DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO

Relatório Técnico

RT-MAC-2010-02

JAM SESSION – KNOWLEDGE-BASED INTERACTION PROTOCOLS FOR INTELLIGENT INTERACTIVE ENVIRONMENTS

FLÁVIO SOARES CORREA DA SILVA

Janeiro de 2010

Jam Session – Knowledge-based Interaction Protocols for Intelligent Interactive Environments

Flavio Soares Correa da Silva

Laboratory of Interactivity and Digital Entertainment
University of Sao Paulo
Rua do Matao, 1010, Sao Paulo, Brazil 05508090
fcs@ime.usp.br

February 3, 2010

Abstract

In the present article we characterise a class of problems, based on the solutions that can be brought to them. This class of problems are those that can be solved by systems based on Intelligent Interactive Environments. We first introduce the notion of Intelligent Interactive Environments, then we introduce the Jam Session language, which is a compact and simple to use conceptual tool to build systems based on the notion of Intelligent Interactive Environments. Finally, we convey our view that such solutions can be particularly effective for those systems which shall address the needs of a wide spectrum of diverse users – as occurs, for example, in the case of Electronic Government systems. In order to convey more clearly this view, we present a small prototype of a system to address an issue related to cross-borders Electronic Government, namely, the issuing of visas to enter a country for specific purposes.

1 Introduction

In the present article we characterise a specific class of problems, for which a confluence of techniques from different areas of specialisation can be relevant. We have coined the solutions to these problems *intelligent interactive environments*. Hence, the class of problems with which we deal in the present article are those problems that can be solved by means of modelling and utilisation of intelligent interactive environments.

An intelligent interactive environment is an environment in which:

- A notion of *location* is relevant, either because it is an inherent feature of the environment (as, for example, when the displacement of objects in a physical environment is relevant) or because this concept can be useful to

characterise and control relevant aspects of the problem we intend to solve (as, for example, when we have specific services in the system that must be switched on or off depending on the status of circumscribed portions of the system - in which case we can model such portions of the system as mobile agents and their status as locations they occupy in a virtual space).

- *Responsiveness* is an expected feature, i.e. the environment is expected to interact with users (which can be called *inhabitants* in this case), in order to generate useful actions in response to their needs, goals and intentions.
- *Intelligence* is also an expected feature, i.e. the environment is expected to sense the behaviour of its inhabitants, reason about the motivations, needs, goals and intentions that may have generated that behaviour, and respond accordingly.

In order to model and implement an intelligent interactive environment, it is required, therefore, that techniques and methods from at least the following areas be used:

- Distributed and mobile computation, in order to capture the notions of mobility of software components, as well as of computational devices that may host them.
- Interaction design, in order to communicate effectively with inhabitants and capture their behaviour and actions with utmost efficiency.
- Artificial intelligence, and more specifically knowledge representation and inference, in order to respond intelligently to the motivations, needs, goals and intentions of inhabitants.

In order to build a more concrete image of the class of problems that can be modeled and solved effectively through intelligent interactive environments, we present in the following paragraphs a few illustrative examples.

- **Augmented Reality**: the availability of location aware portable devices that can communicate to each other and inform their locations (e.g. mobile phones and ultra-portable computers that can locate themselves through GPS sensors or proximity to cellphone base stations, and broadcast this location through Wi-Fi or Bluetooth networks), as well as provide multimedia information to their users, has enabled the possibility to build low cost infrastructures for augmented reality systems, in which users provide input data to virtual worlds by means of interacting with each other and moving about in the physical world, and inhabit simultaneously both the physical and virtual worlds, in such way that these worlds are connected to each other through the portable devices carried by the users.

  Applications of this sort of systems are manifold: they can be used for entertainment, as e.g. in alternate reality games [8]; remote and semi-automated assistance in complex engineering tasks [7]; and augmented and

mixed reality applications for education, tourism and cultural information [10].

Our proposed approach for this sort of applications is based on the construction of a virtual world, inhabited by proxies of the bearers of portable devices in the physical world, whose behaviour is influenced by actions and movements in the physical world. In turn, the behaviour and actions of these proxies in the virtual world influences back the physical world, by communicating with sensors and actuators that change their status accordingly [5, 6].

Hence, we need the appropriate resources to maintain the virtual and the physical worlds synchronised, so that reasoning and intelligent reactions of the environment for actions of its inhabitants can be concentrated in the virtual world.

One important feature of an intelligent interactive environment is that, even though it is clearly a multiagent system, in which computer controlled agents must co-exist with humans and human controlled agents, the environment itself must react to actions that arise from its inhabitants. Environments, based on this view, are a special sort of intelligent agents, which embody all relevant notions to characterise and regulate the allowed interactions among the agents that inhabit them.

This sort of applications can also scale up to highly complex systems. In order to be able to build reliable systems, the tools that are used to design and implement those systems must feature resources to verify and monitor the behaviour of the proposed designs and implementations.

- **Virtual Worlds and Computer Games**: in the majority of systems comprising user interactions through virtual worlds, as well as in many computer games, a simulation of a physical setting is proposed as the arena in which computer controlled and human controlled characters interact.

  Considered this way, virtual worlds and computer games can be envisaged as a special case of augmented reality systems, in which the tracking of movements and actions in the physical world is deemed unnecessary, and all interactions are transported to the virtual environment.

  All features highlighted in the previous example are relevant in this case, hence the tools that are built for augmented reality systems shall also be useful for virtual worlds and computer games.

- **Interactive Systems**: in some cases, the notion of location can be used as an interesting metaphor to characterise context and implement control features to ensure reliability and security of systems, by enabling or disabling, depending on context, specified services and modules within a system.

