Interaction Modeling in PRACTICE

CTT Vs. SCXML - A Comparison of Two Practical Solutions Applying Interaction Modeling Techniques for Multimodal User-System Interaction

Miroslav Sili, Matthias Gira, Markus Müllner-Rieder and Christopher Mayer
AIT Austrian Institute of Technology GmbH, Health & Environment Department, Biomedical Systems, Vienna, Austria

Keywords: Interaction Modeling, User Interaction, Concur Task Trees, CTT, Statecharts, SCXML, Multimodality, Avatar based Systems, Adaptivity.

Abstract: Nowadays, we are surrounded by various devices to interact with digital media and services. Each device and its in- and output modalities can support users' abilities differently. Thus, it is important to cover a wide range of interaction devices. Modeling user interaction instead of modeling single user interfaces customized to the device is a starting point to do so. This work targets the comparison of two different user interaction modeling techniques used for the design of multimodal user interfaces. Next to the general concepts of the two interaction modeling techniques, the corresponding execution frameworks and the practical exploration results are presented. This paper summarizes advantages and disadvantages of each approach and the comparison clarifies that the CTT approach applied in AALuis is more applicable for large and complex user interaction scenarios. The SCXML approach applied in the ibi project is more suitable for lightweight and structurally simpler user interaction scenarios.

1 INTRODUCTION

Model-based user interfaces and user interaction modeling has been a widely discussed research area since the early 90s. Inspired by the very first commercially available user interface builders, interface toolkits, and user interface management systems researchers started to design, develop and evaluate new techniques in order to continuously automatize different steps of the user interface design process (Janssen et al., 1993), (Puerta et al., 1999). Research works in this early stage formed the foundation for model-based user interfaces and their adaptivity, but the majority focused on single and desktop based applications. Nowadays, we do not use just one single point of access to interact with digital media, but we are surrounded by computing systems in form of mobile devices like tablets, smartphones and wearables. A conventional stationary computer, for example, represents just one of many nodes in the Human Computer Interaction (HCI) field. To cover all different kind of interaction devices at once, we need to start modeling user interactions instead of modeling single user interfaces.

This work targets the comparison of two different user interaction modeling techniques used for the design of multimodal user interfaces. Interpretation engines for both techniques have been implemented in separate executions frameworks and both techniques have been evaluated in different Ambient Assisted Living (AAL) projects. This paper targets the comparison of two modeling methods and their evaluation and implementation results from the technical point of view. User involvement results accomplished during the project trial phases are out of the scope for this work.

2 METHODOLOGY

In the beginning, we will provide a general overview about the two selected interaction modeling techniques (section 2.1). The first is based on Concur Task Trees (CTT) (Paternò et al., 1997) and the second is based on State Chart XML (SCXML) (Barnett et al., 2007). To be able to evaluate these interaction modeling techniques, we have developed two execution frameworks (section 2.2). The first one reflects the prototype built within the project AALuis (Aaluis.eu, 2015) and the second one

Sili M., Gira M., Müllner-Rieder M. and Mayer C..
Interaction Modeling in PRACTICE - CTT Vs. SCXML - A Comparison of Two Practical Solutions Applying Interaction Modeling Techniques for Multimodal User-System Interaction.
DOI: 10.5220/0005481502430250
In *Proceedings of the 1st International Conference on Information and Communication Technologies for Ageing Well and e-Health* (ICT4AgeingWell-2015), pages 243-250
ISBN: 978-989-758-102-1

reflects the prototype built within the project ibi (ibi.or.at, 2015). By using these two prototypes, we are able to identify and to evaluate necessary tasks that user interaction developers need to fulfill in order to generate interaction models, to embed and connect these interaction models in the specific framework and finally to interpret them by the execution process during runtime (section 2.3).

2.1 CTTs and SCXMLs as Modeling Methods for User Interaction

The literature relevant in this field mentions a couple of projects using model-based user interface generation approaches (Mori et al., 2004), (Peissner, et al., 2012), (Popp, et al., 2013), (Brambilla et al., 2014). Some rely on CTTs, some on Statecharts (Harel, 1987) and some e.g. on the Business Process Model and Notation (BPMN) (Zur Muehlen, et al., 2008).

