

# Gedae: Past, Present and Future

By William I. Lundgren

18 February 2003

# Past – Overview



- Milestones that define Gedae's development
- Key players that have contributed to the innovation and acceptance of Gedae
- Programs that drove the development and adoption of Gedae

# Past – Important Milestones



- 1987 – initial development
  - Targeted workstations and super computers
  - Built on an object system
- 1995 – refocused Gedae
  - Multiprocessor systems
  - Discarded object system
- 1997 – first credible application
  - ATR application
  - 65 Sharc processors
  - Stressed memory and computation

# Past – Important Milestones



- 1999 – Gedae ready for deployed systems
- 2000 – ESPADON selects Gedae as “Best in Class”
  - Gave Gedae credibility in Europe
- 2001 – Blue Horizon purchases Gedae
- 2001 – CAPTOR Programme Adopts Gedae
- 2002 – Mode Control (signal segments) directly supported
- 2003 – First Gedae Users’ Conference

# Past – Important Players



- Bernie Schaming and Jeff Pridmore – LM ATL
  - Supported Gedae with development dollars
- Kerry Barnes, Mike Both, Clay Fickle and Bill Lundgren
  - Key technology developments
- Doug Smith – US Air Force Research Laboratory
  - First serious / longest continuous user
- Bob Madahar
  - UK Leader of ESPADON Programme naming Gedae “Best in Class”
- Professor John Roulston
  - Accepted the risk - choose Gedae for CAPTOR Tranche II

## Past – RASSP Benchmark 3

- SONAR with modes that tested the command programs and mode control.
  - Demonstrated the automatic generation of a command program from Object Geode
  - Resulted in the development of an extensive command program interface
  - Illustrated what we would now consider a primitive technique for mode control

# Past – RASSP Benchmark 4

- Automatic target recognition
  - 72 processor Sharc system
    - 18 per board
    - 16 with no off-chip memory
  - Forced improvement of:
    - In place operation with cyclic boxes
    - Schedule algorithm efficiency
    - Memory use for implementation

# Past – AFRL Audio Processing



- Forced addressing issues of segmented processing
  - It was actually several years before that capability was complete
    - Gedae 4.0 in September 2002
    - Gedae 4.1 in February 2003

# Past – IDT Processing

- Thales Underwater Systems
- Intelligent Detection and Tracking
  - Developed concepts for data processing
  - Initial implementation was quite complex and used complex database concepts
  - Evolved into a much simpler – easy to upgrade graph

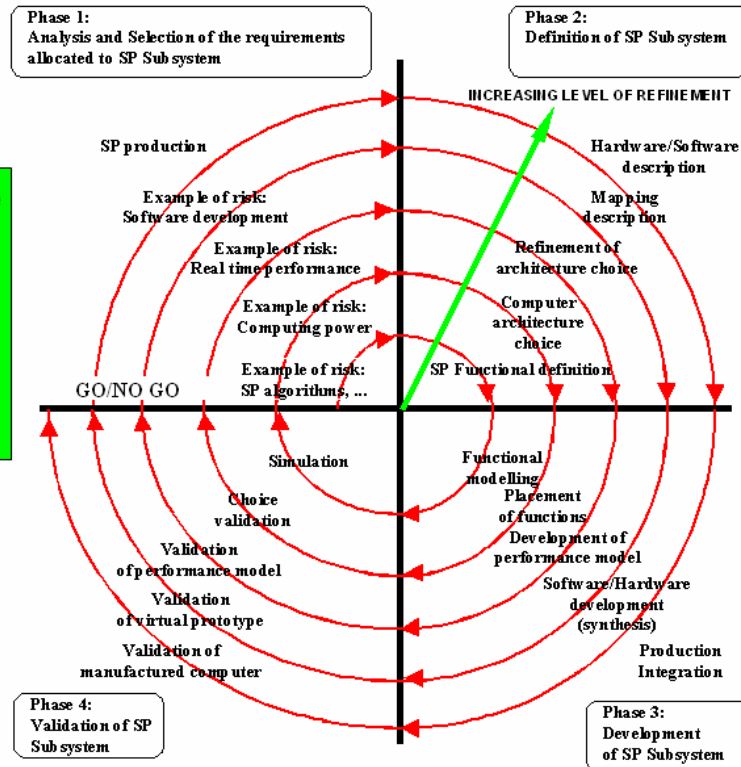
# Past - ESPADON Programme



## The ESPADON Methodology

### Spiral Model Representation

- Risk driven development life cycle
- « Model Year » approach
- Reuse and capitalisation
- Support for:
  - traceability
  - cost performance trade off