  This approach has been used in the development of languages and tools for the specification and implementation of mobile and distributed systems, as can be found, for example, in the Ambient Calculus [2].

Most often than not, languages for algebraic specification of mobile and distributed computing are expressive and mathematically well grounded, but can also be complex and difficult to handle. Additional desired features of tools to specify, design and implement intelligent interactive environments are simplicity and user friendliness.

In the following sections we introduce a proposed conceptual tool for intelligent interactive environments. In section 2 we present a few related projects, which have been particularly influential in the design of our proposed conceptual tool. In section 3 we introduce the Jam Session language for design and implementation of intelligent interactive systems. In section 4 we illustrate Jam Session at work, through a simple concrete example. Finally, in section 5 we draw some conclusions and proposed future work.

Our proposed language has been coined *Jam Session*, in reference to the standard practice of jazz musicians inaugurated during the forties in New York City. "Jam" is actually an acronym and states for "Jazz after midnight", and the original jam sessions were relaxed and innovative performing sessions that professional musicians developed after their duties in clubs and restaurants were over. Our Jam Session language orchestrates previously existing computational resources in original ways for innovative applications, hence the analogy.

2 Related Projects

The development of the Jam Session language has been inspired by existing research initiatives which account for specific subsets of the expected features of this conceptual tool.

A conceptual architecture that bears several similarities with our approach is that of Electronic Institutions [4]. Electronic Institutions are comprised by agents, roles, normative rules that characterise the institutions *per se* and scenes in which regulated dialogues occur involving agents. For an agent to participate in a dialogue, it must assume a specific role that is compatible with that dialogue, and in order to assume a specific role the agent must ensure that it has the necessary capabilities required for that role. Once all roles required for a dialogue are fulfilled by agents, the dialogue can start and, if all norms are followed, the dialogue can run to its end, thus performing a structured and controlled sequence of linguistic actions.

Our notion of location is replaced in Electronic Institutions by the notion of roles. Furthermore, actions in Electronic Institutions are restricted to linguistic actions. It can be an interesting future exercise to verify whether Electronic Institutions and Jam Session specifications are formally equivalent. Clearly, however, they are not *ergonomically* equivalent: systems whose most important actions are primarily linguistic may be better modeled by means of Electronic Institutions, whereas systems inhabited by agents that perform actions of different nature, e.g. physical interactions with the environment and mobility accross physical and/or virtual spaces, may be modeled more naturally using Jam Session.

In Jam Session, a central concept is the concept of *environment*, which is the location where inhabitants meet to interact. The capabilities of the environment are the enablers of interactions between inhabitants as well as between an inhabitant and the environment itself. In the case of Electronic Institutions, the related concept is that of *institutions*, which are abstract sets of norms that specify the possible interactions that agents can perform.

The notion of norm-mediated interactions – thus placing norms as a central notion to regulate and guide agent interactions – has been pushed forward and has taken central stage in other approaches [11]. Indeed, we have also, in the past, focused on norms to specify courses of interactions among agents in virtual worlds [3]. We have reused this notion in Jam Session, complemented with location based situatedness to control the availability of resources for inhabitants.

Another initiative in a similar vein to the ones cited above has been the development of the Lightweight Coordination Calculus (LCC) [9]. Indeed, the LCC is a language to build executable specifications akin to algebraic specification languages such as the Ambient Calculus, similar to Jam Session. It has been proved to have similar expressive power to Electronic Institutions, and it has been used extensively in a variety of applications.

The main difference between the LCC and Jam Session is that the former does not account explicitly and directly for the notion of mobility and situatedness. Even though it is possible to simulate these notions in LCC, specifications of interaction protocols in which these are central notions can become a little terse in that language. We have designed Jam Session specifically to simplify the specification of interactions in this situation.

The LCC has indeed been extended to account for situatedness explicitly, thus resulting in a language coined Ambient LCC, which however seems not to have used in practice, perhaps because it has turned to be a little unfriendly and difficult to implement.

Contrasting to the approaches mentioned thus far, the Multilayered Multiagent Situated Systems architecture (MMASS) [12] considers mobility as the central notion for situatedness and interactions in multiagent systems. In MMASS we find the notion of locations, which are structured by pathways forming a graph of sites through which agents can move to look for specific resources to accomplish their goals. We have borrowed these concepts for Jam Session.

As will be presented in the following sections, the Jam Session language combines the features of all previously mentioned initiatives, to build a user friendly and simple language to specify, design and implement efficiently software solutions based on the notion of intelligent interactive environments.

3 The Jam Session Language

As advanced in the previous sections, a fundamental notion in Jam Session is the concept of *location*. Intuitively, we have services in the environment as a whole which are blocked or unblocked for utilisation, depending on the location of agents. Agents have capabilities, which become active when they

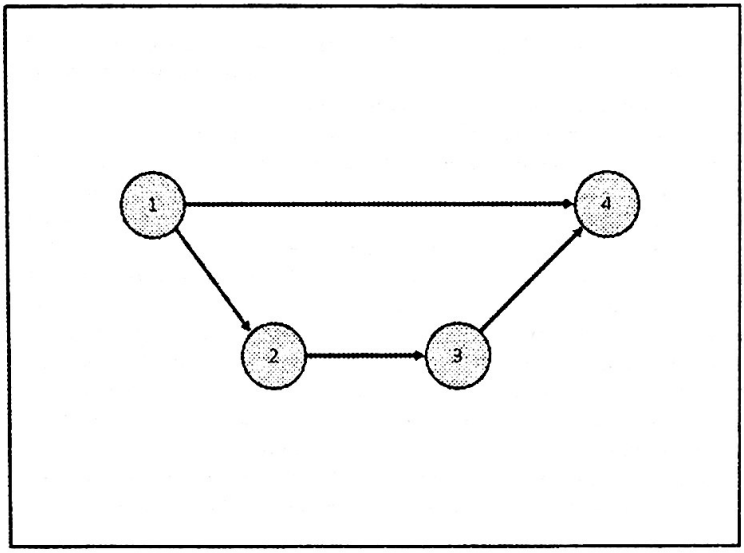

Figure 1: A graph of locations. Agents can move, for example, from location 1 to location 2, but they cannot move from location 2 to location 4

reach specified locations. Whenever we need a specific service, we must make sure that the agent that has the capability of furnishing that service in a specific location has actually moved to that location.

