Runtime User Interface Design and Adaptation

David England  
School of Computing and Mathematical Sciences  
Liverpool John Moores University  
Liverpool, UK  
+44 151 231 2271  
D.England@ljmu.ac.uk

Martin Randles  
School of Computing and Mathematical Sciences  
Liverpool John Moores University  
Liverpool, UK  
+44 151 231 2280  
M.J.Randles@ljmu.ac.uk

A. Taleb-Bendiab  
School of Computing and Mathematical Sciences  
Liverpool John Moores University  
Liverpool, UK  
+44 151 231 2489  
A.TalebBendiab@ljmu.ac.uk

ABSTRACT

In this paper, a method of generating appropriate user interfaces at runtime is investigated. It is proposed to use the established formalism of Situation Calculus to describe and specify user interfaces. It is shown how specific features of the formalism provide many desirable properties in the design and specification of user interfaces that are adaptable to context and composed at runtime. The formalism provides a provably correct deployment, whilst giving a means of deliberation on the optimum configuration that is directly compiled through a developed Neptune scripting language. The major features of the formalism and programming language are described together with an illustration of how this has been used in an implemented e-health case study for decision support with partner institutions in breast cancer care. It is shown how pluggable decision models may be introduced and system adaptation to clinician context achieved, whilst system integrity is maintained.

Categories and Subject Descriptors

D.2.10 [Software Engineering]: Design – methodologies, representation.

General Terms


Keywords

user interface description, situation calculus, runtime adaptation

1. INTRODUCTION

At present user interfaces are most often tailored to individual software systems, at design time, to enable interaction with the systems functions. The advent of ubiquitous computing environments, multiple device interactions and increasing accessibility has introduced much complexity into the provision of user interfaces; as the range of contexts, devices and formats exceeds the designers’ capability to programme every eventuality for the system. As complexity increases it becomes very difficult, if not impossible, to implement user interfaces through the hard coding of all the requirements for heterogeneous device adaptation to the user’s situation. It is now necessary to promote the on demand runtime fabrication of interfaces to match users’ specific contexts (device type and capabilities, location, interaction style, etc.) in a timely and relevant manner for the underlying functional systems.

This paper addresses the problems of user interface definition and production, at runtime using an e-health system as an example domain, via an established layered observation system [19] that takes a particular logic model of the system (Situation Calculus), uniquely able to handle counterfactual reasoning, to produce a deployable user interaction application through an associated scripting language for adaptation (Neptune). This takes an abstract observed model of user interaction and transforms it into an actual working interface. In this way developers are freed from the chore of having to develop separate user interfaces for multiple versions of devices, users or contexts: Rather an abstract initial model is specified, which is reasoned upon, through the Situation Calculus logic, that subsequently supports the autonomous generation of the user interfaces, in the Neptune scripting language, tailored to circumstance. This has the added advantage that the design of the interface accurately converges on an ideal instance the more the user interacts with the system. In addition this work addresses many of the issues for a User Interface Description Language (UIDL); in particular specification, communication, portability, adaptation, platform independence, context awareness and multimodal support: In essence situation calculus is used as a UIDL, which is then run time adaptable through its Neptune implementation. Additionally this approach goes beyond these normal bounds in allowing adaptation to novel circumstance and continuous adaptation to optimize the interface with an associated UIDL model always available for subsequent reuse or independent development.

In short the work proposes an UIDL that includes a meta-UIDL to reason on outcome and context. Thus the paper continues by detailing current related work on UIDLs in Section 2. Section 3 gives an overview of the observation and component relations in the runtime production of the interface. Section 4 presents the Situation Calculus reasoning engine on the abstract high-level initial model, whilst Section 5 reports on the implementation details of Neptune. Section 6 presents a case study based on systems utilised in a clinical setting (dental triage or 2rich decision support) with Section 7 concluding the paper.

