The SIMTEC Simulation Framework

by D. F. BUCKLE, B.Sc., M.Sc.
EASAMS Defence Consultancy

Dave Buckle is a consultant with 15 years' experience at EASAMS. With a Mathematical Physics background, his career at EASAMS began in the Operational Analysis (OA) field, particularly with regard to the modelling and study of radar and electronic warfare systems. Over the last ten years, however, he has become increasingly interested in the application of OO techniques to simplify program structures and achieve software reuse. The combination of experience in OA modelling with OO analysis and design has culminated in his major role in the development of the SIMTEC architecture. (E-mail: dave.buckle@gecm.com)

SIMTEC† is a large C++ object oriented (OO) simulation framework developed by EASAMS Defence Consultancy and sponsored by Weapons Sector of DERA Farnborough. Although it was developed to meet the sponsor's requirement to study electronic combat in particular, SIMTEC is a general-purpose simulation tool capable of addressing a wide range of scenarios. This paper provides a short summary of SIMTEC and its potential benefits to its Users and Developers.

The paper makes two main points:

- SIMTEC is a large model that can represent many different types of interaction between military units in simulated battlespaces at a variety of fidelity levels; and
- SIMTEC is an initiative in software re-use, with several innovative techniques being applied so that the software can be used to produce a wide range of new applications – and to ensure that each new application can work with all the others.

Thus, SIMTEC provides Users with immediately useful functionality and is also a major enabling capability for Developers wishing to produce new simulations.

The paper explores the above two points by summarizing the current functionality of SIMTEC and giving an overview of the techniques used to achieve reusability. The paper argues that, in order to achieve the full benefits of reusability, it is necessary to establish a pool of Users and Developers who employ the same framework, and that framework must combine the benefits of:

- a library of reusable components;
- 'plug-and-play' interfaces* to co-ordinate the operations of models constructed from these components;
- techniques to permit models to be adapted by data configuration wherever possible; and
- the use of commercial off-the-shelf (COTS) tools.

The SIMTEC approach is compared with other techniques, such as the use of Distributed Interactive Simulation (DIS) and High-Level Architecture (HLA) initiatives, that serve to link separately-developed simulations together. It is suggested that, with the right techniques, the use of a single framework, such as SIMTEC, and the DIS/HLA approach are complementary.

† SIMTEC is the **SIM**ULATION **T**ool for **E**lectronic **C**ombat

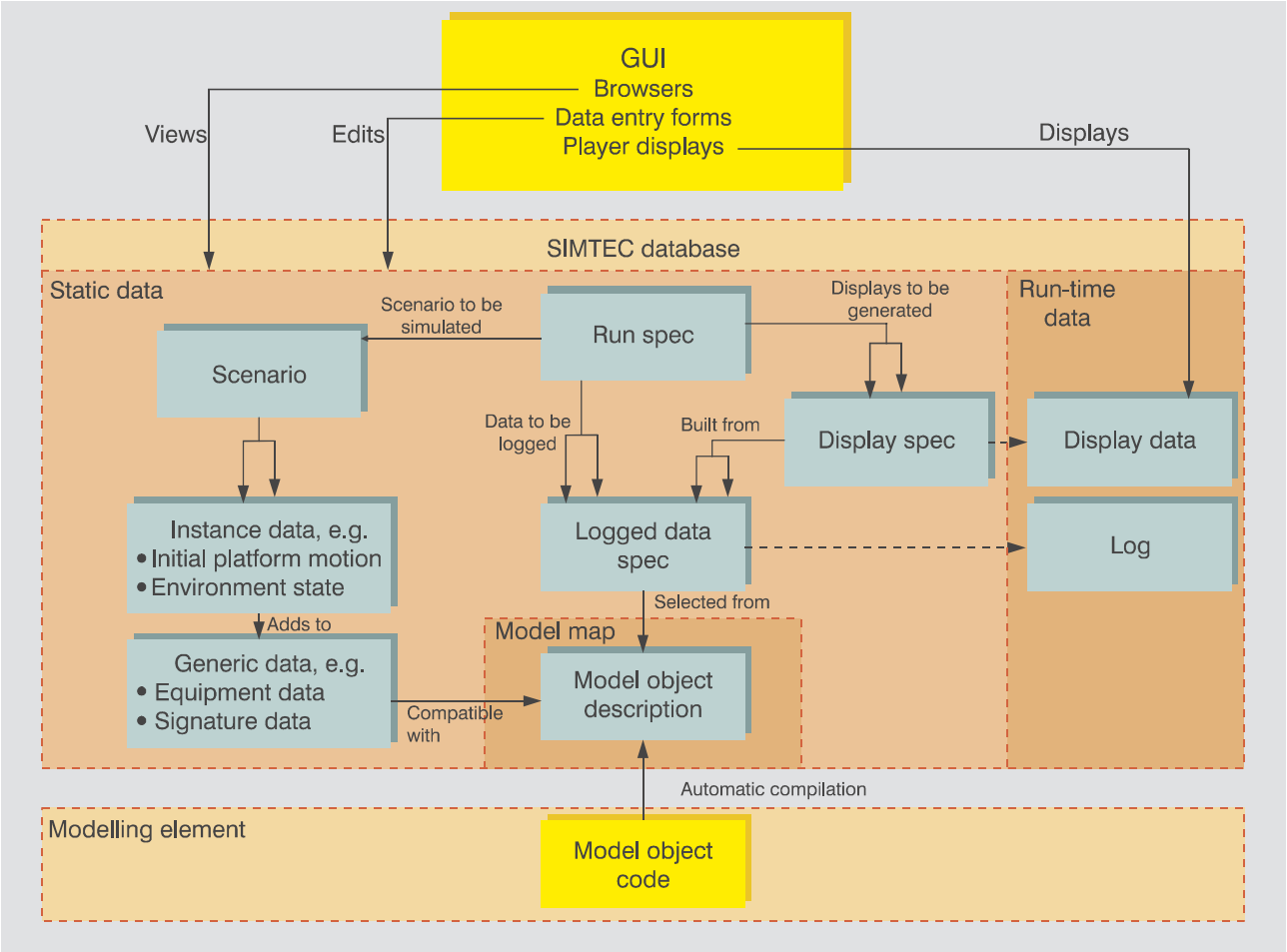
* 'Plug-and-play' interfaces are defined protocols that allow different modules or programs to work together whilst being treated as 'black boxes'.

