ELSEVIER

Computers in Industry 48 (2002) 17–27

COMPUTERS IN INDUSTRY

www.elsevier.com/locate/compind

An agent-based approach to engineering design

Kuo-Ming Chao[a,*], Peter Norman[b], Rachid Anane[a], Anne James[a]

[a]*School of Mathematical & Information Sciences, Coventry University, Priory Street, Coventry CV1 5FB, UK*
[b]*Engineering Design Centre, University of Newcastle-upon-Tyne, Armstrong Building, Newcastle-upon-Tyne, UK*

Abstract

Among the features of concurrent engineering is the notion of distributed design, and the ability to communicate design changes to multidisciplinary teams. Engineering design is a complex activity. Differences in system architectures and information structures, and co-ordination requirements tend to reduce the effectiveness of distributed design. Current thinking indicates that multi-agent systems (MAS) can alleviate some of the complex engineering design problems. In this paper, it is argued that agent attributes such as proactiveness and autonomy can overcome these limitations. Agents provide a flexible and dynamic approach to distributed/multidisciplinary design team which can reduce redundant design activities, and improve co-ordination.

Keywords: Intelligent agent; Proactiveness; ORB; Engineering design

1. Introduction

The successful design of large and complex made-to-order (MTO) products such as ships, offshore oil platforms and aeroplanes, require the collaboration of multidisciplinary design teams. These teams use specialised computer systems to aid their design process. Unfortunately, such computer systems may use different data representations of the product model. They may also utilise different design software packages. These packages may be written in dissimilar languages, for instance C, C++, Java or other languages, and installed on different hardware systems. Consequently, any collaborative communication or co-ordination between such diverse and different models, languages and system architectures may prove difficult.

Within the last decade, a number of design tools have been developed at the Engineering Design Centre, University of Newcastle. The following briefly describes these:

- Process flow diagram (PFD) system [1], used to design a gas-condensate separation process. The system generated a PFD frame and rule model of the process.
- Electrical system [2], computes the power demand and power generation equipment dimensions. The system can also select equipment from the supplier's catalogues. This selection is based on the PFD specifications.
- Plant operation system [2], produces diagnostic rules for functioning plant. These rules are based on the output of the PFD system.
- Associativity data generation (ADG) [3], calculates the strength of the relationship between any two pieces of equipment based on connectivity, function, and cost. The equipment and connectivity data

* Corresponding author. Tel.: +44-24-76888908.
E-mail address: k.chao@coventry.ac.uk (K.-M. Chao).

0166-3615/02/$ – see front matter
PII: S0166-3615(02)00007-6

are carried from the catalogue and the PFD system, to the ADG system via a knowledge sharing and reuse tool [4]. The ADG system also generates the input for the spatial layout design (SLD) system.

- SLD system [5] is responsible for producing an optimised graphical layout in two dimensions. The system is based on simulated annealing algorithms with the consideration of global constraints to search all possible spaces in order to generate a viable layout.
- 3D parametric cad system [6] improved layout system that generates 3D parametric CAD information. The parametric CAD system can forward any design changes to the cost estimating system.
- Cost estimating system [6] includes a set of sophisticated algorithms that can compute different levels of a product model to produce the required cost in the conceptual design. It also allows for a revision of costs following a design change.

Some of these tools are implemented in Unix and MS Windows using C, C++ and Java. Some tools use commercial CAD, PFD packages, or expert system tools to implement their methodologies. Consequently, no source code (apart from the application programming interfaces) will be accessible. The primary aim of this research was the provision of a single collaborative environment that allows for the dynamic interaction of such tools. Therefore, a means for providing that collaborative interaction was needed. Initially, the authors proposed a CORBA-based framework as a wrapper to facilitate communication between design tools. Within the CORBA-based framework an Interface Definition Language (IDL) is used to define the system interfaces. Each system has a client and a server program, so it can trigger other systems and be invoked by yet others. The association between these systems is identified as a sequential flow (see Fig. 1).

The relationship between different individual data models is monitored by a knock-on effect mechanism [2]. The mechanism traces the information flow between models, and is based on the interdependent relationships between objects in the models (i.e. equipment items). When a change to information held within one system is made, changes to the related information within other systems are indicated by this knock-on effect mechanism.

