Architectural Management of Synchronous and Asynchronous Interactions; a Reference Architecture Model for Synchronous and Asynchronous Interactions

Juan Muñoz[1], Jaime Muñoz[1], Francisco J. Álvarez[1], Francisco J. Álvarez[1], Ricardo Mendoza[1], Humberto Cervantes[2]

[1] Universidad Autónoma de Aguascalientes, Av. Universidad 940,
20100 Aguascalientes, México
{JMunoz, JMunozAr, FJAlvar}@correo.uaa.mx; mendozagric@yahoo.com.mx
[2] Universidad Autónoma Metropolitana - Iztapalapa, San Rafael Atlixco 186, Col. Vicentina
09340 Delegación Iztapalapa, D. F., México
hcm@xanum.uam.mx

Abstract. Asynchronous and Synchronous interaction between humans and systems are commonly implemented using differentiated architectural elements. This causes a combination of components and connectors which expands architectural models making them more complicated. In This paper we describe a mechanism to combine both kinds of connections simplifying software architectures at the same time that they can be used in hybrid ways to increase software usability. The mechanism is implemented by means of a general purpose component that can be used as a reference architectural element that can be used in different systems.

Keywords: Software architecture, human interaction, synchronous asynchronous interaction.

1 Introduction

Almost any (known) system has different relations with its environment. Interactions have different effects in the system and the outputs produced by it. Those reactions can be perceived immediately or not.

When we talk about information technologies we can identify many kinds of systems. Also, users can be considered as systems.

Systems that are based on software are designed to interact with other systems in its environment. Their functionalities must respond on different times to request of information depending of volume, speed, order process, etc. Human interaction is one of the most complex but common relations that must be established by software programs.

We can identify synchronous and asynchronous interactions between a human being and a software system. A synchronous interaction is given by a set of one or more sequential action – reaction events which occurs one in response to other, while this sequence occurs the flow of the system is blocked. An asynchronous interaction

represents a set of one or more events that will cause one or more results that not necessary are generated immediately or sequentially.

Dix et al. [18] says that Human Computer Interaction (or HCI) is "the study of people, computer technology and the ways these influence each other. We study HCI to determine how we can make this computer technology more usable by people".

Hewett et al. [1] in their definition of HCI say that the principal concerns of this area are: "the design, evaluation, and implementation of interactive computing systems for human use and the study of major phenomena, surrounding them".

Software architecture describes system components and connectors and rationality used to build and distribute them. An architectural model establishes mechanisms of interaction among components of the system itself, with components from other systems and with users. Meanwhile, HCI field provides theoretical background for supporting the relations between software and people.

Avouris [14] says that HCI also provides support for developing "usable" software systems, this is, systems that will have a set of attributes based in necessary effort for using it and for assessment of their use by a given set of users.

Human System Interaction is an aspect that has been associated with usability in the ISO TR 18529 [3]; according to this standard, the term is related to how people can use system's capabilities in a certain context. In a paper written by Abran et al. [16], we can find that usability has been defined in several ways by researchers and standardization bodies; consequently in this article we don't find a unique definition of the term, but we can read three descriptions for this quality characteristic, each one from a different viewpoint:

"1. For the end-user, software usability is essential because it is a determinant of performance: an application, which features good usability, will allow the user to perform the expected task faster and more efficiently.

2. For managers, usability is a major decision point in selecting a product, as this decision will have a direct influence on the learnability of the chosen system, and hence on the productivity of those who use it.

3. For software developers, usability describes the internal attributes of a system, including issues like design quality, documentation maintainability."

We can see that the affirmation of Abran et al. sounds logic because usability is a broad concept that has been studied and defined by many authors [9], [10], [11], [12], [13], [16], [17] (and a long list of etc.). Those researchers have associated this characteristic to manageability, learnability, attractiveness, performance, time behavior, understandability, correctness, easiness to use and memorize, etc. In this article we only use this concept as a synonym of expected system behavior in a general way, because boarding all these concepts could take us to a different field out of the scope of the paper.

2 Problem

Several authors like Bass et al. [2] have established that usability, a central concern of HCI, needs to be supported by well designed software architectures. But in these works, usability has been closely related only to synchronous interactions.

