PRECISE DEFINITION OF SOFTWARE COMPONENT SPECIFICATIONS

John Horst, Elena Messina, Tom Kramer, & Hui-Min Huang

Intelligent Systems Division
The National Institute of Standards and Technology
Bldg. 220 Rm. B-124, Gaithersburg, MD USA 20899
voice: (301)975-{3430, 3510, 3518, 3427} FAX: (301)990-9688
{john.horst, elena.messina, tom.kramer, hui-min.huang}@nist.gov
http://isd.cme.nist.gov/

Abstract: A set of generic specification categories is presented which can be used to comprehensively define any software component within a certain class. With these categories as a template, a specific set of formal specifications can be generated for each component. Specifications for a particular component (an algorithm that estimates the position and orientation of a physical object using visual sensing) have been defined in EXPRESS, an information modeling language. A few example natural language specifications are presented for this particular component.

Keywords: Components, Computer vision, Formal languages, Formal specification, Software engineering, Software metrics, Software performance, Software specification, Software tools

1. INTRODUCTION

The following is the software system development sequence that this research addresses:

- A new software component is created, and the component developer (i.e., the vendor) wants see it widely used
- A software systems developer (i.e., the user) examines the component (typically via a prose document) to see if it satisfies the requirements of a particular application

This development process is inefficient. In the near term, the situation would improve substantially if the vendor could more precisely communicate to the potential user the functionality and performance of the software component. Ultimately, it would be good to automate this process using design and simulation tools which can exploit 'smart' components.

At best, a software component user has access to a research report and commented source code for the component. Research reports commonly present the component in its most favorable light and may neglect to mention various important aspects of the component of critical importance to a potential user, in part because there are no standards for how a component should be specified in a research report. Similarly, source code (even with copious and meaningful comments) may not reveal a host of critical factors of importance to the user. These include complexity, input data constraints, environmental constraints, upstream computing requirements, and computing system constraints. These and other aspects of the component may decide whether the component will or will not work for the user. Improving the efficiency of this process will become increasingly critical to competitiveness for many industries.

In the hardware development world one sees a design process something like the following: A circuit designer seeks to design a new printed circuit board. Perhaps through a design/simulation tool, the designer chooses chips and chip sets for a board design based upon published specifications (e.g., in a hardware definition language like VHDL) detailing performance characteristics. This hardware system design process is possible because 1) specifications exist that comprehensively define the performance of hardware components and 2) software models of the hardware components are defined by chip vendors and hardware system design/simulation tools are available that use those models to design systems. Our research seeks to enable a similar design process:

- A vendor creates a new software component and develops a set of specifications[1] with the aid of a software tool. These specifications would fully describe the component using a combination of natural and formal languages. The vendor publishes these specifications on the web in the form of static data and/or program 'applets.'
- A potential user, armed with system requirements, searches the web for components meeting those requirements. The user examines the vendor's component through its specifications either by viewing a natural language version or reading a formal language version through the use of a simulation/design tool. The user might conceivably execute (on

[1] 'Specifications,' means component specifications as opposed to overall software system specifications.

a trial basis) all or part of the component through program 'applets.'

Using formal component specifications, a software tool for software system design and simulation would be able to perform very high-level system simulations. For example, such a tool might simulate input error propagation through a set of connected components using data described in the specifications. This can be viewed as high fidelity simulation early in the design process. Meaningful simulations performed early in the design process improves design efficiency.

Initially, these software component specifications would be used by human developers for software component verification, i.e., does the component under consideration meet system requirements. Ultimately, such specifications are intended for use by software system design and simulation tools which analyze, simulate, and integrate independently developed software components according to explicit system requirements. For example, such a tool could simulate time of execution of a set of connected components using data described in the specifications. For example, such a tool could simulate input error propagation through a set of connected components using data described in the specifications. Software component specifications are analogous to existing specifications for hardware components that describe their functionality sufficient for use in hardware system simulation tools. This process (software system composition using component specifications) would occur naturally on the internet. Specification data and program segments (applets) would be easily and securely accessed by systems developers and software system composition tools.

2. COMPONENT SPECIFICATIONS IN THE LITERATURE

Specifications, formal or informal, for software components are addressed in at least three areas in the literature, 1) analysis of algorithms (Hofri, 1995), 2) software engineering (Vick and Ramamoorthy, 1984), and 3) software metrics (Grady and Caswell, 1987). Our interest in component specifications differs slightly from the treatment of the same in these other areas as will now be discussed.

The analysis of algorithms literature seeks to improve the understanding of algorithms in order to suggest improvements and uncover trade-offs. For example, if one can precisely define algorithm complexity and optimality, a more intelligent comparison can be made with competing algorithms of the same class (i.e., those that solve the same problem). Impetus for the development of suboptimal, but efficient approaches can come from such analysis. This literature is often interested in the precise details of analysis, such as precise definitions for algorithm complexity and development of proofs for optimality.

Software engineering is concerned with the specification of components from the standpoint of software systems design, implementation, and maintenance. Component specifications are therefore of interest in as much as they help to define resources (e.g., labor, tools, computer memory, and processor horsepower) and to uncover trade-offs critical for successful system design and maintenance.

Software metrics research seems to be even further removed from an analytical interest in algorithms and is concerned with things like how many lines of code are in a component, how well the code is documented, how many person months are required for design, implementation, testing, and maintenance, and similar measures.

Our research has most in common with software engineering's approach to component specifications. Nonetheless, there is a uniqueness to this research which can be summarized as follows, 1) develop *standardizable* categories for *comprehensively* describing components, 2) instantiate these categories for a variety of components, 3) investigate formal information modeling languages (or specification languages) to define the specifications, 4) focus on the use of component specifications to allow software system developers to more efficiently utilize components, 5) investigate the use of component specifications by design/simulation tools in order to make the software system design process more efficient, and 6) investigate the use of specifications on the web.