2.1.1 The CTT Interaction Model

CTT is an XML-based formal notation to represent task models. It is of hierarchical structure, with graphical syntax. CTT focuses on activities to be executed by users or systems to reach a certain goal. CTT distinguishes between system, user, interaction, and abstract tasks. System tasks are executed by the (software) system alone (e.g., data processing). User tasks represent internal cognitive or physical activities performed by the user of the system (e.g., selecting a problem solving strategy). Interaction tasks are user performed interactions with the system. Abstract tasks are used for composition of task groups in the hierarchical structure of the CTT. The notation provides an exhaustive set of temporal operators, which express the logical temporal relationships between the tasks.

CTTE (Mori et al., 2002) is a tool for the design and analysis of CTTs. This allows creating and editing task trees in a graphical way. The tool also provides a CTT simulator for runtime behavior analysis.

2.1.2 The Statechart Interaction Model

SCXML is an event-based state machine language. It combines concepts from Harel State Tables (Harel, 1987) and Call Control eXtensible Markup Language (CCXML) (W3.org, 2015a), (Romellini et al., 2005). SCXML is widely used for user interfaces and dialog management in many different fields such as AAL, cloud based services or video games (Almeida et al., 2014), (Dragert et al., 2013), (Jeong et al., 2012). It inherits semantics and special features like compound states and parallel states from Harel State Tables and combines it with event handling and the XML representation of CCXML. SCXML is used to describe finite state machines (FSM). A FSM is a mathematical model with a finite number of states where only one state can be active at any given time, which is called current state.

Basic concepts in SCXML are states and transitions, with an event attached to each transition. When a concrete event is fired and the corresponding source state is active, the target state will become active and the source state inactive. The active state can be queried continuously. In the context of user interactions, states represent current dialogs or windows and their transitions concrete user or system actions. Using these techniques a user or system action can evoke a state change. In the ibi prototype, this change invalidates the previously presented user interaction dialog and activates a newly generated user interaction dialog. The state machine can either be created directly in XML notation or generated by using a GUI based tool such as scxmlgui (Code.google.com, 2015a). SCXML interpreters are available in various programming languages such as in Java (Team Commons, 2015), C++ (Code.google.com, 2015b) or Python (GitHub, 2013).

2.2 Execution Frameworks for Interaction Models

2.2.1 AALuis Execution Framework

The AALuis execution framework is an OSGi-based (Alliance OSGi, 2003) flexible middleware layer. The framework dynamically generates user interfaces for connected services that provide CTT-modelled interactions (Mayer et al., 2014). The framework's architecture consists of plug-in based components, which are described in the following:

Figure 1 illustrates modules and the communication flow in the AALuis execution framework. The dialog manager component orchestrates the process from abstract service description and data, to the concrete interface presented for a context specific interaction step. Service managers mediate between service endpoints and the dialog manager. Similarly, device managers act as brokers between the devices and the dialog manager.

The dialog manager administers all interactions between the users and the system. For each

interaction session, this component repeatedly evaluates the interaction status, generates and presents a user interface, applies user input, and re-evaluates the interaction status. The dialog manager stores the execution statuses of multiple service CTTs per session, but only one interaction can be in the foreground. Not only the dialog manager, but also the service itself can initiate interaction. Through signaling of tasks of a certain type, the interaction flow is interrupted in a first-come-last-served manner.

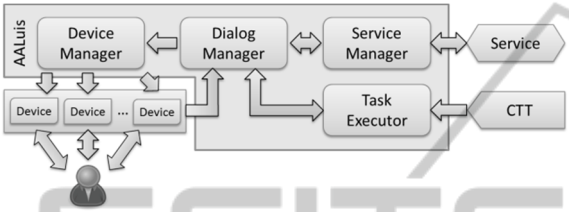

Figure 1: Modules and communication flow in the AALuis execution framework.

The task execution component handles the run-time execution of the CTT. It (a) processes an execution state, in the form of a tree of en/dis-abled states, (b) updates one state, and (c) evaluates the CTT temporal operators to reach a new execution state. Based on these execution steps, the currently actable interactions are used by the UI transformation component to generate a user interface.

