N94-35071

Engineering Large-Scale Agent-Based Systems with Consensus

A. Bokma, A. Slade,
S. Kerridge & K. Johnson

Artificial Intelligence Systems Research Group,
SECS,
University of Durham
DURHAM DH1 3LE
e-mail: Albert.Bokma@durham.ac.uk

Abstract

The paper presents the Consensus method for the development of large-scale Agent-Based Systems. Systems can be developed as networks of Knowledge Based Agents (KBA) which engage in a collaborative problem solving effort. The method provides a comprehensive and integrated approach to the development of this type of system. This includes a systematic analysis of user requirements, as well as a structured approach to generating a system design which exhibits the desired functionality. There is a direct correspondence between system requirements and design components. The benefits of this approach are that requirements are traceable into design components and code thus facilitating verification. The use of the Consensus method with two major test applications showed it to be successful and also provided valuable insight into problems typically associated with the development of large systems.

1. Introduction

In recent years there has been a noticeable shift from large mainframes towards networked hardware architectures, requiring a different approach to the construction of large systems to run on these new platforms. In addition, there is an increased need for faster and more intelligent systems, which, however, need to be engineered to the same rigorous standards expected of systems developed for more traditional implementation paradigms. Consequently, there is a need for methods which cover the following points:

- addressing the special needs of knowledge based systems
- developing distributed solutions
- reducing response times to meet real-time requirements
- meeting validation, verification as well as quality assurance constraints

The Consensus method focusses on the design concerns of:

- distributed knowledge based applications
- systems of arbitrary size (typically large scale)
- real-time systems

A comprehensive method for the development of such systems requires a systematic approach to help the user to specify a solution for a given problem. In the design of concurrent Knowledge Based Systems (KBS) it is important to handle both the technical aspects of specifying systems, as well as the organisational aspect of managing projects, especially where the development of large systems are concerned. The Consensus method includes both a specification technique and a life-cycle model, although the latterwill be covered in less detail.

The method specialises in the development of large real-time applications, that build concurrent knowledge based components into an agent-based architecture, covering the complete life cycle from concept through to operation, and includes:

- an analysis of the need and functioning of the system,
- the selection of design approaches appropriate for the task,
- detailed design and implementation,
- verification, validation and testing,

The Consensus method, is the result of research into software development methods and approaches, and has been refined through practical experience gained from applying the method to the development of a large-scale application in Air Traffic Control, as well as a second application concerned with Dynamic Tactical Planning.

2. Current Developments

There are a number of approaches which have been influential in the formulation of the Consensus method, which can be divided into three distinct areas:

- life-cycle models
- system specification methods
- KBS approaches

There are various life-cycle models which have been proposed over the years to help in a ordered and systematic development and maintenance of systems. Many of the traditional sequential models *[ROY70]* and their variants have been found wanting as they prescribe a one-way development process which does not promote iteration and feedback. This is a serious shortcoming as there are a number of factors that have to be weighed against each other in the process of developing systems which both fulfil the needs and are feasible and well engineered. The Spiral Life Cycle model *[BOE88]* has embraced what is actual development practice, which the other life-cycle models have tended to deny. As this approach is much closer to actual development practices and particularly suited for the development of distributed systems Consensus has opted for this model.

The Spiral Life Cycle Model is iterative by nature and therefore an iterative method for system analysis and design is required. One such method which has increasingly been used in the last two decades is structured analysis and design. Consensus has been influenced particularly by Hatley-Pirbhai *[HAT87]*, a complete specification method for real-time systems which has been used successfully in industrial and commercial applications. It uses both structured analysis and design and proposes the joint development of a system requirements model and a system architecture model. The motivation shared by Consensus is that for successful system specification one needs to balance what the user wants with what can be done, given the available implementation technology. As Hatley-Pirbhai has been successful in actual system development this proved to be a good starting point for a system specification method.

Hatley-Pirbhai also specialises in real-time system development which Consensus is also interested in although it does not specialise in it. At the same time real-time systems are becoming incerasingly widespread and provisions that can be made for them will prove useful in the future. Amongst real-time extensions to the basic structured analysis and design approach, apart from Hatley-Pirbhai there are also Ward-Mellor *[WAR85]* and DARTS *[GOM84]*. There are no fundamental differences between the notations of Ward-Mellor and Hatley-Pirbhai, which would have been significant for Consensus (which adopts the Hatley-Pirbhai conventions). By contrast, DARTS has proven to be interesting as it uses the concept of dividing the problem solving activities into tasks. The guidelines for the identification of tasks were useful for another goal of Consensus, namely the identification of components which could engage in collaborative problem solving and operate in a distributed fashion.

One problem with structured specification methods including Hatley-Pirbhai is that they are geared towards procedural implementation paradigms. Consensus, however, addresses the specification of parallel and cooperating KBS. A different approach was therefore needed to deal with the specific needs of distributed systems. There are a number of approaches which have been developed in recent years and may be thought of as being relevant to Consensus. A number of approaches, methods and tools were examined. Amongst the different approaches the most significant for Consensus was Cassandra *[CRA89]* as it provides an architecture paradigm specific for distributed KBS, which fills the gap left in the architecture model of Hatley-Pirbai.