BAE SYSTEMS Signal and Data Processing Conference, 5 - 7 March 2002, p2





## Captor Tranche 2 Summary

- The CAPTOR Radar System is developed by EURORADAR, a four nation European consortium, for the Eurofighter aircraft.
- The function of the CAPTOR Radar Processor LRI is the
  - Control of the other Radar LRIs
    - Scanner, Receiver, Transmitter, Waveguide Unit
  - Sequential processing of digitised baseband Receiver returns
- Current CAPTOR Tranche 2 development is a redesign of the **Processor** LRI to remove hardware obsolescence
  - In Tranche 1 the Processing task was partitioned into a Data Processor (DAP) and a Signal Processor (SIP) implemented in 68040 and microprogrammed custom ASICs respectively.
  - For Tranche 2
    - re-implementation of SIP algorithms using the GEDAE toolset
    - reusing (recompiling) DAP Ada code
    - AltiVec G4 PowerPC architecture target is currently under development
- The Tranche 2 Signal Processing Software is jointly developed using Gedae by BAE SYSTEMS and EADS
  - Significant undertaking - redesign of existing product size ~100kSLOC (hand coded real-time embedded C)

BAE SYSTEMS

Unclassified

Slide for GUC 03/03/03

# Present – EO Systems Demonstrator

01000111 01000101 01000100 01000001 01000101  
Gedae, Inc.  
01000111 01000101 01000100 01000001 01000101

- BAE SYSTEMS, Sensor Systems Division
- Provides an opportunity to better understand
  - Application control
  - Programming a heterogeneous system with Altivec and FPGAs
- Gedae, Inc. will release a demonstration and documentation that illustrates the techniques

# Present – Overview

- Gedae is unique. It is not just a tool. It is a language and a collection of tools. This section of the presentation defines the components of Gedae and then describes each component.
- This part of the presentation outlines the scientific basis for the collection of tools in Gedae.

## Present – Overview (continued)

- The user requirements
- The Gedae Graph Language (not discussed)
- The Gedae Data Flow Language
- The Command Program Interface (not discussed)
- The essentials of the implementation that make Gedae work
- The present state of Gedae
  - Capability
  - Efficiency (spans present and future)

# Present – What is Gedae?



- The Gedae language allows the user to express software design directly
  - Functionality
  - Implementation
- Gedae automatically builds an implementation from the design
- Gedae reports the implementation and the execution

# Present – User Requirements

- Developers need a language to implement all algorithm and application behavior
  - Gedae needs to know the input, output and local data requirements and data processing for each node
  - Specification of threshold, consume and produce, and a function are sufficient
- The implementation must be efficient

# Present – User Requirements

(continued)

- Problems with using only threshold, consume and produce specified at runtime
  - Specification of some user requirements are too difficult
  - Efficient implementation is impossible
- The Gedae Language is summarized in the next few charts.
  - Items that enable **functional** design are colored green
  - Items that provide **information** to Gedae in support of efficient implementation are colored rust

Functional  
Informational

# Present – Language/Benefits

- Static data flow
  - Allows preplanning an execution schedule
- In place
  - Reuse memory
- Failure and retry
  - Enables synchronization with external events

Functional  
Informational

# Present – Language/Benefits (continued)

01000111 01000101 01000100 01000001 01000101  
Gedae, Inc.  
01000111 01000101 01000100 01000001 01000101

- Dynamic data flow
  - Data flow variation is an inherent part of the signal or data processing
  - Threshold, produce or consume set at runtime
- Combination of static and dynamic data flow – e.g. branch and merges
  - Direct expression of processing control
  - Efficient implementation of control possible

Functional  
Informational

# Present – Language/Benefits (continued)

01000111 01000101 01000100 01000001 01000101  
Gedae, Inc.  
01000111 01000101 01000100 01000001 01000101

- Exclusive family outputs
  - Data is produced on one output stream at a time
  - Sequence of operation is specified
  - State changes can be synchronized across distribution
  - Resources can be reused among branches

Functional  
Informational

# Present – Language/Benefits (continued)

01000111 01000101 01000100 01000001 01000101  
Gedae, Inc.  
01000111 01000101 01000100 01000001 01000101

- Segment boundaries in the data streams are marked. At the boundaries:
  - Local state of a function box can be reset
  - Parameters can be changed
  - Summary results can be output
  - External state can be reset

Functional  
Informational

# Future – Language/Benefits (continued)

01000111 01000101 01000100 01000001 01000101  
Gedae, Inc.  
01000111 01000101 01000100 01000001 01000101

- Pointer inputs and outputs
  - This is the single language benefit that is not implemented. It will be available in Gedae 4.2
  - Supports the use of system memory provided by input devices as schedule memory
  - Unnecessary memory copies are eliminated