Adoption of the standard rendering, and addition of new kinds of interaction, can be achieved in two ways: Either a completely new task-type is introduced into the CTT, or the interpretation of an existing type is refined in the transformation XSL files.

Similarly, formerly unknown output modalities can be added to the execution framework. This entails using the new modality in the XSLT files used for generating the user interface. Output modalities and device types that are already registered can be included and excluded on-the-fly. The framework also provides the possibility for developers to create media conversion plug-in components that enable automatic conversion of one output modality to a new one.

2.2.2 Ibi Execution Framework

The ibi execution framework is a middleware, which connects services handling data, and modalities handling in- and output. These services and modalities can be exchanged and extended to create a tailored solution for ubiquitous applications. The framework contains multiple managers, which are essential for the operation of the system. These are the dialog manager, the modality manager and the service manager.

As depicted in Figure 2, the dialog manager is the central part of the ibi execution framework. It reads the SCXML definitions and parses them into SCXML objects. Dialog objects are created based on SCXML objects. Each represents a single statechart machine. Multiple statechart machines can be used at the same time. The dialog manager distributes the incoming events to all of them.

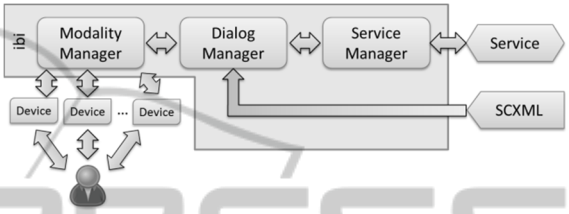

Figure 2: Modules and communication flow in the ibi execution framework.

The dialog manager is logically placed between the service and the modality manager. The service manager contains a list of registered services and sends requests from the dialog manager to the respective services. The modality manager decides which modalities are available and suitable for the current situation and distributes output requests from the service manager to them. The modality manager is also responsible to receive user input from a registered device and to distribute it towards the dialog manager. Modalities are connected to the ibi framework by a so called Connector implementtation. Each Connector is written for a specific in- and output device because of different APIs, interfaces or technologies. Therefore, for each new modality a separate connector implementation has to be developed.

Every service is represented by a state machine, where each state and transition can trigger an action in the service. Initially loaded state machines remain in this initial state until the first trigger event. This event can be timer triggered, user input triggered or externally triggered (e.g., incoming event via HTTP). Each state machine contains a priority value. State machines with higher priority are able to interrupt lower priority machines, which are resumed when the high priority machine finishes.

The SCXML definitions are able to carry additional data within every state. In the ibi approach, this additional data fields are used for the definition of special template-based treatments in the UI generation process. Templates may be defined, e.g., in form of XHTML (W3.org, 2015b) including

XForms (W3.org, 2015c). This allows customizing the generated user interfaces if needed.

2.3 Embedding and Connecting Interaction Models

2.3.1 Integration of CTTs into the AALuis Execution Framework

The task description in CTT is the central document that services have to provide to the AALuis execution framework. It describes the interaction between the service and the user. Additionally, a binding definition, associating the services' business methods to the tasks in the task description, needs to be submitted. These documents can be provided in two ways, depending on the residence of the service (external or internal).

External services are based on Web Service technology. They are not located in the AALuis layer itself nor deployed in the AAL middleware AALuis is deployed in. They are dynamically transformed and bound to an internal OSGi service representation, making them accessible by the layer. The CTT is provided by an URI that either points to a web resource or a local file. Internal services are based on OSGi technology. They have to be embedded into the same AAL middleware, and follow certain conventions. The OSGi bundle has to declare the bundle as AALuis service in the manifest and to contain at least the above mentioned documents. Similar to external services a representation is created and used by the dialog manager. With these choices the service designer can concentrate efforts on the service and the planned interaction patterns alone. Encouraged by a very clear separation of concerns, the service designer does not need to think about the design of the elements that are forming the interaction, which is done independently by a UI designer.

New services can be integrated on-the-fly. Either by making the layer aware of a new external service endpoint or deployment of an internal OSGi bundle - also during runtime.

