Towards Understanding and Evaluating Mobile Code Systems

Authors

Todd Papaioannou and John Edwards

Department of Manufacturing Engineering
Loughborough University
Loughborough
Leicestershire, LE11 3TU, UK
Tel.: +44 1509 228250
Fax.: +44 1509 267723

T.Papaioannou@lboro.ac.uk
J.Edwards@lboro.ac.uk

Abstract

Mobile code has been championed as a solution to a plethora of software problems. This paper describes investigative work undertaken in order to evaluate the mobile code abstractions of Mobile Agents and Mobile Objects, and to understand the implications of using these abstractions to build distributed systems.

We describe two systems built to support the Sales Order Process of a distributed manufacturing enterprise, using IBM's Aglets Software Development Kit. The Sales Order Process model and the requirements for agility used as the basis for these implementations are derived from data collected in an industrial case study.

Both systems are evaluated using the Goal/Question/Metric methodology. Two new metrics for Semantic Alignment and Change Capability are presented and used to evaluate each system with respect to the degree of system agility supported. The systems are evaluated through a set of scenarios generated during the case study in an attempt to see if they support system integration and agility in the manufacturing domain. Further we examine the implications of using a mobile code abstraction when compared with the abstraction offered by traditional distribution technology.

The work described provides evidence that both Mobile Agent and Mobile Object systems have inherent properties that can be used to build agile distributed systems.

Further, Mobile Agents with their additional autonomy provide marginally greater support.

Keywords Mobile Agents, Mobile Objects, Enterprise Integration, Distributed Systems, System Agility, Sales/Order Processing, Aglets.

1 Introduction

Access to an organisation's information is a critical factor in developing business systems. Regardless of the architecture employed, the information stored in an organisation's databases is its lifeblood. The ability to effectively manage, manipulate, and distribute this information was once viewed as a provider of competitive advantage (White 1994). Today it is simply a base requirement for corporate survival within the global market place.

Increasingly, the operations of a manufacturing enterprise are being distributed geographically, and thus the supporting information technology (IT) systems must themselves be capable of distributed operation (SSA 1995). Management of information remains a base requirement, as does the integration of differing IT systems and data sources.

Internet technology has proved to be effective for connecting a mix of different types of computers and computer networks, while also providing location independence to information. Further, analysts predict that the near future will see the evolutionary convergence of the Internet, Intranets and traditional IT models such as client/server and peer to peer (SSA 1995). However there remain a number of problems with this technology.

The saturation of network bandwidth, especially when part of the network in question is the Internet, means that remote database access as required by the distributed enterprise model could mean ineffective IT support for the business. As well as data timeliness, factors such as data integrity and security are also a concern when dealing with the Internet.

Mobile code technology has been proposed as a general solution that has the potential to overcome this set of problems. It achieves this through local interaction (Clements

et. al. 1997) and is equally applicable to the problem of geographically distributed information sources, since mobile agent systems are inherently distributed. Both mobile object and mobile agent technologies are used in our work to build and integrate software systems that support distributed manufacturing enterprises, in order to evaluate how well they support the needs of a real world business. Several scenarios generated during the case study are used to evaluate the agility of the systems. The evaluation of the two systems is conducted using Basilii's GQM methodology (Solingen and Berghout 1999), through which a number of relevant metrics are identified. The following section describes the model used in this work.

2 The Model

The systems described in this paper are implementations of a real-world Sales Order Process (SOP), typical of many manufacturing enterprises. The model upon which this system is based has been derived from an industrial case study undertaken at a high-pressure vacuum component manufacture that is based in the UK (from herein referred to as Acme).

Acme currently uses a mixture of bespoke packages and standard Office products to support its business. However, as the needs of the market change ever more rapidly the frailties in the existing IT infrastructure become increasingly apparent. To survive, Acme must remain agile and competitive, and to achieve this their IT infrastructure must be capable of responding to change. This requirement is not being met by the existing IT infrastructure. For example, in a recent experience at Acme it took two and half years to phase out an old accounting package and introduce the new software; clearly this is not acceptable. Equally, the strategy of buying inflexible, monolithic "off the shelf" packages is beginning to impact on the business, as these rapidly become inflexible legacy applications. The intricate spider's web of interdependencies woven between those packages that are already in place is making it increasingly difficult to contemplate radically changing the supporting infrastructure. As the interdependencies increase, the agility of the company is jeopardised. This legacy IT problem is now well recognised within many application domains.

The overview of Acme's business processes generated from this study can be seen in Figure 1. In this diagram, the separate business processes involved in the SOP are defined by the senior management figures that are responsible for those particular areas. Each core process is surrounded by a dotted line for further clarification.

From this view, it is possible to identify and extract the core business processes and represent them in a higher level abstract view. By examining the interactions between the major processes, a simple top-level process model was generated to represent the entire SOP within the Acme business. Figure 2 shows a representation of the top-level business processes model and a “walk through” description of its elements and their interactions.

3 Industrial Motivation

The case study at Acme has played an important role in providing the model and scenario for the implementations undertaken in this paper. Firstly, it confirmed certain business needs commonly found among manufacturing enterprises: namely, the ability to remain agile and competitive in an ever-changing market. These are lofty goals however, and difficult to examine and judge in detail. More specifically, the study revealed Acme's particular requirements which were identified as the need for an IT support system that:

- could handle the rapid addition of new sales agents
- could handle the addition of new stock control centres
- would allow changes to the business logic of the SOP to be made easily

These requirements are more specific, and will form the basis of the scenarios for change used to evaluate our implementations and the mobile code abstraction that they embody.

3.1 A Model For Sales Order Processing

The core processes identified at Acme that are involved in the SOP were first introduced in Figure 2. This model addresses all of Acme's core business processes. The implementation described in this work examined particular aspects of the overall model, concentrating on the interaction of a sales agent dealing with order requests and the stock control centres. The modified model is shown in Figure 3. Production

Control was excluded since this is an entire field of research in its own right and was deemed external to the objectives of this research. In addition, the greyed out areas in Figure 3, Dispatch and Manufacturing, represent processes that were not considered in our study, but would make excellent candidates for investigation and expansion in any future work.