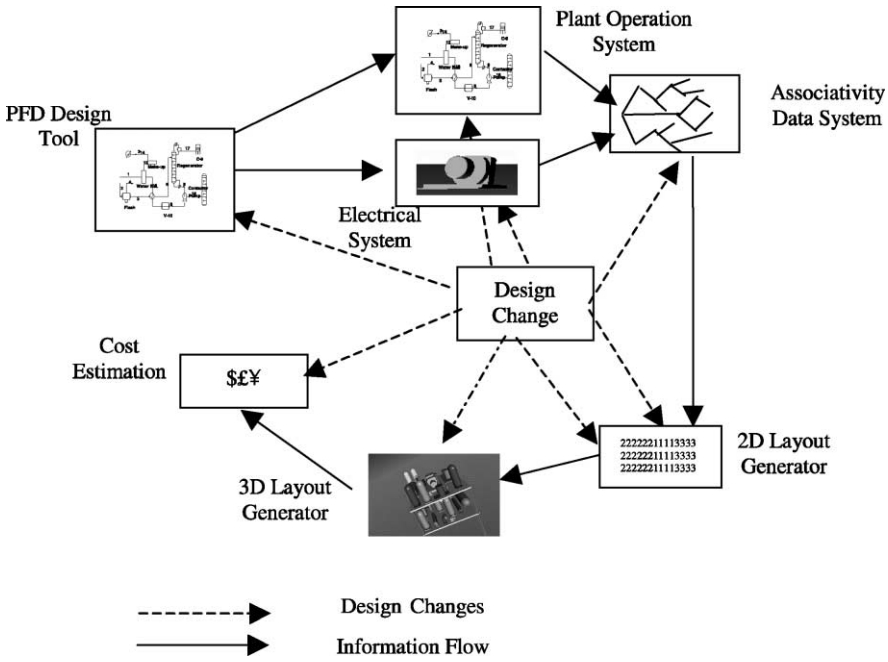

Fig. 1. Interactions between tools.

2. Analysis of a scenario

The following scenario taken from a case study of an offshore petrochemical plant design is used to assess the effectiveness of the CORBA-based framework.

A pump needs to be replaced by a bigger one in order to produce more pressure. The change from the PFD is passed to the electrical system for re-calculating and it produces a new specification of power generators in order to accommodate this change. The simulated PFD diagnosis system also takes this change to analyse the potential faults.

The ADG system takes the result from both systems to produce the new associativity data. The SLD starts to simulate the optimised layout according to the new data from the ADG system. A new optimised 3D layout and cost were produced to reflect this change.

Even though the process described is an automated sequential approach, it is considered to be a concurrent engineering approach. The participating systems in the framework can initiate changes and then trigger other systems to act accordingly. A number of design activities based on different versions can be carried out concurrently. However, the following limitations of using the CORBA-based framework were identified in this exercise:

- The systems were passive. The new pump, for example, could be expensive and its replacement may result in an excessive budget. The framework could not identify this problem until the last process. A system that can observe the changes and proactively provide a solution is required in order to reduce the unnecessary tasks.
- The systems were not autonomous. They could not control their own behaviour. Each system did what it was told by others. A system was unable to refuse invocation. The SLD system, for example, may generate a new layout and the designer may not have the opportunity to look at it. The ADG system with new data may trigger the system again and overwrite it. A level of autonomy is required.
- The interfaces were too specific. This architecture requires users or other systems to have a good knowledge of the functions that other systems provide. Function names and arguments need to be specified by the requested system before the functions can be invoked. The APIs supported in commercial packages include low-level functions that are awkward to use. The other system required, is one to control the others in the correct sequence, in order to produce the result. A level of abstraction is required.

An intelligent agent that possesses properties like proactiveness and autonomy as well as reactiveness provides a promising solution to overcome the aforementioned limitations. Intelligent agents with the most sophisticated capabilities are known as *proactive* agents. Proactive agents have the ability to assess their surroundings and decide what action to take.

An example of this type of agent can be seen in engineering design environments where the agent monitors process flow design and prices of equipment items needed, and estimates the cost in order to conform to budget constraints [7].

These agents can gather the related information and exercise reasoning skills that will then enable it to determine a specific course of action. Thus, the autonomous agents perform tasks without the need for constant human intervention or other agent interaction. Because an agent controls its behaviour within its domain, it can choose whether or not it will approve requests made by other agents [7]. A reactive agent can respond to the requests made by other agents. An agent is an abstraction that represents an application and enables it to interact with its collaborative environment by reacting to the requests from other agents in order to solve common problems. Reactiveness without autonomy would imply obedience and the same problem as that mentioned, where a system blindly follows instructions from another system. Reactiveness with autonomy means that an agent can react to a request in a number of ways, one of which may be to obey the request and another to reject it.

3. The proposed framework for co-operation

The proposed system (see Fig. 2) includes four parts: communication, mental model, observation mechanism and application. The communication mechanism is a message passing mechanism built upon an Object Request Broker (ORB). It transports the