Asynchronous relations between users and software systems are hardly studied in these days where usability of graphical user interfaces represents the main concern of computer communication with humans, and they are expected to give immediate answer to each action. But, a lot of GUI's also need to make use of asynchronous interactions. A web form used for collecting information from the user is an example of this need. If those interactions aren't properly described by architectural models they will affect usability.

Both kinds of interactions have applications for different problems so they need to be considered when designing software architectures for usable systems. Needless to say that synchronous system interaction (SSI) and asynchronous system interaction (ASI) have different characteristics and uses.

SSI is very flexible, and it can be adapted to situations where a process is not defined by an ordered sequence of tasks that will be repeated every time in the same way. User can get control of the application to make tasks requiring precision and immediate response to decisions that must be taken by people.

ASI is suitable for management of large volumes of information which require a lot of processing time and little attention of the user. This mechanism can help to reduce variability of processes and to simplify applications by reducing unexpected inputs. Also, if we cannot establish immediate communication with the system, for example, when we have to take measures on extreme conditions with a remote autonomous device, ASI could be the best approach.

An architectural model must describe at a high abstraction level a solution space for a given set of requirements and constraints established for a group of stakeholders [4]. Therefore, software architecture must define a system with enough elements to satisfy stakeholders' concerns under process constraints.

Deciding when a system must employ asynchronous or synchronous interaction is critical for system usability perception. When a user takes control of a system he expects to receive answers from it in a given way and time under certain conditions. If a user is expecting an immediate (synchronous) response from a system and it isn´t produced he will think that system is not working correctly. In the other hand, if user is not expecting to receive an answer from the system maybe he will not be attending the system when needed and it will stop working until response have been given.

We can describe systems with efficient mechanisms for human-computer interactions in an architectural model. Software architecture will define system capability to interact with people in a synchronous or asynchronous way. Many systems will need to combine both kinds of interactions. Without adequate software architecture, design of those systems could be too complex.

Selecting the best suited interaction type will impact usability perception about the system. The kind of interaction must guide design efforts to select or create architectural patterns to support it and to enhance not only usability but also other qualities of the system like performance or security, for example.

But, architectural patterns might be generalized; after all, they need to be extensively used in the system. Simplification can be done by implementing reusable components to reduce system complexity and encourage developers to produce other critical common functionalities with "less effort" and "more quality".

3 Proposal

This work has its focus on interactions, as one of the elements that must be described by a component to be integrated as a common element of a software architecture. We distinguish two kinds of interactions: synchronous (SSI), which deal with sequential pairs of action-reaction processes; and asynchronous (ASI), related to multiple inputs with no special order, and a final output which contain results and information of found exceptions.

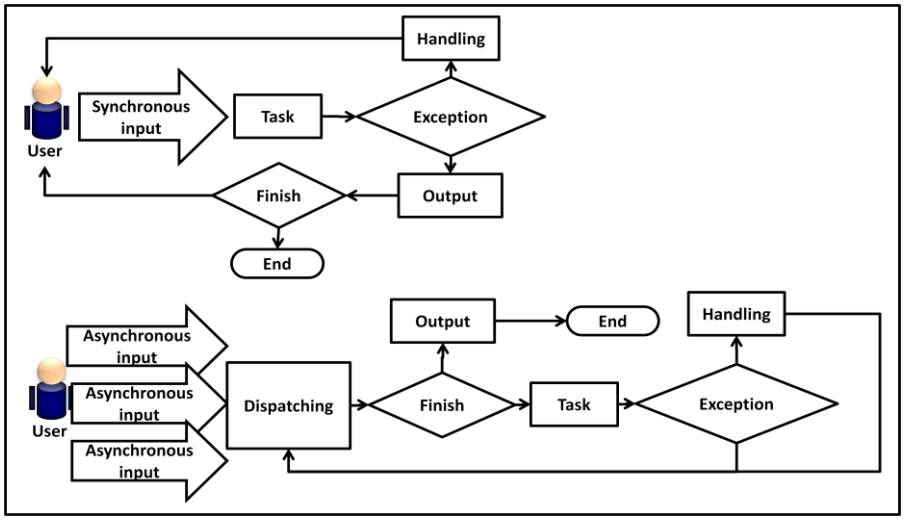

Fig. 1. Synchronous and asynchronous human-system interaction mechanisms

In Fig. 1, we can see how SSI and ASI work. Synchronous interaction establish a close human-system relation; user must be continuously attending to processing behavior and correct immediately any unexpected situation. Asynchronous interaction doesn't expect to have support from the user to continue its operation when an exception occurs.

