mPower: A Component-based Development Framework for Multi-agent Systems to Support Business Processes

H Lee, P Mihailescu, and J. W. Shepherdson

One of the obstacles preventing the widespread adoption of multi-agent systems in industry is the difficulty of implementing heterogeneous interactions among participating agents via asynchronous messages. This difficulty arises from the need to understand how to combine elements of various content languages, ontologies, and interaction protocols in order to construct meaningful and appropriate messages. In this paper mPower, a component-based layered framework for easing the development of multi-agent systems, is described, and the facility for customising the components for reuse in similar domains is explained. The framework builds on the JADE-LEAP platform, which provides a homogeneous layer over diverse operating systems and hardware devices, and allows ubiquitous deployment of applications built on multi-agent systems both in wired and wireless environments. The use of the framework to develop mPower^{mobile} , a multi-agent system to support mobile workforces, is reported.

1 Introduction

Multi-agent system technology has been used on many occasions to automate business processes [][][][]. In such cases, a business process is frequently viewed as a collection of autonomous problem solving entities that negotiate with one another and come to a mutually acceptable agreement detailing how to co-ordinate their independent sub-activities. Multi-agent system technology is preferred as it is deemed to provide greater immunity against changes in business process definition compared with other computing technologies [6].

Not withstanding these advantages, the development of multi-agent systems is considered difficult because of its reliance on message-based communication. The creation and interpretation of a message requires an understanding of agent communication languages and their associated ontologies, content languages and interaction protocols [], which can be difficult for novice agent programmers to grasp. Furthermore, due to a reliance on asynchronous communication, the management of conversations among participating agents can be a burden for developers.

This paper describes a component-based framework that is intended to ease the development of multi-agent systems when automating business processes. This framework utilises reusable conversational components (C-COMs) that provide services for the execution of business tasks via interaction with other agent roles (such as 'Initiator' or 'Respondent' which are described in section 3.2). These C-COMs hide all the message composition and interpretation details from developers and manage the interaction states between collaborating agents. This framework also provides a set of generic workflows that consists of one or more C-COMs, which can be used as templates to automate domain- or organisation-specific business processes. The generic workflows can be used as an architectural pattern [], which is applied to business processes that have different requirements by replacing (or customising) one or more of their components. The framework is based on JADE-LEAP [] and is known as 'mPower'. This paper consists of five sections. The next section briefly reviews related work, whilst section 3 describes the mPower framework which shows the relationship between components, architecture, and applications. Section 4 illustrates how a multi-agent system (mPower^{mobile}) to support mobile workforces, was derived from the mPower framework. Finally, section 5 summarises this paper.

2 Literature review

Multi-agent systems are used as a core technology in various applications, ranging from information retrieval [] to business process automation []. Many multi-agent system platforms are based on Java and must be run on 'heavyweight' (e.g. desktop or server) devices using Java 2 Standard Edition (J2SE) - examples include the Comtec Agent Platform [] and Zeus []. This paper favours JADE-

LEAP [][] as a multi-agent system implementation platform as it enables the key components of the system to run on a wide range of computing devices. Therefore a mobile worker can use a highly portable device (such as a PDA or mobile phone) to access business process automation applications, in preference to a luggable laptop computer when working 'up poles and down holes' or on a Customer's premises.

Agent technology has long been used to support business processes. Huhns and Singh [] summarise the state of the art in agent-based workflows. Shepherdson et al. [] use a multi-agent system for the co-ordination of cross-organisational workflows. Jennings et al. [] insist that a multi-agent system has the necessary features for the support of modern dynamic business processes and propose a suitable multi-agent system architecture.

Multi-agent system reuse has been studied in some detail. Kendal et al. [] applied object oriented design patterns to implement agent concurrency, collaboration, and reasoning. They put forward an agent pattern or architecture which can be used for the development of multi-agent systems in similar domains. On the other hand, Brazier et al. [] propose a generic co-operative agent model that can be refined to generate application-specific multi-agent systems.

The LEAP project introduced the concept of a generic service component (GSC) [], which is a reusable software component that provides a service through message exchange with sub-components that implement one or more agent roles. The C-COMs described within this paper are an extension of the LEAP GSC concept.

A component-based approach to supporting business processes has already been adopted by some commercial companies. IBM's SanFrancisco [] is a framework that provides reusable components such as business objects, functions, and core workflows. SAP [] also provides reusable business components from which a business application can be easily customised. The mPower framework has a similar layered architecture to SanFrancisco. However, the components used in mPower have a different structure compared to those in SanFrancisco and SAP because they abstract and implement the business conversations among process actors, rather than business objects or functions.

