An Architecture for Unified Dialogue in Distributed Object Systems

A. Larsen, P. D. Holmes
Norwegian Computing Center (NR)
Oslo, Norway

Abstract

In traditional information systems, the user interface is controlled by one single application. In distributed systems, several distributed components may want to influence the appearance and logic of the user interface. This paper describes a Unified Dialogue Architecture which enables several distributed components to control the logic and contents of the user dialogue while keeping the dialogue consistent. This architecture is a practical example of using dialogue agents, CORBA and Java. Details are described in connection with a large domain-specific distributed system called SPACE. Discussion is also provided as to other manners in which this architecture may be implemented, followed by a discussion concerning other problem areas in which the Unified Dialogue Architecture can be effectively applied.

1 Introduction

In traditional information systems, the user dialogue is usually controlled by one single application. The parts of the application controlling the user dialogue may be thought of as a *dialogue system* [16]. In a distributed system, several *server components* may be part of the dialogue system. This paper considers distributed systems where each server component may be owned and maintained by separate organisations or parts of an organisation or both. These entities have something in common (i.e., the reason for building or using the system). Even so, *they want to control their own parts of the dialogue, with respect to content and logic, independently from the other components.*

In the system contexts addressed here, the complete system delivers a set of services. Any given service may depend upon functionality realised by co-ordination across some number (perhaps all) of the server components. Therefore we want to keep the interface consistent and uniform for the user, while still allowing the involved organisation(s) to fulfil their content- and logic-related requirements for the user interface. Alternative implementation approaches include:

1. Use of tightly integrated, yet different applications, each having their own user interface. Each could exploit the same basic system infrastructure and services, and employ common user-interface design "style guides" in order to try to achieve a relatively consistent look and feel across the different applications. As a drawback, this approach offers no facilities for co-ordination between the different component owners.
2. Extending the previous solution with an interface agent providing co-ordination facilities (an example is given in [18]). Here, endpoint co-ordination is achieved, but no facilities exist for tighter integration.
3. Designing only one client with one user-interface. Such an interface would have to be flexible enough to satisfy all the components' business objectives within a number of different use environments. At the same time it would have to maintain consistency amongst the interface's various graphical and media elements.

Traditional client-server systems achieve the latter to a certain degree. They do this by implementing a complete dialogue system in the client, sacrificing some flexibility by hard-coding the dialogue in the client. This paper describes a *Unified Dialogue Architecture* which enables *distributed dialogue systems* to deliver *unified dialogues*. A unified dialogue allows the user to conduct an orderly and consistent discourse with the system even though the dialogue is controlled by several independent components. Of equal significance is the condition that the dialogue itself is not known by the client until run-time. The distributed dialogue system enables the different server components' owners to modify their own parts of the dialogue - within the bounds imposed by the established semantic framework - without necessitating change or re-configuration of other parts of the system.

This paper begins with a presentation concerning the basic aspects of unified dialogue handling. The Unified Dialogue Architecture is thereafter discussed, followed a description of its use in relation to a large domain-specific system called SPACE (see [19], [6], and [7]). Interesting characteristics of other possible implementations of the

Unified Dialogue Architecture are then discussed, followed by a discussion concerning other problem areas in which the architecture can be effectively applied. Finally, the results are summarised, and the current status and thoughts about future work are presented.

2 Unified Dialogues

A *unified dialogue* is a user dialogue built up of parts controlled by different server components, while still retaining the look and feel of a single application. The goal of *unified dialogue handling* is to facilitate unified dialogues within distributed systems contexts. To achieve this, two objectives should be fulfilled:

1. the total dialogue should appear logically consistent to the user, and

2. response times should be similar to those of a single application.

To meet these objectives, several aspects must be considered. Table 1 summarises these aspects. Examples of how these aspects affect an actual implementation are given in section 3.3. Section 5 shows how these aspects can be implemented.

The design of the user interface itself is not included within table 1. It is equally important in this kind of system as in any other system. It is not taken into account here since this problem is not specific to unified dialogue handling. The reader can find this subject thoroughly treated within the systems development literature (see, e.g., [15]). The architecture described here is not dependent upon any specific type of user interface. Instead it may provide access to all facilities the client can offer. It also allows the user interface to be designed for a specific system context.

System context	In order to make the system (in particular the user dialogue) appear consistent and logical to its users, some sort of common understanding of the system must be established and agreed upon. This includes the purpose of the system, its functionality, the basic kinds of services and information to be made available through the system, etc. The decisions made in this regard are dependent upon the system context and those decisions influence the need for co-ordination during both system development and system execution.
Service architecture	Aspects of the service architecture relevant for dialogue handling include the responsibility and dependencies between server components, as well as for dialogue content and behaviour.
Interaction model	The interaction model defines how the user interacts with the system and how the results are processed.
Execution model	The execution model defines how client and server objects interact during the execution of the user dialogue.
Implementation and infrastructure	In order to realise this kind of system, several infrastructure and implementation issues arise. These include programming languages, communication standards used, etc.

Table 1: Aspects of dialogue handling in distributed systems.

3 The Unified Dialogue Architecture

This paper describes the *Unified Dialogue Architecture*. The architecture is built using principles from *agent-oriented programming* [17]. The Unified Dialogue Architecture is situated within the context of a larger *service architecture*. The service architecture consists of *server components*, a *common client*, and the necessary communications infrastructure. With respect to the Unified Dialogue Architecture, the server components are responsible for delivering *dialogue agents*. The common client provides a *dialogue agent environment* able to host multiple dialogue agents. The dialogue agent environment also provides facilities for screen management and co-ordination between dialogue agents.