3. MODELING SPECIFICATIONS

A critical goal of this research is to use a formal information modeling language to record information about particular components in the specification categories outlined in section 4. The EXPRESS language developed as part of ISO standard 10303 (generally known as STEP) appears to be adequate for much of the information (ISO, 1994a).

The EXPRESS language allows the definition of types of data either as TYPES (similar to types in C or C++) or as ENTITIES (similar to structs in C or C++). As compared with C or C++, EXPRESS is richer in providing for constraints on data, and making it easy to specify a range of choices for a data type. EXPRESS, however, does not provide methods and is not compilable into an executable computer program. Tools for automatically generating C++ class definitions and access functions from an EXPRESS schema are available. The C++ code generated may be incorporated in a computer program. A graphical representation of an EXPRESS schema may be obtained using the EXPRESS-G language and available tools which generate EXPRESS-G diagrams from EXPRESS schemas.

An EXPRESS model of the generic specifications, given in section 4, has been built. Using this model, particular components may be described in STEP Part 21 exchange files (ISO, 1994b). Tools which can automatically read Part 21 files into a computer program (or write them out) are also available.

The EXPRESS model of the specifications is intended to provide a structure for organizing and representing all (or as much as possible of) the information about a component, for use by a partially or fully automated software system composition tool (a generally applicable fully automatic composition tool is not expected to be feasible in the near future). This model is not intended to provide a set of blanks for a component developer to fill in manually (although the model should be useful in designing a set of blanks). The primary objective of this model is to support an interactive system in which a human makes many of the major decisions.

This model provides many places where natural language entries are expected. The model serves to organize these natural language descriptions. The model also provides several places where either natural language or computer language might be used. The idea is that if computer language text is provided, it can be handed to a language processor for the language used. By providing the two alternatives (formal or natural language), a wide range of meaning is allowed by natural language and more precise meanings can be expressed in the formal language.

The intent is that one exchange file using this EXPRESS model will suffice for representing either a single component or a set of related components.

It would be possible (but difficult and tedious) to represent specific computer languages in EXPRESS, so that, for example, routines and data structures could be defined in detail, not just as strings, in Part 21 files. This is not worth the trouble at this stage in the research, and may never be. The approach used in this model is to identify the language and if it has been used for each particular routine or data structure. If the language has been used, a parser for that language could then be used to deal with the strings.

The authors have examined other languages, such as Z (Zed) for modeling components (Diller, 1994). Z appears to be superior to computer languages in some respects, particularly to specify what a component is supposed to do. Z does not, however, give us many of the easily-used capabilities of EXPRESS. Hence, Z or other similar languages have not been used.

STEP Part 21 files are quite difficult for humans to read. They may be artfully formatted, alleviating some of the difficulty, but knowing the EXPRESS model which corresponds to a particular Part 21 file is required to understand the file. Such files will be hard to read regardless of how artfully printed. The authors anticipate having some intermediary between Part 21 files and humans. The simplest type of intermediary is a reformatter that would put the information from the Part 21 file into a more readable kind of file capable of holding the same information, such as EXPRESS-I. A more sophisticated intermediary would be a software tool allowing human entry of component specification information at a high level of abstraction.

4. GENERIC COMPONENT SPECIFICATION CATEGORIES

Generic, comprehensive, and reasonable specification categories for describing software components are commonly found in the literature on software engineering (Vick, 1984) and the analysis of algorithms (Hofri, 1995). This paper takes a fresh look at such categories based on the research focus described in section 2. Such a set of generic categories is now defined. These generic categories are conceived as a template and are intended to be used by the component vendor to define a useful summary of the operation and performance of a particular component in the class. This template is presented as a series of questions in each category.

Problem definition: What problem is this component intended to solve?

Applications and competing components: What are the potential and known application areas for this component? For each application area, give a set of competing components with references.

Input data: is the input data to the component? What input data sets were used in the testing? What are the dimensional units of the input data? What is the format of the input data? Are the representations chosen for the input data consistent with the expected or typical upstream components? What are the input parameters required (if any) and what meaning do they have for the operation and performance of the component? Are there any input parameters that allow the user to specify the type and/or format of the output?

Output data: What are the outputs of the component? What is the format of the output data?

How are the various formats for the output data specified? For instance, if the output data is contained in files, what are the file formats? Are there any input parameters that allow the user to specify the type and/or format of the output? Are the representations chosen for the output data consistent with the expected or typical downstream components?

Transfer and feedback relations: How do the inputs relate to the outputs, i.e, what are the transfer and the feedback relations? If one can describe these relations analytically, describe these equations, e.g., are they linear or non-linear, and state these equations, e.g., define meanings of variables and write out all relations. Under what conditions are the equations over-determined or under-determined?

Input data constraints: What are the constraints on the format and nature of the input? There may be constraints on the nature of the input beyond what is inherent in the nature of the component. For example, the number of elements in an array might be constrained to be even, or the difference between input numbers might be constrained to be greater than some value, or the number of input values might be constrained to be in some range.

Environmental constraints: 'Environment' means 'factors external to the computing system and its data which affect the performance of the component.' If the task of the component is to examine or manipulate physical objects, what are the constraints on the format and nature of those objects? For instance, what are the rigidity, size, shape, color, surface finish, or illumination conditions? If there is a sensing device, what are the constraints on the type and use of the sensing (e.g., structured light, CCD camera, range camera)? Does the object need to be placed in some approximate pose? Are there special configurations of the environment which might cause the component to fail?

Knowledge data constraints: What are the constraints on the format and nature of the knowledge data? For example, can a CAD drawing in some standard format be used for matching sensed 'features' to model features?

Computing constraint: What, if any, are the operating system requirements of the component? In what computer language is the source code written? Is the source code available? Are there any system architecture requirements for using the component? What kind computing hardware is required by the component?