© The Author 2009.  
Published by the British Computer Society

HCI 2009 – People and Computers XXIII – Celebrating people and technology 463
2. RELATED WORKS

There are many proposed solutions for user interface design based on a UIDL and software based on this UIDL. These most often entail a static model where all possible productions of the user interface are predefined in some way at design time, most often using XML. In contrast this paper proposes a logic approach enabling cognitive observer agents to assemble and deploy the most suitable user interface at run time. This may be achieved because the logical descriptions of the system do not demand a completely predefined state space; instead logical sentences describe what is true in the system and outcome follows through reasoning on actions and logical entailment. The increasing availability and variety of interactive devices has highlighted the need for interface design to be more generic and adaptable to its supporting device and circumstances. This leads to a more intuitive and natural interactive experience for the user and supports Reality-Based Interaction [9]. XML-based representations predominate where logical descriptions of requirements are translated into a user interface description languages such as XIML [18], UIML [11], TERESA@XML [15] and USIXML [10]. These languages, however, do not seek to render the user interface based on user needs or context but rather describe the structure of the interface and the interactions of the interface elements. There is also a lot of interest in bringing interface design into the system model at design time using established software engineering techniques: UML [4] is based on the UML and allows the design and analysis of user interfaces to be achieved through modelling. Similarly Data Flow Diagrams may be used for the user interface task model, as in Diane+ [23], for example. Further works in modelling user interfaces are available through JUST-UI [14], SUPPLE [7], Teatlach [8] and Wisdom [17], whilst model refinement and intention models are used in [3] and [2] respectively. These models capture various features of the user interface requirements based on the given descriptions; generating the user interface automatically based on the task model. Task analysis, however, is only one aspect of user interface design and tasks may become available that are not envisionned within a design time model.

Runtime user interface provision and adaptation, which is the goal of this work, seeks to utilize an extensible model of the application and the requirements and environments of its users. Then, by updating its own internal model with functional requirements, interactions and environmental data, a logic layer selects, infers or adapts the most appropriate interface design. In this way the interface is aligned with a computationally perceived context.

2.1 Providing a User Interface based on Context

Context is often captured by software to provide personalization services to users, and to mould the experience of using the service to the particular requirement of the user [5]. Oftentimes this is facilitated through an authentication model where a user provides credentials to identify themselves within the system, and their personal experience is reproduced and tailored based on stored criteria. Such authentication is widely used within the Internet community to provide forums for discussion, to access personal information, and to digitally sign documentation.

In effect, by providing authentication criteria, a context of the request is formed as additional, meta-level descriptions can be attached to the profile of an authenticated user. For stricter control, by applying the techniques of role-based authentication, users can be placed into groups with specific responsibilities and requirements. Context can therefore provide the first stage of determining user interface functions within a distributed e-health system, where assessed conformance requirements, patient profiles, and access to evolved decision models can be planned according to individual profiling.

In the context of clinical care, different organisations and institutions have different conformance models, which any decision process needs ratification against, NICE guidelines [16], for example. Other organisations do not require such ratification. Through context, a request can be scrutinized accordingly, providing validation for the decision process. As such, there exists a hierarchical model for conformance based on context, as illustrated in Figure 2.

![Figure 1. User hierarchy in a typical clinician scenario.](image)

Accordingly, the individual concerns must conform to the requirements set out by the role in which the individual exists within the organisation. This is then supplemented by the requirements of the organisation as a whole. Thus, to govern adaptation of requirement, each conformance model defined above the requirement level in the hierarchy must ratify the change.

3. A RUNTIME USER INTERFACE

This section describes the observation system and component relations in the process of producing the user interface. In order to give some context to our discussion on models for runtime adaption we will present the case study of post-operative breast cancer diagnosis. In an earlier paper [6] we discussed how decisions on post-operative breast cancer treatment are currently governed by a set of medical guidelines including the National Institute for Clinical Evidence (NICE) guidelines [16] in which the decision process is as follows (Figure 1): The oncologist uses post-operative data to determine a risk category. The risk category is then used to determine the possible treatment options, T1 to T6. However a number of factors make it difficult to provide an acceptable electronic support system in this context.

Ideally we would like such as system to incorporate current medical knowledge and aid clinicians in decision making by providing:

- Improved access to evolving medical knowledge to the clinical team
- An evidence-based approach decision making [22]
- Performing additional supporting tasks, i.e. writing prescriptions
- Warning of potential problems with treatment decisions
4. LOGICAL AND REASONING MODEL

This section describes the implementation procedure for a model control service based in the middleware to reason on and produce the most suitable and adaptable user interface.