Figure 4 depicts the implemented agent model for Acme's SOP. The fundamental operation is as follows: Following an enquiry from a customer to a SalesAgent, an OrderAgent is dispatched to the StockControlAgent, which is resident at a distribution point, where it requests the fulfilment of its order by passing over an Order object. The StockControlAgent queries the stock database to see if enough products are in stock. If there are enough products, the StockControlAgent then returns a DeliveryDate object to the OrderAgent which itself returns to its parent SalesAgent. The SalesAgent is then able to notify the customer of the delivery date.

If there are not enough products in stock to satisfy the order, the OrderAgent migrates to the manufacturing plant where it uses the Product ID encapsulated in the Order object and queries the BOM database for a list of sub-parts or raw materials required. This is then encapsulated within a WorksOrderAgent (again mobile) and dispatched to manufacturing while the OrderAgent returns with a DeliveryDate object containing a standard delivery date. If there are not enough raw materials in stock, agents within the manufacturing plant server generate a PurchaseOrderAgent that encapsulates details of all the required materials.

Our mobile object model is similar to that described above, the key difference being that the results from stock database queries are gathered from remote StockControlAgents by a mobile OrderObject guided by a specific itinery. Instead of processing this information locally to the data source, it is returned to the SalesAgent for processing. The mobile object does not make autonomous decisions based on the acquired information.

4 The Systems

The experimental work described in this paper was implemented using IBM's Aglet Software Development Kit, a mobile agent development framework (Lange and

Oshima 1998). The model, scenario and investigations described have been undertaken using both mobile agent and mobile object systems. The two systems consist of a selection of individual agents and objects. The types identified for the SOP model are SalesAgents, StockControlAgents, ManufacturingAgents, PurchasingAgents and DispatchAgents. There are also two analogous but different components in each system. In the mobile agent system there are OrderAgents, whilst in the mobile object system there are OrderObjects. These are discussed in the following sections.

4.1 Sales Agents

Sales Agents are static, Graphical User Interface (GUI) based agents that are responsible for generating OrderAgents or OrderObjects and dispatching them to distributed agent hosts around the world to interact with StockControlAgents. These can be resident as a very slim client for sales persons working on terminals or NetPCs, or they can be hosted on a laptop for travelling sales persons. They are capable of keeping track of current orders that have been placed. In the mobile agent version the only logic contained within these agents is that required to create a new OrderAgent, with its accompanying Order. They are capable of maintaining a list of spawned OrderAgents, and thus which Orders have been fulfilled, or not. In the mobile object version they also contain the business logic required to process the results returned by their slave OrderObjects.

4.2 Order Agents

The OrderAgents represent the mobile elements in this system. As classified in Section 4.1, they are goal oriented, communicative and mobile. Each OrderAgent encapsulates a single order; and they are responsible for completion of that order. After creation they migrate to a new host to interact with a StockControlAgent. If the stock levels are unable to satisfy the order, they are able to migrate to a new host, and use the encapsulated Product ID to derive the Bill of Materials (BOM). The agent then migrates to the ManufacturingAgent's host. In future the agent would then invoke this manufacturing process or at least establish the required time for manufacture. Currently, this communication consists of a simple message and acknowledgement from the ManufacturingAgent. The valid outcome for the goal of the OrderAgent is

reporting a delivery date for the order to the SalesAgent. In the future this may also include reporting a future time for delivery or an allocation for materials and an internal works order number and time to manufacture. OrderAgents require no interaction with a user and so have no GUI. Although quite simple, OrderAgent's make up the majority of the agent population in the system when it is executing dependant on the number of enquiries received by the SalesAgents. Potentially, there could be large numbers of mobile OrderAgents migrating through the network, attempting to fulfil their own particular order.

4.3 Order Objects

OrderObjects initially appear to perform the same function in the mobile object system, as the OrderAgents described above. However, in contrast to the mobile agent system, it is more appropriate to view the mobile objects as mobile messengers. They are still able to migrate to a data source and take advantage of local interaction and all the advantages this brings, but they do not contain the business logic to autonomously process any results. They collect the stock level information and return to their origin to report findings to their parent SalesAgent, after which they are terminated.

4.4 StockControlAgents and MaterialsStockAgents

The StockControlAgents are another example of static agents, with no GUI. They are responsible for handling all requests for products, parts, or materials, and are interfaced to the stock control databases. As such they act as a wrapper to the data source, a communications bridge between the data and the agents system. All requests for stock allocation must be made through the StockControlAgents.

When designing StockControl Agents that could unify the variety of database systems which could be expected within a manufacturing enterprise, it became apparent that some of the required features of these agents were particular to each database, whilst others were generic to all StockControl agents. In considering this problem the use of a common 'Database Query Agent' was conceived which could be used as a base pattern for all StockControl agents in the system. The advantage of such a technique is the consistency and reusability inherent in using a pattern from which to build agents

that are more complex. The Database Query Agent is discussed in (Papaioannou and Edwards 1999).

This architecture has been put to good use in the MaterialsStockAgents. They perform almost exactly the same role as the finished StockControlAgents, but are tasked with controlling the allocation of raw materials, and out-sourced parts. They are also connected to a Bill of Materials database, which the mobile OrderAgents and OrderObjects are able to query when having to request the manufacture of stock to fulfil an order.

4.5 ManufacturingAgents, PurchasingAgents, DispatchAgents

Currently these three types of agent are represented in the systems by static agents that are communicative. They are able to simply acknowledge communication from other agents, and represent a definite avenue for further investigation and research.

5 Evaluating The Systems

The model clearly shows the interaction of a number of processes that are supported by specific sub systems, demonstrating the importance of system integration in this domain. In order to evaluate whether mobility can help manufacturing enterprise integration we must first understand the integration methods currently employed.

Traditional distribution mechanisms promote location transparency as one of their major advantages, i.e. they achieve communication between different objects by "hiding" the location of these objects from each other. Their messaging systems allow mobile data to be passed between objects and locations. Thus, traditional systems can be characterised as using location transparency and mobile data to integrate and enable communication between distributed objects (Papaioannou and Edwards 1999).

Mobile code systems are quite different however. We characterise mobile object systems as providing local interaction for communicating objects, and providing an abstraction based on mobile messengers with limited autonomy. We characterise mobile agent systems as providing local interaction for communication, through an abstraction based upon mobile logic *and* data. In addition, mobile agents provide greater autonomy.