2.3.2 Integration of SCXMLs into the Ibi Execution Framework

The service and an accompanying SCXML file have to be developed to integrate a new service into the ibi execution framework. The SCXML file may also contain XForms templates for each state. New template types need a new interpretation on the layer side, which means new implementations in form of source code have to be performed. This blurs the responsibilities because neither the service designers (who generate SCXMLs), nor the UI designers (who generate templates and their implementations) should be forced to modify and extend the framework. The ibi approach needs a new strategy to support templates and their interpretations without the need to extend the framework by additional implementations.

A service can be created either as internal or external service. Internal services are integrated into the ibi executable, while external services interact via Representational State Transfer (REST) calls (Richardson et al., 2008). To accomplish this, an external service manager is implemented as an abstraction layer between the external service and the service manager. This abstraction allows that external services register as internal services. This facilitates a consistent access to both kinds of services.

3 RESULTS

This chapter provides a comparison of advantages and disadvantages of the presented approaches based on eight evaluation categories a) Generation of the interaction model, b) UI customization and UI extensibility, c) Concurrency, d) Separation of concerns, e) Modality and device extensibility, f) Integration of new services, g) Tool for the interaction design, and finally h) Modality conversion and alternative modalities. The comparison focuses on applicability aspects from the technical point of view. Concrete user evaluation settings and user involvement results are outside of the scope of this paper and are separately presented on the 17th International Conference on Human-Computer Interaction (Sili, 2015).

3.1 Detail Comparison based on Eight Evaluation Categories

3.1.1 Generation of the Interaction Model

The interaction model in the AALuis approach supports the generation of complex interaction scenarios. The CTT notation allows a high degree of flexibility using temporal operators allowing task processing in different orders, e.g., sequential, concurrent or interrupting. Unfortunately, this flexibility requires additional efforts when learning the CTT notation.

In contrast, the SXML interaction model in the ibi approach is easier to learn. It is composed by just two elements, namely states and transitions. The disadvantage of this simplicity is the limitation when modelling complex interactions. Developers need to define numerous transitions between single states to compensate missing constructs for concurrency and interruption. This makes the system impracticable for rich interaction scenarios.

3.1.2 UI Customization and UI Extensibility

In the AALuis approach, a new UI element (e.g., a calendar widget or captions) is created by defining a new task type in the interaction model. The framework transforms this new task type into an abstract XML element which is finally rendered using XSL rules. Both extensions, the definition of the new task type as well as the XSL rules, can be defined independently and outside of the AALuis framework. This allows extensibility which does not require to rebuild or to recompile the AALuis framework.

In the ibi prototype, UI customization is achieved by defining new templates within the interaction model states. These templates are filled with concrete data during the final rendering process. The framework supports the interpretation of different templates but it is limited to the generation of new UI elements. Interpretations of new template types require new implementations and therefore to rebuild the ibi framework.

3.1.3 Concurrency

The AALuis framework supports concurrent interaction models, which allow multiple user-system interaction dialogs. The current implementation is limited in prioritization of these concurrent interaction models. Because each dialog owns the same priority level, the framework is not able to distinguish between more important dialogs (e.g., security warning) and less important dialogs (e.g., incoming social network message). To overcome this ambiguity, the priority responsibility in the AALuis approach is shifted to the service side. The service has to decide which dialog is prioritized higher and which dialog has to be postponed due its low level priority.

The ibi approach supports concurrent interaction models and different priority levels. Each interaction model is clearly assigned to a priority level. High priority dialogs are able to interrupt low priority dialogs. These are resumed once the higher priority level dialogs are finished. Dialogs with the same priority level are served in the First In First Out (FIFO) order.

3.1.4 Separation of Concerns

The separation of concerns is clearly achieved in the AALuis approach. UI designers, on the one hand, are responsible for extending and maintaining XSL rules for new UI elements, new branding styles and new device types. Service designers, on the other hand, are responsible for defining, maintaining and connecting their services. Therefore, UI designers do not need to be involved during the service integration and oppositely service designers do not need to take care about UI related topics.