The dialogue agent environment together with the dialogue agents facilitates a consistent dialogue, thus mimicking a single application. The use of dialogue agents also influences response time. Retrieving dialogue agents across the network induces network-specific time dependencies; during the agents' local execution, however, the client's responsiveness may be similar to that of a single application.

The Unified Dialogue Architecture bears resemblance to the Open Agent Architecture [2]. The difference lies in that whereas the Open Agent Architecture focuses upon building a complete agent- oriented service architecture, the Unified Dialogue Architecture aims at facilitating unified dialogue within any service architecture.

3.1 Dialogue Agents

A dialogue agent is a *server-delivered communicative autonomous interface agent*[1]. A traditional interface agent is assumed to be acting on the basis of the user's agenda, representing the users' intentions [10]. A server-delivered interface agent is different in that it reflects the needs of the service providers in fulfilling the user's assumed intents, not the user's own agenda. A dialogue agent is communicative in that it may interact with the user and other dialogue agents through the dialogue agent environment. This approach may therefore be best suited for specific services where the user's intent can be at least partially assumed[2].

Each dialogue agent contains a *dialogue script* defining the content and logic of their part of the total user dialogue. In addition, each dialogue agent must know how to communicate the results back to the relevant server components for processing, and how to present the result of this process.

3.2 Dialogue Agent Environment

The dialogue agent environment provides facilities for presenting unified dialogues. This includes facilities for co-ordinating the total dialogue between the different dialogue agents. It co-ordinates screen layout, questions asked, and user-supplied replies. This co-ordination is achieved using rules and policies for screen layout, common name spaces for variables and questions, and rules for handling conflicts between dialogue agents. In a specific system context, additional facilities may be important and thus part of the dialogue agent environment. Other aspects may involve policies and decision defined by central server components. Such co-ordination must be considered in the service architecture.

The dialogue agent environment is also responsible for retrieving dialogue agents and executing them. Retrieval of dialogue agents is achieved through the service infrastructure.

In an implementation of the Unified Dialogue Architecture, the dialogue agent environment can be realised as part of a Java applet. This means that security issues are handled by the web-browser's security mechanisms in addition to restrictions imposed by the service infrastructure and the dialogue agent environment itself. It also means that the dialogue agent environment itself may be implemented as an agent, delivered by appropriate server components.

Because of restrictions in Java applets, the service architecture must be available through server components on the host delivering the Java applet, or through a proxy on the same host. The execution of dialogue agents is dependent on the implementation chosen. Possibilities include special scripting languages and pure Java code.

[1]See [3] for a classification of agents.

[2][20] gives an account of the differences between the users' actual behaviour and intentions, and the assumptions built into the system.

3.3 Semantic Framework

To design an interactive user dialogue which appears logically consistent when controlled by different server components, it is extremely advantageous - though not necessary - to establish a shared semantic framework for the dialogue. Such a framework *is* necessary, however, in order to facilitate semantic co-ordination between dialogue agents.

Rigorously applied, a shared semantic framework ensures semantic equivalence across dialogue agents supplied from distributed components. This enables such agents the possibility of reusing one anothers' results. That is, information supplied by the user - as a result of a dialogue process generated by one dialogue agent - can be employed by other agents and within other server components.

Development of a shared semantic framework can help alleviate - though perhaps not cure - eventual problems related to logical consistency within the user-interface. To help illuminate this, we first point out that the dialogue-related part of each server component must be allowed to specify its own semantic domain; this naturally includes the elements within that domain. In addition, each server component must be allowed to have its own perspective as to what constitutes 'logical consistency' amongst those elements.

Figure 1 provides an example which illustrates the elements within the semantic domains of each of components A, B and C. During a system-wide co-operative effort to identify common semantic elements, it is discovered that Domains A and B have elements $e5_A$ and $e5_B$, respectively - elements which are essentially equivalent from a semantic perspective. As part of the co-operative effort, one of these elements is selected (or a new one agreed upon), and thereafter "renamed" $e5$. From this point on, $e5$ can be employed by Domains A and B (or any other Domain) as a common, *well-defined* element within a system-wide, shared semantic framework. In the same way, $e6$ is also identified and defined as a common element. Section 4.5 will discuss how these elements relate to questions for the user within the SPACE system.

Thus far, the co-operative effort in this example has led to the identification and definition of some common semantic elements (e.g., $e5$ and $e6$). Still, there remains the problem that each server component must be allowed to have its own perspective as to what constitutes 'logical consistency' amongst those elements. From figure 1, it is clear that certain elements are not semantically relevant in

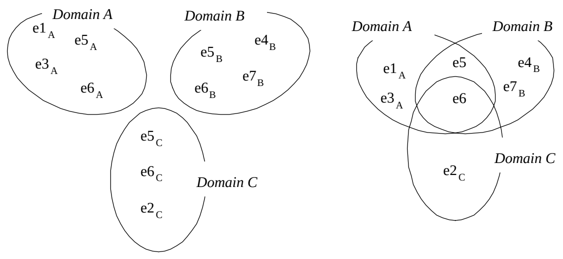

Figure 1: Development of a shared semantic framework

all Domains (e.g., $e3_A$ is not relevant within Domains B nor C).