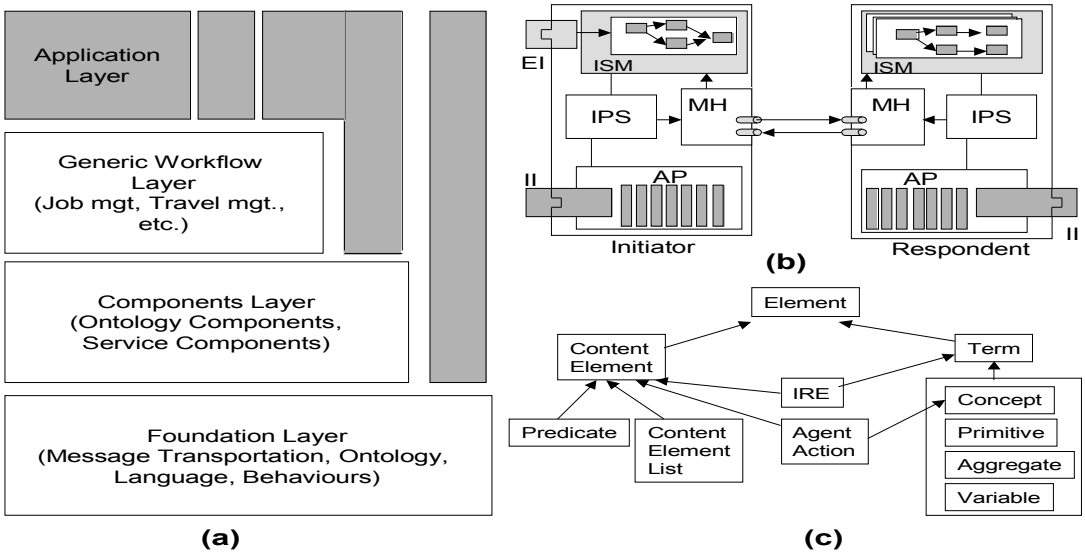

Fig 1 (a) layered architecture of the mPower framework, (b) structure of a conversational component, (c) hierarchy of ontology element in Jade [].

3 mPower: A reusable framework for the development of multi-agent systems

The basic principle of mPower for supporting a business process is to view the latter as a linked set of conversations among participating process actor roles. From this point of view, the application reuse means the reuse of conversations occurring in the target application. Hence the rationale of using a MAS as a key technology to support business processes.

Fig. 1(a) shows a layered view of the mPower framework, which is used to develop component-based multi-agent systems. This framework consists of four layers: foundation, components, generic workflow and applications. The foundation layer contains all the supporting functionality for a multi-agent system, such as message transportation, ontology support, language support etc.

The components layer consists of basic ontology and C-COMs that are common across a number of mobile workforce applications. The ontology components are reusable ontology items such as Customer, Job, and Shift etc. Each C-COM provides a

standard mechanism for accessing a service such as work assignment, route planning and attendance management.

The ontology components are used by C-COMs to standardise and understand the contents of service request and service response messages. The generic workflow layer is a set of pre-composed components (both of type ontology and service) that support generic business processes. At the application layer, a system is a customised collection of components from the layers beneath it.

3.1 Foundation layer

The mPower framework has been implemented using JADE-LEAP, which provides the foundation services, thereby reducing the effort required to develop multi-agent systems. JADE-LEAP provides the following benefits: First, it complies with the FIPA Abstract Architecture Specification; Second, it provides agent management services such as agent registration/deregistration and support for agent lifecycle management; Third, application developers are able to extend a generic agent provided by the JADE-LEAP platform and customise it to meet the specific requirements of a given application. The generic agent is equipped with a behaviour scheduler which controls the goal achieving behaviour of the agent; Finally, JADE-LEAP provides support for the use of FIPA agent communication languages used during inter-agent communication, as it provides ontology support, allows the use of content languages (FIPA SL and LEAP) and comes complete with a number of FIPA-compliant interaction protocols. With this support, developers are more easily able to create messages that are exchanged asynchronously among agents.

3.2 Components layer

The components layer consists of two types of components (ontology and conversational) which are based on the foundation services. The implementation of the ontology components - an abstraction of the JADE-LEAP common ontology items - is based on the underlying ontology support schema. The hierarchy of the ontology items supported by JADE-LEAP is shown in Fig 2 (c). The ontology components map the common ontology items into the hierarchy's predefined categories and detail the attributes of the items in target domains, whereas C-COMs abstract and implement the common message-based interactions among participating agents in target domains. The content of a message refers to the ontology components in order to represent the intention of the message sender. From an application developers' point of view, a C-COM is a black box that hides the details of the creation and interpretation of a set of messages that need to be exchanged by agents in order to achieve a service goal.