Formally, we have a directed graph to specify locations and their connections. The nodes of the graph are the locations, and the arrows characterise the admissible transitions that agents can perform to move about locations (Figure 1).

Agents are entities that inhabit locations. An agent stays in a location until it receives an order to move to a different location.

An order for an agent to move is a triple of the form

$$move(Agent, Location_1, Location_2).$$

In this order, the agent *Agent* is assumed to be in $Location_1$ and is being requested to move to $Location_2$. An order to move can be evaluated, in which case an attempt to execute it shall be performed and a corresponding truth value shall be assigned to it, depending on the success of the execution.

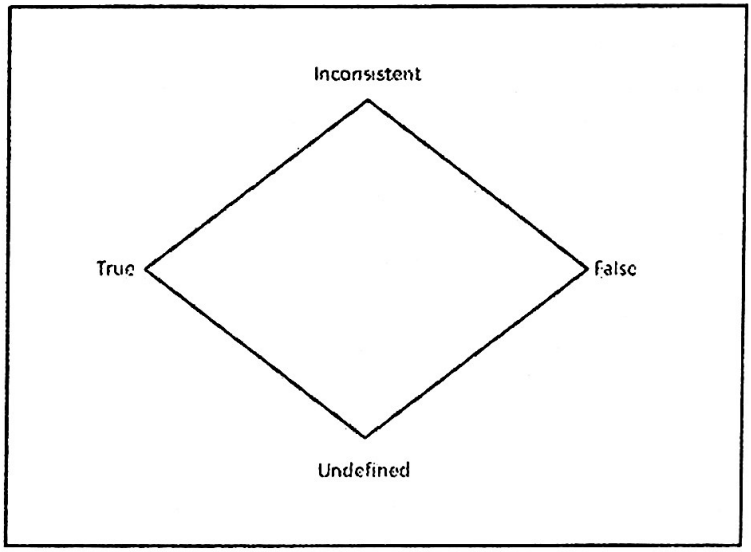

Figure 2: The Belnap four lattice of truth values

In Jam Session, we employ a four valued lattice of truth values known as *Belnap four* [1]. In this lattice, we have the truth values `undefined`, `true`, `false` and `inconsistent`. The truth value `undefined` stands for a statement whose truth value cannot be determined. The truth values `true` and `false` correspond to the usual (classical) notions of truth and falsity. The truth value *inconsistent* stands for a statement whose truth value has been determined as both true and false. The order relations in the Belnap four lattice are depicted in Figure 2.

When an agent receives such order, two situations can occur:

1. Something goes wrong. There are three possibilities for something to go wrong:
   (a) The agent may not be in $Location_1$ in the first place.
   (b) There may not be an arrow starting in $Location_1$ and ending in $Location_2$.
   (c) The agent may have a registered prohibition to move to $Location_2$.

If something goes wrong (i.e. any of the possibilities above occurs), then the agent stays wherever it is and the order is evaluated as bearing the truth value `undefined`.

2. Everything works fine. In other words, all three conditions above are fulfilled. In this case, the agent is actually transported from $Location_1$ to $Location_2$, and the order is evaluated as bearing the truth value `true`.

Jam Session can be envisaged as an orchestrator of external resources. The constructs in this language have been designed to provide the appropriate means to coordinate and to regulate the triggering of available resources. The conceptual tool to perform this coordination and control of resources is the movement of agents through locations.

External resources are named *predicates* in Jam Session. Each predicate is associated to a pair $[Agent, Location]$. Predicates also have parameters, which are formed as classical first-order terms. Hence, a predicate has the form

$$[Agent, Location]predicate(Term_1, ..., Term_n)$$

A predicate can be triggered, i.e. there can be an attempt to evaluate it. Most typically, predicates are used to consume external resources and call external services. The predicate parameters are passed to these external resources and services, and if the resources and services are successful they are expected to send back appropriate instantiations of the parameters.

When a predicate is triggered, three situations can occur:

1. The predicate is blocked. This can occur because the agent *Agent* is not present in the location *Location*, or because the attempt to trigger the predicate is ill specified, e.g. the requested predicate is not actually associated to the given pair $[Agent, Location]$, or the number of parameters that has been passed to the predicate does not correspond to its definition. In this case, no services or resources are consumed and the predicate is evaluated as `undefined`.

2. The predicate is triggered accordingly, and the external corresponding services or resources run successfully. In this case, the parameters are instantiated and the predicate is evaluated as `true`. In this case, there is also consumption of resources and/or services.

3. The predicate is triggered accordingly, but the external corresponding services or resources fail. In this case, the predicate is evaluated to `false`. Notice that, in this case, since an attempt to execute the corresponding external services and/or consume external resources has actually occurred, the predicate parameters may be instantiated to values corresponding to the partial execution of services and utilisation of external resources. In other words, when a predicate fails, there can occur side effects as a consequence of its evaluation.