Cassandra does provide a paradigm for developing KBS as networks of smaller knowledge based components embedded in a collaborative system architecture. This satisfies the goal of Consensus to develop parallel and cooperating KBS. At the same time little help is given as to how to go about specifying large systems and how to arrive at a suitable system architecture for a given problem. By combining the benefits of these approaches and by adding the necessary methodological support to steer the specification process, the systematic specification of distributed KBS can be achieved. Adaptation of existing analysis methods and experience from developing two major test applications helped to fill this gap and to generate guidelines to direct the specification process towards suitable system acrchitectures.

In recent years a number of KBS technologies were developed. One of the most widely known is KADS *[SCH89] [DEH92]*, which allows the modelling of the application domain from different perspectives with the help of a number of interconnected models. Other projects like Reakt *[FJE92]* and AC-Knowledge *[ACK92]* have sought to enhance KADS and to deal with issues such as real-time specification and knowledge acquisistion. Knowledge acquisition tools will be useful for a number of applications but most tend to guide the specification process towards emulating the application domain. Although this may be desirable in some cases the specification of large and complex systems requires to balance both the need of the user and constraints imposed by software engineering principles and the available implementation technology. Structured analysis also uses a modelling technique, but it is more flexible and allows the developing of the requirements and architecture in a way that suits both the need of the user and the constraints of the system. The use of knowledge elicitation tools should therefore be confined to the specification of individual components which have a strong expert knowledge element that cannot be addressed with structured analysis alone. Alternatively, they can be useful as a prototying tool to develop an operational model of components to determine its behaviour and to ensure completeness of requirements.

3. The Consensus Method

The purpose of the Consensus project is the generation of a software engineering method for the development of large parallel Knowledge Based Systems, this involves the development process, from conception through to operation, including the requirements definition, the design of the solution and the implementation culminating in a working system.

Consensus believes to have produced a comprehensive, yet compact method, which is intuitive and easy to use while being effective enough to deal with large complex applications.

It's structured analysis approach facilitates the development of large complex systems by subdividing them into manageable parts which can in turn be further analysed and specified. It allows for systematic exploration of the requirements and forces the analyst to focus on the specific requirements of sub-components and their inter-relationships.

Given the current shift towards networked architectures, distributed systems are becoming more common-place. Distributed, collaborative problem solving is in step with these developments and allows exploitation of the benefits of networked architectures; the response of Consensus is to develop systems as networks of medium grain-sized knowledge based agents, which engage in a collaborative problem solving activity.

Paired with this development is the increased need for intelligent processing in industrial and commercial applications satisfying stringent software quality constraints. Thus the successful advance of Knowledge Based Systems especially for large applications requires a distributed and modular architecture and a method which can deal with the specification of systems of that type.

From a software engineering perspective, one consideration that is often overlooked is that in order to develop successful solutions,

the analysis of desired functionality has to be weighed against constraints imposed on the solution by the implementation platform. Therefore, the analysis of requirements cannot be divided from the specification of the system architecture. Consensus adopts an integrated approach where the requirements are developed in conjunction with the design of the solution, thus ensuring that at every stage of development the resulting system fulfils the requirements while taking account of the constraints.

The structured analysis and design approaches are used to provide a distributed architecture of independent tasks which communicate with other tasks in the overall system process. These can in turn be translated into a network of distributed KBA. Consensus combines a distributed agent architecture with structured analysis and design providing a software engineering method for developing systems as networks of KBA.

The key concepts that the method makes use of, can be summed up as follows and are depicted in figure 1:

- the System Specification should comprise not only the System Requirements but also the System Architecture - these should be developed together;
- the System Requirements specify what the system is to do and is independent of the implementation technology;
- the System Architecture specifies how the system is to be structured and is dependent on the implementation technology.

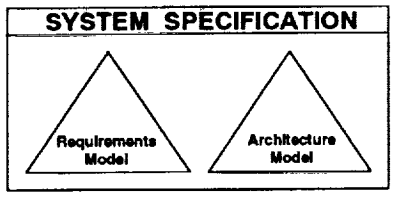

Figure 1: System Specification

The System Requirements and System Architecture are considered together, starting with a high level model of the system, and proceeding by refinement and iteration until a detailed, complete and comprehensive specification of the system is produced.

The primary benefit of this approach is that early partitioning and allocation of functions in the system helps to identify critical functions, which can be prototyped or reappraised, thereby leading to a clearer understanding of the need and the best way to structure the design. This results in a comprehensive and integrated system development process and favours traceability and consistency which are vital in the development of large, complex systems.

3.1 The Requirements Model

Problems are frequently too large and complex for solutions to be developed in one step. The overall problem can be considered as a complex task, and can be more readily tackled by partitioning it into a number of sub-tasks together with an indication of how these sub-tasks interact with each other in order to solve the problem. This process of partitioning, when applied repeatedly, reduces the complexity of an individual section to a level where the problem can be more easily understood and specified.

The purpose of requirements analysis is to specify the requirements as opposed to designing the software components, the focus is on *what* the system must do and not on *how* this is achieved. The result of applying this technique is the generation of a model of the problem to be addressed.