The two main building blocks of a C-COM are an interaction protocol and the role components. The interaction protocol defines the sequence of asynchronous messages sent between the role components, and the role components perform the actions necessary at each stage of the interaction protocol to achieve the service goal. The role components are installed into, and executed by, one or more agents. Fig. 1(b) shows the internal structure of a C-COM. There are two generic role components for each C-COM - Initiator and Respondent. The Initiator component starts an interaction by sending a message and the Respondent component is activated when it receives a message from an Initiator component. These two generic role components can be specialised according to the requirements of a given C-COM. Each role component consists of an Interaction Protocol Scheduler (labelled 'IPS' in Fig. 1), a Message Handler (MH), an Action Pool (AP) and one or more Interfaces. Each role component is in effect a Finite State Machine, driven by internal state changes, and has a different set of internal states according to the role the component plays in the interaction protocol employed for a given C-COM. The Interaction Protocol Scheduler schedules and executes all the actions stored in the Action Pool of a role component according to internal state changes. For this purpose, each role component maintains an Interaction State, which is managed by the Interaction State Manager (ISM). The Message Handler is responsible for validating outgoing messages and interpreting incoming messages. A role component provides a number of interfaces (i.e. sets of method signatures) for customisation purposes. An Initiator role component has two kinds of interfaces: External and Internal (EI and II respectively). An External Interface (which has a single method, named 'execute') defines the input data and the service result which is returned to the service consumer. An Initiator role component contains the implementation of the External Interface. The External Interface is a trigger for the entire C-COM. Calling the execute method in the External Interface activates the Initiator role component which then activates all its other Respondent role components in order. An Internal Interface is called by the role component itself, and an agent (which installs the role component) provides the implementation of that interface. For example, if a Respondent needs access to a knowledge source to retrieve information to populate a response message, the developer should provide the Respondent component with an implementation of an interface when s/he installs the Respondent component in an agent. Then the Respondent component interacts with the application-specific interface implementation to retrieve the required information. From this, applications supporting different mobile business processes can customise the same C-COM by providing different implementations of the interface, which reflect application specific contexts such as different

knowledge sources, business rules, and legacy system APIs etc.

The implementation of C-COM was based on the interaction protocol support within JADE-LEAP, as the latter provides useful components that can be extended to implement application-specific interaction protocols: namely Achieve Rational Effect Initiator/Respondent and Contract Net Initiator/Respondent. These components have been extended by specialising the actions executed at each stage of the interaction protocol (via changes to agent behaviour and ontology component selection) for each target business process.

3.3 Generic workflow layer

A generic workflow is a set of linked C-COMs, which can be reused to support similar business processes in the same domain. shows an example of generic workflow components for job management. Each rectangle represents a C-COM and double arrowhead represents the control transition between C-COMs. The first conversation is between the roles Job Distributor and Job Owner. Then, the Job Owner has two options to start the next conversation, that is, Job Trade or Job Update. The Job Owner role assumes a Job Giver role in the JobTrade conversation and a Job Executor role in the JobUpdate conversation. The JobClose conversation can be reached only from the JobUpdate conversation. This control flow enables an agent to determine the next conversation that a human worker might want to execute.

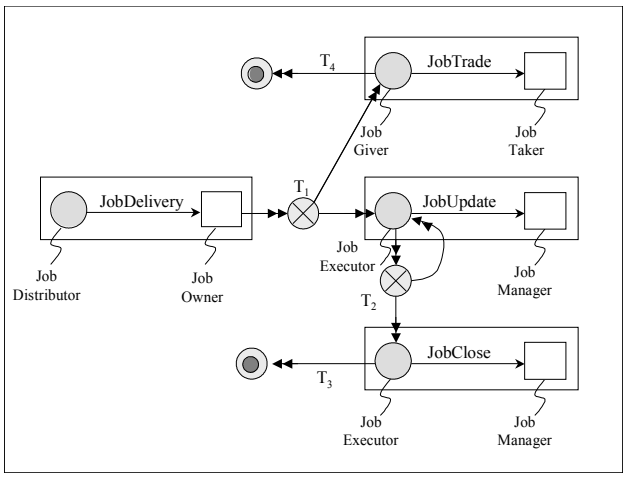

Fig 2 A generic workflow component for job management.

The following shows an example specification of the generic workflow shown in Fig 2.

<Workflow name=”job management cycle”>

<C-COM name=”JobDelivery”>

<Role name=”JobDistributor” type=”Initiator”/>

<Role name=”JobOwner” type=”Respondent” />

</C-COM>

<C-COM name=”JobUpdate”>

<Role name=”JobExecutor” type=”Initiator”/>

<Role name=”JobManager” type=”Respondent” />

</C-COM>

<C-COM name=”JobTrade”>

<Role name=”JobGiver” type=”Initiator”/>

<Role name=”JobTaker” type=”Respondent” />

</C-COM>

<C-COM name=”JobClose”>

<Role name=”JobExecutor” type=”Initiator”/>

<Role name=”JobManager” type=”Respondent” />

</C-COM>

<Transition id=”T_1” type=”XOR”>

<Resource id=”ontology.job_management.Job” />

<PreConversation name=”JobDelivery”

linker=”JobOwner”/>

<PostConversation name=”JobUpdate”

linker=”JobExecutor” />

<PostConversation name=”JobTrade”

linker=”JobGiver”/>

</Transition>

<Transition id=”T_2” type=”XOR”>

<Resource id=”ontology.job_management.Job” />

<PreConversation name=”JobUpdate”

linker=”JobOwner”/>

<PostConversation name=”JobUpdate”

linker=”JobExecutor” />

<PostConversation name=”JobTrade”

linker=”JobGiver”/>

</Transition>

…

</Workflow>

Fig 3 Generic workflow specification example.