We can think that SSI must take care of a mechanism for a sequential model of input, processing of tasks, output and exception handling, blocking the whole system until finishing a task. But, this is just a simplistic approach of the process.

Sometimes, an input must be validated; if it's incorrect a feedback mechanism will be needed. More complex interactions will need to implement mechanism for management of: user identity, access authorization level, contextual help, etc. But, all this interactions have something in common; they have an immediate response to each event.

Events can be produced by human, hardware or software actions and they can occur sequentially or in parallel. SSI processes can be well managed when events take place sequentially; but, parallel events bring some troubles when handling them. In a SSI interaction process with multiple parallel events, users might need to answer to

multiple questions sequentially to go on with a process, but probably they will not know in which order they must do that.

ASI produces an output as SSI does, but we don't expect to have immediate feedback of each event. We can have multiple inputs that aren´t in a specific order; maybe they are processed in a parallel way, and at last will have an output and a report of exceptions.

When using ASI, design of GUI is simplified because outputs will be recorded prior to being showed. When we have parallel processing this characteristic is very useful, because the order of required answers can be stated by a program and displayed in just one view.

ASI has other kind of risks; a large job executed by different processors might stop working or give us incorrect outputs when a failure in one of its task occurs if the user doesn't correct an input when an exception is detected.

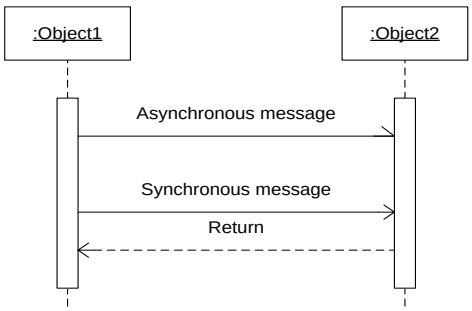

Fig. 2. Synchronous and asynchronous messages representation in UML

Synchronous and asynchronous human system interactions must be combined in a system to complement each one. UML's sequence diagrams can help us to express these interactions (see Figure 2). But, we need to develop architectural components to handle those combinations that could be too complex.

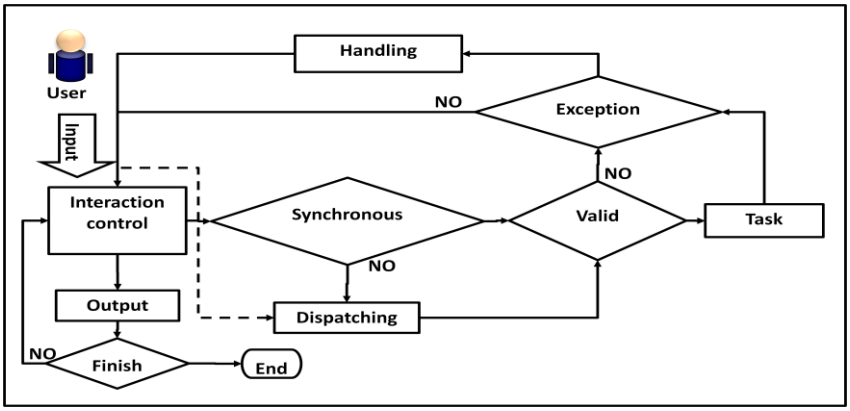

Fig. 3. Asynchronous-Synchronous System Interaction control

A coordination mechanism to guide switching of both interactions can simplify the effort that must be invested while creating instruments for controlling human-system interactions. This could be done by implementing a component to control them, like the one described in Figure 3.

This mechanism has a module of interaction control that conducts any kind of input (synchronous or asynchronous). Each input must be tagged to know the expected behavior (ASI or SSI). High priority messages will be treated all times as Synchronous Interactions.

The interaction control will add a sequence id, a flow id, a context id and output id to guarantee correct processing of all inputs. Also this id can be used to help user on knowing the correct answering sequence to multiple messages. The flow id has an effect of a session control; it helps to manage multiple input devices (or users) acting in parallel.

The mechanism generates multiple dispatching lists for ASI. The dispatching module will inform of any change of state of any element of its lists to the interaction control, doing this, interaction control can monitor state of all inputs, and if it would be necessary, it could make changes in the lists to improve performance, or even, solve deadlocks.