As information flows through a system it is transformed: the system can accept a variety of different forms of input and applies hardware, software and human interaction in order to transform the input into output. Structured analysis is a technique for modelling the flow and content of information by subdividing the overall task performed by the system into a series of individual processes. In the modelling process three different models are used:

- *Data Flow Diagrams* - These specify how data flows through the system and is being processed .

- ❑ *Control Flow Diagrams* - Many applications are time-dependent and process control information rather than data. Optional additional control flow diagrams specify flows of control information and control signals
- ❑ *State Transition Diagrams* - These describe the different states of a process and the transitions between states. They can be used during validation to ensure that control specifications are complete.

A process is the encapsulation of some requirements which perform a specific task. A software system can be represented as an information process and the overall function of the system can be graphically represented as a single process, with a number of inputs to the system from external entities and a number of outputs of the system. As depicted in the figure below, the process can be graphically represented as a bubble and the external entities as rectangular boxes. The process is connected to the external entities via arrows which represent data-flow as well as flow direction, thus denoting inputs and outputs.

The single process can be broken down into a number of smaller processes together with data-flows between them. Each of the processes thus identified can then be further broken down and so on. *Data Flow Diagrams* (DFD) are a graphical representation depicting data flows and processes which are applied to the data in the process of transformation from input to output, and may be used to represent the system at any level of abstraction. This means that it is possible to generate a hierarchy of data flow diagrams to depict the system at arbitrary levels of abstraction. The aim of the Requirements Model is to generate such a hierarchy.

The development of the Requirements Model proceeds top-down, from the most general abstraction to the most specific in a series of levels. By convention the diagrams are labelled, starting from level 0 to level n. Level 0 represents the system at the most general level. It is also known as the context diagram, representing the system as a single process showing its connections to external entities, as depicted in the figure below:

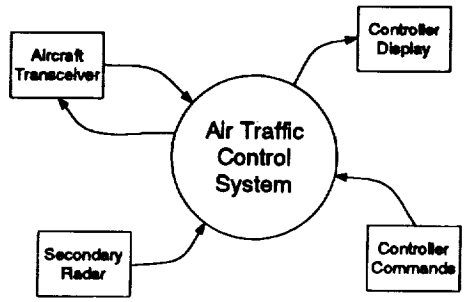

Figure 2: A Sample Context Diagram

In the process of breaking down individual processes it is essential to maintain the existing flows which connect to the parent process. The fundamental principle is that the connections are inherited from the parent process. Thus a check is carried out to ensure that the flows of the parent process are equivalent to those of the child processes, and is called balancing. In addition to inherited flows, new flows need be introduced to interconnect the child processes. All components depicted on the diagram need to be labelled (for reasons of clarity these labels have been omitted from the sample diagrams) and explained in separate process and flow descriptions which are entered in the *requirements dictionnary*. The figure below shows a sample level 1 diagram:

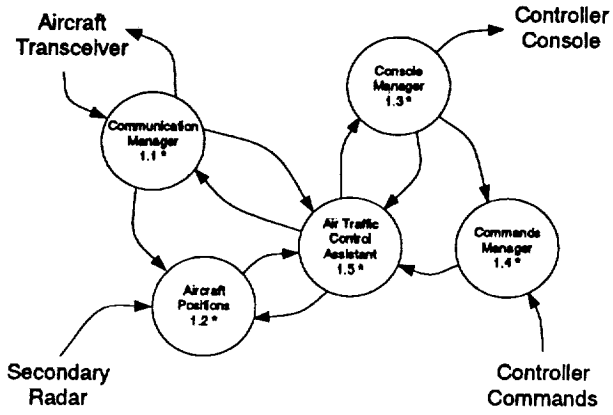

Figure 3: A Sample Level 1 DFD

Processes are labelled with a name and a unique nested numbering system indicating the level and parentage of the each process. There may be a need for data-stores to temporarily store data required for further

processing. These are denoted by two horizontal parallel lines with the datastore name in between, as well as arrows to and from them to indicate which processes modify or read the information contained in them.

Once a point is reached in the analysis where a process cannot be usefully decomposed further, it is specified by a *process specification* (PSPEC) which is readily implementable and describes how the inputs of the process are transformed into the outputs. Processes at this level are known as *primitive processes*. It may take the form of pseudo-code or a function to describe the algorithm used to carry out the transformation. At this point the decomposition stops for that process. The decomposition however continues for all other processes until a point is reached where all processes are primitive and specified by their respective PSPECs. Once this point is reached the hierarchical set of DFDs that have been produced for the system are complete. The PSPECS are then gathered in a separate section for process definitions.

There are a number of applications which are time-dependent and may process more control information than data. Real-time systems in particular interact with external entities on a time frame that is dictated externally and this places a number of important constraints on the demands of the system specification technique. In particular such a technique has to allow the analyst to represent control flow and control processing as well as the usual data flow and processing, which is achieved bt Consensus.