From Fig 3, it can be seen that each C-COM is represented by a name, initiator role, and respondent role. A Transition tag specifies a transition from one conversation (specified by the PreConversation tag) to another (specified by the PostConversation tag). The selection of a conversation from multiple post-conversations is done by checking the relationship

between the pre-conversation and potential post-conversations. From the above specification, the transition "T_1" mandates that only one post-conversation can be performed. Also job information (accessible via the ontology.job_management.Job attribute in the ontology base of the agent) is transferred from the pre-conversation to the post-conversation. On the other hand, the transition "T_2" states that the JobUpdate conversation can be performed iteratively (as JobUpdate is one of the possible post-conversations) before it transits to the JobClose post-conversation.

Each workflow specification is shared by all the agents participating in that workflow, and is used to schedule the relevant C-COMs at run time. For example, if an agent receives a job assignment for its user as the result of the JobDelivery conversation, then it enables the GUI menu items that allow its user to launch the JobTrade and JobUpdate conversations, while disabling the menu item that launches the JobClose conversation.

Table 1: Identified services for mobile workforces.

Domain	Services	Description
Teamwork Coordination	Schedule Work Request	Given a pool of work-requests, enable a mobile worker to add a work-request to his/her current schedule. The pool of work-requests that a mobile worker sees may not be all of those currently available. Only those work requests that a particular mobile worker is can perform will be shown to him/her (this can be due to constraints imposed by the current schedule, by the mobile worker's experience and qualifications, and so on.)
	GenerateWork Schedule	Given a set of work requests, find a work schedule in which all of the constraints in the work requests (times, distances, etc) are satisfied and find routes
	TradeWork Request	Enables mobile workers to swap work-requests from their current schedules.
	Coordinate SocialActivity	Enables mobile workers to arrange social meetings such as a lunch during the working day. This may provide facilities for suggesting possible locations for lunch, determining who can attend lunch at some location (given constraints of time and distance), finding routes to locations and so on.
	SwapShift	Each Mobile Worker has an attendance pattern that defines the shifts they will work. A MW wants to swap a shift on some day for some other shift (on possibly the same day).
	Trade Overtime	A Mobile Worker has registered for overtime that they are no longer able to complete. The deadline for cancelling overtime has past, so the MW wants to find another MW willing to do the overtime.
	RequestLeave Change	A Mobile Worker wants to book leave for some date but is declined due to colleagues having leave booked for that date. The Mobile Worker can issue a request for colleagues to change the dates of their leave.
	CallFor Overtime Registrations	When a lot of unforeseen and urgent work arises, a Manager can request that Mobile Workers register for Overtime. This may be further refined to allow the Manager to target Mobile Workers with specific skills.
	CallToCancel Leave Bookings	When a lot of unforeseen and urgent work arises, a Manager can request that Mobile Workers forego LeaveBookings. This may be further refined to allow the Manager to target Mobile Workers with specific skills.
	Request Expertise	When a mobile worker has a problem that they cannot solve alone, this service will enable them to ask for help with the problem from an expert in the given problem area.
	Communicate WithMentor	As an inexperienced employee will often benefit from a mentoring relationship with a more experienced colleague, this service component enables mobile workers to communicate with a mentor.
	MakeCollecti-veDecision	Called by other agent service components in order to mediate the interactions between mobile workers when a collective decision is necessary.
Travel Management	PlanRoute	Given two locations A and B, calculate a route between A and B, subject to any given constraints (*e.g.* shortest distance, least time taken, must pass through intermediate 'waypoints' etc.)
	RePlanRoute	Following the initial generation of a route plan, the system identifies that the mobile worker is no longer on schedule. This may be due to a number of reasons: the work schedule being changed, new traffic information being received, the mobile worker being delayed, and so on.
	EstimateRoute Cost	Given a route consisting of a set of legs and using information about current conditions, calculate the cost of the route in terms of nominated dimensions such as time, mileage, etc.
Knowledge Management	Decompose Job	Given a job request, identify one or more work-requests that need to be issued and performed in order for the job to be completed.
	FindRelevant Information	Called by other agent service components in order to proactively provide mobile workers with information relevant to the performance of their work.
	Update Knowledge Base	Enable a mobile worker in the field to add knowledge to the knowledge base. Types of knowledge identified so far include feedback from the customer, work reports, technical experience and information about the customer.
	FindExpert	Given a problem, use the knowledge base to identify a colleague who is likely to be able to help in the given problem domain