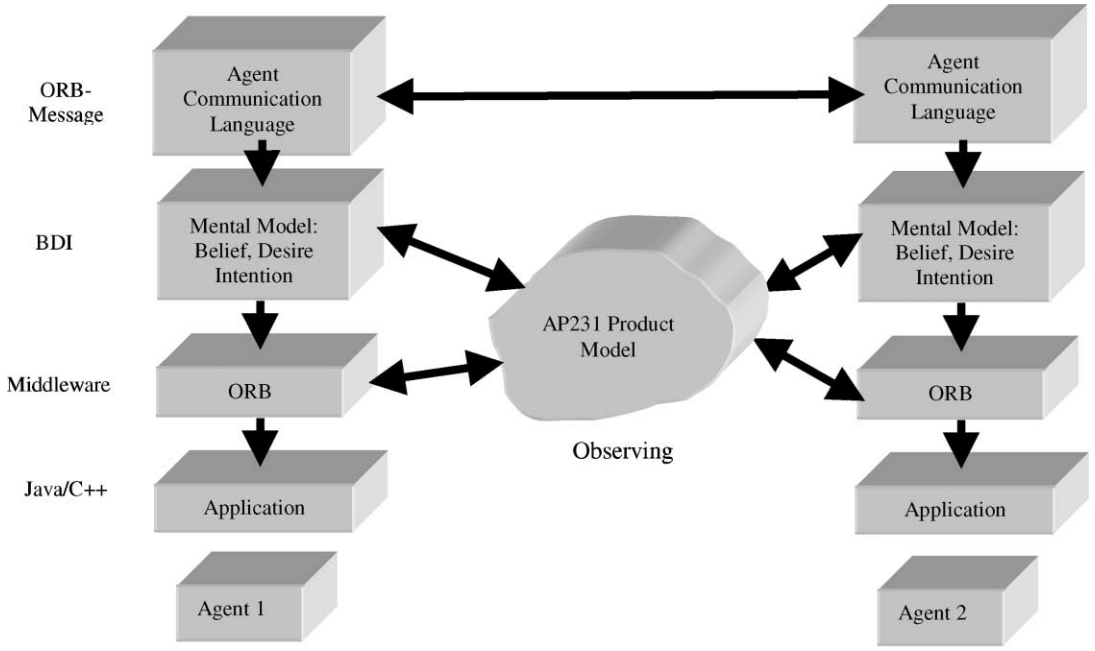

Fig. 2. The proposed architecture with four layers mechanism.

agent's message in the syntax of Agent Communication Language (ACL) to the recipient and the recipient parses the message. The mental model interprets the content of the message, reasons with it and asks the underlying application to perform the task. The underlying application returns the result to the mental model. The mental model generates the appropriate reply and forwards it to the request agent. Even though message passing is used for the communication, and the mental model reasons with the message content, the remote method invocation is inevitably used to invoke the functions in the application. The interface between the application and the mental model uses the ORB in order to accommodate diverse applications. The observation mechanism is associated with the Belief, Desire, Intention (BDI) mental model that allows the agents to autonomously determine which objects in other agents they need to observe and what action should be taken. In addition, a common product data model, in this case built upon STEP AP231 [8], allows the agents to have a consistent interface for accessing each other's product data model. Although Fig. 2 illustrates the interactions between two software agents only, this model can also be applied to multiple agents' conversation.

3.1. The mental model

The agents communicate with each other through an ACL. Since the agents are intended to perform some action by virtue of being sent a message [9], then the ACL messages are actions or communicative acts (CAs). CAs, are a special class of actions, that are modelled on speech act theory [10].

The BDI mental model implementation parses the incoming message from ACL and reasons with it. The BDI mental model then invokes the appropriate server methods accordingly as a client side. The BDI model is a reasoning mechanism that interprets the informational, motivational, and deliberative states of the agents [11]. The following formulas describe the behaviour of the BDI mental model:

- $B' = B(P, B)$ where P is the agent's current perception,
- $D' = D(B, D)$,
- $I' = I(B, D, I)$.

Belief is about facts or information that the agent receives from its internal states or environment status when currently participating in some process. Desire is a list of goals that the agent can or tries to achieve.

Intention is a plan for the agent to achieve its goal. These attributes with three functions (B, D and I) allow the agent to behave dynamically, according to some change in their environment. A new belief (B′) is derived from the old belief (B) and the agent's current perception (P) of the environment and its internal state. The new desire (D′) is derived from the new belief and the old desire (D). The new intention (I′) comes from belief, desire and old intention (I).

3.2. Mobile agent

A mobile agent (MA) is a piece of software that can move from one host to the other over the network to carry out the task that it was designed for. The mobile agent is a delegate of the user who grants the program a certain degree of autonomy to achieve its designed goal. Mobile agents are most likely to be useful in three general situations. One is disconnected computing such as laptops—they frequently disconnect from the network or use a wireless network that might become disconnected on short notice. The second is information retrieval situations—applications where the agent can be sent to the large data source and filter through the data locally. The third category is dynamic deployment of software [12].

3.3. Dual mode of observation

Observation is an important mechanism that enables agents to act proactively. In the engineering design process changing design specifications may include the introduction of new items, the removal of some items, and the replacement of some items with others. The frequency of these changes may vary. In order to deal with this dynamic situation and increase communication efficiency, the observation mechanism includes two components, a dual mode of observation and a set of meta BDI model rules. The introduction of a dual mode of observation is to improve the agent's awareness of the dynamic environment in which it operates. The meta rules allows the agent to have control over the scope and nature of its observation. When an agent needs monitoring, the agent deploys an MA to the observed agent. The design agent through its meta BDI rules generates the rules for the MA. The design agent is responsible for the global observation whilst local observation is performed by the MA. As a result, the filtering of data at the local level and the reduction of communication requirements are supported by a dual mode of observation that stems from the relationship between the agent and the MAs, and their respective roles.