How much RAM memory and disk space is needed? Are there any constraints on the numerical precision of the processing system?

Speed: Based on actual examples run on specific computers, how fast does the component run? What is the execution time of each of the subcomponents of the component? If speed depends on the size or type of input data, give the speed of execution for a fixed and standard size or type of input.

Benchmarks: If there is a standard test suite (a set of benchmarks) for components performing the same task, what is that benchmark and how does the component perform against it? Are there optimal components that can produce the ideal output? How is optimality defined for each component? Are there other measures of performance, e.g., statistical measures? If so, how does the component perform by these measures?

Robustness: How robust is the component, i.e., how does the component perform in the presence of large perturbations, i.e., replacement noise, on a subset of its data values? How does the component perform as replacement noise varies throughout its range, e.g., does performance degrade smoothly or catastrophically? Several aspects of this type of noise can vary, for instance, the size and type of the perturbations and the size of the subset of all data values affected. Is the component able to perform well (or at all) if the input is outside of the specified region? State all models that exist for replacement noise. For instance, what is the model for the 'ideal' world? What is the model for large perturbations (e.g., mismatch) on this ideal world? What is the criterion function for measuring the difference between noisy output and ideal output (Haralick, 1992)?

Noise: How does the component perform in the presence of small perturbations on all data values? How does the component perform as this type of noise varies throughout its range, e.g., does performance degrade smoothly or catastrophically? Several aspects of this type of noise can vary. For instance, both the size and type of the perturbations can vary. State all models that exist for this type of noise. For instance, what is the model for the 'ideal' world, what is the model for small perturbations on this ideal world, and what is the criterion function for measuring the difference between noisy output and ideal output (Haralick, 1992)?

Complexity: What are the relations defining computational complexity of the component (there may be more than one such relation) as functions of the input variables? Complexity relates closely to speed. But an analysis of complexity typically does not deal with the initial fixed cost of the component or with the size of the constants by which various terms of the complexity must be multiplied. What assumptions are made in the complexity analysis?

Convergence: Is the component iterative or closed form? If iterative, under what (if any) conditions is convergence guaranteed? Does the component converge to the global solution?

Internal data representation: If there is some knowledge (i.e., not input/output data) that is explicitly represented within the component, what is the format of the representation?

Reliability: How reliable is the component? Are there known bugs? What reliability tests has the component passed, such as the checking provided by commercially available reliability tools, or a theoretical correctness proof? Example entities checked by such tools are uninitialized local variables, uninitialized malloc'd memory, using freed memory, overwriting array bounds, over-reading array bounds, memory leaks, file descriptor leaks, stack overflow errors, and stack frame boundary errors. Has there been testing by some sort of coverage tool, which keeps track of which code is executed in a given session? Coverage tools give a sense of how much of the code was exercised by other reliability tests. The granularity can be at the function, block, or line level. A coverage will give the potential user greater confidence in a software component's reliability if it had been covered 100% during testing and errors were cleaned out of it.

Testing and analysis: What experiments have been done with the component? If input data is varied over a set of variables, what criteria is chosen to sample the space of variables in order to generate sample input data? Was Monte Carlo testing done? Are there simulators available to generate input data? What kinds of analysis have been done on the results? What kinds of graphs and tables have been produced and what is their format? What statistical methods have been used?

Upstream and downstream requirements: What kind of components need to be performed prior to or subsequent to the execution of this component? Are the representations chosen for the input and output data consistent with the expected or typical upstream and downstream components? Does anything in the component constrain the upstream or downstream components that must be used? The types of upstream and downstream components may depend on the particular area to which the component is applied.

Parallelizability: Can the component be parallelized? Has it been? How does parallelizing affect performance?

Modularizability: Can subcomponents of the component be modularized? Have they been? How might modularization affect component performance?

Errors: What is the error criterion, e.g., least squares? What kinds of input errors and/or internal errors does the component detect? What does it do if such errors are detected? Are there error recovery procedures?

Nature of interaction: How is the component used? Can it be used by function call, or is the component part of a system that runs in client-server mode, or some other more complicated mode?

Coding style: How is the code written, e.g., in functional, procedural, recursive, or object-oriented style?

Compliance to standards: Which (if any) standards (published or defacto) are used and complied with?

5. NATURAL LANGUAGE INSTANTIATION OF GENERIC SPECIFICATIONS

`What is the input data to the algorithm?`
An array of N vectors of the following nine real numbers (doubles) for each $i = 1, 2, \ldots, N$,

$$(a_i, b_i, c_i, x_i^0, y_i^0, z_i^0, \alpha_i, \beta_i, \gamma_i)$$

where N is the number of lines in the image that correspond to lines in the model, (a_i, b_i, c_i) is the unit vector of parameters that solve the equation, $a_i u + b_i v + c_i = 0$ for all (u, v) on the ith measured line in the image, (x_i^0, y_i^0, z_i^0) is the position vector of the initial point of the corresponding model line segment in model coordinates, and $(\alpha_i, \beta_i, \gamma_i)$ is the unit directional vector of the same model line segment also in model coordinates. Also required for input are the parameters for perspective transformation, $(t_x, t_y, t_z, \delta, \varepsilon, \zeta, f)$, where (t_x, t_y, t_z) is the vector of translation from the origin of the machine coordinate system to the center of the camera lens, $(\delta, \varepsilon, \zeta)$ is the vector specifying roll, pitch, and yaw of the camera in radians (yaw is a counter-clockwise spin around the z-axis, pitch is a counter-clockwise spin around the x-axis, and roll is a counter-clockwise spin around the y-axis), and f is the camera focal length. What input data sets were used in the testing? Tan has models of automobiles. The authors have a slightly modified cube for testing which is defined in Mathematica™. What are the dimensional units of the input data? (a_i, b_i, c_i) are dimensionless, (x_i^0, y_i^0, z_i^0) are in meters, and $(\alpha_i, \beta_i, \gamma_i)$ are in radians. (t_x, t_y, t_z) are in meters, $(\delta, \varepsilon, \zeta)$ are in radians, and f is in meters.