The rational for the construction of the two systems was to evaluate both mobile code abstractions and the supporting technology, in an attempt to understand exactly what it is each abstraction has to offer, and how that might affect how we build distributed systems. To do this we must have some tangible method of measurement. The metrics we have generated, and the process undertaken is described in the next section.

5.1 Generating Useable Metrics

Evaluating software architectures is a notoriously hard task. There are very few established techniques or measurements for gathering data, and although software engineering as a discipline strives to emulate the classical sciences, we are still a long way off. Instead of formal equations, we have methodologies for developing metrics. They include: the Quality Function Deployment approach (Kogure 1983), the Software Quality Metrics approach (Boehm et al 1976) (McCall et al 1977) and the Goal Question Metric (GQM) approach (Basili et al 1994) (Solingen and Berghout 1999). Basili's GQM methodology was selected to evaluate the systems as it enjoys widespread popularity and support within the software engineering community.

5.1.1 The Goal

The GQM methodology is based upon the assumption that to gain a practical measure one must first understand and specify the goals of the software being measured, and the goals of the measuring process. More specifically, it is important to specify what is being evaluated, what task it should fulfill and from what perspective to view the measurements. Once this framework has been established, it is possible to direct investigation and measurement towards the data that defines the goals operationally. The generated framework is also useful when interpreting the data.

The overall goal of our evaluation can be stated as:

> "To evaluate the implemented systems from the industrialist perspective,
> with respect to satisfying the industrial motivation to support system agility"

5.1.2 The Questions

Having stated the goal, the process is continued by generating a broad set of questions that may provide some indication of the individual issues encapsulated by the main goal. The objective is to generate as many questions as possible, including redundant or invalid questions. As the process continues, it is usual to develop a hierarchical set of questions that can subsequently be narrowed. This refined set can then be answered through tangible measurements made on the system.

To this end two workshops were held, one within the R.E.D. group at Loughborough University, and one in the Computer Science Department of Reading University. To fulfil the three requirements of I.T.L. specified in Section 3 the initial questions focused on system complexity (how easy is it to understand), and system agility (how easy is it to change). The results of these workshops were a large and varied set of questions, with many superfluous or duplicate entries. Table 1 lists the focused set of questions that remained after refining.

5.1.3 The Metrics

After several iterations of refinement, and some healthy pruning, a set of useable software metrics remained that could be used to evaluate the two mobile code systems. These are shown in Table 2.

On their own, most of the generated metrics are extremely narrow in their focus. However, through combination, it is possible to arrive at some useful measures of a software system. In the following sections, we examine how these metrics can be used to evaluate the implemented systems, and whether the claims in this thesis have been supported or refuted.

5.2 Evaluating Semantic Alignment

It is the authors' belief that mobile code can increase the semantic alignment between software systems and the real world business processes they are intended to support. To evaluate this claim we require some way of measuring how well the abstractions of the real world are embodied in the software, and how well they resemble the real world model. For this, we have developed a term called Conceptual Diffusion.

5.2.1 Conceptual Diffusion

Conceptual Diffusion is defined as a measure of:

> "The degree to which a single concept or semantic abstraction in the application domain maps to the components in a software system."

Therefore, we may say that:

$$CD = \frac{A}{B}$$

Where *CD* is conceptual diffusion, A is the number of concepts included in this abstraction, and B is the number of components in which this abstraction is embodied.

Conceptual diffusion can be examined at different levels of granularity to gain different perspectives on a situation. For example, in a software system that is intended to support a Sales Order Process we expect the concept of an Order to be present. On analysis, we find that in both the agent and the object systems the concept of an Order is split over four separate components. Thus, in these two systems, the concept of an Order can be said to have a conceptual diffusion rating of four (see Table 3).

Table 3 also shows the results of metrics (1) and (2). These metrics are examples of examining conceptual diffusion at a larger level of granularity. For example, metric (1) requires the identification of all the information-based concepts within the real world, and a comparison with their counterparts in the software systems. Since Order is an information-based abstraction, it is therefore included in the results of metric (1). We may use Conceptual Diffusion to gain an insight into how well concepts or abstractions are embodied in software.

5.2.2 Semantic Alignment

Semantic alignment between the real world abstractions and the components of a software system has been shown to be important when attempting to build agile software systems (Coutts and Edwards 1998). Conceptual Diffusion in itself is a measure of how well a software system is semantically aligned with those business processes it is trying to support. As it stands however, the conceptual diffusion

measure remains relatively fine grained in its perspective. It does not offer an overall view of a system, rather an insight into a particular abstraction.

To gain an overall perspective of a system, a compound metric has been devised. It is a combination of metrics (1) to (4) and is termed the Semantic Alignment Metric:

$$SA = \left\{ \frac{Is}{Ir}, \frac{Ps}{Pr}, \frac{Ms}{Mr}, \frac{Ss}{Sr} \right\}$$

where SA is semantic alignment, *I* is information based abstractions, *P* is process based abstractions, *M* is mobile components, *S* is static components, *s* denotes in software and *r* denotes in the real world. Thus, $\frac{Ps}{\Pr}$ is the ratio of process-based abstractions in the software to the process based abstractions in the real world.

This metric can be used to analyse a system and to assess how well the software system reflects the semantics of the application domain. A comparison with the ideal alignment of {1,1,1,1} can be used as a measure to gauge how difficult it might be to understand the software, given an understanding of the application domain. Table 4 shows the results of metrics (3) and (4).

By combining the results of the first four metrics, we are able to state that:

> For the Mobile Object System Semantic Alignment = {4,22/6,1/3,2/3}
>
> For the Mobile Agent System Semantic Alignment = {4,21/6,1/3,2/3}

5.2.3 Summary

The results of the Conceptual Diffusion and Semantic Alignment analysis show that both Mobile Agent and Mobile Object systems should be easy to understand, as the abstractions in the real world align reasonably well with the components of the software systems. The information abstractions from the real world are on average spread over four components in the implementations. When considering mobile and static component alignment, for both systems, a third of the components in the domain

are modelled as mobile in the implementation, and two thirds of the static components in the domain are modelled as static elements in the implementations.