Validation will act for every input based on rules described by scripts the context id will serve to recognize the appropriate script that must be applied. Exceptions will be sent to the exception handling module.

Valid inputs will be sent to the task module, this is an interface to send messages to system objects, running legacy systems, components or web services.

Exceptions will be managed on the handling module using scripts developed to express how the interaction control module must give feedback to the user.

The output module will receive valid outputs and exception feedback to show them on different windows. Those windows can be located on different devices so is possible to multiplex outputs and direct them to various devices. We can include mechanisms to translate those outputs to different languages or graphical representations.

The ASI-SSI control mechanism is a proposal of a multi-purpose component that can be developed as a reusable component or as a service to simplify software architectures. It also helps to implement flexible systems that can be adapted to different input or output devices and technologies.

An example of how the component for interaction control could be implemented is described in figure 4. We start receiving tagged messages from other components of the system; generically will call these components as “sender”. The ASI-SSI control component will work in a synchronous mode if it receives a high priority message.

Normal messages from a sequence received in greater time intervals to an acceptable minimum previously established will be managed in an asynchronous way too.

Synchronous interactions will be subject to time control. If a message doesn’t receive an answer under a limit time it will be changed to asynchronous mode unless it has a high priority tag. High priority messages always will be time controlled.

If an answer is not received in the expected time the component will send the message again. Messages will have a limited number of retries, if the number of predefined retries is rebased an exception message (error) will be generated. Number

of retries can be defined to be infinite, but this can cause that some message could never be answered.