The ibi approach has blurred responsibilities between service and UI designers. Extensions on both sides require to rebuild and to recompile the ibi framework, respectively. Therefore, the UI designer is not able to extend the framework without the explicit involvement of the service designer and vice versa.

3.1.5 Modality and Device Extensibility

Both systems are able to keep track about available and connected devices and modalities, but the extensibility is handled in a different manner. In the AALuis prototype, new modalities are included by applying new XSL transformations. In some cases, these new modalities require also new devices (e.g., modality for Braille lettering). These devices can be included and/or excluded on the fly, because AALuis-enabled devices are able to automatically describe their capabilities towards the AALuis framework. Furthermore, the framework has a complete overview about available and connected devices. A concrete device can be addressed, depending on the required modality.

In the ibi approach, new devices are not able to describe themselves automatically. Each new device requires a connection in form of a concrete source code implementation on the framework side. Once this work is done, the framework will be able to address the specific connected device depending on the required modality.

3.1.6 Integration of New Services

The AALuis prototype is able to dynamically connect SOAP-based external and OSGi-based internal services. Therefore, service designers do not need to extend the framework in form of new source code implementations.

New services in the ibi approach require a

concrete source code implementation. As a result of this new implementation, the whole framework has to be rebuilt. On the other hand, new implementations are not restricted to a specific communication method. A lightweight and easy to deploy REST-based communication method can be used in the ibi approach.

3.1.7 Tool for the Interaction Design

In both prototypes external graphical tools can be used to design the interaction model. The tool used in the AALuis approach has a built-in simulator, which allows an early pretesting of the interaction model.

The graphical tool used in the ibi prototype does not provide a simulator. Considering the lightweight complexity of SCXML elements (states and transitions) a simulator would not be able to provide more information than the design tool.

3.1.8 Modality Conversion and Alternative Modalities

An avatar-based user-system interaction was one of the main requirements in both projects. Therefore, both approaches include a built-in text to avatar modality conversion. Beside this, both systems currently do not support an additional automatic modality conversion. The AALuis prototype supports alternative modalities (e.g., a textual representation of an image) but these alternative elements need to be provided by the service.

In the current implementation the ibi prototype does not support alternative modalities.

3.2 Summary of Evaluation Results

Table 1 summarizes the evaluated advantages and disadvantages of the presented techniques and provides a comparison between them. Positive aspects are marked with a '+' sign, neutral aspects are marked with a 'o' sign and negative aspects are marked with a '-' sign.

3.3 Examples of Generated UIs

Figures 3 and Figure 4 illustrate examples of generated User Interfaces for the TV device. Both approaches use avatars as additional modality and support for the user. The current TV device implementations in both approaches support a remote control based navigation and control. While the AALuis approach supports only arrow based navigation (up, down, left, right), the ibi approach also supports a number based navigation.

Table 1: Comparison between the AALuis and ibi approach (legend: +..positive aspect, o..neutral aspect, -..negative aspect).

	AALuis / CTT	**ibi / SCXML**
Generation of the interaction model	+ Supports the generation of complex interaction scenarios - Efforts needed to learn the CTT notation	- Limitations in the complexity - Easy to understand and easy to learn
UI customization and UI extensibility	+ Supports the definition of new task types and therefore new UI elements + Supports new task type interpretations by external XSL rules	+ Supports template definitions within every state - Interpretations of new template types require new implementations
Concurrency	- Layer supports concurrency of multiple CTTs without prioritization of interactions	+ Layer supports concurrency of multiple SCXMLs and priority definitions
Separation of concerns	+ Clear separation between service designers and UI designers	- Blurred responsibilities between service designers and UI designers
Modality and device extensibility	**o** New modalities require new XSL transformation rules + Devices and modalities may be included and/or excluded on the fly	- New modalities require new implementations - New devices require new implementations
New service integration	+ New services can be integrated on-the-fly + Support of internal or external SOAP based web service	- New services require new implementations + Support of internal or external REST based web service
Tool for the interaction design	+ GUI for the design of CTTs + Simulator allows an early pretesting	+ GUI for the design of SCXMLs - Simulation and pre-testings are not possible
Multimodality conversion and alternative modalities	+ Built-in text to avatar modality conversion - Support for alternative modalities	+ Built-in text to avatar modality conversion - New modality conversions require new implementations