What is the format of the input data? All are input arrays are arrays of doubles. Are the representations chosen for the input data consistent with the expected or typical upstream algorithms? The expected upstream algorithm is a sensed feature to model feature matching algorithm and, since the Tan algorithm requires input line matches for edges on a planar polygonal object, the matching algorithm must produce line matches as well. However, if point or line segment matches are all that is available from the matching algorithm, line parameters can easily be generated from them.

What are the input parameters required (if any) and what meaning do they have for the operation and performance of the algorithm? None required. Are there any input parameters that allow the user to specify the type and/or format of the output? No.

What are the outputs of the algorithm and what is the format of the data? An array of four doubles, (x, y, θ, k), where (x, y, θ) represent the two dimensional position and orientation of the part and k represents the scale of the part (in case the part measured is a scaled version of the model). The remaining three parameters, (z, ϕ, Ψ), required to fully specify the three dimensional position and orientation are assumed to be known and equal to zero *apriori*. How are the various formats for the output data specified, for instance, if the output data is contained in files, what are the file formats? An ANSI C-compliant array. Are there any input parameters that allow the user to specify the type and/or format of the output? No.

What is the format of the output data? All are input arrays are arrays of doubles. Are the representations chosen for the output data consistent with the expected or typical downstream algorithms? Yes.

How do the inputs relate to the outputs, i.e, what are the transfer and the feedback relations? If we can describe these relations analytically, state the relations as formal equations (defining all variables) along with some general description e.g., are they linear or non-linear. Under what conditions are the equations over-determined or under-determined? If the relations cannot be expressed analytically express them in whatever form is appropriate, e.g., high level software code. How do the inputs relate to the outputs, i.e, what are the transfer and the feedback relations? This algorithm is open loop. If we can describe these relations analytically, describe these equations, e.g., are they linear or non-linear. The transfer relationship is non-linear. State these equations, e.g., define meanings of variables and write out all relations. Form the following matrix, $\mathbf{M}_{wI}$, using the input data, $(t_x, t_y, t_z, \delta, \varepsilon, \zeta, f)$,

$$\mathbf{M}_{wI} = (\mathbf{P}\mathbf{T}_I\mathbf{R}_{\varepsilon,\zeta}\mathbf{T}_L)^T, \text{ where } \mathbf{T}_L = \begin{bmatrix} 1 & 0 & 0 & -t_x \\ 0 & 1 & 0 & -t_y \\ 0 & 0 & 1 & -t_z \\ 0 & 0 & 0 & 1 \end{bmatrix},$$

$$\mathbf{R}_{\varepsilon,\zeta} = \begin{bmatrix} \cos\zeta & \sin\zeta & 0 & 0 \\ -\cos\varepsilon\sin\zeta & \cos\varepsilon\cos\zeta & \sin\varepsilon & 0 \\ \sin\varepsilon\sin\zeta & -\sin\varepsilon\cos\zeta & \cos\varepsilon & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix},$$

$$\mathbf{T}_I = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & -f \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \text{ and } \mathbf{P} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & \frac{1}{f} & 0 & 1 \end{bmatrix}.$$

Define vectors, $\mathbf{r}_i = (m_{i1}, m_{i3}, m_{i4})$, $i = 1, 2, 3, 4$, where m_{ij} is the element at the ith row and jth column of $\mathbf{M}_{wI}$. Using the input values, $(a_i, b_i, c_i, x_i^0, y_i^0, z_i^0, \alpha_i, \beta_i, \gamma_i)$ for $i = 1, 2, \ldots, N$, form the following scalar coefficients for each of the N matching lines

$$A = (x^0\mathbf{r}_1 + y^0\mathbf{r}_2) \cdot \mathbf{n}$$

$$B = (x^0\mathbf{r}_2 - y^0\mathbf{r}_1) \cdot \mathbf{n}$$

$$C = \mathbf{r}_1 \cdot \mathbf{n}$$

$$D = \mathbf{r}_2 \cdot \mathbf{n}$$

$$E = z^0\mathbf{r}_3 \cdot \mathbf{n}$$

$$F = (\alpha\mathbf{r}_1 + \beta\mathbf{r}_2) \cdot \mathbf{n}$$

$$G = (\alpha\mathbf{r}_2 - \beta\mathbf{r}_1) \cdot \mathbf{n}$$

$$H = -\gamma\mathbf{r}_3 \cdot \mathbf{n}$$

$$J = -\mathbf{r}_4 \cdot \mathbf{n}$$

Using these coefficients, form the following matrices,

$$\mathbf{A} = \begin{bmatrix} F_1 \ldots F_N & A_1 \ldots A_N \\ G_1 \ldots G_N & B_1 \ldots B_N \end{bmatrix}^T \qquad \text{(EQ 1)}$$

$$\mathbf{B} = \begin{bmatrix} 0\ldots0 & C_1\ldots C_N \\ 0\ldots0 & D_1\ldots D_N \\ 0\ldots0 & -J_1\ldots -J_N \end{bmatrix}^T \quad \text{(EQ 2)}$$

$$\mathbf{C} = \begin{bmatrix} H_1\ldots H_N & -E_1\ldots -E_N \end{bmatrix}^T \quad \text{(EQ 3)}$$

Our desired output is (x, y, θ, k), so define

$$k' = 1/k,\ x' = k'x, \text{ and } y' = k'y \quad \text{(EQ 4)}$$

and define

$$\mathbf{q}_1 = (\cos\theta, \sin\theta) \text{ and } \mathbf{q}_2 = (x', y', k'). \quad \text{(EQ 5)}$$