Functional  
Informational

# Present – Language/Benefits (continued)

01000111 01000101 01000100 01000001 01000101  
Gedae, Inc.  
01000111 01000101 01000100 01000001 01000101

- Parameter Eval boxes
  - Asynchronously calculate stateless parameters
  - Can be collected and code generated into functions
- Parameter Trigger boxes
  - User controls the sequence of execution to effectively handle side effects of GUIs and state machines

Functional  
Informational

# Present – Why / How does Gedae Work?

01000111 01000101 01000100 01000001 01000101  
Gedae, Inc.  
01000111 01000101 01000100 01000001 01000101

- Simple but powerful language matches user requirements but enables automatically building an efficient implementation
- The following charts summarize the implementation layers
  - The functional capabilities supported by the layer are colored green
  - The language features implemented are colored rust
  - Implementation techniques are colored blue

Functional

Informational

Implementation

# Present – Why / How does Gedae Work?

01000111 01000101 01000100 01000001 01000101  
Gedae, Inc.  
01000111 01000101 01000100 01000001 01000101

- Layered implementation
  - Parameter evaluation
    - GUIs, and application parameterization and application control (state machines)
    - Eval and trigger boxes
    - Instantiation, data flow and runtime parameters
    - Code generation

Functional  
Informational  
Implementation

# Present – Why / How does Gedae Work? (continued)

01000111 01000101 01000100 01000001 01000101  
Gedae, Inc.  
01000111 01000101 01000100 01000001 01000101

- Layered implementation (continued)
  - Command programs
    - GUIs and Application Control
    - Extensive Command Program Interface
    - Hand code using C
    - Have been auto coded from state machine tools

Functional  
Informational  
Implementation

# Present – Why / How does Gedae Work? (continued)



- Layered implementation (continued)
  - Segmentation control – mode control
    - Mode control
    - Data processing
    - Mark segment boundaries on data streams
    - Reset box state and output summaries on segment boundaries
    - Synchronize distributed stateful processing
    - Implemented using states and transitions

Functional

Informational

Implementation

# Present – Why / How does Gedae Work? (continued)



- Layered implementation (continued)
  - Dynamic scheduler
    - Variable data streams
      - For example, symbols from a communications stream
    - Conditional processing including loops, switches, and if-then-else's
    - Interface to external devices
    - Data dependent data flow
    - Processing control

Functional  
Informational  
Implementation

# Present – Why / How does Gedae Work? (continued)



- Layered implementation (continued)
  - Dynamic scheduler (continued)
    - Enumerated parameterizations
      - For parameters that effect execution or memory allocation
    - Failure modes
      - Suspend queue wait
      - Suspend retry
      - Terminate conditions
    - OS Scheduling techniques

Functional

Informational

Implementation

# Present – Why / How does Gedae Work? (continued)



- Layered implementation (continued)
  - Static schedule engine
    - Processing chains
    - Static data flow
    - Sets function box state including parameter values, memory and granularity
    - Reentrant schedules for schedule suspend
    - Preplan execution
    - Reuse memory between exclusive segments

Functional

Informational

Implementation

# Present – Why / How does Gedae Work? (continued)



- Layered implementation (continued)
  - Static scheduler (many components)
    - Preschedules execution and memory use
    - Inserts function boxes into a block diagram as necessary
    - Analyzes data flow to produce execution schedule
    - Memory packing algorithms

Functional  
Informational  
Implementation

# Present – Fully Capable

- Signal processing
  - Roadnight – SONAR, Adaptive Beamformer
  - Stevens, Young, Dominy – RADAR, SAR
  - Ostapovich – Electronic Support
  - Humphreys, Graeme, Daniell – EO Image
  - Purdie, Wardell – RADAR
  - Taylor – Active SONAR
  - Yarker – Testing RADAR Mode Graphs

# Present – Fully Capable

- Data processing
  - Challands, Benton – SONAR, Track Repair
- Mode Control
  - Swanson – RADAR
- State Machines
  - Alston, Madahar
- GUI
  - Smith, Curtacci – Audio Processing

Present –  
Efficient Applications

01000111 01000101 01000100 01000001 01000101  
01000111 01000101 01000100 01000001 01000101  
**Gedae, Inc.**  
01000111 01000101 01000100 01000001 01000101

- Approximately hand coded efficiency

## Future – Overview

- This section describes the areas that will receive our greatest attention
- Efficiency is important because it affects every parameter – cost, size, weight, and power
- Adoption of innovation is slow and we must accelerate
- A big part of adoption is documentation

## Future – Overview (continued)

- The level of user feedback possible is unique and not fully exploited
- Von Neumann architectures are general purpose and can implement arbitrary behavior, but the efficiencies provided by alternate architectures must be exploited
- Gedae makes multiprocessor von Neumann architectures easily programmable
- Gedae will make other architectures easily programmable as well

## Future – Overview (continued)

- Gedae as a development environment is not particularly bug free 😊 The applications produced are already reliable. There are many opportunities to ensure an even more reliable software.
- There are many areas of reliability, completeness and ease of use that need attention. These are omitted.