Functionality of SIMTEC

The SIMTEC framework is divided into two elements: the 'User Environment' and the 'Modelling'. The User Environment provides an integrated system for defining, executing and analysing simulation runs (implemented by the Modelling Element).

SIMTEC models are highly data configurable and the User Environment supports this approach. As shown in fig. 1, it is based around a central 'SIMTEC Database' that stores:

- definitions of all the 'generic data' needed to construct one or many scenarios – for example, definition of platform components, such as particular types of equipment and electromagnetic signatures, the configuration of these components into complete platform types, and so on;
- the 'instance data' needed for individual scenarios – for example, the environmental parameters, plus the numbers and types of different platforms, together with their initial motions;



1 ***The SIMTEC User Environment***

- the run-parameters – for example, lists of the variables to be logged during a simulation run and a description of whether single or Monte-Carlo runs are to be performed; and
- Player Displays – the User Environment supports simple, but very configurable displays, whose specifications are stored in the SIMTEC Database.

The User Environment supplies a Graphical User Interface (GUI) that provides the above facilities in a windows format consisting of data entry forms, browsers and 2-D / 3-D Player Displays. The User Environment automatically configures itself to the code in the Modelling Element, generating the required database layout plus data entry forms, browsers and displays compatible with this code.

The User Environment performs the automatic configuration by analysing the code of the C++ objects in the Modelling Element to compile a 'Model Map'. This contains data descriptions of all the modelling objects, including the name of each object, a list of its routines and the input/output parameters of those routines. As shown in fig. 1, all the other elements of the User Interface ultimately link to the Model Map, thus permitting the User Environment to be completely generic, yet sensitive to the structure of the code in the Modelling Element.

Within the Modelling Element, SIMTEC provides a sophisticated Battlespace Model representing the arena in which scenarios occur. A terrain database is used to represent both the elevation of the land and the different culture types it consists of, each of which can be described in terms of its back-scatter and forward scatter properties. (The sea is considered a particular 'culture' type with a wave structure elevation.) The atmosphere is modelled in the way in which it affects electromagnetic propagation and induces drag, and a variety of atmosphere profiles are available. In addition, weather volumes, such as clouds, and rain belts moving in the atmosphere can be represented. The Battlespace Model is coupled to a powerful, ray-path based representation of electromagnetic propagation, that includes attenuation, reflection, refraction and multipath effects.

The SIMTEC framework currently supports a single (although very powerful) application called the *integrated platform* model. This model is a generalized representation of a military platform consisting of a *superstructure* upon which items of *equipment* can be located and a mechanism for controlling these items of *equipment* so that the platform as a whole behaves in an integrated fashion.