At the global level the design agent maintains a table of the active objects that represent the product model. Through its observer mechanism the agent is able to keep the table up-to-date by being set to observe any object creation or deletion that takes place in the observed environment. This updating is essential, since if the environment holds objects of which the agent is not aware, the logical integrity of any decision making process is flawed. Likewise a lack of knowledge that an object has been deleted would lead to a run-time error if the agent were to attempt to reference such a deleted object.

A general schema for the ORB Observer mechanism is presented in Table 1, with its main methods, written in an Interface Definition Language (IDL).

When one agent creates any of the objects, the *addObserver* method is called and the remote object reference is stored in other agents' tables. In the case of objects being identified to monitor the design agent responds by dispatching an MA to the corresponding remote site. The MA is made aware of which object and data to monitor, and where they are, through the remote object reference obtained from the ORB Observer mechanism. When the object is about to reach the end of its life cycle, the *deleteObserver* method will remove the remote object reference from the table and the agent will instruct the MA to stop the monitoring activity on that object to prevent any run-time error.

Table 1
The schema for the ORB

```
module ORBObserver
{
  interface Observer
    {
      void update(in any observables, in any message); };
  interface Observable
    {
      void notifyObserver(in any data);
      void addObserver(in Observer objRef);
      void deleteObserver(in Observer objRef);
  long countObservers(); }; };
```

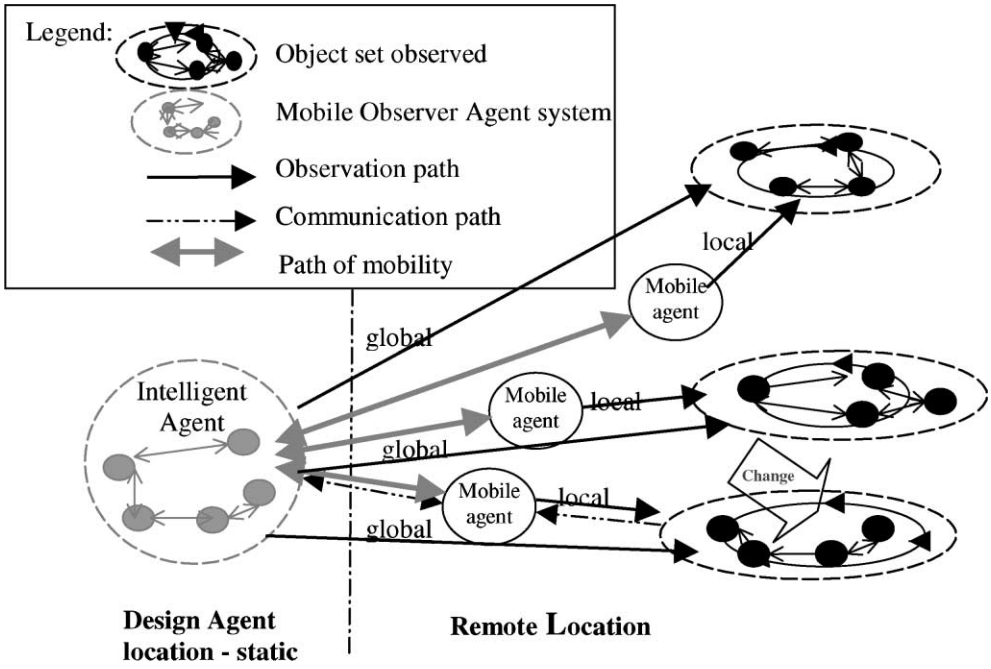

Fig. 3. Dual mode of observation.

Under the local mode of observation, the observer mechanism of the MA and the ORB Observer mechanism in the agent, together with rules allow the MA to monitor specific data. If an abnormal condition is detected in the observed agent, the MA will trigger the *notifyObserver* method to notify the agent. Fig. 3 demonstrates the MA moving between agents and illustrates the global and local observation [13].

3.4. BDI model and observation

The algorithm in Table 2 provides a framework that allows the ORB observer to interact with the BDI model, and the BDI model to work with the design agent to create a new MA for information collection. The MA sends back the filtered information to its design agent. The meta BDI model rules in the design agent generate a set of specific rules and conditions which are then embedded in the MA in order to monitor designated objects. When the rules need to be changed or the object being monitored is deleted, the MA is withdrawn. Such a system was implemented using multiple threads as shown in the Table 2 [14].

3.5. Other components

In order to facilitate the use of the integrated system, a common Graphical User Interface (GUI) was designed (see Fig. 4). The designers can activate and control their specific tools and agent systems through a user-friendly interface. The GUI also allows the users to view the product model in a hierarchical manner through the AP231 product model. The product model is represented in STEP's data definition language, EXPRESS [15], and then translated into IDL. The ORB compiler translates the IDL into C, C++, or Java code according to the requirements of the different applications. Each application maps its own product model to the interfaces generated from IDL. This mechanism allows diverse product models in the design agents to interoperate in distributed objects fashion.