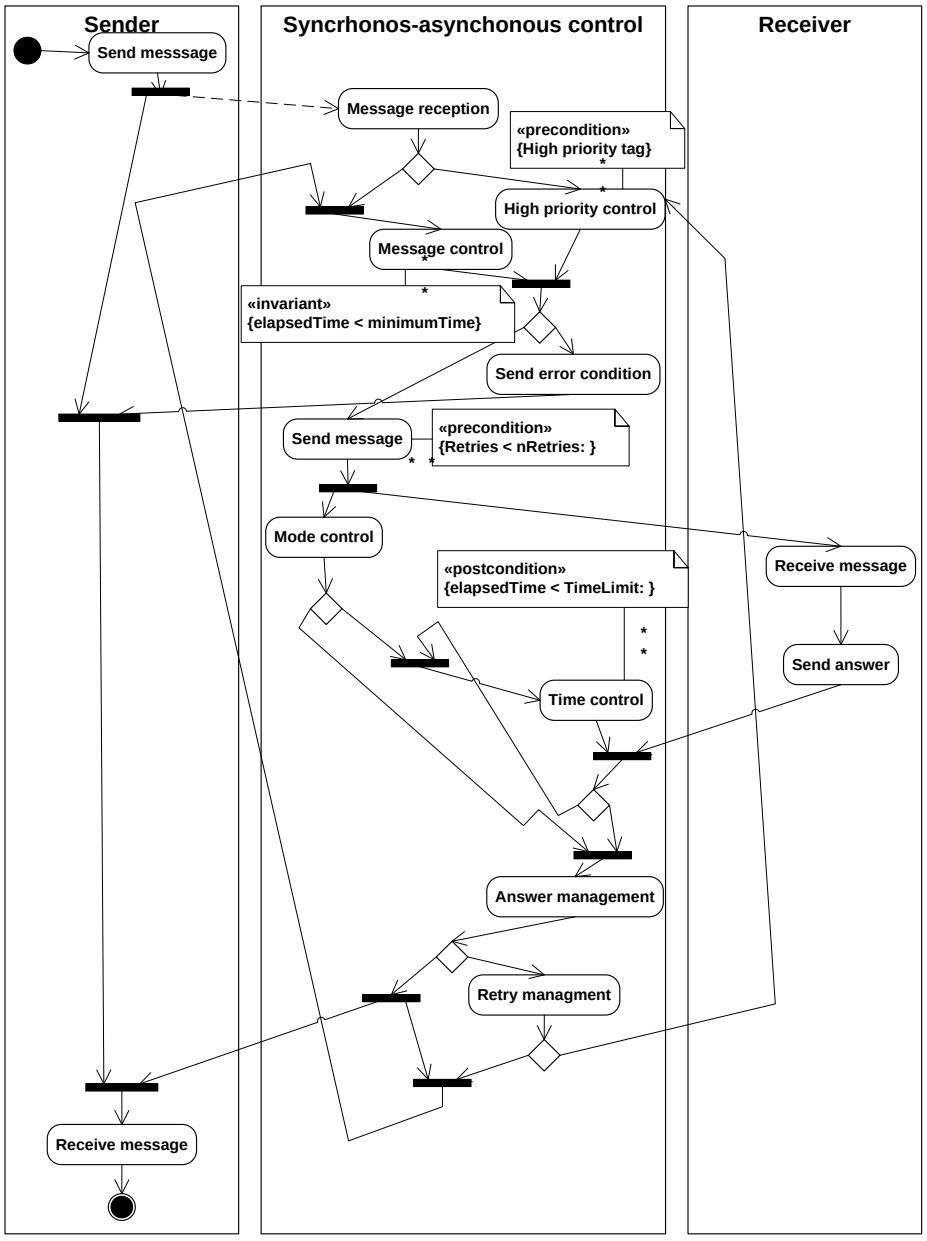

Fig. 4. UML activities model of the interaction control component

Asynchronous messages will be managed by means of a queue. This mechanism will help to sort messages and to have a checklist for ensuring that all messages will have an answer. Message control mechanism will have a defined time politic for resending asynchronous messages, so these messages will not be forgotten.

Asynchronous for a sequence way will try to change to synchronous if answers to the messages are received in times that are lower to the monitoring time interval established for the message control mechanism and the queue for that sequence contains only one message.

Users of systems including this kind of ASI-SSI management component will find that expected behavior and perceived performance will be improved by giving them an automated control of priorities and order of messages from other users and from the system itself.

4 Case Study

Development of software system is known to be a complex task, the use of the object-oriented analysis and design (OOAD) is a great help to identify problems and describe the solutions in terms of object-oriented models. However, learning OOAD is a challenging in particular for younger students because they need to work in group in order to get experience to analyze and design the software system independently of any language programming. This cases, address this issue with a collaborative system called OOADCE (Object-Oriented Analysis and Design Collaborative Environment) [5], which allows to undergraduate students to put in practice out of classroom the OOAD in a collaborative manner.

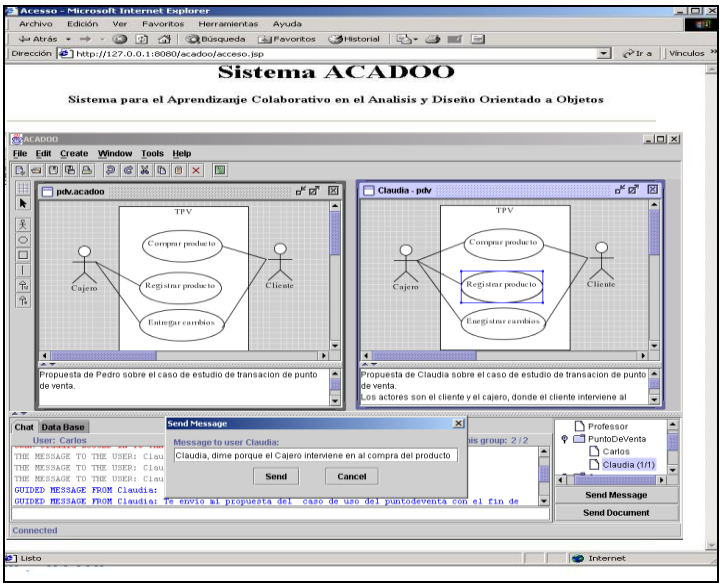

Fig. 5. Collaboration scenario between two students using the OOADCE

We have chosen the collaborative learning system described by the OOADCE model to show an example of how combinations of both communication mechanisms could be used to implement functionalities in a system that needs to manage messages in varied ways.

Combination of ASI and SSI is necessary to implement different kinds of solutions where interactions must switch between synchronous and asynchronous modes many times, maybe in the same session. An example of an architectural implementation using the component to solve the scene: “ask for commentaries about homework at an assisted session”, could be represented by the view of previous figure.

Software for assisted class sessions and individual tests could, in general terms, be developed using SSI functionality of the architectural component; at the same time, download of self-study material and management of homework could make use of ASI functionality, but the use of the component to combine both behaviors, like when groups need to interact designing and correcting models is where it makes easier to design the architectural model for the system.