When mapping these element to questions which each server component wishes to ask of the user, it is not possible to claim that a shared semantic framework can guarantee complete logical consistency within the user-interface: each individual Domain has its own consistent logic but, when all Domains are viewed together as a whole, the union of these Domains may not be logically reconcilable. This is the theoretical perspective.

From a practical working perspective, however, we argue that a shared semantic framework can help alleviate eventual problems related to logical consistency within the user-interface. We argue that the development and use of a shared semantic framework increases the likelihood that it is possible to construct distributed dialogues which - though not *completely* reconcilable from a theoretical perspective - *may co-exist within a user interface without confusing or disturbing the user*.

4 The SPACE Dialogue Architecture

The Unified Dialogue Architecture arose as an essential element within the infrastructure required within the SPACE project (Single Point of Access for Citizens of Europe) [7]. Before addressing the various aspects arising from the requirements of unified dialogue handling, this section opens with a brief introduction to the SPACE Project. This introduction should help provide a concrete context in which to understand the role and necessity of the Unified Dialogue Architecture with respect to SPACE and, perhaps to other problem areas as well.

4.1 The SPACE Project: Background

SPACE addresses the following problem: Within the European Community, the Maastricht treaty grants European Citizens free movement between Member States. These citizens currently face a number of administrative barriers when planning and carrying out such moves, however. These include:

1. ascertaining precisely which rules and regulations are applicable to their own, particular moving situation; and,

2. acquiring the certified documents which will be required in order to register (and de-register) themselves with a variety of Administrative agencies/sectors in the destination (and departure) countries. In general, such documents contain information about the Citizen which is stored within electronic archives controlled by such various Administrations.

In order to receive required information, deliver or certify documents, etc., the conditions above render it necessary for Citizens to contact numerous Administrative agencies. The Citizen may even have to visit some such agencies more than once.

One of the main objectives of the SPACE Project has been to demonstrate the possibility of providing a Single Point of Access for administrative services related to moving within the EU, based upon a telematic infrastructure for the retrieval, assembly and international exchange of such information. Using the SPACE system, authorised civil servants can help provide Citizens with one-stop shopping of administrative services. Using data which is more accurate and reliable, both Citizens and Administrations can benefit from a more efficient registration process.

To meet SPACE's main objectives, work has been done to develop:

- the system concept and the SPACE system's basic, architectural framework; and,

- a Demonstrator based on state-of-the-art technology, in order to illustrate the concept and its benefits, as well as help identify the functionality required for an operational system.

4.2 Information Products Delivered by SPACE

SPACE can deliver two fundamentally different kinds of information packages. Each of them respectively addresses the needs of the Citizen, as described in points 1 and 2 above. The first kind, *Advice Packages*, contains the kind of information one might normally publish and have available as brochures at some administrative office. For the end-user, the SPACE system can deliver Advice Packages containing General Advice and/or Tailored Advice.

General Advice is information which can be provided about moving within the EU when only knowing things such as the Citizen's planned departure and destination states. Tailored Advice, on the other hand, is information which is especially tailored to fit the Citizen's moving situation. Customising advice in this manner requires greater knowledge about the details of the moving case. Such details are gathered from the Citizen via an interactive dialogue process which runs on the Client. This dialogue process is realised through use of the SPACE Dialogue Architecture, whose design is based upon the principles of Unified Dialogue Architecture.

The second kind of information package is *Portfolios*. Within SPACE, the purpose of Portfolios is to contain data about the Citizen which has been electronically retrieved from databases owned by various Administrations. The data content itself is precisely tailored to match the information which the Citizen's destination state will require in order that the moving Citizen can be properly registered in the new state. As with Advice Packages, tailoring this kind of information requires acquisition of details about the Citizen's moving case through an interactive dialogue process with the Citizen. With such details available, the SPACE system has the capacity to determine precisely *which* data elements the foreign state requires as part of its registration process(es). Portfolios also contain a subpackage which is a collection of all data input by the Citizen during the interactive dialogue process.

4.3 System Context

The context of the SPACE System is support for moving citizens and the administrations involved. In SPACE the involved countries, sectors[3] and local administrations need to obtain information about the moving citizen in order to provide the necessary support. This is achieved through use of the SPACE Dialogue Architecture.

In SPACE, each sector owns and maintains its own server components, which includes design of its own dialogue agents. In order to provide dialogue support consistently and effectively across all sectors, the SPACE Dialogue Architecture must handle any inter-dependencies between the sectors within one country, as well as any inter-dependencies which may exist amongst countries.

4.4 Service Architecture

In the SPACE System, objects are distributed according to figure 2.

Each country has its own domain. Each domain contains one Master object and several Expert objects[4]. Each

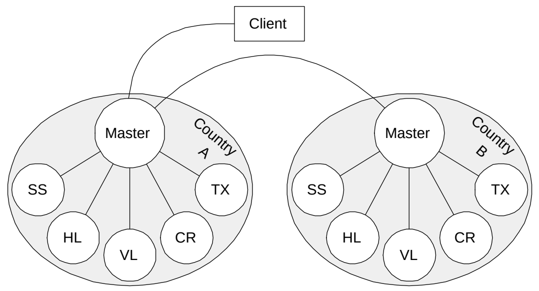

Figure 2: The SPACE Service Architecture

Expert is responsible for one administrative sector, such as social security (SS) or Tax (TX), within its own country. The Master object is responsible for serving clients and delegating requests to its own Experts and to the Master object of another country. It may also enforce country-specific and cross-sectoral policies. Any communication between domains is carried out by the Master objects.