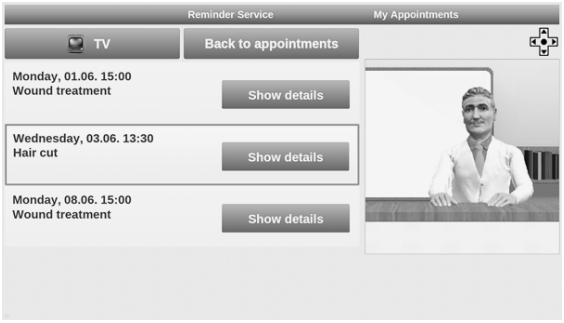

Figure 3: Sample UI generated for the TV device by the AALuis approach.

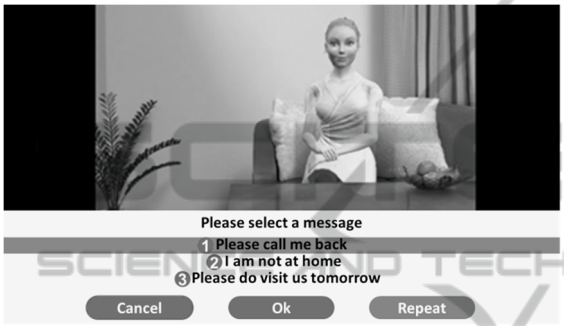

Figure 4: Sample UI generated for the TV device by the ibi approach.

4 CONCLUSIONS

In this paper, two different interaction modeling techniques, the corresponding execution frameworks and the practical exploration results have been presented. The comparison of the exploration results clarifies that the CTT approach applied in AALuis is more applicable for large and complex user interaction scenarios. The SCXML approach applied in the ibi project is more suitable for lightweight and structurally simpler user interaction scenarios.

Although the proposed techniques and execution frameworks already provide beneficial and useable results, we expect to improve both systems. In the AALuis approach, we intend to focus on a semi-automatic generation of CTTs. Introducing some kind of prioritization on service or task-group level would conceivably mitigate the shortcomings of the concurrent execution of the interactions. In the ibi approach, we intend to focus on a semi-automatic binding between the SCXML states and external services. Another improvement for ibi would be the automatic generation of modality specific output without the need of a concrete implementation for the template interpretation.

ACKNOWLEDGEMENTS

The project AALuis was co-funded by the AAL Joint Programme (REF. AAL-2010-3-070) and the following National Authorities and R&D programs in Austria, Germany and The Netherlands: BMVIT, program benefit, FFG (AT), BMBF (DE) and ZonMw (NL).
The ibi project was co-funded by the benefit programme of the Federal Ministry for Transport, Innovation and Technology (BMVIT) of Austria.

REFERENCES

Janssen, C., Weisbecker, A., Ziegler, J., 1993. Generating user interfaces from data models and dialogue net specifications. *Proceedings of the INTERACT'93 and CHI'93 conference on human factors in computing systems*, pp. 418-423.

Puerta, A., Eisenstein, J., 1999. Towards a general computational framework for model-based interface development systems. *Knowledge-Based Systems* 12.8, pp. 433-442.

Paternò, F., Mancini, C., Meniconi, S., 1997. ConcurTaskTrees: A diagrammatic notation for specifying task models. *Human-Computer Interaction INTERACT'97*, pp. 362-369.

Barnett, J., Akolkar, R., Auburn, R. J., Bodell, M., Burnett, D. C., Carter, J., Rosenthal, N. A. (2007). State chart XML (SCXML): State machine notation for control abstraction. W3C working draft.

Aaluis.eu, 2015. AALuis - Ambient Assisted Living user interfaces. Available from <http://www.aaluis.eu>. [16 March 2015].

Ibi.or.at, 2015. Das Projekt. Available from <http://www.ibi.or.at>. [16 March 2015].

Mori, G., Paterno, F., Santoro, C., 2004. Design and development of multidevice user interfaces through multiple logical descriptions. *Software Engineering, IEEE Transactions.* pp. 507-520.