The difference in the two systems is shown when considering the semantic alignment of the business process. Here the mobile agent system is shown to have better semantic alignment than the mobile object system as the process logic for the SOP is contained *solely* within the OrderAgent and not diffused across both the SalesAgent and the OrderObject. Therefore, we can conclude that the mobile agent solution provides better semantic alignment with the real world business processes it supports.

If we examine traditional distributed systems we find they have no facility to construct mobile components in a system. Therefore, they would be unable to implement any of the mobile abstractions. Instead, these abstractions would have to be diffused over several static components. Since mobile code systems are equally adept at building static components, we can conclude that mobile code systems do indeed increase the semantic alignment between the real world and its supporting software systems, for any system that is not constructed from completely static components.

Importantly, these metrics are not merely restricted to use after the fact, but can be used proactively during the specification process, before any software has actually been built. Ensuring good semantic alignment of a software system before production will undoubtedly save both time and money in the long term. In particular, these metrics can be useful for identifying those components that should be mobile, and those that should be static. With increasing numbers of mobile code systems being built, this will prove an increasingly important aspect of system analysis and design.

5.3 Evaluating System Agility

In order to evaluate the agility of a system it is necessary to make changes to that system. The case study of I.T.L. highlighted several real-world industrial requirements for agility that a company may have for a distributed SOP system (see section 3 Using these requirements as scenarios for change, modifications to both the mobile agent and mobile object implementations were undertaken, in order to evaluate the agility of each system.

5.3.1 Change Capability

The GQM methodology enabled the derivation of several metrics that can be used to measure certain changes in a software system after modification. These measurements are specified by metrics (8), (9), (10) and (11). Individually, they enable us to measure narrow slices of change to a system. However, by combining these metrics it is possible to produce a more encompassing measure of agility. This set has been termed Change Capability, and is described by:

$$CC_{á \to â} = \left\{ \sum_{á}^{â} ädo, \sum_{á}^{â} äds, \sum_{á}^{â} äéi, \sum_{á}^{â} äåâ \right\}$$

where Change Capability CC, for a required change -, is the set of the changes to the number of objects (*o*), the number of src files (*s*), the number of interactions (*é*) and the number of conceptual entities (â), between states á and â. A conceptual entity is analogous to the abstraction or concept referred to in the previous sections. For example, it could be an Order, or a StockControlAgent. Interactions are those exchanges of information between objects, usually via method invocations, although for agents this also applies to any messaging dialogue they might enter. Changes to those interactions will usually imply changing a method signature.

Change Capability can be used to compare systems or to get a measure of the agility of the system relative to the ideal {0,0,0,0}. For the mobile object and mobile agent systems Change Capability for each requirement is summarised in Table 5.

5.3.2 Summary

Again, these results show that both systems are relatively easy to change. Adding new sales facilities requires only the instantiation of new SalesAgents that incurs zero changes to the system code. New stock control centres can be added through a low number of changes that are the same for both systems. The difference between the systems becomes apparent when making changes to the Sales Order Process logic. In the mobile agent system, this logic is contained *solely* in the single mobile OrderAgent, whereas in the mobile object system it is contained in both the SalesAgent and the OrderObject.

The Change Capability metric can be used by a system designer to evaluate how responsive to change their system has been after a specific change. It is possible to deduce areas that require refactoring, or are particularly troublesome when undertaking change. For example, consider the CC set {5, 20, 20, 1}. We see that for this change, although only one conceptual entity was changed, there were twenty changes to source files, five changes to objects, and twenty changes to the interactions of those objects. Changing the signature of twenty methods in five objects to enable a change in a single entity can cause serious problems and should lead the designer to review how diffuse this particular entity actually was. Of course, this is also revealed by the Conceptual Diffusion metric.

While both implementations have demonstrated they are relatively agile, the question of whether they are more agile than a traditional distributed system remains open. Certainly, it is unlikely that a traditional system will be any more agile than the mobile object system, since Remote Computation and Client/Server are very close in terms of the abstraction the offer. Nevertheless, the mobile agent system has shown that it is more agile than the mobile object system. This increased agility was due to the reduced conceptual diffusion and improved semantic alignment that the mobile agent abstraction allows. Therefore we may argue qualitatively that a mobile agent system would be more agile than a traditional system. For a definitive answer, further quantitative measures are required. However, in the next section we pursue this matter by examining loose coupling, a central issue to building agile software systems.

5.4 Evaluating Loose Coupling

Loose coupling may be defined as:

> "A measure of the external dependencies of a component defined by the number of links that component has to other components within a software system."

To build loosely coupled systems, components of that system should not be linked directly to form a complex network of interactions and inter-dependencies. Instead, they should remain distinct abstractions, embodying the concept of their real world equivalents. Components can then be assembled into a software system, with no prior knowledge of each other.

Traditional distributed systems such as CORBA do not enable loosely coupled systems inherently (Coutts and Edwards 1998). Components in these systems that wish to communicate require implicit knowledge of each other's interfaces. These interfaces are the central aspect of building distributed systems with traditional technology.

> "You should be able to look only at the IDL and know precisely how to implement against it." (Vinoski 1999)

Therefore, even if the key conceptual abstractions remain embodied in large grained components, for these components to interact they must be aware of each other *a priory*, and inevitably end up intermeshed with each other. The work of Coutts and Edwards has shown that it is possible to build loosely coupled systems with traditional technology by employing additional design patterns and forethought. Here we argue that this enforced route is simply increasing the cognitive complexity of building distributed systems, something that is already an onerous task.

5.4.1 Examining Coupling in Mobile Code Systems

We have already seen in the sections above that distributed systems built with mobile code are able to minimise conceptual diffusion. This enables an extremely good alignment between real world processes and their supporting software counterparts. On examination of the static software entities in our systems, for example SalesAgents, StockControlAgents, ManufacturingAgents, etc, we find that they are fully decoupled from each other. During execution of the system, there is no communication or interaction between any of the static components. Any communication that does take place within the systems is between static and mobile entities. Until a mobile entity alights at a host and attempts to interact with a static one, there is no coupling between any of the components. This is significant, since the system only experiences tighter coupling during a dialogue between components, i.e. when a mobile entity wishes to communicate with a static one. Of course, this dialogue depends upon prior knowledge on the part of the mobile entity as to what language the other agent understands, be it a syntactic dialect, or a more complex semantic conversation. In a private, controlled system however, this knowledge will always be available. In addition, since there are very few types of component that are mobile it is simple to alter the interactions, by updating the mobile agent population.