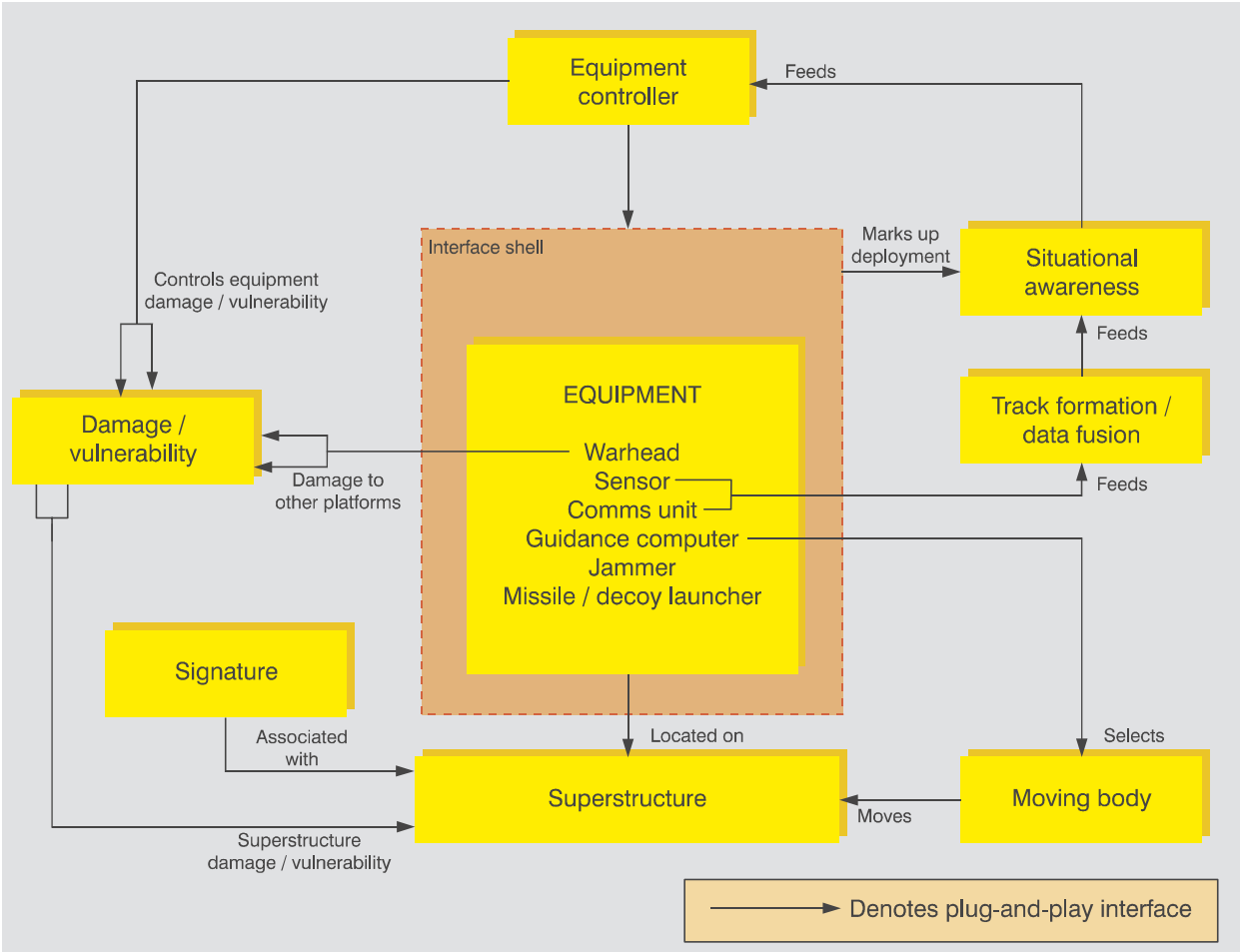
The *integrated platform* model is OO in nature. That is, many different **instances** of the **same** *integrated platform* model can be created at run-time to represent the different platforms, such as ships, aircraft, surface-to-air missile (SAM) units etc., that interact in a simulated battlespace.

In view of the above, the different instances of the *integrated platform* model differ from one another only in their **data**. However, the *integrated platform* model does exhibit a high degree of data configurability in the sense that the **User** (and not the coder) can represent fundamentally different **types** of platform, such as the ships, aircraft and SAMs mentioned above, by defining different data configurations of the common *integrated platform* model.

Thus, to represent a particular scenario, the User sets up generic data sets defining the different types of platform to be studied. One or more sets of instance data are then set-up for each generic data set to define the initial motions of the individual platforms participating in the simulation run. The User can data configure different types of platform because the *integrated platform* has a sophisticated internal architecture consisting of 'components' linked by plug-and-play interfaces.

The architecture of the *integrated platform* model is shown in fig. 2. It consists of a *superstructure* component representing the physical 'shell' of the platform upon which other components can be associated via the supplied plug-and-play interfaces, that is:

- A *signature* component that represents the way in which the platform reflects and radiates energy. A flexible set of *signature* components is provided, including



2 ***Architecture of the integrated platform model***

multi-source radar and optical/infra-red representations. One platform can have several *signatures* located on it. A plug-and-play interface allows the location to be defined in terms of a mounting point that gives each *signature* a definite position and orientation (that is – yaw, pitch and roll) offset from the *superstructure* centroid.