The resource we have in Jam Session for the orcherstration of resources is the construction of *protocols*. A protocol is a structured chain of events, and it specifies who should come after who. Events can be of four types:

1. Orders for agents to move.
2. Triggering of predicates.
3. Triggering of auxiliary protocols.
4. Combination of previous events by means of connectives, as explained below.

Protocols are linked to locations. A request to trigger a protocol can result in the following alternative situations:

1. The requested protocol is not actually defined for the specified location. In this case, the obtained truth value is `undefined`.
2. The requested protocol is defined for the specified location. In this case, the specification of the protocol is retrieved and evaluated, based on the algebraic rules that govern the behaviour of the connectives that are used in the specification of the protocol. The result of the evaluation determines the truth value that shall be assigned to the protocol.

A protocol is denoted as

$$[Location]protocol(Term_1, ..., Term_r)$$

Here, *Location* stands for the location to which the protocol is connected, and $Term_i$ are first order terms as before. The expected utilisation of terms in the specification of protocols is for parameter passing accross predicates and auxiliary protocols.

For the specification of protocols, we employ four binary connectives, whose behaviour is as follows:

1. Classical conjunction ($\wedge$): classical conjunction builds the greatest lower bound of the truth values of its operands: given two (structures of) events α and β, and assuming that we can assess the truth values associated to each of them, then the truth value of $\alpha \wedge \beta$ shall be the greatest lower bound for the truth values of α and β.
2. Classical disjunction ($\vee$): classical disjunction builds the least upper bound of the truth values of its operands: given two (structures of) events α and β, and assuming that we can assess the truth values associated to each of them, then the truth value of $\alpha \vee \beta$ shall be the least upper bound for the truth values of α and β.

3. Sequential conjunction ($\sqcap$): sequential conjunction evaluates left to right, and employs different rules depending on the event to the right. The basic idea is to regulate the evaluation of chains of events, by taking into account the success or failure of orders for agents to move:

   (a) If the event to the right is an order to move, then the truth value of the event to the left is expected to be `true`. In other words, an order to move is expected to be triggered only if its antecedents have been successful. Intuitively, this resource has been implemented in Jam Session as a tool to regulate the global behaviour and security of systems. When an agent moves to a new location, previously available services and resources are locked up, and new services and resources are released. The sequential conjunction provides the means, in Jam Session, to implement conditional rules for the switching of available services and resources.

   More specifically, if the event to the right of a sequential conjunction is an order to move, then we have the following table for the evaluation of the truth value of the whole expression:

   - $\alpha \sqcap move(A, L_1, L_2)$:
     truth value of α is
     `undefined` $\Longrightarrow$ result is `undefined`.
   - $\alpha \sqcap move(A, L_1, L_2)$:
     truth value of α is either `false` or
     `inconsistent` $\Longrightarrow$ result is `false`.
   - $\alpha \sqcap move(A, L_1, L_2)$:
     truth value of α is
     `true` $\Longrightarrow$ order to move is triggered, end truth value is the result of classical conjunction between what comes out of the attempt to move and `true`.

   (b) If the event to the right is any other event different from an order to move, then the truth evaluation follows the rules of classical conjunction.

4. Sequential disjunction($\sqcup$): sequential disjunction evaluates left to right, and employs different rules depending on the event to the left. The basic idea here is to attempt different alternative solutions only if necessary. When a satisfactory truth value is achieved, the evaluation is resumed.

   More specifically, the evaluation of sequential disjunction goes as follows:

   (a) $\alpha \sqcup \beta$:
   truth value of α is `inconsistent` $\Longrightarrow$ `inconsistent`.

   (b) $\alpha \sqcup \beta$:
   truth value of α is `true` $\Longrightarrow$ `true`.

(c) $\alpha \sqcup \beta$:

truth value of α is either `undefined` or `false` $\Longrightarrow$ order to move is triggered, end truth value is the result of classical disjunction between both operands.

The specification of a protocol is a chain of events linked by connectives, thus forming nested pairs of the form $(\alpha_1\ conn_1\ \alpha_2)\ conn_2\ \alpha_3)...)\ conn_u\ \alpha_{u+1})$, in which α_i are events and $conn_j$ are connectives.

Sequential conjunction has priority over classical conjunction; both conjunctions have priority over sequential disjunction; all these three connectives have priority over classical disjunction.

Protocols can be threaded. This means that more than one single protocol may be running at any time. It shall be left to the programmer to ensure that the resulting concurrent groups of threaded protocols generate behaviours that are in accordance with the existing systems requirements for each application.

Just to see how the specification of a protocol looks like, we consider the following simple protocol:

$$[loc_1]prot(X,Y) ::= [loc_1, ag_1]pred_1(X) \sqcap move(ag_1, loc_1, loc_2) \sqcap [loc_2, ag_1]pred_2(Y) \sqcup [loc_1, ag_2]pred_3(X,Y).$$

In this protocol, we have two locations – loc_1 and loc_2 – and two agents – ag_1 and ag_2. There are also two variables that act as placeholders for specific values when the protocol is triggered.

The protocol executes as follows:

1. The predicate $pred_1$ must be defined for ag_1 in loc_1. If it is not, then the result of this evaluation is `undefined`, otherwise $pred_1(X)$ is evaluated, possibly producing as a side effect a value for X.

2. If the evaluation of $pred_1(X)$ is `true`, then agent ag_1 receives an order to move from loc_1 to loc_2.

3. If the attempt to move to loc_1 is successful, then the move receives a value `true`.

4. All results being `true` so far, then the predicate $pred_2$ is verified and attempted to evaluate, if this is the case. The end result of the protocol is the result of the classical conjunction applied to all events.

