers. To try out the tool, users can visit the project's web site at http://geppeto.fer.hr.

ACKNOWLEDGMENT

We were fortunate that a research team lead by Roy Mendelssohn, Bob Simons, and Lynn DeWitt from National Oceanic and Atmospheric Administration (NOAA), California, who were trying to develop a QoE-enabled application based on widget workflow, contacted us and recognized our work on consumer-programmable widgets. Many thanks on their valuable discussions on how to improve the architecture and usability of the *Geppeto* framework and for being such gracious hosts at the picturesque NOAA facility in Pacific Grove. The authors also wish to acknowledge the support from the Ministry of Science, Education, and Sports of the Republic of Croatia through research project no. 036-0362980-1921 "Computing Environments for Ubiquitous Distributed Systems" and Google, Inc. for the Google Research Award project "End-User Tool for Gadget Composition". Furthermore, the authors wish to thank Ivan Žužak, Miroslav Popović, Vladimir Klemo, Marin Silić, Goran Delač, Jakov Krolo, and Ivan Budiselić from Faculty of Electrical Engineering and Computing at University of Zagreb, Ivan Skuliber and Ivan Benc from Ericsson Nikola Tesla Zagreb, Franjo Plavec, Ivan Matošević, and Davor Čapalija from University of Toronto, Boris Debić from Google, Inc., and Dalibor F. Vrsalović for their help with preparing this manuscript. Finally, many thanks to student members of our research projects who participated in the implementation of the *Geppeto* framework.

REFERENCES

[1] T. Tullis, B. Albert, **Measuring the User Experience: Collecting, Analyzing, and Presenting Usability Metrics**, Morgan Kaufmann, 2008.

[2] P. H. Bloch, F. F. Brunel, T. J. Arnold, **Individual Differences in the Centrality of Visual Product Aesthetics: Concept and Measurement**, Journal of Consumer Research, vol. 29, no. 4, pp. 551–565, March 2003.

[3] I. Foster, M. Fidler, A. Roy, V. Sander, L. Winkler, **End-to-End Quality of Service for High-End Applications**, Computer Communications, vol. 27, no. 14, pp. 1375–1388, September 2004.

[4] R. Chakravorty, S. Kar, P. Farjami, **End-to-End Internet Quality of Service (QoS): An Overview of Issues, Architectures and Frameworks**, Proceedings of the ICIT 2000, December 2000.

[5] A. Campbell, C. Aurrecoechea, L. Hauw, **A Review of QoS Architectures**, ACM Multimedia Systems Journal, vol. 6, no. 3, pp. 138–151, May 1998.

[6] T. Strandvall, **Eye Tracking in Human-Computer Interaction and Usability Research**, Lecture Notes in Computer Science, vol. 5727, pp. 936–937, August 2009.

[7] M. Nakayama, M. Katsukura, **Assessing Usability with Eye-Movement Frequency Analysis**, Proceedings of the 2008 Symposium on Eye Tracking Research & Applications, pp. 77, 2008.

[8] Y. Lin, W. J. Zhang, **Evaluating Interface Usability Based on Eye Movement and Hand Movement Behavioral Parameters**, The 47th Human Factors and Ergonomics Society Annual Meeting, pp. 653–657, Denver, Colorado, USA, October 2003.

[9] D. Ribbink, A. C. R. van Riel, V. Liljander, S. Streukens, **Comfort Your Online Customer: Quality, Trust and Loyalty on the Internet**, Managing Service Quality, vol. 14, no. 6, pp. 446–456, 2004.

[10] S.-S. Liaw, **Understanding User Perceptions of World-Wide Web Environments**, Journal of Computer Assisted Learning, vol. 18, no. 2, pp. 137–148, June 2002.

[11] H. Chen, R. Wigand, M. Nilan, **Optimal Experience of Web Activities**, Computers in Human Behavior, vol. 15, no. 5, pp. 585–608, September 1999.

[12] R. Guo, B. B. Zhu, M. Feng, A. Pan, B. Zhou, **Compoweb: A Component-Oriented Web Architecture**, Proceedings of the 17th International Conference on World Wide Web, pp. 545–554, Beijing, China, April 2008.