In order to distinguish normal data flows and processes from control processing, some additional notation is required. Thus control flows are used to describe the flow of control information and continuing the conventions established for data flow and processing, control flows are denoted by a dashed line as opposed to a solid line. A process that handles only control flows is called a *control process*, again denoted by a bubble with a dashed line, and associated with each will be a *control specification* (CSPEC). The appropriate descriptions of control processes and flows needs to be entered in the *requirements dictionnary* (and the associated CSPECs entered).

Control flows and processes are crucial to the behaviour of a system, and there are two ways in which the appropriate functional behaviour can be defined. A *Program Activation Table* shows the different permutations of states and the actions taken in each case to ensure that all possible states have been covered and receive the appropriate action. Alternatively, *State Transition Diagrams* (STD) can be used to give a behavioural model of the control process which shows the different states the system may be in and the connection between states. The latter is of particular significance for the purpose of verification. The use of STD (otherwise known as finite state machines) can be extremely powerful, but this observation is not widely appreciated, especially in current approaches for the development of KBS.

In some applications system inputs must be received at a certain rate and sytem outputs generated within a given time. These requirements are termed *timing requirements* and are associated with specific flows and processes. There are a variety of potential timing requirements from less strict response times to user input for interactive interfaces, to strict output rates of real-time and safety critical systems. An indication should therefore be given as to whether or not particular timing requirements are critical. Timing specifications connect specific input and output events. As a result there needs to be a list of input events and the respective output events and the timing relations between them.

3.2 The Architecture Model

The Architecture Model is developed in conjunction with the Requirements Model, describing and defining the system in terms of the implementation platform. The process of functional decomposition should produce collections of functions, which can be grouped together into separate processes which can work concurrently, and collaborate by communication towards the overall goal of the system.

The Consensus method specialises in the development of parallel KBS architectures, where a collection of interconnected KBA

collaborate to meet the requirements of the application. These independent KBA communicate via pre-defined data channels and co-operate towards the system's goals.

The purpose of the Architecture Model is both to translate and map the requirements into actual system components, as well as to guide the requirements analysis to take into account potential constraints imposed by the target platform. It may be possible to identify groups of potentially concurrent functions at different degrees of resolution, and as the decision on this matter is more dependent on implementation constraints it needs to be based on the system Architecture Model. The aim is to achieve a specification which fulfils the requirements and is implementable. There are three important aspects provided by the Architecture Model:

- an architecture paradigm for large, parallel KBS
- a mapping of requirements into design components
- feedback to help reject unfeasible requirements

The construction of large KBS as a single, monolithic system presents problems for a number of reasons. The execution of KBS is not strictly procedural by nature, and an order of processing may have to be enforced to meet requirements. Conflicts may arise when different knowledge sources or rules want to execute on the same data, thus interfering with each other. The risk of these situations occurring becomes more acute as systems increase in size. In addition, large monolithic systems may be too slow to meet real-time performance requirements. An approach which leads to the development of modular systems, which can execute concurrently on a distributed platform, and which can deal with interference problems is therefore preferable. The Consensus method builds systems as collections of connected knowledge based agents (KBA), each with the same basic structure containing:

- a local blackboard
- local knowledge sources
- a local controller
- communication channels

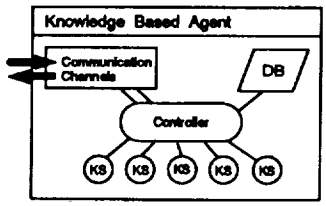

Figure 4: A Knowledge Based Agent

The local blackboard contains data and results of processing. Access to it is restricted to the local knowledge sources and controlled by the local controller. Each KBA has a set of local knowledge sources which perform the tasks assigned to that KBA. They operate solely on the information contained in the local blackboard and may request the local controller to communicate with other KBA to transfer data. The local controller is in complete control of processing at a local level within the agent. It schedules the execution of local knowledge sources which operate on the local blackboard and initiates communication with other KBA on behalf of its knowledge sources as well as receiving messages and posting them to the local blackboard.

The KBA which make up a particular system are connected by pre-defined communication channels. A number of important constraints are imposed on the communication mechanism in order to preserve the benefits of modularity and reduce interference between individual KBA. Thus communication with other KBA is managed by the local controller on behalf of its knowledge sources, and no other contact between KBA is allowed. These constraints are designed to help ensure consistency between KBA and in avoiding deadlock.

There are a number of important properties of this architecture that are relevant to the aims of Consensus:

- **Modularity**: Designing systems as collections of autonomous agents encourages a partitioning of the system into clearly defined system modules.
- **Information Hiding**: hiding the details and data of each KBA from the view of other KBA avoids uninten-

tional interference by one KBA with another.

- **Deadlock Avoidance:** insisting that the communication between KBA occurs indirectly, avoids direct interference between KBA, their data and control and is designed to reduce the risk of dead-lock. Although it is difficult to completely rule out deadlock, tests can be carried out to spot circular dependencies.
- **Traceability:** A direct mapping from general to specific requirements, and a direct mapping from specific requirements to system design components and code ensure traceability of requirements.
- **Concurrency:** Developing systems with an agent based architecture is an effective approach for building systems as sets of independent cooperating modules which are inherently concurrent and parallel.