Because of OOAD nature, student could need time for thinking and deciding. In this case, activities that are taking place in a synchronous way could be switched to an asynchronous mode.

With the proposed architectural model, the system can manage needs of synchronous and asynchronous messaging giving continuity and congruency to the exchange of messages among users (teacher and students) and the system. Also, by using the proposed interaction control component, delays and composition (integration and sorting) of messages can be integrated in a way that users perceive as “a natural system behavior”.

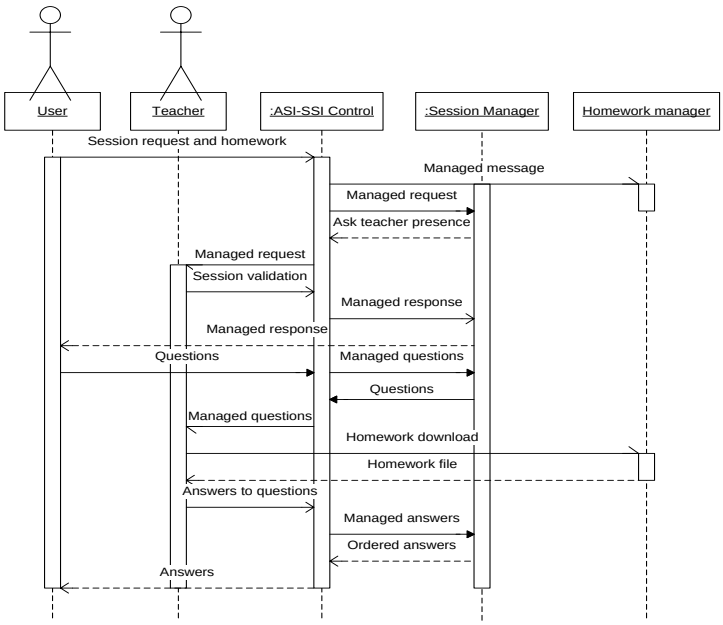

Fig. 6. Use of the Asynchronous-Synchronous System Interacion (ASI-SSI) control component

This case is not the only application in which we can use the ASI-SSI control component. Software for virtual meetings can use the component to establish similar functionalities to the described in the OOADCE model. Coordinator of a meeting can switch from a synchronous mode to an asynchronous mode and vice-versa to load contextual documents, or give time for decisions taking and then switch mode to make corrections on a collaborative way.

Other example of the potential use of this component is in a remote CCTV surveillance system that must run on Internet. Under normal situations the system can use ASI mode to send pictures from different places that are being monitored by several cameras. When an unexpected event appears, the system must switch the mode to SSI, so images from the camera that has focused the event can be sent in real time. All other, cameras will continue sending signals in ASI mode. Voice communication could be activated in SSI mode too to support coordination between guards. The camera will continue in this mode until the emergence finishes, after that the system will take the ASI mode again for this camera. SSI mode will be still working for voice streams until finishing the conversation.

5 Related Work

Folmer and Bosch [6] describe architectural patterns that are considered to have a positive impact on usability. Those patterns are common solutions to provide better user-system interaction but they must be applied to specific kinds of interactions (synchronous or asynchronous).

Graunke y Krishnamurthi [7] presents a set of design patterns to build flexible user interfaces. These patterns are focused on synchronous interactions and make use of configuration files to implement flexibility.

On his book, Ian Garton [8] shows how message oriented middleware is used to manage synchronous and asynchronous interactions by using message queues but his description doesn't describe how to make hybrid mechanisms to combine both kinds of interactions in an architectural model.

Systems that work with synchronous and asynchronous interactions are described in the paper of Preguiça et al. [15]. In this paper we can see how both kind of interactions can be handled in a fixed way, but they don´t show a mechanism for control of a dynamic behavioral change responding to different rates.

6 Conclusions and Future Work

We cannot say that Asynchronous-Synchronous System Interaction (ASI-SSI) control mechanism is a pattern because it must be implemented as a component which will be responsible of conducting inputs and outputs in the system. The component must have a switching mechanism to decide between different interactions modes.

This simple mechanism makes easier to implement traditional functionalities which help to improve system usability, like: undo, redo, cancelation, contextual help, etc. but also it can be used to help on implementing heterogeneous system integration,

simultaneous language translation, broadcasting systems, multiple devices handling, etc.