Define the following matrices

$$\mathbf{D} = \mathbf{A}^T\mathbf{A} - \mathbf{A}^T\mathbf{B}(\mathbf{B}^T\mathbf{B})^{-1}\mathbf{B}^T\mathbf{A} = \begin{bmatrix} a_1 & a_2 \\ a_3 & a_4 \end{bmatrix}$$

$$\mathbf{h} = \mathbf{A}^T\mathbf{C} - \mathbf{A}^T\mathbf{B}(\mathbf{B}^T\mathbf{B})^{-1}\mathbf{B}^T\mathbf{C} = \begin{bmatrix} h_1 \\ h_2 \end{bmatrix}$$

Define the following constants

$$c_3 = 2(a_1 + a_4)$$

$$c_2 = (a_1 + a_4)^2 + 2(a_1a_4 - a_2a_3) - (h_1^2 - h_2^2)$$

$$c_1 = c_3(a_1a_4 - a_2a_3) + 2h_1h_2(a_2 + a_3) - 2(a_1h_1^2 - a_4h_2^2)$$

$$c_0 = (a_1a_4 - a_2a_3)^2 - (a_4h_1 - a_2h_2)^2 - (a_1h_2 - a_4h_1)^2$$

Solve the following equation using these constants,

$$\lambda^4 + c_3\lambda^3 + c_2\lambda^2 + c_1\lambda + c_0 = 0 \quad \text{(EQ 6)}$$

and for each real solution, λ, find $\mathbf{q}_1$ and $\mathbf{q}_2$ that solves the following two equations:

$$\mathbf{q}_2 = (\mathbf{B}^T\mathbf{B})^{-1}\mathbf{B}^T\mathbf{C} - (\mathbf{B}^T\mathbf{B})^{-1}\mathbf{B}^T\mathbf{A}\mathbf{q}_1$$

$$(\mathbf{D} + \lambda\mathbf{I}_2)\mathbf{q}_1 = \mathbf{h}$$

The optimal $(\mathbf{q}_1, \mathbf{q}_2)$ is taken as the pair that minimizes

$$\|\mathbf{A}\mathbf{q}_1 + \mathbf{B}\mathbf{q}_2 - \mathbf{C}\|^2. \quad \text{(EQ 7)}$$

Under what conditions are the equations over-determined or under-determined? Since a non-linear least squares technique is used, equations should typically be overdetermined. In order for the equations not to be under-determined, there must be 3 or more non--degenerate line matches, i.e., $N \geq 3$. A line match is degenerate if 1) $(\alpha_i, \beta_i) = (0, 0)$ (i.e., the model line of a match is vertical to the ground plane), 2) for $i \neq j$, $(\alpha_i, \beta_i, \gamma_i) = (\alpha_j, \beta_j, \gamma_j)$ (i.e., the model lines are parallel) and $(a_i, b_i, c_i) = (a_j, b_j, c_j)$ (the image lines are collinear), or 3) if there is a single unique solution to the equation,

$$p_i^0 + t(\alpha_i, \beta_i, \gamma_i) = p_j^0 + t(\alpha_j, \beta_j, \gamma_j)$$

for $i \neq j$ and $p_i^0 = (x_i^0, y_i^0, z_i^0)$ (i.e., the model lines intersect), and $(a_i, b_i, c_i) = (a_j, b_j, c_j)$ (i.e., the image lines are collinear).

How do the output values vary with respect to the input, e.g., if it is non-linear, describe the nature of the non-linearity. Because the pose estimation depends on the solution of a fourth order polynomial with at least four real solutions, if there is increasing input noise and if the correct solution of the four suddenly no longer produces the minimum, the solution will switch to another of the four and may cause the output pose estimation to change suddenly in value.

What are the constraints on the format and nature of the input? There must be three or more non-degenerate line matches, i.e., $N \geq 3$ (the degenerate case is already defined above).

What are the constraints on the format and nature of the environment? 1) The lighting must be such as to avoid excess specular reflection and shadows, since the algorithm is not explicitly designed to handle replacement errors, i.e., outliers. 2) Camera calibration must have been performed *a priori*. 3) There must be no roll in the camera. If there is a sensing device, what are the constraints on the type and use of the sensing (e.g., structured light, CCD camera, range camera)? CCD camera placed with the object fully within the field of view.

What are the constraints on the format and nature of the knowledge data? 1) The pitch, roll, and z-position of the part coordinate system must be known and identically zero in the machine coordinate system. 2) Model features for matching must be linear and, furthermore, must correspond to sensed features perceivable by standard edge detection algorithms. This constrains the model to be planar polygonal.

What computer language is the source code written in? Mathematica™. Is the source code available? No (for research only). What (if any) are the operating system requirements of the algorithm? Mathematica runs on UNIX, MacOS, Windows, Windows 95, MS DOS, Windows NT. Are there any system architecture requirements for using the algorithm? No. What kind computing hardware is required by the algorithm? Any hardware running UNIX, Macintosh, IBM-PC-compatibles. How much RAM memory and disc space is needed? About 8 megabytes RAM for Mac or PC. File size is about 2 megabytes. Are there any constaints on the numerical precision of the processing system? The computing system must allow computation that is precise to at least 24 decimal digits. This unusually high precision is due to the sensitivity of the fourth order polynomial, (EQ 6), i.e., for a certain data set, in the presence of little or no noise, the correct solution to (EQ 6) has been found to be on the order of 10^{-21}, which would be detected as effectively zero on some computing systems.

How fast does the algorithm run, based on actual examples run on specific computers? s?

If there is a standard test suite (a set of benchmarks) for algorithms performing the same task, what is that benchmark and how does the algorithm perform against it? No. Are there optimal algorithms that can produce

the ideal output? No. How is optimality defined for each optimal algorithm? N/A.