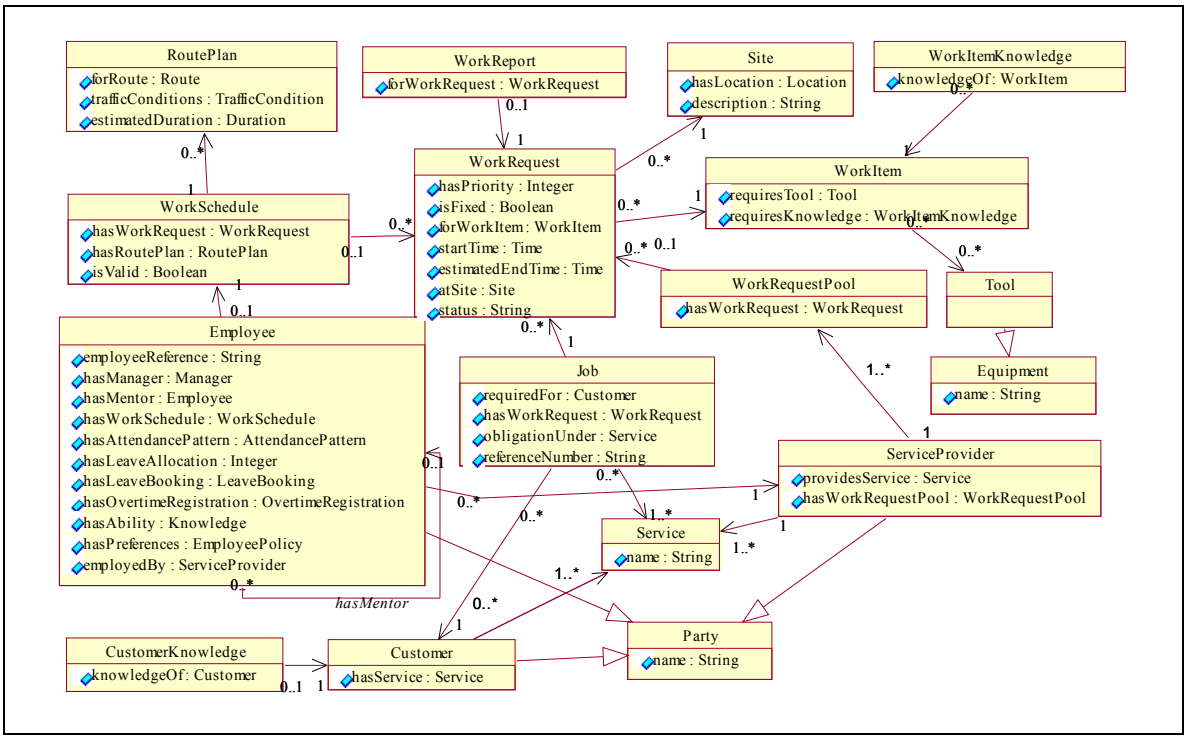

Fig 4 Some of the ontology components used to support mobile workforces.

4. mPower[mobile] : Customising mPower for mobile workforces

The mPower customisation process, to derive an application specific multi-agent system, consists of four steps: Identification of Services, Identification and Customisation of Components, Agent Identification, and Component Distribution. This sub-section details the process by illustrating how the generic components (C-COM and ontology components) and workflows were developed for mPower[mobile], a multi-agent system to support mobile workforces.

4.1 Identification of services

The first step was to identify the services required in the target application. Consideration of the nature and activities of mobile workforces pointed to four important service groupings.

- **Teamwork co-ordination** - empowering individuals to collectively co-ordinate activities (*e.g.* by trading jobs, automatically negotiating for work, and expressing personal preferences) within an agreed policy framework; facilitating 'buddying' between mobile workers where team members can exchange tacit knowledge, for example between experienced and trainee workers.
- **Travel management** - providing up-to-date information and guidance on travel planning. Ensuring travel time is minimised, thus saving resource and reducing traffic congestion. The Travel Management service anticipates a mobile worker's travel needs, providing guidance and time estimation so as to synchronise the movements of virtual teams working over vast geographic areas.
- **Knowledge management** - anticipating a mobile worker's knowledge requirements by accessing and customising knowledge (based on the mobile worker's skill, location, current job and type of display) and providing access to collective knowledge assets in the team (*e.g.* by putting novices in touch with experts, as and when required).
- **Job Management** – providing support for delivering jobs to assigned workers on the fly, updating job progress status, and closing assigned jobs with complete job closure data.

On closer inspection, each of the Job Management services turned out to be similar to services in one of the other three groupings, and as such could be developed by simply customising other services. Table 1 details the services from the three remaining groupings.

4.2. Identification and customisation of components

The next stage was to identify mPower components to implement the services identified in the first stage. As the services identified in the previous stage are generic, they were implemented using functionality from the Foundation layer and added to the components layer of mPower. First, the ontology components that the necessary C-COMs rely on were identified and implemented using the ontology support provided by JADE-LEAP. Fig 3 shows an example of the implemented ontology components. Second, based on the ontology components, the necessary C-COMs were implemented to produce the services identified in the previous stage. Third, job management related C-COMs were identified and customised from existing C-COMs (JobDelivery from AchieveReInitiator/Respondent in the Foundation layer, JobUpdate and JobClose from UpdateKnowledgeBase C-COM in the Components Layer, and JobTrade from TradeWorkRequest in the Components Layer). These job management related C-COMs were linked to form a generic workflow, as shown in Fig 2.