4.1 The User Interface Description Language

For a number of reasons Situation Calculus with an extensible representation of system knowledge, ideal states and action sequences [20, 21] is used as a User Interface Description Language, in this work, to provide the major specification formalism and reasoning mechanisms. Firstly the action-based semantics of the language provide an in-built description for every available user interactive action and system-generated event; unpredictable environmental events are also expressible in the formalism, at runtime, through action histories. Secondly the effects of user interactions are predictable through the use of successor state axioms; providing a context and prediction for the consequences of action choices: Uniquely, counterfactual reasoning with branching timelines is permitted, thus reasoning may proceed, completely automatically, based on “what-if” scenarios. Thirdly, there is no requirement for a complete state enumeration and transition model; rather what is true in the system can be logically stated reasoned upon and updated whilst behaviour follows by logical consequence: The Current circumstance (situation), for the production of a user interface, is conceived as a causal action (event) history. Fourthly, properties of the specification can be proved entirely within the logic, whereas other formalisms require a separate mechanism to prove correctness properties of the interface deployment. Fifthly, the user interface, described in Situation Calculus, is directly implementable through Neptune scripts [12], which are runtime generable and adaptable; allowing rapid uptake and updating of decision models with runtime reasoning to incorporate current clinical knowledge with historical data in an integrated, fully audited and provably correct manner.

4.2 Implementing the User Interface

Although full details of the implementation are outside the scope of this paper, the specification and adaptation of the runtime user interactions are coded and executed directly from the Situation Calculus into Neptune [12]; further details are available in the given references. To summarize, the implementation is executed through the Cloud architecture; the federation of services (component agents) and resources, with appropriately derived user interface descriptions. It is defined to enable the autonomic framework to function as a User Interface production module using the specially developed language, Neptune which allows management objects to be compiled and inspected at runtime. A system space provides persistent data storage for service registration and state information giving the means to coordinate the application service activities into an object model and associated User Interfaces based on the recorded interaction model and functional requirements. Reasoning can then proceed based on the Situation Calculus model, whereby the user interface descriptions are derived, inferred or adapted. Neptune exposes policies and decision models for system governance, derived from the Situation Calculus/Extensible Decision model, as compiled objects that can be inspected, modified and executed at runtime. Thus the system can evolve as modelled by the logical specification in a safe and predictable manner giving the adjustable self-management required. Neptune objects are executed on demand through an event model exposed by the Cloud architecture.

Figure 2. Decision process

However, there are a number of complicating factors which make the task more difficult and which advanced run-time adaptability may support, for example

- As medical knowledge advances the underlying decision support data needs to be updated in a rigorous and cost-effective way
- There are a number of staging techniques used to categorise tumours
- The staging techniques may have their own protocols and guidelines
- The views and aims of NICE may differ from those of clinicians
- We need to provide governance mechanisms to monitor and record treatment decisions without undermining clinical judgment
- Decisions may be affected by local and national resource limitations

This is not an exhaustive list (see [13]) but gives an indication of the scope of the problem. Within our project we were also examining prognostic decision making by comparing current treatment data with historical data so that current treatment decisions can be triangulated with past decisions – allowing for possible changes in treatment regimes over the time period. Thus an adaptable run-time user interface in this context needs to provide:

- Pluggable decision trees which can be updated as medical knowledge advances and is validated for staging decision-making
- A flexible decision making engine which can use one or more sources to triangulate, say, current clinical data with historical data
- Pluggable views, to give different presentations of the results from current and historical models.
- Auditing mechanisms that record decisions and warn of possible out-of-range decisions.

Such requirements are difficult to specify in traditional state based approaches as it is impossible, at design time, to foresee or predict the emergence of new practices or decision models and their affect upon existing, still relevant, components. For this main reason it is necessary to consider formalisms based in mathematical logic so that, at any particular point in time, the circumstances apparent in the system can be derived reasoned upon and the most appropriate user interface displayed for the perceived situation.
The system controller with an associated Observation System controls access to and from the individual services and resources within the Cloud. It brokers requests to the system, through the contrived User Interface, based on system status and governance rules, in Neptune objects, derived from the deliberative process as stated above. An overview of the Observation system is shown in Figure 3.