Research is being undertaken so a dialogue may be established with no foreknowledge. Although this is currently in the static, intelligent agents domain, it will naturally be applied to that of mobile agents.

5.4.2 Summary

The abstraction employed in traditional distributed systems does not support loose coupling inherently. Distributed systems built with this abstraction rely on component interface signatures for identification, and to facilitate communication. Coutts and Edwards have demonstrated that with further software architectures a certain degree of loose coupling can be achieved. Their use of the Mediator pattern has one drawback however – all components that wish to interact must do so via the Mediator. The strength of this approach is also its main weakness. By enforcing a policy of mediation, the distributed system is also subjected to centralised control, and thus the Mediator is a single point of failure. Building distributed software systems with a single point of failure has been shown to be bad.

In a traditional distributed system the concept of physical location is hidden. However, for two components to interact there must be some form of identification involved. This identification manifests itself through the interface types of the interacting components. Therefore, in reality the purpose of identification by interface is to enable the location of a component that can provide the required services. The core information in the task of locating a component is no longer physical location, rather it is the interface. Although the major tenet of this abstraction is location transparency, it is clear that the task of locating components remains. It has merely been replaced by an alternative method.

On the other hand, components in distributed systems built with the mobile code abstraction do not rely on interface signatures to be located. Instead they employ physical location as the information required for location. This is an important difference. By retaining location as the locator, the mobile code abstraction divorces the distribution mechanism from the dialogue constraints. This is shown in Table 6.

This separation has important implications for how tightly coupled a system might be. By divorcing distribution from dialogue, distributed systems can be much more loosely coupled until runtime. At the outset, all that two components who wish to

communicate must know about each other is their respective locations. It is only when they actually wish to interact that they become more tightly coupled, but this is no different to traditional technologies, the difference is the timing of when it is required.

6 Conclusions

Evaluating software systems is never an easy task. The motivation for the experimental work described in this paper was to evaluate the Mobile Agent and Mobile Object abstractions. The evaluations in this paper have been undertaken following Basili's GQM methodology. Using this technique a set of tangible metrics were developed. These can also be used to assist a system designer in identifying which components in their system, if any should be mobile

The experimental work has shown that by reducing the conceptual diffusion in a system, mobile code abstractions are able to offer improved semantic alignment with the business process the system is intended to support. Further, the implication of using either of these abstractions is that the system designer is able to divorce the dialogue from the issues of distribution. This separation leads to the construction of loosely coupled systems, which is an essential aspect of any system that wishes to remain agile.

The experiments have shown that mobile code systems are relatively agile, with the mobile agent abstraction being slightly more so than the mobile object abstraction. The differences in each implementation with respect to agility are identical to the differences in semantic alignment. This is due to lower conceptual diffusion in the mobile agent system, something that is enabled by the autonomy of the agent metaphor.

Employing the correct abstraction can have fundamental consequences when building distributed systems. Instead of the flat plane of components offered by traditional technologies, mobile code abstractions remove this opacity and exposes the rich network environment.

7 References

Basili, V.R., Caldiera, G., Rombach, H.D., (1994), “The Goal Question Metric Approach”, Encyclopedia of Software Engineering, pp 528-532, Wiley and Sons.

Boehm, W., Brown, J.R., Lipow, M., “Quantitative Evaluation of Software Quality”, Proc. 2nd International Conference on Software Engineering, 1976, pp 592-605.

Chess, D., Harrison, C., Kershenbaum, A., (1997), “Mobile Agents: Are they a Good Idea?” in Vitec (1997).

Clements, P.E., Papaioannou, T. and Edwards, J.M., (1997), "Aglets: Enabling the Virtual Enterprise" , Proceedings of the 1st International Conference on Managing Enterprises - Stakeholders, Engineering, Logistics and Achievement, ME-SELA '97 , Wright, Rudolph, Hanna, Gillingwater and Burns (eds), Mechanical Engineering Publications, Loughborough University, July 1997, pp 425-432, ISBN 1-86058-066-1.

Coutts, I., Edwards, J., (1998), "Support for Component Based Systems:Can Contemporary Technology Cope?", Intelligent Systems For Manufacturing, Edited by L.M. Camarinha-Matos et. Al., Kluwer Academic Publishers, ISBN 0-412-84670-5, pp279-288."Basys paper"

Franklin, S and Graesser, A., (1996) "Is it an Agent, or just a Program?: A Taxonomy for Autonomous Agents", Proc. of the 3rd Int. Workshop on Agent Theories, Architectures, and Languages, Berlin, Springer-Verlag.

Gray, R., (1997), "Agent Tcl: A flexible and secure mobile-agent system", PhD thesis, Dept. of Comp Sci, Dartmouth College.

ISO, (1992), International Standards Organisation, "Basic Reference Model of Open Distributed Processing, Part 1: Overview and guide to use", ISO/IEC JTC1/SC212/WG7 CD 10746-1.

Kiniry, J., Zimmerman, D., (1997) “A hands-on look at Java Mboile Agents”, IEEE Internet Computing, Vol (1) 4, July/August, 1997.

Kogure, M., Akao, Y., “Quality Function Deployment and CWQC in Japan”, Quality Progress, October 1983, pp 25-29.

Lange, D.B., Oshima, M., (1998) "Mobile Agents with Java: The Aglet API", World Wide Web Journal.

Lange, D.B., Oshima, M., (1999) "Seven Good Reasons for Mobile Agents”, Comm. ACM, Vol 42 (3), p88-89.

McCall, J.A., Richards, P.K.z, Walters, G.F., “Factors in Software Quality”, Rome Air Development Centre, RADC TR-77-369, 1977.

Minar, N., Gray, M., Roup, O., Krikorian, R., Maes, P., (1999), “Hive: Distributed Agents for Networking Things”, Proceedings of. ASA/MA ‘99.