The applications, the distributed/multidisciplinary design tools including their domain knowledge, are implemented using various tools such as CAD, PFD packages, expert system shells and programming languages (i.e. C++, Java). These tools provide different levels of APIs for the developers. The APIs are

Table 2
The schema for dual mode of observation

```
ObjectRefs = AddingObject(obj) //The ORB observing notes the new object
ObjectRefs = DeletingObject(obj) //The ORB observing notes an object deletion
//implement void update(in any observables, in any message) and assume objects being //maintained in the array of ObjectRefs.
//This pseudo operates as one thread in a multithread environment.
While (Observing_is_Active)
InterInform = GetInform(Internal_Application) //Get internal application data
ExterInform =GetInform(Other_Agent)
// Get the information e.g. Request from agents and DMA and Inform ACL
Belief= Modify(Belief, InterInform)// store the information from internal applications into BDI
Belief = Modify(Belief, ExternInform) //store the information from other agents into BDI
  If Monitoring_is_Requried() ==True and Monitoring_Is_Not_Activated == True
  Then
    ObjRef = DeterminingtheObject (ObjectRef[]) // Get observable object reference
    Attributes = DeterminingtheAttribute(ObjRef) // Get the observable attributes
    Rules = BDIRuleGenerator(ObjRef, Attributes) // Get rules from rule generator
    MA = Instantiating_Mobile_Agent(Rules) // Instantiate the mobile agent
    Send (MA, AgentAddress) // Send the mobile element to observable agent
    Belief = Modify(MA, AgentAddress, True) //Change the belief state
  Endif
```

wrapped by the ORB in order to communicate with the BDI model. The BDI model invokes the ORB interfaces to activate the functions in the applications and obtain the results.

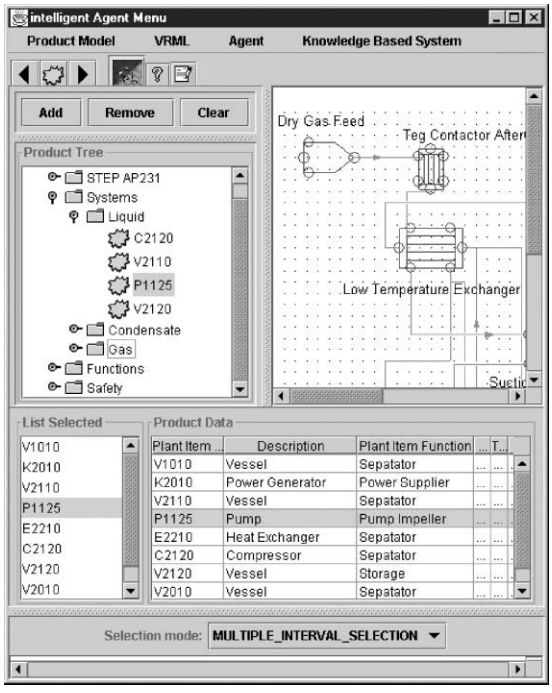

Fig. 4. A common Graphical User Interface.

This approach is similar to the previous CORBA-based solution but here the applications do not directly invoke other applications. Interaction is carried out through agents. The course of action of the application is controlled by its BDI mental model. Moreover, the knowledge and reasoning mechanism in the mental model can assess and determine whether the agent should fulfil the requests. Consequently, this mechanism allows the agents to have a certain degree of autonomy.

4. An example from the design of a petrochemical plant

Let us consider an example from the petrochemical plant design with three scenarios.

Scenario 1: The process engineer is required to change the operating temperature drop across the low temperature exchanger and so adjusts the flowsheet parameters accordingly. New equipment is selected and the change is passed to the other design agents. The layout and electrical systems report that this does not conflict with their existing models. They accept these changes and update their models accordingly. However, the cost engineer decides that this exchanger configuration would be cheaper if replaced by a set of two exchangers.

Scenario 2: The process engineer responds to this suggestion, accepting it and changes his model accordingly. The electrical system also accepts this change. However, the extra space required for the two exchangers configurations cannot be accommodated in the existing area. In model terms, constraints in the layout design have now been violated and the layout engineer cannot accept this change.

Scenario 3: In response to this conflict the electrical system produces a new design for the platform power supply. This alternative uses a generator set up with similar power output but which is smaller. However, this generator configuration is more expensive. The cost engineer states this violates a global project constraint.

A number of design processes have been carried out. These scenarios are used to examine the effectiveness of the proposed system. In Scenario 1, there is no improvement compared with the solution derived from the original CORBA-based framework. Scenario 2 also shows little improvement over this. Before the electrical system starts to run, the layout system has already determined that the implementation is not feasible. So the electrical system rejects the request from the PFD system. The diagnosis system, however, has already carried out the simulation.

In Scenario 3, the cost agent observed the changes in the electrical system, its mental model determined that the new design has violated the global constraint and autonomously issued it to the electrical system through an ACL at an early stage before the new design propagates to other agents. In this scenario, the number of design processes was reduced under the proposed systems. The 2D layout generator, 3D layout, and associativity data systems did not activate the simulations, after the cost system identified the excessive cost in the new proposed design.