4.3 Agent identification

Having identified the reusable components, the agents were designed to take on the roles involved in those components. Usually, an agent takes more than one role, which means it is involved in multiple conversations. Furthermore, it is possible for an agent to take on all the roles in a given conversation. To support mobile workforces, four types of agent were designed. First, a *Personal Agent* which plays a personal assistant role to support a mobile worker for the execution of their assigned tasks. The support includes receipt of assigned tasks from other agents, update of job status according to progress, delivery of relevant information from knowledge sources, and coordination with other personal agents to reassign jobs, organise group meetings, swap shifts, swap annual leave, and so on. Second, a *Workflow (WF) Agent* which is responsible for interacting with a legacy Workflow Management System (WFMS) via a predefined API. It retrieves all the tasks assigned to a mobile team or a team member. The retrieved tasks are stored in a local database that is managed by the WF Agent, and notification sent to the Personal Agent of the worker that the job is assigned to, either on occurrence (push) or on demand (pull), as required. Task status is updated via the interaction between a Personal Agent and the WF Agent. Third, a *Library Agent* is an administrative agent which should be present in every application as it is responsible for the management of a library that contains the C-COMs used for conversations between the application agents. As all communication between participating agents is performed via C-COMs, modifying the conversation mechanisms used by the agents is achieved by updating the C-COM library. Then, the participating agents update corresponding C-COMs by version checking. Finally, an *Information Agent* collects information from various information sources, such as Web services, Corporate knowledge management systems, and Intranet directory services etc. As each knowledge source potentially uses a different interaction protocol to provide information to its client, the Information Agent must register a C-COM with the Library Agent for the Personal Agent to install and execute, in order to interact with it.

4.4. Component distribution

The last task is to install the identified components in the various agents according to the roles played by the agents in each conversation. This task is fairly straightforward, however the developer should ensure that the linkage between any two components corresponds to their respective interface definitions.

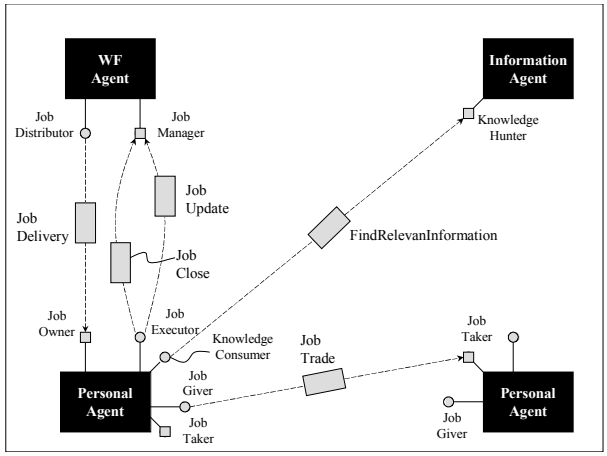

Fig 5 Components distribution diagram.

Fig 5 shows an example diagram for component distribution among identified agents. A component distribution diagram shows a structural view of conversations among participating agents in a target application in terms of C-COM. Each black box represents an agent, and each agent is annotated with its role components in its participating conversations. An initiator role component is represented by a small circle and a respondent role component by a small grey rectangle. A conversation between roles is represented as a dotted arrow with a rectangle attached in the middle. From Fig 5, it can be seen that the Personal Agent has three initiator components, namely JobExecutor, KnowledgeConsumer, and JobGiver, and two respondent components, namely JobOwner and JobTaker.

4.5 Personal agent architecture for the management of C-COM and generic workflows

The Personal Agent is comprised of four individual modules, each of which supports a specific functional area:

User Manager is responsible for managing a user's preferences by monitoring their interaction with the user interface. Through observing a user's interaction behaviour over a period of time, the User Manager is better able to tailor the application's functionality to meet the needs of the user. For example, if the User Manager observes that the user seldom views the routing information for a job, it

may decide to only download this information on demand and not when the job details are first downloaded.

Coordination Manager is responsible for fulfilling a service request by selecting a goal plan that meets the requirements of the requested service from a list of available goal plans. Each goal plan contains details of the tasks involved and their execution sequence. Typically a task will execute one or more C-COMs, or access a resource from the Resource Manager or interact with the user during its execution. The Coordination Manager is able to execute multiple goal plans concurrently, and is able to dynamically install new goal plans.

Resource Manager is responsible for managing all the resources required to support the execution of the Personal Agent, and application specific components. Resources can be classified into one of three types: 1) **Information objects**, 2) **Executable objects**, and 3) **External objects**. Information objects represent a piece of information, such as a list of user jobs, or a list of team members. Executable objects are C-COMs which are used during the completion of a service request. External objects are third party programs such as Microsoft Pocket Word™ which can be utilised to enhance the functionality of an existing service.

User Interface Manager is responsible for managing the flow of information between the user and the Personal Agent without restricting a user's freedom. A non-blocking approach is employed which does not force a user to wait for a service to complete before they can interact with the user interface. Instead, a user is able to launch multiple service requests from one part of the user interface and still be able to interact with another part of the user interface.