- *Kill mechanisms* and *vulnerability* components, which represent the ability of the *superstructure* to inflict damage on other platforms and, in turn, suffer damage. The current suite of components of this type allows both physical impact and explosion interactions to be represented. The supplied plug-and-play interface allows the User to indicate the lethal interactions in which the platform as a whole can participate by associating the appropriate *kill mechanisms* and *vulnerability* components with the superstructure. Another interface allows individual items of equipment to be knocked out or inflict damage (for example, warhead explosion).
- An *equipment* component represents an item of equipment located on the platform. A range of *equipment* components are available, including:
  - *sensors*, for detecting other platforms, which may be radar, electro-optic (EO), infra-red (IR) or electronic support measure (ESM) devices;
  - *weapons systems*, for engaging platforms such as *missile launchers*, *decoy launchers*, *jammers* and *warheads*;
  - *comms units*, for, relaying messages to other (friendly) platforms;
  - *guidance computers*, for controlling the motion of own platform, which can presently manage either predefined – for example, multi-waypoint motion – or implement missile guidance laws, such as proportional navigation or command-to-line-of-sight.
- A plug-and-play interface similar to that used for *signatures* allows equipment components to be located on mounting points.
- Each platform has one *track formation* component and one *situational awareness* component. The *track formation* component takes the output from the *sensor* type *equipments* and use this to form tracks on the targets in the platform's vicinity. In this case, the plug-and-play interface requires that all *sensors* report in a common format. The supplied *track formation* process can also use data fusion to merge *tracks* received from other platforms (via *comms units*). Note that the target *tracks* are sophisticated descriptions holding the perceived positions and motions of the detected targets and the errors in these data plus additional information such as the assumed target classifications, allegiances, emission characteristics etc. The *situational awareness* consists of the target *tracks* marked-up with additional information about the way the platform has deployed its *equipments* against those *tracks* as discussed next.
- *Equipment controller* components exist in one-to-one correspondence with the *equipment* components. Each *equipment controller* operates by deploying the item of *equipment* it manages against one or several *tracks* or (alternatively just switching it on or off). The *equipment controllers* are data configured with rules set up by the User. These rules are based on filtering the *tracks* in the *situational awareness* to determine potential engagements and arranging the *tracks* passing the filter into priority order. The facilities provided to the User in setting these rules are very extensive, so that the filtering and ordering can be performed, based on a large number of criteria. The rules can also take into account the way other *equipments* are deployed, via the mark-ups in the *situational awareness*, so that co-ordinated behaviour can be produced. In addition, timing delays can be added to the *equipment controllers* to represent the latencies in real systems.

Thus, the User can configure a generic data set to define a particular type of platform by specifying its *superstructure*, *signatures* and *equipment* fits and setting up the rules to control its behaviour. It should also be noted that the *integrated platform* model is **recursive** in the sense that munitions deployed by platforms, such as missiles and decoys, can also be described by different data configurations of the *integrated platform* model. Thus, extremely complex scenarios can be simulated in SIMTEC by data configuring the single, generic *integrated platform* model.

Standard Software Reuse Techniques

SIMTEC is an initiative in software reuse. This is achieved by using all of the 'standard' reuse techniques in an integrated fashion (often using innovative technology) that capitalizes on the advantages of each technique whilst minimizing its disadvantages. The 'standard' software reuse techniques employed are:

- library of reusable modules,
- plug-and-play interfaces,
- data configurability, and
- COTS tools.

The idea behind the first technique is to develop a library of software modules that can be used in many different applications. This technique has two great advantages:

- it achieves **fine-grain** reuse – that is, the modules can consists of just a few tens or hundreds of lines; and
- it can be applied to any problem (at least when an OO approach is used).

There are, however, three disadvantages with a library approach:

- it takes power away from the User because it requires a Developer to link together the library modules;
- libraries produced by different vendors are usually mutually incompatible, and the Developer often has to 'buy-in' to one particular library and discard the others; and
- the approach does not guarantee that different models constructed from the same library modules are compatible with one other.

The provision of plug-and-play interfaces has complementary advantages and disadvantages. Its main disadvantages are that:

- it is not universally applicable – that is, plug-and-play works well only when there is some 'natural' division between components existing in the problem space (it is the job of the Developer to find these natural interfaces);
- it usually provides only **coarse grain** reuse – that is, the components it unites are often very large and, indeed, may contain significant areas of overlap; and
- it can introduce inflexibility and overheads (although these problems can be minimized by innovative approaches).

Plug-and-play interfaces do have advantages, however, in that they can:

- permit the products of different vendors to be linked – that is, they can get round the 'buy-in' problem;
- provide the discipline necessary to ensure that models developed from different library components can work together; and
- assist data configurability and the use of COTS. That is, the components linked by the interfaces can be separately configured and/or implemented by COTS packages.

The main advantages of data configurability in its own right is that it:

- puts power in the hands of the User, as opposed to the Developer; and
- it can be much more efficient than hard-coding in tailoring available functionality to applications.