# of pixels translation error	2	4	6	8	10	12	14	16	18	20
pose error (x only) in meters	0.08	0.16	0.24	0.32	0.40	0.48	0.56	0.64	0.72	0.8
pose error (y only) in meters	0.04	0.09	0.14	0.19	0.24	0.29	0.34	0.39	0.44	0.49
pose error (θ only) in deg	0.5	0.54	0.58	0.62	0.66	0.7	0.74	0.78	0.82	0.86
scale error	0.03	0.045	0.06	0.075	0.09	0.105	0.13	0.145	0.16	0.175

Table 1: Propagation of error: image feature translation error to pose and scale error (direction error fixed at three degrees, number of line matches fixed at ten)

How does the algorithm perform in the presence of small perturbations on all data values? Experimentation has been done in the presence of noise on synthetic model data, a cuboid of size 3 x 2 x 1.2m^3. The synthetic object was placed about 22 meters from the center of the camera. Image size was 512 x 512 pixels. Small perturbations in translation were introduced by translating each ideal image line segment along its normal direction. Small perturbations in orientation were introduced by rotating each ideal image line segment around its midpoint. The magnitudes t and ω of the translation and rotation were assumed to be uniformly distributed over [-T, T] pixels and $[-\Omega, \Omega]$. Monte Carlo simulations were conducted to discover the propagation of error. Tables 1 and 2 report these results.

# of degrees direction error	2	4	6	8	10	12	14	16	18	20
pose error (x only) in meters	0.08	0.16	0.24	0.32	0.40	0.48	0.56	0.64	0.72	0.8
pose error (y only) in meters	0.04	0.09	0.14	0.19	0.24	0.29	0.34	0.39	0.44	0.49
pose error (θ only) in deg	0.1	0.4	0.7	1.0	1.3	1.6	1.9	2.2	2.5	2.8
scale error	0.03	0.04	0.05	0.06	0.07	0.08	0.09	0.10	0.11	0.12

Table 2: Propagation of error: image feature translation error to pose and scale error (translation error fixed at three pixels, number of line matches fixed at ten)

How does the algorithm perform as this type of noise varies throughout its range, e.g., does performance degrade smoothly or catastrophically? The degradation is roughly linear for the error in all dimensions as can be seen from Tables 1 and 2. State all models that exist for this type of noise. At each error level, 200 Monte Carlo simulations were done, absolute error between the ideal value of the parameter and the noisy output value is computed, and all 200 error values are averaged.

How robust is the algorithm, namely, how does the algorithm perform in the presence of large perturbations (replacement noise) on a subset of its data values? It is not designed to perform successfully with replacement noise nor has it been tested under such noise. How does the algorithm perform as replacement noise varies throughout its range, e.g., does performance degrade smoothly or catastrophically? N/A. Is the algorithm able to perform well (or at all) if the input is outside of the specified region? N/A. State all models that exist for replacement noise. None

How robust is the algorithm, namely, how does the algorithm perform in the presence of large perturbations (replacement noise) on a subset of its data values? It is not designed to perform successfully with replacement noise nor has it been tested under such noise. How does the algorithm perform as replacement noise varies throughout its range, e.g., does performance degrade smoothly or catastrophically? N/A. Is the algorithm able to perform well (or at all) if the input is outside of the specified region? N/A. State all models that exist for replacement noise. None

What are the relations defining computational complexity of the algorithm (there may be more than one such relation) as functions of the input variables? With N equal to the number of input line matches, the algorithm complexity is roughly $335 + 153N$ time intervals. What assumptions were made in this complexity analysis? We assume that square roots, divides, adds, and multiplies are roughly equivalent in time complexity. These times can be significantly dependent on the type of computing architecture employed. Decision steps were counted as 10 time intervals and there were two of them: 1) determining which of the four solutions to the quartic (EQ 6) are real and 2) determining the optimal $(\mathbf{q}_1, \mathbf{q}_2)$ the minimizes (EQ 7).

Is the algorithm iterative or closed form? Closed form. If iterative, under what (if any) conditions is convergence guarenteed? N/A If iterative, does the algorithm converge to the global solution? N/A

If there is some knowledge internal to the algorithm (i.e., not input/output data) to be represented, what is the format of the representation? The object is represented in terms of the parameters for lines (not line segments) for each of the edges on the planar polygonal part.

If the algorithm is available as source code, a library, or embedded in a system, how reliable is it? Not known. Are there known bugs? No. What reliability tests has the algorithm passed ? None. Has there been testing by some sort of coverage tool, which keeps track of which code is executed in a given session[1].

How widely has the algorithm been used? Not known. What is the reported experience with the algorithm? None. What are the potential application areas for this algorithm and, for each application area, give a set of competing algorithms with references? Pose estimation of automobiles on roads of known orientation and rigid objects on flat surfaces of known orientation. Some competing algorithms are (Dementhon, 1995), (Haralick, 1989), and (Huttenlocher, 1990).

What kind of algorithms are typically done prior to or subsequent to the execution of this algorithm? Matching algorithms are typically upstream. Are the representations chosen for the input and output data consistent with the expected or typical upstream and downstream algorithms? The expected upstream algorithm is a sensed feature to model feature matching algorithm and, since the Tan algorithm requires input line matches for edges on a planar polygonal object, the matching algorithm must produce line matches as well. However, if point or line segment matches are all that is available from the matching algorithm, line parameters can easily be generated from them. Does anything in the algorithm constrain the upstream or downstream algorithms that must be used? Line matches (or matches from which line matches can be easily derived) must be presented to the algorithm. Surface or curve matches are not acceptable.

Can the algorithm be parallelized? Some minor aspects of the algorithm can be parallelized. Has it been? No. How does parallelizing affect performance? Very little.

Can subcomponents of the algorithm be modularized? Yes. Have they been? Yes. How does modularization affect performance? Slows.