5. If the end result of the evaluation of this branch of the protocol results in `true`, then the evaluation resumes and this is the end result. Otherwise, the second branch of the protocol is verified, and the end result is evaluated as a classical disjunction between the two branches. If the first branch (i.e. the piece of the protocol that comes before the $\sqcup$ connective) is evaluated to `false` and the second branch is evaluated to `true`, then the overall protocol is evaluated to `inconsistent`; if the first branch is evaluated to `undefined` and the second branch is evaluated to `true`, then the overall protocol is evaluated to `true`; and so on.

If we take again the illustrative examples of section 1, we can sketch the corresponding protocols that could ground the corresponding intelligent interactive environments that solve them.

1. **Augmented Reality**: in order to build intelligent interactive environments to support augmented reality systems, we must keep track of the physical and the virtual location of every inhabitant of the environment, to ensure that the appropriate services and resources are the only ones available at all times. We must also have the means for users to consume services and resources.

   These features can be implemented by using three special sorts of protocols:

   (a) *Infinite loop protocols*: an infinite loop protocol can be written as a chain of events connected by sequential conjunctions, in which the last event is a recursive call to the protocol itself as an auxiliary protocol. Infinite loop protocols must be carefully designed, to ensure that, upon arrival to the recursive call, the global state of the environment – locations of agents, instantiations of terms etc. – has been returned to the initial state that could be found at the moment the protocol was first triggered.

   Infinite loop protocols can be employed, for example, to monitor the presence of agents in locations.

   (b) *Threaded protocols*: threaded protocols are collections of protocols that are triggered concurrently and compete for the consumption of resources and services. The design of threaded protocols can be quite complex, as we must ensure that the asynchronous consumption of resources and services does not lead to unexpected or undesired global states of the environment. In future work, we shall develop tools for the automated or semi-automated verification of threaded protocols, possibly based on simulated execution of protocols, resorting to techniques similar to those employed in model checking in general.

   Threaded protocols can be employed, for example, to monitor the migrations of all agents simultaneously, as well as to satisfy the requests from external users in multiuser scenarios. In these cases, the protocols can update persistent variables that stay as resources in specified locations, as if leaving messages in message boards (or clues in treasure hunting) for other protocols to access.

   (c) *User controlled protocols*: user controlled protocols are protocols that can be triggered directly by users through appropriate user interfaces. User interfaces can be also implemented as infinite loop protocols, whose external resources can check the value of user controlled parameters and trigger auxiliary protocols accordingly.

   Infinite loop, threaded and user controlled protocols can be combined to

implemented the necessary responsiveness of intelligent interactive environments.

2. **Virtual Worlds and Computer Games**: as referred to in previous sections, interactions through virtual worlds can be comprised as a special case of augmented reality, in which certain interactions do not occur. Therefore, similar programming resources to those referred to for augmented reality can be employed in order to effectively build virtual worlds and highly immersive computer games.

3. **Interactive Systems**: as also referred to in previous sections, interactive systems can also be envisaged as a special case of virtual worlds, in which the notions of location and mobility can be taken metaphorically as well as literally. Hence, the same programming resources can be employed for these systems.

In the next section we introduce a highly simplified example of system for the delivery of cross border public services, based on the notions above.

4 A Simple Illustrative Example

In this section we put Jam Session to work. Our goal is to show how it can be used to orchestrate the consumption of external resources, in order to compose complex systems.

Our illustrative example belongs to the domain of Electronic Government – more specifically, to cross borders e-GOV services. It is well known that the requests for special purpose visas for individuals to visit foreign countries can be quite complicated. Significant portions of the workflow to obtain a visa could be automated and presented to citizens through the web, thus creating opportunities to simplify the corresponding procedures, as well as to make the system more user friendly and cost effective. Furthermore, the web presentation of services and resources related to e-GOV should always be highly user friendly and directed to the broadest possible spectrum of individuals.

We have implemented a simple prototype to inform Brazilian citizens about the required procedures to obtain student visas to study in Chile[1]. In this prototype, we have five locations and one agent. Depending on what location the agent is, different predicates are available, which trigger external resources that provide speech-based information about the procedures to follow, as well as capture information from the user.

There are at least two intuitive interpretations of what is going on when we use this system:

1. The system can be envisaged as mimicking the actual physical interactions a user could go through while at the Chilean consulate in Brazil. Each

[1]We thank Prof. Rosa Alarcon, from Pontificia Universidad Catolica de Chile, for invaluable information about the workflow to obtain student and work visas to Chile.

location can be seen as the simulation of a desk, a clerk or a specific acitivity developed by a clerk at the consulate, and when the agent moves to a different location the user could have a sense of having moved to a different step in the process to obtain the visa.

2. The system can be taken more leisurely as a computer game. The user can feel as if solving a puzzle, which happens to be a carefully built one so that, as a side effect, once it is solved, the user may have learned about the information he/she needs to obtain the needed visa.

In both cases, the idea is to build a user friendly, fun to use, accessible for all system, whose utilisation would break barriers that users may face in order to access complex services.

Our prototype can only provide useful information to citizens. It would actually be simple to extend it to perform services that could support the provision of visas to citizens. All that would be necessary would be to replace certain services – which at the moment only provide relevant information to citizens – by services which could effectively perform parts of the required workflow in order to simplify the issuing of visas.

A prototype for the Jam Session language has been implemented in PROLOG, to build a demo system for the provision of information to study in Chile. In this prototype, the user is advised about what to do by an animated cartoon character. The messages that are spoken by the animated character are prerecorded messages[2]. A screenshot of the demo, in which the animated cartoon appears providing information to the user, is presented in Figure 3.

The five locations are organised as a tree, as depicted in Figure 4.