Its chief problems are that:

- the efficiency increase applies only in some circumstances, which must be carefully selected by the Developer; and
- it can also be quite a problem to equip the User with the tools required to perform data configuration in complex applications, although, as will be seen, this is another area where novel approaches can be used.

Perhaps the most direct way of addressing software reuse is to take one particular problem and provide a generic COTS product that solves it once and for all. This is a very good solution if it works, but historically has proved successful only with certain problems that lend themselves to this treatment. Use of COTS can also lead to the problems of 'buy-in' and the functionality of different COTS tools may overlap.

SIMTEC Design Principles Enabling Reuse

SIMTEC uses two main principles – an **OO toolkit approach** and the use of **smart noticeboard** plug-and-play interfaces – to implement the four standard software reuse techniques discussed above. These two principles enable all of the four standard techniques to be implemented in an integrated fashion that capitalizes on the advantages of each technique, whilst minimizing its disadvantages.

The OO paradigm is applied to the **whole** of SIMTEC – that is, the User Environment, the Battlespace Model, the plug-and-play interfaces and the *integrated platform* model application. A 'toolkit'

approach to OO is used whereby several rules of good practice are applied to ensure that the objects are (i) small code modules, typically of the order of 100 lines, and (ii) designed as 'servers' so they act like tools in a toolkit in that they offer services to other objects, their 'clients' but are unaware of the nature or structure of those clients.

Application of the toolkit approach causes the whole of SIMTEC to have the form of a large library of small toolkit objects. This is represented in fig. 3 using building blocks to represent the toolkit objects. The toolkit objects fall into a loose hierarchy with objects higher in the hierarchy using the services of ones lower down to build up increasingly sophisticated functionality, represented in fig. 3 by layers of building blocks put one on top of the other. Existing COTS tools are used where applicable to support the toolkit objects.

The OO toolkit approach turns one of the conventional problems with libraries, namely their limited applicability, on its head, because the entire program is a library! However, the other disadvantages, namely the need to 'buy-in' to the library and the problems of model incompatibility remain.

Plug-and-play interfaces are used to get around the incompatibility difficulties. With the toolkit approach, plug-and-play interfaces are implemented by objects like everything else. That is, there is no **structural** difference between the interfaces and the rest of the code. The interfaces exhibit themselves only as **usage patterns** within the interactions between objects – that is, the fact that some objects, the 'interfaces', are choke points in the interaction patterns. This removes one of the potential disadvantages with plug-and-play interfaces, namely their overheads, but does not, by itself, make the interfaces more flexible.

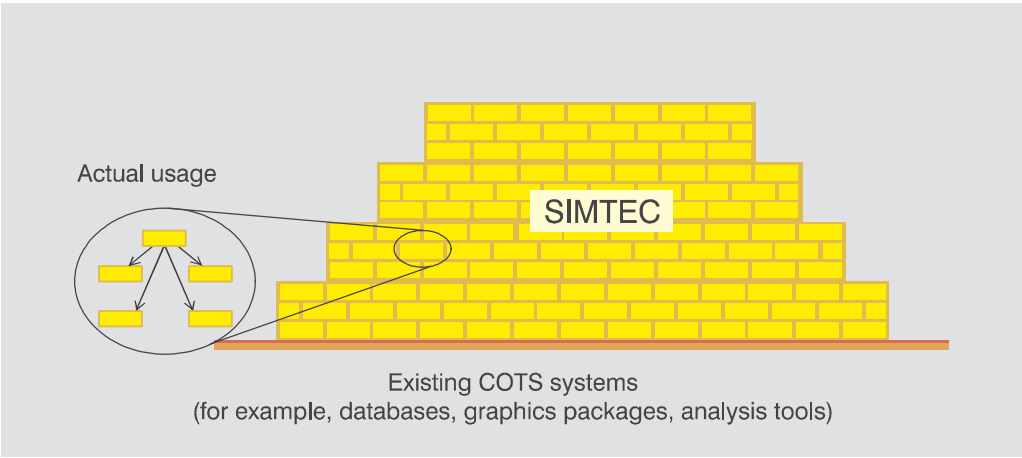
Although similar to any other code structure in SIMTEC, the plug-and-play interfaces are, of course, designed as such by the Developer. In SIMTEC a 'smart noticeboard' design pattern is used to improve the flexibility of the interfaces. This involves:

- making the interfaces globally-visible areas, so they become 'noticeboards'; and
- posting information on the noticeboards as objects – that is, entities with both data and functionality – so the noticeboards become 'smart'.