In Scenarios 2 and 3, the proposed framework produces better results by reducing the number of design activities by one and three, respectively over the simple CORBA-based system. These measurements only consider the direct knock-on effect to the design with taking into account related activities such as co-ordination efforts. It is difficult to represent the improvements in time-scale, because the layout system uses random seeds to generate optimised layouts, so the system process time varies. These scenarios, however, demonstrate how the use of a middle layer, i.e. mental model working with the observation mechanism in the federation creates the proactiveness and autonomy of the underlying systems. The representation of beliefs, desires and intentions allows conflicts to be identified by distributed/multidisciplinary design teams before expensive work had been undertaken by the system. In other words, this approach enhances concurrency in the design process. Fig. 5 shows a 3D product model that satisfies all design agents, after a number of iterations of design changes and interaction between the designed agents.

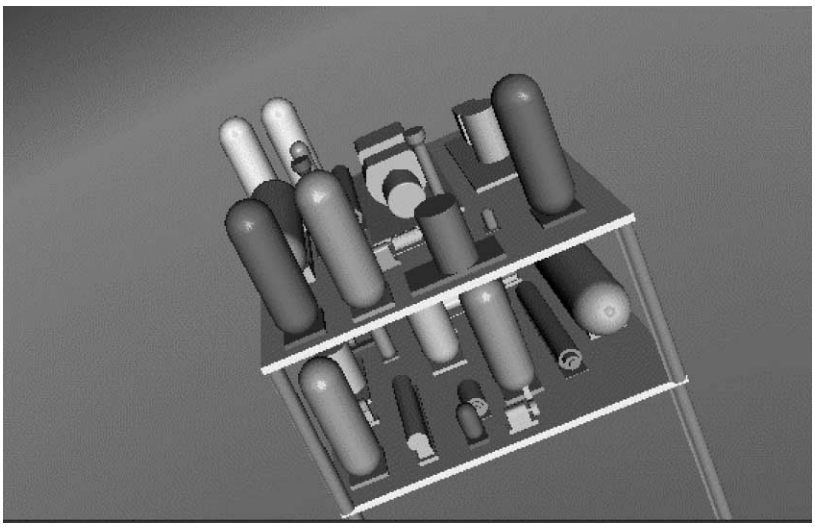

Fig. 5. A recommended design in 3D.

5. Discussion

There are a number of research groups using agent technologies to tackle various issues in concurrent engineering. However, none of them really explores or emphasises the potential of intelligent agents' mental attributes such as proactiveness and autonomy in reducing or avoiding redundant design tasks.

PACT [16] is one of the pioneering systems using agent technology to achieve collaborative engineering design in a distributed environment. The agent in PACT encapsulates program modules and invokes them remotely as network services when needed. The aim of the experiment is to demonstrate its framework for enabling knowledge reuse and sharing among design agents. Olsen et al. [17] and Khedro and Genesereth [18] use the same architecture to demonstrate applications in car design and the building engineering environment. The facilitator, a task-independent mechanism, co-ordinates the run-time activities of the individual software agents. The agent communication language allows agents to communicate in the same expressive language. An ontology is used to define common terms in the distributed/multi-disciplinary agents in order to support effective communication. Edmonds et al. [19] proposed a multi-agent group support framework to provide agents with context-sensitive dialogue and to enable group interaction with a remote geographic information system.

Agent-based blackboards [20], like federation systems, use grouping to manage agent interactions. Each local group of agents shares a data repository that is provided specifically for the efficient storage and retrieval of active shared data. Along with design data, tactical control knowledge can be represented in the shared repository, enabling reasoning about how to proceed with the design process and thus acquiring the same status and priority as the reasoning about the design itself. Within an agent group, a control shell (analogous to the federation's facilitator) notifies appropriate agents of relevant events [20].

Mori and Cutkosky [21] proposed reactive design agents that internally maintain states and rule-based knowledge and they do not yet believe that autonomous agents are practical for most engineering applications. A new initiative from DARPA [22] proposed a control strategy to enable autonomous agents to form super applications at run time. This results from the recognition that the autonomy and partial knowledge make agents useful for addressing the problems of complex engineering design environments. The power of the autonomous agent, however, can cause problems such as misinterpreting the request, making poor decisions and interacting with other agents in destructive ways [22]. Thus, it requires a control strategy to reduce risks of misuse or malfunction.

The A-Design theory [23] is founded on the notion that engineering design occurs in an ever-changing environment and therefore computer tools developed to aid in the design process should be adaptive to these changes. The essence of agents is to perceive their environment (design states) through sensors (i.e. functional inputs) and act upon their environment through effectors (i.e. modifications to the design states). The agents in A-Design do not fully conform to the definition of agency since its agents do not display specific behaviours of autonomy, mobility, or sociability [23].

Shen and Douglas [24] carried out a comprehensive literature survey in agent-based systems for intelligent manufacturing and offered a good classification of existing approaches. The survey, however, did not highlight the potential of proactiveness in the agent in preventing faults in the manufacturing process.