An efficient control of both kinds of interactions improves system usability perception and overall user experience when interacting with computers. Correct switching between both modes can help user to take better control of system assisted processes by giving them superior ways for watching and answering to notification messages, to correct deviations on time, to find important information and to avoid missing of relevant data, per example.

Maybe some people could thing that the ASI-SSI control mechanism is just a transformation mechanism from SSI to ASI; but we can argue that input behavior is conserved as a classic synchronic interaction, unless transformation is necessary.

It's necessary to develop a language to generate the scripts needed to implement the validation part of the component doing it more reusable and less coupled.

References

1. Hewett, T., Baecker, R., Card, S., Carey, T., Gasen, J., Mantei, M., Perlman, G., Strong, G., Verplank, W., ACM SIGCHI Curricula for Human-Computer Interaction, ACM SIGCHI (1996)
2. Bass, L., John, B., Kates, J., Achieving Usability Through Software Architecture, Technical Report CMU/SEI-2001-TR-005 ESC-TR-2001-005, Carnegie Mellon Software Engineering Institute, Pittsburg, PA (2001) 1-87
3. ISO, ISO TR 18529: Ergonomics for human-system interaction-Human-centered lifecycle process descriptions, ISO (2000)
4. Muñoz, J., Muñoz, J., Álvarez, F. J., Rodríguez, G., Specification and Evaluation of the Needed Requirements for the Design of Software Architectures, IASTED conference on Software Engineering and Applications 2006, Dallas, TX, USA (2006)
5. Muñoz, A., Rodríguez, F., Garza, L., Pinales, F., Modelo Para el Aprendizaje Colaborativo de Análisis y Diseño Orientado a Objetos Soportado Por Computadora, Revista Apertura del Sistema Universidad Virtual, Univesidad de Guadalajara, No. 1, México (2005)
6. Folmer, E., Bosch, J., Architecturally Sensitive Usability Patterns, Department of Mathematics and Computing Science, University of Gronigen, Netherlands (2003) 1-19
7. Graunke, P., Krishnamurthi, S., Advanced Control Flows for Flexible Graphical User Interfaces. Or, growing GUIs on Trees; or Bookmarking GUIs, ICSE'02, ACM, Orlando FL, USA (2002) 277-287
8. Garton, I., Essential Software Architecture, Springer-Verlag, Germany (2006)
9. Rozanski, N., Woods, E., Software Systems Architecture, Addison Wesley, Pearson Education Inc., USA (2005)
10. ISO/IEC, "ISO/IEC CD 25012: Software engineering: Software Quality Requirements and Evaluation (SQuaRE) Data Quality Model", ISO/IEC JTC1/SC7, Canadá, (2005) 1-32
11. Malan, R., Bredemeyer, D., Defining Non-Functional Requirements, Bredemeyer Consulting, www.bredemeyer.com, USA (2001) 1-8
12. ISO/IEC, ISO/IEC 9126. Software Engineering – Product Quality – Part 1 Quality Model, ISO/IEC, Canada (2001)
13. Maciasek, L., Roundtrip Architectural Modeling, Second Asia-Pacific Conference on Concpetual Modelling (APCCM2005), Conferences in Research and Practice in Information Technology Vol. 43, Australian Computer Society/Sven Hartman and Markus Strumper, ACM, Newcastle, Australia (2005) 17-23

14. Avouris, N., An Introduction to Software Usability, Report Research, University of Patras, Rio Patras, Greece (2001) 1-10
15. Pregui¸ca, N., Legatheaux, J., Domingos, H., Duarte, S., Integrating Synchronous and Asynchronous Interactions in Groupware Applications, Proceedings of CRIWG'2005, Springer, Costa Rica (2005) 1-16
16. Abran, A., Khelifi, A., Suryn W., Usability Meanings and Interpretations in ISO Standards, Software Quality Journal, 11, Kluwer Academic Publishers, Netherlands (2003) 325-338
17. Preece, J., Rogers, Y., Sharp, H., Benyon, D., Holland, S., Carey, T., Human-Computer Interaction, Addison Wesley, USA (1994).
18. Dix, A., Finlay, J., Abowd, G., Beale, R., Human-Computer Interaction (3rd Edition), Prentice-Hall, USA (2003).