MML, The Mobility Mailing List, http://mobility.lboro.ac.uk

ObjectSpace: Voyager Core Package Technical Overview (1997), http://www.objectspace.com/Voyager

OMG (1994), Object Management Group, "The Common Object Request Broker: Architecture and Specification", Object Management Group Inc. 492 Old Connecticut Path, Framingham, MA., USA 1994.

Orfali, R., Harkey, D., Edwards, J., (1996), “The Essential Distributed Objects Survival Guide", John Wiley and Sons Inc. New York. USA 1996.

Papaioannou, T., Edwards, J.M., (1999) “Using mobile agents to improve the alignment between manufacturing and its IT support systems”, Journal of Robotics and Autonomous Systems, Vol 27, pp45-57.

Papaioannou, T., Minar, N., (1999), “Mobile Agents in the Context of Competition and Cooperation”, Proc. MAC3 workshop, part of Autonomous Agents ‘99 conference, Seattle.

Rothermel, K., Hohl, F., (eds) (1998), “Mobile Agents”, Proc. 2nd International Workshop, Stuttgart, Germany, September 1998, Lecture Notes in Computer Science 1477, Springer-Verlag.

Solingen, R.V., Berghout, E., (1999), “The Goal/Question/Metric Method”,McGraw Hill, ISBN 0-07-709553-7

System Software Associates Inc., (1995), "BPCS Client/Server Distributed Object Computing Architecture".

Straßer, M, Baumann, J., Hohl, F., (1996), “Mole - A java Based Mobile Agent System”, in Proc. ECOOP’96 workhop on Mobile Object Systems.

Sun Microsystems Inc., (1998), "Java Remote Method Invocation Specification", Revision 1.50.

Vinoski, S., Chief Architect at Iona Technologies Inc, comment made in dist-obj mailing list. Thu, 15 Jul, 1999.

Vitek, J., Tschudin, C., eds, (1997), "Mobile Object Systems, Towards the Programmable Internet", Lecture Notes in Computer Science 1222, Springer-Verlag.

White, J. E, (1994), "Telescript technology: the foundation for the electronic marketplace", White Paper, General Magic Inc., 1994, also appears in “Software Agents”, J. M. Bradshaw (Editor). MIT Press, 1997. ISBN 0-262-52234-9.

Waldo, J., Wyant, G., Wollrath, A., Kendall, S., (1994), “A note on distributed computing”, Sun Microsystems Technical Report SML 94-29, 1994.

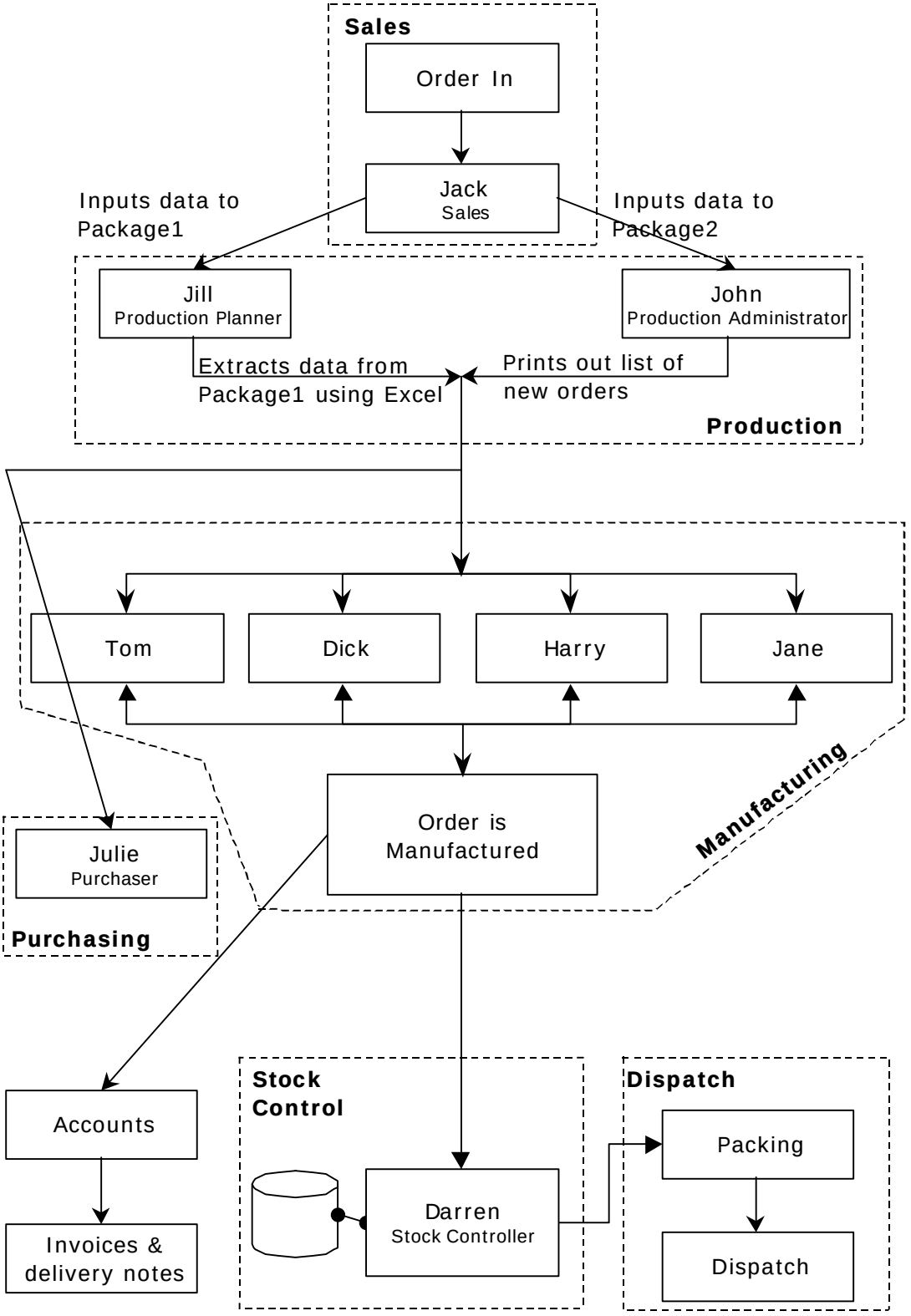
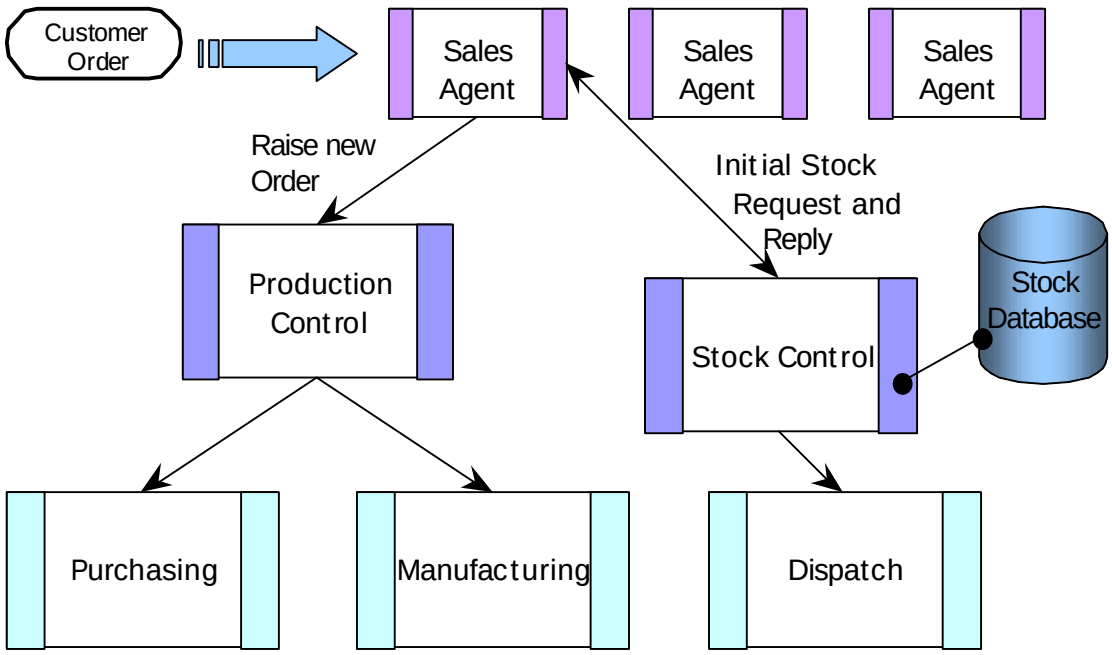