![Figure 3. The Observation system](image)

Each service and resource when it first registers itself to the Cloud sends a meta-object serialized from an XML definition file. This meta-object contains the properties and state data of the service it is describing and is stored within the System Space at registration. Each service maintains its own meta-object and updates the System Space when changes in state occur. The XML definition file contains all information required for the Cloud to discover the service through registration contained in the service element and prepare the appropriate User Interface. In addition to the meta-objects exposing properties of a service within the Cloud, they also describe the interface events that can be fired, caught and handled, allowing multi-modal interfaces to be composed. The event model begins by the service informing the System Controller when an event is fired, which itself marshals this event to the System Space to provide the appropriate scope. It should be noted however, that the event model is abstracted from the components within the system, and is controlled by the Neptune scripting language that sends and receives the appropriate event calls to the controller. The Neptune scripting language is structured in terms of rules, conditional statements and variable assignments that are translated from the Situation Calculus specification to software system objects, encapsulating all the logical inference processes and variable instantiations for the production of the most relevant interaction model and associated interface. An overview of this process is shown in Figure 4.

![Figure 4. User interface production at runtime](image)

In this way the base rules for deliberation to control the Cloud architecture, through enhanced user interaction, have been transcribed, from the Situation Calculus reasoned representation, into Neptune objects that can be modified as a result of Observation System deliberation on system events.

5. RUNTIME ADAPTATION for the USER INTERFACE

Neptune has been developed to solve many of the outstanding issues highlighted in current research into runtime analysis and adaptation, including the runtime production of the most suitable user interface for the perceived context. Neptune can enable full runtime adaptation for autonomic self-management and user interface production by addressing the issues previously identified, indicating that the solution should sit above and be separate from any single language used to produce language components. In effect, the components are hidden from Neptune by way of a semantic interface: Neptune need only be able to dispatch calls defined within the semantic interface, rather than needing to directly host and manage the component itself. In this way, a bridge between Neptune and a base language is required both to facilitate communication between the base language and Neptune so that the language can instruct Neptune and interpret results from it, but also so Neptune can dispatch a call that is required to be handled by the base language. Figure 5 shows a schematic view of the method in which Neptune communicates with base language components.

![Figure 5. Base language interface](image)
These calls are marshalled either directly to the higher Neptune Framework for processing, or situated as Neptune Base Language Objects (NBLOs), a semantic form that describes the intentions and conditions needed by a base language component to render the derived user interface. This process is illustrated in Figure 6.

![Figure 6. The Neptune framework](image)

Whilst the system can facilitate adaptation in its use of Neptune and CA-SPAs to represent the governance of the system, it is the ability of these methodologies to represent autonomic behaviour that is of particular interest to adaptation processes.

By capturing evolving requirements during the lifetime of the application, all the constructs exist so that seamless evolution of the system occurs based on the requirements of its users. This evolution is monitored and controlled by the self-governance specified through the system, to provide validation and boundaries for the system that in turn are fed back to the user via the interface. Hence patient, clinician, organizational, and external requirements can all be represented and enacted to produce a stable system that takes all requirement into account.

An early example of these techniques has been produced, detailed in the following case study. By analyzing the usage patterns of a system user, the interface can itself adapt to speed up the process of using the system, by tailoring its process to match that of its user. Thus, options rarely used by the user are hidden, and stages can be skipped based on usage. This is illustrated in Figure 7, where to get from step A to step B within the process, a set of stages are required.

![Figure 7. A staged process](image)

Based on a situation, which is entered when the user decides “No”, in the above process, for 5 subsequent times, these additional steps are removed from the process creating a direct path from A to B. Provision exists within the interface to restate the original process flow, if required at a later stage, as does the threshold in which steps are removed from the process (in this case, 5 subsequent uses) as this can adapt as required. In addition, patient concerns are expressed an anxiety scale where documentation for a patient is produced based on the assessed anxiety (and therefore requirement) of the patient.