All SPACE functions are initiated by the client by way of a local Master. This Master may call upon another country's Master and/or its own Experts. The results of these calls are collected and returned to the client. This means that all run-time co-ordination of content supplied by sector or country objects must be done in the Master object or in the client itself. To reduce the interaction between the client and the local Master, we have chosen to make the client responsible for this type of co-ordination.

This architecture makes it easy to add new (or remove existing) sectors and countries. It also caters for differences between countries in that the sectors are logical abstractions, not necessarily reflected in the governmental structure.

4.5 Dialogue Agents and the Dialogue Agent Environment

All run-time co-ordination of content supplied by Masters and Experts is performed by the client; this includes facilitating co-ordination between the dialogue agents through the dialogue agent environment. To achieve such facilitation, each valid question, along with its associated set of valid alternatives, is pre-defined[5]. This means that even though the actual dialogues are defined and implemented by the local administrations, all questions and their

[3]In SPACE, the sectors addressed are Social Security (SS), Civil Registration (CR), Health (HL), Vehicle Registration and Driver's License (VL) and Tax (TX).

[4]The Master and Expert object types are implemented in a generic manner. For this reason, these types can be reused across different countries and sectors. When initialised each object type is given a specific "personality"; that is, there is an instance of a Norwegian Master, a Finnish Master, a Danish Civil Registration Expert, a Finnish Tax Expert, etc. Several actual instances of each personality may be available to provide resilience and load-balancing. For further details, see [6].

[5]Type-in is allowed as a valid alternative to some questions.

respective valid alternatives must be co-ordinated between the participating administrations.

Briefly looking back, this pre-definition and co-ordination activity is precisely that work aimed to develop (one part of) a shared semantic framework within SPACE. In this case, each element scrutinised for semantic equivalence across Domains is a question along with its set of valid alternative replies (see section 3.3). For SPACE, the most important reasons for this effort are to ensure that:

1. even though several administrations may want to ask some of the same questions, the citizen only sees such questions once, and
2. when one question is posed and answered, all interested administrations know how to interpret the answer.

Pre-definition of questions and their alternatives is performed in two steps. First, each question and each alternative is assigned a unique identifier. Second, an association between a question and its valid alternatives is created. This association is referenced using the question's identifier. For example, assume one wishes to ask a user: *"What is your civil status?"*, along with a set of valid alternatives: *"Single"*, *"Married"*, *"Legal partner"*, *"Widow/widower"*. The question itself could be given the identifier *civ_stat*, and the alternatives identifiers such as *single*, *married*, *partner* and *widowed*, respectively.

All the questions, alternatives and their associations are stored and published within a central *question repository*. The repository is available during execution as well as during dialogue development. The dialogue agents only provide the identifier for the questions - and thus the associations - they need the client to present for the user. It is then up to the dialogue agent environment to retrieve the language-dependent renderings for each of the questions and alternatives. Incidentally, these unique identifier also facilitate multi-lingual dialogues (see section 4.9).

The SPACE Dialogue Agents are written in a language called SDDL[6]. SDDL defines three basic building blocks: question references, parameters and events. A question reference refers to a specific association of a question and its valid alternatives. Each question also has an associated parameter which has the same name as the question's id; the parameter's value is automatically set according to the answer supplied for the question[7]. Parameters are internal variables for storing data. To enable run-time co-ordination, all parameters are located in the same namespace. This means that a specific dialogue agent may have access to parameters set by previously-executed dialogue agents, as well as those set by its own contemporaries.

Pre-conditions can be specified in SDDL, making it possible to create dependencies amongst questions, such that the presentation of any question can be made dependent upon the answers supplied for other questions.

Events are actions that occur based upon certain tests; such tests usually involve a parameter value being logically or arithmetically compared to some other value. Events may be used to set values for one or more other parameters.

The SPACE dialogue agent environment is capable of retrieving and executing SDDL scripts. It knows where to place relevant questions on the screen, how to order and group them, and how to propagate answers supplied by the users to the different dialogue agents. It is also responsible for controlling when to retrieve new dialogue agents, and for handling the results of the dialogue.

4.6 Interaction Model

The SPACE Client user interface is built around some central design principles. These include a task- oriented, single screen-page approach; and, a "what-you-can-see-is-what-you-can-do-right-now" principle [5]. These influence the dialogue mechanism in that

1. everything must be designed to fit on one screen-page, and
2. some sort of logical order within tasks must be implemented.

The single screen-page used is designed with general screen areas available for different purposes. There are areas for user input, task-flow and system control, as well as for displaying help texts and information retrieved from the system. With respect to dialogue handling, the user input area is the most important screen area. Here, questions are posed to the user using standard screen components such as input fields and drop-down boxes.

When a question has been answered, pre-conditions may render other questions relevant; in such cases, these other questions are immediately displayed within the user input area. This behaviour is controlled by the dialogue agents. When all relevant questions have been answered, the client uses the user- supplied replies to retrieve information from the server components (see section 4.2).

4.7 Execution Model

In the SPACE System, the total dialogue is divided into levels. Each level has specific sources and dependencies. Table 2 shows the sources of dialogue agents for each level,

[6] The SPACE Dialogue Definition Language. A small example is given in Table 3.