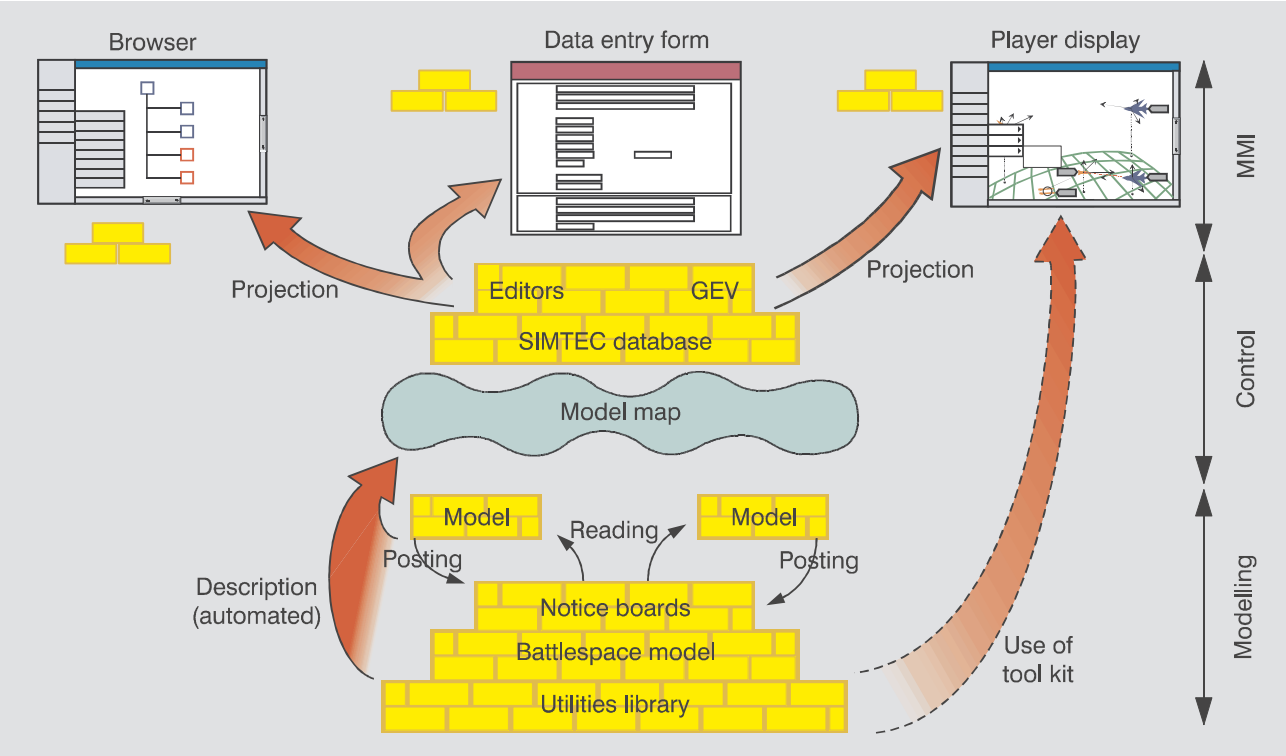
Posting objects on noticeboards improves flexibility because client objects reading the noticeboards can take as much or little of the information as they required (and in fact it is also possible to post multiple descriptions of the same phenomena at different fidelity levels).

Smart noticeboards do not, however, remove the need to choose carefully the areas where plug-and-play interfaces are applied. The interfaces used in SIMTEC are shown in fig. 4 and discussed below.

- Within the User Environment, a 'Projection' interface is used to separate out the functionality, or 'Control' element, from the screen display, or 'MMI' element. Here the information and actions provided by the Control are globally available and are then 'projected' onto the GUI. This has the advantage that as soon as new control functionality is added, in C++, new buttons and/or windows are automatically generated for display.
- The Model Map interface separates the Modelling Element from the User Environment. As previously explained, this map holds a complete data description of all the modelling objects. In addition, however, it also has the functionality to analyse relationships between objects – for



3 ***Library structure of SIMTEC***



4 ***Plug-and-play interfaces in SIMTEC***

example, the tree of related objects that comprise the integrated platform model – and invoke routines knowing only the name of the routine and the I/O parameters. This enables the User Environment to execute simulation runs by:

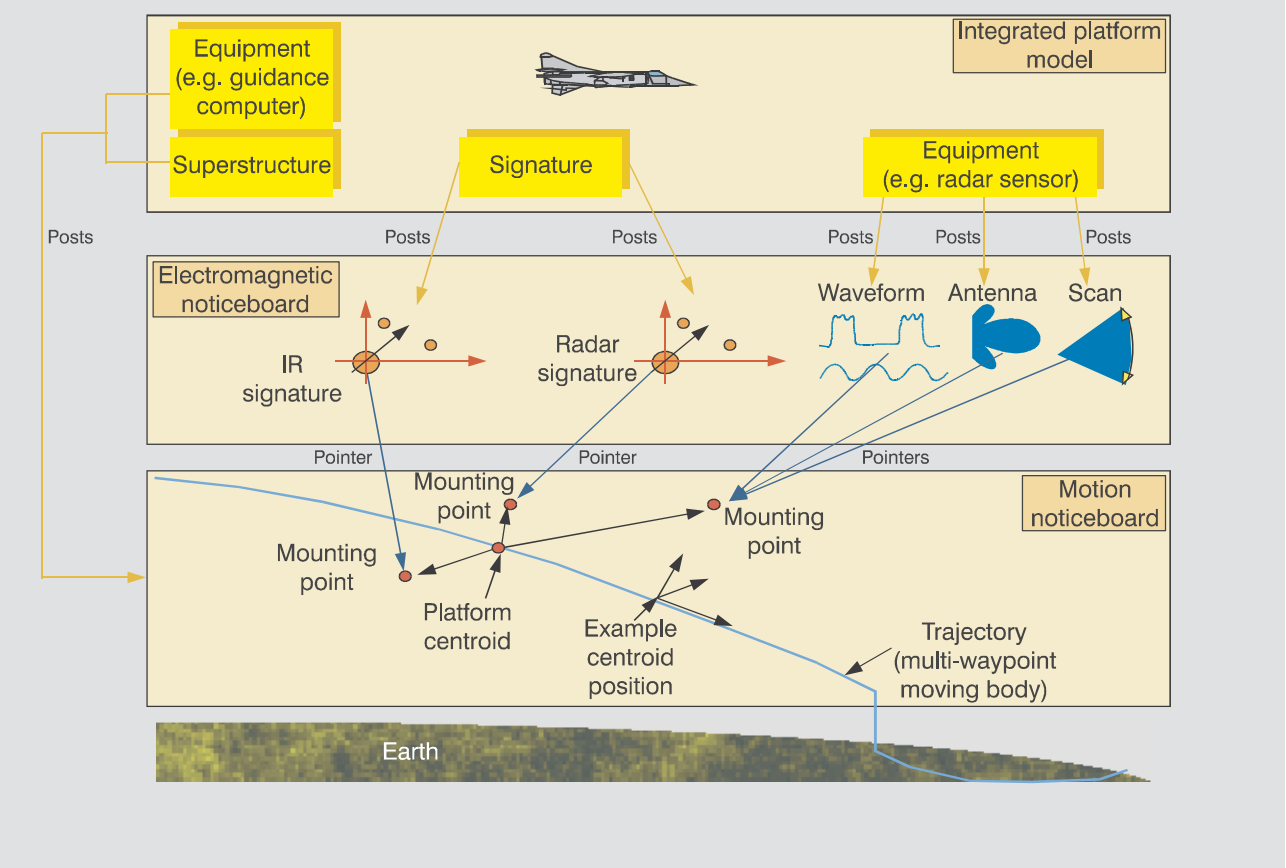
- invoking the necessary routines to set up the battlesapce environment and create all the platforms at the start of the scenario (using the description of the scenario stored in the SIMTEC database as shown in fig. 1);
- running the simulation by driving the event list to advance simulation time; and
- calling the relevant routines to log the data and drive the displays requested by the User.

- Within the modelling code, six smart noticeboards are used to handle inter-platform interactions covering:
  - platform motion,
  - electromagnetic interactions,
  - communications,
  - command and control,
  - kills and vulnerabilities, and
  - electronic warfare.
- In addition, smart noticeboards are used to implement the intra-platform plug-and-play interfaces within the integrated platform model application, as discussed earlier and shown in fig. 2.

It is not possible to discuss all the above interfaces in detail in this paper. However, fig. 5 gives an example of two very important cases, namely the way in which motion and electromagnetic interactions between platforms are described.

As shown in fig. 5, a SIMTEC model of a platform, such as the *integrated platform* discussed earlier, posts a description of the motion of the platform centroid as a *moving body* model. This describes motion as a function of time in six degrees of freedom (6 dof) and applies 'until further notice'.

The *moving body* description can be complex and contain discontinuities – for example, a multi-waypoint trajectory with sharp corners as in fig. 5. Because functionality as well as data are posted, however, objects reading the noticeboard (for example, other instances of *integrated platform*) can always obtain exact (6 dof) data at any point in simulation time that they require. In addition, the (6 dof) locations of the platform *mounting points* are also posted on the Motion Noticeboard, so reading objects know the precise position and orientation of all the *signatures* and *equipments* on the platform.



5 ***Examples of smart noticeboards***

As shown on fig. 5, some items of equipment can also emit electromagnetic radiation. These emissions are specified by objects describing the emitted waveforms, antenna patterns and scan patterns, all of which are functions of time. Thus, at any point any object reading the inter-platform noticeboards has access to a full description of the factors influencing the electromagnetic signals in the battlespace and can, as required, call up either simple 'cookie cutter' models of detection or access the detailed electromagnetic propagation representations in the supplied Battlespace Model.