# Future – Efficient Applications



- Potential areas of improvement
  - Memory copies
  - Combine data reorganization with transfers
  - Static schedule engine
  - Runtime Kernel
- Gedae will provide levels of optimization that are impractical to do by hand

# Future – Innovation

- Gedae is an innovation – it changes
  - Engineering processes
  - Developing algorithms
  - Software specification
  - Implementation
  - Optimization
  - Nature of software support layers
    - Communication libraries
    - Optimized libraries
    - OS (Gedae Runtime Kernel)

# Future – Learning and Teaching Gedae Language Techniques

01000111 01000101 01000100 01000001 01000101  
Gedae, Inc.  
01000111 01000101 01000100 01000001 01000101

- The Gedae language is powerful
  - Techniques for using the language need to evolve
    - State machines use
    - Gedae, Inc. is building examples that need to be turned into documentation
    - Users are building examples – State machine example by Ian Alston
  - Complete documentation

# Future – User Feedback

- Gedae is in the unique position of having every detail of the implementation mirrored in data structures
  - Simple to record and report events (execution trace)
  - Implementation is algorithmic and all identified problems and potential problems can be reported in detail
  - Data structures can be algorithmically analyzed and reported (for example, statistics)
  - Debugging can be extensive and detailed including post mortem dumps

# Future – Support for Alternate Architectures

01000111 01000101 01000100 01000001 01000101  
01000111 01000101 01000100 01000001 01000101  
**Gedae, Inc.**  
01000111 01000101 01000100 01000001 01000101

- Non-Von Neumann architectures are an integral part of signal processing hardware solutions. FPGAs are the most popular example.
  - Gedae has adopted RTL and data flow as a starting point
  - We will evolve those languages to provide the same levels of algorithm expression and implementation efficiency as the current Gedae
    - Biggest issue is, unlike DSP multiprocessors, there seems to be a lack of standard techniques

# Future – Support for Alternate Architectures (continued)



- From the C perspective a vector processor can be considered an alternate architecture. One of the criteria for a successful language is the ability to compile to an efficient implementation on vector processors.
- Jim Steed – A new important player because of his responsibility in this area
- Transtech DSP – Gedae has formed a relationship with Transtech DSP to add a comprehensive heterogeneous capability to program G4's, Tiger SHARCs and FPGA.

# Future – Software Assurance / Safety Critical



- We do not support safety critical software now. We may in the future.
- Gedae is an ideal implementation for the software assurance and the eventual support of safety critical applications
- As already mentioned, the detailed implementation is recorded in data structures and subject to algorithmic analysis. Gedae already does a lot of analysis but can do more.

# Future – Software Assurance / Safety Critical

01000111 01000101 01000100 01000001 01000101  
Gedae, Inc.  
01000111 01000101 01000100 01000001 01000101

- There are 3 areas that we can pursue.
  - Assurance of consistent memory use
    - Certify individual functional components use of memory
    - Algorithmically analyze exposed computation of coefficients that effect memory access
    - May require some language limitations. These would be reported but not made illegal

# Future – Software Assurance / Safety Critical (continued)

01000111 01000101 01000100 01000001 01000101  
**Gedae, Inc.**  
01000111 01000101 01000100 01000001 01000101  
01000111 01000101 01000100 01000001 01000101

- There are 3 areas that we can pursue (continued)
  - Assurance of implementation algorithm correctness
    - Implementation algorithms analyze data flow
    - Since dynamic and non-deterministic data flow is hidden, it will require some certification and exposure of the behavior of those boxes.

# Future – Software Assurance / Safety Critical (continued)



- There are 3 areas that we can pursue (continued)
  - Partitioning of safety critical components
    - All the memory and processing is controlled by Gedae
    - It can place safety critical memory in secure partitions
    - It can place safety critical functions in secure time slots

# Summary

01000111 01000101 01000100 01000001 01000101  
Gedae, Inc.  
01000111 01000101 01000100 01000001 01000101

- The picture of Gedae presented is innovative, bold and ambitious. It is real. The users presenting over the next two days are a testament to its reality. Its value today makes investment recovery short, even with process changes and the learning curve. The value will grow.