[7] For example, when a user answers *"Married"* to the question about his/her civil status, the parameter *civ_stat* is assigned the value *married*.

Level	Source	Dependencies
0	Local Master	All local Masters deliver the same level 0 dialogue agent. This agent is responsible for European and cross-country co-ordination.
1	Country Master	Each country can deliver a level 1 dialogue agent. Which country Master(s) do so is determined by the replies supplied to the level 0 dialogue. Level 1 dialogue agents are responsible for inter-country and cross-sectoral co-ordination.
2	Sector Expert	Each sector Expert can deliver a level 2 dialogue agent. Which Experts do so is determined by the replies supplied to the level 0 and level 1 dialogues. Level 2 dialogue agents are responsible for inter-sectoral co-ordination.

Table 2: SPACE Dialogue Levels

dependencies between different dialogue levels, and the co-ordination responsibilities of each level.

In the SPACE System, the dialogue levels and the service architecture structure are essential in enforcing European, country, and sector-level policies. This is achieved partly by the strict communication patterns, and partly by retrieving and executing one level of dialogue to completion, before a new level is loaded. Prior to retrieving a new level of dialogue, the system fixes the values of all parameters set by agents from earlier levels. Fixing parameter values in this way is essential, since the script contained within each new dialogue agent can logically depend upon those values[8].

When the SPACE Client initialises itself, it contacts its local Master and retrieves the level 0 dialogue agent. The agent is instantiated, textual renderings for the questions are retrieved by the local Master and the dialogue is executed. When all the level 0 questions have been answered, parameters set by the level 0 agents are fixed, and level 1 dialogue agents are thereafter retrieved from the destination country Master. The level 1 dialogue is then interpreted, rendered and executed. At this point, the parameters set by both the level 0 and level 1 dialogue agents are fixed, prior to the retrieval of the level 2 dialogue agents.

For level 2 dialogue agents, the destination country Master is called and the call is propagated to all the sector Experts in that country. The Master collects the dialogue agents from those Experts and returns them to the client. The client interprets, renders and executes this dialogue. If a question is requested by more than one dialogue agent, it will be shown at most once in accordance with a *logical union* of any pre-conditions.

[8]See the discussion of ‘Logical Selectors‘ in section 4.8.

Use of a logical union in these circumstances is judged to be the most fair manner by which to resolve pre-conditions supplied from different dialogue agents. Ultimately, such pre-conditions could be in conflict with one another, since each agent must be allowed to maintain its own view as to what constitutes logical consistency. Considering Figure 1 and the user interface, this equates to a simultaneous presentation of elements from different Domains - in this case, questions to the user and their valid alternatives.

Here, the total set of questions displayed to the user could implicitly contain logical conflict or, of equal significance, contain conflicts which disturb the users perception of and/or understanding of the system. As mentioned earlier, we argue that work aimed at developing a shared semantic framework helps reduce the likelihood of such occurrences. In SPACE, certain problems of this kind were eliminated in advance through the use of dialogue levels and by mapping these levels to the server component hierarchy (see figure 2).

Table 3 shows a small example of a sequence of dialogue scripts. The level 0 dialogue was delivered by a LocalMaster. The level 1 dialogue may have been delivered by a Norwegian CountryMaster and level 2 by a Norwegian Vehicles and Driver License Expert.

Level	Script

0	`ask departure`
	`ask destination`

1	`ask age`
	`ask citizenship`

2	`ask drivers_license`
	`   if age >= #18`
	`event age < #18`
	`   set drivers_licence = no`

Table 3: A small SDDL example

Dialogue agents are one of two parts of the dialogue execution. The result of the first part of the dialogue execution is a number of answers to question. This constitutes a *specification* of the moving citizen’s situation. The SPACE Dialogue Architecture includes mechanisms for acting on the basis of this specification. For instance, such specifications can be used to retrieve advice about moving which has been tailored to the citizen’s situation. In addition, these specifications can be used to determine exactly which data elements will be required, by the destination administrations, for Citizen registration (see section 4.2). For example, the result of the dialogue in table 3 may be used to retrieve advice about how to handle driver's licenses in Norway.

4.8 Implementation and Infrastructure

The SPACE System is implemented as a distributed object system. The client is implemented as a Java applet[9] while the server objects are implemented using C++. Distribution is realised using a CORBA 2.0 compliant orb[10]. In the basic system there are generic implementations of a full Master and of Experts. As mentioned on page 5, these generic implementations may be given specific "personalities" during initialisation. All communication between the client and the different server objects is handled by a Master object. Figure 3 shows the relevant interfaces for this object.

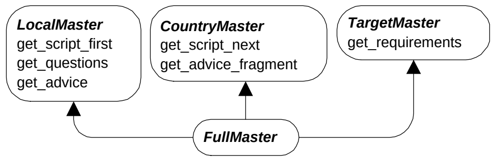

Figure 3: The SPACE Master Interface

Level 0 dialogue is retrieved by the *LocalMaster.get_script_first* method. All textual renderings for the dialogues are retrieved using *LocalMaster.get_questions*. Level 1 and 2 dialogues are retrieved using *CountryMaster.get_script_next*. For level 2 dialogues this call is propagated to the country's Experts which all have similar methods. The return value for *LocalMaster.get_script_first* and *CountryMaster.get_script_next* is an octet array containing a script written in SDDL. *LocalMaster.get_questions* returns a sequence of questions with valid alternatives.