Location 1 is the entrance door to the system (and to the virtual environment that provides information to the user). When a user starts to use the system, the agent is placed in location 1, which also hosts a predicate that can trigger an external resource which salutes the user, provides information about how the system works and asks for some basic information.

Depending on what the user replies, the agent is directed to either location 2 or location 3. Once in one of these new locations, the agent can trigger the predicates that are hosted by that location, which shall provide the user with additional information and ask for additional answers. This way, as the agent hops from location to location, different system states are characterised, different resources are released, and different behaviours of the system can be perceived by the user.

[2]In order to create the spoken messages from written text, we have employed a commercial product called CrazyTalk©.

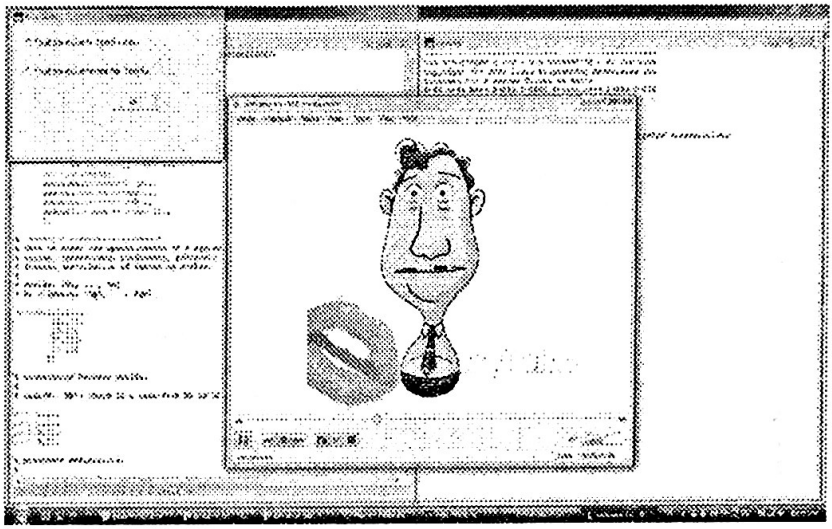

Figure 3: Screenshot of the demo system to inform about how to obtain student visas to Chile

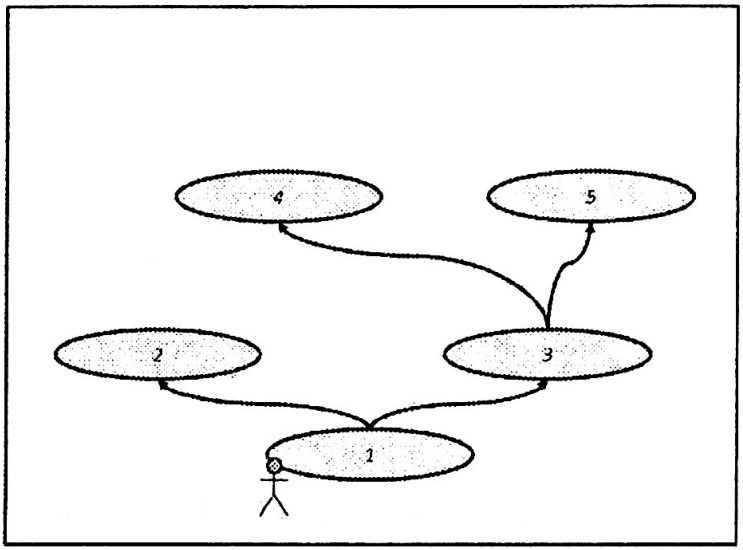

Figure 4: The structure of locations in the demo

In our demo, once the agent reaches one of the "leaf" locations, the system halts. More sophisticated protocols could be devised, in which e.g. paths connecting the leaf locations back to the entrance door could be added, and the protocols could loop back to the starting state to attend the next user.

A set of protocols that implement this demo can be as below. In these protocols, we use PROLOG standard notation for logical terms and variables. We also assume that the user interface captures user input and stores it as publicly accessible terms, which can be read by the protocols as values of the PROLOG fact $lastInput(Input)$. We have added some parentheses to simplify the reading of the protocols.

- The predicate $predWelcome(\ )$ triggers the external resource that contains an animation that greets the user, provides initial information to him/her and captures some initial user input.
- The predicate $predCompareString(X, Y)$ compares two strings, and returns `true` if they are identical or `false` otherwise.
- The predicates
  - $predShortStay(\ )$,
  - $predCompleteDocs(\ )$ and
  - $predGoToOffice(\ )$

  provide additional informations to the user and halt.
- The predicate $predDocsPending(\ )$ provides additional information to the user and waits for additional input, which then updates the fact $lastInput(Input)$.

  - $[1]protWelcome(\ ) ::=$
    $[1,1]predWelcome(\ ) \sqcap protMigrate2or3(lastInput(Input))$.
  - $[1]protMigrate2or3(lastInput(Input_1)) ::=$
    $([1,1]predCompareString(Input_1,$
    `I shall stay in Chile for 3 months or less`$) \sqcap move(1,1,2)$
    $\sqcap [1,2]predShortStay(\ )) \sqcup (move(1,1,3) \sqcap [1,3]predDocsPending(\ )$
    $\sqcap [3]protMigrate4or5(lastInput(Input_2)))$.
  - $[3]protMigrate4or5(lastInput(Input_2)) ::=$
    $([1,3]predCompareString(Input_2,$
    `I still must obtain some documents`$) \sqcap move(1,3,4)$
    $\sqcap [1,4]predCompleteDocs(\ )) \sqcup (move(1,3,5)$
    $\sqcap [1,5]predGoToOffice(\ ))$.