What is the error criterion, e.g., least squares? Using definitions in (EQ 1), (EQ 2), (EQ 3), (EQ 4), and (EQ 5), the task is to solve the overconstrained equation $\mathbf{A}\mathbf{q}_1 + \mathbf{B}\mathbf{q}_2 = \mathbf{C}$ for (x, y, θ, k). This is a non-linear least squares problem. The least squares solution is found by minimizing the squared residual $\|\mathbf{A}\mathbf{q}_1 + \mathbf{B}\mathbf{q}_2 - \mathbf{C}\|^2$ subject to the trigonometric constraint $\|\mathbf{q}_1\|^2 = 1$. This is accomplished by introducing a Lagrange multiplier, λ, and minimizing the following function with respect to $\mathbf{q}_1$, $\mathbf{q}_2$, and λ,

$$\varepsilon(\mathbf{q}_1, \mathbf{q}_2, \lambda) = \|\mathbf{A}\mathbf{q}_1 + \mathbf{B}\mathbf{q}_2 - \mathbf{C}\|^2 + \lambda(\|\mathbf{q}_1\|^2 - 1)$$

What input errors and/or internal errors does the algorithm detect? None. What does it do if such errors are detected? N/A Are there error recovery procedures? No.

How does the algorithm interact with other algorithms and systems? In Mathematica™, it is a simple function call. In order for Mathematica™ to exchange data with other software and systems, Mathematica™'s MathLink™ communications protocol must be used.

How is the code written, e.g., in functional, procedural, recursive, or object-oriented style. Mathematica™ code is functional, procedural, and interpreted.

What experiments have been done with the algorithm? Experimentation has been done in the presence of noise on synthetic model data, a cuboid of size 3 x 2 x 1.2m^3. The synthetic object was placed about 22 meters from the center of the camera. Image size was 512 x 512 pixels. If input data is varied over a set of variables, what criteria is chosen to sample the space of variables in order to generate sample input data? Small perturbations in translation were introduced by translating each ideal image line segment along its normal direction. Small perturbations in orientation were introduced by rotating each ideal image line segment around its midpoint. The magnitudes t and ω of the translation and rotation were assumed to be uniformly distributed over $[-T, T]$ pixels and $[-\Omega, \Omega]$. Was Monte Carlo testing done? Monte Carlo simulations were conducted to discover the propagation of error. Tables 1 and 2 report these results. Are there simulators available to generate input data? Yes, but only for the Mathematica™ code.

What kinds of analysis have been done on the results? Propagation of error analysis using Monte Carlo simulations of small perturbations on the sensed input lines. What kinds of graphs and tables have been produced and what is their format? See Tables 1 and 2. What statistical methods have been used? Simple arithmetic mean on all the errors as shown in Tables 1 and 2.

Which (if any) interface or data standards (published or defacto) are used and complied with? None.

6. FORMAL LANGUAGE INSTANTIATION OF GENERIC SPECIFICATIONS USING EXPRESS

The full 15-page EXPRESS specification is too long to include in this paper. To give the flavor of the formal specification, Fig. 1 and Fig. 2 show the EXPRESS definitions of the entities functionality_frame and functionality_frame_set. Functionality_frame has 23 attributes. In Fig. 1, each attribute is given on the left and indented. Each attribute name is followed by a colon and then the type of data required for the attribute. All the data types in Fig. 1 are defined elsewhere in the EXPRESS specification. The "where"

[1]Coverage tools give a sense of how much of the code was exercised by other reliability tests. The granularity can be at the function, block, or line level. A coverage will give the potential user greater confidence in a software component's reliability if it had been covered 100% during testing and errors were cleaned out of it.

clause near the end of the first entity definition is a simple example of a constraint. Attributes which are expected to be common to several related functionality_frames, such as author or source language, are included in the definition of functionality_frame_set.

The authors have instantiated these generic categories for a particular software component, which is an implementation built at NIST of an algorithm due to T. N. Tan and others for finding the pose of a solid part from an image of the part (Tan, *et al.*, 1992; Tan, *et al.*, 1994; Tan, *et al.*, 1996). The instantiation of the Tan component has been expressed in natural language and in the formal language of a STEP Part 21 exchange file (ISO, 1994b). Space limitations preclude the inclusion of either instantiated specification in its entirety. To give the flavor of the formal instantiated specification, Fig. 3 shows a very abbreviated version of the instantiation of the functionality_frame for the Tan component as it would appear in a Part 21 file. One referenced entity instance (#14, an io_item) is also defined in the figure.

In Fig. 3, data types which are strings appear between single quotes, data types which are lists appear inside parentheses, and data types which are EXPRESS entity instances appear as references of the form #n. Each referenced entity instance must be defined elsewhere in the same file. The attribute names in Fig. 3 were inserted manually as comments in italics and are not machine-readable; attributes are identified during machine reading by their position in the instance definition.

```
ENTITY functionality_frame;
  analysis_of_results : verbiage;
  application_experience : verbiage;
  benchmarks : SET [0:?] OF test_description;
  complexity : complexity_measure;
  convergence : convergence_statement;
  data_structures : LIST [0:?] OF data_structure;
  error_handling : verbiage;
  input_data : LIST [0:?] OF io_item;
  input_data_constraints :
      LIST [0:100] OF multiple_data_constraint;
  input_parameters : LIST [0:?] OF io_item;
  input_parameter_constraints :
      LIST [0:?] OF multiple_data_constraint;
  internals : internals_statement;
  known_bugs : verbiage;
  method_of_use : use_statement;
  name : identifier;
  niche : calls;
  noise_handling : verbiage;
  output_data : LIST [0:?] OF io_item;
  parallelizability : verbiage;
  precis : several_lines;
  robustness : verbiage;
  speed : verbiage;
  test_descriptions : LIST [0:?] OF test_description;
  WHERE
      precis_not_too_long :
      SIZEOF (precis) < 100;
END_ENTITY;
```