During the dialogue execution, the SPACE Client has a partial specification of the moving citizen's situation. This specification may be used by the Master call:

CountryMaster.get_script_next The partial specification used here consists of the answer to previous dialogue levels (0 and possibly 1). This specification is used by the involved objects to determine exactly which dialogue script to return.

After the dialogue execution, the SPACE Client has a complete specification of the moving citizen's situation. This specification may be used by the Master calls:

LocalMaster.get_advice Here the specification is used to construct an Advice Package, a document providing advice which has been tailored to the citizen's moving situation.

TargetMaster.get_requirements Here the specification is used to determine exactly which information elements will be required by the destination administrations.

The central mechanism for determining which elements apply to a given specification is called a *Logical Selector*[11]. Within the services provided by the Master and Expert object instances, the purpose of the Logical Selectors is to perform analysis of the replies supplied during dialogue. The results of such analysis is to determine either

- which new dialogue agent to return to the client (i.e., the *get_script_next* service);
- which advice is relevant to the specification (i.e., the *get_advice* and *get_advice_fragment* services); or,
- which data elements are required for registration within a foreign state (i.e., the *get_requirements* service, see also section 4.2).

In the SPACE System, dialogue agents and logical selectors are stored in database tables. Each object type (LocalMaster, CountryMaster, Expert and so on) has a specific set of database table structures associated with it. Each object personality (Norwegian CountryMaster, Finnish Civil Registration Expert) has *specific instances* of these tables. Each LocalMaster and CountryMaster personality has its own *dialogue_definition_table*, containing all its dialogue scripts. In addition each CountryMaster has a *dialogue_selector_table* containing the LSDL expressions used to determine which dialogue scripts to return. Similarly there exists *advice_tables* and *advice_selector_tables* as well as *requirements_tables* and *requirements_selector_tables*. In addition there is a common repository containing things like the question repository.

As a result of this generic design, all that usually has to be done to change the behaviour of the system is to change the database content. In addition, the participating administrations may change their own tables independently. The underlying CORBA infrastructure even allows administrations to change their own implementations, as long as they strictly abide to the specifications of the service interfaces.

4.9 Other Aspects

The aspects described above are the generic aspects of the SPACE Dialogue Architecture. The architecture also

[9]The SPACE Client employs JDK 1.0.2, making it suitable for most web-browsers.
[10]DAIS 3.2 from ICL.
[11]Logical Selectors are written in the Logical Selector Definition Language, a subset of SDDL.

includes support for other aspects more specific to the SPACE context.

1. Multi-lingual support: The SPACE System is designed for international use, thus support for multiple languages is an inherent part of the user-interface design. For the dialogue, this is achieved by including several languages within the question repository.

2. Information from databases (e.g., in SPACE, those owned by various public administrations): The SPACE Dialogue Architecture includes support for retrieving values for parameters from these kinds of databases. This means that instead of asking the user for his 'age' and 'marital status', corresponding values for these parameters can be retrieved.

5 Other Implementations of the Unified Dialogue Architecture

The SPACE Dialogue Architecture is only *one* implementation of the Unified Dialogue Architecture. The architecture can also be implemented in other ways, and applied to other domains. The following sections address the different aspects of dialogue handling with respect to these issues.

5.1 System Context

The Unified Dialogue Architecture may be employed within any system context which involves the use of distributed systems. A given system context will influence all the other aspects of unified dialogue handling. Even so, it is possible to implement one dialogue architecture which can facilitate several different system contexts. Here, we will briefly describe how the SPACE implementation may be applied to system contexts other than the original.

The SPACE Dialogue Architecture described in section 4 is not exclusive to movement within the EU. It may also be relevant for supporting movement between other nations as well as movement within a single country. The solution is not restricted to supporting movement either. For instance, the advice functionality can be applied to a wide range of subjects. The only limitations are that sensible dialogues can be constructed and that the resulting specification can be used to tailor an advice (or any other information) package.

5.2 Service Architecture

The Unified Dialogue Architecture is not strongly connected to any kind of object model or service architecture. The dialogue architecture can handle arbitrary object and invocation models as long as each object adheres to the specific interaction and execution models; use of interoperable implementations and infrastructures is an obvious precondition.

The SPACE system illustrates a good example of how system context can influence service architecture. The SPACE Service Architecture is strongly influenced by administrative and security issues. The different object types are based upon logical abstractions of existing governmental and administrative structures. Security and policy issues dictate the division into countries, as well as any further restrictions on object relations and method invocations.

5.2 Interaction and Execution Models

In implementing the architecture, the interaction and execution models should be considered together. Here we exemplify extensions of the SPACE Dialogue Architecture. Other implementations may of course use other models.

One possible extension is to relax the restrictions upon the number of dialogue levels. This yields an arbitrary number of levels based on dependencies and arbitrary sources for each level. In the context of the SPACE system, this would mean that every server component could deliver dialogue agents for level 0 or any other level. It also means that the number of levels would not be not fixed.

Another possible extension is to relax the strict task-order (first complete the dialogue, then process the results). This could yield a more interactive system, where required elements, advice or other specification-dependent data are shown on the screen as they become relevant. Here, the execution model should be taken into careful consideration. Depending on the object model, this kind of interaction could lead to a great deal of communication between the client and the server objects, or to very large dialogue agents.

A third possibility is to make the dialogue agents responsible for controlling when to retrieve new dialogue agents. This removes the level restrictions completely while still retaining the necessary co- ordination facilities. Another option is to extend the support for input components to graphical constructs, voice recognition, etc.