Agents as presented, whether referred to as software or intelligent agents, are marked by some important characteristics. In tackling large and complex concurrent engineering problems they focus on knowledge sharing and reuse, and design co-ordination. However, our view about intelligent agents goes beyond these aspects. It is not only consistent with these features, but includes also the desire to minimise redundant design activities. A reduction in design activities benefits other areas in design. It reduces the interaction of systems and minimises the number of design versions [25]. Before the agent formally publishes its design output, the feasibility of solutions is assessed. It minimises the number of design versions and the required space for design archives. A fundamental aspect of the research in the area of Distributed Artificial Intelligent is co-ordination, in a distributed environment [26]. An oversight in this research area, however, is the failure to identify the need for and development of agent-oriented systems acting proactively and autonomously.

This approach could be applied to other fields. However, a number of changes in the framework would be required for new application areas. Domain

knowledge in the applications and the mental models represented by the agents would need to be changed according to the characteristics of the new application areas. The IDL used to link the mental model and the applications would also need to change to reflect the new functions.

Tools such as Aglets (IBM [27]), FIPA-OS [9] and JAM agents [28] used in the experiment are prototype systems. Although their reliability is not adequate for real applications, they are able to demonstrate the effectiveness of this research. The monitored attributes (such as the cost and functions of equipment items) in the agents and the workflow are determined according to the human expert's arbitrary experiences in the current design environment. In addition, each agent only holds partial knowledge and information about the overall design, so the suggestions may not always be critical through the design life cycle. This also leads to a number of conflicts increased among design agents. In order to resolve this issue, an automated negotiation mechanism [29] is under development. More agent attributes having positive impacts on reducing design activities need to be explored. However, two characteristics of agents, proactiveness and autonomy have been shown to have an impact on the effectiveness and efficiency of engineering design.

6. Summary and conclusions

One requirement of concurrent engineering is the ability to communicate the effects of a design change (within a distributed environment) to all distributed/multidisciplinary design teams. However, the successful communication of such a change faces many hurdles. For example, different design tools may be based on different system architectures or methodologies. Such tools will often be located at different sites, and each tool may have a different data model of the design process. Consequently, the difficulties in understanding the diverse knowledge and information generated from these distributed/multidisciplinary design teams may reduce the effectiveness of a distributed design environment.

Initially, the authors proposed a CORBA-based framework to address these problems, but as the work progressed, the limitations of such a framework became self-evident. As a result of these limitations an agent-based framework (incorporating CORBA technologies) was developed. Agent attributes (for example, proactive, reactive and autonomous properties) are ideally suited to this particular problem. They provide a more flexible and dynamic means by which multidisciplinary design teams situated within distributed design environments can successfully design large and complex MTO products.

Acknowledgements

This research was supported by the UK Engineering and Physical Sciences Research Council (Grant No. GR/L25387) to the Newcastle EDC. We wish to thank AMEC Process and Energy Ltd. for their help with the case study and the members in the Newcastle EDC who have contributed to this work.

References

[1] B. King, Automatic Extraction of Knowledge from Design Data, Ph.D. Thesis, University of Sunderland, Sunderland, 1995.
[2] M. Guenov, Modelling design change propagation in an integrated design environment, Journal of Computer Modelling and Simulation in Engineering 1 (1996) 353–367.
[3] K.-M. Chao, B. Florida-James, P. Norman, P. Smith, B. Hills, Use of virtual reality and agents in engineering design, in: Proceedings of the Conference of EXEPERSYS-98, 16–17 November, 1998, Virgina Beach, Virgina, USA, pp. 183–188.
[4] K.-M. Chao, P. Smith, W. Hills, B. Florida-James, P. Norman, Knowledge sharing and reuse for engineering design integration, Journal of Expert System with Applications 14 (1998) 399–408.
[5] N. Smith, W. Hill, G. Cleland, A layout design system for complex made-to-order products, Journal of Engineering Design 7 (1996) 11–20.
[6] M. Guenov, K.-M. Chao, B. Florida-James, N. Smith, B. Hills, I. Buxton, Tracing the effects of design changes across distributed design agents, in: Proceedings of the Second World Conference on Integrated Design and Process Technology, 1–4 December, 1990, Atlantic City, Florida, USA.
[7] F. Greenstein, Electronic Commerce: Security Risk Management and Control, McGraw-Hill, New York, 2000.
[8] J. Owen, STEP An Introduction, Information Geometer Ltd., Winchester, UK, 1994.
[9] FIPA, Agent Communication Language Specifications 97 (http://www.fipa.org), 1997.
[10] J.R. Searle, Speech Acts, Cambridge University Press, Cambridge, 1969.
[11] S.A. Rao, M.P. Georgeff, BDI agents: from theory to practice, in: Proceedings of the 1st International Conference on Multiple Agent System, 12–14 June, 1995, Sans Francisco, California, USA, pp. 312–319.