Evidently, for a simple system as the one depicted in this demo, much simpler solutions could be developed. We have used this example only as an illustration of how Jam Session works and can be used. The advantages of Jam Session over more traditional development tools becomes more evident as we scale up the complexity of interactions in the intelligent interactive environment that is being developed.

5 Conclusion and Future Work

In the present work we have identified a special class of problems, for which interesting solutions can be developed based on what we have coined intelligent interactive environments. We have also introduced the Jam Session language, which can provide the means to design and implement sytem solutions based on the notion of intelligent interactive environments.

Further empirical assessment of Jam Session must still be developed, to ensure that the expressiveness of the proposed language is appropriate for the envisaged solutions that shall be developed using this conceptual tool. Moreover, we have founded many of our arguments towards Jam Session on the view that friendly interactive environments can be useful to improve the quality of services provided by computational systems to broad spectra of users, as occurs, for example, in the development of systems for Electronic Government. This view shall also, in the future, be empirically assessed.

We have developed a small interpreter for Jam Session, written in PROLOG, which can cope with one protocol at a time. The code for this interpreter can be accessed from the Jam Session project webpage[3]. As can be appreciated by looking at the code that implements this interpreter, the Jam Session language is indeed simple, compact and easy to understand and to use.

This interpreter has been used to implement a prototype for a system that provides information to citizens interested in obtaining student visas to study abroad. The source code, as well as all required auxiliary resources to run this prototype, can also be downloaded freely from the Jam Session web site.

We are, at the moment, working on a more extensive implementation of Jam Session, that shall be capable of dealing with threaded protocols and multiuser, distributed environments. Future implementations of interpreters for Jam Session shall also be published for free access from the Jam Session project web page.

References

[1] Belnap, N.D.: A useful four-valued logic. Modern Uses of Multiple-Valued Logic. (Dunn, J. M., Epstein, G. - eds.). Dordrecht: Reidel (1977) 8-37

[2] Cardelli, L.: Mobility and Security. Lecture Notes for the Marktoberdorf Summer School – Foundations of Secure Computations. Bauer, F. L., Steinbrggen, R. (eds.). NATO Science Series, IOS Press. Germany (1999) 3-37

[3] Correa da Silva, F. S. Vasconcelos, W.: Rule schemata for game artificial intelligence. International Joint Conference IBERAMIA/SBIA. Springer-Verlag Lecture Notes in Artificial Intelligence 4140 (2006) 451–461

[3] `http://lidet.ime.usp.br/JamSession`.

[4] Esteva, M., Rodriguez-Aguilar, J. A., Sierra, C., Garcia, P., Arcos, J. L.: On the formal specification of electronic institutions. Agent-mediated electronic commerce. Dignum, F., Sierra, C. (eds.). Springer-Verlag Lecture Notes in Artificial Intelligence 1991 (2001) 126–147

[5] Guerra, C. A. N., Correa da Silva, F. S.: Semantic web services for intelligent responsive environments. Workshop on Intelligent Agents and Services for Smart Environments. (Correa da Silva, F. S., Bandini, S. - organisers). Proceedings of British Society for Studies of Artificial Intelligence and Simulation of Behaviour. Scotland (2008)

[6] Guerra, C. A. N., Correa da Silva, F. S.: A middleware for smart environments. Workshop on Intelligent Agents and Services for Smart Environments. (Correa da Silva, F. S., Bandini, S. - organisers). Proceedings of British Society for Studies of Artificial Intelligence and Simulation of Behaviour. Scotland (2008)

[7] Henderson, S., Feiner, S.: Evaluating the benefits of augmented reality for task localization in maintenance of an armored personnel carrier turret. Procs. IEEE International Symposium on Mixed and Augmented Reality. USA (2009) 135-144

[8] Kim, J., Lee, E., Thomas, T., Dombrowski, C.: Storytelling in new media: the case of alternate reality games, 2001-2009. First Monday, v. 14(6) (2009)

[9] Robertson, D.: Declarative Agent Languages and Technologies. Leite, J. A., Omicini, A., Torroni, P., Yolum, P. (eds.). (2004) 236-249

[10] Shapshak, M.: New approaches for mixed reality in urban environments: the CINeSPACE project. 5th International Conference - Virtual City and Territory. Spain (2009)

[11] Vasconcelos, W., Kollingbaum, M. J., Norman, T. J.: Normative conflict-resolution in multi-agent systems. Autonomous Agents and Multiagent Systems, v. 19(2) (2009)

[12] Vizzari, G.: Dynamic interaction spaces and situated multi-agent systems: from a multi-layered model to a distributed architecture. PhD thesis. University of Milano-Bicocca. Italy (2004)

RELATÓRIOS TÉCNICOS

DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO
Instituto de Matemática e Estatística da USP

A listagem contendo os relatórios técnicos anteriores a 2006 poderá ser consultada ou solicitada à Secretaria do Departamento, pessoalmente, por carta ou e-mail (mac@ime.usp.br).

Marco A. S. Netto, Alfredo Goldman and Pierre-François Dutot
A FLEXIBLE ARCHITECTURE FOR SCHEDULING PARALLEL APLICATIONS ON OPPORTUNISTIC COMPUTER NETWORKS
RT- MAC 2006-01 – Janeiro 2006, 18 pp.

Julio M. Stern
COGNITIVE CONSTRUCTIVISM AND LANGUAGE
RT – MAC 2006-02 – Maio 2006, 67 pp.