### 6. CASE STUDY

The following section outlines how Situation Calculus, Neptune and the techniques described in this paper were used to produce an adaptable, distributed user interface to a guideline model for a breast-cancer decision support system as part of the 2nrich research project [19]. The project represented collaboration between computer scientists, statisticians and clinicians from Liverpool John Moores University, the Christie Hospital, Manchester and the Linda McCartney Centre of the Royal Liverpool Hospital to provide decision support for post-operative breast cancer care. A major concern of the project was that clinicians’ decisions ought not to be supplanted, but rather supported and adapted to, by such a system. Thus it was necessary for the system to possess a separated cognitive meta-system to reason on the behaviour of the system as a whole, including the clinician’s actual decision. This facilitated the three stated aims of the project: Supporting the clinician’s decision within the National Institute for Health and Clinical Excellence (NICE) guidelines; the analysis of historical data to produce new rules for treatment choice and a combination of the previous two notions to provide a decision support system that is adaptive to clinicians’ needs and changing environments.

The Neptune IDE (Integrated Development Environment) provides a graphical interface to allow flow models to be edited, and a text editor for logical statements to be written from the situation calculus UIDL. NeptuneScript, the language used to encode the situation calculus logical statements, is based on the notion of IF and THEN statements and familiar English semantics to provide readability. Figure 8 shows a screenshot of the Neptune Flow Editor, where several flow objects have been dropped onto the page and linked to provide a notion of flow. Tools exist that allow step-by-step walkthroughs of the possible routes available through the model, so that errors in logic, such as flows that repeat and loop indefinitely, can be identified and rectified accordingly. In summary, by dropping these shapes, linking them, and encoding the logic behind the model, a complete decision process can be created and adapted externally from a software system. The software then executes this model, and behaves according to the model giving an initial model of user interaction that is refined and adapted as the system evolves. While the project encompasses much research, a major requirement for the system was that it should adapt its operation to provide the most appropriate user interface based on the user and their circumstances leading to a classification of patient treatment based on historical evidence to improve the level of care given to patients.
In addition, government targets concerning the care of cancer patients, including waiting times, and budget concerns needed to be considered. Consequently, a guideline decision process published by National Institute of Clinical Excellence (NICE), UK [16] was implemented in Neptune and was used as a basis upon which to present choices and treatment options for patients through the adaptable interface. Theoretical management requirements were introduced so that certain treatments were no longer available and that given a choice between treatments, the cheaper would be preferable. While the NICE guidelines remained static, these new requirements were added as new decisions to supplement the original model and update the user interface, thereby adapting its behaviour. In response, the model and software no longer recommended expensive treatment plans and opted for cheaper alternatives if available.

As an example, Figure 9 shows a section of the NICE guidelines for operative breast cancer care, encoded within Neptune. Object serialisation techniques, allowing an object to be stored, transferred and retrieved from information repositories, can thus be applied to Neptune objects, allowing the safe and efficient distribution of decision models. In this way, Neptune allows the modification and enhancement of the decision process during any point of its lifetime whilst maintaining efficient transportation and storage. By representing both the situation and required behaviour to enact upon the system entering the situation in a policy form, a defined boundary of behaviour is produced to derive the user interface options. In other words, a model for user interactions is achieved. Basing these representations within Neptune, each stage of the user experience can be adapted as required, using the introspective nature of the language. 

```
define rule calnpi1
    variables.sd = 0.2 * dataset.tumou usize +
    dataset.paramstatus + dataset.HLSCoMpi1
    if (variables.sd < 5,289)
        variables.npi = 1
    else
        variables.npi = 1
    end if
end define
```

Figure 9. A NICE Guideline Rule Encoded in Neptune

In particular local decision models may be inferred from the actions of the clinicians; thus an institutional decision model may be made available based on the combined clinical decisions at a specific location or a personal decision model may be customised to reflect an individual clinicians decision preferences: An example interface is shown in Figure 10 where the risk calculation is inferred from the code shown in Figure 9. These decision models are then easily pluggable into the decision interface.

The developed methodology produces a policy model, based on a situation, action ontology, and a predicted situation that the system should find itself within after execution of the action ontology. This final predicted situation is important to ensure that the intention of the policy has been completed and gives a context for user interface descriptions. To demonstrate this model efficiently, the earlier example of selecting a treatment plan against the conformance hierarchy can be modeled as a policy. The only situation, in which a treatment plan can be accepted, is when the treatment plan conforms to the organisation’s own guidelines, and for the purposes of illustration, a set of government guidelines.
These guideline models are themselves encapsulated within Neptune objects: organisationGuidelines and governmentGuidelines respectively.