The four modules are able to directly interact with one another by passing events. Currently there are three recognised event types:

1. **User interface event**: This event is used to request a change in the current state of the user interface. For example, a goal plan may request a screen transition to show the results from a completed service.
2. **Goal event**: This event is used to request the execution of a service, and to report the status of an executing service. For example, a user may request a job trade service via the user interface to trade a job with other team members.
3. **Resource event**: This event is used to request access to a resource. For example, a task within a goal plan may request access to an installed C-COM in order to complete its execution.

External components such as an application user interface screen, or an external program are not permitted to directly interact with any of the four Personal Agent modules. Instead all events generated from external components are captured centrally by the Personal Agent which may perform any event filtering before dispatching the event to the appropriate module, as shown in Fig. 5.

Each event type contains the following five properties:

1. **Sender ID**: This identifies where the event originated.
2. **Type**: This identifies the event type.
3. **Action**: This identifies the type of action requested, which is dependent on the event type. For example, a goal event requesting the execution of a service will contain the 'achieve goal' value within its action property, whereas a user interface event requesting a screen transition will contain 'transition' within its action property.
4. **Action arguments**: This is an optional property which may contain multiple arguments, that are dependent on the type of action. For example, a user interface event with an action property set to 'change cursor', may contain the 'wait cursor' value within its action arguments properties.

User arguments: This is an optional property which may contain multiple user-defined arguments that are dependent on the event type and action. For example, a goal event with the 'achieve goal' value within its action property may contain some input values for the goal plan that will be selected to fulfil this service request.

Table. 2 provides an example of some of the pre-defined actions available for the three recognised event types. The Personal Agent supports both asynchronous and synchronous event delivery mode.

Event type	Action
User interface	Change cursor, transition, screen action
Goal	Achieve goal, goal success, goal failure
Resource	Put resource, get resource, delete resource

Table 2 List of pre-defined actions for event types.

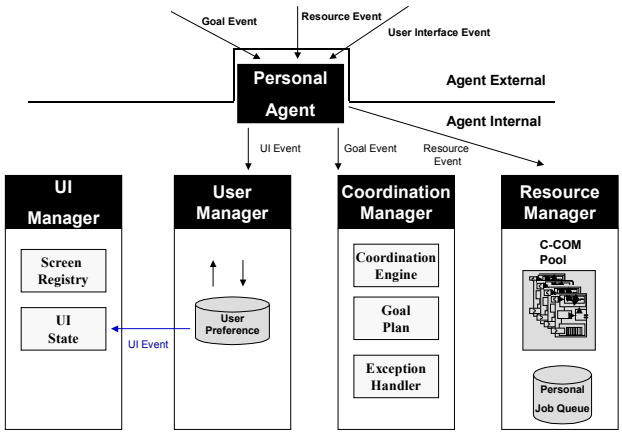

Fig 5 Personal Agent event dispatching model.

An example of the flow of events that occur within the Personal Agent architecture during a sample service request

will now be presented. The simulated service is called 'deliver jobs', and retrieves all jobs that have been assigned to a user. The flow of events is shown in Fig. 6, and discussed below:

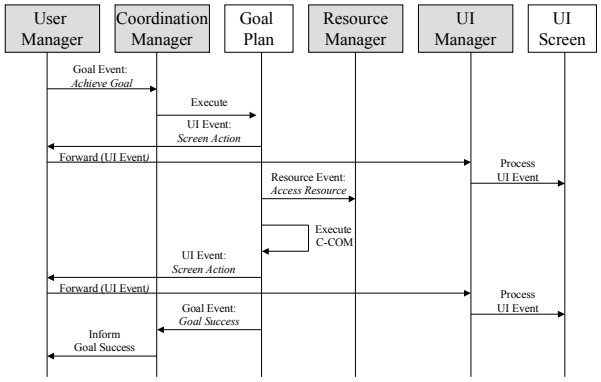

Fig 6 Sample service request event interaction scenario.

1. The service request is initiated by the User Manager which sends a goal event direct to the Coordinator Manager. The goal event contains the following properties: (**Sender**: User Manager, **Type**: Goal Event, **Action**: Achieve goal, **Action arguments**: Retrieve jobs, **User arguments**: Blank).
2. Upon receiving the goal event the Coordination Manager selects the most appropriate goal plan to complete this service request and executes it.
3. During the goal plan's execution it sends a user interface event to the User Manager, requesting that a progress bar is displayed in order to provide visual feedback to the user on the progress of the service. The user interface event contains the following properties: (**Sender**: Goal Plan ID, **Type**: User interface event, **Action**: Screen action, **Action arguments**: Job Queue Screen, **User arguments**: Show progress bar). The user interface event is then forwarded to the User Interface Manager which will hand the event to the user interface screen for processing.
4. The goal plan then dispatches a resource event to the Resource Manager to retrieve an executable object. The resource event contains the following properties: (**Sender**: Goal Plan ID, **Type**: Resource event, **Action**: Get resource, **Action arguments**: C-COM Retrieve Jobs, **User arguments**: blank). Once the C-COM is obtained it will be executed.
5. When the goal plan has fulfilled the service request it dispatches a user interface event to the User Manager requesting that the visual progress bar is removed. The user interface event contains the following properties: (**Sender**: Goal Plan ID, **Type**: User interface event, **Action**: Screen action, **Action arguments**: Job Queue Screen, **User arguments**: Remove progress bar). The user interface event is then forwarded to the User Interface Manager which will hand the event to the user interface screen for processing.
6. Finally the goal plan sends a goal event to the Coordination Manager informing it that it has successfully completed the requested service. The properties of the goal event are: (**Sender**: Goal Plan ID, **type**: Goal event, **Action**: Goal success, **Action arguments**: User Manager, **User arguments**: Service result). The Coordination Manager may then choose to release any resources which the goal plan may still have open before forwarding the goal event to the User Manager.