5.3 Dialogue Agents and the Dialogue Agent Environment

Capabilities of dialogue agents and specific characteristics of the dialogue agent environment must be considered when implementing the Unified Dialogue Architecture. One possibility is to extend SDDL to achieve the functionality desired. Another possibility is to use a standard scripting language such as Tcl/Tk [14]. A third possibility is to implement the dialogue agents in Java. There are essentially three ways of approaching this latter alternative:

1. Use of Java source code. This means that the client must be able to compile and execute this code. (Requires compilation facilities in the client)

2. Use of static Java executables. This means downloading a class file, instantiating and executing it. (Available in JDK 1.0.2)

3. Use of dynamic Java executables. This means serialising a running object on the server, transferring it to the client and executing it there. (Available in JDK 1.1)

Of these, the two latter ones are the most interesting ones since the necessary features are already an integral part of Java. The most fascinating aspect of this solution is the ability for the client to provide common services for dialogue agents and putting the logic and content inside server-provided Java classes. This enables dialogue agents to contain method calls necessary to invoke services on server objects. This feature may be very important in cases where the server objects are heterogeneous or when they employ different types of communication (e.g., CORBA, RMI, DCOM).

A possible extension to the dialogue environment could be to allow parts of the environment to be made up of special agents. This would allow some of the service providers to provide their own special parts of the dialogue environment, whilst sharing the basic environment facilities. This could facilitate co- ordination and interoperability in complex and highly heterogeneous systems.

5.4 Implementation and Infrastructure

The implementation and infrastructure are of course highly dependent upon the other aspects. Still, some general observations can still be made. For the server components, any implementation method suitable to the local system environment may be used, as long as a suitable infrastructure can be built.

Using Java in the client offers significant advantages. This includes the ability to run the client in most web-browsers, built-in support for implementing dialogue agents in Java, and pre-defined and adjustable security features. This does not mean that other programming languages may not be suitable, especially in homogenous environments (e.g., Microsoft Windows based environments).

The SPACE System uses CORBA as its communication platform. The Unified Dialogue Architecture is not strongly connected to this standard. Other ORB technologies (such as Microsoft's DCOM) may also be employed. The only requirements are that the different objects must be able to talk to each other and pass dialogue agents amongst themselves. Thus even HTTP or pure transport protocols such as TCP/IP may be used.

6 Application Areas for the Unified Dialogue Architecture

The Unified Dialogue Architecture is independent of system context. When applied to a system context, however, there is an implication for system co-ordination. These areas include:

- co-ordination across component owners during design of the dialogue agents,

- administration and maintenance of dialogue agents, and

- runtime co-ordination within the dialogue agent environment.

Because of its flexibility, the Unified Dialogue Architecture may be employed in a wide variety of distributed object systems. It offers clients wherein the user dialogue is controlled by dialogue agents, not hard-coded into the client. It may be particularly well-suited to heterogeneous system environments. Possible applications of the Unified Dialogue Architecture include:

- Domain-Specific Systems. The Unified Dialogue Architecture can enable consistent, adaptable interfaces for distributed administrative systems such as a banking, accounting or case-handling. Other domain-specific applications are also possible.

- Unified Interface to Search Engines. This architecture can be used to build intelligent interfaces for search engines. This may be particularly useful in cases where the underlying databases are heterogeneous. Each server can then deliver its own dialogue script for execution. This approach can be used to develop a single, unified interface for things like Z39.50 and X.500 engines, and web-searching tools. Similar solutions are shown in [12] and in [18].

- Information Package Construction. This is equivalent to the advice functionality of SPACE. Several server objects know how to deliver different parts of a package and may employ the dialogue architecture to tailor their parts.

- Surveys and Direct-Marketing. Since the Unified Dialogue Architecture can deliver specifications of the user, it can also be used to facilitate survey or direct-marketing services.

The Unified Dialogue Architecture is particularly well-suited for access to services through a central access point. This includes access through an information kiosk (e.g., customer guidance in a shopping mall), and access through a common web-service (e.g., to government information normally open and available to the public).

7 Conclusion

The trend towards net-centric computing introduces new challenges in systems design. The Unified Dialogue Architecture is a significant contribution to this type of design. It enables a client which can handle a wide range of different user dialogues based on input from server components. The logic and content of the dialogues is not determined by the client but rather by the server components themselves. Furthermore the user dialogue is not known by the client until run-time. This enables new systems to be built which employ distributed object technologies, while still giving the impression of a single, consistent application.

The object-oriented solution described is very flexible, yet still concrete enough to be applied to a wide range of problems without large amounts of re-design work. Some of the core functionality is available as easy to use components. Once a system is built, it is easy to add (or remove) server components without changing the existing ones.

8 Status and Future Work

The SPACE Dialogue Mechanism as described in section 4 has been implemented and tested. The Unified Dialogue Architecture as outlined in sections 3 and 5 has been partially designed. Future work includes:

- Formalisation of specification and usage guidelines based on systems development research. [9] presents open implementation design guidelines which may be relevant. [4] discusses how to hook into object-oriented application frameworks. This may be useful in designing usage guidelines.

- Consideration of dynamic interface issues such as look and layout [8], as well as other types of user input [13]. This includes investigating possibilities within newer Java releases (i.e., Java Foundation Classes) and other libraries.