Developing the Architecture Model

In accordance with the framework provided by the system specification, there is a close correlation between architecture and requirements, where processes in the requirement model will have corresponding entities in the Architecture Model and vice-versa. It is the purpose of the system Architecture Model to:

- identify the set of agents which form the system
- define the information flow between the agents in the system
- specify the channels on which the information flows

In the process of developing the system architecture the decomposition and specifications produced by the system requirements are used as the basis to develop the design of the system. The requirements which specify what the system is to do need to be transformed into a viable design. As KBA are more coarse grained than the primitive processes identified and specified in the Requirements Model, one needs to group components identified there into KBA which can operate relatively independently and in parallel. This process includes the identification of KBA, mapping requirements to individual KBA and specifying the interfaces.

As the Requirements Model is developed so the development of the Architecture Model follows one step behind to try identify a suitable system design which fulfills the requirements while taking into account external constraints. The second role of the Architecture Model is to force the develoment of the Requirements to take account of limitations in the specification process.

Starting from a global view of the system in a "first-cut" approach, the design needs to be refined and finalised to a point where ultimately it is very close to source code. The process of design of the system architecture takes the decomposition of DFDs from the Requirements Model and performs a transform analysis.

One has to determine properties of the DFDs which are important to a candidate system structure with a view at arriving at a suitable architecture that meets both functional requirements and additional constraints imposed by the implementation platform (where the platform includes both the target hardware and software). The following steps help drive a design:

1) **Review of the DFDs:** The review looks at the DFDs with the purpose of exploring the fundamental structure and characteristics of the system. It is important to familiarise oneself with the Requirements Model (especially if that is developed by another team) and to understand the structure of the requirements which may influence the structure of the design, and secondly, to consider the properties of the target platform in order to try to identify requirements or their decomposition which could create problems. If there are problems, the affected components need to be re-developed either at the present level or at the parent level to produce a new decomposition. The earlier problems are identified the easier it is to redevelop the affected components

2) **Analysis of Flow Characteristics:** In the next step the DFDs are examined to determine whether they display *trans-*

form or *transaction* characteristics by the following definition. If a DFD shows a chain of transformations where incoming information is gradually modified and transformed into an outgoing flow, the diagram is said to display a transform characteristic. Alternatively, transaction flows are characterised by processes which have only few inputs but a range of action paths to different processes.

3) **Preliminary Grouping:** If the DFD displays transform characteristics, the transformation centre needs to be isolated from the incoming and outgoing flows. Three main parts are thus identified: *inflow transforms*, the *transaction centre* and the *outflow transforms* are candidates for implementation as separate KBA, producing an incoming flow controller which co-ordinates the receipt of incoming data, a transform controller supervising the main transformation of data, and an outgoing controller co-ordinating the generation of the output information.

Alternatively, if the DFD displays transaction characteristics, the transaction centre needs to be identified and is characterised by a number of action paths that flow radially from it. The reception path leading to the transaction needs to be isolated and so do the separate action paths leading out from the transaction centre. Each of the reception and despatch or action paths need to be re-examined to determine whether they have, themselves, transform or transaction characteristics. Each of these components identified are candidates for implementation as separate KBA.

4) **Review of Grouping:** The preliminary architecture is analysed, to consider whether the proposed KBA are sufficiently large and complex to be viable as separate KBA. The scope of the proposed KBA should be reviewed to reduce coupling and increase cohesion:

Grouping should try to minimise the number of communication channels; processes which are highly connected should be in the same KBA. High fan-out from the outgoing flows of a KBA to a large number of other KBA should be avoided.

KBA should have a clear purpose and function and candidates lacking in this respect may need to be redefined, or may point to problems in the Requirements Model.

The candidate KBA identified in the preliminary grouping step need to be analysed concerning their size and complexity, to determine whether they are rich enough to be implemented as separate KBA or whether they need to be subdivided. This decision can be taken once the next level decomposition is known and the next iteration of the design process is carried out.

5) **Overall Review of Design:** The emphasis in the initial architecture specification stages is on analysis of the requirements from an implementation perspective, to ensure that the requirements definition does not contravene constraints imposed by the target platform. In later stages candidate KBA are identified, and finalised in an architecture diagram once the decomposition of the requirements is complete. During the review, one needs to decide whether previously identified KBA are viable on their own or whether they need to be further divided. In addition, the mapping of the processes from the DFDs supported by the requirements dictionary, needs to be specified in the architecture dictionnary. Architecture considerations may also influence the next level of decomposition of the requirements once external constraints become apparent.

Once the transform and transaction analysis has successfully been completed and the KBA have all been identified, the architecture diagrams can be finalised. At this point the additional documentation known as the *architecture dictionary* needs to be completed and contains the following:

- a definition of each architecture component depicted on the diagrams
- a definition of each architecture flow connecting KBA

- a definition of each architecture flow to external entities
- a narrative description of each KBA

The purpose of the narrative is to give a brief description of the purpose and functionality of each KBA. In addition a precise statement is required to associate all the components of the Requirements Model inherited by the KBA. This is an important step to allow verification as well as helping during maintenance. The local data structures need to be defined. The precise interfaces and communication mechanisms also need to be specified for the communication channels between KBA to allow the separate development of KBA by different teams. Finally, attention needs to be given to critical functions to ensure that they will not cause "bottle-necks" in the implemented system, thus limiting the overall performance of the system especially if there are real-time constraints.