Fig 1. Acme's business processes on receipt of an Order



A new Customer Order is placed with a Sales Agent. The Sales Agent then interrogates Stock Control to see if the order can be fulfilled from the existing stock. If it can, a new Order is raised and the items are allocated to that order number before being dispatched to the customer, along with an invoice.

If the items are not in stock, then the order is passed to production control where again, an Order is raised. Accompanying this Order is a new Works Order for the required manufacturing of the requested products, or product parts. The Works Order is then passed to manufacturing for completion, and if necessary purchasing for replacement of raw materials. Once the product or parts are completed, they are booked into Stock Control before being checked out again for dispatch. The standard delivery time at Acme is three weeks, unless the order is being specially manufactured to specifications submitted by the customer.

Fig 2. Top level view of business processes within Acme

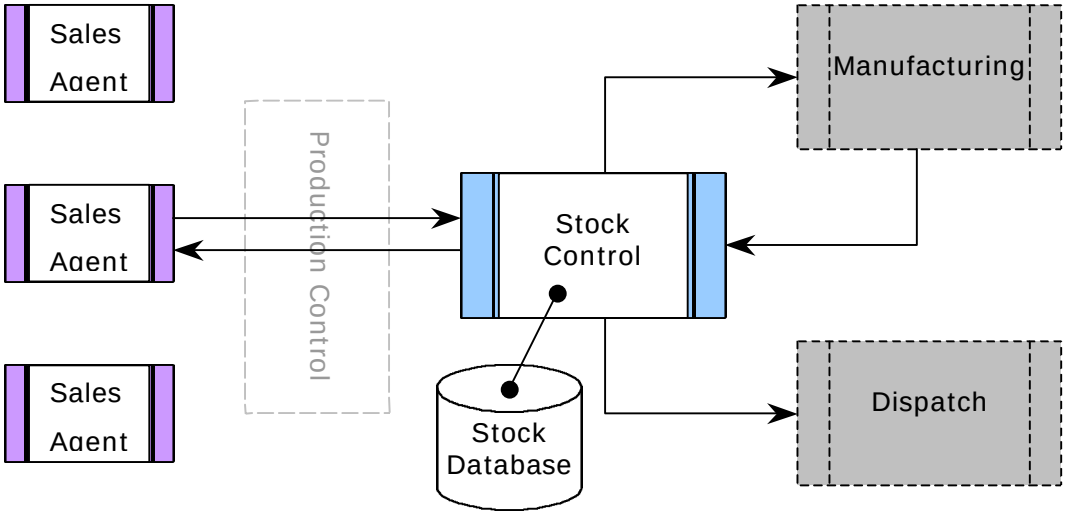


Fig 3. Modified Process Model for Acme

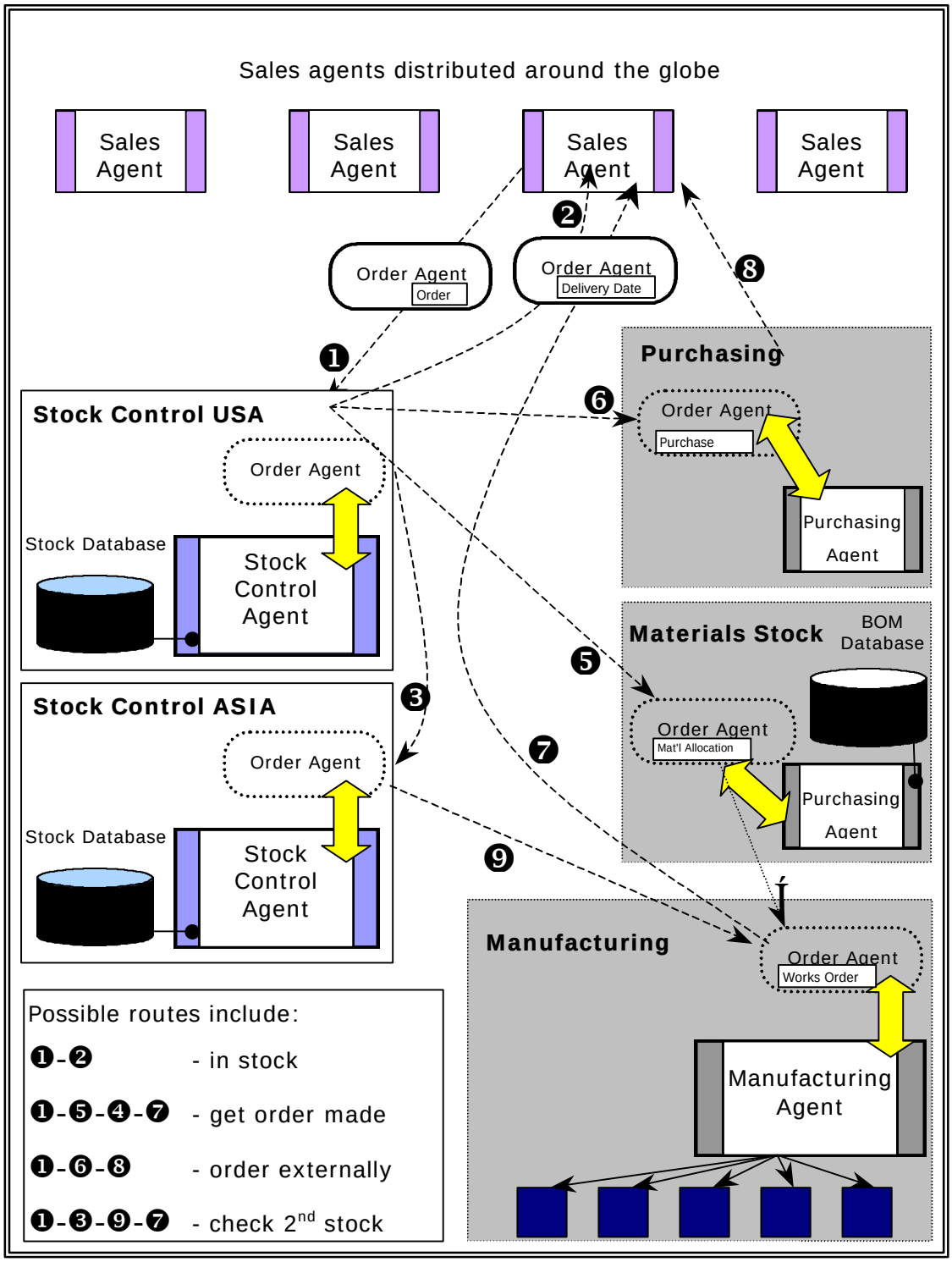


Fig 4. Agent Sales Order Process Model – with example routes for OrderAgents

Generated Questions	Metric Number
How well does the system support change?	
How easy is it to understand the system?	
How many business entities map onto data abstractions	(1)
How many business processes map to software methods	(2)
Which real world entities that are mobile are also mobile in the system	(3)
Which real world entities that are static are also static in the system	(4)
How many components are there in the system	(5)
How many lines of code are there	(6)
How many comments are there	(7)
How easy it was to modify the system?	
How many conceptual entities must be changed - for example requirement a)	(8)
How many objects must be changed	(9)
How many src files must be changed	(10)
How many interactions must be changed	(11)
How many components are there in the system relative to the size	(5) + (6)
How many real world entities map to a software component	(1)+(2)+(3)+(4)
How many components must be changed	(9)
How many interactions must be changed	(11)
How many inter-entity connections are there	(12)
How many methods of the object are public	(13)

Table 1. Questions generated using the Basilli GQM Method