[13] Gadgets API, Google Inc., http://code.google.com/apis/gadgets, accessed 21 Oct. 2009.

[14] Konfabulator Reference Manual, Yahoo! Inc., http://manual.widgets.yahoo.com, accessed 21 Oct. 2009.

[15] Windows Live Gadget Developer's Guide, Microsoft Corporation, http://dev.live.com/gadgets/sdk/docs/default.htm, accessed 21 Oct. 2009.

[16] Netvibes Developers Network, Netvibes Ltd., http://dev.netvibes.com/doc/start, accessed 21 Oct. 2009.

[17] G. C. Low, D. R. Jeffery, **Function Points in the Estimation and Evaluation of the Software Process**, IEEE Transactions on Software Engineering, vol. 16, no. 1, pp. 64–71, January 1990.

[18] D. Cleary, **Web-Based Development and Functional Size Measurement**, Proceedings of the IFPUG Annual Conference, San Diego, USA, 2000.

[19] L. H. Putnam, W. Myers, **Measures for Excellence: Reliable Software on Time, Within Budget**, Prentice Hall, 1991.

[20] M. Rasalan, **Global Developer Population and Demographic Study 2009 v1**, Evans Data Report, April 2009.

[21] C. Scaffidi, M. Shaw, B. Myers, **Estimating the Numbers of End Users and End User Programmers**, Proceedings of the 2005 IEEE Symposium on Visual Languages and Human-Centric Computing, pp. 207–214, 2005.

[22] Internet Usage Statistics, Internet World Stats, http://www.internetworldstats.com/stats.htm, accessed 7 Nov. 2009.

[23] Apache Shindig, The Apache Software Foundation, http://incubator.apache.org/shindig/index.html, accessed 7 Nov. 2009.

Siniša Srbljić is currently a professor at the Faculty of Electrical Engineering and Computing, University of Zagreb, and head of the Consumer Computing Laboratory. His career also spans Silicon Valley where he worked on large-scale distributed systems at AT&T Labs. He was visiting the University of Toronto, where he worked on the NUMAchine multiprocessor project, and the University of California, Irvine. His research interests include consumer Web computing and widget-oriented programming. In teaching, he is involved in the theory of computing, programming language translation, service-oriented computing, and network middleware systems.

Dejan Škvorc is a Ph.D. candidate at the Faculty of Electrical Engineering and Computing, University of Zagreb, Croatia. He received his B.Sc. degree in 2003 and M.Sc. degree in 2006, both from School of Electrical Engineering and Computing, University of Zagreb. During the 2007, Dejan Skvorc spent four months as a software engineering intern in Google's Mountain View office with Google Gadgets group. He is a coauthor and one of the architects of the inter-gadget communication framework based on publish-subscribe paradigm - the PubSub framework. His research interests include service-oriented architectures, end-user development, and consumer programming.

Daniel Skrobo is a solution architect at Ericsson Nikola Tesla d.d., Zagreb, Croatia. He received his Ph.D., M.Sc., and B.Sc. degrees from the Faculty of Electrical Engineering and Computing, University of Zagreb (FER Zagreb). Currently he is working on development of health-care records storage and analysis systems. He held research assistantship position at FER Zagreb and was a research engineering intern at Google's Mountain View office, CA, USA. His engineering and research interests are program translation systems and service-oriented computing systems.

AUTHORS' ADDRESSES

Prof. Siniša Srbljić, Ph.D.
Dejan Škvorc, M.Sc.
Department of Electronics, Microelectronics, Computer and Intelligent Systems,
Faculty of Electrical Engineering and Computing,
University of Zagreb,
Unska 3, HR-10000 Zagreb, Croatia
emails: sinisa.srbljic@fer.hr,
dejan.skvorc@fer.hr

Daniel Skrobo, Ph.D.
Ericsson Nikola Tesla,
Krapinska 45, HR-10000 Zagreb, Croatia,
email: daniel.skrobo@ericsson.com

Received: 2009-11-02
Accepted: 2009-11-27