[12] D. Milojicic, Mobile agent applications, IEEE Concurrency 7 (1999) 80–89.
[13] J. Plumley, K.-M. Chao, R. Anane, N. Godwin, Proactiveness and effective observer mechanisms in intelligent agents, in: Proceedings of the Conference on Intelligent Agent Technology, 23–26 October, 2001, Maebashi, Japan, pp. 144–149.
[14] K.-M. Chao, R. Anane, J. Plumley, N. Godwin, R.N.G. Naguib, A Mobile Agent Framework for Telecardiology, in: Proceeding of the 23rd Annual International Conference of the IEEE Engineering in Medicine and Biology Society, 25–28 June, 2001, Istanbul, Turkey, p. 788.
[15] D.A. Schenck, P.R. Wilson, Information Modelling: The EXPRESS Way, Oxford University Press, Oxford, 1994.
[16] M. Cutkosky, R.S. Engelmore, R.F. Fikes, M.R. Genesereth, T.R. Gruber, W.S. Mark, W.S. Tenenbaum, J.C. Weber, PACT: an experiment in integrating concurrent engineering system, IEEE Computer 26 (1993) 28–37.
[17] G.R. Olsen, M. Cukosky, J.M. Tenenbaum, T.R. Gruber, Collaborative engineering based on knowledge sharing agreements, Journal of Concurrent Engineering: Research and Applications 3 (1995) 145–159.
[18] T. Khedro, M.R. Genesereth, The federation architecture for interoperable agent-based concurrent engineering systems, Journal of Concurrent Engineering: Research and Applications 2 (1994) 125–131.
[19] E.A. Edmonds, L. Candy, R. Jones, B. Soufi, Support for collaborative design: agents and emergence, Communication of ACM 37 (1994) 41–47.
[20] S.E. Lander, Issues on multiagent design systems, IEEE Expert 12 (1997) 18–26.
[21] T. Mori, M. Cutkosky, Agent-based collaborative design of parts in assembly, in: Proceeding of ASME Design Engineering Technical Conference, DETC98/CIS-5697, 13–16 September, 1998, Athens, Greece.
[22] D.E. Dyer, Multiagent systems and DARPA, Communication of ACM 42 (1999) 53.
[23] M.I. Campbell, J. Cagan, K. Kotovsky, A-Design: an agent-based approach to conceptual design in a dynamic environment, Journal of Research in Engineering Design 11 (1999) 192–192.
[24] W. Shen, H.N. Douglas, Agent-based systems for intelligent manufacturing: a state-of-the-art survey, International Journal of Knowledge and Information Systems 1 (1998) 129–156.
[25] B. Florida-James, N. Rossiter, K.-M. Chao, An agent system for collaborative version control in engineering, The International Journal of Manufacturing Technology Management 11 (2000) 258–266.
[26] G. Coates, I. Ritchey, A.H.B. Duffy, W. Hills, R.I. Whitfield, Integrated engineering environments for large complex products, Concurrent Engineering—Research and Applications 8 (2000) 171–182.
[27] IBM, Aglets Software Development Kit (http://www.trl.ibm.com/aglets/).
[28] M.J. Huber, JAM Agents in a Nutshell (http://members.home.net/marcush/IRS), 1999.
[29] J.-H. Chen, K.-M. Chao, N. Godwin, C. Reeves, P. Smith, An automated negotiation mechanism based on co-evolution and game theory, in: Proceedings of the Conference of the 17th ACM Symposium on Applied Computing, 10–14 March, 2002, Madrid Spain.

Kuo-Ming Chao received the MSc degree and the PhD degree in computer science from Sunderland University, UK, in 1993 and 1997. He was a Research Associate in the Engineering Design Centre at University of Newcastle-upon-Tyne in 1997–2000. In late 2000, he joined the School of Mathematical and Information Sciences at Coventry University as a senior Lecturer. His research interest includes intelligent agent technologies, game theory, engineering design and supply chain management.

Peter Norman is Associate Director of the Newcastle Engineering Design Centre and a senior Lecturer in the Department of Chemical and Process Engineering. Current research interests centre on systems integration, integrated modelling environments and product data modelling, as well as design for a clean environment. He has been principal Investigator for projects on design process integration and eco-design decision support. For the past 10 years Dr. Norman has closely followed the development of STEP (ISO 10303) as applied to the process industries and the systems integration area.

Rachid Anane is a senior lecturer in computer science in the School of Mathematical and Information Sciences, Coventry University, UK. He holds a BSc in computer science from the University of Manchester, an MSc and a PhD in computer science from the University of Birmingham. His research interests include software engineering, distributed systems and the modelling and analysis of historical data. Recently, he has taken an active interest in the role of software agents in supporting distributed applications.

Anne James is the research group leader for the data and knowledge engineering research group (DKERG) at Coventry University. Dr. James received the BSc degree in computer science and languages studies from the University of Aston in 1980. She was awarded a PhD degree in the area of databases and data modelling in 1986 after a programme of work completed at The Polytechnic, Wolverhampton. Since then, Dr. James has remained engaged in teaching and research in the areas of data and knowledge representation. She is an active member of the British Computer Society and participates on a number of programme committees for national and international conferences. Dr. James is currently acting Associate Head of Computer Science at Coventry University.