Integrated Approach

The use of the toolkit approach and smart noticeboards is integrated with the use of COTS and data configurability so that the benefits of all the software reuse techniques reinforce one another. Existing COTS tools are used where appropriate, but some parts of SIMTEC can themselves be considered COTS products because they provide generic solutions, namely:

- the User Environment, which is completely generic because of its use of the Model Map smart noticeboard; and
- the Battlespace Model (shown in fig. 4), which represents the physical environment in which the interactions on the inter-platform noticeboards occur.

Data configurability is a natural outcome of the OO toolkit approach when coupled with a model-sensitive User Environment.

The way the OO paradigm unites data and functionality means that it possible to data configure **operations**. For example, the 'rules' used in the *integrated platform* models to form priority lists are actually algorithms for filtering and sorting *tracks*. Also, the OO mechanisms of 'inheritance' and 'polymorphism' allow many different objects with different behaviours to have the same interfaces. Thus, the User can choose between alternative behaviour patterns as part of data configuration. The toolkit approach causes the objects available for data configuration to be small and relatively self-contained, thus improving the granularity with which the data configurations can be performed. A model-sensitive User Environment is required, however, to enable the User to access this fine granularity.

Developing SIMTEC Models

Although SIMTEC models are highly data configurable, this does not, of course, obviate the need for code development. It is important to stress that, from the Developer's point of view, SIMTEC is a **clear box** facility designed to produce **fine grain** software reuse and assist the **internal structure** of the models to be produced, as well as providing interfaces to allow different models to interact.

Although the Developer has to write some code from scratch when producing the internal structure of a new model, it will often be the case that much of the required functionality can be obtained from the existing library of toolkit objects. The Developer is free to use these objects as required (that is, SIMTEC is 'clear box'). Also, as all the toolkit objects are small, primitive items of functionality, typically about 100 lines long, they can address the Developer's precise requirements (that is, the reuse is 'fine grain'). The Developer may also construct new toolkit objects that can then be added to the general library.

In addition, provided the new model uses the existing plug-and-play interfaces, it will be able to:

- use the COTS facilities of the SIMTEC framework, such as the User Environment and the Battlespace/propagation models, thus reducing the development effort; and
- interact with all the models produced by other Developers, thereby greatly increasing its usefulness.

In fact, the plug-and-play interfaces can be extended and adapted by Developers and, provided this is done in a controlled fashion, the ability of all models to interact can be maintained.

A pool of SIMTEC Developers is desirable in order to set up a mutually-reinforcing improvement cycle, where each new project contributes new toolkit objects and models and provides the motivation for upgrading the COTS facilities and extending the interfaces. Developers must, however, 'buy in' – that is, commit to using SIMTEC – in order to achieve the benefits of its clear box, fine grain approach.

Other software reuse approaches, such as DIS/HLA, have concentrated on linking together different products via machine and language-independent plug-and-play interfaces. Although this avoids the need to buy-into any one product, it is fundamentally **black box** in nature and therefore produces only **coarse grain** reuse when compared with SIMTEC.

SIMTEC has, in fact, been linked to an experimental model federation via the HLA. It may be, therefore, that fine-grain/buy-in approaches and coarse-grain/linking techniques are complementary. That is, SIMTEC could be used to produce the models that are then linked with other systems via HLA/DIS.

Conclusions

The development of SIMTEC has shown that an extremely high degree of software reuse is possible. This has enabled SIMTEC to support a wide range of modelling applications and provide a completely generic User Environment that is automatically configured to the modelling code.

SIMTEC's software reuse has been achieved by

(i) simultaneously applying several techniques,

(ii) undertaking development on a large enough scale for these techniques to be effective, and

(iii) accepting certain trade-offs.

The paper has identified four main reuse techniques:

- OO toolkit approach,
- plug-and-play interfaces,
- design for data configurability, and
- use and production of COTS tools.

It has been found that a development in excess of 100,000 C++ statements has been required to achieve the full benefits of these techniques.

The major trade-off with the SIMTEC initiative is that fine-grained software reuse – that is, at the individual module level – is possible only if Developers commit, or 'buy-in', to one particular framework. There is little reason to suppose that it will be possible to swop individual modules seamlessly between frameworks any time in the near future.

Initiatives such as HLA and DIS, make different trade-offs. Although they achieve a coarser level of software reuse they avoid the requirement for Developers to buy-in to individual frameworks. This paper has argued that large frameworks that use the right design approaches are complementary to DIS/HLA-type initiatives – that is, frameworks can be used to construct the models linked by DIS and HLA.

For more detailed information on the SIMTEC framework, please contact the author.

Acknowledgements

The author would like to thank Sandy Flett of Weapons Sector, Farnborough for initiating and supporting the SIMTEC project. Thanks are also due to John Witney and Ian Begg of EASAMS; the former for project managing SIMTEC and performing much of its detailed design, and the latter for implementing some of SIMTEC's advanced concepts (such as the Model Map interface) as a practical software solution.