Metric	Nature of metric
(1)	Identify information-based abstractions in the real world. Compare with info based abstractions in the software
(2)	Identify process-based abstractions in the real world. Compare with processes evident in the software.
(3)	Identify mobile elements of the real world, compare with mobile elements in the software
(4)	Identify static elements of the real world, compare with static elements in the software
(5)	Count the components
(6)	Count lines of code
(7)	Count comments, and get ratio of comments/method
(8)	Count changes for each requirement
(9)	Count changes for each requirement
(10)	Count changes for each requirement
(11)	Count how many files are changed for each requirement
(12)	Count number of inter object method invocations
(13)	Count number of public methods

Table 2. Metrics Generated using the GQM Method

Objects	Info Abstractions		Process Abstractions						SOP Logic	
	Order	Customer	SA	SCA	PC	M	P	D	MobAg	MobOb
BaseAglet			✓	✓		✓	✓	✓		
DBAglet				✓						
OrderAglet	✓								✓	✓
SlaveItin									✓	✓
SlaveDetails				✓						
SalesAglet			✓							✓
Result									✓	✓
GenericTask									✓	✓
StockCommit Task									✓	✓
DBStockRequest Task									✓	✓
NewOrderDialog			✓							
Order	✓									
OrderListEntry			✓							
OrderList			✓							
Product	✓									
ProductList				✓						
FutureLevels	✓									
OrderNumbers			✓							
SlaveList			✓							
Conceptual Diffusion	P	N/A	7	P	N/A	1	1	1	6	7

Table 3. Analysis of Conceptual Diffusion Present in Mobile Code

Mobile elements	Mobile agent	Mobile object
Order	✓	✓
Products	×	×
Materials	×	×
Static elements	**Mobile agent**	**Mobile object**
Sales	✓	✓
Stock Control	✓	✓
Production Ctrl	×	×
Manufacturing	✓	✓
Purchasing	×	×
Dispatch	✓	✓

Table 4. Results of Metrics (3) and (4)

Industrial Requirement	System	
	Mobile Agent	Mobile Object
The addition of new sales agents	{0,0,0,0}	{0,0,0,0}
The addition of new stock control centres	{3,3,1,2}	{3,3,1,2}
The removal of new additions	As A or B	As A or B
Allowing changes to the business logic of the SOP to be made easily	{1,1,0,1}	{2,2,0,2}

Table 5. Change Capability sets of SOP implementations after Industrial requirements

Distributed System Technology	Locator Requirement	Dialogue Requirement
Traditional Technology	Interface	Interface
Mobile code systems	Location	Interface

Table 6. Requirement of Distributed Systems