4. Decision Support for ATC

The Consensus project has developed a test application which implements a decision support system, designed to help air traffic controllers in their task of safely controlling air traffic in their sector (based on [BEL88]). The system, known as the ATC Workstation, specialises in en- route air traffic control, i.e. air traffic control for sectors which do not contain aerodromes and where aircraft will pass through on their way to their final destination. The system also includes a simulator to simulate air traffic for testing and training purposes.

The controller workstation provides an integrated set of tools to support the en-route controller. These include the following:

- A **Predictor 2.1** which, given the current aircraft position and the current flight plan, predicts the courses of aircraft and warns of potential conflicts.
- The **Wotifer 2.2** which enables the controller to plan new routes for aircraft which need to be re-routed, while assessing the consequences of candidate new plans in the light of the current air traffic situation.
- A **Communicator 2.4** to provide electronic communication link between the controller and aircraft under his control.
- A **Monitor 2.5** to check the progress of aircraft, to warn the controller about aircraft which deviate from their flight plans or behave abnormally.
- The **Man Machine Interface 2.6** which handles the user interface relating information from the separate tools to the controller and directs the controller input to the appropriate tools.

The system operates an ATC workstation and, apart from the decision support functions, emulates a typical working environment.

Figure 5 shows the top level decomposition of the Requirements Model for the ATC workstation, followed by the further decomposition of the Wotifer in Figure 6.

The basic functions of the Wotifer is to recommend possible routes to the controller, to check the feasibility of candidate routes and to allow the controller to modify them at leisure and finally to accept one of them as the new flight plan for a particular aircraft.

Supposing a wotif is requested for an aircraft, the Wotifer goes through a number of steps. This involves the collection of all routes from present position to the required destination and pruning all plans the aircraft is not capable of following. Checking the remaining plans for compatibility with current air traffic situation, one has to select the best routes and to give them to the controller to select and/or modify. It may then be useful to store routes which are amended by the controller for future reference and obviously to file the route selected by the controller as the new flight plan for the aircraft.

The decomposition of the Wotifer was based on the sequence of steps required to carry