As such, the situation for the CA-SPA is thus:

\[
\text{If (organisationGuidelines.Conform} \land \text{governmentGuidelines.Conform)}
\]

When this situation is entered, we can accept the treatment plan, as specified within the ontology of the policy:

\[
\text{patientTreatmentPlan.Accept}
\]

Giving the predicted situation, after the action ontology is executed, to represent the fact a treatment plan has been accepted for the patient:

\[
\text{Patient.TreatmentPlan} \neq \text{Nothing}
\]

In addition, as Neptune provides a formal model for the user interface construction, within a decision process, when a patient profile was executed against the NICE guidelines, comprehensive historical data were stored that could later form the basis of evidence given to machine-learning and data-mining processes. This study of usage enabled deviations from guidelines by clinicians to be audited and presented accordingly. Another consequence of automating the storage of patient details and their path through the decision model was that patients modelled in the prototype could return to the hospital at a later date, and clinicians would be presented with a tailored interface containing both the patient details and the exact reasoning behind the decisions of the information system, essential in ascertaining correct patient care.

6.1 Application of Techniques to the Wider Field of e-Healthcare

Whilst the scope of this paper has focused on supporting breast cancer prognosis through enhanced user interaction, the techniques described herein can equally be applied to other areas of e-Healthcare as shown by the production of a prototype decision support system for dental triage services, for the Royal Liverpool Hospital using the Neptune and Situation Calculus policy methodology. The original system used by the Royal Liverpool Hospital combined a mixture of Microsoft Word and Microsoft Excel documents to provide flow through the process of triage for patients. In this way, information elicited from the patient formed the basis of decisions on their perceived health. The more critical the condition (such as persistent bleeding) the higher priority the patient was given to see a dentist. The main requirement of the case study was to produce a system that could log patient details and their eventual classification by the system, with the end goal being that the system could adapt its classification based on historical evidence to improve the level of care given to patients forming additional user interaction models. Furthermore, government targets facing the care of dental patient, included waiting times, which needed to be updated and assessed to feedback impacts to the user of any chosen decision.

By encoding the process flow within a Neptune object, an instance of the object could be created for each patient as they passed through triage. Consequently, the decision process, or rather, why the patient received the classification they did, is encapsulated within the object instance itself. Storing this object instance, along with the patient’s profile, comprehensive historical data was stored that could later form the basis of evidence given to machine learning and data mining processes for improved user information.

7. CONCLUSION

This paper has described the major features of a process whereby runtime adaptable user interfaces were provided for clinicians to interact with a breast cancer care decision support system. This allowed doctors to move between centres whilst maintaining their individual preferences, integrity and context for decision making. The provision of pluggable decision models, maintaining consistency between current practice and historical data, giving perspectives based on current context and the full auditing of decision procedures, with appropriate annotation of unexpected outcomes, were all achieved through adaptable user interface design specified in Situation Calculus as a User Interface Description Language. The description was then directly implemented through NeptuneScripts sitting above the base language of the support systems. It is also noted that the Neptune flow editor provides many starting points for the process depending on the system user: User interfaces are coded from a graphical input of the flow model by a clinician, with a formal model available as an output. Alternatively a system designer may input a formally specified model and be able to work through a graphical representation of the flow model, for demonstrating the process to clinicians, with the NeptuneScript code automatically generated.

This result represents an early proof of concept for such an approach to User Interface Description Languages in this field and much further work is planned to scale up and evaluate empirically the solution for this and larger more complex systems. Indeed major studies are required to investigate the acceptance and usability of such systems by expert clinicians: Initial inputs from clinicians, into this work, showed a high level of scepticism regarding the value of automated decision processes in medicine and clinical care pathways. This work, to some extent, addressed these issues by presenting clinicians with a more integrated personalised toolset, achieved by adaptation through and of their user interface. Additionally work is ongoing producing a case tool to automatically code adaptable user interface designs based on observation and deliberation of the underlying system from the Situation Calculus descriptions. Later work will also seek to include multiple modalities aligned with clinician preference, user type/role or optimised input.

8. REFERENCES