Peissner, M., Häbe, D., Janssen, D., Sellner, T., 2012. MyUI: generating accessible user interfaces from multimodal design patterns. *Proceedings of the 4th ACM SIGCHI symposium on Engineering interactive computing systems*. pp. 81-90.

Popp, R., Raneburger, D., Hermann K., 2013. Tool support for automated multidevice GUI generation from discoursebased communication models. *Proceedings of the 5th ACM SIGCHI Symposium on Engineering Interactive computing systems.* pp. 145-150.

Brambilla, M., Mauri, A., Umuhoza, E., 2014. IFML: Building the FrontEnd of Web and Mobile Applications with OMG's Interaction Flow Modeling Language. *Web Engineering,* Springer LNCS vol. 8640. p. 575.

Harel, D., 1987. Statecharts: A visual formalism for complex systems. *Science of computer programming 8.3*. pp. 231-274.

Zur Muehlen, M., Recker, J., 2008. How much language is enough? Theoretical and practical use of the business process modeling notation. *Advanced information systems engineering* pp. 465-479.

Mori, G., Paternò, F., Santoro C., 2002. CTTE: Support for Developing and Analyzing Task Models for Interactive System Design. *IEEE Trans. Softw. Eng., 28(8)*. pp. 797–813.

Romellini, C. Tonelli, F., 2005. CCXML: The Power of Standardization. *Loquendo*.

W3.org, 2015a. Voice Browser Call Control: CCXML Version 1.0. Available from <http://www.w3.org/TR/ccxml/>. [18 March 2015].

Almeida, N., Silva, S., Teixeira, A., 2014. Multimodal Multi Device Application Supported by an SCXML State Chart Machine. *Engineering Interactive Computer Systems with SCXML*. p. 12.

Dragert, C. W., 2013. ModelDriven Development of AI for Digital Games. Doctoral dissertation, McGill University.

Jeong, H., Kim, S., Do, H., Choi, E., Jeong, Y., Kang, Y. 2012. Multimodal Interface for Mobile Cloud Computing. *Latest Advances in Information Science and Applications* WSEAS Press. pp. 270 – 274.

Code.google.com, 2015a. scxmlgui - A graphical editor for SCXML finite state machines. - Google Project Hosting. Available from <https://code.google.com/p/scxmlgui/>. [16 March 2015].

Team Commons, 2015. SCXML - Commons SCXML. Commons.apache.org. Available from <http://commons.apache.org/proper/commons-scxml/>. [16 March 2015].

Code.google.com, 2015b. scxmlcc - scxml to C++ compiler - Google Project Hosting. Available from <https://code.google.com/p/scxmlcc/>. [16 March 2015].

GitHub, 2013. jroxendal/PySCXML. Available from <https://github.com/jroxendal/PySCXML>. [16 March 2015].

Alliance, OSGi, 2003. Osgi service platform, release 3. IOS Press, Inc.

Mayer, C., Zimmermann, G., Grguric, A., Alexandersson J., Sili, M., Strobbe, C., 2015. A comparative study of systems for the design of flexible user interfaces, *Journal of Ambient Intelligence and Smart Environments*. in press. ("accepted").

W3.org, 2015b. *XHTML 1.0: The Extensible HyperText Markup Language (Second Edition)*. [online] Available at: Available from <http://www.w3.org/ TR/xhtml1 [Accessed 18 Mar. 2015].

W3.org, 2015c. *XHTML 1.0: The Extensible HyperText Markup Language (Second Edition)*. [online] Available at: Available from <http://www.w3.org/TR/ xhtml1 [Accessed 18 Mar. 2015].

Richardson, L., Ruby, S., 2008. RESTful web services *O'Reilly Media, Inc*.

Sili, M., Bobeth, J., Sandner, E., Hanke, S., Schwarz, S., Mayer, C., 2014. Talking Faces in Lab and Field Trials - A View on Evaluation Settings and User Involvement Results of Avatar Based User Interaction Techniques in Three Ambient Assisted Living Projects. *International Conference on Human-Computer Interaction.* in press.