Arlindo Flávio da Conceição and Fabio Kon
EXPERIMENTS AND ANALYSIS OF VOICE OVER IEEE 802.11 INFRASTRUCTURED NETWORKS
RT – MAC 2006-03 – Junho 2006,

Giuliano Mega and Fabio Kon
DISTRIBUTED SYMBOLIC DEBUGGING FOR THE COMMON PROGRAMMER
RT – MAC 2006-04 – Junho 2006

Pedro J. Fernandez, Julio M. Stern, Carlos Alberto de Bragança Pereira and Marcelo S. Lauretto
A NEW MEDIA OPTMIZER BASED ON THE MEAN-VARIANCE MODEL
RT – MAC 2006-05 – Junho 2006, 24 pp.

P. Feofiloff, C.G. Fernandes, C.E. Ferreira and J.C. Pina,
"A NOTE ON JOHNSON, MINKOFF AND PHILLIPS' ALGORITHM FOR THE PRIZE-COLLECTING STEINER TREE PROBLEM"
RT-MAC2006-06 – Setembro 2006, 11 pp.

Julio Michael Stern
DECOUPLING, SPARSITY, RANDOMIZATION, AND OBJECTIVE BAYESIAN INFERENCE
RT-MAC2006-07 – Novembro 2006, 36 pp.

Cristiane Maria Sato, Yoshiharu Kohayakawa
ENTROPIA DE GRAFOS
RT – MAC2006-08 – Dezembro 2006 , 44 pp.

Julio M. Stern
LANGUAGE, METAPHOR AND METAPHYSICS: THE SUBJECTIVE SIDE OF SCIENCE
RT-MAC-2006-09 – Dezembro 2006, 35 pp.

Thiago A. de André and Paulo J. S. Silva
EXACT PENALTIES FOR KKT SYSTEMS ASSOCIATED TO VARIATIONAL INEQUALITIES
RT-MAC-2007-01– Março 2007, 21 pp.

Flávio Soares Correa da Silva, Rogério Panigassi and Carlos Hulot
LEARNING MANAGEMENT SYSTEMS DESIDERATA FOR COMPETITIVE UNIVERSITIES
RT-MAC-2007-02 – Maio 2007, 12 pp.

Alexandre Freire da Silva, Fabio Kon, Alfredo Goldman
THREE ANTI-PRACTICES WHILE TEACHING AGILE METHODS
RT-MAC-2007-03 – Maio 2007, 20pp.

Silvio do Lago Pereira e Leliane Nunes de Barros
PLANEJAMENTO BASEADO EM PROCESSOS DE DECISÃO MARKOVIANOS
RT-MAC-2007-04 – Maio 2007, 17pp.

Silvio do Lago Pereira e Leliane Nunes de Barros
DIAGRAMAS DE DECISÃO BINÁRIA
RT-MAC-2007-05 – Maio 2007, 16pp.

Carlos Alberto de Bragança Pereira and Julio Michael Stern
AN ESSAY ON THE ROLE OF BERNOULLI AND POISSON PROCESSES IN BAYESIAN STATISTICS
RT-MAC-2007-06 – Junho 2007, 39pp.

Flávio Soares Corrêa da Silva
ARGUMENTS IN FAVOR OF A CONTROLLED PLURALITY OF OFFICE FORMATTING STANDARDS
RT-MAC-2007-07 – Junho 2007, 9pp.

Silvio do Lago Pereira, Leliane Nunes de Barros and Fábio Gagliardi Cozman
STRONG PROBABILISTIC PLANNING
RT-MAC-2007-08 – Junho 2007, 25pp.

Silvio do Lago Pereira and Leliane Nunes de Barros
FORMALIZING PLANNING ALGORITHMS FOR TEMPORALLY EXTENDED GOALS
RT-MAC-2007-09 – Junho 2007, 22pp.

Flávio S. Corrêa da Silva
THEPROLOGPLAY: AN INTERACTIVE VIRTUAL ENVIRONMENT FOR ARTIFICIAL INTELLIGENCE AND RELATED FIELDS
RT-MAC-2007-10 – Setembro 2007, 12pp

Vanessa Sabino and Fabio Kon
*LICENÇAS DE SOFTWARE LIVRE HISTÓRIA E CARACTERÍSTICAS**
RT-MAC-2009-01 - Março 2009, 40pp.

Alexandre Noma, Ana B. V. Graciano, Roberto M. César Jr., Luis A. Consularo and Isabelle Bloch
INEXACT GRAPH MATCHING FOR SEGMENTATION AND RECOGNITION OF OBJECT PARTS
RT-MAC-2009-02 – Março 2009, 31pp.

Claudia J. Abro de Arajo and Flávio S. Corrêa da Silva
GOVERNMENTAL VIRTUAL INSTITUTIONS
RT-MAC-2009-03 – junho 2009, 19pp.

Cristina Gomes Fernandes e Rafael Crivellari Saliba Schouery
ALGORITMOS DE APROXIMAÇÃO E PROBLEMAS COM SEQÜÊNCIAS
RT-MAC-2009-04 – agosto 2009, 38pp.

Marcelo Finger e Glauber De Bona
UMA CONJECTURA REFUTADA SOBRE SATISFAZIBILIDADE PROBABILÍSTICA
RT-MAC-2009-05 – dezembro 2009, 18pp.

Alexandre Matos Arruda e Marcelo Finger
CARACTERIZAÇÃO DA INDEPENDÊNCIA CONDICIONAL EM LÓGICA MODAL
RT-MAC-2010-01 – Janeiro 2010, 9pp.

Flávio Soares Correa da Silva
JAM SESSION – KNOWLEDGE-BASED INTERACTION PROTOCOLS FOR INTELLIGENT INTERACTIVE ENVIRONMENTS
RT-MAC-2010-02 – Fevereiro 2010, 23 pp.