5. Conclusion

One of the critical success factors for the widespread adoption of multi-agent system technology in industry is to provide application developers with supporting tools that reduce the burden of building multi-agent systems. The mPower framework described in this paper aims to enable application developers to assemble a multi-agent system by customising pre-built components according to application specific requirements.

The framework provides three layers of components. The foundation layer provides the basic functional components, via the JADE-LEAP platform. The components layer provides ontology components and C-COMs that abstract and implement the frequently used interactions among participating roles for each business domain. The generic workflow layer provides workflow components that consist of two or more C-COMs to achieve a business objective. A multi-agent system-based application can be derived by reusing the components in each layer (or by mixing components in different layers).

This paper has shown how the mPower^{mobile} application was derived from the mPower framework to support mobile workforces. Finally, a Personal Agent architecture has been proposed to explain how the components of mPower can be installed and used by an agent to provide services to mobile workforces.

Acknowledgements

The work described here was funded by BT Exact's OSS and Customer Satisfaction Venture – many thanks to Paul O'Brien for his advice and guidance.

References

1. FIPA: The Foundation for Intelligent Physical Agents. http://www.fipa.org/ (2002)
2. Berger, M., Buckland, B., Bouzid, M., Lee, H., Lhuillier, N., Olpp, D., Picault, J., and Shepherdson, J.W.: An Approach to Agent-based Service Composition and it's Application to

Mobile Business Processes. IEEE Transactions on Mobile Computing, 2 (3), 2003.

3. Bose, R.: Intelligent Agent Framework for development knowledge-based decision support system for collaborative organisational processes. Expert Systems with Applications, 11 (3), (1996) 247-261

4. Brazier, F.M.T., Cornelissen, F., Jonker, C.M., and, Treur, J.: Compositional Specification of a Reusable Co-operative Agent Model. International Journal of Cooperative Information Systems, 9, (2000) 171-207

5. Buschmann, F. et. al.: A System of Patterns – Pattern Oriented Software Architecture. Wiley, ISBN 0-471-95869-71 (1996)

6. Caire, Giovanni: JADE Tutorial –Application defined content languages and ontologies. http://jade.cselt.it/ (2002)

7. Chang, J.W. and Scott, C.T.: Agent-based workflow: TRP support environment (TSE). Computer Networks and ISDN Systems. 28, (1996) 1501-1511

8. Klusch, M.: Intelligent Information Agents : Agent-based Information Discovery and Management on the Internet. Springer, Berlin-Heidelberg-New York, (1999)

9. Jennings, N. R., Norman, T. J., Faratin, P., O'Brien P., and Odgers, B.: Autonomous agents for business process management. Int. Journal of Applied AI 14(2), (2000) 145—189

10. Comtec Agent Platform. http://ias.comtec.co.jp/ap/

11. Collis, J., Ndumu, D., Nwana, H., and Lee, L.: The Zeus Agent Building Tool-Kit. BT Technology Journal, 16(3), July (1998) 60-68

12. LEAP: Lightweight Extensible Agent Platform. http://leap.crm-paris.com, (2002)

13. Huhns, M.N. and Singh, M.P.: Workflow Agents. IEEE Internet Computing, 2 (4), (1998) 94-96

14. Shepherdson J.W., Thompson S.G. and Odgers B.: Cross Organisational Workflow Co-ordinated by Software Agents. WACC '99 (Work Activity Co-ordination and Collaboration) Workshop Paper, February (1999)

15. Caire, G., Lhuillier, N. and Rimassa G.: A communication protocol for agents on handheld devices. In Proceedings of Workshop on ubiquitous agents on embedded, wearable, and mobile devices, held in conjunction with the 2002 Conf. On Autonomous Agents & Multiagent systems, Bologna, Italy, (2002)

16. Kendall, E., Malkoun, M.T., and Jiang, C.H.: Multiagent System Design Based on Object Oriented Patterns. The Report on Object Oriented Analysis and Design in conjunction with The Journal of Object Oriented Programming, June (1997)

17. Rubin, B.S., Christ, A.R., and Bohrer, K.A.: Java and IBM San Francisco Project, IBM Systems Journal, 37 (3), (1998)

18. SAP, http://www.sap.com/, 2003.