Fig. 1: EXPRESS definition of the functionality_frame entity

```
ENTITY functionality_frame_set;
  additional_information : information_statement;
  authors : LIST [1:?] OF person;
  compiled_available : LIST [0:?] OF computer_os;
  constraints : SET [0:?] OF constraint;
  data_acquisition : verbiage;
  executable_available :
      LIST [0:?] OF computer_os;
  interface_components:
      SET [1:?] OF functionality_frame;
  name : identifier;
  set_intent : verbiage;
  source_language : computer_language;
  source_available : BOOLEAN;
  standards_used : verbiage;
END_ENTITY;
```

Fig. 2: EXPRESS definition of the functionality_frame_set entity

```
#80 = FUNCTIONALITY_FRAME(
/* analysis_of_results */
      ('propagation of error using Monte Carlo',
      'simulation was done. error tables tabulated')
/* application_experience */   'none',
/* benchmarks */               (),
/* complexity */               #1,
/* convergence */              #2,
/* data_structures */          (#3, #4, #5),
/* error_handling */
      ('does not detect internal errors',
      'has no error recovery procedures')
/* input_data */               (#6, #7, #8),
/* input_data_constraints */   (#9),
/* input_parameters */         (#10),
/* input_parameter_constraints */ (),
/* internals */                #11,
/* known_bugs */               'none',
/* method_of_use */            #12,
/* name */                     'part pose calculation',
/* niche */                    #13,
/* noise_handling */
      'response linear to image line segment offset',
/* output_data */              (#14),
/* parallelizability */
      ('minor aspects appear to be parallelizable',
      'but it has not been attempted')
/* precis */
      ('Calculates part pose from location in image',
      'of points with known location on part')
/* robustness */               'unknown',
/* speed */                    'not measured',
/* test_descriptions */
      ('tested with synthetic data model of cuboid',
      'with well-characterized noise added'));

#14 = IO_ITEM(
/* name */                     'pose_and_scale',
/* data_type */                #15,
/* default_value */            'none',
/* item_constraints */         (#16, #17),
/* optionality */              .F.,
/* explanation */
      ('an array of four doubles representing',
      'x-offset, y-offset, rotation, and scale'));
```

Fig. 3: Functionality_frame instance

7. CONCLUSION

There often exists a long time lag between the creation of a new and promising software component and its profitable use in the commercial sector. Software component specifications would help shorten this time lag and be of significant profit to software component and software system developers alike. If defined as precisely as possible, such specifications would allow software system design tools to perform early but informative simulation. Early simulation is often critical to efficient software system development.

Software component specifications, as have been described, are analogous to existing specifications describing the functionality of hardware components (e.g., VHDL) sufficient for use in hardware system design and simulation tools.

This process (software system composition using component specifications) would occur naturally on the internet. Specification data and program segments (applets) would be easily and securely accessed by systems developers and software system composition tools.

Future plans are to pursue the following tasks:

- Continue to investigate formal languages (e.g., EXPRESS) and define these specification categories in the chosen language(s) as data and interpretable components (e.g., 'applets' in Java).
- Using the specification categories as a template, generate specifications for more components. Publish these specifications on the web in the form of static data and program 'applets.'
- Investigate the definition of software system requirements (for a target problem) and investigate the use of a search engine. The search engine would employ the requirements to scan the web (or a data base on the web) for components meeting required criteria.
- Using a semi-automated software system composition tool, investigate the use of these specifications with the tool to facilitate system design and simulation.
- Present the concept to industry, academia, and standards bodies through informal and formal interactions, web site development, conference presentations, and journal publications.

REFERENCES

Dementhon, D. F. and L. S. Davis, (1995). Model-based object pose in 25 lines of code, *International Journal of Computer Vision*, **Vol. 15**, pp. 123-141.

Diller, Antoni, (1994). *Z: An Introduction to Formal Methods*; 2nd Edition, John Wiley & Sons, New York.

Grady, Robert, D. L. Caswell, (1987). *Software Metrics: Establishing a Company-wide Program*; Prentice-Hall, Englewood Cliffs, N.J.

Haralick, R. M., *et al.*, (1989). Pose estimation from corresponding point data, *Machine Vision for Inspection and Measurement*, Academic Press.

Haralick, R. M., (1992). Performance characterization in computer vision, *Proceedings of the 3rd British Machine Vision Conference*.

Hofri, Micha, (1995). *Analysis of Algorithms: Computational Methods and Mathematical Tools*; Oxford University Press, N.Y. & Oxford.

Huttenlocher, D.P. and S. Ullman, (1990). Recognizing solid objects by alignment with an image, *International Journal of Computer Vision*, **Vol. 5**, No. 2, pp. 195-212.

Tan, T. N., G. D. Sullivan, and K. D. Baker, (1992). Linear algorithms for object pose estimation, *Proceedings of the 3rd British Machine Vision Conference*.

Tan, T. N., G. D. Sullivan, and K. D. Baker, (1994). Pose determination and recognition of vehicles in traffic scenes, *European Conference on Computer Vision*.

Tan, T. N., G. D. Sullivan, and K. D. Baker, (1996). Closed-form algorithms for object pose and scale recovery in constrained scenes, *Pattern Recognition*, **Vol. 29**, No. 3, 449-461.

Vick, C. R., C. V. Ramamoorthy, (1984). *Handbook of Software Engineering*; Van Nostrand Reinhold Co., N.Y.

ISO 10303-11:1994, (1994a). *Industrial automation systems and integration, Product data representation and exchange - Part 11: EXPRESS Language Reference Manual.*

ISO 10303-21:1994, (1994b). *Industrial automation systems and integration, Product data representation and exchange - Part 21: Clear Text Encoding of the Exchange Structure.*