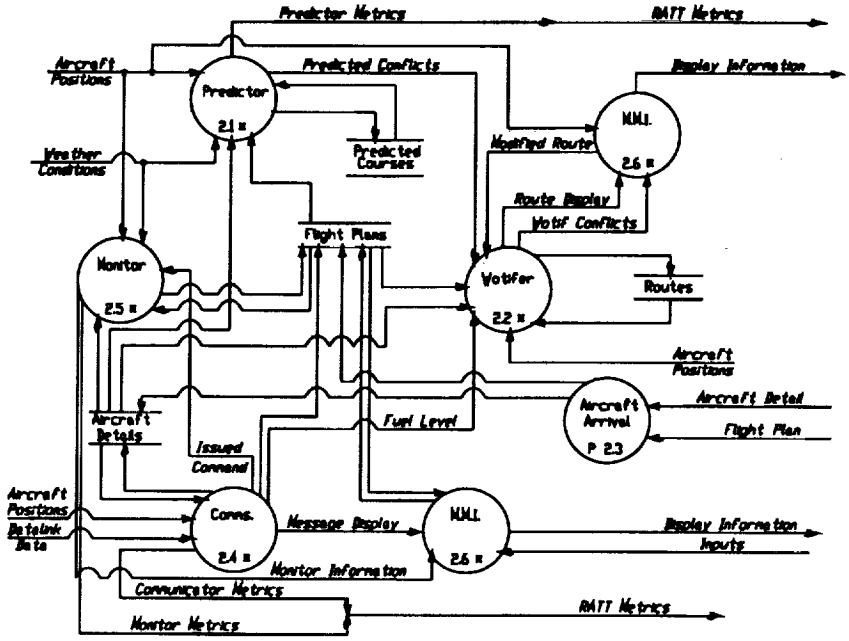

Figure 5: DFD Level 1 Decomposition of the ATC Workstation

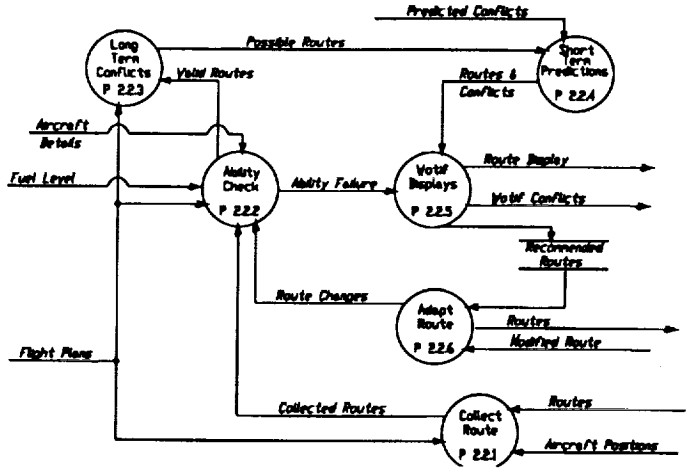

Figure 6: DFD Level 2 Decomposition for the Wotifer 2.2

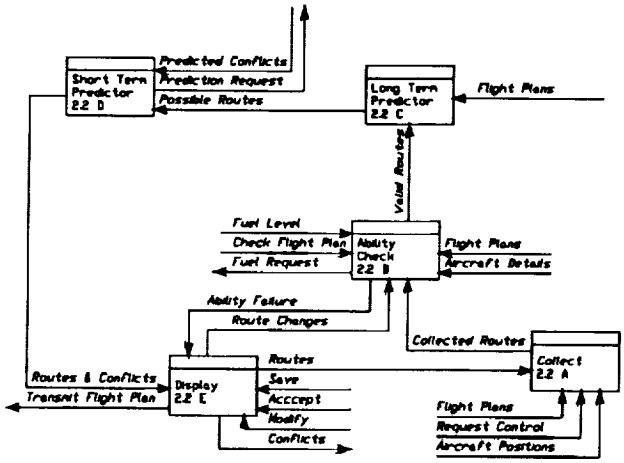

Figure 7: KBA Design for the Wotifer 2.2

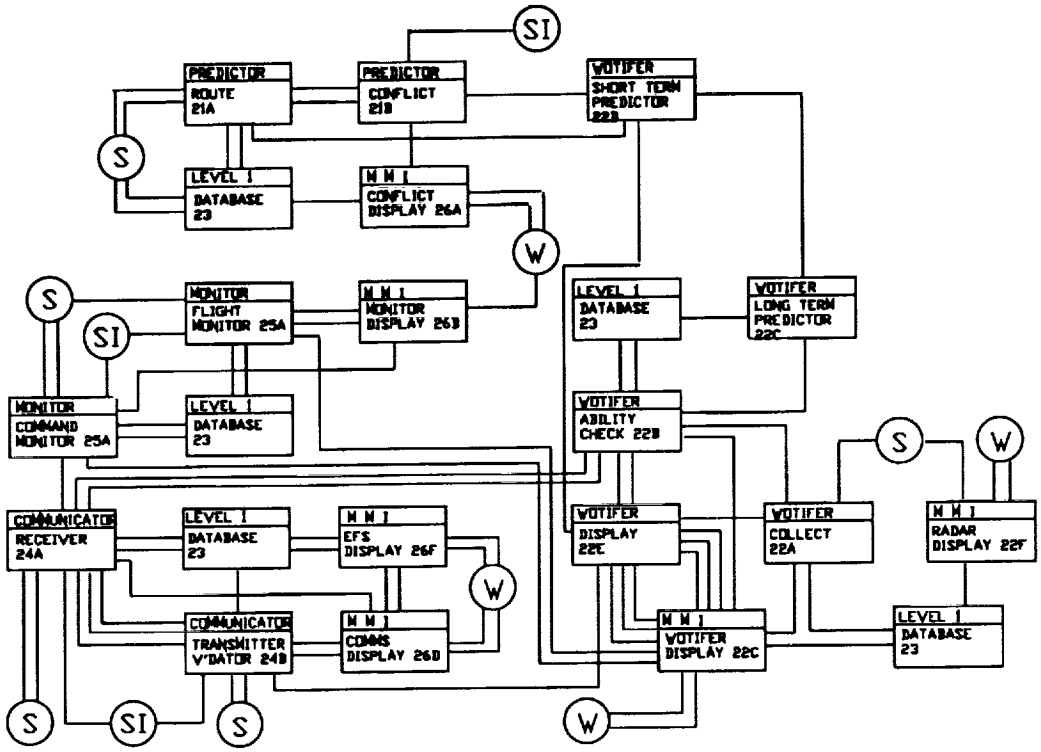

Figure 8: KBA Architecture of the ATC Workstation

out the task of planning and replanning of routes for given aircraft. The different processes depicted represent the major steps in that process.

This results in the following processes specified in the Level 2 decomposition shown in figure 6:

- **Collect Route 2.2.1** identifies all possible routes which would take a specified aircraft from its current position to the point where it wishes to leave the sector.
- **Ability Check 2.2.2** checks that the aircraft in question can perform the manoeuvres required.
- **Long Term Conflicts 2.2.3** identifies the more obvious potential conflicts that would be caused if the route in question was adopted as the aircraft's new flight plan.
- **Short Term Predictions 2.2.4**, in conjunction with the Predictor calculates the precise conflict points for the next 20 minutes.
- **Wotif Displays 2.2.5** calculates characteristics about a route to be presented to the controller and sends them to the MMI.
- **Adapt Route 2.2.6** deals with the response from the controller and deals with modifying, accepting and saving a route.

In the identification of KBA from the decomposition of the System Requirements, one aim is to exploit parallelism where ever possible. Thus, if a process can perform an action at the same time as another, this may be a reason for not joining them into a single KBA. In this case, the routes are processed sequentially, and it is possible for the Collect Route 2.2.1 process to collect a route and pass it onto the checking process Ability Check 2.2.2 which can then pass it onto the Long Term Conflicts 2.2.3 process which, in turn, can pass it on to the Short Term Prediction 2.2.4 process.