- Following up and implementing some of the aspects described in sections 3 and 5. This includes formalising the dialogue agent environment and refining reusable components in the service architecture and in the dialogue agent environment.

- In the SPACE Dialogue Architecture, the dialogue and its use are strictly separated. One possible area of interest may be to automatically generate dialogue agents based on requirements for the specifications (i.e., how parameters are dependent upon each other). Here techniques like those described in [1] may be useful.

- Tool support for building the architecture may be useful. This includes tools for building dialogue agents (like in [11]), for defining and building the basic service architecture and for customising the basic dialogue agent environment (i.e., the common client).

Acknowledgements

The architecture described here was designed as part of the SPACE project [6, 7], therewith partially funded by EU's Telematics Applications Programme.

Thanks to Stein Myrseth for valuable discussions about the design of this architecture. Thanks to Michael Gritzman and Jannicke Riisnæs for feedback on this paper.

References

[1] P. Castells, P. Szekely, and E. Salcher. Declarative models of presentation. In J. Moore, E. Edmonds, and A. Puerta, editors, *IUI'97: 1997 International Conference on Intelligent User Interfaces*, pages 137–144, 1997.

[2] P. R. Cohen, A. J. Cheyer, M. Wang, and S. C. Baeg. An open agent architecture. In O. Etizoni, editor, *Proceedings of the AAAI Spring Symposium Series on Software Agents*, pages 1–8. American Association for Artificial Intelligence, Stanford, California, March 1994.

[3] S. Franklin and S. Graesser. Is it an agent, or just a program?: A taxonomy for autonomous agents. In *Proceedings of the Third International Workshop on Agent Theories, Architectures, and Languages*. Springer-Verlag, 1996.

[4] G. Froehlich, H. J. Hoover, L. Liu, and P. Sorenson. Hooking into object-oriented application frame-

works. In *ICSE'97: Proceedings of the 1997 International Conference on Software Engineering*, pages 491–501, 1997.

[5] M. Gritzman, A. Kluge, and H. Lovett. Task orientation in user interface design. In K. Nordby, P. Helmersen, D. J. Gilmore, and S. A. Arnesen, editors, *Human-Computer Interaction, Interact'95*, pages 97–102. Chapman and Hall, London, Glasgow, 1995.

[6] P. D. Holmes, A. Larsen, M. Grtizman, L. Lundsgaard, H. Shardhammar, and R. Pohjosmäki. Space client and server specification: Infrastructure specification for the space technical platform. EU Project Report SPACE Deliverable D902/D903 (Confidential), 1997. Also reprinted as NR Technical Note Nr. IMEDIA/06/97, December 30, 1997.

[7] P. D. Holmes, A. Larsen, S. Myrseth, and M. Gritzman. SPACE: An architecture for coordinated intra-european assembly and exchange of citizen data. NR Note IMEDIA/01/98, Norwegian Computing Center (NR), 1998. Submitted to *The 5th ISPE International Conference on Concurrent Engineering*, Tokyo, Japan, July 15-17 1998.

[8] S. E. Hudson and I. Smith. Supporting dynamic downloadable appearances in an extensible user interface toolkit. In *UIST'97: Proceedings of the ACM Symposium on User Interface Software and Technology*, pages 159–168, 1997.

[9] G. Kiczales, J. Lamping, C. V. Lopes, C. Maeda, A. Mendhekar, and G. Murphy. Open implementation design guidlines. In *ICSE'97: Proceedings of the 1997 International Conference on Software Engineering*, pages 481–490, 1997.

[10] P. Maes. Intelligent interfaces. In J. Moore, E. Edmonds, and A. Puerta, editors, *IUI'97: 1997 International Conference on Intelligent User Interfaces*, pages 41–46, 1997.

[11] D. L. Martin, A. J. Cheyer, and G.-L. Lee. Agent development tools for the Open Agent Architecture. In *Proceedings of the First International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology*, pages 387–404. London, The Practical Application Company Ltd., April 1996.

[12] D. L. Martin, H. Oohama, D. Moran, and A. Cheyer. Information brokering in an agent architecture. In *Proccedings of the Seconds International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology*. London, The Practical Application Company Ltd., April 1997.

[13] D. B. Moran, A. J. Cheyer, L. E. Julia, D. L. Martin, and S. Park. Multimodal user interfaces in the open agent architecture. In J. Moore, E. Edmonds, and A. Puerta, editors, *IUI'97: 1997 International Conference on Intelligent User Interfaces*, pages 61–68, 1997.

[14] J. K. Ousterhout. *Tcl and the Tk Toolkit*. Addison-Wesley Publishing Company, New York, 1994.

[15] B. Schneiderman. *Designing the User Interface: Strategies for effective human-computer-interaction*. Addison Wesley Longman, 3rd edition, 1998.

[16] A. Scmitt. *Dialogsysteme*. Bibliographisches Institut, Mannheim, 1983. In German.

[17] Y. Shoham. Agent-oriented programming. *Aritificial Intelligence*, 60:51–92, 1993.

[18] A. F. Smeaton and F. Crimmins. Using a data fusion agent for searching the www. School of Computer Applications, Dublin City University. Glasnevin, Dublin 9, IRELAND.

[19] SPACE. Single point of access for citizens in europe, telematics applications programme, project ad 1014, technical annex (annex 1).

[20] L. A. Suchman. *Plans and Situated Actions*. Press Syndicate of the University of Cambridge, 1987.