The processes can be pipelined to operate in parallel and in this case it meant that they were kept as separate KBA. The other two processes only process information when all routes have been processed. They work on the same information (Recommended Routes) but at different times, and because of the information sharing and asynchronous processing the processes were joined into a single KBA.

One problem that needed to be overcome was to determine what to do with the datastore of possible routes (Routes), which are used by the collecting process but updated by the modify process. One could have used a separate KBA, but the information is not used in many different places and does not change a great deal. It was the fact that the information does not change much which led to the decision to put the datastore together with the collect process. This led to five separate KBA which could work in parallel.

In Figure 7 the overall architecture of the ATC Workstation is depicted (note that S = Simulator, SI = Simulator Interface and W = Windowing System), thus combining the Wotifer Architecture with the other components of the system:

To conclude, the ATC System, including the simulator has a number of interesting statistics which demonstrate its size and complexity:

- 29 individual UNIX processes (including the Simulator and a C front end)
- networked on 3 to 13 workstations
- communicating via 174 socket connections
- 350 class definitions, 450 rules and 400 procedures
- about 3.5 man years of effort

This shows the ATC system to be a substantial application. The Consensus method gained considerably from this realistic test application by ensuring that the provisions capable of dealing with systems of this size and complexity, as well as giving valuable input for the generation of suitable development guidelines in the areas of the architecture of knowledge based components, hardware and software platform considerations, modularity and logical partitioning of the System Requirements and Architecture, considerations of concurrency, distribution and performance, as well as performance considerations and the handling of datastores.

5. Conclusion

The Consensus method fills a gap in the current field of software engineering, providing a comprehensive and integrated method for the development of large agent-based systems. As the Air Traffic Control application shows, agent-based systems can be successfully applied to systems which are relevant to current needs. The method covers the whole process from the statement of the user's need, to a detailed specification and through to implementation. The gradual progression from basic user requirements to a more refined specification with the help of a diagrammatic approach allows the customer to verify their requirements. The direct mapping from detailed requirements to system architecture and process specification also facilitates validation. There is a strong emphasis on sound software engineering principles in the system development process.

The application of Air Traffic Control is exemplary for a range of possible applications in the domain of aerospace and the experience gained by the project shows that Agent-Based Systems have considerable potential to provide solutions in this area.

There are a number of reasons for using distributed systems of this kind, one of which is the fact that it is difficult to conceive of using mainframe computere on board an air- or space-craft. Given high processing requirements and the need to keep weight to a minimum, the use of a number of networked processors would therefore apear to be much more appealing. Considering that systems on board, say, a shuttle have a number of distinct functions to fulfill, such as (amongst others) manoeuvering the craft, life-support systems, communication with ground-based systems and aiding in scien-

tific tasks, it seems sensible to divide a system to combine these functions into entirely separate sub-systems which could run on different processors and communicate with each other as and when necessary. This could be combined with a separate control system to supervise the proper functioning of these separate sub-systems, and which would take appropriate action when problems areise. Given the need to avoid malfunctioning at all cost, such a distributed system could also be made to be more resilient than a monolithic system as the malfunctioning of a sub-system would leave the other sub-systems still operative and enable the control system to take corrective action to re-establish full functionality. In addition, given stringent real-time requirements the lack of performance of one sub-system need not necessarily affect the performance of other sub-systems.

Bibliography

[ACK92] A Anjewierden, B Wielinga & N Shadbolt: Supporting Knowledge Acquisition: The ACKnowledge Project, ESPRIT-92 Knowledge Engineering, (Eds. L Steels & B Lepape)

[BOE88] Boehm B W, A Spiral Model of Software Development and Enhancement, IEEE, May 1988

[CRA89] Craig I D, The Cassandra Architecture, Distributed Control in a Blackboard System. Ellis Horwood, 1987, ISBN 0-7458-0579-5

COU91] Coulson I.C. et al., "Design Document D7", Consensus Report No. CNS-BAE-WIT-3.1-INT-IDC-056A.

[JOH91] Johnson K & Slade A, ATC Design Document, Wotifer 2.2, August 1991, Consensus Doc No. CNS-DUR-DCS-3.1-INT-KJ-052A

[JOH92] Johnson K. & Slade A, "Consensus Methodology Standards", Consensus Report No. CNS-DUR-CSC-6-INT-KJ-083B.

[HAT88] Hatley D & Pirbhai I, Strategies for Real-time System Specification, Dorset House, 1988

[KAD89] Taylor R M (Ed.), System Evolution - Principles and Methods, KBS Centre of Touche Ross Management Consultants, London, December 1989

[STA87] The STARTS Guide to methods and software tools for the construction of large real-time systems, NCC Publications, Hobbs Southampton, 1987

[FJE92] Fjellheim R, Pettersen T & Chriastoffersen B, REAKT Application Methodology Overview, ESPRIT-IIP5146 REAKT, Computas Expert Systems A.S, October 1992, CX Doc.# CXN92125

[ROY70] Royce W, Managing the Development of Large Software Systems: Concepts and Techniques, Proceedings of WestCon, August 1970, (Chapter 3)